

# The action of Kontsevich's graph complex on Poisson structures and star products: an implementation

Dissertation submitted for  
the award of the title *Doctor of Natural Sciences*  
to the Faculty Physics, Mathematics, and Computer Science  
of Johannes Gutenberg University Mainz  
in Mainz

Ricardo Thomas Buring

Born in Groningen, the Netherlands.

Mainz, October 31<sup>st</sup> 2022.

Supervisor: Dr. hab. Arthemy V. Kiselev  
University of Groningen, The Netherlands

Supervisor: Prof. Dr. Duco van Straten  
Johannes Gutenberg-University of Mainz, Germany

Date of doctoral examination: October 13<sup>th</sup>, 2022

## Abstract

Poisson brackets emerge whenever the pointwise product of scalar functions on an affine manifold is deformed in such a way that it stays associative. Kontsevich proved the converse: a universal formula assigns such an associative deformation to every Poisson bracket. Likewise, Poisson brackets can be deformed by universal formulae. In both constructions, the universal formulas are built by using graphs.

To handle the thousands of graphs, we develop and present the software package `gcaops` (*Graph Complex Action on Poisson Structures*) for SageMath. Using this package, • we expand Kontsevich's  $\star$ -product up to  $\bar{o}(\hbar^4)$ ; • we assemble  $\star \bmod \bar{o}(\hbar^6)$  from external data by Banks–Panzer–Pym and we obtain the star product  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$  for affine Poisson brackets; • we verify that graph weights found by Banks–Panzer–Pym up to  $\bar{o}(\hbar^6)$  satisfy many known relations; • we illustrate the explicit proof of the associativity for the full star product modulo  $\bar{o}(\hbar^6)$  and for the affine star product modulo  $\bar{o}(\hbar^7)$ ; • we find new explicit formulas of graph cocycles and universal Poisson cocycles, and • we prove the factorization of the Poisson cocycle condition via the Jacobi identity in each case.



# Contents

Overview	1
Actual background	1
Local embedding	4
Research problem	4
Scientific novelty of results	7
Practical significance of results	10
Personal contribution	10
Approbation of results	12
Publications and citation analysis	12
Structure of the dissertation	13
Scientific outline	16
Star products	16
Poisson flows	26
Conclusion	35
List of publications	37
Abstract	39
<b>I Computer demonstrations</b>	<b>40</b>
0 Introduction	41
0.1 Installation	41
0.2 Functions	42
0.2.1 Polynomials	42
0.2.2 Differential polynomials	43
0.2.3 Symbolic expressions	44
1 Implementation of star products	45
1.1 Star products	45
1.2 Gauge transformations	47
1.3 Polydifferential operators	48
2 Implementation of Poisson structures	55
2.1 Superfunctions	55
2.2 Vector fields	57
2.3 Bi-vector fields	57
2.4 Poisson structures	58
2.5 Poisson complex	58
2.6 Homogeneous polynomial Poisson complex	59

2.7	Stable Poisson cocycles . . . . .	61
3	Implementation of Formality . . . . .	65
3.1	Formality graphs . . . . .	65
3.2	The bi-colored operad of Formality graphs . . . . .	68
3.3	Formality graph complex . . . . .	72
3.4	Kontsevich star product . . . . .	74
3.4.1	Multiplicity . . . . .	74
3.4.2	Star product . . . . .	75
3.4.3	Star product from Formality morphism . . . . .	80
3.4.4	Star product associativity via Leibniz graphs . . . . .	83
3.5	Leibniz graph expansion and factorization(s) . . . . .	87
3.5.1	Iterative production of Leibniz graphs . . . . .	87
3.5.2	Leibniz graph factorization (non)uniqueness . . . . .	93
3.6	Cyclic weight relations . . . . .	97
3.6.1	From a basis to relations . . . . .	97
3.6.2	Kontsevich graphs in $f \star g$ . . . . .	98
3.6.3	Leibniz graphs in $(f \star g) \star h - f \star (g \star h)$ . . . . .	100
3.6.4	Known weights satisfy the cyclic weight relations . . . . .	103
3.7	Kontsevich's $\star$ product for affine Poisson structures . . . . .	107
3.7.1	Affine Kontsevich star product mod $\bar{o}(\hbar^6)$ . . . . .	107
3.7.2	Relations between the weights at $\hbar^7$ . . . . .	108
3.7.3	From weight relations to master parameters . . . . .	111
3.7.4	Substitute master parameters into $\star_{\text{aff}}$ mod $\bar{o}(\hbar^7)$ and its associator . . . . .	112
3.7.5	Restrict onto generic affine Poisson bivector on $\mathbb{R}^2$ . . . . .	114
3.7.6	Restrict onto affine rescaled Nambu–Poisson bivector on $\mathbb{R}^3$ . . . . .	114
3.7.7	Direct calculation of master parameter values . . . . .	115
3.7.8	Certificate of associativity . . . . .	116
3.7.9	Rationality of $\star_{\text{aff}}$ mod $\bar{o}(\hbar^7)$ . . . . .	121
4	Implementation of the graph complex . . . . .	125
4.1	Graphs . . . . .	125
4.2	Graph operad . . . . .	127
4.3	Full undirected graph complex . . . . .	128
4.4	Graph bases: storing them in cache . . . . .	130
4.5	Undirected graph complex . . . . .	131
4.6	Directed graph complex . . . . .	132
4.7	Undirected graph operations . . . . .	134
5	Examples of graph cocycles . . . . .	137
5.1	The tetrahedron cocycle $\gamma_3$ . . . . .	138
5.2	The five-wheel cocycle $\gamma_5$ . . . . .	140
5.3	The coboundary $\delta_6 = d(\beta_6)$ . . . . .	141
5.4	The heptagon-wheel cocycle $\gamma_7$ . . . . .	144
5.5	The commutator $[\gamma_3, \gamma_5]$ . . . . .	146
6	Graph complex action on Poisson structures in dimension two . . . . .	151
6.1	Tetrahedral $\gamma_3$ flow . . . . .	152
6.2	Five-wheel $\gamma_5$ flow . . . . .	154
6.3	Graph coboundary $\delta_6 = d(\beta_6)$ and the Poisson-trivial flow . . . . .	158
6.4	Heptagon-wheel $\gamma_7$ flow . . . . .	161

6.5	Commutator $[\gamma_3, \gamma_5]$ flow . . . . .	166
7	Graph complex action on rank two rescaled Nambu–Poisson structures . . . . .	169
7.1	Tetrahedral flow on rescaled Nambu–Poisson structures on $\mathbb{R}^3$ . . . . .	169
7.1.1	Superfunction algebra . . . . .	170
7.1.2	Tetrahedral flow . . . . .	170
7.1.3	The induced flow . . . . .	171
7.1.4	Symmetry . . . . .	172
7.1.5	Differential polynomial triviality . . . . .	173
7.1.6	Total skew-symmetry of the trivializing vector field . . . . .	175
7.2	Tetrahedral flow on Nambu–Poisson structures on $\mathbb{R}^4$ . . . . .	177
7.2.1	Symmetry . . . . .	180
7.2.2	Differential polynomial (non)triviality . . . . .	180
7.3	Tetrahedral flow on rescaled Nambu–Poisson structures on $\mathbb{R}^4$ . . . . .	181
7.3.1	Symmetry . . . . .	184
7.3.2	Differential polynomial (non)triviality . . . . .	185
7.4	Five-wheel flow on rescaled Nambu–Poisson structures on $\mathbb{R}^3$ . . . . .	186
8	Graph complex action on $R$ -matrix Poisson structures . . . . .	191
8.1	The Lie algebra of $2 \times 2$ matrices . . . . .	193
8.2	The Lie algebra of $3 \times 3$ matrices . . . . .	196
8.3	The Lie algebra of traceless $2 \times 2$ matrices . . . . .	199
8.4	The Lie algebra of traceless $3 \times 3$ matrices . . . . .	200
9	Graph complex action on star products . . . . .	203
9.1	Poisson-trivial deformations and gauge transformations . . . . .	203
9.2	Poisson-trivial deformation and the gauge transform in terms of graphs . . . . .	207
9.3	How the tetrahedral flow deforms the star-product . . . . .	208
	List of references . . . . .	211
<b>II Research articles</b> . . . . .		215
10	On the Kontsevich $\star$ -product associativity mechanism . . . . .	217
11	The expansion $\star \bmod \bar{o}(\hbar^4)$ and computer-assisted proof schemes in the Kontsevich deformation quantization . . . . .	223
12	Formality morphism as the mechanism of $\star$ -product associativity: how it works . . . . .	303
13	The heptagon-wheel cocycle in the Kontsevich graph complex . . . . .	321
14	Infinitesimal deformations of Poisson bi-vectors using the Kontsevich graph calculus . . . . .	345
15	The Kontsevich tetrahedral flow revisited . . . . .	359
16	Poisson brackets symmetry from the pentagon-wheel cocycle in the graph complex . . . . .	389
17	The orientation morphism: from graph cocycles to deformations of Poisson structures . . . . .	401

18	The Kontsevich graph orientation morphism revisited	415
19	Universal cocycles and the graph complex action on homogeneous Poisson brackets by diffeomorphisms	435
20	The hidden symmetry of Kontsevich's graph flows on the spaces of Nambu-determinant Poisson brackets	445
	<b>Back matter</b>	473
	Curriculum Vitae	473
	Acknowledgements	475
	Zusammenfassung	477
	Samenvatting	479
	Summary for Laymen	481
	<b>Appendices</b>	484
A	Introduction to SageMath	485
B	Kontsevich's star product $\star \bmod \bar{o}(\hbar^6)$	507
	B.1 Kontsevich's star product $\star \bmod \bar{o}(\hbar^4)$	507
	B.2 Associativity of Kontsevich's $\star \bmod \bar{o}(\hbar^6)$	511
C	Kontsevich's affine star product $\star \bmod \bar{o}(\hbar^7)$	519
	C.1 Original expansion $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$	519
	C.2 Reduced expansion $\star_{\text{aff}}^{\text{red}} \bmod \bar{o}(\hbar^7)$	539
	C.3 Associativity of $\star_{\text{aff}}^{\text{red}} \bmod \bar{o}(\hbar^7)$	551
D	Flows $Q_\gamma(P)$ and factorizations $\llbracket P, Q_\gamma(P) \rrbracket = \diamond_\gamma(P, \llbracket P, P \rrbracket)$	565
	D.1 The tetrahedral flow $Q_{\gamma_3}(P)$	565
	D.2 The pentagon-wheel flow $Q_{\gamma_5}(P)$	566
E	Graph cocycles	569
	E.1 The tetrahedron $\gamma_3$	569
	E.2 The pentagon-wheel cocycle $\gamma_5$	569
	E.3 The heptagon-wheel cocycle $\gamma_7$	569
	E.4 The commutator $[\gamma_3, \gamma_5] \in \ker d$ on 9 vertices and 16 edges	570
F	Reference documentation for the <code>gcaops</code> software	573



# Overview

The Formality theorem by Kontsevich introduced an extensive calculus of graphs into both deformation quantization and universal deformations of Poisson structures. The goal of this dissertation is to implement the Kontsevich graph calculus by algorithms and software modules. By this, we make it possible for every scholar to do explicit computations with Kontsevich's star product, graph flows, and graph complex. The software, together with the examples presented in this dissertation are available from<sup>1</sup>

<https://github.com/rburing/gcaops>

under the permissive MIT free software license. This new implementation allowed us to illustrate the theory by examples, inspect and verify conjectures, reveal properties of objects, and phrase new conjectures.

The domain of Kontsevich graph calculus not only offers theoretical concepts, but—by its explicit nature—also suggests the design of algorithms and development of software. Therefore, our present task is the translation of the theoretical framework into computational problems, examples, equations and solutions.

By using this software, we find the explicit expansion of the Kontsevich star product up to order 4, we find explicit representatives of graph cohomology classes, and we calculate the action of several graph cocycles on arbitrary 2D Poisson structures, as well as on families of rescaled Nambu–Poisson structures in 3D and 4D, and on homogeneous quadratic and cubic Poisson structures associated with  $R$ -matrices (those  $R$ -matrices are associated with splittings of the Lie algebras  $\mathfrak{gl}_2(\mathbb{R})$  and  $\mathfrak{gl}_3(\mathbb{R})$ ). Finally, we calculate examples of the graph complex action on star products.

This dissertation is such that it can also be used in education. Indeed, the examples and demos in this dissertation have already served as a basis for tutorials in an advanced master's course on deformation quantization and the graph complex (2020/21). The dissertation lets students get acquainted with applications of the graph calculus in other domains: how this theory is used in other parts of mathematics and physics. In this way, through natural examples and illustrations, the dissertation opens the door to domains such as abstract algebra, group theory, group actions, cohomology theory, deformation theory, combinatorics, combinatorial topology, Poisson geometry, calculus of multivectors, supergeometry, Lie groups and algebras, homotopy Lie algebras, jet bundles, geometry of differential equations, and topological methods in physics, including the Feynman path integral. We illustrate how these topics are united by the Kontsevich concept of graphs in deformation quantization.

**Actual background.** Sophus Lie introduced the notion of a Lie algebra as a generalization for linear Poisson brackets. Lie groups combine properties of manifolds and abstract groups. They are integral objects for Lie algebras. The theory developed in

---

<sup>1</sup>The abbreviation `gcaops` stands for Graph Complex Action on Poisson Structures.

the papers by H. Poincaré, E. Cartan, H. Weyl, H.S.M. Coxeter, E. Dynkin, V.G. Kac, and R. Moody has become fundamental in modern mathematics. Physical applications, namely the theory of gauge fields as connections in principal fibre bundles, united abstract ideas of mathematics with methods of theoretical physics. We now see that such is the gauge model of electroweak interactions, and such is the hypothesis/description of strong interactions. Together, they constitute the Standard Model. Independently, the pseudogroup of diffeomorphisms of spacetime acts in the geometry of Einstein's general relativity, i.e. in a classical description of gravity. With the general theory of Lie groups and algebras, we associate the problem of their representations in spaces of endomorphisms  $\text{End}(\mathbb{k}^n)$  (using matrices) and realizations in the spaces  $\mathfrak{X}^1(\mathbb{k}^n)$  of vector fields, i.e. derivations on the algebra of functions. The Cartan differential associates with a Lie algebra a chain complex, so that there appear the corresponding Chevalley–Eilenberg cohomology groups. Kontsevich in 1993–1994 discovered another natural class of Lie algebras. Namely, he discovered the structure of differential graded Lie algebra (dgLa) on the spaces of undirected graphs with external ordering of edges. Willwacher in 2010–2015 related suitable cocycles from the Kontsevich unoriented graph complex to the Lie algebra **grt** of the (infinitely generated) Grothendieck–Teichmüller group discovered by Drinfeld in 1990. Obviously we need effective tools to operate with the calculus of graphs, to manipulate graph expressions, to calculate cohomology, Betti numbers, and to do all the natural operations (such as the calculation of the graph differential).

Independently from the abstract group theory (e.g. permutations) and Lie groups, the idea of noncommutativity was a starting point for the emergence of new quantum mechanics by Dirac and Heisenberg, in contrast with the “old” quantum theory by Bohr and Sommerfeld. In the course of quantization, classical dynamical variables lose permutability. In that sense, the transition from Poisson algebra of dynamical functions to the representations of coordinates and momenta in the spaces of endomorphisms (self-adjoint Hermitian operators that act on typically infinite-dimensional spaces of wave functions for quantum objects) is a leap. This leap amounts to a radical change of both the physical sense of the geometry of the model, and of its mathematical description. Quite naturally, there appears the problem of linking quantum and classical, i.e. of semiclassical approximation, that would connect the classical picture and the quantum wave picture. But the noncommutativity of quantum objects, by itself requiring that the objects be ordered, is lost in the old classical purely commutative description. The theorem by H. Groenewold (1946) and L. Van Hove (1951) establishes that a naive correspondence of classical Poisson brackets and quantum commutators is not well-defined; counterexamples are immediately produced. E. Wigner and H. Weyl posed a natural question. Can we reinstate the noncommutativity already in the classical picture, so that we make all the diagrams commutative? To be precise, so that we bring noncommutativity into the classical picture in such a way that the (now, suddenly) deformed Poisson bracket is mapped under the Wigner–Weyl transform to the commutator of Hermitian operators on the spaces of quasiprobability distributions? Thus was created the theory of noncommutative associative star products, bridging the classical Poisson geometry and quantum mechanics by Heisenberg and Dirac. Fifty years passed, and a breakthrough result by Kontsevich established that every finite-dimensional Poisson manifold can be deformed-quantized. To solve that problem, Kontsevich developed and applied the language of directed graphs, by this interpreting the deformation quantization problem from analytic into topological combinatorial. Again, by using the standard technique by Gerstenhaber–Schlessinger–Stasheff, Kontsevich described the world of star products as a gauge theory

with cochain complexes. So there are equivalence classes of star products, and in turn the products themselves depend on the Poisson classes of Poisson brackets in their own Poisson cohomologies. Having related star products through the use of weighted graphs to the Lobachevsky hyperbolic geometry, Kontsevich pointed out in 1999 the existence of other classes of star products, with their weights not defined by the harmonic propagators. These classes of star products were explored by Alekseev, Rossi, Torossian, Willwacher in 2014. It is now expected that the coefficients of Kontsevich graphs in the star product contain many hidden numbers, the properties of which constitute open problems in number theory, e.g.  $\zeta(3)$  as pointed out by Felder–Wilwacher in 2008. A link of Kontsevich’s graph calculus with quantum field theory was explicated by Cattaneo and Felder (1999), Kontsevich’s graphs in the star product expansion are Feynman diagrams as those appear in the course of calculation of the correlation functions for the Ikeda–Izawa Poisson sigma model (1993–1994) by using the path integral expansion. The weights of Kontsevich graphs in the star product are calculated through the harmonic propagators in the quantum field theory Poisson sigma model. Now, so much has become understood about star products and about the relation of deformation quantization to problems of number theory. There naturally appears the problem of calculating Kontsevich’s star product expansion up to reasonably high order ( $\hbar^4, \hbar^5, \hbar^6, \hbar^7$ ). In this direction worked Penkava–Vanhaecke ( $\sim \hbar^3$  in 1998), Buring–Kiselev ( $\sim \hbar^4$  in 2017), and Banks–Panzer–Pym ( $\sim \hbar^5, \hbar^6$  in 2017, 2018) who based their work on that of Francis Brown. Obviously, because there are thousands of graphs in the star product expansion, we must have effective software to implement the directed graph calculus, in the world of star products.

The deformed Poisson bracket becomes a derivation with respect to the new noncommutative product. At the same time, it is interesting to construct symmetries of the old space of Poisson brackets over the fixed nondeformed algebra of functions. This amounts to the problem of constructing nontrivial second Poisson cohomology classes, such that their construction would be universal to all Poisson manifolds of arbitrary topologies. (If for a given Poisson manifold the second Lichnerowicz–Poisson cohomology were zero, then if such a universal construction existed, then it would still work and produce trivial classes.) Although a priori the formulation of this problem looks overly optimistic and strange, Kontsevich nevertheless formulated in 1996 a solution to this problem. It again appeals to the language of Kontsevich directed graphs, i.e. the same classes of graphs as were used in the solution of the deformation quantization problem. Willwacher in 2010 establishes the existence of a countable infinitely generated set of universal infinitesimal deformations  $\dot{P} = Q(P)$  of Poisson structures, i.e. second Poisson cohomology classes. They are differential polynomial in the coefficients of the given Poisson brackets. These infinitesimal deformations are obtained by orienting suitable graph cocycles, namely the wheel cocycles in the Kontsevich unoriented graph complex, and their iterated commutators. Kontsevich’s tetrahedral flow is the minimal nontrivial example. Until recently, any study of Kontsevich’s universal flows on the spaces of Poisson structures was hopeless. The programming of simple fast accessible computational tools would allow verification and illustration of all previously accumulated theoretical findings, as well as further new experiments, and prediction of new properties. A theoretical aspect of the study of such deformations is in that, as Kontsevich in 2019 (private communication) points out, universal deformations of Poisson brackets by using the language of graphs, both in the context of flows and noncommutative brackets, rediscover the quantization of the phase volume in the “old” Bohr–Sommerfeld quantization.

The software by Dror Bar-Natan (2000) and other authors (Willwacher–Živković 2014) was designed for the Kontsevich unoriented graph complex, to serve the count of graphs with up to 24 vertices and 36 edges, and the count of dimensions of graph homology groups e.g. in vertex-edge bi-grading  $(n, 2n - 2)$  up to  $n = 8$ . In the context of Kontsevich’s star products with harmonic propagators, the software to calculate the weights of directed graphs has been applied systematically up to  $\hbar^5, \hbar^6$  (Banks–Panzer–Pym). *Jets* by Michal Marvan serves a completely different purpose of geometric analysis of differential equations, and in that respect it is suitable for calculations in Poisson cohomology. Nevertheless, software packages which would have been capable to correlate the dgLa of unoriented graphs with the Lichnerowicz–Poisson complex and the calculus of directed graphs in the theory of star products did not exist. The ad hoc use of general purpose mathematical software was possible in principle, but the translation and transfer of data structures or the glueing together of scripts would be laborious and slow (as experienced firsthand by the author). Both the verification of previously existing material and further advancement required a new multi-functional package for calculation, computation, verification, and experiment. The present dissertation is devoted to the development and use of precisely such a package. We present the first general implementation of the action of Kontsevich’s graph complex on Poisson structures and star products.

**Local embedding.** The Institute of Mathematics at Johannes Gutenberg-University of Mainz, as well as the faculty of Physics, Mathematics, and Computer Science at large, has a strong tradition in areas of mathematics overarching the topic of this dissertation.

[REDACTED]

The Institute of Mathematics has been and is being part of large projects in fundamentals of science: SFB 45 “Periods, Moduli Spaces, and Arithmetic of Algebraic Varieties” consortium between Mainz, Bonn, and Essen (2007–2019) and CRC 326 GAUS “Geometry and Arithmetic of Uniformized Structures”, involving Frankfurt, TU Darmstadt, Heidelberg, Mainz, and Munich, now ongoing. Both of the grant projects are naturally related to the subject of this dissertation.

**Research problem.** *Problems about  $\star$ -products.* The Kontsevich star-product is a formal power series in  $\hbar$  defined through a weighted sum of graphs. While the weights are perfectly well defined numbers given by integrals, their explicit closed-form values are not easy to obtain. For any use of the formula in physics, an explicit expansion up to some order  $k$  would be essential. The main problem is therefore to determine the (harmonic) weights of graphs that appear in the Kontsevich star-product up to order  $k$ . An auxiliary problem is to explicitly verify the associativity of the  $\star$ -product expansion up to order  $k$ . This would reinforce the correctness of the found weights, and would also let us inspect the extensibility of the formula e.g. to the variational setting. Here a task is to re-examine the mechanism of associativity and the appearance of the Jacobi identity in the language of graphs. Since the number of graphs in the Kontsevich star-product already counts in

the hundreds at low orders, we set ourselves the main task of implementing the Kontsevich star product (and related objects) in computer software. As a byproduct, this will allow us to undertake the subtask of illustrating this theory where examples are sparse. We choose an indirect approach to the problem of determining the weights. Rather than calculating integrals directly, we consider the weights as unknowns, and our task is to obtain as many relations between them as we can. A subtask is how to get the most out of the associativity equation for the star-product. In this context we ask to what extent the rescaled Nambu–Poisson structures can be used to find relations between weights. A completely different method is the use of the Shoikhet–Willwacher–Felder cyclic weight relations. To what extent do they constrain the weights? Independently, a task is to verify the output of the software by Banks–Panzer–Pym, by substituting their weights into the relations we find. Another orthogonal task is to implement gauge transformations of star products, and to inspect how many graphs can be gauged out from the star-products. With a view toward the Formality morphism as the mechanism for star-product associativity, we also ask for a set of conventions such that left-hand sides are equal to right-hand sides in the Formality identities, not only morally but also arithmetically. Lastly we consider the non-uniqueness of the Formality morphism and hence the star-product; in particular we ask how the action of the graph complex on star products looks explicitly.

*Problems about graph flows.* A guiding question is the following open problem: Does the graph complex act nontrivially on Poisson structures? Within the scientific community (e.g. Kontsevich, Willwacher, Dolgushev *et al.*) it is conjectured that it does. The possibility of an abstract proof of the existence of a graph cocycle that acts nontrivially on a Poisson structure is a priori not excluded, but it would leave the author unsatisfied. Therefore we set ourselves the following optimistic task: Find a graph cocycle that acts nontrivially on a Poisson structure. Our chosen method is that of experimental mathematics: implement everything that is required, and try examples systematically. The calculation of the action requires explicit formulas for Poisson brackets and graph cocycles. While many Poisson brackets can be found in the literature, this is not true of explicit formulas for graph cocycles. A subtask is therefore to find explicit representatives of graph cohomology classes. For optimization purposes, it is moreover desirable to find a representative with as few terms as possible. Next we need software to orient the graph cocycles, and restrict to graphs with out-degree at most two. To realize the insertion of multi-vectors into graphs, an implementation of  $\mathbb{Z}_2$ -graded math is needed.

Regarding particular classes of Poisson structures we ask: Does the graph complex action preserve the class of (rank two) rescaled Nambu–Poisson structures? The task is to obtain the explicit formulas for infinitesimal deformations  $\dot{\rho}$  and  $\dot{a}$  of the functional parameters, and to verify Kontsevich’s conjecture about shape of the evolution  $\dot{a}$ . It is then an exercise to express the large differential polynomials  $\dot{\rho}$  and  $\dot{a}$  as total skew-symmetrizations of much smaller differential polynomials. Again we ask whether the graph complex action is trivial or not. In this case we can ask more specifically if there exists a trivializing vector field with differential polynomial coefficients. Another class of Poisson structures to consider is that of the homogeneous quadratic and cubic Poisson structures associated with  $R$ -matrices. In this case the homogeneity of the Poisson structure and hence of the Poisson differential reduces the problem of finding a trivializing vector field to a problem of finite-dimensional (but often high-dimensional) linear algebra.

On top of that, using the same graph language, we ask if it is possible to produce universal 1-cocycles (not 2-cocycles) for homogeneous Poisson structures. In this setting

there is also the task of illustrating this concept by (non)trivial examples.

Our final large-scale problem is: How far can the use of (known properties of) special (classes of) Poisson structures further our quest for the (non)trivial action?

---

*Research subject.* This research is interdisciplinary within mathematics and computer science. It refers to abstract concepts such as deformation quantization, Lie algebras, differential graded Lie algebras (dgLas), homotopy Lie algebras and homotopy morphisms ( $L_\infty$ ), Gerstenhaber’s deformation theory, supergeometry and supermanifolds, graded calculus of multivectors, Poisson geometry, Lichnerowicz–Poisson cohomology, nonlinear partial differential equations, jet spaces, dynamical systems, combinatorics, combinatorial topology, graph theory, but also algorithms and data structures (representing combinatorial data), high performance computational methods (fast computational linear algebra, count of dimension and rank), scientific programming and computation, symbolic computation, implementation of algebra of graphs and multivectors (dgLa), software design and engineering. We keep in mind that this research is carried out in the context of topological methods in physics.

*Research object.* Graphs, vector spaces spanned by graphs, algebraic structures on the spaces of graphs (commutator, differential), multivectors (with differential polynomial coefficients or symbolic coefficients), algebraic—specifically dgLa—structures on the space of multivectors (Schouten bracket  $\llbracket \cdot, \cdot \rrbracket$ ), differential polynomials, equations and identities for graphs, equations and identities involving multivectors, (ir)rational numbers as the weights of graphs (partly known and partly unknown: undetermined variables, expressed via multiple zeta values), multi-differential operators, graded objects (e.g. edges in graphs in the Kontsevich graph complex), Poisson brackets (examples and classes of), nonlinear partial differential equations and dynamical systems (in the context of the Poisson cohomology triviality condition for the evolution of the Nambu–Poisson brackets in 3D, 4D), Lie algebras,  $R$ -matrices. On the computer science side of this project, there are algorithms, computer programs, software libraries (linear algebra, `nauty`). We operate with theoretical concepts, ideas and conjectures (primarily, by Kontsevich); the new software helps us to convert conjectures into true statements.

*Research strategy.* We use a wide spectrum of analytic and computational methods to study examples case by case (so we do experimental mathematics, on the basis of fundamental mathematics by Kontsevich). This work combines logical reasoning (proof of theorems), design of experiments, and running extensive computations.

Proof of statements. Within deformation theory, we first study infinitesimal deformations, and only then we consider possible obstructions to the integrability of the infinitesimal deformations. (Instead of groups as integral objects, we study algebras, and apply the deformation cohomology theory.) We describe cohomology classes by producing their representatives. Overall, we operate modulo trivial objects (e.g. modulo zero graphs, modulo graph coboundaries, modulo gauge transformations). We regularly make use of gradings and homogeneity with respect to those gradings (e.g. numbers of vertices and edges in graphs, degrees of polynomial coefficients of multivector fields). We deploy methods of differential geometry and supergeometry (e.g. in the calculus of multivectors), and in the description of Formality morphism we refer to methods of homotopy Lie theory.

Experimental verification. We aim at proof or disproof of theoretical claims and conjectures. We calculate the weights of Kontsevich graphs and Leibniz graphs using external software (available from Banks–Panzer–Pym; the weights are expressed in terms of the multiple zeta values), and verify that they satisfy relations produced by our software.

In the case study of examples, we borrow Poisson structures from the literature and use them for a completely different purpose, in a new context (star products and graph flows). By these examples, we verify and illustrate the associativity of the star product via the Formality morphism, and similarly the Poisson cocycle condition for the Poisson graph flows.

Digital processing of data structures. Our guiding principle is representation of large objects by small simple markers. Our programs are optimized for both size and speed, by using the normal forms of graphs, and sparse vectors. We aim to express the graph cocycles by using a minimal number of terms, i.e. we work with the smallest possible representatives of cohomology classes. The graph cocycles are stored as sparse vectors with respect to a basis consisting of graphs. Finding basis of graph cohomology is then done using fast linear algebra methods, such as Markowitz pivoting. Let us emphasize that large linear systems are solved using rationals, not invoking finite precision floating point operations. When found, rational solutions are verified to be exact by direct substitution.

The software is designed in such a way that transfer of data structures between tasks is easy. For example, the object  $\text{Or}(\gamma_5)$ —defined in terms of a graph cocycle  $\gamma_5$ —is an endomorphism on the space of multivectors; the software is able to evaluate it at given Poisson structure. Since some problems are big, it is convenient to store intermediate results; this is achieved using the pickling in the Python language, as well as encoding in plain text files.

Identities which hold by force of the Jacobi identity can be processed in many ways. In particular, the software is able to restrict an identity onto the 3D rescaled Nambu–Poisson structures. Becoming a differential polynomial in the components  $\rho, a$  of the Nambu–Poisson bracket, the identity now splits into many homogeneous components. This idea of homogeneity and splitting is used extensively in all our software.

Within the geometry of differential equations and group theoretic methods for PDEs (by Sophus Lie), we do symbolic calculations in differential calculus on jet bundles. For example, we take the restriction of Kontsevich’s tetrahedral flow on the class of 3D Nambu–Poisson brackets and by using the `gcaops` software, we establish the existence of a trivializing vector field with differential polynomial coefficients.

The output of the software can be used independently of the software. We provide both the graph encodings and the analytic formulas encoded by these graphs. Whoever wants, can use either result.

**Scientific novelty of results.** The main results which reflect the scientific novelty and which are presented for defense are these:

1. The action of Kontsevich’s graph complex on Poisson structures and star products is implemented in the free software package <https://github.com/rburing/gcaops> written in free, open source software Python and SageMath.
2. Taking into account all conventions in multivector calculus and graph calculus, the `gcaops` software confirms the exact equality of left-hand sides and right-hand sides in identities such as the Leibniz-graph factorization of associativity for star products and the Poisson cocycle condition for graph flows (see Proposition 13 on p. 19 and Proposition 26 on p. 26, respectively). The balance of associativity requires recalculation constants; their values  $c_n = n/6$ , conjectured in Chapter 12 of Part II, are verified by computations in Chapter 3 of Part I, whereas the latter equality for graph flows is exact and absolute (see Chapter 5).

3. The `gcaops` software is able to find explicit representatives of the non-trivial graph cocycles  $\gamma_3, \gamma_5, \gamma_7$ , and calculate commutators of graphs, in particular  $[\gamma_3, \gamma_5]$ , in the Kontsevich graph complex with the vertex-expanding differential (see Propositions 42 and 43 on p. 32).
4. The canonical factorization (à la Kontsevich) of the Poisson cocycle condition for  $Q_{\gamma_3}(P)$  and  $Q_{\gamma_5}(P)$  via differential consequences of the Jacobi identity for (generic) Poisson structures, using Leibniz graphs, is given in Example 27 on p. 27 and in Table 0.4 on p. 28. The solution of this factorization problem is not unique in either case; there exist sums of Leibniz graphs such that their expansion into sums of Kontsevich graphs equals zero identically. The nullity of the Leibniz graph expansion map, restricted to the bi-gradings  $(3, n - 1)$  for  $n = 2, \dots, 5$  is reported in Table 0.8 on p. 34.
5. The graph cocycles  $\gamma_3, \gamma_5$  act on rescaled Nambu–Poisson structures. Computer experiment shows (in 3D and 4D for  $\gamma_3$ , and in 3D for  $\gamma_5$ ) that they preserve this class of Poisson structures ( $Q_\gamma(P[\rho, a]) = P[\dot{\rho}, a] + P[\rho, \dot{a}]$  in 3D, and similarly in 4D). The following conjecture by M. Kontsevich is true: the evolution of the functional parameters has the shape reported in Propositions 32, 33, 34 on pp. 29–29. (The huge formulas can be realized as the total skew-symmetrization of tiny differential polynomial expressions.<sup>2</sup>) Moreover, there exists a vector field  $X_{\gamma_3}$  with differential polynomial coefficients that trivializes the tetrahedral  $\gamma_3$  flow over  $\mathbb{R}^3$ , i.e. the equation  $Q_{\gamma_3}(P) = \llbracket P, X_{\gamma_3} \rrbracket$  has a solution  $X_{\gamma_3}$  with differential polynomial coefficients, again realized in Proposition 32 as the total skew-symmetrization of tiny differential polynomial expressions.
6. The graph cocycle  $\gamma_3$  acts on homogeneous quadratic and cubic Poisson structures associated with  $R$ -matrices. Computer experiment establishes that this action is Poisson-trivial in the cases of those  $R$ -matrices associated with splittings of Lie algebras  $\mathfrak{gl}_2(\mathbb{R})$  and  $\mathfrak{gl}_3(\mathbb{R})$ , see Proposition 36 on p. 30.
7. In 2-dimensional Poisson geometries, the Poisson cocycles  $Q_\gamma(P)$  defined by graph cocycles  $\gamma \in \{\gamma_3, \gamma_5, \delta_6, \gamma_7\}$  are Poisson-trivial, see Proposition 28 on p. 28. Namely, there exist vector fields  $X_\gamma(P)$ , differential polynomial in  $P$ , that trivialize the flows  $Q_\gamma(P) = \llbracket P, X_\gamma(P) \rrbracket$ . Moreover, with respect to the standard symplectic structure on  $\mathbb{R}^2$ , every such vector field  $X_\gamma(P)$  is the Hamiltonian vector field of a Hamiltonian  $H_\gamma(P)$ , again differential polynomial in  $P$  and expressed using Kontsevich graphs (see Proposition 31 on p. 28). The case of  $\gamma_3$  was known to Kontsevich (1996), and the respective Hamiltonian was found by Bouisaghouane (2016/17). The remaining cases are established by the new software.
8. In the Kontsevich star product at order 4, at most 256 digraph isomorphism classes of Kontsevich graphs can in principle appear. The weights of all these graphs are completely determined by the weights of 149 basic graphs (those with positive differential order which are nonzero and prime) and the known weights of graphs at lower orders. By using the associativity, the cyclic weight relations, and the known vanishing of (integrands of) some weights, the basic graphs weights are

---

<sup>2</sup>This was shown in collaboration with D. Lipper (2020).



expressed in terms of only 10 master parameters. The software by Banks–Panzer–Pym calculates the exact (rational) values of these master parameters, giving us the entire star product mod  $\bar{o}(\hbar^4)$  in Equation (13) in Chapter 11.

9. The Leibniz graphs are building blocks in the combinatorial mechanism for validity of identities which hold by force of the Jacobi identity and its differential consequences. The Leibniz graph realizations of the right-hand sides, when expanded to sums of Kontsevich graphs with a copy of the Poisson bracket in every internal vertex, ensure the vanishing of left-hand sides of identities, see Proposition 13 on p. 19 (with Table 0.2 on p. 20) and Proposition 26 on p. 26 (with Table 0.4 on p. 28). Examples of the work of this mechanism are in Example 14 on p. 20 and Example 27 on p. 27 (see also Proposition 46 on p. 34).
10. The new software is able to output the Shoikhet–Felder–Willwacher (2008) cyclic weight relations
  - between Kontsevich graphs that appear in the star product at orders 3, 4, 5, and
  - between Leibniz graphs that appear in the associativity equation for the Kontsevich star-product at orders 3, 4, 5 in  $\hbar$ .

We used the software by Banks–Panzer–Pym to calculate the weights of Leibniz graphs; we establish that all these values do satisfy the cyclic weight relations (see Proposition 15 on p. 20). We discover also that the cyclic weight relations (as well as the associativity itself, the known vanishing of some weights, and many other relevant properties) constrain the weights of Leibniz graphs in the associator and of Kontsevich graphs in the star product but do not completely determine them (see Table 0.2 on p. 20 and Table 0.3 on p. 21, and Proposition 15 and 16 respectively). Therefore, direct calculation of weights is inevitable.

11. We verify the associativity up to  $\bar{o}(\hbar^6)$  for the full Kontsevich star-product, known modulo  $\bar{o}(\hbar^6)$  from Banks–Panzer–Pym [1] for arbitrary (non)linear Poisson brackets and arbitrary arguments, by realizing (every homogeneous tri-differential component of) the associator as a sum of Leibniz graphs from the 0th layer, that is the Leibniz graphs produced at once by contracting edges in the Kontsevich graphs from the associator (see Proposition 17 on p. 22).
12. The Kontsevich  $\star$ -product admits a restriction to the class of Poisson brackets with *affine* coefficients on finite-dimensional affine manifolds (e.g., such are the Kirillov–Kostant Poisson brackets on the duals of Lie algebras: their coefficients are strictly linear without constant terms). In Section 3.7, see Proposition 18 on p. 23 below, we obtain the expansion  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$  of *affine* Kontsevich’s star-product: in all the Kontsevich graphs in it the in-degrees of aerial vertices are bounded by  $\leq 1$ . The graph expansion  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$  is contained in Appendix C.1; at  $\hbar^6$  and  $\hbar^7$ , the coefficients of many Kontsevich graphs in  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$  contain  $\zeta(3)^2/\pi^6$  times a nonzero rational factor, plus a rational part.
13. We discover that in both the orders  $\hbar^6$  and  $\hbar^7$  in  $\star_{\text{aff}}$  with the harmonic graph weights, the entire coefficient of  $\zeta(3)^2/\pi^6$ , itself a  $\mathbb{Q}$ -linear combination of Kontsevich graphs, assimilates into a linear combination of Leibniz graphs (doing so at

once, without need of the 1st layer of Leibniz graphs). In consequence, whenever the affine star product  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$  is restricted to an arbitrary affine Poisson structure, the constant  $\zeta(3)^2/\pi^6$  does not appear at all in the resulting analytic expression. In fact, more terms in  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$  can be absorbed into Leibniz graphs. Namely, in §3.7.9 we obtain and in Appendix C.2 we list the affine Kontsevich graph expansion  $\star_{\text{aff}}^{\text{red}} \bmod \bar{o}(\hbar^7)$ , at once excluding the part which is now known to vanish identically: there remain only 326 nonzero rational coefficients of Kontsevich graphs (at all orders, up to  $\hbar^7$ ), in contrast with the original graph expansion of the Kontsevich star product  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$  in which the Kontsevich integral formula yields 1423 nonzero  $\mathbb{Q}$ -linear combinations of 1 and  $\zeta(3)^2/\pi^6$  for the Kontsevich graph coefficients.<sup>3</sup>

14. We verify the associativity up to  $\bar{o}(\hbar^7)$  of the affine Kontsevich star product, known modulo  $\bar{o}(\hbar^7)$  for arbitrary affine Poisson brackets and arbitrary arguments, by realizing (every homogeneous tri-differential component of) the associator as a sum of Leibniz graphs. We establish that, for the tri-differential orders  $\{(3, 3, 2), (2, 3, 3), (3, 2, 3), (2, 4, 2)\}$  the 0th layer of Leibniz graphs is not enough for any such factorization to exist, yet solutions appear in presence of the 1st layer of Leibniz graphs in each of the four exceptional cases; see Proposition 19 on p. 23 below, and see the *Proof scheme (for the reduced affine star product  $\star_{\text{aff}}^{\text{red}} \bmod \bar{o}(\hbar^7)$ )* on p. 24 specifically about the properties of the associator for the reduced affine star product  $\star_{\text{aff}}^{\text{red}} \bmod \bar{o}(\hbar^7)$  with only rational coefficients. We finally deduce that the indispensability of the first layer of Leibniz graphs is such that it carries on to any factorization of the associator at  $\hbar^7$  for the full star product modulo  $\bar{o}(\hbar^7)$ ; see Example 20 on p. 24.

**Practical significance of results.** The PhD dissertation has both theoretical and practical character. The software is designed for verification of theoretical predictions, and also for verification of other software (computational results). Independently, it can be used to verify, extend and re-check results of computations in other works. Every scholar can use the free open source software to solve relevant problems in this domain of science by themselves. The implementations of multivector calculus, polydifferential operators, and differential polynomials can be used independently from the graph complex modules.

Both the text of the dissertation and the software implementation (available online) can be used in education. Examples and demo sessions based on this research have already been approbated in the advanced master's course on deformation quantization and the graph complex (2020/21).

**Personal contribution.** All results which are set for defense, the dissertant accomplished in person. In the works which are published jointly with other authors, the contribution of the dissertant is as follows.

In [1] entitled *On the Kontsevich star-product associativity mechanism*, I did the calculation of the star product expansion up to order 3, and the graphical calculation of the associator up to the same order. I provided the first explicit example illustrating the

---

<sup>3</sup>In the above reduction of  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$  none of the Kontsevich graph weights was altered or were redefined; the reduction of the number of terms which effectively contribute to the star-product of functions and to its associator is due to a revealed property of those graphs and their Kontsevich weights.

work of Kontsevich’s Formality morphism, by expressing the  $\star$ -product associativity up to order 3 as a differential consequence of the Jacobi identity.

In [2] entitled *The expansion  $\star \bmod \bar{o}(\hbar^4)$  and computer-assisted proof schemes in the Kontsevich deformation quantization*, I wrote all the computer implementations in `kontsevich_graph_series-cpp`. Moreover I created all the examples, based on discussions with A.V. Kiselev. The text was written in collaboration with A.V. Kiselev. Reducing the number of parameters by using a gauge transformation was suggested by A.V. Kiselev and performed by me. I did all the comparison with the literature in the Discussion. The (rational) values of the 10 master parameters were received from Banks–Panzer–Pym (2017).

In [3] entitled *Formality morphism as the mechanism of  $\star$ -product associativity: how it works*, I calculated all of the examples in Section 4 and 5. The text was written together with A.V. Kiselev.

In [4] entitled *The heptagon-wheel cocycle in the Kontsevich graph complex*, I found a representative of the heptagon wheel cocycle, using my own software, and I counted the dimensions of graph cohomology groups (two variants: valency-dependent). I made minor contributions to the text, which (with extra examples and illustrations) is mainly due to A.V. Kiselev and N. Rutten. I wrote ad hoc SageMath code for the graph differential in Appendix B. (Similar code is now used in `gcaops`.) I assisted N. Rutten in generating L<sup>A</sup>T<sub>E</sub>X pictures of graphs.

In [5] entitled *Infinitesimal deformations of Poisson bi-vectors using the Kontsevich graph calculus*, I wrote the code with some help from N. Rutten. Moreover I contributed to the text describing the algorithms.

In [6] entitled *The Kontsevich tetrahedral flow revisited*, I did the graphical calculation of the Poisson differential  $\llbracket P, Q_{\text{tetra}} \rrbracket$ , including the skew-symmetrization. I assisted A. Bouisaghouane in finding an example where  $a : b = 1 : \frac{6}{2}$  is necessary. I wrote the computer program to find a Leibniz graph factorization of  $\llbracket P, Q_{\text{tetra}} \rrbracket$ . The software is discussed further in [2].

In [7] entitled *Poisson brackets symmetry from the pentagon-wheel cocycle in the graph complex*, I calculated the orientation of the five-wheel cocycle jointly with N. Rutten. My software produced the analytic formula in Appendix A.

In [8] entitled *The orientation morphism: from graph cocycles to deformations of Poisson structures*, I wrote the software `graph_complex-cpp` for the graph differential and orientation of graph cocycles. The example of a coboundary  $\delta_6 = d(\beta_6)$  was calculated using that software. The text was written together with A.V. Kiselev. I calculated the numbers of graphs in Table 2 that shows the size of the problem.

In [9] entitled *The Kontsevich graph orientation morphism revisited*, I provided the encodings of graphs used in all examples; the text is due to A.V. Kiselev.

In [10] entitled *Universal cocycles and the graph complex action on homogeneous Poisson brackets by diffeomorphisms*, I proposed the idea of evaluating  $\text{Or}\vec{\gamma}$  at tuples other than  $P^{\otimes n}$ , and initiated the use of homogeneous  $P$  in this context. I evaluated the flow  $Q_{\gamma_3}$  on Poisson structures associated with two  $R$ -matrices for  $\mathfrak{gl}_2(\mathbb{R})$ , and found trivializing vector fields for those flows. I calculated the action of the tetrahedral flow on the rescaled Nambu–Poisson brackets depending on functional parameters  $(a, \rho)$ , inducing the flow  $(\dot{a}, \dot{\rho})$ . The graph realization of  $\dot{a}$  was pointed out by M. Kontsevich immediately, during discussion with A.V. Kiselev and myself at the IHÉS (December 2019). I verified the claim by Kontsevich about the shape of  $\dot{a}$ , and also found  $\dot{\rho}$ .

In [11] entitled *The hidden symmetry of Kontsevich’s graph flows on the spaces of*

*Nambu-determinant Poisson brackets* I calculated the induced evolution of the functional parameters for the tetrahedral  $\gamma_3$  flow over  $\mathbb{R}^3$  and  $\mathbb{R}^4$  and for the pentagon-wheel  $\gamma_5$  flow over  $\mathbb{R}^3$ . The realization of those formulas by using Civita symbols is joint work with D. Lipper. I established the Poisson-triviality of the  $\gamma_3$  flow over  $\mathbb{R}^3$  in the Poisson cohomology of the respective Nambu–Poisson structure: I represented the trivializing vector field by using the Kontsevich graphs in which the vertices with  $\rho$  and Civita symbols are resolved against the vertices with Casimirs.

**Approbation of results.** Prior to the start (April 2017) and in the middle (December 2019) of the project, the dissertant visited the IHÉS for one week (with A.V. Kiselev), where the content and progress was discussed with and presented to M. Kontsevich. The conjectures and approaches suggested there have been pursued in this work, and the comments and feedback given there has been incorporated into the dissertation.

The work in this dissertation was presented at the following events:

*International conferences.* The 34th International Colloquium on Group Theoretical Methods in Physics (Strasbourg, France, July 18–22, 2022); Poisson Geometry, Lie Groupoids and Differentiable Stacks (Banff, Canada, June 5–10, 2022); Symmetry and Integrability of Equations of Mathematical Physics (Kyiv, Ukraine, December 22–23, 2018); Homotopy algebras, deformation theory and quantization (Bełdewo, Poland, September 16–22, 2018); The 32nd International Colloquium on Group Theoretical Methods in Physics (Prague, Czech Republic, July 9–13, 2018); Symposium on Advances in Semi-Classical Methods in Mathematics and Physics (Groningen, The Netherlands, October 19–21, 2016); Group Analysis of Differential Equations and Integrable Systems (Larnaca, Cyprus, June 12–17, 2016); Symmetries of Discrete Systems and Processes III (Děčín, Czech Republic, August 3–7, 2015).

*Colloquia.* Two GQT Schools and Colloquia (Den Dolder, The Netherlands, July 3–7, 2017 and July 1–3, 2019); two Ph.D. meetings of the Sonderforschungsbereich/Transregio 45 (Physikzentrum Bad Honnef, January 26–29, 2018, and Universität Duisburg–Essen, February 1–3, 2019); Spring school Enumerative Invariants from Differential Graded Lie Algebras and Categories (Montegufoni, Italy, March 25–31, 2018).

*Seminars.* Two talks at the Informal Seminar on Mathematical Aspects of Scattering Amplitudes (JGU Mainz, Germany, January 9 and May 15, 2019); Working group on Grothendieck–Teichmüller groups (MPIM Bonn, Germany, December 12, 2018); Floris Takens Dynamical Systems Seminar (DSGMP, Bernoulli Institute, University of Groningen, The Netherlands, September 11, 2018); Junior Geometry and Topology seminar (University of Oxford, United Kingdom, May 3, 2017); DIAMANT Intercity Number Theory Seminar (University of Groningen, The Netherlands, April 7, 2017).

The feedback from seminars is incorporated into publications on which the dissertation is based. Reciprocally, much of the work of the dissertant has served as the basis of talks by other coauthors. Lastly, demos and examples from the dissertation have been approbated in a master’s course on deformation quantization and the graph complex (Autumn semester 2020/21), contributing to 15 tutorials and two PC demo sessions.

**Publications and citation analysis.** The dissertation is based on ten journal publications and one preprint, as well as twenty externally stored **Jupyter** notebooks with computer demonstrations, based on free open source software packages. The main results are contained in [2, 6, 3, 8]. All ten journal publications underwent anonymous peer review by referees. Four papers are published in journals which are indexed by Mathematical

Reviews (MR), independently five works are indexed in zbMATH (formerly Zentralblatt MATH), and three papers are indexed by the IAEA.

*Citations.* The paper [6] has been immediately cited by Kontsevich<sup>4</sup> in his Séminaire Bourbaki talk (January 2017). The concept and result of our paper [2] was used by the Oxford group of Banks–Panzer–Pym<sup>5</sup> to check their result. New explicit examples of graph cocycles and Poisson bracket flows from [6, 8, 4, 5, 7, 10] are recognized in the topical review by Morand.<sup>6</sup>

**Structure of the dissertation.** The dissertation consists of two major parts, as well as this overview and appendices. Part I combines an introduction to theory, introduction to software (computer demonstrations), and main examples which motivate further study in Part II. By looking at Part I, the reader gets an impression of the technology of doing this mathematics.

In Chapter 0 which is the *Introduction*, we inform the reader how to obtain and install the `gcaops` software; we show the basics how functions (such as polynomials and differential polynomials) are manipulated.

In Chapter 1 entitled *Implementation of star products*, we begin by recalling the notion of star product; how we truncate it and the associativity equation. We recall the idea of gauge transformation, and the notions of Gerstenhaber bracket, Hochschild differential and Maurer–Cartan equation.

In Chapter 2 entitled *Implementation of Poisson structures*, we first explain the supercalculus on the space of multivectors, endowed with the Schouten bracket. The Schouten bracket provides the Jacobi identity for Poisson bi-vectors. Likewise, the Schouten bracket gives us the Poisson differential and Lichnerowicz–Poisson cohomology. We illustrate the concept by giving examples of Poisson brackets (in particular, with homogeneous polynomial coefficients) and Poisson cocycles. In particular Kontsevich’s tetrahedral flow is a 2-cocycle, and when restricted to a specific Poisson bracket it is a coboundary.

In Chapter 3 entitled *Implementation of Formality* we first recall the construction and properties of the graphs in Kontsevich’s Formality morphism; then we build Kontsevich’s  $\star$ -product modulo  $\bar{o}(\hbar^4)$  using Kontsevich’s graphs and we verify the associativity of  $\star \bmod \bar{o}(\hbar^4)$  by using Leibniz graphs. Moreover we investigate the (co)ranks of linear algebraic systems of the Shoikhet–Felder–Willwacher cyclic weight relations that constrain the graph weights at a given order  $\hbar^k$  (here,  $k \leq 5$  for the Kontsevich graphs in the star-product and hence  $n \leq 4$  for the Leibniz graphs in the associator). In §3.5 we discuss Leibniz graphs in detail, and we factor (every homogeneous tri-differential component of) the associator modulo  $\bar{o}(\hbar^6)$  by using the 0th layer of Leibniz graphs, that is those Leibniz produced at once from the Kontsevich graphs in the associator itself. In §3.7 we obtain the affine star product  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$ , we contrast its associativity mechanism with the previously known mechanism that worked at orders  $\leq 6$ , and we reduce the affine star product at orders  $\hbar^6, \hbar^7$  by absorbing terms such as the  $\mathbb{Q}$ -linear combinations of Kontsevich graphs near  $\zeta(3)^2/\pi^6$  into linear combinations of Leibniz graphs. The graph encoding of the reduced affine star product  $\star_{\text{aff}}^{\text{red}} \bmod \bar{o}(\hbar^7)$  is given in Appendix C.2.

---

<sup>4</sup>Maxim Kontsevich. Derived Grothendieck–Teichmüller group and graph complexes [after T. Willwacher]. *Séminaire Bourbaki*. Vol 2016/2017. Exposés 1120–1135. Astérisque No. 407 (2019), Exp. No. 1126, 183–211. ISBN: 978-2-85629-897-8.

<sup>5</sup>Peter Banks, Erik Panzer, Brent Pym. Multiple zeta values in deformation quantization. *Invent. Math.* 222 (2020), no. 1, 79–159.

<sup>6</sup>Morand, Kevin. M. Kontsevich’s graph complexes and universal structures on graded symplectic manifolds I. arXiv:1908.08253 [math.QA] – 42 pages.



In Chapter 4 entitled *Implementation of the graph complex* we demonstrate how the definition of the graph complex is implemented in software: how graphs are encoded, how their brackets are calculated, and how the differential acts. We give basic examples: the stick, Kontsevich’s tetrahedron, and the pentagon wheel cocycle by Kontsevich–Willwacher; we show that they all are cocycles. After the undirected graph complex, we study directed graphs and we introduce the directed graph complex. We give an example of a directed graph cocycle, by illustrating how to convert from the undirected to the directed graph complex. We recall and illustrate how the Schouten bracket comes from the stick graph, and how the tetrahedral flow originates from the tetrahedron  $\gamma_3$ .

Chapter 5 is entitled *Examples of graph cocycles*; now we start a systematic search for new graph cocycles. We deploy methods of linear algebra and make the search automatic. In this way, we find a coboundary  $\delta_6$ , and nontrivial graph cocycles  $\gamma_7$ , and  $[\gamma_3, \gamma_5]$ . Further, we prove the factorization of the Poisson cocycle condition via the Jacobi identity in each case, by providing the necessary Leibniz graphs.

In Chapter 6 entitled *Graph complex action on Poisson structures in dimension two* we evaluate several graph flows at a generic Poisson structure on  $\mathbb{R}^2$ . In each case, we obtain the formula of the flow, as well as the differential polynomial expression for the coefficients of the trivializing vector field. Moreover, we discover its Hamiltonian with respect to the standard symplectic structure and a graph realization of the Hamiltonian. This is done for the tetrahedron  $\gamma_3$ , the pentagon-wheel cocycle  $\gamma_5$ , the coboundary  $\delta_6$ , and for the heptagon wheel cocycle  $\gamma_7$ .

Chapter 7 entitled *Graph complex action on rank two rescaled Nambu–Poisson structures* is about Poisson bi-vectors of the form  $P = \rho da / (dx \wedge dy \wedge dz) = \llbracket \rho \partial_x \wedge \partial_y \wedge \partial_z, a \rrbracket$ , with, obviously, coefficients which are differential polynomial in  $\rho$  and  $a$ . We establish that this class of Poisson brackets is preserved by the tetrahedral  $\gamma_3$  flow and by the pentagon-wheel  $\gamma_5$  flow (as are similar brackets in dimension 4). In every case we express the evolution  $\dot{\rho}, \dot{a}$  as differential polynomials in  $\rho$  and  $a$ . Because these expressions are highly symmetric, we collapse them by using the markers of minimal size and Civita symbols. Independently, for the Kontsevich tetrahedral flow  $\gamma_3$  we establish the existence of a trivializing vector field  $X[\rho, a]$  with differential polynomial coefficients, and again we collapse it by using Civita symbols and marker monomials, which are realized by using graphs.

Chapter 8 entitled *Graph complex action on R-matrix Poisson structures* is about homogeneous quadratic and cubic Poisson brackets associated with  $R$ -matrices (they are borrowed from Li–Parmentier); in turn those  $R$ -matrices are constructed for Lie algebras  $\mathfrak{gl}_2(\mathbb{R})$  and  $\mathfrak{gl}_3(\mathbb{R})$ . We use the homogeneity to establish that the Kontsevich tetrahedral  $\gamma_3$  flow is trivial in each case; we give explicit formulas for the trivializing vector fields.

Chapter 9 entitled *Graph complex action on star products* combines star products and gauge transformations, unoriented graph cocycles, and Poisson structures and their Kontsevich’s flows. Finally, we give two examples of the graph complex action on  $\star$ -products. We show how the Poisson (non)trivial evolution induces an evolution of star products. For the fourth order expansion of the Kontsevich star product and for the Kontsevich tetrahedral flow we find out whether the induced deformation of the star product amounts to a gauge transformation.

All the software demonstrations in Part I are based on SageMath Jupyter notebooks. Those notebooks can be retrieved from the same place as the programs themselves, i.e. <https://github.com/rburing/gcaops>; the output data files (e.g. graph encodings, Kontsevich graph weights and Leibniz graph weights, and the Shoikhet–Felder–Willwacher

cyclic weight relations) are also found there. The notebooks use the software, and the data files can be appreciated separately.<sup>7</sup>

---

Based on research articles, Part II is more theoretic. Publications are clustered in three groups: about star products, about graph calculus, and about Kontsevich flows of Poisson structures encoded by graphs. We study relevant theory and prove new lemmas and theorems; computational results are obtained by using the same software as in Part I, as well as by the software modules `kontsevich_graph_series-cpp` in C++ also by the author (2015–2019).

We begin with [1] entitled *On the Kontsevich  $\star$ -product associativity mechanism*, in which we factor associativity through differential consequences of the Jacobi identity. Here we meet Leibniz graphs for the first time, in the associator for  $\star$ -products. The core publication in this group of articles is [2] entitled *The expansion  $\star \bmod \bar{o}(\hbar^4)$  and computer-assisted proof schemes in the Kontsevich deformation quantization*. We express the weights of all graphs in  $\star \bmod \bar{o}(\hbar^4)$  in terms of only 10 master parameters. We import the values of these master parameters from Banks–Panzer–Pym, write down the formula of authentic Kontsevich star product, and verify its associativity by using Leibniz graphs. In [3] entitled *Formality morphism as the mechanism of  $\star$ -product associativity: how it works* we study the algebraic mechanism of associativity in terms of the Formality morphism, and we illustrate it. In hindsight, papers [1] and [2] allow us to illustrate the concept with explicit examples at orders 3 and 4.

The paper [4] entitled *The heptagon-wheel cocycle in the Kontsevich graph complex* is an introduction to the Kontsevich graph complex, in which we re-derive graph cohomology classes related to the `grt` Lie algebra, namely the tetrahedron  $\gamma_3$ , the pentagon-wheel cocycle  $\gamma_5$ , the heptagon-wheel cocycle  $\gamma_7$ , and the commutator  $[\gamma_3, \gamma_5]$ .

In [5] entitled *Infinitesimal deformations of Poisson bi-vectors using the Kontsevich graph calculus*, we begin the quest for Poisson flows defined by Kontsevich directed graphs. Here, we design algorithms and we do the full run through Kontsevich graph flows for few-vertex graphs. The conclusion is that there are only the `grt`-related flows and no others.

The Kontsevich graph flows  $\gamma_3, \gamma_5, \gamma_7$  related to `grt` are presented in [6], [7], [8] respectively. In [6] entitled *The Kontsevich tetrahedral flow revisited* we find the correct balance 8 : 24 of graph coefficients in Kontsevich’s tetrahedral graph flow. In [7] entitled *Poisson brackets symmetry from the pentagon-wheel cocycle in the graph complex* we obtain the coefficients of oriented graphs in the pentagon-wheel flow, and we establish the (non-unique) Leibniz graph factorization of the Poisson cocycle condition. In [8] entitled *The orientation morphism: from graph cocycles to deformations of Poisson structures* we not only provide the encoding of the heptagon wheel flow, but also analyze the factorization mechanism, originally by Kontsevich, in much detail.

We revisit the graph orientation morphism in [9], where the morphism is expressed combinatorially in terms of graphs themselves. Next, in [10] entitled *Universal cocycles and the graph complex action on homogeneous Poisson brackets by diffeomorphisms*, we construct universal Poisson 1-cocycles for homogeneous Poisson structures. We also examine the (non)triviality of universal Poisson 2-cocycles for the brackets obtained from  $R$ -matrices. Finally, we report on the rescaled Nambu–Poisson structures  $P[\rho, a]$ : the Kontsevich flows preserve this class of brackets, forcing the nonlinear evolutions  $\dot{\rho}, \dot{a}$  with

---

<sup>7</sup>© The copyright for all newly designed software modules is retained by R. Buring; provisions of the MIT free software license apply.

differential polynomial r.-h.s. In [11] entitled *The hidden symmetry of Kontsevich's graph flows on the spaces of Nambu-determinant Poisson brackets* we show that the tetrahedral flow and pentagon-wheel flow preserve the class of Nambu–Poisson bi-vectors over  $\mathbb{R}^3$  and  $\mathbb{R}^4$ , we collapse the induced evolution of the functional parameters using the Civita symbols, we find a further discrete symmetry of these evolution equations. For the class of Nambu–Poisson bivectors over  $\mathbb{R}^3$  we establish that the Poisson bracket evolution with respect to the tetrahedral  $\gamma_3$  flow is trivial in the respective Poisson cohomology and we collapse the formula of the trivializing vector field by using the Civita symbols again.

The dissertation concludes with 6 appendices, as well as with a Curriculum Vitae, acknowledgements, three abstracts (in English, Dutch, and German), and an abstract for laymen. In the appendices that follow, we provide in particular the encoding of  $\star \bmod \bar{o}(\hbar^4)$  of the Kontsevich authentic  $\star$ -product, the encoding of  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$  of the affine Kontsevich  $\star$ -product, and an appendix with software reference documentation.

**Scientific outline.** In this overview we recall the actual background, we formulate the research problem, set up goals which we pursue, and phrase the main results. We keep in mind that this dissertation consists of two large parts. Part I contains ten chapters (enumerated from 0 to 9) with computer demonstrations, each including an analysis of the computational results. At the top of each demo chapter, we summarize its content. Part II is based on the journal publications and one preprint. Here is a brief summary.

*Star products.* The phase space formulation of quantum mechanics avoids the formalism of operators on the Hilbert spaces of functions. Here, the operator multiplication is replaced by an associative non-commutative product defined for functions on the phase space; this intermediate construction was proposed by Groenewold *et al.* around 1946, building on earlier ideas by Weyl and Wigner. To be precise, we consider (formal) deformation quantization in the following sense.

**Definition 1.** A *star product* on a smooth real manifold  $M$  with algebra of smooth functions  $A = C^\infty(M)$  is a  $\mathbb{R}[[\hbar]]$ -bilinear associative product  $\star$  on the algebra of formal power series  $A[[\hbar]]$  that deforms the associative pointwise product on  $A$ , i.e.  $f \star g = f \cdot g + \sum_{n=1}^{\infty} \hbar^n B_n(f, g)$  for  $f, g \in A$ , in such a way that the  $B_n$  are bi-differential operators.

In physics one worries about the convergence of series, perhaps restricting the domain of the product to a subalgebra; besides, the formal parameter  $\hbar$  is replaced by  $\frac{i\hbar}{2}$ . In this dissertation we only consider formal series in  $\hbar$ , and we will not worry about convergence.

It is easily seen that the skew-symmetric part of  $B_1$  defined by  $B_1^-(f, g) = \frac{1}{2}(B_1(f, g) - B_1(g, f))$  is a Poisson bracket on  $A$ , i.e. a Lie bracket (bi-linear anti-symmetric bracket satisfying the Jacobi identity) which is also a bi-derivation with respect to the pointwise product. Hence the natural inverse problem is to construct a  $\star$ -product such that  $B_1^-$  equals a given Poisson bracket  $\{f, g\} = \frac{1}{2} \sum_{i,j=1}^n P^{ij} \cdot \partial_i(f) \cdot \partial_j(g)$ , where  $\partial_\ell := \frac{\partial}{\partial x^\ell}$  is the derivative with respect to a local coordinate  $x^\ell$  in a chart on  $M$ .

**Example 2.** For a generic Poisson bracket with coefficients  $P^{ij}$ , an analytic formula for



a  $\star$ -product modulo  $\bar{o}(\hbar^3)$ , with  $B_1$  equal to the Poisson bracket, is given by

$$\begin{aligned}
f \star g &= f \cdot g + \hbar P^{ij} \cdot \partial_i f \cdot \partial_j g + \hbar^2 \left( \frac{1}{2} P^{ij} \cdot P^{kl} \cdot \partial_k \partial_i f \cdot \partial_\ell \partial_j g + \frac{1}{3} \partial_\ell P^{ij} \cdot P^{kl} \cdot \partial_k \partial_i f \cdot \partial_j g \right. \\
&- \frac{1}{3} \partial_\ell P^{ij} \cdot P^{kl} \cdot \partial_i f \partial_k \cdot \partial_j g - \frac{1}{6} \partial_\ell P^{ij} \cdot \partial_j P^{kl} \cdot \partial_i f \cdot \partial_k g \left. \right) + \hbar^3 \left( \frac{1}{6} P^{ij} \cdot P^{kl} \cdot P^{mn} \cdot \partial_m \partial_k \partial_i f \cdot \partial_n \partial_\ell \partial_j g \right. \\
&- \frac{1}{6} \partial_m \partial_\ell P^{ij} \cdot \partial_n \partial_j P^{kl} \cdot P^{mn} \cdot \partial_i f \cdot \partial_k g - \frac{1}{6} P^{ij} \cdot \partial_n P^{kl} \cdot \partial_\ell P^{mn} \cdot \partial_k \partial_i f \cdot \partial_m \partial_j g \\
&- \frac{1}{6} \partial_m \partial_\ell P^{ij} \cdot \partial_n P^{kl} \cdot P^{mn} \cdot \partial_k \partial_i f \cdot \partial_j g - \frac{1}{6} \partial_m \partial_\ell P^{ij} \cdot \partial_n P^{kl} \cdot P^{mn} \cdot \partial_i f \cdot \partial_k \partial_j g \\
&+ \frac{1}{6} \partial_n \partial_\ell P^{ij} \cdot P^{kl} \cdot P^{mn} \cdot \partial_m \partial_k \partial_i f \cdot \partial_j g + \frac{1}{6} \partial_n \partial_\ell P^{ij} \cdot P^{kl} \cdot P^{mn} \cdot \partial_i f \cdot \partial_m \partial_k \partial_j g \\
&+ \frac{1}{3} \partial_n P^{ij} \cdot P^{kl} \cdot P^{mn} \cdot \partial_m \partial_k \partial_i f \cdot \partial_\ell \partial_j g - \frac{1}{3} \partial_n P^{ij} \cdot P^{kl} \cdot P^{mn} \cdot \partial_k \partial_i f \cdot \partial_m \partial_\ell \partial_j g \\
&- \frac{1}{6} \partial_\ell P^{ij} \cdot \partial_n \partial_j P^{kl} \cdot P^{mn} \cdot \partial_m \partial_i f \cdot \partial_k g + \frac{1}{6} \partial_n \partial_\ell P^{ij} \cdot \partial_j P^{kl} \cdot P^{mn} \cdot \partial_i f \cdot \partial_m \partial_k g \\
&\left. - \frac{1}{6} \partial_n P^{ij} \cdot P^{kl} \partial_\ell \cdot P^{mn} \cdot \partial_k \partial_i f \cdot \partial_m \partial_j g - \frac{1}{6} \partial_\ell P^{ij} \cdot \partial_n P^{kl} \cdot P^{mn} \cdot \partial_k \partial_i f \cdot \partial_m \partial_j g \right) + \bar{o}(\hbar^3),
\end{aligned}$$

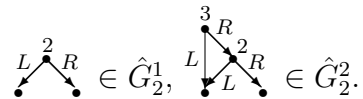
where the sum over all indices—each index running from 1 to  $\dim(M)$ —is implicit. This formula illustrates a major result by Kontsevich (1997) stating that there always exists a solution to the inverse problem of constructing  $\star = \star(P)$  on finite-dimensional affine Poisson manifolds  $(M, P)$ .

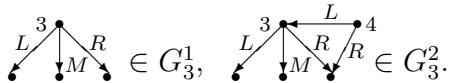
Inspired by the technique of Feynman diagrams, Kontsevich assigned formulas to the following class of graphs.

**Definition 3.** A *Formality graph* is a simple directed graph (that is, without double edges and without tadpoles) on  $m+n$  vertices  $\{0, \dots, m-1, m, \dots, m+n-1\}$ , such that the  $m$  *ground* vertices  $0, \dots, m-1$  are sinks (with no outgoing edges) and the  $n$  vertices  $m, \dots, m+n-1$  are called *aerial*. The set of edges of the graph is endowed with a total ordering.

In the graphs that we will meet in practice, the ground vertices will be drawn on the page along  $\mathbb{R} = \partial\mathbb{H}^2$  unlabeled from left to right. The aerial vertices inside  $\mathbb{H}^2$  will typically have two or three outgoing edges, which will be labeled  $L \prec R$  or  $L \prec M \prec R$  respectively. The total ordering on the set of edges is then inherited from the ordering of aerial vertices and the ordering of edges at each aerial vertex. In pictures of Formality graphs without edge labels, the edge ordering is by convention lexicographic.

**Notation 4.** The set of all Formality graphs with  $m$  ground vertices and  $n$  aerial vertices will be denoted by  $G_m^n$ . The subset of *Kontsevich graphs* built of wedges (with each aerial vertex having exactly two outgoing edges) will be denoted by  $\hat{G}_m^n \subset G_m^n$ .


**Example 5.** Kontsevich graphs built of wedges:   $\in \hat{G}_2^1, \in \hat{G}_2^2$ .

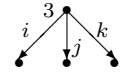
Formality graphs containing a tripod:   $\in G_3^1, \in G_3^2$ .

Formulas are associated with Formality graphs as follows.

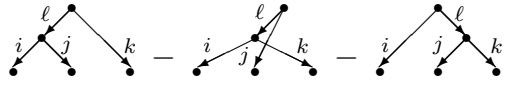
**Convention 6.** To the edges  $L$  and  $R$  of the wedge graph  $\Lambda = \text{wedge}(L, R)$  we ascribe independent indices  $i$  and  $j$  respectively, and with this graph  $\Lambda$  we associate the operator  $\Lambda(P)(f, g) = P^{ij} \cdot \partial_i f \cdot \partial_j g$  which is the Poisson bracket. More generally, for a Formality graph  $\Gamma \in G_m^n$  we ascribe independent indices to all edges; the multi-linear multi-differential operator  $\Gamma(J_0, \dots, J_{n-1})(f_0, \dots, f_{m-1})$  associated with the graph  $\Gamma$  is then a

sum over those indices, with each summand being a product over the (differentiated) contents of vertices, the ground vertex  $k$  containing the argument  $f_k$  of the operator, and the aerial vertex  $m + \ell$  containing the component of the multi-vector field  $J_\ell$  specified by indices of the ordered outgoing edges; here the content of each vertex is differentiated with respect to the local coordinates specified by the incoming edges (if any).

**Example 7.** To the Kontsevich graph  $\Gamma = \begin{array}{c} \bullet \\ \swarrow \quad \searrow \\ \bullet \quad \bullet \\ \swarrow \quad \searrow \\ \bullet \quad \bullet \end{array} \in \hat{G}_2^2$  we ascribe the indices  and hence with  $\Gamma$  we associate the operator  $\Gamma(P, P)(f, g) = P^{k\ell} \cdot \partial_\ell(P^{ij}) \cdot \partial_k \partial_i(f) \cdot \partial_j(g)$ .

To the Formality graph  $T := \begin{array}{c} \bullet \\ \swarrow \quad \searrow \\ \bullet \quad \bullet \\ \swarrow \quad \searrow \\ \bullet \quad \bullet \end{array} \in G_3^1$  we ascribe the indices  and hence with it we associate the operator  $T(B)(f, g, h) = B^{ijk} \cdot \partial_i(f) \cdot \partial_j(g) \cdot \partial_k(h)$ .

To the sum of Kontsevich graphs  $J := \begin{array}{c} \bullet \\ \swarrow \quad \searrow \\ \bullet \quad \bullet \\ \swarrow \quad \searrow \\ \bullet \quad \bullet \end{array} - \begin{array}{c} \bullet \\ \swarrow \quad \searrow \\ \bullet \quad \bullet \\ \swarrow \quad \searrow \\ \bullet \quad \bullet \end{array} - \begin{array}{c} \bullet \\ \swarrow \quad \searrow \\ \bullet \quad \bullet \\ \swarrow \quad \searrow \\ \bullet \quad \bullet \end{array}$  we ascribe

the indices  and hence with it we associate the operator

$$\begin{aligned} J(P, P)(f, g, h) &= (\partial_\ell(P^{ij}) \cdot P^{\ell k} - \partial_\ell(P^{ik}) \cdot P^{\ell j} - \partial_\ell(P^{jk}) \cdot P^{i\ell}) \cdot \partial_i(f) \cdot \partial_j(g) \cdot \partial_k(h) \\ &= \{\{f, g\}, h\} + \{\{h, f\}, g\} + \{\{g, h\}, f\}, \end{aligned}$$

which is the Jacobiator for  $P$ . We have the identity  $T(\frac{1}{2}[[P, P]])(f, g, h) = J(P, P)(f, g, h)$ , where  $[[\cdot, \cdot]]$  denotes the Schouten bracket.

**Theorem-Definition 8** (Kontsevich, 1997). *For every Poisson bi-vector  $P$  on a finite-dimensional affine real manifold  $M$  and an infinitesimal deformation  $\times \mapsto \times + \hbar\{\cdot, \cdot\}_P + \bar{o}(\hbar)$  towards the respective Poisson bracket, there exists a system of weights  $w(\Gamma)$ , uniformly given by an integral formula, such that the  $\mathbb{R}[[\hbar]]$ -bilinear star-product*

$$\star = \times + \sum_{n \geq 1} \frac{\hbar^n}{n!} \sum_{\Gamma \in \hat{G}_2^n} w(\Gamma) \cdot \Gamma(P, \dots, P)(\cdot, \cdot) \quad (1)$$

*is associative.*

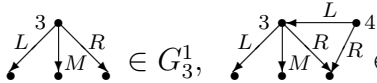
Elementary properties of the graph weights  $w(\Gamma)$  are summarized in [2, Lemma 1–5 and Remark 8], and the Shoikhet–Felder–Willwacher cyclic weight relations are recalled in [2, Proposition 7]. These relations are not enough to determine the weights completely. Another ample source of relations between weights is the associativity of  $\star$ ; this can be exploited as in [2, Method 1–3]. To make these methods effective, the evaluation of operators associated with graphs (having Poisson structures implanted into them) is implemented in software. In this way, a system of equations is formed with the weights  $w(\Gamma)$  as unknowns. Up to  $\bar{o}(\hbar^4)$ , the above methods express the weights of all graphs in terms of just 10 (and up to gauge transformations, just 6) parameters. The weights of those remaining 10 graphs are imported from Banks–Panzer–Pym (2017).

**Proposition 9.** *The analytic formula for the Kontsevich star product modulo  $\bar{o}(\hbar^4)$  is displayed in Chapter 11, Eq (11) in Conclusion thereof. The encodings of all the graphs in  $\star \bmod \bar{o}(\hbar^4)$  together with their coefficients are given in Encoding 1 in Appendix B.1.*

Out of the 247 graphs showing up in the Kontsevich  $\star$ -product at order four, as many as 138 contain two-cycles [2, Appendix A.2]. We are now in a position to inspect whether this  $\star$ -product will be associative for a generic Poisson structure, in arbitrary dimension. In this direction we make use of the following lemma.

**Lemma 10** (Lemma 1 in [1]). *A tri-differential operator  $\sum_{|I|,|J|,|K|\geq 0} c^{IJK} \partial_I \otimes \partial_J \otimes \partial_K$  vanishes identically iff all its coefficients vanish:  $c^{IJK} = 0$  for every triple  $(I, J, K)$  of multi-indices; here  $\partial_L = \partial_1^{\alpha_1} \circ \dots \circ \partial_n^{\alpha_n}$  for a multi-index  $L = (\alpha_1, \dots, \alpha_n)$ . Moreover, the sums  $\sum_{|I|=i,|J|=j,|K|=k} c^{IJK} \partial_I \otimes \partial_J \otimes \partial_K$  are then zero for all homogeneity orders  $(i, j, k)$ .*

**Definition 11.** A *Leibniz graph* is a Formality graph containing at least one aerial vertex with three outgoing edges, such that those three edges have three distinct targets, and none of those three edges are tadpoles. The other aerial vertices (if any) have two outgoing edges, and the ground vertices are as usual. These graphs will be evaluated with the Jacobiator  $\frac{1}{2}[[P, P]]$  of the Poisson structure  $P$  in the vertex with three outgoing edges, hence representing a differential operator that is identically zero whenever  $P$  is Poisson.

**Example 12.** Leibniz graphs:   $\in G_3^1$ ,  $\in G_3^2$ .

We recall from Kontsevich (1997) the guaranteed existence of a factorization of the star-product associator via Leibniz graphs. To the best of our knowledge, nobody checked the general mechanism of associativity explicitly before. In [3] we analyze the mechanism and illustrate it in detail (see section 5 in [3]). An (earlier) explicit example of the factorization of the associator up to order 3 is in [1].

**Proposition 13** (Corollary 4 and Conjecture ending §4 in [3]). *The operator  $\diamond$  that solves the factorization problem*

$$\text{Assoc}(\star)(P)(f, g, h) = \diamond(P, [[P, P]])(f, g, h), \quad f, g, h \in A[[\hbar]], \quad (2)$$

is given by

$$\diamond = 2 \cdot \sum_{n \geq 1} \frac{\hbar^n}{n!} \cdot c_n \cdot \mathcal{F}_{n-1}([[P, P]], P, \dots, P). \quad (3)$$

where  $\mathcal{F}_k$  is the  $k$ -ary component of the Formality  $L_\infty$ -morphism, and where we claim that the constants  $c_n$  are equal to  $n/6$ .

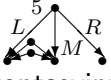
The number of graphs which actually show up at order  $\hbar^k$  in the left and right-hand sides of factorization problem (2) is reported in Table 0.1.

**Table 0.1:** Number of graphs in either side of the associator's factorization.

$k$	2	3	4	5	6	7
LHS: # Kontsevich graphs, coeff $\neq 0$	3 (Jac)	39	740	12464	290305	?
RHS: # Leibniz graphs, coeff $\neq 0$	1 (Jac)	13	241	4609	?	?

In [3, Section 5] we inspect many graphs of different orders, and establish the equality of sums of Kontsevich graphs in the associator and and sums of Leibniz graphs—in the factorizing operator  $\diamond$ —after they are expanded into the Kontsevich graphs.

**Example 14** (Example 8 in [3] and Cells 56–58 in Section 3.4). The Leibniz graph

$L_{331} :=$   of differential orders (3, 3, 1) has the weight 1/24 according to Panzer's kontsevint. Multiplied by a universal (for all graphs at  $\hbar^4$ ) factor  $2^4 = 16$  and the factor  $1/(\# \text{Aut}(L_{331})) = 1/2$  due to this graph's symmetry ( $3 \rightleftharpoons 4$ ), it expands to  $\frac{1}{3}([01; 01; 01; 52] + [01; 01; 12; 50] + [01; 01; 20; 51])$  by the definition of Jacobi's identity. This sum of three weighted Kontsevich oriented graphs reproduces exactly the component  $A_{331}^{(4)}$  of homogeneity order (3, 3, 1) in the associator at  $\hbar^4$ , which is known from [2, Table 8 in App. D].

In the right-hand side of the associator for  $\star$ , there are Leibniz graphs: at  $\hbar^{k \geq 2}$ , such Leibniz graphs have 3 sinks,  $k - 1$  aerial vertices (of which one vertex, the Jacobiator, has three outgoing edges, and the remaining  $k - 2$  vertices (if any) each have two outgoing edges), and, by the above,  $3 + (k - 2) \cdot 2 = 2k - 1$  edges; tadpoles are not allowed, graphs with multiple edges are discarded. For each  $k = 2, 3, 4, 5$  we generate all such admissible Leibniz graphs (those can be zero graphs with a parity-reversing automorphism); the respective number of such Leibniz graphs at each order  $\hbar^k$  is in Table 0.2. At every order  $k$ , we generate the entire set of the cyclic weight relations (Willwacher–Felder (2008), Shoikhet (2000)); every cyclic weight relation is a linear constraint upon the weights of several Leibniz graphs (all those weights are given by the Kontsevich integral formula (1997)). The number of these linear relations and the (co)rank of this linear algebraic system follow in Table 0.2.

**Table 0.2:** The count of admissible Leibniz graphs in the associator for Kontsevich's  $\star$ .

$k$	2	3	4	5	6
# Leibniz graphs, generated	1	24	520	11680	293748
# Leibniz graphs generated, nonzero	1	24	490	11260	285684
# Leibniz graphs generated, nonzero, diff. order $> 0$	1	15	301	6741	171528
• of them, with in-degree(aerial vertices) $\leq 1$	1	15	177	1573	12045
# Leibniz graphs (coeff $\neq 0$ in associator)	1	13	241	4609	?
# Cyclic weight relations	1	15	301	6741	171528
Corank of linear algebraic system	0	3	66	1469	?

Banks–Panzer–Pym do not list the weights of Leibniz graphs (as in Table 0.2 above), for these graphs do not show up in the  $\star$ -product itself where the vertex-edge valency is different for the Kontsevich graphs. We use the software by Banks–Panzer–Pym (2018) to calculate the Kontsevich weights of all the Leibniz graphs which are admissible for the right-hand side of star-product's associator. (Some weights can—and actually do—vanish because either the graph is zero, or the weight integrand is identically zero, or the weight formula integrates to a zero number.) The count of admissible Leibniz graphs with nonzero weights is in the fourth line of Table 0.2: the corresponding line in Table 0.1 is reproduced verbatim. (The Leibniz graphs with zero weights do nominally show up in the cyclic weight relations for Leibniz graphs, but in fact stay invisible in the formulas.)

**Proposition 15** (Cells 56–70 in Chapter 3). *The numeric values of the Kontsevich weights of Leibniz graphs with  $k$  aerial vertices on 3 sinks, which we calculated using*

Panzer’s *kontsevint*, do satisfy the system of linear algebraic equations given by the cyclic weight relations, for  $k = 1, 2, 3, 4$ .<sup>8</sup>

For the Kontsevich graphs admissible for the  $\star$ -product at  $\hbar^n$ , that is on two sinks, on  $n$  aerial vertices and  $2n$  edges (from  $n$  wedges), all of the above is repeated for  $n = 1, 2, 3, 4, 5, 6, 7$ . The various counts of Kontsevich graphs and (co)ranks of the cyclic weight relation systems are in Table 0.3.

**Table 0.3:** The count of admissible Kontsevich graphs in the  $\star$ -product.

$n$	1	2	3	4	5	6	7
# Kontsevich graphs, generated	1	6	44	475	6874	126750	2814225
# Kontsevich generated, nonzero	1	6	38	445	6488	122521	2744336
# Kontsevich generated, nonzero, diff. order $> 0$	1	4	30	331	4907	91694	2053511
# Kontsevich generated, nonzero, diff. order $> 0$ , connected	1	4	30	330	4893	91489	2049704
• of them, with in-degree(aerial vertices) $\leq 2$	1	4	30	265	2801	33690	451927
• of them, prime	1	3	24	215	2327	28649	391958
• of them, with in-degree(aerial vertices) $\leq 1$	1	4	14	51	161	542	1723
• of them, prime	1	3	8	23	59	171	477
# Kontsevich graphs (coeff $\neq 0$ in $\star$ at $\hbar^n$ )	1	4	13	247	2356	66041	?
• of them, with in-degree(aerial vertices) $\leq 1$	1	4	6	35	84	334	958
# Cyclic weight relations	1	4	30	331	4907	91694	2053511
Corank of linear algebraic system	0	1	11	103	1561	?	?

Now for such admissible Kontsevich graphs, the values of their weights can be directly imported from the on-line *kontsevint* repository of Panzer.<sup>9</sup> We compose the linear algebraic system of cyclic weight relations, but now, we merge these systems with many other linear equations (upon the weights) that stem from the associativity of  $\star$ , as well as from the elementary properties of graph weights such as mirror reflections. In April 2017, we submitted the agglomerated system of linear algebraic constraints upon the weights of Kontsevich graphs in  $\star \bmod \bar{o}(\hbar^5)$  to the developers of *kontsevint*; Banks–Panzer–Pym confirmed that all of the relations are satisfied by the weight values found by using their own software. We verified this independently by using our software:

**Proposition 16** (Cells 37–55 in Chapter 3 and Chapter 11). *The numeric values of the weights of Kontsevich graphs with  $\leq 5$  aerial vertices on 2 sinks, generated by Banks–Panzer–Pym (2018), do satisfy the entire system of constraints given by the basic properties [2, Lemma 1–5], the cyclic weight relations up to order 5, and the system of equations obtained by restricting the associativity of the Kontsevich  $\star$ -product  $\bmod \bar{o}(\hbar^5)$  to the 3D rescaled Nambu–Poisson structure.*

In Chapter 3, further links to externally stored plain text files are available: the files contain graphs, weights, and relations.

<sup>8</sup>The relations are satisfied *exactly*, without involvement of any conventional recalculating constants and normalizations (in contrast with the mandatory use of auxiliary constants  $c_n = n/6$  in Proposition 13, see above). But let us remember that the multiplicativity of Kontsevich weights is more subtle for graphs on three ground vertices than for Kontsevich’s graphs on two sinks.

<sup>9</sup><https://bitbucket.org/PanzerErik/kontsevint/>

**Proposition 17** (Cells in Appendix B.2). *The Kontsevich  $\star$ -product with the harmonic graph weights, known up to  $\bar{o}(\hbar^6)$  from Banks–Panzer–Pym (2018), is associative modulo  $\bar{o}(\hbar^6)$ : every tri-differential homogeneous component of the associator admits some realization by Leibniz graphs; to find every such solution, the 0th layer of Leibniz graphs suffices for each of the tri-differential orders.*

*Proof scheme.* The associativity of Kontsevich’s  $\star$ -product up to  $\bar{o}(\hbar^4)$ , that is,  $\text{Assoc}(\star(P))(f, g, h) \bmod \bar{o}(\hbar^4) = \diamond(P, \llbracket P, P \rrbracket)(f, g, h) \bmod \bar{o}(\hbar^4)$ , is the core of paper [2], which is Chapter 11 in Part II below; see also §3.5.1 in Part I. Next, in §3.5.1 we provide a realization of the component  $\sim \hbar^5$  in the associator  $\text{Assoc}(\star) \bmod \bar{o}(\hbar^5)$  in terms of the Leibniz graphs from the 0th layer, that is, by using the Leibniz graphs obtained at once by contracting edges between aerial vertices in the Kontsevich graphs from the associator. (We keep in mind that the representability of the associator by using the 0th layer Leibniz graphs is previewed in the proof of Kontsevich’s Formality theorem, and we seek to illustrate this.)

There are 105 homogeneous tri-differential order components at  $\hbar^6$  in the associator  $\text{Assoc}(\star) \bmod \bar{o}(\hbar^6)$ . We import the harmonic graph weights at  $\hbar^5$  and  $\hbar^6$  in  $\star \bmod \bar{o}(\hbar^6)$  from the kontsevint repository of E. Panzer (Oxford). At order  $\hbar^6$ , the weights of Kontsevich graphs in  $\star$  are expressed as  $\mathbb{Q}$ -linear combinations of 1 and  $\zeta(3)^2/\pi^6$ . In consequence, the coefficients of Kontsevich graphs in the associator at order  $\hbar^6$  are also  $\mathbb{Q}$ -linear combinations of that kind. Every tri-differential homogeneous component of the associator is thus split into the rational- and  $\zeta(3)^2/\pi^6$ -slice: either of the slices is a linear combination of Kontsevich’s graphs with rational coefficients. The rational slices are met in all of the 105 tri-differential orders; we detect that in every such slice the Kontsevich graphs provide the 0th layer of Leibniz graphs, which suffices to realize that sum of Kontsevich graphs as a linear combination of these Leibniz graphs. The  $\zeta(3)^2/\pi^6$ -slice is nontrivial in 28 tri-differential orders of the associator at  $\hbar^6$ ; here the Formality mechanism works as follows. For all but 6 tri-differential orders, the Kontsevich graphs from the linear combination near  $\zeta(3)^2/\pi^6$  suffice to provide the set of 0th layer Leibniz graphs which are enough for a solution of the factorization problem. The tri-differential orders  $\{(1, 1, 3), (3, 1, 1), (2, 1, 2), (1, 2, 2), (2, 2, 1), (1, 3, 1)\}$  are special: for a solution to appear, the sets of Kontsevich graphs from the rational and  $\zeta(3)^2/\pi^6$ -slices within that tri-differential order must be merged and then the union set is enough to provide a factorization of the  $\zeta(3)^2/\pi^6$ -slice by the 0th layer of Leibniz graphs. The corresponding computations are presented in Appendix B.2. We conclude that at order 6 for the full Kontsevich star product, Kontsevich’s Formality mechanism works as expected.  $\square$

All of the above was true for arbitrary Poisson structures (on affine finite-dimensional real manifolds). For the class of Poisson brackets with *affine* coefficients (whose higher derivatives vanish identically), e.g. the Kirillov–Kostant linear brackets, we advance further in the expansion of the Kontsevich  $\star$ -product.

Indeed, the restriction of Kontsevich’s  $\star$ -product to the spaces of affine Poisson brackets is well-defined: all the Kontsevich graphs in  $\star_{\text{aff}} \bmod \bar{o}(\hbar^n)$  and in its associator up to  $\bar{o}(\hbar^n)$  only have aerial vertices with in-degree  $\leq 1$ . The linear algebraic system of the Shoikhet–Felder–Willwacher cyclic weight relations is by construction triangular with respect to the weights of Kontsevich graphs (in  $\star$ ) with an overall bound for the in-degrees of aerial vertices. (The linear system of cyclic weight relations is also triangular with respect to the in-degrees of aerial vertices in the Leibniz graphs which can be used to express the associator via differential consequences of the Jacobi identity.)



**Proposition 18** (see Section 3.7). *The encoding and analytic formula of Kontsevich’s affine star product  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$ —in particular, for all the Kirillov–Kostant Poisson brackets, linear on the duals  $\mathfrak{g}^*$  of finite-dimensional Lie algebras—is given in Appendix C. There are 1423 nonzero Kontsevich weights of affine Kontsevich graphs in  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$  terms overall at all orders  $\leq 7$ . The multiple zeta value  $\zeta(3)^2/\pi^6$  starts appearing in the weights at  $n \geq 6$  vertices.<sup>10</sup>*

*Proof scheme.* The ansatz for  $\star_{\text{aff}} \bmod \bar{o}(\hbar^n)$  contains, at  $\hbar^7$ , 1731 affine Kontsevich graphs with in-degree  $\leq 1$  of aerial vertices; their Kontsevich weights are constrained by the elementary properties (such as mirror reflections, whence *basic* graphs), by the weights multiplicativity (whence the *prime* graphs), by the vanishing statements for the Kontsevich graphs which are disconnected over the sinks, and for the Kontsevich graphs which contain a triangle subgraph standing on a sink (as in Example 23 on p. 25 below), and by the cyclic weight relations: the corank of the merged linear algebraic system upon the 1731 unknowns equals 76. The restriction of the associator for  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$  to a generic affine Poisson bracket  $P = (ax + by + c)\partial_x \wedge \partial_y$  on  $\mathbb{R}^2$  decreases the corank down to 74. The values of the 76 master parameters (themselves the weights of certain affine Kontsevich graphs on  $n = 7$  aerial vertices in the affine star-product  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$ ) have been computed using the `kontsevint` program by Banks–Panzer–Pym; these values are listed in Cell 54 in Section 3.7 below.  $\square$

This affine star product expansion is associative up to  $\bar{o}(\hbar^7)$ :

**Proposition 19** (Cells 58–71 in Section 3.7). *The affine Kontsevich star product expansion  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$  found in Proposition 18 above is associative modulo  $\bar{o}(\hbar^7)$ . Namely, (every homogeneous tri-differential component of) the associator  $(f \star_{\text{aff}} g) \star_{\text{aff}} h - f \star_{\text{aff}} (g \star_{\text{aff}} h) \bmod \bar{o}(\hbar^7)$  is realized as some sum of Leibniz graphs.*

*Proof scheme.* With not yet specified undetermined coefficients of Kontsevich graphs at  $\hbar^7$  in the affine star product  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$ , its associator’s part at  $\hbar^7$  expands to 203 tri-differential order components. As soon as the weights of all the new Kontsevich graphs on  $n = 7$  aerial vertices are fixed by Proposition 18, the number of tri-differential orders  $(d_0, d_1, d_2)$  actually showing up at  $\hbar^7$  in the associator  $\mathbf{A}$  for  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$  drops to 161. For all but four tri-differential order components  $\mathbf{A}_{d_0 d_1 d_2}$  in the associator  $\mathbf{A}$ , the 0th layer of Leibniz graphs, which are obtained by contracting<sup>11</sup> one edge between aerial vertices in the Kontsevich graphs of every such tri-differential component  $\mathbf{A}_{d_0 d_1 d_2}$ , is enough to provide a solution for the factorization problem,  $\mathbf{A}_{d_0 d_1 d_2} = \diamond_{d_0 d_1 d_2}(P, \llbracket P, P \rrbracket)$ , expressing that component by using differential consequences of the Jacobi identity (encoded by Leibniz graphs). We detect that for the tri-differential orders  $(d_0, d_1, d_2)$  in the set  $\{(3, 3, 2), (2, 3, 3), (3, 2, 3), (2, 4, 2)\}$ , the Leibniz graphs from the 0th layer are not enough to reach a solution  $\diamond_{d_0 d_1 d_2}$ ; still a solution  $\diamond_{d_0 d_1 d_2}$  appears in each of these four exceptional cases after we add the Leibniz graphs from the 1st layer (i.e. those graphs obtained by contraction of edges in the Kontsevich graph expansion of Leibniz graphs from the previous layer; see [10]). (There are 2294 Kontsevich graphs in  $\mathbf{A}_{2,3,3}$ , producing 3584 Leibniz graphs in the respective 0th layer immediately after the edge contractions; the component  $\mathbf{A}_{3,3,2}$  contains equally many Kontsevich graphs and the same number of

<sup>10</sup>The Kontsevich weight of the Felder–Willwacher affine graph (2008) equals  $\frac{13}{2903040} - \frac{1}{256}\zeta(3)^2/\pi^6$ , thus now correcting a typo in the `kontsevint` program description by Banks–Panzer–Pym.

<sup>11</sup>Note that the Leibniz graphs in the 0th layer have vertices of in-degree  $\leq 2$  because they are obtained by the contraction of a single edge in the Kontsevich graphs with vertices of in-degree  $\leq 1$ .

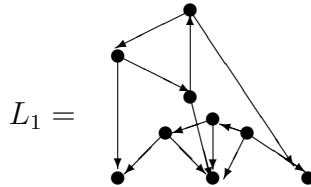
Leibniz graphs in the 0th layer; the largest component  $A_{3,2,3}$  contains 2331 Kontsevich graphs and gives 3603 Leibniz graphs in the 0th layer; and finally  $A_{2,4,2}$  contains 1246 Kontsevich graphs and produces 2041 Leibniz graphs in the 0th layer.) In Section 3.7.8 of Part I below we generate a Leibniz graph factorization of *all* tri-differential components in the associator for  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$  and we provide the data files of Leibniz graphs and their coefficients.  $\square$

*Proof scheme (for the reduced affine star product  $\star_{\text{aff}}^{\text{red}} \bmod \bar{o}(\hbar^7)$ ).* The reduced affine star product  $\star_{\text{aff}}^{\text{red}} \bmod \bar{o}(\hbar^7)$  is obtained from the affine star product  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$  by realizing the coefficient of  $\zeta(3)^2/\pi^6$  as the Kontsevich graph expansion of a linear combination of Leibniz graphs with rational coefficients and, now that this combination does not contribute to either the star product or its associator when restricted to any affine Poisson structure, by discarding this part of  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$  proportional to  $\zeta(3)^2/\pi^6$ . In the reduced affine star product  $\star_{\text{aff}}^{\text{red}} \bmod \bar{o}(\hbar^7)$  there remain only 326 nonzero rational coefficients of Kontsevich graphs at  $\hbar^k$  for  $k = 0, \dots, 7$  (in contrast with 1423 nonzero (ir)rational coefficients at orders up to  $\hbar^7$  in  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$ ).

The associator for  $\star_{\text{aff}}^{\text{red}}$  contains 95 tri-differential orders at  $\hbar^6$  and 161 tri-differential orders at  $\hbar^7$ . We see that the associator  $\text{Assoc}(\star_{\text{aff}}^{\text{red}}) \bmod \bar{o}(\hbar^7)$  becomes much smaller than  $\text{Assoc}(\star_{\text{aff}}) \bmod \bar{o}(\hbar^7)$ , now containing only 29371 Kontsevich graphs instead of 59905. But the work of the associativity mechanism for  $\star_{\text{aff}}^{\text{red}}$  requires the use of the 1st and higher layer(s) of Leibniz graphs much more often than it already was for the affine star product  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$  before the reduction. Now, at orders  $\leq 7$  in  $\hbar$ , new Leibniz graphs from the layer(s) beyond the 0th are indispensable for the factorization of 114 out of 336 homogeneous tri-differential order components of the associator, see Appendix C.3 where we list all these exceptional orders.

We observe that the number  $\zeta(3)^2/\pi^6$ , not showing up in any restriction of the affine star product  $f \star_{\text{aff}} g \bmod \bar{o}(\hbar^7)$  to an affine Poisson structure and any arguments  $f, g \in C^\infty(M)[[\hbar]]$ , acts in effect as a placeholder of the Kontsevich graphs which, by contributing to the associator and then creating the Leibniz graphs by edge contraction, provide almost all of the Leibniz graphs needed for a factorization of the associator for  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$  via the Jacobi identity. When the  $\zeta(3)^2/\pi^6$ -part of  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$  itself is eliminated by using the Jacobi identity for affine Poisson structures, the remaining  $\star_{\text{aff}}^{\text{red}} \bmod \bar{o}(\hbar^7)$  and its associator rely heavily on the use of higher layers of Leibniz graphs for a factorization solution to be achieved.  $\square$

**Example 20.** Consider the Leibniz graph



on three sinks 0, 1, 2 with  $\tilde{n} = 6$  aerial vertices, and with edges  $[(3, 2), (3, 7), (4, 1), (4, 8), (5, 1), (5, 3), (6, 1), (6, 2), (6, 4), (7, 0), (7, 5), (8, 0), (8, 1)]$ . This Leibniz graph is needed for the factorization of the tri-differential component of order  $(2, 4, 2)$  at  $\hbar^7$  in the associator for  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$ . This graph appears only in the 1st layer of Leibniz graphs, not in the 0th layer, as we contract edges of Kontsevich's graphs on  $n = 7$  aerial vertices in the associator for  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$ , and as we expand the resulting Leibniz



graphs to the old and possibly new Kontsevich graphs.<sup>12</sup> This Leibniz graph created in the 1st layer appears with coefficient  $2/135$  in an iteratively found factorization of the associator. The genuine Kontsevich weight of this Leibniz graph calculated by using the program `kontsevint` by E. Panzer is also nonzero:  $w(L_1) = -3/128 \cdot \frac{\zeta(3)^2}{\pi^6} + 31/725760$ . The actual coefficient of  $L_1$  in the *canonical* factorization of the associator, as guaranteed by the Formality Theorem, equals  $w(L_1)$  multiplied by some nonzero constant. The discrepancy between the found rational value in *some* solution and the (ir)rational value in Kontsevich’s canonical solution is likely due to an identity between Leibniz graphs which expand to a zero sum of Kontsevich graphs (see §3.5.2 on p. 93 below). But anyway, based on this empiric evidence we conclude that the 0th layer of Leibniz graphs is *not enough* to provide a factorization of the associator for the (either affine or full) Kontsevich star product at order  $\hbar^7$ , whereas, according to Proposition 17 above, the 0th layer of Leibniz graphs *was enough* at order  $\hbar^6$  to factor the associator for the full star product.

**Remark 21.** The above iterative scheme gives us *a* solution to the weak factorization problem: each tri-differential component  $A_{d_0 d_1 d_2}$  is factorized independently from the others, so that the coefficients of the Leibniz graphs are not yet constrained overall—over different components—by the Shoikhet–Felder–Willwacher cyclic weight relations and other relations. In particular, the above scheme does not guarantee that the found coefficients of Leibniz graphs are equal (up to the multiplicity and recalculation constants) to the genuine Kontsevich weights of those Leibniz graphs. The above scheme provides the necessary minimum number of layers of Leibniz graphs, whereas the calculation of Kontsevich’s genuine weights of Leibniz graphs is sufficient to build a solution (the canonical one) for the associator’s factorization problem. We remember that there exist identities, i.e. sums of Leibniz graphs which expand to zero sums of Kontsevich graphs (here, in the associator); such identities could make unnecessary the use of a Leibniz graph with nonzero genuine weight from a (high number — in particular the last) layer. Hypothetically it might be that any solution needs the 0th and 1st layers, hence they are “necessary”, but Kontsevich’s canonical solution stretches over the 0th, 1st and 2nd layers, thus they are “sufficient”. In conclusion, the above scheme does not guarantee that the genuine Kontsevich weight of a Leibniz graph in the known associator’s factorization at order  $\hbar^7$  will definitely be equal (up to the multiplicity and recalculation constants) to this Leibniz graph’s coefficient in the last necessary layer.

In section 2.5 of [2] and in Chapters 1 and 9 below, we study gauge transformations of star products.

**Definition 22.** Let  $\star: A[[\hbar]] \times A[[\hbar]] \rightarrow A[[\hbar]]$  be a star-product, where  $A = C^\infty(M)$ . A *gauge transformation* is an  $\mathbb{R}[[\hbar]]$ -linear map  $T: A[[\hbar]] \rightarrow A[[\hbar]]$  of the form  $f \mapsto f + \hbar D_1(f) + \hbar^2 D_2(f) + \dots$  for  $f \in A$ , where the  $D_i$  are differential operators; by construction,  $T$  is formally invertible. The star-product  $\star'$  defined by  $f \star' g = T(T^{-1}(f) \star T^{-1}(g))$  is called *gauge equivalent* to  $\star$ .

**Example 23** (Examples 23 and 24 in [2]). The map defined by  $\bullet \mapsto \bullet + \frac{\hbar^2}{12} \text{⚡}$  or  $f \mapsto f + \frac{\hbar^2}{12} \partial_k P^{ij} \partial_j P^{kl} \partial_i \partial_l f$  for  $f \in C^\infty(M)$  is a gauge transformation. When applied to

<sup>12</sup>This Leibniz graph cannot originate from any Kontsevich graph in the associator itself — even with aerial vertex in-degree  $\geq 2$ . Namely, all candidate Kontsevich graphs are composite with one of the factors having zero weight.

the Kontsevich  $\star$ -product, the respective gauge-equivalent product  $\star'$  contains no graph with loop at  $\hbar^2$ .

Gauge transformations enable us to bring down the number of unknown parameters in  $\star \bmod \bar{o}(\hbar^4)$  from 10 down to 6 in [2, Theorem 14]. Independently, gauge transformations allow us to verify claims about (non)equivalence of star products. In section 4 of [2] we compare some earlier calculations of  $\star$ -product expansions up to  $\bar{o}(\hbar^3)$  with the authentic Kontsevich  $\star$ -product expansion. For more details about gauge transformations, see Chapter 9.

*Poisson flows.* We want to deform Poisson structures in such a way that they stay Poisson.

**Definition 24.** Let  $P$  be a Poisson bi-vector on the manifold  $M$  at hand and consider its deformation  $P + \varepsilon Q + \bar{o}(\varepsilon)$  where  $Q$  is a bi-vector and  $\varepsilon$  is a formal parameter. We say that after such deformation the bi-vector stays *infinitesimally Poisson* if  $\llbracket P + \varepsilon Q + \bar{o}(\varepsilon), P + \varepsilon Q + \bar{o}(\varepsilon) \rrbracket = \bar{o}(\varepsilon)$ , that is if  $\llbracket P, Q \rrbracket = 0$ . The deformation  $P + \varepsilon Q + \bar{o}(\varepsilon)$  is called *trivial* if  $Q$  is a coboundary in the Poisson complex w.r.t. the differential  $\partial_P = \llbracket P, - \rrbracket$ , i.e. if there exists a vector field  $X$  such that  $Q = \llbracket P, X \rrbracket$ .

The existence and classification of (non)trivial deformations of a Poisson structure  $P$  naturally depends strongly on the manifold  $M$  and the Poisson structure  $P$ . Nevertheless we can ask if there exist *universal* deformations, in the sense of a general formula or recipe  $P \mapsto P + \varepsilon Q(P) + \bar{o}(\varepsilon)$ , which is defined for all (affine) manifolds in terms of the coefficients  $P^{ij}$  of the Poisson bi-vector. The existence of such a formula would then also require a universal proof of the Poisson cocycle condition  $\llbracket P, Q(P) \rrbracket = 0$ .

**Example 25.** As a byproduct of Kontsevich's *Formality Conjecture* (1996), the formula

$$\begin{aligned} Q_{\text{tetra}}(P) = & 1 \cdot \left( \frac{\partial^3 P^{ij}}{\partial x^k \partial x^\ell \partial x^m} \frac{\partial P^{kk'}}{\partial x^{\ell'}} \frac{\partial P^{\ell\ell'}}{\partial x^{m'}} \frac{\partial P^{mm'}}{\partial x^{k'}} \right) \frac{\partial}{\partial x^i} \wedge \frac{\partial}{\partial x^j} \\ & + 6 \cdot \left( \frac{\partial^2 P^{ij}}{\partial x^k \partial x^\ell} \frac{\partial^2 P^{km}}{\partial x^{k'} \partial x^{\ell'}} \frac{\partial P^{k'\ell}}{\partial x^{m'}} \frac{\partial P^{m'\ell'}}{\partial x^j} \right) \frac{\partial}{\partial x^i} \wedge \frac{\partial}{\partial x^m} \end{aligned}$$

defines a universal deformation of Poisson structures. Here the balance 1 : 6 is necessary, as shown by experiment with 3D rescaled Nambu–Poisson structures in [6]. A very detailed illustration of the pictorial proof of the Poisson cocycle condition's factorization  $\llbracket P, Q_{\text{tetra}}(P) \rrbracket = \diamond(P, \text{Jac}(P))$ , with the Kontsevich graphs in the left-hand being expressed as the expansion of sums of Leibniz graphs in the right-hand side, is given in [6].

In fact there are more such universal flows, originating from cocycles  $\gamma \in \ker d$  in the Kontsevich graph complex (which will be discussed soon, see p. 31 below).

**Proposition 26** (Theorem 1 and Corollary 3 in [8]). *Whenever  $P$  is a Poisson bi-vector so that the Schouten bracket  $\pi_S(P, P)$  vanishes ( $\llbracket P, P \rrbracket = 0$ ), and whenever  $\gamma \in \ker d$  is a cocycle on  $k$  vertices and  $2k - 2$  edges (so that  $d(\gamma) = [\bullet\bullet, \gamma] = 0$  in Kontsevich's unoriented graph complex), then  $\text{Of}(\gamma)(P^{\otimes k})$  is a Poisson 2-cocycle (so that  $\llbracket P, \text{Of}(\gamma)(P^{\otimes k}) \rrbracket \doteq 0$  modulo the Jacobi identity  $\frac{1}{2}\llbracket P, P \rrbracket = 0$  for the Poisson structure). The operator  $\diamond$  in the factorization problem*

$$\partial_P(\text{Of}(\gamma)(P^{\otimes k})) = \diamond(P, \llbracket P, P \rrbracket), \quad \gamma \in \ker d,$$

is the sum of Leibniz graphs obtained from the graph cocycle  $\gamma$  by inserting the Jacobiator  $\frac{1}{2}[[P, P]]$  into one of its vertices (by the Leibniz rule) and skew-symmetrizing w.r.t. the sinks.

Proposition 26 is proved and illustrated explicitly in [8] and references therein: for the tetrahedron  $\gamma_3$  (see also Example 27 below), the five-wheel cocycle  $\gamma_5$ , an example of a coboundary  $\delta_6 = d(\beta_6)$ , and the heptagon-wheel cocycle  $\gamma_7$ .<sup>13</sup> In particular the formula for  $\diamond$  is exact, giving the factorizing operator for the Poisson cocycle condition in each case (see Table 0.4). Each of these canonical factorizations is also given in Chapter 5 below.

**Example 27** (see Equation (11) in [6] and Cells 16–22 in Chapter 5). For the tetrahedral flow we have

$$\begin{aligned}
 \diamond = & \text{Diagram 1} + 3 \sum_{\tau \in \mathcal{S}_2} (-)^\tau \text{Diagram 2} + 3 \sum_{\circlearrowleft} \text{Diagram 3} \\
 & + 3 \sum_{\circlearrowright} \left\{ \text{Diagram 4} + \text{Diagram 5} + \text{Diagram 6} \right\} \\
 & + 3 \sum_{\sigma \in \mathcal{S}_3} (-)^\sigma \left\{ \text{Diagram 7} + \text{Diagram 8} \right\}.
 \end{aligned}$$

The flow associated with a graph  $d$ -coboundary  $\delta = d(\beta)$  in the graph complex is universally Poisson-trivial; the vector field associated with  $\beta$  being the trivializing vector field [10, Corollary 4]. Flows associated with graph  $d$ -cocycles which are not coboundaries (e.g.,  $\gamma_3, \gamma_5, \gamma_7, [\gamma_3, \gamma_5], \gamma_9$ ) are not universally Poisson-trivial in that particular way. Still the question remains whether there exists a Poisson structure on an affine manifold on which the graph complex truly acts nontrivially.

In this direction we can say more when we restrict ourselves to particular (classes of) Poisson structures. This is the subject of Chapters 6, 7, and 8 below. The classes will be arbitrary bi-vectors on  $\mathbb{R}^2$ , as well as rescaled Nambu–Poisson structures  $P = \rho(x, y, z) da / (dx \wedge dy \wedge dz)$  on  $\mathbb{R}^3$  (and similar brackets on  $\mathbb{R}^4$ ), and finally, Poisson brackets constructed by using  $R$ -matrices.

<sup>13</sup>Pictures of these graph cocycles are drawn in Example 41 on p. 32 below.

**Table 0.4:** The number of graphs in the problem  $\llbracket P, \text{Or}\vec{\gamma}(P) \rrbracket = \diamond(P, \llbracket P, P \rrbracket)$ .

Cocycle:	$\gamma_3$	$\gamma_5$	$\delta_6 = d(\beta_6)$	$\gamma_7$	$[\gamma_3, \gamma_5]$
#vertices:	4	6	7	8	9
#edges:	6	10	12	14	16
#graphs:	1	2	4	46	68
#or.graphs in $Q(P) = \text{Or}\vec{\gamma}(P, \dots, P)$ :	3	167	1,500	37,185	?
#or.graphs in $\llbracket P, Q(P) \rrbracket$ :	39	3,495	35,949	1,003,611	?
#Leibniz graphs in $\diamond(P, \llbracket P, P \rrbracket)$ :	27	3,876	45,965	?	?
#skew Leibniz graphs in $\diamond(P, \llbracket P, P \rrbracket)$ :	8	843	9,556	293,654	?

In dimension two the cocycle condition for Poisson 2-cocycles is satisfied for every bivector, but the condition to be a 2-coboundary is generally still nontrivial. Nevertheless, all the universal flows which we have tried are trivial.

**Proposition 28** (Cells 17, 31, 45, 67 in Chapter 6). *In 2-dimensional Poisson geometries, the Poisson cocycles  $Q_\gamma(P)$  defined by graph cocycles  $\gamma \in \{\gamma_3, \gamma_5, \delta_6, \gamma_7\}$  are Poisson-trivial. Namely, there exist vector fields  $X_\gamma(P)$ , differential polynomial in  $P$ , that trivialize the flows  $Q_\gamma(P) = \llbracket P, X_\gamma(P) \rrbracket$ . Moreover, with respect to the standard symplectic structure on  $\mathbb{R}^2$ , every such vector field  $X_\gamma(P)$  is the Hamiltonian vector field of a Hamiltonian  $H_\gamma(P)$ , again differential polynomial in  $P$ .*

The case of  $\gamma_3$  was known to Kontsevich (1996), and the respective Hamiltonian was found by our colleague Bouisaghouane (2016/17). The remaining cases are established by the new software.

**Example 29.** Letting  $P = u \partial_x \wedge \partial_y$  be the generic Poisson bi-vector on  $\mathbb{R}^2$ , we have  $H_{\gamma_3} = 8u_y^2 u_{xx} - 16u_x u_y u_{xy} + 8u_x^2 u_{yy}$ ,  $H_{\gamma_5} = 6u_y^2 u_{xx} u_{xy}^2 - 12u_x u_y u_{xy}^3 - 6u_y^2 u_{xx}^2 u_{yy} + 12u_x u_y u_{xx} u_{xy} u_{yy} + 6u_x^2 u_{xy} u_{yy} - 6u_x^2 u_{xx} u_{yy}^2 - 2u_y^3 u_{xy} u_{xxx} + 2u_x u_y^2 u_{yy} u_{xxx} + 2u_y^3 u_{xx} u_{xxy} + 2u_x u_y^2 u_{xy} u_{xxy} - 4u_x^2 u_y u_{yy} u_{xxy} - 4u_x u_y^2 u_{xx} u_{xxy} + 2u_x^2 u_y u_{xy} u_{xyy} + 2u_x^3 u_{yy} u_{xyy} + 2u_x^2 u_y u_{xx} u_{yyy} - 2u_x^3 u_{xy} u_{yyy} - 2u_y^4 u_{xxxx} + 8u_x u_y^3 u_{xxy} - 12u_x^2 u_y^2 u_{xxyy} + 8u_x^3 u_y u_{xyyy} - 2u_x^4 u_{yyyy}$ , and  $H_{\gamma_7} = \frac{199}{4} u_y^2 u_{xx} u_{xy}^4 - \frac{199}{2} u_x u_y u_{xy}^5 - \dots$ . The full formula for  $H_{\gamma_7}$  is given in cell 68 on p. 163 in Chapter 6.

**Remark 30.** In [6, Appendix F, Remark 13] we establish that the formula for the trivializing vector field  $X_{\gamma_3}$  can be realized as a sum of Kontsevich graphs (although this representation generally does not provide a trivializing vector field in dimensions greater than two). Moreover it was established by Bouisaghouane (2016) that the Hamiltonian  $H_{\gamma_3}$  itself is a sum of Kontsevich graphs.

The fact that the Hamiltonian associated with  $\gamma_3$  can be represented as a sum of Kontsevich graphs is not an isolated incident:

**Proposition 31** (Cells 23, 37, 54, 72 in Chapter 6). *For  $\gamma \in \{\gamma_3, \gamma_5, \delta_6, \gamma_7\}$  and an arbitrary Poisson structure  $P$  on  $\mathbb{R}^2$ , the differential polynomial Hamiltonians  $H_\gamma(P)$  from Proposition 28 are sums of Kontsevich graphs; their shapes are on pp. 154, 157, 161, 164 in Chapter 6. The respective trivializing vector fields are obtained from these Hamiltonians by using the standard symplectic structure  $\omega = dx \wedge dy$  on  $\mathbb{R}^2$ .*

Next we consider the rescaled Nambu–Poisson structures in three and four dimensions.

**Proposition 32** (see Proposition 18 in [10] and Proposition 1, Corollary 2, Example 5, and Theorem 7 in [11]). *The tetrahedral flow  $\dot{P} = \text{O}\vec{\Gamma}(\gamma_3)(P^{\otimes 4})$  restricts to the class of rescaled Nambu–Poisson brackets  $P = \rho da/(dx dy dz) = \llbracket \rho \partial_x \wedge \partial_y \wedge \partial_z, a \rrbracket$  on  $\mathbb{R}^3$  with coordinates  $x, y, z$ , that is there exist  $(\dot{\rho}, \dot{a})$  such that  $Q_{\gamma_3}(P[\rho, a]) = P[\dot{\rho}, a] + P[\rho, \dot{a}]$ .*

- The velocity  $\dot{a}$  is given by Kontsevich’s graphs:  $\dot{a} = Q_{\gamma_3}(P, P, P, a)$ .
- The velocity  $\dot{\rho}$  is expressed by  $\dot{\rho} = (Q_{\gamma_3}(P[\rho, a]) - P[\rho, \dot{a}])/P[1, a]$ .
- The velocities  $\dot{\rho}, \dot{a}$  are obtained by total skew-symmetrization.<sup>14</sup>

$$\begin{aligned} \dot{a} &= \sum_{\sigma, \tau, \zeta \in S_3} (-)^\sigma (-)^\tau (-)^\zeta (2a_{u_1} a_{u_2} a_{u_3} \rho_{w_1} \rho_{w_2} \rho_{w_3} a_{v_1} v_2 v_3 - 6\rho a_{u_1} v_2 a_{u_2} a_{u_3} \rho_{w_1} \rho_{w_2} \rho_{w_3} a_{v_1} v_3 w_2 \\ &\quad - 6\rho^2 a_{u_1} a_{u_2} a_{u_3} a_{v_1} v_2 \rho_{w_3} a_{v_3} w_1 w_2) \\ \dot{\rho} &= \sum_{\sigma, \tau, \zeta \in S_3} (-)^\sigma (-)^\tau (-)^\zeta (-2a_{u_1} a_{u_2} a_{u_3} \rho_{v_1} \rho_{v_2} \rho_{v_3} \rho_{w_1} w_2 w_3 + 6a_{u_1} v_2 a_{u_2} a_{u_3} \rho_{v_1} \rho_{v_3} \rho_{w_2} \rho_{w_1} w_3 \\ &\quad - 12\rho a_{u_1} a_{u_2} a_{u_3} a_{v_1} v_2 \rho_{v_3} \rho_{w_1} \rho_{w_2} w_3 - 6\rho a_{u_1} v_2 a_{u_2} a_{u_3} \rho_{v_1} \rho_{v_3} \rho_{w_1} w_2 w_3 + 6\rho^2 a_{u_1} a_{u_2} a_{u_3} a_{v_1} v_2 \rho_{v_3} \rho_{w_1} w_2 w_3), \end{aligned}$$

where each sum runs over three permutations  $\sigma, \tau, \zeta \in S_3$ , giving three triples  $(u_1, v_1, w_1) = (\sigma(x), \sigma(y), \sigma(z))$ ,  $(u_2, v_2, w_2) = (\tau(x), \tau(y), \tau(z))$ , and  $(u_3, v_3, w_3) = (\zeta(x), \zeta(y), \zeta(z))$ .

- The cocycle  $Q_{\gamma_3}(P[\rho, a])$  is the coboundary of a vector field  $X[\rho, a]$  with differential polynomial coefficients (cubic in both  $\rho$  and  $a$ , of total differential order eight).
- The vector field  $X[\rho, a]$  is again realized—by using Civita symbols—as the total skew-symmetrization of tiny differential polynomial expressions, themselves encoded by graphs.
- The induced velocities are such that  $\dot{a} = -\llbracket X, a \rrbracket$  and  $\dot{\rho} \partial_x \wedge \partial_y \wedge \partial_z = -\llbracket X, \rho \partial_x \wedge \partial_y \wedge \partial_z \rrbracket$ .

This is demonstrated in detail in §7.1, and established in [11].

**Proposition 33** (see Example 6 in [11]). *The tetrahedral flow restricts to the class of rescaled Nambu–Poisson brackets  $P = \rho da db/(dx dy dz dw) = \llbracket \llbracket \rho \partial_x \wedge \partial_y \wedge \partial_z \wedge \partial_w, a \rrbracket, b \rrbracket$  on  $\mathbb{R}^4$  with coordinates  $x, y, z, w$ , that is there exist  $(\dot{\rho}, \dot{a}, \dot{b})$  such that  $Q_{\gamma_3}(P[\rho, a, b]) = P[\dot{\rho}, a, b] + P[\rho, \dot{a}, b] + P[\rho, a, \dot{b}]$ .*

- The velocities  $\dot{a}, \dot{b}$  are given by Kontsevich graphs:  $\dot{a} = Q_{\gamma_3}(P, P, P, a)$  and  $\dot{b} = Q_{\gamma_3}(P, P, P, b)$ .
- The velocity  $\dot{\rho}$  is expressed by  $\dot{\rho} = (Q_{\gamma_3}(P[\rho, a, b]) - P[\rho, \dot{a}, b] - P[\rho, a, \dot{b}])/P[1, a, b]$ .
- The velocities  $\dot{\rho}, \dot{a}, \dot{b}$  are obtained by total skew-symmetrization.<sup>14</sup> in particular, we have that

$$\begin{aligned} \dot{a} &= \sum_{\sigma, \tau, \zeta \in S_4} (-)^\sigma (-)^\tau (-)^\zeta \\ &\quad + 3a_{s_1} u_2 u_3 a_{t_1} t_2 b_{s_2} b_{s_3} v_1 b_{t_3} u_1 a_{v_2} a_{v_3} \rho^3 + 6a_{s_1} u_2 a_{t_1} a_{t_2} v_3 a_{u_3} v_1 v_2 b_{t_3} u_1 b_{s_2} b_{s_3} \rho^3 + 3a_{v_2} a_{t_1} u_2 v_3 b_{v_1} \rho_{s_1} a_{u_1} a_{u_3} b_{s_2} t_3 b_{s_3} t_2 \rho^2 \\ &\quad - 6a_{s_1} v_3 b_{s_2} t_1 \rho_{v_1} a_{s_3} u_1 v_2 a_{u_2} a_{u_3} b_{t_2} b_{t_3} \rho^2 - 6a_{s_1} v_2 v_3 a_{t_1} a_{u_2} a_{u_3} v_1 b_{s_2} b_{s_3} u_1 b_{t_3} \rho_{t_2} \rho^2 + 6a_{s_1} a_{s_2} v_3 a_{s_3} u_1 a_{t_1} t_2 t_3 b_{v_1} \rho_{v_2} b_{u_2} b_{u_3} \rho^2 \\ &\quad - 6a_{s_1} a_{t_1} u_2 u_3 b_{u_1} v_2 a_{t_2} a_{t_3} b_{s_2} b_{s_3} \rho_{v_1} \rho_{v_3} \rho + 6a_{v_2} a_{s_1} a_{s_2} s_3 t_1 a_{t_2} t_3 \rho_{v_1} \rho_{v_3} b_{u_1} b_{u_2} b_{u_3} \rho - 2a_{s_1} a_{t_2} a_{t_3} u_1 u_2 a_{u_3} b_{t_1} b_{s_2} b_{s_3} \rho_{v_1} \rho_{v_2} \rho_{v_3}, \end{aligned}$$

where  $(s_1, t_1, u_1, v_1) = (\sigma(x), \sigma(y), \sigma(z), \sigma(w))$ ,  $(s_2, t_2, u_2, v_2) = (\tau(x), \tau(y), \tau(z), \tau(w))$ , and  $(s_3, t_3, u_3, v_3) = (\zeta(x), \zeta(y), \zeta(z), \zeta(w))$ . The other formulas, for  $\dot{b}$  and  $\dot{\rho}$ , are on p. 185 in Chapter 7.

This is discussed in §7.3; the case with scale  $\rho \equiv 1$  is discussed first in §7.2.

**Proposition 34** (see Example 7 in [11]). *The pentagon-wheel flow  $\dot{P} = \text{O}\vec{\Gamma}(\gamma_5)(P^{\otimes 6})$  restricts to the class of rescaled Nambu–Poisson brackets on  $\mathbb{R}^3$ , with the same mechanism  $\dot{a} = \text{O}\vec{\Gamma}(\gamma_5)(P, P, P, P, P, a)$  and the same subtract-and-divide mechanism for  $\dot{\rho}$  as above; again  $\dot{\rho}$  and  $\dot{a}$  are obtained by total skew-symmetrization using 5 permutations of the independent coordinates  $x, y, z$  on  $\mathbb{R}^3$ .*

<sup>14</sup>Their expressions as total skew-symmetrizations were obtained in collaboration with D. Lipper (2020).

**Remark 35.** For the tetrahedral  $\gamma_3$  flow on (the space of) Nambu–Poisson brackets on  $\mathbb{R}^3$ , the above totally skew-symmetric expressions for  $\dot{\rho}$  and  $\dot{a}$  enjoy a further hypersymmetry property. Namely, choosing *any* differential monomial (with a nonzero coefficient) in  $\dot{\rho}$  (resp.  $\dot{a}$ ), and choosing *any* way to skew-symmetrize it<sup>15</sup> still not producing zero identically, the skew-symmetrization then reproduces the entire homogeneous component in which the respective monomial is contained. There are three homogeneous components in  $\dot{a}$ , and five in  $\dot{\rho}$ .

Finally we consider the class of  $R$ -matrix Poisson brackets. With a Lie algebra  $\mathfrak{g}$  equipped with a non-degenerate bilinear form (such as  $A, B \mapsto \text{tr}(AB)$  on some matrix Lie algebras), we associate  $R$ -matrices (e.g., from a direct sum decomposition of  $\mathfrak{g}$  one obtains a difference of projections), as in Chapter 10 of the book by Gengoux–Pichereau–Vanhaecke (2013). These ingredients then allow us to cook homogeneous linear, quadratic, and cubic Poisson brackets. We consider the tetrahedral  $\gamma_3$  flow for these  $R$ -matrix Poisson brackets.

**Proposition 36** (Cells 22–27, 31–35 and the rest in Chapter 8). *The graph cocycle  $\gamma_3$  acts on homogeneous quadratic and cubic Poisson structures associated with  $R$ -matrices.*

- On  $\mathfrak{gl}_2(\mathbb{R})$  with coordinates  $x, y, z, v$  there is an  $R$ -matrix  $\begin{pmatrix} x & y \\ z & v \end{pmatrix} \mapsto \begin{pmatrix} 0 & y \\ -z & 0 \end{pmatrix}$ . The cubic polynomial Poisson structure associated with this  $R$ -matrix is  $P = (x^2y + y^2z)\partial_x \wedge \partial_y + (x^2z + yz^2)\partial_x \wedge \partial_z + (2xyz + 2yzv)\partial_x \wedge \partial_v + (y^2z + yv^2)\partial_y \wedge \partial_v + (yz^2 + zv^2)\partial_z \wedge \partial_v$ . The tetrahedral flow of this Poisson structure is  $\dot{P} = Q_{\gamma_3}(P) = (-48x^5y - 288x^3y^2z - 240xy^3z^2 + 192y^3z^2v - 384xy^2zv^2 - 192y^2zv^3)\partial_x \wedge \partial_y + (-48x^5z - 288x^3yz^2 - 240xy^2z^3 + 192y^2z^3v - 384xyz^2v^2 - 192yz^2v^3)\partial_x \wedge \partial_z + (-336x^4yz - 480x^2y^2z^2 - 576x^3yzv + 480y^2z^2v^2 + 576xyzv^3 + 336yzv^4)\partial_x \wedge \partial_v + (192x^3y^2z - 192xy^3z^2 + 288y^2zv^3 + 48yv^5 + 48(8x^2y^2z + 5y^3z^2)v)\partial_y \wedge \partial_v + (192x^3yz^2 - 192xy^2z^3 + 288yz^2v^3 + 48zv^5 + 48(8x^2yz^2 + 5y^2z^3)v)\partial_z \wedge \partial_v$ . We detect that this bi-vector is a coboundary,  $Q_{\gamma_3}(P) = \llbracket \vec{Y}, P \rrbracket$  with the vector  $\vec{Y} = (-24x^4 + 120y^2z^2 - 96yzv^2)\partial_x + (96x^3y - 96yv^3)\partial_y + (96x^3z - 96zv^3)\partial_z + (96x^2yz - 120y^2z^2 + 24v^4)\partial_v$ .
- Similarly also on  $\mathfrak{gl}_2(\mathbb{R})$  there is the  $R$ -matrix  $\begin{pmatrix} x & y \\ z & v \end{pmatrix} \mapsto \begin{pmatrix} x & y \\ -z & v \end{pmatrix}$  and the cubic polynomial Poisson structure associated with it; its tetrahedral flow is also Poisson-trivial.<sup>16</sup>
- On  $\mathfrak{gl}_3(\mathbb{R})$  there are quadratic and cubic Poisson brackets associated with the  $R$ -matrices  $\begin{pmatrix} x_0 & x_1 & x_2 \\ x_3 & x_4 & x_5 \\ x_6 & x_7 & x_8 \end{pmatrix} \mapsto \begin{pmatrix} 0 & x_1 & x_2 \\ -x_3 & 0 & x_5 \\ -x_6 & -x_7 & 0 \end{pmatrix}$  and  $\begin{pmatrix} x_0 & x_1 & x_2 \\ x_3 & x_4 & x_5 \\ x_6 & x_7 & x_8 \end{pmatrix} \mapsto \begin{pmatrix} x_0 & x_1 & x_2 \\ -x_3 & x_4 & x_5 \\ -x_6 & -x_7 & x_8 \end{pmatrix}$ . In those cases the tetrahedral flow is also Poisson-trivial.<sup>16</sup>

Staying in the same context of homogeneous Poisson brackets, we provide a construction of universal Poisson 1-cocycles in [10]; this is Proposition 48 on p. 35 below. We consider two examples: in the above cases of  $R$ -matrices associated with the splittings of  $\mathfrak{gl}_2(\mathbb{R})$ , the 1-cocycles are identically zero.

The list of classes of Poisson brackets discussed so far is not exhaustive. There are more methods for constructing Poisson brackets. One such further method is available from Gengoux–Pichereau–Vanhaecke: they construct symplectic brackets with polynomial coefficients. Those Poisson structures can also be tried by using our algorithms and software. Another approach to the (non)triviality of Kontsevich’s universal flows in the context of algebraic varieties is studied by Dolgushev–Rogers–Willwacher (2015).

<sup>15</sup>That is, prescribing which of the letters  $xxxxxyyzzz$  in that monomial belong to which of the three triples  $xyz$ .

<sup>16</sup>The full formulas are given in Cells 31–35, 41–43, 44–48, 49–51, 54–56 in Chapter 8.



**Remark 37.** We have studied infinitesimal deformations  $P \mapsto P + \varepsilon Q + \bar{o}(\varepsilon)$  defined via graph cocycles. These infinitesimal deformations can be formally integrated; higher order terms in the series  $P + \varepsilon Q + \frac{\varepsilon^2}{2}R + \dots$  are obtained recursively by inserting the graphs from  $Q$  into vertices of themselves. The study of convergence of the series  $P(\varepsilon)$  is a different aspect, which we do not consider in this dissertation.

*Graphs.* Just as the Kontsevich  $\star$ -product is a byproduct of the respective Formality  $L_\infty$ -morphism  $\mathcal{F}: T_{\text{poly}}(\mathbb{R}^d) \rightarrow D_{\text{poly}}(\mathbb{R}^d)$ , the universal flows on the spaces of Poisson structures are a byproduct of universal  $L_\infty$ -automorphisms  $T_{\text{poly}}(\mathbb{R}^d) \rightarrow T_{\text{poly}}(\mathbb{R}^d)$ , which themselves are defined via graph cocycles in the Kontsevich graph complex. We presently recall the construction of this cochain complex, in which the elements are sums of graphs (with extra structure, namely an ordering of edges) modulo relations. As the next step, we will (re)compute the dimensions of graded parts of the respective graph cohomology, and we find (new) explicit representatives of cohomology classes, which are necessary to evaluate the flows on spaces of Poisson structures.

**Notation 38.** For  $n \in \mathbb{N}_{\geq 1}$  let  $\text{Gra}(n)$  denote the  $\mathbb{N}_{\geq 0}$ -graded vector space with the  $k$ th component spanned by simple<sup>17</sup> undirected graphs on the vertex set  $\{0, \dots, n-1\}$  with an ordered set of  $k$  edges labeled<sup>18</sup> from 0 to  $k-1$ , modulo the relations  $\gamma^\sigma = (-)^\sigma \gamma$ , where the graphs  $\gamma$  and  $\gamma^\sigma$  differ only by the permutation  $\sigma$  of edge labels. A graph with an automorphism that induces an odd permutation on edges is called a *zero graph*.

**Definition 39.** The *insertion*  $\gamma_1 \vec{\sigma}_i \gamma_2$  of a graph  $\gamma_1$  on  $n_1$  vertices into the  $i$ th vertex of another graph  $\gamma_2$  on  $n_2$  vertices is a sum of graphs on  $n_1 + n_2 - 1$  vertices. Each graph in the sum consists of the graph  $\gamma_2$  with its  $i$ th vertex replaced<sup>19</sup> by the entire graph  $\gamma_1$ ; the edges which were incident to the  $i$ th vertex in  $\gamma_2$  are re-attached to the vertices of  $\gamma_1$  in all possible ways (each possible way to re-attach edges provides one term in the sum of graphs).

The operation  $\vec{\sigma}_i$  is extended to linear combinations of graphs in  $\text{Gra}(n)$  by bilinearity. The *insertion*  $\vec{\sigma}: \text{Gra}(n) \otimes \text{Gra}(m) \rightarrow \text{Gra}(n+m-1)$  is defined for graphs by the sum  $\gamma_1 \vec{\sigma} \gamma_2 = \sum_{i=0}^{m-1} \gamma_1 \vec{\sigma}_i \gamma_2$  of insertions into all vertices of  $\gamma_2$ , and extended to linear combinations of graphs by bilinearity.

With the partial composition operations  $\vec{\sigma}_i$  and the action of the group  $S_n$  that permutes the labels of vertices, the collection of graded vector spaces  $\text{Gra}(n)$  forms an operad.

**Definition 40.** The *full Kontsevich graph complex* fGC is the collection of graded<sup>20</sup> vector spaces  $\text{fGC}(n)$  defined as the quotient of  $\text{Gra}(n)$ : namely, graphs that differ only by their vertex labeling are identified. That collection of vector spaces is equipped with the vertex-expanding differential  $d$  defined by  $d = [\bullet\bullet, -]$ , where the Lie bracket is defined for graphs  $\gamma_1$  on  $e_1$  edges and  $\gamma_2$  on  $e_2$  edges by the commutator of insertions  $[\gamma_1, \gamma_2] = \gamma_1 \vec{\sigma} \gamma_2 - (-)^{e_1 e_2} \gamma_2 \vec{\sigma} \gamma_1$ , and the Lie bracket is extended to fGC by bilinearity.

<sup>17</sup>Without double edges and without tadpoles, not necessarily (strongly) connected, not necessarily with each vertex at least trivalent.

<sup>18</sup>In the papers [4, 11, 8] the edges are labeled by using roman numbers  $I, II, \dots$  so that the ordered set of edges  $E(\gamma)$  is  $I \wedge II \wedge \dots$

<sup>19</sup>The labels of the vertices of  $\gamma_1$  are shifted up by  $i$ , and the labels of the vertices  $> i$  in  $\gamma_2$  are shifted up by  $n_1 - 1$ . The labels of the edges of  $\gamma_1$  are shifted up by the number of edges in  $\gamma_2$ .

<sup>20</sup>The grading initially inherited from Notation 38 can be shifted, as seen in the literature.

See also [4] for an elementary introduction to the graph complex. Detailed proofs of the defining properties of this graph complex are found in Rutten–Kiselev (2018).

**Example 41.** Three examples of graph cocycles are  $\gamma_3 = \triangle$ ,

$$\gamma_5 = \text{pentagon-wheel} + \frac{5}{2} \text{rectangle} \quad \text{and} \quad \delta_6 = d(\beta_6) \quad \text{where} \quad \beta_6 = \text{square}.$$

The tetrahedral cocycle  $\gamma_3$  was found by Kontsevich (1996); the pentagon-wheel cocycle  $\gamma_5$  was known to Kontsevich and to Willwacher; the coboundary  $\delta_6$  is an example of a trivial cocycle. Further examples and pictures of graph cocycles are found in Chapter 5.

**Proposition 42** (see Chapter 5). *The new software is able to find explicit representatives of the non-trivial graph cocycles  $\gamma_3, \gamma_5, \gamma_7$ , and calculate commutators of graphs, in particular  $[\gamma_3, \gamma_5] \in \ker d$ , in the Kontsevich graph complex with the vertex-expanding differential. The encodings of these four graph cocycles are given in Appendix E.*

More details about and various calculations in the Kontsevich graph complex are found in Chapter 5. There are variants (in fact, subcomplexes) of the full graph complex spanned e.g. by connected graphs in which each vertex has degree at least three. The full graph complex is a symmetric product of the subcomplex spanned by connected graphs, and Willwacher (2010) proved that the cohomology of the connected graph complex is expressed as the direct sum of the cohomology of the degree-restricted graph complex and some known classes, namely  $(4n + 1)$ -gons for  $n \geq 1$  (with  $4n + 1$  two-valent vertices and  $4n + 1$  edges).

**Proposition 43.** *The new software is capable of finding the dimensions of the graded parts of the (connected) graph cohomology spaces. In Table 0.5 we give a precise count, up to 9 vertices; the meaning of  $N_\delta, N_{\ker}, N_0, N_{\text{im}}$  is explained in Chapter 13.*

**Table 0.5:** Dimensions of connected graph spaces and cohomology groups.

$n$	$\#E$	$\#(\text{graphs})$	$\#(= 0)$	$\#(\neq 0), N_\delta$	$N_{\ker}, N_0$	$N_{\text{im}}$	$\dim H^*(n)$
4	6	1	0	1	1	–	1
3	5	0	–	–	–	–	–
5	8	2	2	0	–	–	0
4	7	0	–	–	–	–	–
6	10	14	8	6	1	–	1
5	9	1	1	–	0	–	–
7	12	126	78	48	1	–	0
6	11	9	8	–	1	0	1
8	14	1579	605	974	36	–	1
7	13	95	60	–	35	0	35
9	16	26631	7557	19074	883	–	1
8	15	1515	602	–	913	31	882



The analogous counts for graph cohomology spanned by connected graphs having at least trivalent vertices are given in [4, Table 3]. The same numbers of nonzero graphs were previously calculated by Willwacher–Živković (2014); their methods partly depended on floating-point arithmetic. We provide explicit representatives of graph cohomology classes in Chapter 5.

We now recall the origin of the graph orientation morphism  $\gamma \mapsto \text{Or}\vec{(\gamma)}$ , which maps graph cocycles  $\gamma$  on  $n$  vertices and  $2n - 2$  edges to infinitesimal symmetries  $\dot{P} = \text{Or}\vec{(\gamma)}(P^{\otimes n})$  of Poisson bi-vectors  $P$  on affine manifolds. The reasoning in [8] is based on that of Jost (2013), which in turn follows an outline by Willwacher (2010), itself referring to the seminal paper by Kontsevich (1996). A combinatorial interpretation of the orientation morphism is in [9]. Examples of graph cocycles suitable as input to the orientation morphism can be borrowed from [4]. An extension of the above technique, that now yields universal 1-cocycles in the case of homogeneous Poisson bi-vectors, is contained in [10].

The inspiration for the orientation morphism comes from a very precise analogy between two worlds: that of graphs on the one hand, and that of endomorphisms on the spaces of multi-vectors on affine manifolds on the other hand. In fact, graphs describe endomorphisms defined by natural formulas. The world of endomorphisms on the spaces of multi-vectors is recalled in [8, Section 1]. The first natural example of such an endomorphism is the Schouten bracket  $\pi_S = \pm[\cdot, \cdot]$ . Besides, endomorphisms can be inserted one into the other, so that there is the Nijenhuis–Richardson bracket  $[\cdot, \cdot]_{\text{NR}}$  which is the commutator of insertions, and there is the differential  $[\pi_S, -]_{\text{NR}}$  which is the bracket with the Maurer–Cartan element  $\pi_S$ . Here is the dictionary that we explore: see Table 0.6.

**Table 0.6:** From graphs to endomorphisms: the respective objects or structures.

World of graphs	World of endomorphisms
Graphs $(\gamma, E(\gamma))$	Endomorphisms
Insertion $\vec{\sigma}_i$ of graph into $i^{\text{th}}$ vertex	Insertion of endomorphism into $i^{\text{th}}$ argument
Insertion $\vec{\sigma}$ of graph into graph	Insertion $\vec{\sigma}$
Bracket $[a, b] = a \vec{\sigma} b - (-)^{ E(a)  \cdot  E(b) } b \vec{\sigma} a$	Bracket $[a, b] = a \vec{\sigma} b - (-)^{ a  \cdot  b } b \vec{\sigma} a$
Lie bracket $([a, b], E([a, b]) := E(a) \wedge E(b))$	Nijenhuis–Richardson bracket $[a, b]_{\text{NR}}$ on the space of skew endomorphisms
The stick $\bullet\!\!\!\bullet$	The Schouten bracket $\pi_S = \pm[\cdot, \cdot]$
Master equation $[\bullet\!\!\!\bullet, \bullet\!\!\!\bullet] = 0$	Master equation $[\pi_S, \pi_S]_{\text{NR}} = 0$
Graded Jacobi identity for $[\cdot, \cdot]$	Graded Jacobi identity for $[\cdot, \cdot]_{\text{NR}}$
Differential $d = [\bullet\!\!\!\bullet, \cdot]$	Differential $\partial = [\pi_S, \cdot]_{\text{NR}}$

We analyze this correspondence in more detail in [8]. By using this dictionary, it becomes easy to provide the canonical factorizing operator  $\diamond$  for the Poisson cocycle condition  $\llbracket P, \text{Or}\vec{(\gamma)}(P^{\otimes n}) \rrbracket = \diamond(P, \llbracket P, P \rrbracket)$ . This is a corollary of the following result:

**Proposition 44** (Proposition 2 in [8]). *The mapping  $\text{Or}\vec{(\cdot)}: \bigoplus_n (\text{Gra}_{\# \text{Vert}=n \geq 1}^{\wedge_i \text{edge}_i})_{S_n} \rightarrow \text{End}_{\text{skew}}^{*,*}(T_{\text{poly}}(M)[1])$  is a Lie algebra morphism:  $\text{Or}\vec{([\beta, \gamma])} = [\text{Or}\vec{(\beta)}, \text{Or}\vec{(\gamma)}]_{\text{NR}}$ .*

In the case when  $\beta = \bullet\!\!\!\bullet$  and  $\gamma$  is a graph cocycle, this fact is a key to the factorization of the Poisson cocycle condition. Namely in that case the left-hand side  $\text{Or}\vec{(d(\gamma))}$  evaluates to zero and the right-hand side,  $[\pi_S, \text{Or}\vec{(\gamma)}]_{\text{NR}}$ , evaluated at  $n + 1$  copies of Poisson bi-vector  $P$ , is a linear combination of  $\llbracket P, \text{Or}\vec{(\gamma)}(P^{\otimes n}) \rrbracket$  and  $\text{Or}\vec{(\gamma)}(P^{\otimes(n-1)} \otimes \llbracket P, P \rrbracket)$ , where the former has a nonzero coefficient and the latter is a sum of Leibniz graphs which evaluates to zero at every Poisson bi-vector  $P$ . We refer again to Table 0.4 for statistics.

Proposition 44 is not the only mechanism which provides solutions  $\diamond$  to the factorization problem  $\llbracket P, \text{Or}(\gamma)(P^{\otimes n}) \rrbracket = \diamond(P, \llbracket P, P \rrbracket)$  of the Poisson cocycle condition. The above was the original construction by Kontsevich. We can consider more generally the problem of constructing the (minimal) set of Leibniz graphs needed to express  $\llbracket P, Q_\gamma \rrbracket$  as the expansion of a linear combination of Leibniz graphs from that set.

To build all the potentially needed Leibniz graphs, we take the Kontsevich graph expansion of the left-hand side  $\llbracket P, Q_\gamma \rrbracket$  of the Poisson cocycle condition, and find all the Leibniz graphs in whose expansion (again into Kontsevich graphs) we reproduce at least one previously known graph. Now having expanded these Leibniz graphs into Kontsevich graphs built of wedges, we have clearly reproduced the original set, but we can also obtain more (new) Kontsevich graphs — previously not contained in the Poisson cocycle condition's left-hand side, or after the previous iterations, now to start. Repeating this step iteratively, until saturation, we get a large set of Leibniz graphs, see [5, §1.2].

**Example 45.** For the pentagon-wheel cocycle  $\gamma_5$  and the flow  $\dot{P} = Q_{\gamma_5}(P)$ , an example of such saturation is reported in Table 0.7 (Table 1 in [5]).

**Table 0.7:** The number of skew Leibniz graphs produced iteratively for  $\llbracket P, Q_{\gamma_5}(P) \rrbracket$ .

No. iteration $i$	1	2	3	4	5	6	7	8
# of graphs	1518	14846	41031	54188	56318	56503	56509	56509
of them new	all	+13328	+26185	+13157	+2130	+185	+6	none

Lastly, we equate the Kontsevich graph expansions of all these Leibniz graphs, taken with undetermined coefficients, to the Poisson cocycle condition left-hand side. Solving the arising linear algebraic system upon the coefficients of Leibniz graphs, we discover multiple solutions, naturally including Kontsevich's canonical solution. The properties of the canonical solution are listed in Table 0.4. Let  $L_3^n$  denote the set of Leibniz graphs with positive differential order over 3 ground vertices and  $n$  aerial vertices (of which  $n - 1$  are wedges and one is a tripod). The non-uniqueness of Leibniz graph factorizations is expressed in the following proposition.

**Proposition 46** (see Section 3.5.2). *The solution of a Leibniz graph factorization problem is not unique as soon as the number of aerial vertices in the Leibniz graphs exceeds 2; then there exist sums of Leibniz graphs such that their expansion into sums of Kontsevich graphs equals zero identically. The nullity of the Leibniz graph expansion map, restricted to the bi-gradings  $(3, n - 1)$  for  $n = 2, \dots, 5$  is reported in Table 0.8.*

**Table 0.8:** The nullity of the Leibniz graph expansion map restricted to  $L_3^n$

$n$	1	2	3	4
#Leibniz graphs in $L_3^n$	1	15	301	6741
Nullity of Leibniz graph expansion map restricted to $L_3^n$	0	0	12	538

**Remark 47.** Viewing Proposition 44 in the case where  $\gamma$  is a  $d$ -coboundary, there is a trivializing vector field for  $\text{Or}(\gamma)(P^{\otimes n})$  defined by graphs; see [8, Corollary 4]. We illustrate this in Chapter 6 with the coboundary  $\delta_6 = d(\beta_6)$ .

Let us now look at Proposition 44 from another perspective. Let both  $\beta$  and  $\gamma$  be graph cocycles. Then on the one hand one can commute the graph cocycles, and on the other hand one can commute the respective flows. The relation in Proposition 44 can be verified using the new software. The graph commutator  $[\gamma_3, \gamma_5]$  offers the minimal nontrivial illustration (cf. Section 6.5).

Another use for Proposition 44 is found by evaluating the endomorphism  $\text{Or}(\gamma)$  at tuples different from  $P^{\otimes n}$ :

**Proposition 48** (Theorem 4 in [10]). *Let  $(M, P)$  be an affine finite-dimensional real Poisson manifold with  $P = \llbracket \vec{V}, P \rrbracket$  homogeneous. Let  $\gamma = \sum_a c_a \cdot \gamma_a$  be a graph cocycle consisting of unoriented graphs  $\gamma_a$  over  $n$  vertices and  $2n-2$  edges (with a fixed ordering of edges in each  $\gamma_a$ ). Then the 1-vector  $\vec{X}(\gamma, \vec{V}, P) = \text{Or}(\gamma)(\vec{V} \otimes P^{\otimes n-1})$ , which is obtained by representing each edge  $i-j$  with the operator<sup>21</sup>  $\vec{\Delta}_{ij}$  and by (graded-)symmetrizing over all the ways  $\sigma \in \mathbb{S}_n$  to send the  $n$ -tuple  $\vec{V} \otimes P^{\otimes n-1}$  into the  $n$  vertices in each  $\gamma_a$ , is a Poisson cocycle:  $\vec{X} \in \ker \llbracket P, \cdot \rrbracket$ . The vector field  $\vec{X}$  is defined up to adding arbitrary Poisson 1-cocycles  $\vec{Z} \in \ker \llbracket P, \cdot \rrbracket$ .*

As said, Proposition 48 is illustrated by using two examples of Poisson structures obtained from  $R$ -matrices, see Proposition 36.

**Conclusion.** *Star products.* The new software package `gcaops` developed in this dissertation allows independent verification of results obtained by other software (within different implementations, written in different languages). The software illustrates the theory and verifies theoretical predictions. In particular, we know the cyclic weight relations for *all* Kontsevich- and Leibniz graphs up to order 5 in  $\hbar$ , and we know these relations specifically for those Kontsevich graphs on  $n = 7$  aerial vertices with their in-degrees bounded by  $\leq 1$ , that is for the affine Kontsevich graphs in the affine star product  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$ . We verified that the values of weights obtained by Banks–Panzer–Pym satisfy all the relations found by our software (e.g. the cyclic weight relations for Kontsevich graphs at  $\hbar^5$ , and the relations from the associativity of  $\star$ -product at  $\hbar^6$ ).

- To advance in the determination of closed-form expressions for the weights, we need to find more relations between them (in the case of linear relations, we must increase the rank). Kontsevich claims that the weights have many links to number theory: interesting numbers are hidden in the weights of graphs. We expect that there are probably also many more hidden relations between the weights. In this context, it is natural to wonder if the question of the (ir)rationality of  $\zeta(3)/\pi^3$  can be resolved in this way, because by knowing star products' high order expansions, we obtain (known and possibly new) algebraic relations between the Riemann zeta values (with  $\zeta(3)$  in particular).
- In another direction (as Gutt *et al.* (2008), for  $\star$ -product modulo  $\bar{o}(\hbar^3)$ ), for universal star products on manifolds with a torsion-free linear connection (the coefficients of the star product depending in a differential polynomial way on not only the Poisson structure but also the curvature tensor), an explicit expansion up to orders higher than three—and its uniqueness—is an open problem. We propose to use a variation of Kontsevich's graphs, now with two colors of aerial vertices: some vertices containing copies of the Poisson bivector, and other vertices containing the curvature tensor — in such a way that the graph encodes a tensorial expression over the curved Poisson manifold. The `gcaops` software can be extended to handle different content of aerial vertices in the graphs.

<sup>21</sup>These superoperators  $\vec{\Delta}_{ij}$ , acting from left to right, were introduced by Kontsevich in Ascona '96.

- In the proof of the associativity of the Kontsevich star product via the Formality morphism, we have verified the identity  $\text{Assoc}(\star(P))(f, g, h) = \diamond(P, \text{Jac}(P)(\cdot, \cdot, \cdot))(f, g, h)$  up to some recalculation constants (we give their values). Understanding all the conventions from which these recalculation constants originate will deepen our understanding of the work of Kontsevich’s Formality mechanism in deformation quantization.

*Poisson flows.* In this dissertation we extended the collection of explicit examples of flows and the respective factorizations, all realized in software. Our initial software implementation in [6] immediately detected the correct balance for the components of the tetrahedral flow which was invented in 1996. Moreover, we verified the mechanism of validity of the Poisson cocycle condition which was predicted by Kontsevich in ‘96. For all the flows under consideration we establish a factorization of the Poisson cocycle condition via the Jacobi identity exactly, taking into account all the normalizations and conventions.

- We found that the class of rescaled Nambu–Poisson structures  $P[\rho, a]$  is closed under the Kontsevich graph flows. We discover that the explicit infinitesimal evolution  $\dot{\rho}, \dot{a}$  of the functional parameters  $\rho, a$  is hypersymmetric (expressed as the total skew-symmetrization of differential polynomials, with arbitrarily chosen markers). We found the reason for the tetrahedral flow on rescaled 3D Nambu–Poisson structures to be Poisson cohomology trivial, by producing a graph realization of the trivializing vector field  $X[\rho, a]$ ; its coefficients are differential polynomials in the functional parameters. It remains to be seen how the evolution works for the five-wheel flow and for Nambu–Poisson structures in higher dimensions.
- Although the flows are designed to be not universally trivial, all the examples of flows evaluated at concrete Poisson structures in this dissertation are Poisson cohomology trivial, without exception. While we have verified the Poisson-triviality of the Kontsevich graph flows in 2-dimensional Poisson geometries (for  $\gamma_3, \gamma_5, \delta_6, \gamma_7$ ) and we have proved the Poisson-triviality of the tetrahedral flow for the class of 3D Nambu–Poisson brackets, it is still an open problem why it happens in other cases. This needs further exploration: in particular, how universal is the (non)triviality of Kontsevich’s graph construction with respect to the dimension of Poisson manifolds? (We refer to Chapter 6 of this dissertation where we obtain a graph realization of the Hamiltonians for the trivializing vector fields in dimension two.)

*Graph complex.* The Poisson flows discussed above originate from cocycles in the Kontsevich undirected graph complex with the vertex-expanding differential. In this domain we counted the dimensions of bi-graded parts, verifying the calculations by Willwacher–Živković. A natural open problem is to predict a universal formula for the coefficients of graph cocycles. In this direction Willwacher–Rossi (2014) give integral formulas. It would be interesting to illustrate these formulas by examples, and to find whether alternative formulas are possible.

*Programming.* On an average desktop computer, one can study the Kontsevich graph theory up to around 7 or 8 vertices, dealing with on the order of one million graphs. Namely, one can verify the associativity of the full star product up to order 6 (and up to orders 7 and 8 for the restricted case of the affine star product for affine Poisson structures), find explicit factorizations via Leibniz graphs, calculate relations between the weights of graphs, do gauge transformations of star products, insert a Poisson structure into a given flow, factor the Poisson cocycle condition via Leibniz graphs, and more.

The initial implementations `kontsevich_graph_series-cpp` and `graph_complex-cpp` (written in C++) were fast, but their design (as separate programs, communicating by

file input/output) was not very convenient for fast experimentation. Adding that level of interactivity to C++ programs would have been a lot of extra parsing work. The new software `gcaops` is based on Python and SageMath, which have interactivity built into their interpreters; for our purposes the Jupyter notebook interface is very convenient, as it allows calling the interpreters and receiving not only plain text output but also graph pictures. High performance is achieved by calling into C/C++ libraries such as `nauty` for graphs, `libSingular` for polynomials, and whatever SageMath uses for linear algebra.

*Educational.* The software developed for this dissertation could be suitable as a whole or in part as a new package for SageMath. It has been designed to be consistent with the conventions of SageMath, it is documented with examples (and its formal description in Appendix F), and it is intended to be easy to use. Hence it can be used by students to learn the subjects of deformation quantization and the graph complex. We hope that by this we may renew the interest in the topic.

**Publications.** The results of this dissertation are contained in the following publications and one preprint. All the texts which are published are freely open access from arXiv.

- [1] R. Buring and A.V. Kiselev. On the Kontsevich  $\star$ -product associativity mechanism. *Physics of Particles and Nuclei Letters*, **14**(2), 403–407, 2017. (Preprint [arXiv:1602.09036](https://arxiv.org/abs/1602.09036) [q-alg] – 4 p.) IAEA 48098722
- [2] R. Buring and A.V. Kiselev. The expansion  $\star \bmod \bar{o}(\hbar^4)$  and computer-assisted proof schemes in the Kontsevich deformation quantization. *Experimental Math.* **31**(3) or (4), 54 p., 2022 (in press). (doi:10.1080/10586458.2019.168046) (Preprint [arXiv:1702.00681](https://arxiv.org/abs/1702.00681) [math.CO] – 77 p.) MR (pending), Zbl. (pending)
- [3] R. Buring and A.V. Kiselev. Formality morphism as the mechanism of  $\star$ -product associativity: how it works. *Collected works Inst. Math., Kyiv* **16**:1 Symmetry & Integrability of Equations of Mathematical Physics, 22–43, 2019. (Preprint [arXiv:1907.00639](https://arxiv.org/abs/1907.00639) [q-alg] – 16 p.) Zbl. 143853125
- [4] R. Buring, A. V. Kiselev, and N. J. Rutten. The heptagon-wheel cocycle in the Kontsevich graph complex. *J. Nonlin. Math. Phys.*, **24**: Suppl. 1 ‘Local & Nonlocal Symmetries in Mathematical Physics’, 157–173, 2017. (Preprint [arXiv:1710.00658](https://arxiv.org/abs/1710.00658) [math.CO] – 17 p.) MR3750843; Zbl. 142053084
- [5] R. Buring, A. V. Kiselev, and N. J. Rutten. Infinitesimal deformations of Poisson bi-vectors using the Kontsevich graph calculus. *J. Phys.: Conf. Ser.*, **965**: Proc. XXV Int.conf. ‘Integrable Systems & Quantum Symmetries’ (6–10 June 2017, CVUT Prague, Czech Republic), Paper 012010, 2018. (Preprint [arXiv:1710.02405](https://arxiv.org/abs/1710.02405) [math.CO] – 12 p.)
- [6] A. Bouisaghouane, R. Buring, and A.V. Kiselev. The Kontsevich tetrahedral flow revisited. *J. Geom. Phys.*, **119**, 272–285, 2017. (Preprint [arXiv:1608.01710](https://arxiv.org/abs/1608.01710) [q-alg] – 29 p.) MR3661536; Zbl. 137153092
- [7] R. Buring, A. V. Kiselev, and N. J. Rutten. Poisson brackets symmetry from the pentagon-wheel cocycle in the graph complex. *Physics of Particles and Nuclei*, **49**(5): Supersymmetry and Quantum Symmetries 2017, 924–928, 2018. (Preprint [arXiv:1712.05259](https://arxiv.org/abs/1712.05259) [math-ph] – 4 p.) IAEA 52022014

- [8] R. Buring and A. V. Kiselev. The orientation morphism: from graph cocycles to deformations of Poisson structures. *J. Phys.: Conf. Ser.* **1194**: Proc. 32nd Int. colloquium on Group-theoretical methods in Physics: Group32 (9–13 July 2018, CVUT Prague, Czech Republic), Paper 012017, 2019. (Preprint [arXiv:1811.07878](#) [math.CO] – 10 p.)
- [9] A.V. Kiselev and R. Buring. The Kontsevich graph orientation morphism revisited. *Banach Center Publ.* **123** Homotopy algebras, deformation theory & quantization, 123–139, 2021. (Preprint [arXiv:1904.13293](#) [math.CO] – 18 p.) MR4276046; Zbl. 07350411
- [10] R. Buring and A.V. Kiselev. Universal cocycles and the graph complex action on homogeneous Poisson brackets by diffeomorphisms. *Physics of Particles and Nuclei Letters* **17**:5 Supersymmetry and Quantum Symmetries 2019, 707–713, 2020. (Preprint [arXiv:1912.12664](#) [math.SG] – 8 p.) IAEA (pending)
- [11] R. Buring, D. Lipper and A. V. Kiselev. The hidden symmetry of Kontsevich’s graph flows on the spaces of Nambu-determinant Poisson brackets. (*submitted*) (Preprint [arXiv:2112.03897](#) [math.SG] – 23+iv p.)



## Buring R.T. The action of Kontsevich’s graph complex on Poisson structures and star products: an implementation.

Dissertation submitted for the award of the title *Doctor of Natural Sciences* to the Faculty of Physics, Mathematics, and Computer Science at Johannes Gutenberg-University Mainz in Mainz, Germany.

**Abstract.** Whenever the associative pointwise product of scalar functions on an affine manifold is deformed by adding a bi-differential term proportional to a formal deformation parameter  $\hbar$ , the deformed product remains associative modulo  $\bar{o}(\hbar^2)$  if and only if the skew-symmetric part of the leading deformation term is a Poisson bracket. The problem of deformation quantization of Poisson manifolds is to extend such a first-order deformation to a formal power series in  $\hbar$ , preserving associativity, with a bi-differential operator at each order in  $\hbar$ . In 1997, M. Kontsevich proved the Formality Theorem and thus solved the problem of deformation quantization for all (affine) Poisson manifolds at once, by introducing graphs which represent universal differential polynomial expressions and integrals over configuration spaces of points in the Lobachevsky plane. If one wishes to work with Kontsevich’s  $\star$ -product and illustrate this theory explicitly, by expanding formal power series in  $\hbar$ , one runs into millions of graphs already at the order  $\hbar^7$ .

In a similar style, Kontsevich’s universal flows on the spaces of Poisson structures (1996), given by Poisson cocycles in the Lichnerowicz–Poisson cohomology, are built from graph cocycles in Kontsevich’s graph complex with the vertex-expanding differential. In this setting the graph cocycles with nine vertices already contain thousands of graphs. To operate with all the graphs and evaluate the universal formulas at particular Poisson structures, efficient software is needed.

We develop and present the software package `gcaops` (*Graph Complex Action on Poisson Structures*) for `SageMath`, to deal with both the  $\star$ -products and universal flows on spaces of Poisson structures. Using this package, we achieve the following: • we expand the entire Kontsevich  $\star$ -product, i.e. for generic Poisson structures and with harmonic weights, up to  $\bar{o}(\hbar^4)$ ; • we assemble  $\star \bmod \bar{o}(\hbar^6)$  from external data by Banks–Panzer–Pym, and we obtain the star product  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$  for affine Poisson brackets; • we illustrate the explicit proof of the associativity for the full star product modulo  $\bar{o}(\hbar^6)$  and for the affine star product modulo  $\bar{o}(\hbar^7)$ ; • we verify that the weights found by Banks–Panzer–Pym (2018) up to  $\bar{o}(\hbar^6)$  satisfy many known relations; • we find new explicit formulas of graph cocycles and universal Poisson cocycles, and • we prove the factorization of the Poisson cocycle condition via the Jacobi identity in each case.

Although Kontsevich’s universal Poisson flows associated with nonzero graph cohomology classes are designed to be not universally Poisson-trivial, we establish that the flows associated with the `grt`-related graph cocycles  $\gamma_i$  are Poisson trivial in these cases:  $\gamma_3, \gamma_5, \gamma_7$  in two-dimensional Poisson geometries,  $\gamma_3$  for 3D Nambu–Poisson structures, and  $\gamma_3$  for several (quadratic and cubic) homogeneous polynomial Poisson brackets associated with some  $R$ -matrices for Lie algebras  $\mathfrak{gl}_2(\mathbb{R})$  and  $\mathfrak{gl}_3(\mathbb{R})$ . Finally, we illustrate explicitly how the tetrahedron  $\gamma_3$  in the graph complex and the resulting flow on the spaces of Poisson structures act upon the Kontsevich  $\star$ -product, and we find out whether this action of the graph cocycle  $\gamma_3$  on Kontsevich’s  $\star$ -product is gauge trivial or not.

**Keywords.** *Poisson bracket, star-product, deformation quantization, Kontsevich graph complex, Poisson cohomology, differential graded Lie algebras, software design.*

**Mathematics Subject Classification (2020).** 17B63, 17B70, 18M30, 18M60, 53-04, 53-08, 53D17, 53D55, 68W30, 97-02.

# Part I

## Computer demonstrations

### Abstract

This part provides a tutorial for the `gcaops` (*Graph Complex Action on Poisson Structures*) software, using the SageMath notebook interface. This software grew out of the course *Deformation Quantization, Graph Complex, and Number Theory*, which was taught in the Dutch national Mastermath program in Fall 2020, with lectures by Arthemy Kiselev and exercise classes led by the author. Tutorials on November 5th and December 10th (both in 2020) were dedicated to the demonstration of this software. The present tutorial has been annotated and extended, with further examples added.

### Contents of Part I

0	Introduction .....	41
1	Implementation of star products .....	45
2	Implementation of Poisson structures .....	55
3	Implementation of Formality .....	65
4	Implementation of the graph complex .....	125
5	Examples of graph cocycles .....	137
6	Graph complex action on Poisson structures in dimension two .....	151
7	Graph complex action on rank two rescaled Nambu–Poisson structures .....	169
8	Graph complex action on $R$ -matrix Poisson structures .....	191
9	Graph complex action on star products .....	203



# Chapter 0

## Introduction

The `gcaops` (*Graph Complex Action on Poisson Structures*) software is a package written in Python 3, designed to be used with SageMath version 9.2 or later [15]. It is released under the MIT free software license. It is available from <https://github.com/rburing/gcaops>. An *Introduction to SageMath* is included as Appendix A in the dissertation.

### 0.1 Installation

This software can be obtained from <https://github.com/rburing/gcaops>. Up-to-date installation instructions are also listed there.

How to install the package:

1. Navigate to <https://github.com/rburing/gcaops> in a web browser; press the `Code` button and click the `Download ZIP` link.
2. Extract the ZIP file to a directory such as `/path/to/gcaops-master`.
3. In a terminal (e.g. the SageMath Shell on Windows), run the following:

```
sage -pip install --upgrade /path/to/gcaops/master
```

This completes the installation.

4. It is optional but highly recommended to configure a default directory where data (such as lists of graphs) can be stored, so it doesn't have to be re-computed each time.

This can be done by setting the environment variable `GCAOPS_DATA_DIR` to the path you desire, before starting SageMath. A convenient way to achieve this is by adding a line such as the following to SageMath's `shell script` `sagerc`:

```
export GCAOPS_DATA_DIR='/home/sage/Documents/gcaops_data/'
```

Be warned that this directory can grow large. If no directory is configured, then graphs are only stored in memory (which may be limiting).

5. It is optional but convenient to enable the importing of all names from the `gcaops` package (e.g. `UndirectedGraphComplex`) into the global namespace of every SageMath session, so that the functionality can be used immediately.

This can be done by adding the following line to SageMath's [startup script](#) `init.sage`:

```
from gcaops.all import *
```

Now the functionality can be imported, e.g.:

```
[1]: from gcaops.algebra.superfunction_algebra import SuperfunctionAlgebra
      from gcaops.algebra.differential_polynomial_ring import DifferentialPolynomialRing
      from gcaops.graph.undirected_graph_complex import UndirectedGraphComplex
      from gcaops.graph.directed_graph_complex import DirectedGraphComplex
```

Or import all the relevant functionality at once:

```
[2]: from gcaops.all import *
```

## 0.2 Functions

The meaning of the word “function” depends on the context. We shall presently consider three types of functions: those given (as coordinate expressions) by polynomials, by differential polynomials, and by symbolic expressions. We now describe how to work with each of these in SageMath.

### 0.2.1 Polynomials

We can restrict ourselves to functions defined by polynomials. For our purposes (that is, to work with formulas), polynomial functions (over a field of characteristic zero) are identified with polynomials in a polynomial ring.

```
[3]: R.<y1,y2,y3> = PolynomialRing(QQ); R
```

[3]: Multivariate Polynomial Ring in y1, y2, y3 over Rational Field

This simultaneously defines the variables `y1`, `y2`, `y3` and the ring `R` which acts as a parent for all the polynomials in these variables. All objects `f` in SageMath which are “elements” of some sort have a parent object `f.parent()` that they belong to. This is helpful for bookkeeping purposes, and in particular for *conversion* from one type of object to another, as will be seen later.

We can now define polynomials:

```
[4]: f = y1^2 + y2^2 - 1
```

```
[5]: f.parent() is R
```

[5]: True

The arithmetic with polynomials works as usual, and we can also differentiate polynomials with respect to the variables:

```
[6]: f.derivative(y1)
```

```
[6]: 2*y1
```

```
[7]: diff(f, y1)
```

```
[7]: 2*y1
```

For the notion of “parameters” in this context, one can define a polynomial ring over (the fraction field of) another polynomial ring.

```
[8]: C.<c1,c2> = PolynomialRing(QQ)
      S.<w1,w2,w3> = PolynomialRing(C)
      c1^2*w1 + (c1+c2)*w2^2 + w3^3
```

```
[8]: w3^3 + (c1 + c2)*w2^2 + c1^2*w1
```

## 0.2.2 Differential polynomials

We will often work with expressions involving sums of products of derivatives. The `gcaops` package provides differential polynomials, which are polynomials in jet variables.

Define a differential polynomial ring with fibre variable  $u$  and base variables  $z_1, z_2$ :

```
[9]: D = DifferentialPolynomialRing(QQ, ('u',), ('z1','z2'), max_differential_orders=[3]);
      ↪D
```

```
[9]: Differential Polynomial Ring in z1, z2, u, u_z1, u_z2, u_z1z1, u_z1z2, u_z2z2,
      u_z1z1z1, u_z1z1z2, u_z1z2z2, u_z2z2z2 over Rational Field
```

Retrieve the fibre and base variables:

```
[10]: u, = D.fibre_variables()
       z1, z2 = D.base_variables()
```

Arithmetic with differential polynomials works as usual. The (default) derivative in this context is the total derivative:

```
[11]: (u^2).derivative(z1)
```

```
[11]: 2*u*u_z1
```

```
[12]: diff(u^2, z1)
```

```
[12]: 2*u*u_z1
```

It is also possible to take partial derivatives:

```
[13]: (u^2).partial_derivative(u)
```

```
[13]: 2*u
```

### 0.2.3 Symbolic expressions

Finally we can consider functions defined by symbolic expressions (possibly more general than polynomials).

For this we first define symbolic variables, which we interpret as coordinates in a chart:

```
[14]: var('x1,x2,x3')
```

```
[14]: (x1, x2, x3)
```

The commutative arithmetic with these variables works as usual. An example of a function of  $(x_1, x_2, x_3)$  is:

```
[15]: g = x1^2*sin(x2^2)*exp(1/x3); g
```

```
[15]: x1^2*e^(1/x3)*sin(x2^2)
```

We can differentiate symbolic expressions:

```
[16]: diff(g, x3)
```

```
[16]: -x1^2*e^(1/x3)*sin(x2^2)/x3^2
```

The identification of functions with symbolic expressions does not limit us to elementary functions, because we can use symbolic (indeterminate) functions, evaluated at the coordinates:

```
[17]: h = function('h')(x1,x2,x3); h
```

```
[17]: h(x1, x2, x3)
```

```
[18]: diff(h, x1)
```

```
[18]: diff(h(x1, x2, x3), x1)
```

Symbolic expressions in SageMath are elements of a “ring”: the Symbolic Ring **SR**.

```
[19]: SR
```

```
[19]: Symbolic Ring
```

Of course this is not implemented as a set containing all symbolic expressions, but it acts as a parent for all symbolic expressions; that is, for any particular symbolic expression which actually occurs.

# Chapter 1

## Implementation of star products

This chapter is an introduction to the usage of star products, polydifferential operators, the Hochschild complex, and the associativity as a Maurer–Cartan equation in the `gcaops` software package. The theory is standard, see e.g. Chapter 11 and references therein, as well as the foundational papers by Gerstenhaber [22] [23], Hochschild [25], Groenewold [24], Weyl [34], DeWilde–Lecomte [14], Fedosov [19]; the expository works by Esposito [18], Cattaneo–Indelicato [12], and Gengoux–Pichereau–Vanhaecke [30]; from a historic perspective Bayen–Flato–Frønsdal–Lichnerowicz–Sternheimer [2], and also the MSc thesis of Willem de Kok at the University of Groningen [13] for interpretations and more references.

### 1.1 Star products

Let  $f, g, h$  be scalar functions on  $\mathbb{R}^2$ :

```
[1]: var('x,y')
      f = function('f')(x,y)
      g = function('g')(x,y)
      h = function('h')(x,y)
```

The ordinary pointwise product of functions is associative:

```
[2]: bool(f*(g*h) == (f*g)*h)
```

```
[2]: True
```

The pointwise product is also commutative:

```
[3]: g*f
```

```
[3]: f(x, y)*g(x, y)
```

```
[4]: bool(f*g == g*f)
```

```
[4]: True
```

We want to deform the pointwise product  $f \cdot g$  to a product  $f \star g = f \cdot g + \hbar B_1(f, g) + \hbar^2 B_2(f, g) + \dots$  where  $\hbar$  is a formal parameter,  $B_k$  is a bi-linear bi-differential operator on  $C^\infty(\mathbb{R}^d)$  for each  $k \geq 1$ , and  $\star$  remains associative (but not necessarily commutative).

Let us keep in mind that in physical applications, the formal parameter equals  $\frac{i\hbar}{2}$  where this  $\hbar = h/2\pi$  is the Planck constant.

**Example 1.** Here is an example  $\star_1$  of such a star-product modulo  $\bar{o}(\hbar^2)$ :

```
[5]: var('hbar')
star1 = Lambda f,g: f*g + hbar*x*(diff(f,x)*diff(g,y) - diff(f,y)*diff(g,x)) + \
hbar^2*x^2*(diff(f,x,x)*diff(g,y,y) - 2*diff(f,x,y)*diff(g,x,y) + \
↳diff(f,y,y)*diff(g,x,x))/2 + \
hbar^2*x*(diff(f,y,y)*diff(g,x) - diff(f,x,y)*diff(g,y) + diff(f,x)*diff(g,y,y) - \
↳diff(f,y)*diff(g,x,y))/3 + \
hbar^2*diff(f,y)*diff(g,y)/6
```

The product is noncommutative:

```
[6]: (star1(f,g) - star1(g,f)).coefficient(hbar)
```

```
[6]: -2*(diff(f(x, y), y)*diff(g(x, y), x) - diff(f(x, y), x)*diff(g(x, y), y))*x
```

```
[7]: star1(x,y) - star1(y,x)
```

```
[7]: 2*hbar*x
```

The product is associative modulo  $\bar{o}(\hbar^2)$ :

```
[8]: assoc1 = (star1(star1(f,g),h) - star1(f,star1(g,h))).expand()
```

```
[9]: assoc1.subs(hbar==0), assoc1.coefficient(hbar), assoc1.coefficient(hbar^2)
```

```
[9]: (0, 0, 0)
```

The above 3-tuple contains the free term in the associator and the terms at  $\hbar^1$  and  $\hbar^2$ .

**Example 2.** Here is another choice  $\star_2$  (cf. M. Kontsevich's [29]):

```
[10]: star2 = Lambda f,g: f*g + hbar*x*y*(diff(f,x)*diff(g,y) - diff(f,y)*diff(g,x))/2 + \
hbar^2*x^2*y^2*(diff(f,x,x)*diff(g,y,y) + diff(f,y,y)*diff(g,x,x) - \
↳2*diff(f,x,y)*diff(g,x,y))/8 + \
hbar^2*x*y*(x*diff(f,x,x)*diff(g,y) + y*diff(f,y,y)*diff(g,x) - \
↳x*diff(f,x,y)*diff(g,x) - y*diff(f,x,y)*diff(g,y))/12 + \
hbar^2*x*y*(y*diff(f,x)*diff(g,y,y) + x*diff(f,y)*diff(g,x,x) - \
↳y*diff(f,y)*diff(g,x,y) - x*diff(f,x)*diff(g,x,y))/12 + \
hbar^2*(x*y*diff(f,x)*diff(g,y) + x*y*diff(f,y)*diff(g,x) - y^2*diff(f,y)*diff(g,y) - \
↳x^2*diff(f,x)*diff(g,x))/24
```

It is also associative modulo  $\bar{o}(\hbar^2)$ :

```
[11]: assoc2 = (star2(star2(f,g),h) - star2(f,star2(g,h))).expand()
```

```
[12]: assoc2.subs(hbar==0), assoc2.coefficient(hbar), assoc2.coefficient(hbar^2)
```

```
[12]: (0, 0, 0)
```

We have the property that  $x \star_2 y = \exp(\hbar)(y \star_2 x)$ , see §1.30 in [29].

```
[13]: star2_symm = (star2(x,y) - (1 + hbar + hbar^2/2)*star2(y,x)).expand()
```

```
[14]: star2_symm.subs(hbar==0), star2_symm.coefficient(hbar), star2_symm.coefficient(hbar^2)
```

```
[14]: (0, 0, 0)
```

**Example 3.** Here is an example  $\star_3$  in which the first-order term  $B_1$  is a symmetric operator:

```
[15]: star3 = lambda f,g: f*g + hbar*(diff(f,x)*diff(g,y) + diff(f,y)*diff(g,x)) + \
hbar^2*(diff(f,x,x)*diff(g,y,y) + 2*diff(f,x,y)*diff(g,x,y) + \
->diff(f,y,y)*diff(g,x,x))/2
```

```
[16]: assoc3 = (star3(star3(f,g),h) - star3(f,star3(g,h))).expand()
```

```
[17]: assoc3.subs(hbar==0), assoc3.coefficient(hbar), assoc3.coefficient(hbar^2)
```

```
[17]: (0, 0, 0)
```

## 1.2 Gauge transformations

We can transform a star-product using a gauge transformation  $T : C^\infty(M)[[\hbar]] \rightarrow C^\infty(M)[[\hbar]]$ , namely a formally invertible unary  $\mathbb{R}[[\hbar]]$ -linear differential operator of the form  $T = \text{id} + \sum_{k=1}^{\infty} \hbar^k D_k$  where each  $D_k$  is a finite order differential operator. Gauge transformations  $T$  yield the gauged star-products  $f \star' g = T^{-1}(T(f) \star T(g))$ .

**Example 4.** We transform the star-product  $\star_1$  from the previous section using a gauge transformation  $T_1(f) = f + \hbar^2 \frac{\partial^2 f}{\partial y^2}$ :

```
[18]: T1 = lambda f: f + hbar^2*diff(f,y,y)
```

```
[19]: T1_inverse = lambda f: f - hbar^2*diff(f,y,y)
```

The gauged star product is  $f \star'_1 g = T_1^{-1}(T_1(f) \star_1 T_1(g))$ :

```
[20]: star1_gauged = lambda f,g: T1_inverse(star1(T1(f),T1(g))).expand()
```

This gauge transformation leaves the first order term untouched:

```
[21]: bool(star1_gauged(f,g).coefficient(hbar) == star1(f,g).coefficient(hbar))
```

```
[21]: True
```

The second order term of  $\star'_1$  has one summand fewer than  $\star_1$ :

```
[22]: (star1_gauged(f,g).coefficient(hbar^2) - star1(f,g).coefficient(hbar^2)).expand()
```

```
[22]: -2*diff(f(x, y), y)*diff(g(x, y), y)
```

The gauged star product is also associative modulo  $\bar{o}(\hbar^2)$ :

```
[23]: gauged_assoc = star1_gauged(f,star1_gauged(g,h)) - star1_gauged(star1_gauged(f,g),h)
```

```
[24]: gauged_assoc.subs(hbar==0), gauged_assoc.coefficient(hbar), gauged_assoc.
->coefficient(hbar^2)
```

[24]: (0, 0, 0)

**Example 5.** We transform the symmetric star-product  $\star_3$  from the previous section using a gauge transformation  $T_2(f) = f + \hbar \frac{\partial^2 f}{\partial x \partial y}$ :

[25]: `T2 = lambda f: f + hbar*diff(f,x,y)`

[26]: `T2_inverse = lambda f: f - hbar*diff(f,x,y) + hbar^2*diff(f,x,x,y,y)`

[27]: `star3_gauged = lambda f,g: T2_inverse(star3(T2(f),T2(g))).expand()`

In the transformed product  $\star'_3$  the first-order term is zero:

[28]: `star3_gauged(f,g).coefficient(hbar)`

[28]: 0

The product  $\star'_3$  is also associative modulo  $\bar{o}(\hbar^2)$ :

[29]: `gauged_assoc3 = star3_gauged(f,star3_gauged(g,h)) - star3_gauged(star3_gauged(f,g),h)`

[30]: `gauged_assoc3.subs(hbar==0), gauged_assoc3.coefficient(hbar), gauged_assoc3.  
↪coefficient(hbar^2)`

[30]: (0, 0, 0)

We inspect the vanishing of the associator for  $\star'_3$  only modulo  $\bar{o}(\hbar)$  because the inverse  $T_2^{-1}$  is taken only up to  $\bar{o}(\hbar)$ . By expanding  $T_2^{-1} \bmod \bar{o}(\hbar^2)$  we could inspect the vanishing of the associator already up to  $\bar{o}(\hbar^2)$ .

### 1.3 Polydifferential operators

We now rewrite the associativity equation using more algebraic structures; this will be helpful later.

Namely, we introduce the algebra of polydifferential operators; it is endowed with the structure of differential graded Lie algebra by using the Gerstenhaber bracket (see below).

[31]: `from gcaops.algebra.polydifferential_operator import PolyDifferentialOperatorAlgebra  
D.<ddx,ddy> = PolyDifferentialOperatorAlgebra(SR, var('x,y'),  
↪simplify='expand',is_zero='is_trivial_zero'); D`

[31]: Algebra of multi-linear polydifferential operators over Symbolic Ring with coordinates (x, y) and derivatives (ddx, ddy)

In this algebraic setting we can define a  $\star$ -product as an operator as follows (cf.  $\star_1$  in Example 1):

[32]: `m1 = D.multiplication_operator() + hbar*x*(D(ddx,ddy) - D(ddy,ddx)) + \\  
hbar^2*x^2*(D(ddx^2, ddy^2) - 2*D(ddx*ddy, ddx*ddy) + D(ddy^2, ddx^2))/2 + \\  
hbar^2*x*(D(ddy^2, ddx) - D(ddx*ddy, ddy) + D(ddx, ddy^2) - D(ddy,ddx*ddy))/3 + \\  
hbar^2*D(ddy,ddy)/6; m1`



[32]:  $(\text{id} \otimes \text{id}) + (\hbar x) \cdot (\text{ddx} \otimes \text{ddy}) + (-\hbar x) \cdot (\text{ddy} \otimes \text{ddx}) + (1/2 \hbar^2 x^2) \cdot (\text{ddx}^2 \otimes \text{ddy}^2) + (-\hbar^2 x^2) \cdot (\text{ddx} \cdot \text{ddy} \otimes \text{ddx} \cdot \text{ddy}) + (1/2 \hbar^2 x^2) \cdot (\text{ddy}^2 \otimes \text{ddx}^2) + (1/3 \hbar^2 x) \cdot (\text{ddy}^2 \otimes \text{ddx}) + (-1/3 \hbar^2 x) \cdot (\text{ddx} \cdot \text{ddy} \otimes \text{ddy}) + (1/3 \hbar^2 x) \cdot (\text{ddx} \otimes \text{ddy}^2) + (-1/3 \hbar^2 x) \cdot (\text{ddy} \otimes \text{ddx} \cdot \text{ddy}) + (1/6 \hbar^2) \cdot (\text{ddy} \otimes \text{ddy})$

The natural extension of composition to *multi*-linear operators is the following.

**Definition.** The pre-Lie product of  $(k_1 + 1)$ -linear polydifferential operator  $\Phi_1$  and of  $(k_2 + 1)$ -linear polydifferential operator  $\Phi_2$  is the sum of insertions of  $\Phi_2$  into an argument slot of  $\Phi_1$ ,

$$(\Phi_1 \circ \Phi_2)(a_0 \otimes \dots \otimes a_{k_1+k_2}) = \sum_{i=0}^{k_1} (-)^{ik_2} \Phi_1(a_0 \otimes \dots \otimes \Phi_2(a_i \otimes \dots \otimes a_{i+k_2}) \otimes a_{i+k_2+1} \otimes \dots \otimes a_{k_1+k_2}).$$

The *Gerstenhaber bracket* is the shifted-graded commutator of pre-Lie products,

$$[\Phi_1, \Phi_2]_G = \Phi_1 \circ \Phi_2 - (-)^{k_1 \cdot k_2} \Phi_2 \circ \Phi_1.$$

The Gerstenhaber bracket is thus shifted-graded skew-symmetric.

In the new algebraic language, a bi-linear bi-differential operator  $m$  is associative if and only if it satisfies the master equation  $[m, m]_G = 0$ .

(In what follows, we shall take the tensor products of algebras with the algebra of formal power series  $\mathbb{k}[[\hbar]]$ , so that the bi-linear bi-differential operator  $m$  can be a finite order bi-differential operator at every finite order of  $\hbar$ , but unbounded-order overall.)

[33]: `m1_assoc = (1/2)*m1.bracket(m1)`

[34]: `m1_assoc == m1.insertion(0,m1) - m1.insertion(1,m1)`

[34]: True

[35]: `m1_assoc.subs(hbar==0), m1_assoc.coefficient(hbar), m1_assoc.coefficient(hbar^2)`

[35]: (0, 0, 0)

Similarly, we can define  $\star_2$  from Example 2:

[36]: `m2 = D.multiplication_operator() + hbar*x*y*(D(ddx,ddy) - D(ddy,ddx))/2 + \
hbar^2*x^2*y^2*(D(ddx^2,ddy^2) + D(ddy^2,ddx^2) - 2*D(ddx*ddy, ddx*ddy))/8 + \
hbar^2*x*y*(x*D(ddx^2,ddy) + y*D(ddy^2,ddx) - x*D(ddx*ddy,ddx) - y*D(ddx*ddy,ddy))/12_
↪+ \
hbar^2*x*y*(y*D(ddx,ddy^2) + x*D(ddy,ddx^2) - y*D(ddy,ddx*ddy) - x*D(ddx,ddx*ddy))/12_
↪+ \
hbar^2*(x*y*D(ddx,ddy) + x*y*D(ddy,ddx) - y^2*D(ddy,ddy) - x^2*D(ddx,ddx))/24`

[37]: `m2_assoc = (1/2)*m2.bracket(m2)`

[38]: `m2_assoc.subs(hbar==0), m2_assoc.coefficient(hbar), m2_assoc.coefficient(hbar^2)`

[38]: (0, 0, 0)

Likewise,  $\star_3$  from Example 3:

```
[39]: m3 = D.multiplication_operator() + hbar*(D(ddx,ddy) + D(ddy,ddx)) + \
hbar^2*(D(ddx^2,ddy^2) + 2*D(ddx*ddy,ddx*ddy) + D(ddy^2,ddx^2))/2; m3
```

```
[39]: (id ⊗ id) + (hbar)*(ddx ⊗ ddy) + (hbar)*(ddy ⊗ ddx) + (1/2*hbar^2)*(ddx^2 ⊗ ddy^2) +
(hbar^2)*(ddx*ddy ⊗ ddx*ddy) + (1/2*hbar^2)*(ddy^2 ⊗ ddx^2)
```

```
[40]: m3_assoc = (1/2)*m3.bracket(m3)
```

```
[41]: m3_assoc.subs(hbar==0), m3_assoc.coefficient(hbar), m3_assoc.coefficient(hbar^2)
```

```
[41]: (0, 0, 0)
```

The Gerstenhaber bracket satisfies its own shifted-graded Jacobi identity,

$$(-)^{(|a|-1)(|c|-1)}[a, [b, c]_G]_G + (-)^{(|b|-1)(|a|-1)}[b, [c, a]_G]_G + (-)^{(|c|-1)(|b|-1)}[c, [a, b]_G]_G = 0,$$

where  $|a|$  is the arity (number of arguments) of the polydifferential operator  $a$ .

Here is an example (for brevity, we calculate the arity signs by hand):

```
[42]: jac_m123 = (-1)*m1.bracket(m2.bracket(m3)) + (-1)*m2.bracket(m3.bracket(m1)) +
↪ (-1)*m3.bracket(m1.bracket(m2))
```

```
[43]: jac_m123
```

```
[43]: 0
```

Taking the Gerstenhaber bracket with an associative product  $m$  is called the Hochschild differential,  $d_H = [m, -]_G$ .

Let  $\mu$  be the ordinary product in the algebra of scalar functions  $C^\infty(\mathbb{R}^d)$ ; the respective Hochschild differential is implemented as the `hochschild_differential` method. Indeed, it is a differential:

```
[44]: m1.hochschild_differential().hochschild_differential()
```

```
[44]: 0
```

Moreover, taking the bracket with a product which is associative modulo  $\bar{o}(\hbar^n)$  is also a differential modulo  $\bar{o}(\hbar^n)$ :

```
[45]: m2_diff_m1 = m1.bracket(m1.bracket(m2))
m2_diff_m1.subs(hbar==0), m2_diff_m1.coefficient(hbar), m2_diff_m1.coefficient(hbar^2)
```

```
[45]: (0, 0, 0)
```

We recall from Gerstenhaber [23] that the problem of extending an associative product  $f \star g = f \cdot g + \sum_{k=1}^n \hbar^k B_k(f, g) + \bar{o}(\hbar^n)$ , already given modulo  $\bar{o}(\hbar^n)$  for  $n \geq 1$ , to a next-order associative product modulo  $\bar{o}(\hbar^{n+1})$  is equivalent to expressing the Hochschild 3-cocycle  $C_{n+1} = -\frac{1}{2} \sum_{i \neq 0, j \neq 0}^{i+j=n+1} [B_i, B_j]$  as a 3-coboundary  $d_H(B_{n+1})$  with respect to the Hochschild differential  $d_H = [\mu, -]$  containing the leading term associative structure  $\mu$  at  $\hbar^0$ .

Let us define the object  $C_2$  for the product  $\star_1$  and show that it is a Hochschild 3-cocycle:

```
[46]: C2 = -(1/2)*(m1.coefficient(hbar).bracket(m1.coefficient(hbar)))
      C2.hochschild_differential()
```

[46]: 0

Indeed, it equals the 3-coboundary  $d_H(B_2)$ :

```
[47]: m1.coefficient(hbar^2).hochschild_differential() == C2
```

[47]: True

Also,  $C_3$  is a Hochschild 3-cocycle:

```
[48]: C3 = -(1/2)*(m1.coefficient(hbar).bracket(m1.coefficient(hbar^2)) + m1.
      ↪coefficient(hbar^2).bracket(m1.coefficient(hbar)))
      C3.hochschild_differential()
```

[48]: 0

The associativity of a binary operator  $\star = \mu + B$  can be expressed as the Maurer–Cartan equation for  $B$ , which reads  $d_H(B) + \frac{1}{2}[B, B] = 0$ .

Let us give an example based on  $\star_1$ :

```
[49]: B = m1 - D.multiplication_operator()
      mc = B.hochschild_differential() + (1/2)*B.bracket(B)
      mc.subs(hbar==0), mc.coefficient(hbar), mc.coefficient(hbar^2)
```

[49]: (0, 0, 0)

This is relevant for the Formality morphism which sends the Maurer–Cartan elements  $P$  in the Poisson world to the Maurer–Cartan elements  $B$  in the associative world (see Chapter 3).

Gauge transformations  $T$  (which we discussed in Section 1.2) mod  $\bar{o}(\hbar^k)$  which are concentrated in degree  $k$  (that is, without terms at  $\hbar^\ell$  for  $\ell \neq 0$  and  $\ell \neq k$ ), act on associative star-product expansions modulo  $\bar{o}(\hbar^k)$  in a particular way, namely by adding Hochschild coboundaries:

**Proposition.** *If  $\star' = \sum \hbar^n B'_n \bmod \bar{o}(\hbar^k)$  is a star-product expansion obtained by a gauge transformation of the form  $T = \text{id} + \hbar^k T_k \bmod \bar{o}(\hbar^k)$  acting on  $\star = \sum \hbar^n B_n \bmod \bar{o}(\hbar^k)$  via  $f \star' g = T^{-1}(T(f) \star T(g))$ , then the difference  $B'_k - B_k$  is a Hochschild coboundary; specifically it is none other than  $d_H(T_k)$ .*

**Example 6.** We revisit the gauge transformation  $T_1(f) = f + \hbar^2 \frac{\partial^2 f}{\partial y^2}$  of  $\star_1 \bmod \bar{o}(\hbar^2)$  from Example 4.

```
[50]: t1 = D.identity_operator() + hbar^2*ddy^2; t1
```

[50]: id + (hbar^2)\*ddy^2

We now have  $k = 2$ , so that the gauge transformation acts nontrivially at  $\hbar^2$  and its inverse is taken modulo  $\bar{o}(\hbar^2)$ .

```
[51]: t1_inverse = D.identity_operator() - hbar^2*ddy^2; t1_inverse
```

[51]: `id + (-hbar^2)*ddy^2`

We (re)calculate the gauged star-product expansion  $\star'_1 \bmod \bar{o}(\hbar^2)$ :

[52]: `m1_gauged = t1_inverse.insertion(0, m1.insertion(0, t1).insertion(1, t1))`

At  $\hbar^2$ , the difference  $\star'_1 - \star_1$  consists of a single term:

[53]: `m1_gauged.coefficient(hbar^2) - m1.coefficient(hbar^2)`

[53]: `(-2)*(ddy @ ddy)`

That term is indeed a Hochschild coboundary  $d_H(\frac{\partial^2}{\partial y^2})$ , i.e. the Hochschild differential applied to the second-order part in the gauge transformation:

[54]: `(ddy^2).hochschild_differential()`

[54]: `(-2)*(ddy @ ddy)`

This line of reasoning about gauge transformations is continued in Chapter 9.

Each  $\star$ -product gives rise to a bracket on functions defined by

$$\{f, g\}_\star = \left. \frac{f \star g - g \star f}{2\hbar} \right|_{\hbar=0} = \frac{1}{2}(B_1(f, g) - B_1(g, f)).$$

**Example 7.** Here are the brackets associated with three star products in this section:

$\{-, -\}_{\star_1}$ :

[55]: `m1.coefficient(hbar).skew_symmetrization()/2`

[55]: `(x)*(ddx @ ddy) + (-x)*(ddy @ ddx)`

$\{-, -\}_{\star_2}$ :

[56]: `m2.coefficient(hbar).skew_symmetrization()/2`

[56]: `(1/2*x*y)*(ddx @ ddy) + (-1/2*x*y)*(ddy @ ddx)`

$\{-, -\}_{\star_3}$ :

[57]: `m3.coefficient(hbar).skew_symmetrization()/2`

[57]: `0`

By construction the bracket  $\{f, g\}_\star$  associated with a star product  $\star$  is bi-linear and skew-symmetric. Moreover the associativity of  $\star \bmod \bar{o}(\hbar^1)$  implies that  $\{f, g\}_\star$  is a bi-derivation, and associativity  $\bmod \bar{o}(\hbar^2)$  implies that  $\{f, g\}_\star$  satisfies the Jacobi identity

$$\{f, \{g, h\}_\star\}_\star + \{g, \{h, f\}_\star\}_\star + \{h, \{f, g\}_\star\}_\star = 0.$$

These properties of  $\{f, g\}_\star$  are the defining properties of a *Poisson bracket*. We will study Poisson brackets in the next chapter, and in Chapter 3 we explore Kontsevich's solution to the natural inverse problem of constructing a star-product  $\star$  such that  $\{-, -\}_\star$  equals a given Poisson bracket.

See [12] and references therein for more details about the algebraic structures in this section.



# Chapter 2

## Implementation of Poisson structures

This chapter is an introduction to the usage of Poisson structures and (stable) Poisson cohomology in the `gcaops` software package; the theory is standard (see e.g. [30]).

### 2.1 Superfunctions

Superfunctions in a chart are polynomials in odd variables  $\xi_k$  satisfying the anticommutation relations  $\xi_j \xi_i = -\xi_i \xi_j$  (in particular  $\xi_i^2 = 0$ ) with functions as coefficients of those polynomials.

Define an algebra of superfunctions on  $\mathbb{R}^3$  with variables  $x^1, x^2, x^3$  as even coordinates (the symbolic ring `SR` is considered to be the base ring):

```
[1]: from gcaops.algebra.superfunction_algebra import SuperfunctionAlgebra
SA.<xi1,xi2,xi3> = SuperfunctionAlgebra(SR, var('x1,x2,x3'))
print(SA)
```

Superfunction algebra over Symbolic Ring with even coordinates (x1, x2, x3) and odd coordinates (xi1, xi2, xi3)

This simultaneously defines the odd coordinates `xi1`, `xi2`, `xi3` and the object `SA` which acts as a parent for all the superfunctions depending on these variables. We can create superfunctions by entering them as polynomials in the odd coordinates.

Do some basic arithmetic:

```
[2]: xi2*xi1
```

```
[2]: (-1)*xi1*xi2
```

```
[3]: xi1^2
```

```
[3]: 0
```

```
[4]: x1*xi1 + x2*xi2
```



```
[4]: (x1)*xi1 + (x2)*xi2
```

Retrieve the coefficients of a superfunction (as a polynomial in the odd variables  $\xi_k$ ):

```
[5]: V = x1*xi1 + x2*xi2
V[0], V[1]
```

```
[5]: (x1, x2)
```

```
[6]: B = x1*xi2*xi3 + x2*xi3*xi1 + x3*xi1*xi2
B[0,2]
```

```
[6]: -x2
```

```
[7]: B[2,0]
```

```
[7]: x2
```

Calculate even and odd derivatives (note e.g.  $\frac{\partial}{\partial \xi_2}(\xi_1 \xi_2) = \frac{\partial}{\partial \xi_2}(-\xi_2 \xi_1) = -\frac{\partial}{\partial \xi_2}(\xi_2 \xi_1) = -\xi_1$ ):

```
[8]: (x2^2*xi1*xi2).diff(x2)
```

```
[8]: (2*x2)*xi1*xi2
```

```
[9]: (xi1*xi2).diff(xi1)
```

```
[9]: xi2
```

```
[10]: (xi1*xi2).diff(xi2)
```

```
[10]: (-1)*xi1
```

By default, the coefficients of results are not automatically simplified or expanded (on the other hand, the products of supervariables  $\xi_{i_1} \cdot \xi_{i_2} \cdot \dots$  are normalized by the ordering  $i_1 < i_2 < \dots$ ):

```
[11]: Z = (x1+x2)^2*xi1*xi2 + (x1^2 + 2*x1*x2 + x2^2)*xi2*xi1; Z
```

```
[11]: ((x1 + x2)^2 - x1^2 - 2*x1*x2 - x2^2)*xi1*xi2
```

But the expression can be simplified manually, by calling the `expand` method on each coefficient:

```
[12]: Z.map_coefficients(lambda z: z.expand())
```

```
[12]: 0
```

For convenience, in the definition of the superfunction algebra we can pass a `simplify` method to be applied to the coefficients after each operation:

```
[13]: SA.<xi1,xi2,xi3> = SuperfunctionAlgebra(SR, var('x1,x2,x3'), simplify='expand',
↪is_zero='is_trivial_zero')
print(SA)
```

Superfunction algebra over Symbolic Ring with even coordinates (x1, x2, x3) and odd coordinates (xi1, xi2, xi3)

Now the tool simplifies all coefficients before displaying them:

```
[14]: W = (x1+x2)^2*x11 - (x1^2 + 2*x1*x2 + x2^2)*xi1; W
```

```
[14]: 0
```

## 2.2 Vector fields

Vector fields  $X^i \frac{\partial}{\partial x^i}$  can be identified with superfunctions  $X^i \xi_i$  which are linear in the odd coordinates  $\xi_i$ :

```
[15]: X = x1*x11
      Y = x2*x11 + x3*x12
```

```
[16]: X + Y
```

```
[16]: (x1 + x2)*xi1 + (x3)*xi2
```

The Lie bracket of vector fields is  $[X, Y] = \sum_j (X^i \frac{\partial}{\partial x^i} Y^j - Y^i \frac{\partial}{\partial x^i} X^j) \frac{\partial}{\partial x^j}$ :

```
[17]: X.bracket(Y)
```

```
[17]: (-x2)*xi1
```

## 2.3 Bi-vector fields

Bi-vector fields  $B^{ij} \frac{\partial}{\partial x^i} \wedge \frac{\partial}{\partial x^j}$  can be identified with superfunctions  $B^{ij} \xi_i \xi_j$  quadratic in  $\xi_k$ .

An example of such an object is a wedge product of vector fields:

```
[18]: X*Y
```

```
[18]: (x1*x3)*xi1*xi2
```

The Schouten bracket (or odd Poisson bracket on a finite-dimensional (super)manifold of finite (super)dimension  $(d|d)$ ),

$$[[A, B]] = A \overleftarrow{\frac{\partial}{\partial \xi_k}} \overrightarrow{\frac{\partial}{\partial x^k}} B - A \overleftarrow{\frac{\partial}{\partial x^k}} \overrightarrow{\frac{\partial}{\partial \xi_k}} B,$$

is the natural extension of the Lie bracket of vector fields on the underlying manifold of dimension  $d$ .

```
[19]: Y.bracket(X*Y)
```

```
[19]: (x2*x3)*xi1*xi2
```

```
[20]: Y*X.bracket(Y)
```

```
[20]: (x2*x3)*xi1*xi2
```

Let  $B$  be a generic bi-vector field:

```
[21]: B = function('B12')(x1,x2,x3)*xi1*xi2 + function('B13')(x1,x2,x3)*xi1*xi3 +
      ↪function('B23')(x1,x2,x3)*xi2*xi3; B
```

```
[21]: (B12(x1, x2, x3))*xi1*xi2 + (B13(x1, x2, x3))*xi1*xi3 + (B23(x1, x2, x3))*xi2*xi3
```

Then  $\llbracket B, X \rrbracket$  is the following bi-vector field:

```
[22]: B.bracket(X)
```

```
[22]: (-x1*diff(B12(x1, x2, x3), x1) + B12(x1, x2, x3))*xi1*xi2 + (-x1*diff(B13(x1, x2, x3),
      x1) + B13(x1, x2, x3))*xi1*xi3 + (-x1*diff(B23(x1, x2, x3), x1))*xi2*xi3
```

More generally an  $m$ -vector field is represented by a homogeneous polynomial of degree  $m$  in the odd coordinates  $\xi_k$ .

## 2.4 Poisson structures

Poisson structures are bi-vector fields  $P$  satisfying the Jacobi identity  $\frac{1}{2}\llbracket P, P \rrbracket = 0$ .

Here is an example of a rescaled Nambu–Poisson structure:

```
[23]: rho = x1^2 - x2*x3
      a = (x1^2 + x2^2 + x3^2)/2
      P = rho*(diff(a,x1)*xi2*xi3 + diff(a,x2)*xi3*xi1 + diff(a,x3)*xi1*xi2); P
```

```
[23]: (x1^3 - x1*x2*x3)*xi2*xi3 + (-x1^2*x2 + x2^2*x3)*xi1*xi3 + (x1^2*x3 - x2*x3^2)*xi1*xi2
```

```
[24]: P.bracket(P)
```

```
[24]: 0
```

Another natural class of Poisson structures consists of those of the form  $P = E \wedge V$ , where  $E = x^i \frac{\partial}{\partial x^i}$  is the Euler vector field and  $V$  is a vector field such that each of its coefficients is a homogeneous polynomial of the same degree.

```
[25]: E = x1*xi1 + x2*xi2 + x3*xi3
      V = (x1^2-x2*x3)*xi1 + (x2^2-x1*x3)*xi2 + (x3^2-x1*x2)*xi3
      P_wedge = E*V; P_wedge
```

```
[25]: (-x1^2*x2 + x1*x2^2 - x1^2*x3 + x2^2*x3)*xi1*xi2 + (-x1^2*x2 - x1^2*x3 + x1*x3^2 +
      x2*x3^2)*xi1*xi3 + (-x1*x2^2 - x2^2*x3 + x1*x3^2 + x2*x3^2)*xi2*xi3
```

```
[26]: P_wedge.bracket(P_wedge)
```

```
[26]: 0
```

## 2.5 Poisson complex

The graded Jacobi identity for the Schouten bracket implies that  $\partial_P = \llbracket P, - \rrbracket$  for a Poisson structure  $P$  defines a differential on the space of multivector fields ( $\partial_P^2 = 0$ ).

We illustrate this for the Poisson structure  $P$  from the previous section:

```
[27]: P.bracket(P.bracket(x1^2 + x2^3))
```

```
[27]: 0
```

```
[28]: P.bracket(P.bracket(x1*xi1 + x2*xi2))
```

```
[28]: 0
```

Hence, there is the Poisson complex of  $P$ , where the cochains are multivector fields (starting with functions as 0-vector fields).

An  $m$ -vector field in the kernel of the Poisson differential is called a Poisson  $m$ -cocycle:

```
[29]: H = x1^2 + x2^2 + x3^2
P.bracket(H)
```

```
[29]: 0
```

```
[30]: V0 = (x1^2*x3 - x2*x3^2)*xi2 + (-x1^2*x2 + x2^2*x3)*xi3
P.bracket(V0)
```

```
[30]: 0
```

So,  $H$  is a Poisson 0-cocycle for the bi-vector  $P$ , and an example of a Poisson 1-cocycle is  $V_0$ .

An  $m$ -vector field in the image of the Poisson differential is called a Poisson  $m$ -coboundary:

```
[31]: E = x1*xi1 + x2*xi2 + x3*xi3
PbracketE = P.bracket(E); PbracketE
```

```
[31]: (x1^2*x2 - x2^2*x3)*xi1*xi3 + (-x1^2*x3 + x2*x3^2)*xi1*xi2 + (-x1^3 +
x1*x2*x3)*xi2*xi3
```

The object  $E$  is the Euler vector field, and the Poisson structure  $P$  is homogeneous:

```
[32]: PbracketE == -P
```

```
[32]: True
```

Every Poisson coboundary is a Poisson cocycle:

```
[33]: P.bracket(PbracketE)
```

```
[33]: 0
```

A Poisson cohomology class is the equivalence class of a Poisson cocycle modulo arbitrary Poisson coboundaries for that Poisson differential.

## 2.6 Homogeneous polynomial Poisson complex

Construct the algebra of polynomial superfunctions over a polynomial ring  $\mathbb{Q}[x^1, x^2, x^3]$ :

```
[34]: PSA = SuperfunctionAlgebra(PolynomialRing(QQ, names='x1,x2,x3'), names='xi1,xi2,xi3')
print(PSA)
```

Superfunction algebra over Multivariate Polynomial Ring in x1, x2, x3 over Rational Field with even coordinates (x1, x2, x3) and odd coordinates (xi1, xi2, xi3)

Consider  $P$  as a homogeneous polynomial Poisson structure, by converting it into an element of PSA:

```
[35]: P_poly = PSA(P); P_poly
```

```
[35]: (x1^3 - x1*x2*x3)*xi2*xi3 + (-x1^2*x2 + x2^2*x3)*xi1*xi3 + (x1^2*x3 - x2*x3^2)*xi1*xi2
```

```
[36]: P_poly.bracket(P_poly)
```

```
[36]: 0
```

For a Poisson structure  $P$  with homogeneous polynomial coefficients, the Poisson (sub)complex of homogeneous polynomial Poisson cochains has finite-dimensional bigraded components, and if a homogeneous polynomial Poisson cochain is a coboundary (in the smooth Poisson complex), then it is in particular the coboundary of a homogeneous polynomial Poisson cochain.

*Proof:* The map  $j_0^\infty$  that takes a smooth function to its infinite jet at the origin extends to a map from the smooth Poisson complex (with differential  $d_P$ ) to the formal Poisson complex (with differential  $d_{P_0}$  where  $P_0 = j_0^\infty(P)$ ) which respects the differentials:  $d_{P_0} \circ j_0^\infty = j_0^\infty \circ d_P$ . Suppose  $Q = d_P(X)$  is a  $d_P$ -coboundary. Then  $j_0^\infty(Q) = j_0^\infty(d_P(X)) = d_{P_0}(j_0^\infty(X))$  is a  $d_{P_0}$ -coboundary. Now taking Poisson cochains  $P, Q$  with homogeneous polynomial coefficients of degrees  $p, q$  respectively implies  $Q = j_0^\infty(Q)$  and  $P = j_0^\infty(P)$ , hence  $Q = d_P(j_0^\infty(X))$ , where  $j_0^\infty(X)$  has formal power series coefficients. Finally, compare degrees: since  $Q$  has homogeneous polynomial coefficients of degree  $q$  and  $d_P$  is homogeneous of degree  $p - 1$ , it follows that  $j_0^\infty(X) = X_0 + X'$  where  $X_0$  has homogeneous polynomial coefficients of degree  $q - (p - 1)$  and  $d_P(X_0) = Q$ , as desired. (The remaining terms in  $X'$  can be ignored as  $d_P(X') = 0$ .)

This proven possibility to work only with polynomials splits the problem of determining Poisson cohomology for homogeneous Poisson structures into subproblems that can be solved using finite-dimensional linear algebra, as will be seen below.

```
[37]: from gcaops.algebra.homogeneous_polynomial_poisson_complex import PoissonComplex
PC = PoissonComplex(P_poly)
print(PC)
```

```
Poisson complex of (x1^3 - x1*x2*x3)*xi2*xi3 + (-x1^2*x2 + x2^2*x3)*xi1*xi3 + (x1^2*x3
- x2*x3^2)*xi1*xi2
```

Consider  $P\_poly$  as a cochain in its own Poisson complex:

```
[38]: PC(P_poly)
```

[38]: Poisson cochain  $(x_1^2 x_3 - x_2 x_3^2) x_{i1} x_{i2} + (-x_1^2 x_2 + x_2^2 x_3) x_{i1} x_{i3} + (x_1^3 - x_1 x_2 x_3) x_{i2} x_{i3}$

[39]: `PC(P_poly).differential()`

[39]: Poisson cochain  $\theta$

Test whether this cocycle is a coboundary:

[40]: `PC(P_poly).is_coboundary()`

[40]: True

Find a trivializing vector field  $V$  such that  $\llbracket P, V \rrbracket = P$ :

[41]: `V = PC(P_poly).is_coboundary(certificate=True)[1]; V`

[41]: Poisson cochain  $(-x_1) x_{i1} + (-x_2) x_{i2} + (-x_3) x_{i3}$

We obtain  $V = -E$  where  $E$  is the Euler vector field.

Determine a basis of the cohomology in several bi-graded components (with respect to arity and degree):

[42]: `PC.cohomology_basis(2,3)`

[42]: [Poisson cochain  $(x_1^2 x_2) x_{i1} x_{i2} + (-x_2^3 - x_1^2 x_3 + x_2 x_3^2) x_{i1} x_{i3} + (x_1 x_2^2) x_{i2} x_{i3}$ ,  
Poisson cochain  $(x_1^2 x_3) x_{i1} x_{i2} + (-x_2^2 x_3) x_{i1} x_{i3} + (x_1 x_2 x_3) x_{i2} x_{i3}$ ,  
Poisson cochain  $(x_2^3) x_{i1} x_{i2} + (x_2^3 - x_2 x_3^2 - x_3^3) x_{i1} x_{i3} + (-3 x_1 x_2^2 - 3 x_1 x_2 x_3 - 2 x_1 x_3^2) x_{i2} x_{i3}$ ]

[43]: `PC.cohomology_basis(1,2)`

[43]: [Poisson cochain  $(x_2^2 - x_3^2) x_{i1} + (-x_1 x_2 - 2 x_1 x_3) x_{i2} + (2 x_1 x_2 + x_1 x_3) x_{i3}$ ]

In bi-grading  $(2, 3)$  we have obtained all the second Poisson cohomology classes which are represented by bi-vector cocycles with polynomial coefficients of degree 3. Likewise, we learn that there is a unique cohomology class of Poisson 1-cocycles with degree 2 polynomial coefficients.

## 2.7 Stable Poisson cocycles

The condition for a multivector to be a Poisson cocycle is generally non-trivial:

[44]: `P.bracket(x1)`

[44]:  $(x_1^2 x_2 - x_2^2 x_3) x_{i3} + (-x_1^2 x_3 + x_2 x_3^2) x_{i2}$

[45]: `P.bracket(xi1)`

[45]:  $(-3 x_1^2 + x_2 x_3) x_{i2} x_{i3} + (2 x_1 x_2) x_{i1} x_{i3} + (-2 x_1 x_3) x_{i1} x_{i2}$

Nevertheless, there exist formulas — depending on the Poisson structure coefficients  $P^{ij}$  — that *always* define a Poisson cocycle, such as [27]:

$$Q_{\text{tetra}}(P) = 1 \cdot \left( \frac{\partial^3 P^{ij}}{\partial x^k \partial x^\ell \partial x^m} \frac{\partial P^{kk'}}{\partial x^{\ell'}} \frac{\partial P^{\ell\ell'}}{\partial x^{m'}} \frac{\partial P^{mm'}}{\partial x^{k'}} \right) \frac{\partial}{\partial x^i} \wedge \frac{\partial}{\partial x^j} \\ + 6 \cdot \left( \frac{\partial^2 P^{ij}}{\partial x^k \partial x^\ell} \frac{\partial^2 P^{km}}{\partial x^{k'} \partial x^{\ell'}} \frac{\partial P^{k'\ell}}{\partial x^{m'}} \frac{\partial P^{m'\ell'}}{\partial x^j} \right) \frac{\partial}{\partial x^i} \wedge \frac{\partial}{\partial x^m}.$$

It can be implemented as follows (see also the tetrahedron graph  $\gamma_3$  below):

```
[46]: x = SA.even_coordinates()
xi = SA.odd_coordinates()
dimension = len(x)
import itertools
Gamma1 = sum(sum(diff(P[i,j],x[k],x[l],x[m]) * diff(P[k,ka],x[la]) *
↳diff(P[l,la],x[ma]) * diff(P[m,ma],x[ka]) for (k,l,m,ka,la,ma) in itertools.
↳product(range(0,dimension),repeat=6))*xi[i]*xi[j] for (i,j) in itertools.
↳combinations(range(0,dimension),2))
Gamma2 = sum(sum((diff(P[i,j],x[k],x[l]) * diff(P[k,m],x[ka],x[la]) -
↳diff(P[m,j],x[k],x[l]) * diff(P[k,i],x[ka],x[la]))/2 * diff(P[ka,l],x[ma]) *
↳diff(P[ma,la],x[j]) for (j,k,l,ka,la,ma) in itertools.
↳product(range(0,dimension),repeat=6))*xi[i]*xi[m] for (i,m) in itertools.
↳combinations(range(0,dimension),2))
Q_tetra = 1*Gamma1 + 6*Gamma2
Q_tetra
```

```
[46]: (36*x1^3*x2^2*x3 - 36*x1*x2^3*x3^2 - 36*x1^3*x3^3 + 36*x1*x2*x3^4)*xi1*xi2 +
(-36*x1^3*x2^3 + 36*x1*x2^4*x3 + 36*x1^3*x2*x3^2 - 36*x1*x2^2*x3^3)*xi1*xi3 +
(36*x1^4*x2^2 - 36*x1^2*x2^3*x3 - 36*x1^4*x3^2 + 36*x1^2*x2*x3^3)*xi2*xi3
```

Indeed,  $Q_{\text{tetra}}(P)$  is a Poisson cocycle for this particular Poisson bi-vector  $P$ :

```
[47]: P.bracket(Q_tetra)
```

```
[47]: 0
```

A good question is whether this Poisson 2-cocycle is or is not a Poisson coboundary.

Define a vector field  $V$  with undetermined coefficients:

```
[48]: V = function('V1')(x1,x2,x3)*xi1 + function('V2')(x1,x2,x3)*xi2 +
↳function('V3')(x1,x2,x3)*xi3; V
```

```
[48]: (V1(x1, x2, x3))*xi1 + (V2(x1, x2, x3))*xi2 + (V3(x1, x2, x3))*xi3
```

The equation to solve is  $\llbracket P, V \rrbracket = Q_{\text{tetra}}$ , or equivalently the vanishing of the components of the following bi-vector field:

```
[49]: P.bracket(V) - Q_tetra
```

```
[49]: (36*x1^3*x2^3 - 36*x1*x2^4*x3 - 36*x1^3*x2*x3^2 + 36*x1*x2^2*x3^3 -
x1^2*x2*diff(V1(x1, x2, x3), x1) + x2^2*x3*diff(V1(x1, x2, x3), x1) + x1^3*diff(V1(x1,
x2, x3), x2) - x1*x2*x3*diff(V1(x1, x2, x3), x2) + x1^2*x3*diff(V3(x1, x2, x3), x2) -
x2*x3^2*diff(V3(x1, x2, x3), x2) - x1^2*x2*diff(V3(x1, x2, x3), x3) +
x2^2*x3*diff(V3(x1, x2, x3), x3) + 2*x1*x2*V1(x1, x2, x3) + x1^2*V2(x1, x2, x3) -
2*x2*x3*V2(x1, x2, x3) - x2^2*V3(x1, x2, x3))*xi1*xi3 + (-36*x1^4*x2^2 +
36*x1^2*x2^3*x3 + 36*x1^4*x3^2 - 36*x1^2*x2*x3^3 - x1^2*x2*diff(V2(x1, x2, x3), x1) +
```



$$\begin{aligned}
& x2^2*x3*diff(V2(x1, x2, x3), x1) + x1^3*diff(V2(x1, x2, x3), x2) - \\
& x1*x2*x3*diff(V2(x1, x2, x3), x2) - x1^2*x3*diff(V3(x1, x2, x3), x1) + \\
& x2*x3^2*diff(V3(x1, x2, x3), x1) + x1^3*diff(V3(x1, x2, x3), x3) - \\
& x1*x2*x3*diff(V3(x1, x2, x3), x3) - 3*x1^2*V1(x1, x2, x3) + x2*x3*V1(x1, x2, x3) + \\
& x1*x3*V2(x1, x2, x3) + x1*x2*V3(x1, x2, x3))*xi2*xi3 + (-36*x1^3*x2^2*x3 + \\
& 36*x1*x2^3*x3^2 + 36*x1^3*x3^3 - 36*x1*x2*x3^4 + x1^2*x3*diff(V1(x1, x2, x3), x1) - \\
& x2*x3^2*diff(V1(x1, x2, x3), x1) - x1^3*diff(V1(x1, x2, x3), x3) + \\
& x1*x2*x3*diff(V1(x1, x2, x3), x3) + x1^2*x3*diff(V2(x1, x2, x3), x2) - \\
& x2*x3^2*diff(V2(x1, x2, x3), x2) - x1^2*x2*diff(V2(x1, x2, x3), x3) + \\
& x2^2*x3*diff(V2(x1, x2, x3), x3) - 2*x1*x3*V1(x1, x2, x3) + x3^2*V2(x1, x2, x3) - \\
& x1^2*V3(x1, x2, x3) + 2*x2*x3*V3(x1, x2, x3))*xi1*xi2
\end{aligned}$$

In general  $\llbracket P, V \rrbracket = Q_{\text{tetra}}$  is a difficult-to-solve system of partial differential equations.

If  $P$  and  $Q$  have homogeneous polynomial coefficients, then the existence of smooth  $V$  satisfying  $Q = \llbracket P, V \rrbracket$  implies that there is such a  $V$  with homogeneous polynomial coefficients, so it suffices to search for  $V$  of that type.

In our case  $P$  has degree  $d = 3$  and  $Q$  has degree  $4d - 6 = 6$ , so  $\deg(Q) = \deg(\llbracket P, V \rrbracket) = \deg(P) + \deg(V) - 1$  implies  $V$  must be of degree 4.

Convert  $Q_{\text{tetra}}$  from a superfunction with symbolic coefficients to a superfunction with polynomial coefficients:

```
[50]: Q_tetra_poly = PSA(Q_tetra); Q_tetra_poly
```

```
[50]: (36*x1^3*x2^2*x3 - 36*x1*x2^3*x3^2 - 36*x1^3*x3^3 + 36*x1*x2*x3^4)*xi1*xi2 +
(-36*x1^3*x2^3 + 36*x1*x2^4*x3 + 36*x1^3*x2*x3^2 - 36*x1*x2^2*x3^3)*xi1*xi3 +
(36*x1^4*x2^2 - 36*x1^2*x2^3*x3 - 36*x1^4*x3^2 + 36*x1^2*x2*x3^3)*xi2*xi3
```

Test whether it is a coboundary in the Poisson complex of  $P$ :

```
[51]: PC(Q_tetra_poly).is_coboundary()
```

```
[51]: True
```

Find a trivializing vector field  $V$  such that  $\llbracket P, V \rrbracket = Q_{\text{tetra}}$ :

```
[52]: V = PC(Q_tetra_poly).is_coboundary(certificate=True)[1]; V
```

```
[52]: Poisson cochain (18*x1^2*x2^2 - 18*x1^2*x3^2)*xi1 + (-18*x1^3*x2 - 36*x1^3*x3)*xi2 +
(36*x1^3*x2 + 18*x1^3*x3)*xi3
```

```
[53]: V.differential()
```

```
[53]: Poisson cochain (36*x1^3*x2^2*x3 - 36*x1*x2^3*x3^2 - 36*x1^3*x3^3 +
36*x1*x2*x3^4)*xi1*xi2 + (-36*x1^3*x2^3 + 36*x1*x2^4*x3 + 36*x1^3*x2*x3^2 -
36*x1*x2^2*x3^3)*xi1*xi3 + (36*x1^4*x2^2 - 36*x1^2*x2^3*x3 - 36*x1^4*x3^2 +
36*x1^2*x2*x3^3)*xi2*xi3
```

```
[54]: P.bracket(V.lift()) == Q_tetra
```

```
[54]: True
```

For this particular Poisson bi-vector  $P$ , the tetrahedral flow  $Q_{\text{tetra}}$  is Poisson-trivial. We shall explain the origin of the formula for  $Q_{\text{tetra}}(P)$  in §4.7, we will recall why it works

in §5.1, and we will inspect the (non)triviality of the Poisson 2-cocycle in other cases in Chapters 6, 7, and 8.

In the next chapter, we first return to the subject of star products as we had it in Chapter 1.

# Chapter 3

## Implementation of Formality

First, we recall the construction and properties of the graphs in Kontsevich’s Formality morphism and the respective  $\star$ -product: e.g. Kontsevich graphs and Leibniz graphs. Taken modulo the equivalence relation provided by the graph automorphisms acting on the edges, the directed Formality graphs span vector spaces, so that sums of graphs are well-defined and bases of graphs can be introduced. We recall how these vector spaces –of directed Formality graphs with two types of vertices, aerial ones and sinks– assemble to a bi-colored operad with respect to the graph insertion. When we take the quotient of each vector space modulo the labeling of aerial vertices, we obtain the Kontsevich Formality graph complex FGC; the differential is the graph realization of the Hochschild differential with respect to the usual multiplication. We implement all of the above in software. In §3.4 we write present a high-order expansion  $\star \bmod \bar{o}(\hbar^4)$  of the Kontsevich star product, using Kontsevich’s graphs, and verify the associativity of  $\star \bmod \bar{o}(\hbar^4)$  modulo the Jacobi identity by using Leibniz graphs. To constrain the graph weights even further, we generate the cyclic weight relations [21, Appendix E]. We confirm that all the previously known weights –in particular from the work of Banks–Panzer–Pym [1]– do satisfy all the relations which we produce/generate here explicitly for the first time.

All our presentation in this chapter is accompanied by large data files with graphs, weights and relations. These plain text files are stored at <https://rburing.nl/gcaops/>.

### 3.1 Formality graphs

We recall the construction of graphs which show up in Kontsevich’s proof of the Formality Theorem in [28].

```
[1]: from gcaops.graph.formality_graph import FormalityGraph
     from gcaops.graph.formality_graph_operad import FormalityGraphOperad
     from gcaops.graph.formality_graph_complex import FormalityGraphComplex
```

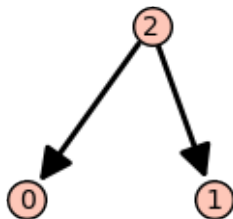
Let us describe the relevant graphs.

**Definition 1.** A *Formality graph* is a directed graph on  $m + n$  vertices  $\{0, \dots, m - 1, m, \dots, m + n - 1\}$  such that the  $m$  *ground* vertices  $0, \dots, m - 1$  are sinks (with no outgoing edges) and the  $n$  vertices  $m, \dots, m + n - 1$  are called *aerial*. The set of edges of the graph is endowed with a total ordering.

Every Formality graph is encoded by the vertex numbers  $(m, n)$  followed by the ordered list of directed edges (represented by the vertex pairs).

**Example 1.** The *wedge* is the main example of a Formality graph; in fact it is a building block in the Kontsevich graphs.

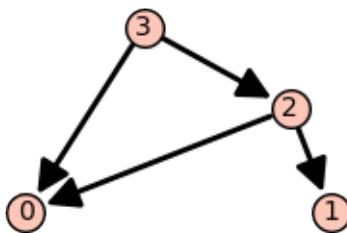
```
[2]: wedge = FormalityGraph(2,1,[(2,0),(2,1)]); wedge.show(figsize=2)
```



**Definition 2.** A *Kontsevich graph* is a Formality graph built of wedges, i.e. with each aerial vertex having exactly two outgoing edges.

**Example 2.** Here is another example of a Kontsevich graph:

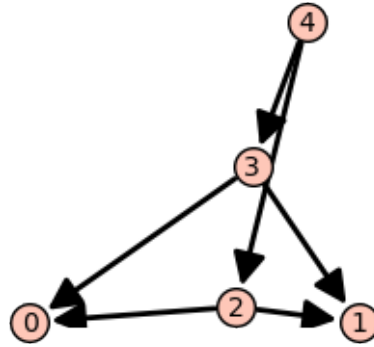
```
[3]: g = FormalityGraph(2,2,[(2,0),(2,1),(3,0),(3,2)]); g.show(figsize=2)
```



**Definition 3.** An *automorphism* of a Formality graph is an automorphism of the directed graph which preserves the ground vertices pointwise. A Formality graph is a *zero graph* if it admits an automorphism that induces an odd permutation on the set of directed edges.

**Example 3.** Zero graphs exist. Here is an example:

```
[4]: g_zero = FormalityGraph(2,3,[(2,0),(2,1),(3,0),(3,1),(4,2),(4,3)]); g_zero.
      ↪ show(figsize=3)
```



Let us relabel the aerial vertices but preserve the ordering of edges, so that we obtain a “different” Kontsevich graph:

```
[5]: g_zero_relabeled = g_zero.relabeled({0:0,1:1,2:3,3:2,4:4}); g_zero_relabeled
```

```
[5]: FormalityGraph(2, 3, [(3, 0), (3, 1), (2, 0), (2, 1), (4, 3), (4, 2)])
```

We repeat: the ordering of the edges is the old one (same as in `g_zero`); every directed edge is encoded by an ordered pair of labeled vertices; the labeling of aerial vertices in `g_zero_relabeled` is new with respect to the original labeling in `g_zero`; this is why the ordering of edges in `g_zero_relabeled` is no longer lexicographic.

The method `canonicalize_edges` (re)orders the edges lexicographically –that is, it modifies the graph– and returns the sign of the permutation needed to reach that new order.

```
[6]: g_zero_relabeled.canonicalize_edges()
```

```
[6]: -1
```

That is, a parity odd permutation was needed to reorder the edges lexicographically.

Now, the graphs `g_zero` and `g_zero_relabeled` coincide identically:

```
[7]: g_zero_relabeled
```

```
[7]: FormalityGraph(2, 3, [(2, 0), (2, 1), (3, 0), (3, 1), (4, 2), (4, 3)])
```

```
[8]: g_zero_relabeled == g_zero
```

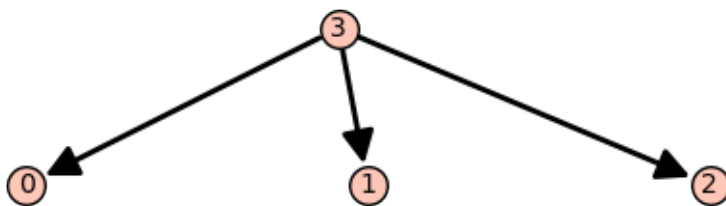
```
[8]: True
```

Originally, `g_zero` and `g_zero_relabeled` had different edge orderings, and it took a parity odd permutation to bring the edge ordering of `g_zero_relabeled` to that of `g_zero`. This shows that `g_zero` is a zero Formality graph.

**Definition 4.** A *Leibniz graph* is a Formality graph built of at least one tripod and further (if at all) built from wedges, i.e. with at least one aerial vertex having exactly three outgoing edges and the other aerial vertices (if any) having exactly two outgoing edges.

**Example 4.** The tripod:

```
[9]: FormalityGraph(3,1,[(3,0),(3,1),(3,2)]).show(figsize=4)
```



## 3.2 The bi-colored operad of Formality graphs

The Formality graphs assemble to a bi-colored operad; see [36] and [16].

**Definition 5.** Consider the tri-graded  $\mathbb{k}$ -vector space with components of grading  $(m, n, e)$  spanned by Formality graphs with  $m$  ground vertices,  $n$  aerial vertices, and  $e$  edges, modulo the relation that a permutation of edges in a graph amounts to multiplication of this graph by the sign of that permutation. We now construct the bi-colored *Formality graph operad* FGO; the two colors are ‘ground’ and ‘air’. The ground component of the operad is the direct sum of all the tri-graded quotient spaces with Formality graphs. The aerial component of the operad contains only the quotient spaces of *vacuum* Formality graphs (without ground vertices). In this section we give the two definitions of the *operadic insertions*: a ground-component graph into a sink of a ground-component graph, and secondly an aerial-component graph into an aerial vertex of a Formality graph from any component.

We begin by discussing the  $\mathbb{k}$ -vector space structure.

```
[10]: FGO = FormalityGraphOperad(QQ); FGO
```

```
[10]: Operad of formality graphs over Rational Field
```

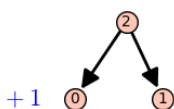
Zero graphs are equal to the zero element in the quotient vector space, because they equal minus themselves:

```
[11]: FGO(g_zero)
```

```
[11]: 0
```

**Example 5.** The wedge is a nonzero element of FGO:

```
[12]: FGO(wedge).show()
```



**Convention.** When a graph is converted into an element of the operad FGO, it is automatically expressed in terms of an internally chosen basis.

The above graph with the reversed ordering of the two edges is equal to minus that graph:

```
[13]: FGO(FormalityGraph(2,1,[(2,1),(2,0)]))
```

```
[13]: (-1)*FormalityGraph(2, 1, [(2, 0), (2, 1)])
```

This does not make the wedge “equal to minus itself”, because swapping the ground vertices does not amount to a Formality graph automorphism (in such an automorphism, the ground vertices must always be preserved pointwise).

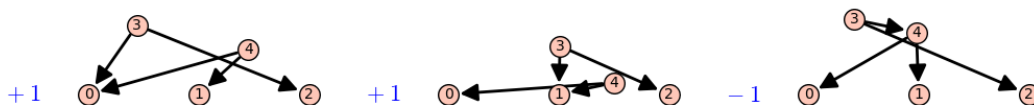
We can insert a graph into a vertex of another graph: into a ground vertex (first case) and into an aerial vertex (second case).

**Definition 6** (graph insertion into a ground vertex). Let  $\Gamma_1$  be a Formality graph with a ground vertex  $\ell$ , and let  $\Gamma_2$  be another Formality graph with  $\lambda \geq 0$  ground vertices. We define the right-into-left *insertion*  $\Gamma_1 \circ_\ell \Gamma_2$  of  $\Gamma_2$  into the  $\ell$ th ground vertex of  $\Gamma_1$ . By definition, the result is a sum of graphs in FGO. In each term, the sink  $\ell$  of  $\Gamma_1$  is replaced by the whole inserted graph  $\Gamma_2$  (with shifted labels), every edge originally coming into that sink  $\ell$  becomes aimed —consecutively, over all vertices of  $\Gamma_2$ — at a vertex (aerial or sink) of  $\Gamma_2$ ; appointing a target for one incoming edge is completely independent from choosing a target for any other incoming edge.

- In every term, the labeling of sinks becomes as follows: first go the sinks  $0, \dots, \ell - 1$  of  $\Gamma_1$ , preceding the sink  $\ell$  into which a whole new graph  $\Gamma_2$  is inserted, then follow the sinks of the inserted graph  $\Gamma_2$  (their labels are shifted by  $+\ell$ ), and finally the remaining sinks (if any) of the graph  $\Gamma_1$  follow in their original order (their labels shifted by  $+\lambda - 1$ ).
- Likewise, in every term the labeling of aerial vertices becomes as follows: all the aerial vertices of  $\Gamma_1$  go first (their labels may be shifted by the count of ground vertices, as above, if  $\lambda > 0$ ), followed by all the aerial vertices of  $\Gamma_2$  (again, their labeling is possibly shifted); in either case the consecutive ordering of aerial vertices is not interrupted.
- Finally, the edge ordering: first go all the edges of the graph  $\Gamma_1$ , last go all the edges of the graph  $\Gamma_2$ ; neither of the two orderings of the edges is anywhere broken.

**Example 6.** Let the wedge (with its own ordering of the edges) be inserted into the leftmost ground vertex 0 of another wedge (with its own edge ordering):

```
[14]: FGO(wedge).insertion(0, FGO(wedge)).show()
```





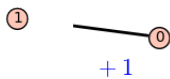
By construction, the Left  $\prec$  Right edges of the wedge whose top is vertex 3 precede the Left  $\prec$  Right edges of the wedge over sinks 0, 1 and top in vertex 4 in every term. The internal choice of the basis by the software is such that the third graph acquires the coefficient  $-1$  with respect to that basis. Indeed, the ordering of edges in basic graphs is lexicographic.

**Definition 7** (graph insertion into an aerial vertex). Into aerial vertices, *only* vacuum Formality graphs (without ground vertices) can be inserted. Let  $\Gamma_1$  be a Formality graph with an aerial vertex  $k$ , and let  $\Gamma_2$  be a vacuum Formality graph on  $\kappa$  aerial vertices. Let us define the right-into-left *insertion*  $\Gamma_1 \circ_k \Gamma_2$  of  $\Gamma_2$  into the  $k$ th aerial vertex of  $\Gamma_1$ . By definition, the result is a sum of graphs in FGO. In each term, the aerial vertex  $k$  of  $\Gamma_1$  is replaced by the whole inserted graph  $\Gamma_2$  (with shifted labels of aerial vertices). Every edge originally incident to that aerial vertex  $k$  becomes incident —consecutively over vertices of  $\Gamma_2$ , which all are aerial— to an aerial vertex of  $\Gamma_2$ ; appointing a new source or target for one such edge is completely independent from choosing a source (respectively, target) for any other such edges.

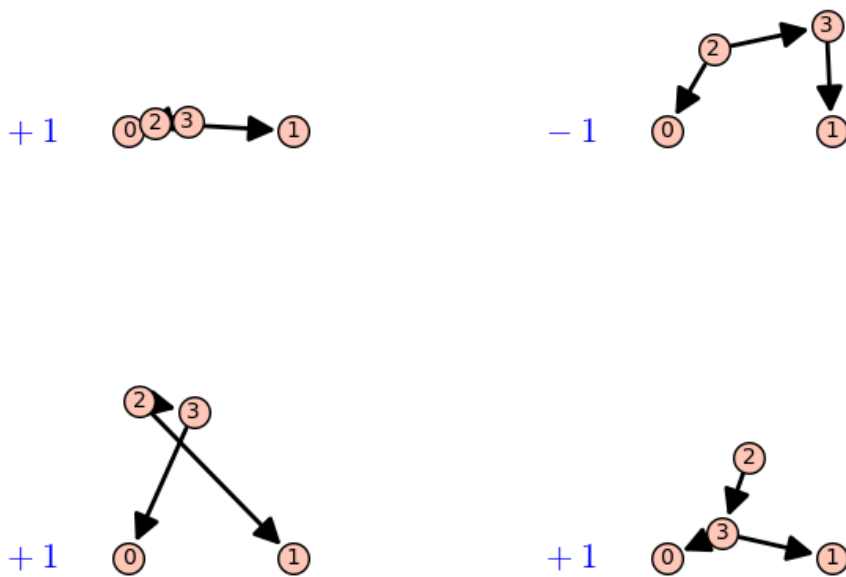
- In every term, the labeling of sinks is preserved from the labeling of sinks in  $\Gamma_1$ , but the labeling of aerial vertices becomes as follows: first go the aerial vertices up to and including (if any)  $k - 1$  preceding the aerial vertex  $k$  into which the whole new graph  $\Gamma_2$  is inserted, then follow the aerial vertices of the inserted graph  $\Gamma_2$  (their labels are shifted by  $+k$ ), and finally go the remaining aerial vertices (if any) of the graph  $\Gamma_1$ : they follow in their original ordering (now, their labels are shifted by  $+\kappa - 1$ ).
- About the edge ordering: first go all the edges of the graph  $\Gamma_1$ , last go all the edges of the graph  $\Gamma_2$ ; neither of the two orderings of the edges is anywhere changed.

**Example 7.** Insertion of a directed stick graph into the aerial vertex of a wedge:

```
[15]: right_stick = FGO(FormalityGraph(0,2,[(0,1)])); right_stick.show()
```



```
[16]: FGO(wedge).insertion(2, right_stick).show(ncols=2)
```



Definitions 6 and 7 are extended to sums of graphs by linearity in the respective component. We claim that both operadic insertions respect the quotient vector space structure (e.g. insertion of a zero graph into any graph produces zero graphs, and the other way around); the proof is similar to [33].

**Remark.** Note that aerial vertices are still distinguishable by their labels in the operad FGO:

```
[17]: sum_of_graphs = FGO([(1, FormalityGraph(3,2,[(3,0),(3,1),(4,1),(4,2)])), (1,
  ↪FormalityGraph(3,2,[(4,0),(4,1),(3,1),(3,2])]])
sum_of_graphs.show()
```



The output confirms that the two graphs are linearly independent (otherwise, the sum would have simplified): by definition, the labeling of the (aerial) vertices is part of the data. In the above two graphs, vertices 3 and 4 stand on different pairs of sinks.

### 3.3 Formality graph complex

By taking the quotient modulo aerial vertex labeling, we obtain a graph complex. With the graphs originally by Kontsevich, this graphical construction was studied by Willwacher in [36] and by Dolgushev in [16].

**Definition 8.** First, from now on we restrict to the ground (non-aerial) component of the Formality graph operad FGO. Next, we quotient every vector space in this ground component modulo aerial vertex labeling. This collection of quotient vector spaces forms a differential graded Lie algebra (dgLa), with the graphical Gerstenhaber bracket and with the graphical Hochschild differential (see below). On this differential graded Lie algebra we thus obtain the structure of *Formality graph complex* FGC.

```
[18]: FGC = FormalityGraphComplex(QQ); FGC
```

```
[18]: Formality graph complex over Rational Field with Basis consisting of representatives of isomorphism classes of formality graphs with no automorphisms that induce an odd permutation on edges
```

By having constructed FGC, we obtain an object which represents a basis of nonzero Formality graphs:

```
[19]: FGC.basis()
```

```
[19]: Basis consisting of representatives of isomorphism classes of formality graphs with no automorphisms that induce an odd permutation on edges
```

The software is now able to generate a basis of directed graphs for the tri-graded  $(m, n, e)$  homogeneity component with a given number  $m$  of ground vertices,  $n$  aerial vertices, and  $e$  edges. Those bases are automatically created by using the `nauty` programs `geng`, `directg` and `pickg` to generate the respective isomorphism classes of directed graphs, followed by permutations of the ground vertices and filtering out zero graphs.

The graph bases are important for us, e.g. because the cyclic weight relations for the weights of Formality graphs will be explicitly referred to those bases, i.e. to the ordered lists of basic graphs with ordered sets of edges.

**Example 8** (wedge). In the basis at tri-grading  $(m, n, e) = (2, 1, 2)$  there is one graph:

```
[20]: list(FGC.basis().graphs(2,1,2))
```

```
[20]: [FormalityGraph(2, 1, [(2, 0), (2, 1)])]
```

So the list of graphs in the basis consists of just one line: the wedge itself.

**Example 9.**

```
[21]: list(FGC.basis().graphs(1,2,4))
```

```
[21]: [FormalityGraph(1, 2, [(1, 0), (1, 2), (2, 0), (2, 1)])]
```

This graph was used in Kontsevich's breakthrough paper [28] to construct the first example of a universal gauge transformation for Kontsevich's star-product.

**Example 10.**

```
[22]: list(FGC.basis().graphs(2,2,4))
```

```
[22]: [FormalityGraph(2, 2, [(2, 1), (2, 3), (3, 0), (3, 1)]),
      FormalityGraph(2, 2, [(2, 0), (2, 3), (3, 0), (3, 1)]),
      FormalityGraph(2, 2, [(2, 1), (3, 0), (3, 1), (3, 2)]),
      FormalityGraph(2, 2, [(2, 0), (3, 0), (3, 1), (3, 2)]),
      FormalityGraph(2, 2, [(2, 0), (2, 1), (3, 0), (3, 1)]),
      FormalityGraph(2, 2, [(2, 3), (3, 0), (3, 1), (3, 2)]),
      FormalityGraph(2, 2, [(2, 1), (2, 3), (3, 0), (3, 2)]),
      FormalityGraph(2, 2, [(2, 1), (2, 3), (3, 1), (3, 2)]),
      FormalityGraph(2, 2, [(2, 0), (2, 3), (3, 0), (3, 2)])]
```

Not all of these Formality graphs on 2 sinks, 2 aerial vertices and 4 edges are built of wedges. Indeed, three of them contain a tripod vertex — and they are *not* Leibniz graphs!

**Convention.** Whenever a Formality graph is converted into an element of FGC, it is automatically expressed in terms of the internally chosen basis. This also defines our *normal form* for Kontsevich graphs and Leibniz graphs.

For example:

```
[23]: FGC(FormalityGraph(3,2,[(3,0),(3,1),(4,1),(4,2)]))
```

```
[23]: 1*FormalityGraph(3, 2, [(3, 1), (3, 2), (4, 0), (4, 1)])
```

In the above, aerial vertices were relabeled ( $3 \rightleftharpoons 4$ ) and pairs of outgoing edges were reordered.

```
[24]: FGC(FormalityGraph(3,2,[(3,1),(3,2),(4,1),(4,0)]))
```

```
[24]: (-1)*FormalityGraph(3, 2, [(3, 1), (3, 2), (4, 0), (4, 1)])
```

This time, the ordering of outgoing edges at aerial vertex 4 is swapped, at a price of the coefficient  $-1$  appearing in front of the basic graph.

**Remark.** In `gcaops` the graph bases are generated by using the `nauty` software, as mentioned above. In [9] (§1.1, Definition 2) a different convention was used to generate graph bases (only containing graphs with wedges): it refers to minimal base- $(m+n)$  integer numbers with respect to all permutations of labels for  $n$  aerial vertices and reordering of Left  $\prec$  Right outgoing edges in the  $n$  pairs making a Kontsevich graph.

**Example 11.** In cell [23] just above, our input is the minimal encoding of a Kontsevich graph, whereas its internal storage by `gcaops`, as seen from the output in cell [23], is *not* in the *minimal* base- $(n+m)$  form.

**Corrigendum.** Normal forms for Leibniz graphs with one Jacobiator were introduced in [10] (Definition 5): the idea was to re-use the normal form for Kontsevich graphs. Namely, the Jacobiator is expanded into the sum of three Kontsevich graphs (built of wedges), all the incoming arrows (to the top of the tripod) are formally directed to the top of the lower wedge in each Kontsevich graph, and then we find the normal forms of the resulting three Kontsevich graphs, while also remembering where the internal edge of the Jacobiator is located in those normal forms. The normal form of the Leibniz graph then was: choose the minimal (w.r.t. base- $(m+n)$  numbers) Kontsevich graph encoding, supplemented with the indication of the internal Jacobiator edge. This definition, i.e. the pair (Kontsevich graph, marked edge) is unfortunately not a true normal form of

the Leibniz graph. Namely, it can happen that the resulting Kontsevich graph has an automorphism that maps the marked edge elsewhere, to a new place in the graph. Consequently, two isomorphic Leibniz graphs could have different “normal forms” (differing only by the marking where the internal Jacobiator edge is). This led to a visible pathology, namely to redundant parameters in the systems of equations: one and the same Leibniz graph, encoded differently, acquired two unrelated coefficients. Fortunately, the effect disappeared when Leibniz graphs were expanded to sums of Kontsevich graphs and similar terms were collected. (Besides, it is necessary to pay attention to whether the internal Jacobiator edge is labeled Left or Right, in order to expand the Leibniz graph with the correct sign  $\pm 1$ .) In consequence, that normal form was abandoned in favor of inambiguous (and fast) description of Leibniz graphs by using the `nauty` software.

Examples of data files containing Formality graphs can be found at the following locations:

- In Appendix B there is the encoding of Kontsevich’s  $\star$ -product modulo  $\bar{o}(\hbar^4)$ .
- In Appendix C there is the encoding of Kontsevich’s affine  $\star$ -product modulo  $\bar{o}(\hbar^7)$ .
- In Appendix D there are the encodings of flows  $Q_\gamma$  built of Kontsevich graphs and operators  $\diamond_\gamma$  built of Leibniz graphs.
- In §3.6 there are links to files with bases of graphs, with weights of graphs, and with linear cyclic weight relations referred to those graph bases.

## 3.4 Kontsevich star product

In this section we construct a fourth order expansion  $\star \bmod \bar{o}(\hbar^4)$  of Kontsevich’s star-product, first with undetermined coefficients and then with coefficients as determined by Panzer’s `kontsevint` software from the joint paper [1] by Banks, Panzer, Pym (2018).

### 3.4.1 Multiplicity

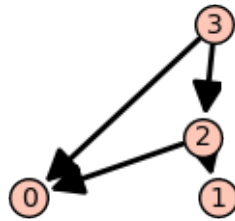
In this section we will take sums over *all* Formality graphs of a certain type (namely, Kontsevich graphs and Leibniz graphs). We keep in mind that every Formality graph is a topological combinatorial structure (a graph) equipped with a global ordering of edges (in particular, ordering of outgoing edges at every aerial vertex); every Formality graph is taken, in the quotient vector space `FGC`, modulo labeling of aerial vertices. To make this process —of taking sums of graphs— more efficient, we can instead sum over isomorphism classes of Formality graphs, with multiplicities.

**Definition.** The *multiplicity*  $m(\Gamma)$  of a Formality graph  $\Gamma$  is the number of Formality graphs isomorphic to  $\Gamma$ , under permutations of aerial vertex labels (leaving the ground vertices invariant) and reorderings of the list of edges issued from each particular vertex.

The count of aerial vertex relabelings yields the number  $n! / \#\text{Aut}(\Gamma)$ , i.e. not merely  $n!$  for a Formality graph  $\Gamma$  on  $n$  aerial vertices and with a non-trivial automorphism group  $\text{Aut}(\Gamma)$ .

**Example 1.** Recall the graph `g`:

```
[1]: from gcaops.graph.formality_graph import FormalityGraph
     g = FormalityGraph(2,2,[(2,0),(2,1),(3,0),(3,2)]); g.show(figsize=2)
```



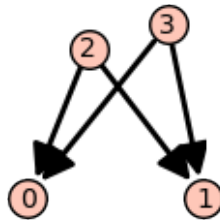
```
[2]: g.multiplicity()
```

```
[2]: 8
```

Indeed, for the graph  $\mathbf{g}$  there are two ways to (re)label the internal vertices and  $2 \cdot 2 = 4$  ways to (re)order the edges. The number of options is therefore equal to  $2 \cdot 4 = 8$ .

### Example 2.

```
[3]: wedgewedge = FormalityGraph(2,2,[(2,0),(2,1),(3,0),(3,1)]); wedgewedge.show(figsize=2)
```



```
[4]: wedgewedge.multiplicity()
```

```
[4]: 4
```

There are  $2 \cdot 2 = 4$  ways to (re)order the edges; there are 2 ways,  $\{\text{id}, 2 \rightleftharpoons 3\}$ , to (re)label the aerial vertices, but the swap  $2 \rightleftharpoons 3$  does not produce any different Formality graph because the swap is a symmetry of `wedgewedge`, so  $2! / \#\text{Aut}(\Gamma) = 1$  (here  $n = 2$ ).

### 3.4.2 Star product

The non-commutative associative star-product [28] of two functions  $f, g \in C^\infty(M)$  is realized by a sum over Kontsevich graphs,

$$f \star g = f \cdot g + \sum_{n \geq 1} \frac{\hbar^n}{n!} \sum_{\Gamma \in \hat{\mathcal{G}}_2^n} w(\Gamma) \Gamma(P, \dots, P)(f, g).$$

The Kontsevich formula for the weight of a Formality graph  $\Gamma$  on  $m \geq 0$  ground vertices and  $n \geq 0$  aerial vertices is:

$$w(\Gamma) := \frac{1}{(2\pi)^{2n+m-2}} \int_{\tilde{C}_{n,m}^+} \bigwedge_{e \in E(\Gamma)} d\varphi_e.$$

The wedge product of angle 1-forms  $d\varphi_e$  is integrated over the compactified configuration space  $\tilde{C}_{n,m}^+$  of  $n$  points inside the hyperbolic plane  $\mathbb{H}^2$  and  $m$  strictly ordered sinks on  $\partial\mathbb{H}^2 = \mathbb{R}$ , modulo the affine group action. That construction is given in [28].

**Remark.** The weights in Kontsevich's original construction from [28] are defined using a different convention,

$$W_\Gamma = \left( \prod_{k=1}^n \frac{1}{(\#\text{Star}(k))!} \right) w(\Gamma),$$

where  $\#\text{Star}(k)$  is the number of arrows starting at the aerial vertex  $m + k - 1$ .

The **weight** function in the `kontsevint` software by Panzer [1] computes the weights  $w(\Gamma)$ .

The  $n$ -linear polydifferential operator  $\Gamma(P, \dots, P)$  and the weight  $w(\Gamma)$  behave under permutations of edges and of aerial vertex labels in such a way that the product  $w(\Gamma) \Gamma(P, \dots, P)$  is invariant under both types of permutations. Hence, the star-product can be expressed in a more computationally efficient way as a sum over graphs with multiplicities, i.e.

$$f \star g = f \cdot g + \sum_{n \geq 1} \frac{\hbar^n}{n!} \sum_{[\Gamma]} m(\Gamma) w(\Gamma) \Gamma(P, \dots, P)(f, g),$$

where  $m(\Gamma)$  is the multiplicity defined above, and the sum now runs over equivalence classes  $[\Gamma]$  of Formality graphs in  $\hat{G}_2^n$  modulo labeling of aerial vertices and ordering of edges.

```
[5]: from gcaops.graph.formality_graph_complex import FormalityGraphComplex
#FGC = FormalityGraphComplex(SR, implementation='vector'); FGC
FGC = FormalityGraphComplex(SR, lazy=True); FGC
```

[5]: Formality graph complex over Symbolic Ring with Basis consisting of representatives of isomorphism classes of formality graphs with no automorphisms that induce an odd permutation on edges

**Example.** Let us convert the known expansion  $\star \bmod \bar{o}(\hbar^3)$ , given as a list of encodings of Kontsevich graphs (in terms of ordered pairs of target vertices for edges [9]), into the `gcaops` format of weighted Formality graphs.

```
[6]: star3 = FGC.element_from_kgs_encoding("""h^0:
2 0 1      1
h^1:
2 1 1    0 1    1
h^2:
2 2 1    0 3 1 2    -1/6
2 2 1    0 1 1 2    -1/3
2 2 1    0 1 0 2    1/3
2 2 1    0 1 0 1    1/2
h^3:"")
```



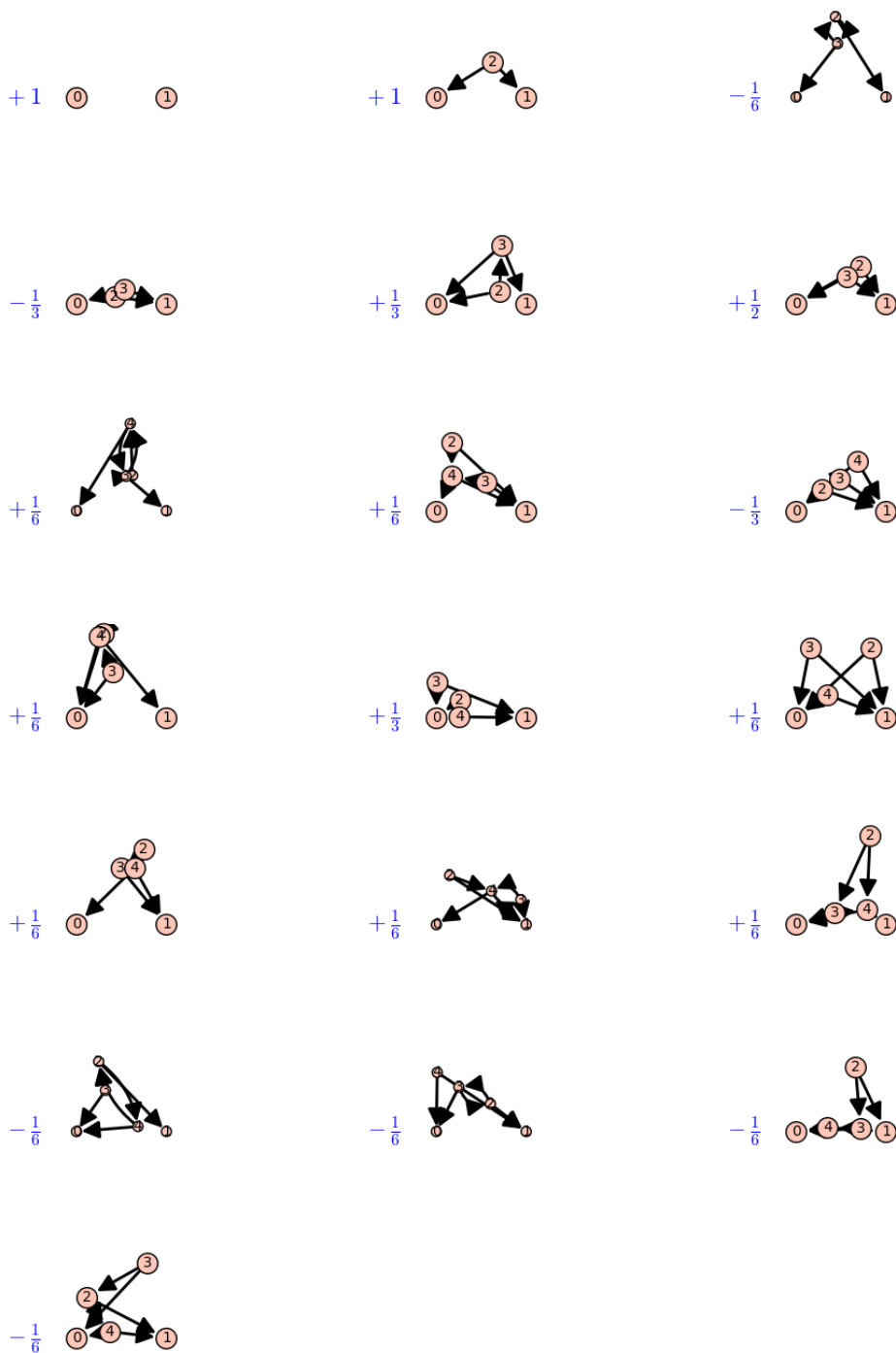
```

2 3 1  0 3 1 2 2 3  -1/6
2 3 1  0 1 1 2 1 2   1/6
2 3 1  0 1 0 1 1 2  -1/3
2 3 1  0 1 0 2 0 2   1/6
2 3 1  0 1 0 1 0 2   1/3
2 3 1  0 1 0 1 0 1   1/6
2 3 1  0 1 1 2 2 3  -1/6
2 3 1  0 3 1 2 1 2   1/6
2 3 1  0 1 0 2 2 3  -1/6
2 3 1  0 3 1 2 0 3  -1/6
2 3 1  0 1 0 4 1 3  -1/6
2 3 1  0 1 0 2 1 3  -1/6
2 3 1  0 1 0 4 1 2  -1/6""")

```

Let us see how this graph expansion looks like; another drawing of this formula for  $\star \bmod \bar{o}(\hbar^3)$  is given in Figure 1 of Chapter 11.

```
[7]: star3.show()
```



First we construct the associator  $(f \star g) \star h - f \star (g \star h)$  modulo  $\bar{o}(\hbar^3)$  for the star product expansion  $\star \bmod \bar{o}(\hbar^3)$  and for  $f, g, h \in C^\infty(M)$ .

```
[8]: %time star3_assoc = star3.insertion(0, star3, max_num_aerial=3) - star3.insertion(1,
↳ star3, max_num_aerial=3)
```

CPU times: user 258 ms, sys: 7.76 ms, total: 266 ms  
Wall time: 264 ms

Let us inspect the leading order term in the associator.

```
[9]: star3_assoc.homogeneous_part(3, 0, 0)
```

```
[9]: 0
```

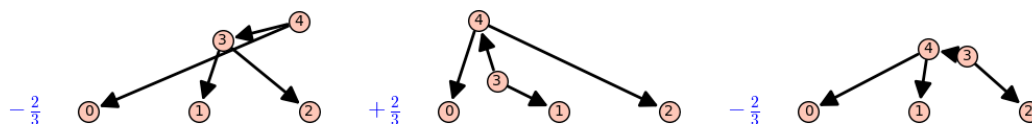
So, the coefficient of  $\hbar^0 = 1$  vanishes. Next, let us inspect the leading deformation term at  $\hbar^1$  in the associator.

```
[10]: star3_assoc.homogeneous_part(3, 1, 2)
```

```
[10]: 0
```

In fact, the associator nontrivially starts at the order two in its  $\hbar$ -expansion.

```
[11]: star3_assoc.homogeneous_part(3, 2, 4).show()
```



Obviously, this is the Jacobiator appearing with nonzero coefficient  $2/3$ . This tells us at once that for the  $\star$ -product to be associative, the Jacobi identity (for the bi-vector  $P$  whose copies are placed in the aerial vertices) is a *necessary* condition.

At the order  $\hbar^3$  we have more Formality graphs in the associator:

```
[12]: len(star3_assoc.homogeneous_part(3, 3, 6))
```

```
[12]: 39
```

These 39 graphs are discussed in full detail in Chapter 10; see also Chapter 12.

```
[13]: #star3_assoc.homogeneous_part(3, 3, 6).show()
```

The associativity of the  $\star$ -product can be expressed as the equation  $\frac{1}{2}[\star, \star]_G = 0$ , where the bracket is the Gerstenhaber bracket on the space of polydifferential operators on  $C^\infty(M)[[\hbar]]$ .

```
[14]: star3_assoc == (1/2)*star3.gerstenhaber_bracket(star3, max_num_aerial=3)
```

```
[14]: True
```

Equivalently, the associativity of  $\star = \mu + B$ , where  $\mu$  is the ordinary multiplication in  $C^\infty(M)[[\hbar]]$ , is the Maurer–Cartan equation  $d_H(B) + \frac{1}{2}[B, B]_G = 0$  for the deformation tail  $B$ .

```
[15]: mu = FGC(FormalityGraph(2,0,[])); mu.show()
```

+1 0 1

We recall that by construction, the content  $f, g$  of the respective sinks 0, 1 is here not differentiated (indeed, there are no edges), and then the usual product of the sinks' content is taken with coefficient +1, thus giving  $f \cdot g$ .

```
[16]: B = star3 - mu
```

```
[17]: star3_assoc == B.hochschild_differential() + (1/2)*B.gerstenhaber_bracket(B,
↳max_num_aerial=3)
```

```
[17]: True
```

### 3.4.3 Star product from Formality morphism

Now, by using the `gcaops` software, let us *construct* a third order expansion of Kontsevich's star product.

We take for granted the known expansion of  $\star \bmod \bar{o}(\hbar^2)$ : each coefficient is obtained by direct integration of Kontsevich's formula (see Appendix A.1 in Chapter 11).

```
[18]: star2 = FGC(FormalityGraph(2,0,[])) + FGC(FormalityGraph(2,1,[(2,0),(2,1)])) + \
FGC([(1/2, FormalityGraph(2,2,[(2,0),(2,1),(3,0),(3,1)])), (1/3,
↳FormalityGraph(2,2,[(2,0),(2,1),(3,0),(3,2)])), (-1/3,
↳FormalityGraph(2,2,[(2,0),(2,1),(3,1),(3,2)])), (-1/6,
↳FormalityGraph(2,2,[(2,0),(2,3),(3,1),(3,2)]))]); star2
```

```
[18]: 1*FormalityGraph(2, 0, []) + 1*FormalityGraph(2, 1, [(2, 0), (2, 1)]) +
(1/2)*FormalityGraph(2, 2, [(2, 0), (2, 1), (3, 0), (3, 1)]) + (1/3)*FormalityGraph(2,
2, [(2, 0), (2, 3), (3, 0), (3, 1)]) + (-1/3)*FormalityGraph(2, 2, [(2, 1), (2, 3),
(3, 0), (3, 1)]) + (-1/6)*FormalityGraph(2, 2, [(2, 1), (2, 3), (3, 0), (3, 2)])
```

**Example.** By evaluating  $\star \bmod \bar{o}(\hbar^2)$  at the Poisson structure  $\{x, y\} = xy/2$  on  $\mathbb{R}^2$  we reproduce the polydifferential operator from Example 2 in Section 1.2.

```
[19]: from gcaops.algebra.superfunction_algebra import SuperfunctionAlgebra
from gcaops.algebra.polydifferential_operator import PolyDifferentialOperatorAlgebra
SA.<xi1,xi2> = SuperfunctionAlgebra(SR, var('x,y'))
PA.<ddx,ddy> = PolyDifferentialOperatorAlgebra(SR, var('x,y'))
```

```
[20]: from gcaops.graph.formality_graph_operator import FormalityGraphOperator
      star2_operator = FormalityGraphOperator(SA, PA, star2)
```

```
[21]: P = (x*y/2)*xi1*xi2; P
```

```
[21]: (1/2*x*y)*xi1*xi2
```

```
[22]: star2_op = star2_operator.value_at_copies_of(var('hbar')*P); star2_op
```

```
[22]: (1/8*hbar^2*x^2*y^2)*(ddx^2 ⊗ ddy^2) + (id ⊗ id) + (-1/4*hbar^2*x^2*y^2)*(ddx*ddy ⊗
      ddx*ddy) + (1/8*hbar^2*x^2*y^2)*(ddy^2 ⊗ ddx^2) + (1/12*hbar^2*x^2*y^2)*(ddx^2 ⊗ ddy) +
      (-1/12*hbar^2*x^2*y^2)*(ddx*ddy ⊗ ddx) + (-1/12*hbar^2*x*y^2)*(ddx*ddy ⊗ ddy) +
      (1/12*hbar^2*x*y^2)*(ddy^2 ⊗ ddx) + (-1/12*hbar^2*x^2*y^2)*(ddx ⊗ ddx*ddy) +
      (1/12*hbar^2*x^2*y^2)*(ddy ⊗ ddx^2) + (1/12*hbar^2*x*y^2)*(ddx ⊗ ddy^2) +
      (-1/12*hbar^2*x*y^2)*(ddy ⊗ ddx*ddy) + (-1/24*hbar^2*x^2)*(ddx ⊗ ddx) +
      (1/24*hbar^2*x*y - 1/2*hbar*x*y)*(ddy ⊗ ddx) + (1/24*hbar^2*x*y + 1/2*hbar*x*y)*(ddx ⊗
      ddy) + (-1/24*hbar^2*y^2)*(ddy ⊗ ddy)
```

This is literally the formula in Cell [36] in Section 1.3.

Next, we add Kontsevich graphs with undetermined coefficients at  $\hbar^3$  to the already known  $\star \bmod \bar{o}(\hbar^2)$  — again, with a generic Poisson structure  $P$ .

```
[23]: from gcaops.graph.formality_graph_basis import KontsevichGraphBasis
      KGB = KontsevichGraphBasis(positive_differential_order=True)
```

```
[24]: star3c = star2 + 1/6*FGC([(g.multiplicity()*var('c{}'.format(k)), g) for (k,g) in
      ↪ enumerate(KGB.graphs(2,3))]); star3c
```

```
[24]: 1*FormalityGraph(2, 0, []) + 1*FormalityGraph(2, 1, [(2, 0), (2, 1)]) +
      (1/2)*FormalityGraph(2, 2, [(2, 0), (2, 1), (3, 0), (3, 1)]) + (1/3)*FormalityGraph(2,
      2, [(2, 0), (2, 3), (3, 0), (3, 1)]) + (-1/3)*FormalityGraph(2, 2, [(2, 1), (2, 3),
      (3, 0), (3, 1)]) + (-1/6)*FormalityGraph(2, 2, [(2, 1), (2, 3), (3, 0), (3, 2)]) +
      (4*c0)*FormalityGraph(2, 3, [(2, 1), (2, 4), (3, 1), (3, 4), (4, 0), (4, 1)]) +
      (4*c1)*FormalityGraph(2, 3, [(2, 0), (2, 4), (3, 0), (3, 4), (4, 0), (4, 1)]) +
      (8*c2)*FormalityGraph(2, 3, [(2, 3), (2, 4), (3, 0), (3, 4), (4, 0), (4, 1)]) +
      (8*c3)*FormalityGraph(2, 3, [(2, 3), (2, 4), (3, 1), (3, 4), (4, 0), (4, 1)]) +
      (4/3*c4)*FormalityGraph(2, 3, [(2, 0), (2, 1), (3, 0), (3, 1), (4, 0), (4, 1)]) +
      (8*c5)*FormalityGraph(2, 3, [(2, 1), (2, 4), (3, 0), (3, 4), (4, 0), (4, 1)]) +
      (8*c6)*FormalityGraph(2, 3, [(2, 1), (2, 3), (3, 0), (3, 4), (4, 0), (4, 1)]) +
      (8*c7)*FormalityGraph(2, 3, [(2, 1), (2, 4), (3, 0), (3, 2), (4, 0), (4, 1)]) +
      (8*c8)*FormalityGraph(2, 3, [(2, 1), (2, 4), (3, 0), (3, 1), (4, 0), (4, 1)]) +
      (8*c9)*FormalityGraph(2, 3, [(2, 0), (2, 4), (3, 0), (3, 1), (4, 0), (4, 1)]) +
      (8*c10)*FormalityGraph(2, 3, [(2, 1), (2, 3), (3, 1), (3, 4), (4, 0), (4, 2)]) +
      (8*c11)*FormalityGraph(2, 3, [(2, 1), (2, 3), (3, 0), (3, 4), (4, 0), (4, 2)]) +
      (8*c12)*FormalityGraph(2, 3, [(2, 1), (2, 3), (3, 1), (3, 4), (4, 0), (4, 1)]) +
      (8*c13)*FormalityGraph(2, 3, [(2, 0), (2, 3), (3, 0), (3, 4), (4, 0), (4, 1)]) +
      (4*c14)*FormalityGraph(2, 3, [(2, 3), (2, 4), (3, 2), (3, 4), (4, 0), (4, 1)]) +
      (8*c15)*FormalityGraph(2, 3, [(2, 1), (2, 4), (3, 1), (3, 4), (4, 0), (4, 3)]) +
      (8*c16)*FormalityGraph(2, 3, [(2, 1), (2, 4), (3, 0), (3, 2), (4, 0), (4, 2)]) +
      (8*c17)*FormalityGraph(2, 3, [(2, 3), (2, 4), (3, 0), (3, 2), (4, 0), (4, 1)]) +
      (8*c18)*FormalityGraph(2, 3, [(2, 3), (2, 4), (3, 1), (3, 2), (4, 0), (4, 1)]) +
      (8*c19)*FormalityGraph(2, 3, [(2, 3), (2, 4), (3, 1), (3, 4), (4, 0), (4, 3)]) +
      (8*c20)*FormalityGraph(2, 3, [(2, 3), (2, 4), (3, 1), (3, 4), (4, 0), (4, 2)]) +
      (8*c21)*FormalityGraph(2, 3, [(2, 3), (2, 4), (3, 1), (3, 2), (4, 0), (4, 3)]) +
      (8*c22)*FormalityGraph(2, 3, [(2, 1), (2, 4), (3, 0), (3, 4), (4, 0), (4, 2)]) +
```

```
(8*c23)*FormalityGraph(2, 3, [(2, 1), (2, 3), (3, 1), (3, 4), (4, 0), (4, 3)]) +
(8*c24)*FormalityGraph(2, 3, [(2, 1), (2, 4), (3, 0), (3, 4), (4, 0), (4, 3)]) +
(8*c25)*FormalityGraph(2, 3, [(2, 1), (2, 3), (3, 1), (3, 2), (4, 0), (4, 3)]) +
(4*c26)*FormalityGraph(2, 3, [(2, 1), (2, 3), (3, 1), (3, 2), (4, 0), (4, 1)]) +
(4*c27)*FormalityGraph(2, 3, [(2, 0), (2, 3), (3, 0), (3, 2), (4, 0), (4, 1)]) +
(8*c28)*FormalityGraph(2, 3, [(2, 1), (2, 3), (3, 0), (3, 2), (4, 0), (4, 1)]) +
(8*c29)*FormalityGraph(2, 3, [(2, 3), (2, 4), (3, 1), (3, 2), (4, 0), (4, 2)])
```

This time, we take the associator modulo  $\bar{o}(\hbar^3)$  for the ansatz  $\star \bmod \bar{o}(\hbar^3)$  and insert the 3D Nambu–Poisson structure into the associator. The term at  $\hbar^2$  must vanish because the Jacobi identity holds for the Nambu–Poisson structure. There remains only the term near  $\hbar^3$ : it is differential polynomial in  $a$  and  $\rho$  (inside  $P$ ) and it is linear with respect to the constants  $c_i$  in the top-degree ansatz. That term must vanish for  $\star$  to be associative. We solve the arising system of linear algebraic equations with respect to the top-degree coefficients (of Kontsevich graphs at  $\hbar^3$ ).

```
[25]: %%time assoc3c = star3c.insertion(0, star3c, max_num_aerial=3) - star3c.insertion(1,
      ↪star3c, max_num_aerial=3)
```

CPU times: user 405 ms, sys: 0 ns, total: 405 ms

Wall time: 404 ms

```
[26]: from gcaops.algebra.differential_polynomial_ring import DifferentialPolynomialRing
D3 = DifferentialPolynomialRing(SR, ('rho', 'a'), ('x', 'y', 'z'),
      ↪max_differential_orders=[5,6])
rho, a = D3.fibre_variables()
SA3 = SuperfunctionAlgebra(D3, D3.base_variables())
xi1, xi2, xi3 = SA3.odd_coordinates()
x, y, z = SA3.even_coordinates()
PA3 = PolyDifferentialOperatorAlgebra(D3, D3.base_variables())
P3 = rho*(xi1*xi2*diff(a,z) + xi2*xi3*diff(a,x) + xi3*xi1*diff(a,y)); P3
```

```
[26]: (rho*a_z)*xi0*xi1 + (rho*a_x)*xi1*xi2 + (-rho*a_y)*xi0*xi2
```

```
[27]: assoc3c3 = assoc3c.homogeneous_part(3, 3, 6)
```

```
[28]: %%time
linear_system = []
for diff_order in assoc3c3.differential_orders():
    print(diff_order)
    assoc3c3_operator = FormalityGraphOperator(SA3, PA3, assoc3c3.
      ↪part_of_differential_order(diff_order))
    assoc3c3_op = assoc3c3_operator.value_at_copies_of(P3)
    assoc3c3_op_coeffs_diffpoly = [assoc3c3_op[m] for m in assoc3c3_op.
      ↪multi_indices()]
    assoc3c3_op_coeffs_consts = sum([diffpoly.coefficients() for diffpoly in
      ↪assoc3c3_op_coeffs_diffpoly], [])
    linear_system.extend(list(set(assoc3c3_op_coeffs_consts)))
```

(2, 1, 1)

(1, 2, 1)

(1, 1, 1)

(2, 1, 3)

```
(1, 2, 3)
(1, 1, 2)
(1, 1, 3)
(2, 1, 2)
(1, 2, 2)
(3, 1, 2)
(3, 1, 1)
(2, 2, 1)
(3, 2, 1)
CPU times: user 1min 37s, sys: 14.6 ms, total: 1min 37s
Wall time: 1min 37s
```

```
[29]: c = [var('c{}'.format(k)) for k in range(len(KGB.graphs(2,3)))]
```

```
[30]: solve(linear_system, c)
```

```
[30]: [[c0 == (1/24), c1 == (1/24), c2 == r5, c3 == r3, c4 == (1/8), c5 == 0, c6 == -r2 -
1/24, c7 == r2, c8 == (-1/24), c9 == (1/24), c10 == r4, c11 == r4, c12 == 0, c13 == 0,
c14 == r8, c15 == -r3 + r5 + r6 + 1/48, c16 == 2*r5 + r6 - 1/16, c17 == r5 - 1/48, c18
== r3 - 1/48, c19 == r7, c20 == r10, c21 == r9, c22 == r5 + r6 - 1/48, c23 == r5 + r6
- 1/48, c24 == r6, c25 == r3 + r5 + r6 - 1/24, c26 == 0, c27 == 0, c28 == (-1/48), c29
== r1]]
```

So, there remain only 10 parameters not yet constrained by the postulate of associativity for the 3D Nambu–Poisson structure (some of them will never be constrained by associativity alone, due to the gauge freedom).

We can obtain the missing values either by direct integration (see Appendix A.1 in Chapter 11), or by importing these values from [1], or by using all the methods in Chapter 11 (specifically, in Example 26) to find the few missing values.

```
[31]: c_values = [1/24, 1/24, 1/48, 1/48, 1/8, 0, -1/48, -1/48, -1/24, 1/24, 0, 0, 0, 0, 0,
↪1/48, -1/48, 0, 0, 1/48, 0, 0, 0, 0, 0, 0, 0, -1/48, 0]
c_subs = dict(zip(c, c_values))
```

This results in the cubic expansion  $\star \bmod \bar{o}(\hbar^3)$  as seen from Cell [6], [7] in Section 3.4.2.

```
[32]: star3c_substituted = star3c.map_coefficients(lambda d: d.subs(c_subs))
```

```
[33]: star3c_substituted == star3
```

```
[33]: True
```

**Remark.** The associativity of  $\star \bmod \bar{o}(\hbar^3)$  has been established (with these values of coefficients in the top-degree ansatz) *only* for a restriction of  $\star$  to a particular (however, large) class of Poisson brackets. We claim nevertheless that the built star-product is associative modulo  $\bar{o}(\hbar^3)$  for *generic* Poisson structures in any finite dimension. Let us demonstrate this in the next section.

### 3.4.4 Star product associativity via Leibniz graphs

Now we consider at once Kontsevich’s star product  $\star \bmod \bar{o}(\hbar^4)$ : we have it from Chapter 11.

```
[34]: star4_txt = open('data/star4.txt').read().rstrip()
      star4 = FGC.element_from_kgs_encoding(star4_txt) #; star4
      %time star4_assoc = star4.insertion(0, star4, max_num_aerial=4) - star4.insertion(1,
      ↪star4, max_num_aerial=4)
```

CPU times: user 3.03 s, sys: 4 ms, total: 3.03 s

Wall time: 3.03 s

In what follows, the associator of  $\star$  is expressed as the sum of Leibniz graphs,

$$\text{Assoc}(\star(P))(f, g, h) = \sum_{n \geq 1} \frac{\hbar^{n+1}}{(n+1)!} \text{coeff}(n) \sum_{[L] \in L_3^n} m(L) w(L) L(P, \dots, P, \llbracket P, P \rrbracket)(f, g, h).$$

The weights  $w(L)$  are calculated by using the `kontsevint` software by Panzer [1].

To show the factorization  $\text{Assoc}(\star(P)) = \diamond(P, \llbracket P, P \rrbracket) \bmod \bar{o}(\hbar^4)$  for generic Poisson brackets  $P$  we must expand Leibniz graphs in  $\diamond$  in the right-hand side to Kontsevich graphs. Leibniz graphs contain tripod vertices (and possibly wedges), and Kontsevich graphs are built of wedges: expanding the Jacobiator  $\frac{1}{2}\llbracket P, P \rrbracket$  amounts to inserting the stick (edge) graph into the trident vertex. In effect, we can insert the stick graph into *every* aerial vertex of  $L$  and then select only those graphs which are built of wedges.

```
[35]: stick = FGC(FormalityGraph(0,2,[(0,1)])); stick
```

```
[35]: 1*FormalityGraph(0, 2, [(0, 1)])
```

```
[36]: from gcaops.graph.formality_graph_basis import LeibnizGraphBasis
      LGB = LeibnizGraphBasis(positive_differential_order=True); LGB
```

```
[36]: Basis consisting of representatives of isomorphism classes of Leibniz graphs (of
      positive differential order) with no automorphisms that induce an odd permutation on
      edges
```

We start with order  $\hbar^2$  in the associator. The list  $L_3^1$  of Leibniz graphs on 3 sinks and 1 aerial vertex amounts to the tripod itself:

```
[37]: l31 = list(LGB.graphs(3,1)); l31
```

```
[37]: [FormalityGraph(3, 1, [(3, 0), (3, 1), (3, 2)])]
```

We will use a helper function to import the graph weights:

```
[38]: def vector_from_file(filename):
      with open(filename) as f:
          return vector(sage_eval('{}'.format(','.join(f.readlines()))))
```

The list of weights  $w(L)$  (here, consisting of just one number) is this:

```
[39]: wl31 = vector_from_file('data/weights_leibniz_3_1.txt'); wl31
```

```
[39]: (1/6)
```

We are able to present the sum of Leibniz graphs to balance the associator for  $\star$  at  $\hbar^2$  for generic Poisson structure  $P$ :



```
[40]: L31 = -2/3 * FGC([(c*g.multiplicity(),g) for (c,g) in zip(wl31,l31)]); L31 #.show()
```

```
[40]: (-2/3)*FormalityGraph(3, 1, [(3, 0), (3, 1), (3, 2)])
```

As said, the Leibniz graphs must now be expanded into Kontsevich graphs:

```
[41]: L31_expanded = L31.insertion(3,stick,max_out_degree=2)
```

And they balance the order  $\hbar^2$  in the associator:

```
[42]: L31_expanded == star4_assoc.homogeneous_part(3, 2, 4)
```

```
[42]: True
```

- Next order:  $\hbar^3$  in the associator and Leibniz graphs on 3 sinks and 2 aerial vertices.

```
[43]: l32 = list(LGB.graphs(3,2))
wl32 = vector_from_file('data/weights_leibniz_3_2.txt') #; wl32
L32 = -1/3 * FGC([(c*g.multiplicity(),g) for (c,g) in zip(wl32,l32)]); L32 #.show()
```

```
[43]: (1/3)*FormalityGraph(3, 2, [(3, 2), (3, 4), (4, 0), (4, 1), (4, 2)]) +
(-1/3)*FormalityGraph(3, 2, [(3, 0), (3, 4), (4, 0), (4, 1), (4, 2)]) +
(-2/3)*FormalityGraph(3, 2, [(3, 0), (3, 1), (4, 0), (4, 1), (4, 2)]) +
(-2/3)*FormalityGraph(3, 2, [(3, 0), (3, 2), (4, 0), (4, 1), (4, 2)]) +
(-2/3)*FormalityGraph(3, 2, [(3, 1), (3, 2), (4, 0), (4, 1), (4, 2)]) +
(-1/6)*FormalityGraph(3, 2, [(3, 0), (3, 2), (4, 0), (4, 1), (4, 3)]) +
(1/6)*FormalityGraph(3, 2, [(3, 0), (3, 2), (3, 4), (4, 0), (4, 1)]) +
(-1/3)*FormalityGraph(3, 2, [(3, 1), (3, 2), (4, 0), (4, 1), (4, 3)]) +
(-1/3)*FormalityGraph(3, 2, [(3, 1), (3, 2), (3, 4), (4, 0), (4, 1)]) +
(1/6)*FormalityGraph(3, 2, [(3, 1), (3, 2), (4, 0), (4, 2), (4, 3)]) +
(-1/6)*FormalityGraph(3, 2, [(3, 1), (3, 2), (3, 4), (4, 0), (4, 2)]) +
(1/6)*FormalityGraph(3, 2, [(3, 2), (3, 4), (4, 0), (4, 1), (4, 3)]) +
(-1/6)*FormalityGraph(3, 2, [(3, 1), (3, 2), (3, 4), (4, 0), (4, 3)])
```

The number of Leibniz graphs with nonzero coefficients is 13:

```
[44]: len(L32)
```

```
[44]: 13
```

```
[45]: L32_expanded = sum(L32.insertion(k,stick,max_out_degree=2) for k in [3,4])
```

These are precisely the 39 Kontsevich graphs at order  $\hbar^3$  in the associator of  $\star$  for generic Poisson structure.

```
[46]: len(L32_expanded)
```

```
[46]: 39
```

```
[47]: L32_expanded == star4_assoc.homogeneous_part(3, 3, 6)
```

```
[47]: True
```

The claim of associator's factorization modulo  $\bar{o}(\hbar^3)$  is thus established.

Let us proceed to the next, fourth, order!

As usual, we generate the list of Leibniz graphs, import their weights  $w(L)$ , and take the (weighted) sum with multiplicities and with an overall leading coefficient  $\text{coeff}(3) = -8/3$ .

```
[48]: l33 = list(LGB.graphs(3,3))
```

```
[49]: wl33 = vector_from_file('data/weights_leibniz_3_3.txt') #; wl33
```

```
[50]: len(wl33) - list(wl33).count(0)
```

```
[50]: 241
```

```
[51]: L33 = -1/9 * FGC([(c*g.multiplicity(),g) for (c,g) in zip(wl33,l33)])
```

The number of Leibniz graphs with nonzero coefficients is 241:

```
[52]: len(L33)
```

```
[52]: 241
```

Leibniz graphs are now expanded into Kontsevich's graphs built of wedges:

```
[53]: L33_expanded = sum(L33.insertion(k,stick,max_out_degree=2) for k in [3,4,5])
```

```
[54]: len(L33_expanded), len(star4_assoc.homogeneous_part(3, 4, 8))
```

```
[54]: (740, 740)
```

The two lengths of lists match, and moreover: these are the same sums of Kontsevich graphs with the same coefficients!

```
[55]: L33_expanded == star4_assoc.homogeneous_part(3, 4, 8)
```

```
[55]: True
```

This proves that the star product expansion  $\star \bmod \bar{o}(\hbar^4)$  is associative modulo  $\bar{o}(\hbar^4)$  for generic Poisson structure.

**Example.** Let us inspect the order  $(3, 3, 1)$  in the  $\star$ -product associator in particular:

```
[56]: star4_assoc.homogeneous_part(3, 4, 8).part_of_differential_order((3,3,1))
```

```
[56]: (-1/3)*FormalityGraph(3, 4, [(3, 1), (3, 2), (4, 0), (4, 3), (5, 0), (5, 1), (6, 0),
(6, 1)]) + (1/3)*FormalityGraph(3, 4, [(3, 1), (3, 4), (4, 0), (4, 2), (5, 0), (5, 1),
(6, 0), (6, 1)]) + (-1/3)*FormalityGraph(3, 4, [(3, 2), (3, 6), (4, 0), (4, 1), (5,
0), (5, 1), (6, 0), (6, 1)])
```

It is realized by the expansion of a single Leibniz graph:

```
[57]: L_331 = l33[114]; L_331
```

```
[57]: FormalityGraph(3, 3, [(3, 0), (3, 1), (4, 0), (4, 1), (5, 0), (5, 1), (5, 2)])
```

```
[58]: -1/9 * wl33[114] * L_331.multiplicity() * sum(FGC(L_331).
↪insertion(k,stick,max_out_degree=2) for k in [3,4,5])
```

```
[58]: (-1/3)*FormalityGraph(3, 4, [(3, 1), (3, 2), (4, 0), (4, 3), (5, 0), (5, 1), (6, 0),
(6, 1)]) + (1/3)*FormalityGraph(3, 4, [(3, 1), (3, 4), (4, 0), (4, 2), (5, 0), (5, 1),
(6, 0), (6, 1)]) + (-1/3)*FormalityGraph(3, 4, [(3, 2), (3, 6), (4, 0), (4, 1), (5,
0), (5, 1), (6, 0), (6, 1)])
```

---

In the next section we will express the associator for Kontsevich's star-product  $\star \bmod \bar{o}(\hbar^6)$  as a sum of Leibniz graphs with *some* coefficients. The coefficients we will obtain there are not necessarily (and probably not) those guaranteed by the Formality theorem (which we had in this section), yet any such coefficients do suffice to prove the associativity up to  $\bar{o}(\hbar^6)$ .

## 3.5 Leibniz graph expansion and factorization(s)

In this section we discuss two further aspects of Leibniz graphs: the iterative production of Leibniz graphs from Kontsevich graphs, and the (non)uniqueness of Leibniz graph factorizations.

### 3.5.1 Iterative production of Leibniz graphs

Some sums of Kontsevich graphs amount to the zero polydifferential operator, when evaluated at copies of a Poisson structure. This is the case e.g. for the associator  $(f \star g) \star h - f \star (g \star h)$  of Kontsevich's  $\star$ -product. To prove that a sum of Kontsevich graphs built of wedges amounts to zero when evaluated at copies of a Poisson structure, it suffices to realize it as (the expansion of) a sum of Leibniz graphs. Here we consider the problem of finding the (potentially) necessary Leibniz graphs by using an iterative process; see Chapter 14.

**Example.** First we import Kontsevich's  $\star \bmod \bar{o}(\hbar^4)$ :

```
[1]: from gcaops.graph.formality_graph_complex import FormalityGraphComplex
FGC = FormalityGraphComplex(SR, lazy=True)
```

```
[2]: star4 = FGC.element_from_kgs_encoding(open('data/star4.txt').read().rstrip())
```

We calculate the associator:

```
[3]: star4_assoc = star4.insertion(0, star4, max_num_aerial=4) - star4.insertion(1, star4,
↳max_num_aerial=4)
```

```
[4]: star4_assoc.homogeneous_part(3, 2, 4)
```

```
[4]: (-2/3)*FormalityGraph(3, 2, [(3, 1), (3, 2), (4, 0), (4, 3)]) +
(2/3)*FormalityGraph(3, 2, [(3, 1), (3, 4), (4, 0), (4, 2)]) +
(-2/3)*FormalityGraph(3, 2, [(3, 2), (3, 4), (4, 0), (4, 1)])
```

So at  $\hbar^2$  in the associator there are three Kontsevich graphs which themselves assimilate to the Jacobiator on the associator's three sinks. This was known from the seminal works of [2] and [28]; we discuss the third order expansion of  $\text{Assoc}(\star) \bmod \bar{o}(\hbar^3)$  in Chapter 10.

```
[5]: star4_assoc4 = star4_assoc.homogeneous_part(3, 4, 8)
```

We generate the necessary Leibniz graphs from the Kontsevich graphs at  $\hbar^4$ , i.e. built of 4 wedges over 3 sinks, by contraction of one internal edge in every such Kontsevich graph:

```
[6]: from gcaops.graph.leibniz_graph_expansion import _
      ↪_kontsevich_graph_sum_to_leibniz_graphs
      leibniz_graphs = []
      _kontsevich_graph_sum_to_leibniz_graphs(star4_assoc4, leibniz_graphs)
      len(leibniz_graphs)
```

```
[6]: 274
```

Referring to Chapter 12 below, we claim that these 274 Leibniz graphs, but not their neighbors (obtained by contracting one edge between aerial vertices in each Kontsevich graph in the expansion of those Leibniz graphs), are already enough to factorize the respective part of the associator.

For optimization purposes, we can split the problem into many (here 23) smaller parts of fixed differential orders (in-degrees of ground vertices):

```
[7]: len(list(star4_assoc4.differential_orders()))
```

```
[7]: 23
```

In general we can expand the found Leibniz graphs to Kontsevich graphs and contract one edge (in all possible ways) in the resulting Kontsevich graphs. At each iterative step, we form the linear system with the left-hand side containing (with one undetermined coefficient for each Leibniz graph) the Kontsevich graph expansion of the Leibniz graphs, and the right-hand side containing the Kontsevich graphs with their coefficients from the input (here, the associator). If there is a solution to the linear system, we are happy and we return it. If there is no solution to the linear system, then we either go to the next step (if still new Leibniz graphs can be found by contracting edges in Kontsevich graphs in the left-hand side) or report that there is no solution.

The following lines of output contain the grading of a tri-differential component of the associator at  $\hbar^4$  for  $\star \bmod \bar{o}(\hbar^4)$ , followed by the number of Kontsevich graphs in that component, followed by the number of new Leibniz graphs and new Kontsevich graphs obtained in each step. As soon as there appears a solution, the program reports writes **True** about the existence of some factorization in that tri-differential order.

```
[8]: from gcaops.graph.leibniz_graph_expansion import _
      ↪kontsevich_graph_sum_to_leibniz_graph_sum
      from gcaops.graph.leibniz_graph_expansion import _
      ↪leibniz_graph_sum_to_kontsevich_graph_sum
      for diff_order in star4_assoc4.differential_orders():
          print(diff_order, end=': ')
          part = star4_assoc4.part_of_differential_order(diff_order)
          part_leibniz = kontsevich_graph_sum_to_leibniz_graph_sum(part, verbose=True)
          print(leibniz_graph_sum_to_kontsevich_graph_sum(part_leibniz) == part)
```

```
(3, 1, 3): 3K -> +1L -> +0K
True
```

```

(2, 2, 3): 3K -> +1L -> +0K
True
(1, 3, 3): 3K -> +1L -> +0K
True
(3, 1, 2): 12K -> +4L -> +0K
True
(2, 2, 2): 26K -> +12L -> +7K
True
(1, 3, 2): 7K -> +4L -> +5K
True
(2, 1, 3): 12K -> +4L -> +0K
True
(1, 2, 3): 12K -> +4L -> +0K
True
(3, 1, 1): 26K -> +9L -> +1K
True
(2, 2, 1): 53K -> +24L -> +6K
True
(1, 3, 1): 26K -> +9L -> +1K
True
(1, 1, 3): 26K -> +9L -> +1K
True
(2, 1, 2): 55K -> +24L -> +4K
True
(1, 2, 2): 53K -> +24L -> +6K
True
(2, 1, 1): 94K -> +35L -> +7K
True
(1, 2, 1): 90K -> +35L -> +11K
True
(1, 1, 1): 117K -> +28L -> +6K
True
(1, 1, 2): 94K -> +35L -> +7K
True
(3, 2, 1): 12K -> +4L -> +0K
True
(3, 2, 2): 3K -> +1L -> +0K
True
(2, 3, 2): 3K -> +1L -> +0K
True
(2, 3, 1): 7K -> +4L -> +5K
True
(3, 3, 1): 3K -> +1L -> +0K
True

```

The number of actually used layers of Leibniz graphs can be read from the number of times the count of Leibniz graphs is printed: single time appearance means that the 0th layer is enough. Indeed we observe that the 0th layer of Leibniz graphs is enough to find a factorization in this case of  $\text{Assoc}(\star) \bmod \bar{o}(\hbar^4)$ .

**Example.** We repeat the above example for Kontsevich's  $\star \bmod \bar{o}(\hbar^5)$ , known from [1].

```
[9]: star5 = FGC.element_from_kgs_encoding(open('data/star5.txt').read().rstrip())
```

```
[10]: star5_assoc = star5.insertion(0, star5, max_num_aerial=5) - star5.insertion(1, star5,
↪max_num_aerial=5)
```

```
[11]: star5_assoc5 = star5_assoc.homogeneous_part(3, 5, 10)
```

The Leibniz graph factorization problem splits at  $\hbar^5$  into 54 tri-differential parts within the associator.

```
[12]: len(list(star5_assoc5.differential_orders()))
```

```
[12]: 54
```

For every tri-differential order, we solve each Leibniz graph factorization problem iteratively over the layers of Leibniz graphs:

```
[13]: for diff_order in star5_assoc5.differential_orders():
    print(diff_order, end=': ')
    part = star5_assoc5.part_of_differential_order(diff_order)
    part_leibniz = kontsevich_graph_sum_to_leibniz_graph_sum(part, verbose=True)
    print(leibniz_graph_sum_to_kontsevich_graph_sum(part_leibniz) == part)
```

```
(3, 1, 4): 12K -> +4L -> +0K
True
(2, 2, 4): 15K -> +5L -> +0K
True
(2, 1, 4): 38K -> +13L -> +1K
True
(1, 3, 4): 12K -> +4L -> +0K
True
(1, 2, 4): 38K -> +13L -> +1K
True
(1, 1, 4): 74K -> +27L -> +7K
True
(4, 2, 3): 3K -> +1L -> +0K
True
(4, 1, 3): 12K -> +4L -> +0K
True
(3, 3, 3): 3K -> +1L -> +0K
True
(3, 2, 3): 32K -> +14L -> +7K
True
(3, 1, 3): 79K -> +35L -> +4K
True
(2, 4, 3): 3K -> +1L -> +0K
True
(2, 3, 3): 32K -> +14L -> +7K
True
(2, 2, 3): 165K -> +79L -> +15K
True
(2, 1, 3): 316K -> +151L -> +23K
True
(1, 4, 3): 7K -> +4L -> +5K
True
(1, 3, 3): 72K -> +35L -> +11K
True
(1, 2, 3): 317K -> +151L -> +22K
True
(1, 1, 3): 486K -> +223L -> +64K
True
(4, 1, 4): 3K -> +1L -> +0K
```

True  
(3, 2, 4): 3K -> +1L -> +0K  
True  
(2, 3, 4): 3K -> +1L -> +0K  
True  
(1, 4, 4): 3K -> +1L -> +0K  
True  
(4, 2, 2): 15K -> +5L -> +0K  
True  
(4, 1, 2): 38K -> +13L -> +1K  
True  
(3, 3, 2): 32K -> +14L -> +7K  
True  
(3, 2, 2): 165K -> +79L -> +15K  
True  
(3, 1, 2): 316K -> +151L -> +23K  
True  
(2, 3, 2): 166K -> +78L -> +14K  
True  
(2, 2, 2): 638K -> +340L -> +76K  
True  
(2, 1, 2): 958K -> +483L -> +116K  
True  
(1, 4, 2): 33K -> +13L -> +6K  
True  
(1, 3, 2): 296K -> +151L -> +43K  
True  
(1, 2, 2): 964K -> +481L -> +108K  
True  
(1, 1, 2): 1213K -> +530L -> +201K  
True  
(2, 4, 2): 10K -> +5L -> +5K  
True  
(4, 2, 1): 38K -> +13L -> +1K  
True  
(4, 1, 1): 74K -> +27L -> +7K  
True  
(3, 3, 1): 72K -> +35L -> +11K  
True  
(3, 2, 1): 317K -> +151L -> +22K  
True  
(3, 1, 1): 486K -> +223L -> +64K  
True  
(2, 4, 1): 33K -> +13L -> +6K  
True  
(2, 3, 1): 296K -> +151L -> +43K  
True  
(2, 2, 1): 964K -> +481L -> +108K  
True  
(2, 1, 1): 1213K -> +530L -> +201K  
True  
(1, 4, 1): 54K -> +27L -> +27K  
True  
(1, 3, 1): 355K -> +200L -> +170K  
True  
(1, 2, 1): 934K -> +462L -> +415K  
True  
(1, 1, 1): 1028K -> +400L -> +449K

```

True
(4, 3, 1): 12K -> +4L -> +0K
True
(4, 3, 2): 3K -> +1L -> +0K
True
(3, 4, 2): 3K -> +1L -> +0K
True
(3, 4, 1): 7K -> +4L -> +5K
True
(4, 4, 1): 3K -> +1L -> +0K
True

```

Again we observe that the 0th layer of Leibniz graphs is enough to find a factorization in this case of  $\text{Assoc}(\star) \bmod \bar{o}(\hbar^5)$

**Example.** We import Kontsevich’s tetrahedral flow from the future Section 5.1.

```

[14]: from gcaops.graph.formality_graph import FormalityGraph
Q_tetra = FGC([(-1, FormalityGraph(2, 4, [(2, 4), (2, 5), (3, 2), (3, 5), (4, 3), (4, ↵
↵5), (5, 0), (5, 1)])),
              (-3, FormalityGraph(2, 4, [(2, 3), (2, 5), (3, 4), (3, 5), (4, 1), (4, ↵
↵2), (5, 0), (5, 4)])),
              (-3, FormalityGraph(2, 4, [(2, 3), (2, 4), (3, 4), (3, 5), (4, 1), (4, ↵
↵5), (5, 0), (5, 2)])))]

```

We calculate its Schouten bracket with the wedge graph:

```

[15]: wedge = FGC(FormalityGraph(2,1,[(2,0),(2,1)]))

```

```

[16]: P_Q_tetra = wedge.schouten_bracket(Q_tetra)

```

```

[17]: len(P_Q_tetra)

```

```

[17]: 39

```

This time, let us repeat the iterations until saturation when no new Leibniz graphs are produced any longer:

```

[18]: kontsevich_graph_sum_to_leibniz_graph_sum(P_Q_tetra, force_saturation=True, ↵
↵verbose=True);

```

```

39K -> +46L -> +306K -> +209L -> +570K -> +138L -> +459K -> +67L -> +189K -> +3L ->
+12K

```

This iterative count means the following: starting with a given sum of 39 Kontsevich’s graphs in  $\llbracket P, Q \rrbracket$  and contracting one internal edge in each graph, we obtain 46 Leibniz graphs in the (initial) 0th layer. Expanding them into Kontsevich’s graphs we do obtain 306 new such graphs. Repeating the contraction and expansion procedure we then generate 209, 138, etc. new Leibniz graphs in every next layer of neighbors — until the saturation, when no new Leibniz graphs are produced. By construction, the factorization of  $\llbracket P, Q \rrbracket$  via Leibniz graphs cannot refer to any Leibniz graphs other than the ones which have been produced by the iterative algorithm. For more details, we refer to Chapter 14.



**Remark.** In Section 5.1 below, we illustrate another method (by Kontsevich, 1996) to find all the Leibniz graphs which are sufficient for a factorization of the Poisson cocycle condition,  $\llbracket P, Q \rrbracket = \diamond(P, \llbracket P, P \rrbracket)$ . This is achieved by orienting the tetrahedron over the three sinks such that there are three wedges and one trident; the extra edges (not in the tetrahedron) go to the three ground vertices.

### 3.5.2 Leibniz graph factorization (non)uniqueness

Some sums of Leibniz graphs amount to zero when expanded to Kontsevich graphs; this can lead to non-uniqueness of solutions to the factorization problems. Let us illustrate this effect by examples.

We begin with a count. Leibniz graphs are expanded to sums of Kontsevich graphs by inserting the stick graph into the trivalent vertex. This expansion map may have a nontrivial kernel. We now compute the dimension of the kernel of that map restricted to Leibniz graphs of a given bi-grading.

```
[13]: def leibniz_graph_expansion_nullity(num_ground, num_aerial, skew=False):
    from gcaops.graph.formality_graph_basis import LeibnizGraphBasis
    LGB = LeibnizGraphBasis(positive_differential_order=True,
    ↪mod_ground_permutations=skew)
    from gcaops.graph.formality_graph_basis import KontsevichGraphBasis
    KGB = KontsevichGraphBasis(positive_differential_order=True)
    from gcaops.graph.formality_graph_complex import FormalityGraphComplex
    FGC = FormalityGraphComplex(QQ, lazy=True)
    from gcaops.graph.formality_graph import FormalityGraph
    stick = FGC(FormalityGraph(0,2,[(0,1)]))
    K = KGB.graphs(num_ground, num_aerial+1)
    #print(list(K))
    LL = LGB.graphs(num_ground, num_aerial)
    if skew:
        LL = [L for L in LL if FGC(L).ground_skew_symmetrization() != 0]
    #print(len(K), len(LL))
    M = matrix(QQ, len(K), len(LL), sparse=True)
    for i, L in enumerate(LL):
        L = FGC(L)
        if skew:
            L = L.ground_skew_symmetrization()
        L_expanded = sum(L.insertion(k,stick,max_out_degree=2) for k in
    ↪range(num_ground,num_ground+num_aerial))
        for (c,g) in L_expanded:
            #print(g)
            M[K.index(g), i] = c # NOTE: uses that normal form of graphs in KGB is
    ↪same as in FGC
    return M.right_nullity()
    #for v in M.right_kernel().basis():
    #    if list(v).count(0) != len(v) - 1:
    #        print(v)
    #        break
```

```
[14]: for n in range(1,5):
    print((3, n), '--> nullity', leibniz_graph_expansion_nullity(3, n))
```

```
(3, 1) --> nullity 0
(3, 2) --> nullity 0
```

```
(3, 3) --> nullity 12
(3, 4) --> nullity 538
```

So, e.g. for the Leibniz graphs on 3 sinks and 3 aerial vertices (of which one is a trident and two are wedges), there are twelve linearly independent linear combinations of Leibniz graphs that expand to zero sums of Kontsevich graphs built of wedges.

The bi-grading (3, 4) is relevant for (the Poisson cocycle condition for) Kontsevich's tetrahedral flow. Yet, let us point out, referring to the second example in the preceding section 3.5.1, that many of the identities for the Leibniz graphs are specific about the graphs which did not show up in the iterative process of building the layers of Leibniz graphs that started with  $\llbracket P, Q \rrbracket$  for the tetrahedral flow. That is, the number of solutions  $\diamond$  for the cocycle condition factorization problem which have the property that they are *in the span* of the graphs produced by the iterative process, would be less than 538; we know 2.

**Example.** Here is a sum of nine nonzero Leibniz graphs on 3 sinks and 3 aerial vertices that expands to zero.

```
[15]: from gcaops.graph.formality_graph_basis import LeibnizGraphBasis
      LGB = LeibnizGraphBasis(positive_differential_order=True)
```

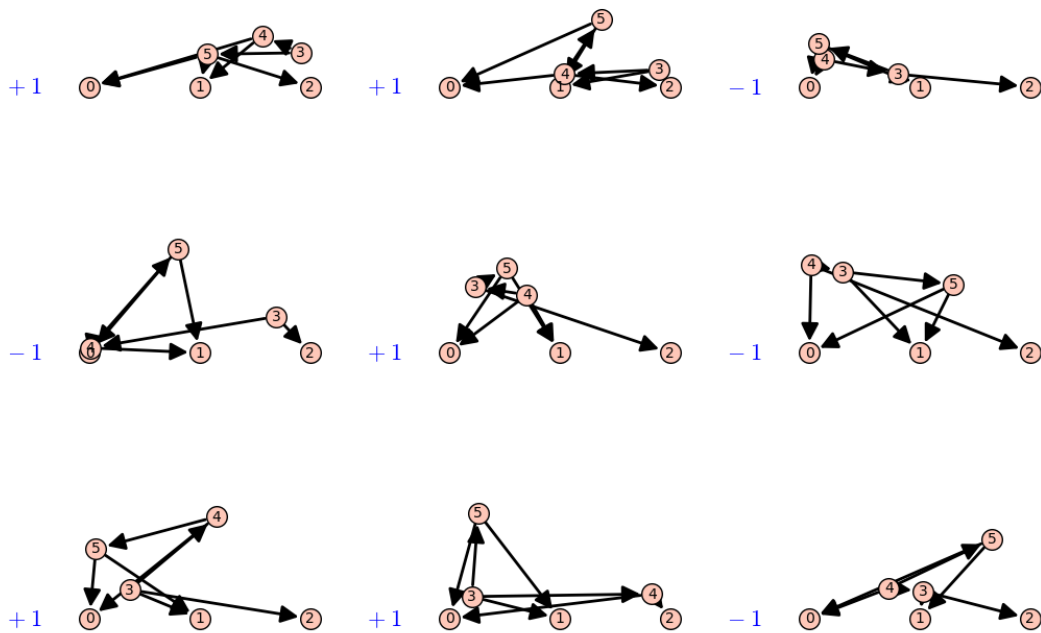
```
[16]: from gcaops.graph.formality_graph import FormalityGraph
      stick = FGC(FormalityGraph(0,2,[(0,1)]))
```

Here is the promised nontrivial linear combination of nine nonzero Leibniz graphs:

```
[17]: mystery = FGC([(s, LGB.graphs(3,3)[k]) for k,s in
      ↪ zip([6,28,29,84,113,159,161,165,167], [1,1,-1,-1,1,-1,1,1,-1])]); mystery
```

```
[17]: 1*FormalityGraph(3, 3, [(3, 4), (3, 5), (4, 0), (4, 1), (5, 0), (5, 1), (5, 2)]) +
      1*FormalityGraph(3, 3, [(3, 1), (3, 4), (4, 0), (4, 2), (4, 5), (5, 0), (5, 1)]) +
      (-1)*FormalityGraph(3, 3, [(3, 1), (3, 2), (3, 5), (4, 0), (4, 3), (5, 0), (5, 1)]) +
      (-1)*FormalityGraph(3, 3, [(3, 2), (3, 4), (4, 0), (4, 1), (4, 5), (5, 0), (5, 1)]) +
      1*FormalityGraph(3, 3, [(3, 2), (3, 5), (4, 0), (4, 1), (4, 3), (5, 0), (5, 1)]) +
      (-1)*FormalityGraph(3, 3, [(3, 1), (3, 5), (4, 0), (4, 2), (4, 3), (5, 0), (5, 1)]) +
      1*FormalityGraph(3, 3, [(3, 1), (3, 2), (3, 4), (4, 0), (4, 5), (5, 0), (5, 1)]) +
      1*FormalityGraph(3, 3, [(3, 1), (3, 4), (3, 5), (4, 0), (4, 2), (5, 0), (5, 1)]) +
      (-1)*FormalityGraph(3, 3, [(3, 1), (3, 2), (4, 0), (4, 3), (4, 5), (5, 0), (5, 1)])
```

```
[18]: mystery.show()
```



```
[19]: sum(mystery.insertion(k,stick,max_out_degree=2) for k in [3,4,5])
```

```
[19]: 0
```

Equivalently, we can expand sums of Leibniz graphs into Kontsevich graphs by using another method:

```
[ ]: #leibniz_graph_sum_to_kontsevich_graph_sum(mystery)
```

**Example.** Here is a sum of 14 skew Leibniz graphs on 2 sinks and 4 aerial vertices, such that its expansion into Kontsevich graphs vanishes.

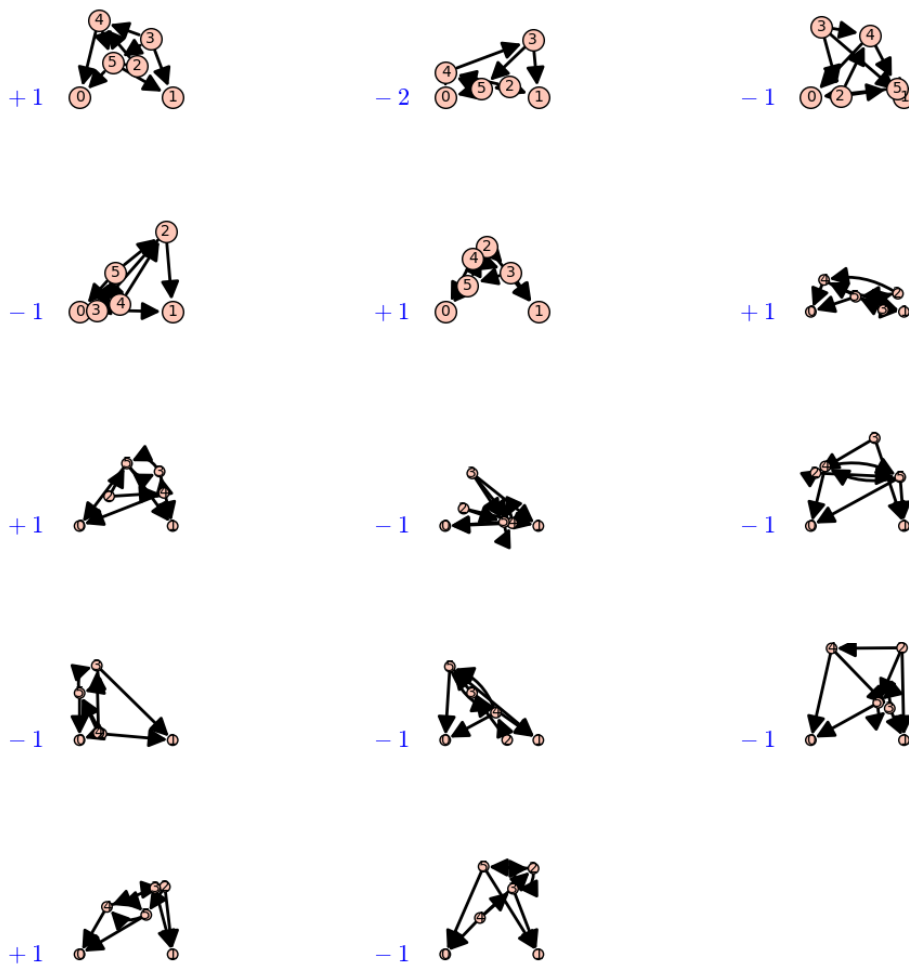
```
[20]: LGB_skew = LeibnizGraphBasis(positive_differential_order=True,
↳ mod_ground_permutations=True)
```

```
[21]: skew_mystery = FGC([(s, LGB_skew.graphs(2,4)[k]) for k,s in
↳ zip([2,52,93,128,129,193,199,328,338,349,422,438,479,484],
↳ [1,-2,-1,-1,1,1,1,-1,-1,-1,-1,-1,1,-1])]); skew_mystery
```

```
[21]: 1*FormalityGraph(2, 4, [(2, 4), (2, 5), (3, 1), (3, 4), (3, 5), (4, 0), (4, 5), (5,
0), (5, 1)]) + (-2)*FormalityGraph(2, 4, [(2, 1), (2, 4), (2, 5), (3, 1), (3, 5), (4,
0), (4, 3), (5, 0), (5, 4)]) + (-1)*FormalityGraph(2, 4, [(2, 4), (2, 5), (3, 1), (3,
2), (3, 4), (4, 0), (4, 5), (5, 0), (5, 1)]) + (-1)*FormalityGraph(2, 4, [(2, 1), (2,
3), (3, 1), (3, 5), (4, 0), (4, 2), (4, 5), (5, 0), (5, 2)]) + 1*FormalityGraph(2, 4,
[(2, 1), (2, 3), (3, 1), (3, 5), (4, 0), (4, 3), (4, 5), (5, 0), (5, 2)]) +
1*FormalityGraph(2, 4, [(2, 4), (2, 5), (3, 1), (3, 4), (3, 5), (4, 0), (4, 2), (5,
0), (5, 1)]) + 1*FormalityGraph(2, 4, [(2, 4), (2, 5), (3, 1), (3, 5), (4, 0), (4, 3),
(5, 0), (5, 1), (5, 3)]) + (-1)*FormalityGraph(2, 4, [(2, 1), (2, 5), (3, 1), (3, 4),
```

```
(3, 5), (4, 0), (4, 5), (5, 0), (5, 4]]) + (-1)*FormalityGraph(2, 4, [(2, 4), (2, 5),
(3, 1), (3, 2), (4, 0), (4, 5), (5, 0), (5, 1), (5, 4)]) + (-1)*FormalityGraph(2, 4,
[(2, 1), (2, 5), (3, 1), (3, 5), (4, 0), (4, 3), (4, 5), (5, 0), (5, 3)]) +
(-1)*FormalityGraph(2, 4, [(2, 3), (2, 4), (3, 1), (3, 5), (4, 0), (4, 5), (5, 0), (5,
1), (5, 4)]) + (-1)*FormalityGraph(2, 4, [(2, 1), (2, 4), (2, 5), (3, 1), (3, 5), (4,
0), (4, 3), (5, 0), (5, 3)]) + 1*FormalityGraph(2, 4, [(2, 1), (2, 3), (2, 5), (3, 1),
(3, 4), (4, 0), (4, 5), (5, 0), (5, 4)]) + (-1)*FormalityGraph(2, 4, [(2, 3), (2, 5),
(3, 1), (3, 2), (4, 0), (4, 2), (4, 3), (5, 0), (5, 1)])
```

```
[22]: skew_mystery.show()
```



When we skew-symmetrize the above sum of Leibniz graphs over the content of two sinks, the result remains nonzero:

```
[23]: skew_mystery.ground_skew_symmetrization() == 0
```

```
[23]: False
```

Finally, let us expand the skew-symmetrized Leibniz graphs into Kontsevich's graphs built of wedges:

```
[24]: leibniz_graph_sum_to_kontsevich_graph_sum(skew_mystery.ground_skew_symmetrization())
```

```
[24]: 0
```

This results in zero, as promised.

On 3 sinks and 4 or 6 aerial vertices (i.e. 3 or 5 wedges and one trident) the same mechanism is responsible for the coexistence of solutions  $\diamond_1$  and  $\diamond_2$  in the factorization problem for the Poisson cocycle condition  $[[P, Q_\gamma(P^{\otimes n})]] \doteq 0$  on  $\text{Jac}(P) = 0$  (here  $\gamma = \gamma_3$  or  $\gamma_5$ , respectively), see Chapters 5, 15, 16 and 17.

## 3.6 Cyclic weight relations

On the basis of previous work by Shoikhet on cyclic formality [20], Willwacher and Felder [21] showed that the weights of graphs in Kontsevich's deformation quantization [28] satisfy a class of linear relations, namely the cyclic weight relations. In this section we obtain the linear algebraic system of cyclic weight relations between the weights of Kontsevich graphs in Kontsevich's star-product  $f \star g \bmod \bar{o}(\hbar^5)$  and between the weights of Leibniz graphs in the associator  $(f \star g) \star h - f \star (g \star h) \bmod \bar{o}(\hbar^5)$ .

### 3.6.1 From a basis to relations

We define a basis for the vector space spanned by Kontsevich graphs built of wedges, that is by the Formality graphs which show up in Kontsevich's star-product  $f \star g$  and its associator  $(f \star g) \star h - f \star (g \star h)$ :

```
[2]: from gcaops.graph.formality_graph_basis import FormalityGraphComplexBasis, □
      ↪KontsevichGraphBasis
      KGB = KontsevichGraphBasis(positive_differential_order=True); KGB
```

[2]: Basis consisting of representatives of isomorphism classes of Kontsevich graphs (of positive differential order) with no automorphisms that induce an odd permutation on edges

Recall that we can then generate a basis of directed graphs for the bi-graded homogeneity component of degree  $(m, n)$  with a given number  $m$  of ground vertices,  $n$  aerial vertices, and  $e = 2n$  edges.

Here is an example:

```
[3]: list(KGB.graphs(2,2))
```

```
[3]: [FormalityGraph(2, 2, [(2, 1), (2, 3), (3, 0), (3, 1)]),
      FormalityGraph(2, 2, [(2, 0), (2, 3), (3, 0), (3, 1)]),
      FormalityGraph(2, 2, [(2, 0), (2, 1), (3, 0), (3, 1)]),
      FormalityGraph(2, 2, [(2, 1), (2, 3), (3, 0), (3, 2)])]
```

Using this basis and treating the graphs as placeholders for their Kontsevich weights, we can generate the linear algebraic system of cyclic weight relations  $\sum_j c_{ij} w(\Gamma_j) = 0$  by

calling the `cyclic_weight_relations` method on the graph basis, passing the bi-grading in the input:

```
[4]: KGB.cyclic_weight_relations(2,2)
```

```
[4]: [-1  2 -1  0]
      [ 1  1  0  0]
      [ 0  0  0  0]
      [ 0 -1  0 -2]
```

By construction, there is one cyclic weight relation per graph (although not all of the relations in the resulting system are necessarily linearly independent). The above method returns the square matrix  $C = (c_{ij})$  in which the cyclic weight relations are encoded in the following way: every row  $(c_{i1}, \dots, c_{ik})$  corresponds to the graph  $\Gamma_i$  that marks the relation; the entries of that row are the coefficients  $c_{ij}$  of weights  $w(\Gamma_j)$  of the graphs  $\Gamma_j$  in the basis at the given bi-grading. Every such linear combination  $\sum_j c_{ij} w(\Gamma_j)$  vanishes (under the default assumption that the list of graphs in the basis is enough to make the relation well-defined). Let us illustrate this construction.

### 3.6.2 Kontsevich graphs in $f \star g$

First we consider the Kontsevich graphs appearing at each order in Kontsevich's star-product, i.e. graphs built of wedges, and having two sinks. We now start producing (and listing) and counting the cyclic weight relations for Kontsevich's graphs on  $n$  aerial vertices, i.e. at  $\hbar^n$  in the  $\star$ -product.

```
[5]: len(list(KGB.graphs(2,1)))
```

```
[5]: 1
```

```
[6]: ck21 = KGB.cyclic_weight_relations(2,1); ck21
```

```
[6]: [0]
```

So, there are no cyclic weight relations other than  $0 = 0$  for Kontsevich's wedge graph at  $\hbar^1$  in the star product.

```
[7]: len(list(KGB.graphs(2,2)))
```

```
[7]: 4
```

We recognize all these four graphs on two sinks and two aerial vertices at  $\hbar^2$  in the star-product expansion, see Figure 1 in Chapter 11. We now establish that the Kontsevich weights of these four graphs are constrained by four cyclic weight relations, of which three are linearly independent, and the remaining one is a tautology: the linear combination of weights with zero coefficients equals zero.

```
[8]: ck22 = KGB.cyclic_weight_relations(2,2); ck22
```

```
[8]: [-1  2 -1  0]
      [ 1  1  0  0]
      [ 0  0  0  0]
      [ 0 -1  0 -2]
```

The tautology in the third line is produced by the Moyal graph at  $\hbar^2$  in  $\star$ .

```
[9]: ck22.right_nullity()
```

```
[9]: 1
```

That is, the corank of the linear algebraic system equals one.

Next, let us count nonzero Kontsevich's graphs built of three wedges over two sinks. This time, not all of them show up – in the authentic  $\star$ -product – with nonzero weights.

```
[10]: len(list(KGB.graphs(2,3)))
```

```
[10]: 30
```

```
[11]: ck23 = KGB.cyclic_weight_relations(2,3)
      ck23.right_nullity()
```

```
[11]: 11
```

So the rank of the system is  $30 - 11 = 19$ .

**Remark.** Also nonzero graphs with zero Kontsevich weights can produce cyclic weight relations that increase the rank of the system! This is very important for constraining the weights of Kontsevich graphs which actually show up in the star-product. For example, only 13 graphs are seen at  $\hbar^3$  in  $\star$  (cf. Figure 1 in Chapter 11); still the rank of the linear system at hand is  $19 > 13$ . The missing six nontrivial relations were produced by the invisible nonzero graphs.

```
[12]: len(list(KGB.graphs(2,4)))
```

```
[12]: 331
```

```
[13]: ck24 = KGB.cyclic_weight_relations(2,4)
      ck24.right_nullity()
```

```
[13]: 103
```

Hence the rank of the system is 228. From the main result in Chapter 11, i.e. the explicit formula  $\star \bmod \bar{o}(\hbar^4)$  we remember that there are 247 Kontsevich's graphs showing up with nonzero coefficients at  $\hbar^4$  in the star-product. Hence the system of cyclic weight relations, taken alone, does not yet fully constrain those weights: firstly because there are more nonzero weights than the rank of the system. Secondly the cyclic weight relations constrain the weights of *all* the relevant graphs, including those weights which are a posteriori found to be zero numbers. The overall number of unknowns (331) is thus much greater than 247.

```
[14]: len(list(KGB.graphs(2,5)))
```

```
[14]: 4907
```

```
[15]: ck25 = KGB.cyclic_weight_relations(2,5)
      ck25.right_nullity()
```

[15]: 1561

Likewise, here the rank equals 3346.

```
[16]: len(list(KGB.graphs(2,6)))
```

[16]: 91694

```
[17]: #%time ck26 = KGB.cyclic_weight_relations(2,6)
```

```
[18]: #%time ck26.right_nullity()
```

```
[19]: %time len(list(KGB.graphs(2,7)))
```

CPU times: user 3min 21s, sys: 1.68 s, total: 3min 22s  
Wall time: 3min 22s

[19]: 2053511

All the cyclic weight relations corresponding to all the Kontsevich graphs (of positive differential order) in the star-product modulo  $\bar{o}(\hbar^5)$  are available from <https://rburing.nl/gcaops>. The files come in pairs: for each order  $n \leq 6$  there is a file `kontsevich_2_n.txt` containing an ordered basis of Kontsevich graphs built of  $n$  wedges over two sinks, and a file `cyclic_kontsevich_2_n.txt` with all cyclic weight relations between those graphs (as a sparse matrix  $C$  with respect to the ordered basis: each line is of the form  $i \ j \ C[i,j]$ ). The full list of pairs of files is as follows:

- `kontsevich_2_1.txt` and `cyclic_kontsevich_2_1.txt`
- `kontsevich_2_2.txt` and `cyclic_kontsevich_2_2.txt`
- `kontsevich_2_3.txt` and `cyclic_kontsevich_2_3.txt`
- `kontsevich_2_4.txt` and `cyclic_kontsevich_2_4.txt`
- `kontsevich_2_5.txt` and `cyclic_kontsevich_2_5.txt`
- `kontsevich_2_6.txt` and `cyclic_kontsevich_2_6.txt`

### 3.6.3 Leibniz graphs in $(f \star g) \star h - f \star (g \star h)$

We consider the Leibniz graphs appearing in the associator of Kontsevich's star-product:

```
[20]: from gcaops.graph.formality_graph_basis import LeibnizGraphBasis
```

```
[21]: LGB = LeibnizGraphBasis(positive_differential_order=True); LGB
```

[21]: Basis consisting of representatives of isomorphism classes of Leibniz graphs (of positive differential order) with no automorphisms that induce an odd permutation on edges

Note that there is a shift of degrees: Leibniz graphs with  $n - 1$  aerial vertices appear at  $\hbar^n$  in the associator for  $\star$ .

First of all we can count such graphs; recall that all aerial vertices but one are the tops of wedges and one aerial vertex is the top of a tripod.



```
[22]: len(list(LGB.graphs(3,1)))
```

```
[22]: 1
```

Indeed, this is the tripod itself, standing on three sinks.

```
[23]: cl31 = LGB.cyclic_weight_relations(3,1); cl31
```

```
[23]: [0]
```

```
[24]: len(list(LGB.graphs(3,2)))
```

```
[24]: 15
```

The ordered list of encodings of these 15 nonzero Leibniz graphs on three sinks and two aerial vertices is available as file `leibniz_graphs_3_1.txt` from <https://rburing.nl/gcaops>. We notice that with two aerial vertices, no Leibniz graph can be a zero graph (because the aerial vertices have different nature: a wedge versus a tripod, hence there is no automorphism preserving the sinks).

```
[25]: cl32 = LGB.cyclic_weight_relations(3,2); cl32
```

```
[25]: [-1  1  1  0 -1  0  0  0  0  0  0  0  0  0  0]
[  0 -1  2 -1  0  0  0  0  0  0  0  0  0  0  0]
[  1  0  1  0  0  0  0  0  0  0  0  0  0  0  0]
[  0  0  0  1 -1  0  0  0  0  0  0  0  0  0  0]
[  0  0  0  0  1 -1  0  0  0  0  0  0  0  0  0]
[  0  0  0  0  1 -1  0  0  0  0  0  0  0  0  0]
[  0  0  0  0  0  0  1  0  0  0  1  0  0  0  0]
[  0  0  0  0  0  0  0  1  0  0  0  1  0  0  0]
[  0  0  0  0  0  0  1 -1 -1  0  0  0  0  0  0]
[  0  0  0  1  0  0 -1  1  0 -1  0  0  0  0  0]
[  0  0  0  0  0  0  0 -1  0 -1 -1  0  0  0  0]
[  0  0  0  0  1  0 -1  0 -1  0  0 -1  0  0  0]
[  0  0  0  0  0  0 -1  0  0  0  0  0 -1 -1  0]
[  0  0  0  0  0  0  0 -1  0  0  0  0  0 -1 -1]
[  0  0  1  0  0  0  0  0  0  0  0  1  0 -1]
```

**Remark.** By construction, such matrices always have integer entries; the entries are bounded by the number of graphs (that is, by the size of the square matrix). For more information about these relations, interpretations and their origin in the cyclic cohomology theory we refer to Willwacher–Felder [21] and the original paper by Shoikhet [20].

```
[26]: cl32.right_nullity()
```

```
[26]: 3
```

In consequence, the rank of the system of 15 linear algebraic equations at  $\hbar^3$  is 12.

```
[27]: len(list(LGB.graphs(3,3)))
```

```
[27]: 301
```

```
[28]: cl33 = LGB.cyclic_weight_relations(3,3)
      cl33.right_nullity()
```

[28]: 66

The rank of the system of cyclic weight relations for the Leibniz graphs at  $\hbar^4$  in the factorization of the associator is equal to 235.

```
[29]: len(list(LGB.graphs(3,4)))
```

[29]: 6741

```
[30]: cl34 = LGB.cyclic_weight_relations(3,4)
      cl34.right_nullity()
```

[30]: 1469

Thus by having reached 4 aerial vertices in Leibniz graphs, we have reached the order  $\hbar^5$  in the associator; the rank of the system for the weights of Leibniz graphs is 5272.

```
[31]: len(list(LGB.graphs(3,5)))
```

[31]: 171528

```
[32]: #cl35 = LGB.cyclic_weight_relations(3,5)
      #cl35.right_nullity()
```

We are approaching terra incognita; at  $\hbar^6$  in the associator's factorization we know the full list of suitable Leibniz graphs (with 5 aerial vertices), yet we have not computed the rank of the linear system of cyclic weight relations.

```
[33]: %time len(list(LGB.graphs(3,6)))
```

```
CPU times: user 7min 59s, sys: 3.62 s, total: 8min 2s
Wall time: 8min 2s
```

[33]: 4902838

The normal form encodings (with respect to the `gcaops` internal format, based on `nauty`) of bases of Leibniz graphs (of positive differential order) with  $n \leq 5$  aerial vertices at  $\hbar^{n+1}$  are stored in the files `leibniz_3_n.txt` at <https://rburing.nl/gcaops>. The sparse coefficient matrices  $\mathbf{C}$  of cyclic weight relations, referred to the ordering of graphs in the bases, are contained in the plain text files `cyclic_leibniz_3_n.txt` (each line is of the form `i j C[i,j]`). The full list of pairs of files is as follows:

- `leibniz_3_1.txt` and `cyclic_leibniz_3_1.txt`
- `leibniz_3_2.txt` and `cyclic_leibniz_3_2.txt`
- `leibniz_3_3.txt` and `cyclic_leibniz_3_3.txt`
- `leibniz_3_4.txt` and `cyclic_leibniz_3_4.txt`
- `leibniz_3_5.txt` and `cyclic_leibniz_3_5.txt`

### 3.6.4 Known weights satisfy the cyclic weight relations

**Proposition.** We confirm that all these cyclic weight relations up to  $n = 5$  vertices are satisfied by the weights of Kontsevich graphs in the authentic  $\star$ -product and by the weights of Leibniz graphs in its associator; we calculated all these weights by using the `kontsevint` program by Panzer [1].

**Notation.** In Panzer's `kontsevint` the  $m$  ground vertices of a Formality graph are labeled by  $p_1, \dots, p_m$  and the  $n$  internal vertices by integers  $1, \dots, n$ ; the encoding then consists of the list of lists of targets of the aerial vertices.

```
[34]: def kontsevint_weights_maple_program(graph_list):
      return '[]';'.format(', '.join(['weight({})'.format(g.kontsevint_encoding()) for
      ↪g in graph_list]))
```

```
[35]: def vector_from_file(filename):
      with open(filename) as f:
          return vector(sage_eval('[]'.format(', '.join(f.readlines()))))
```

**Example.**

```
[36]: print(kontsevint_weights_maple_program(KGB.graphs(2,1)))
```

```
[weight([[p1,p2]])];
```

```
[37]: wk21 = vector_from_file('data/weights_kontsevich_2_1.txt'); wk21
```

```
[37]: (1/2)
```

```
[38]: len(wk21) - list(wk21).count(0)
```

```
[38]: 1
```

```
[39]: ck21*wk21
```

```
[39]: (0)
```

**Example.**

```
[40]: print(kontsevint_weights_maple_program(KGB.graphs(2,2)))
```

```
[weight([[p2,2],[p1,p2]]),weight([[p1,2],[p1,p2]]),weight([[p1,p2],[p1,p2]],weight([[
p2,2],[p1,1]])];
```

```
[41]: wk22 = vector_from_file('data/weights_kontsevich_2_2.txt'); wk22
```

```
[41]: (-1/12, 1/12, 1/4, -1/24)
```

```
[42]: len(wk22) - list(wk22).count(0)
```

```
[42]: 4
```



```
[52]: #print(kontsevint_weights_maple_program(Kgraphs(2,5)))
```

```
[53]: wk25 = vector_from_file('data/weights_kontsevich_2_5.txt') #; wk25
```

```
[54]: len(wk25) - list(wk25).count(0)
```

```
[54]: 2356
```

```
[55]: (ck25*wk25).norm()
```

```
[55]: 0
```

The files `kontsevich_2_n.txt` with the normal form encodings of all these Kontsevich graphs (of positive differential order) and the files `weights_kontsevich_2_n.txt` with their weights (up to  $n = 6$ ) are stored at <https://rburing.nl/gcaops>. The full list of pairs of files is as follows:

- `kontsevich_2_1.txt` and `weights_kontsevich_2_1.txt`
- `kontsevich_2_2.txt` and `weights_kontsevich_2_2.txt`
- `kontsevich_2_3.txt` and `weights_kontsevich_2_3.txt`
- `kontsevich_2_4.txt` and `weights_kontsevich_2_4.txt`
- `kontsevich_2_5.txt` and `weights_kontsevich_2_5.txt`
- `kontsevich_2_6.txt` and `weights_kontsevich_2_6.txt`

Now we deal with the Leibniz graphs.

```
[56]: print(kontsevint_weights_maple_program(LGB.graphs(3,1)))
```

```
[weight([[p1,p2,p3]])];
```

This is the tripod.

```
[57]: wl31 = vector_from_file('data/weights_leibniz_3_1.txt'); wl31
```

```
[57]: (1/6)
```

```
[58]: len(wl31) - list(wl31).count(0)
```

```
[58]: 1
```

```
[59]: cl31*wl31
```

```
[59]: (0)
```

```
[60]: print(kontsevint_weights_maple_program(LGB.graphs(3,2)))
```

```
[weight([[p3,2],[p1,p2,p3]]),weight([[p2,2],[p1,p2,p3]]),weight([[p1,2],[p1,p2,p3]]),weight([[p1,p2],[p1,p2,p3]]),weight([[p1,p3],[p1,p2,p3]]),weight([[p2,p3],[p1,p2,p3]]),weight([[p1,p3],[p1,p2,1]]),weight([[p1,p3,2],[p1,p2]]),weight([[p2,p3],[p1,p2,1]]),weight([[p2,p3,2],[p1,p2]]),weight([[p2,p3],[p1,p3,1]]),weight([[p2,p3,2],[p1,p3]]),weight([[p3,2],[p1,p2,1]]),weight([[p2,2],[p1,p3,1]]),weight([[p2,p3,2],[p1,1]])];
```





**Definition.** A Kontsevich graph (built of wedges) is called *affine* if the in-degree of each of its aerial vertices is  $\leq 1$ .

We define the basis of relevant affine graphs:

```
[1]: KGB_affine = KontsevichGraphBasis(positive_differential_order=True,
↳max_aerial_in_degree=1); KGB_affine
```

[1]: Basis consisting of representatives of isomorphism classes of Kontsevich graphs (of positive differential order, with aerial vertices of in-degree  $\leq 1$ ) with no automorphisms that induce an odd permutation on edges

The affine  $\star$ -product expansion and the expansion of its associator will be defined as elements of FGC:

```
[2]: from gcaops.graph.formality_graph_complex import FormalityGraphComplex
FGC = FormalityGraphComplex(SR, lazy=True); FGC
```

[2]: Formality graph complex over Symbolic Ring with Basis consisting of representatives of isomorphism classes of formality graphs with no automorphisms that induce an odd permutation on edges

We can restrict the full Kontsevich  $\star$ -product mod  $\bar{o}(\hbar^6)$  to affine graphs, and store the result in a file:

```
[3]: #star6_txt = open('data/star6.txt').read().rstrip()
#star6 = FGC.element_from_kgs_encoding(star6_txt) #; star6
#affine_star6 = star6.filter(max_aerial_in_degree=1)
#with open('data/affine_star6.txt', 'w') as f:
#    f.write(affine_star6.kgs_encoding())
```

After this is done once and the result is stored (or the file `data/affine_star6.txt` is imported from elsewhere), we can read back the result:

```
[4]: affine_star6_txt = open('data/affine_star6.txt').read().rstrip()
affine_star6 = FGC.element_from_kgs_encoding(affine_star6_txt) #; affine_star6
```

We obtain 465 graphs in total in the affine star product up to  $\bar{o}(\hbar^6)$  — only about 0.678% of the original 68663 graphs showing up in Kontsevich's  $\star$  mod  $\bar{o}(\hbar^6)$  with the harmonic weights:

```
[5]: len(affine_star6)
```

[5]: 465

### 3.7.2 Relations between the weights at $\hbar^7$

In the basis of affine Kontsevich graphs suitable for  $\star$ -products there are 1731 graphs with 7 aerial vertices on 2 ground vertices:

```
[6]: len(KGB_affine.graphs(2,7))
```

[6]: 1731

Let us find the relations between the Kontsevich weights, to reduce the number of unknowns.



**Lemma.** Upon flipping a graph on  $n$  vertices, i.e. interchanging the two sinks, its weight is multiplied by  $(-1)^n$ .

```
[7]: %time flipping_matrix = KGB_affine.flipping_weight_relations(2,7)
```

```
CPU times: user 1.11 s, sys: 0 ns, total: 1.11 s
Wall time: 1.11 s
```

**Lemma.** The cyclic weight relations restrict to the subset of affine graphs.

*Proof.* Re-directing edges to ground vertices does not affect the in-degree of *aerial* vertices.

```
[8]: %time cyclic_matrix = KGB_affine.cyclic_weight_relations(2,7)
```

```
CPU times: user 1min 9s, sys: 1.46 s, total: 1min 10s
Wall time: 1min 10s
```

**Lemma.** The weight of a graph with an “eye on ground”, i.e. containing a 2-cycle between aerial vertices such that both vertices in the 2-cycle are connected to the same ground vertex, vanishes.

*Proof.* By a dimension count.

```
[9]: %time eye_on_ground_matrix = KGB_affine.eye_on_ground_weight_relations(2,7)
```

```
CPU times: user 266 ms, sys: 4.05 ms, total: 270 ms
Wall time: 267 ms
```

**Lemma.** The weight of a disconnected graph vanishes.

(Note that such graphs can consist of two components, each standing on one sink, but also e.g., of one component standing on two sinks and a purely aerial component.)

```
[10]: %time disconnected_indices = [k for k,g in enumerate(KGB_affine.graphs(2,7)) if not
↳ DiGraph(g.edges()).is_connected()]
%time disconnected_matrix = Matrix(ZZ, len(disconnected_indices), len(KGB_affine.
↳ graphs(2,7)), {(i,k) : 1 for (i,k) in enumerate(disconnected_indices)})
```

```
CPU times: user 236 ms, sys: 91 μs, total: 236 ms
Wall time: 232 ms
CPU times: user 1.4 ms, sys: 0 ns, total: 1.4 ms
Wall time: 1.34 ms
```

**Lemma.** The weight of a composite Kontsevich graph  $\Gamma = \Gamma_1 \bar{\times} \Gamma_2$  on  $n = n_1 + n_2$  aerial vertices is (by the graph weights’ multiplicativity) equal to the product of the weights:  $w(\Gamma_1 \bar{\times} \Gamma_2) = w(\Gamma_1) \cdot w(\Gamma_2)$ .

The multiplicativity of weights yields (inhomogenous) *linear* relations for the weights of composite graphs at order  $n$  as soon as the weights of graphs at lower orders  $k < n$  are known. The weights of graphs at lower orders can be read off from the  $\star$ -product at  $\hbar^k$  by multiplying the coefficient of a graph  $\Gamma$  by  $\frac{k!}{m(\Gamma)}$ , where  $m(\Gamma)$  is the multiplicity.

```
[11]: affine_weight_vectors = {k : vector(SR, len(KGB_affine.graphs(2, k)), {KGB_affine.
↳graphs(2,k).index(g) : c*factorial(k)/g.multiplicity() for c, g in affine_star6.
↳homogeneous_part(2, k, 2*k}}, sparse=True) for k in range(1,7)}
```

We can store these weight vectors in plain text files for posterity:

```
[12]: #for k in range(1,7):
#     with open('data/affine_weights_kontsevich_2_{}.txt'.format(k), 'w') as f:
#         for c in affine_weight_vectors[k]:
#             f.write(str(c).replace(' ', '') + '\n')
```

For  $k = 1, 2, 3, 4, 5$  the harmonic weights of graphs in the (affine) Kontsevich  $\star$ -product at  $\hbar^k$  are rational numbers. At  $\hbar^6$  the harmonic weights of graphs in the  $\star$ -product are  $\mathbb{Q}$ -linear combinations of 1 and  $\zeta(3)^2/\pi^6$ . We will express each of the composite weights at  $\hbar^7$  as such a  $\mathbb{Q}$ -linear combination, so that we obtain relations with rational coefficients. (See also the Remark at the end of this section.)

```
[13]: %%time
w0 = SR.wild() # Wildcard, used for substitution.
composite_vectors = {}
for (p1,p2) in Partitions(7,length=2):
    for (g_idx, h_idx, plusminus_gh_idx, plusminus) in KGB_affine.
↳multiplication_table(2,p1,p2):
        prod_weight = 
↳plusminus*affine_weight_vectors[p1][g_idx]*affine_weight_vectors[p2][h_idx]
        # Replace `zeta(3)^2/pi^6` by `z`.
        prod_weight = prod_weight.subs({zeta(3)^2/pi^6 : var('z'), w0*zeta(3)^2/pi^6 :
↳ w0*var('z')})
        prod_weight_a = prod_weight.subs({var('z'):0})
        prod_weight_b = prod_weight.coefficient(var('z'))
        v = vector(QQ, len(KGB_affine.graphs(2,7)) + 2, {
            plusminus_gh_idx : 1,
            len(KGB_affine.graphs(2,7)) : -prod_weight_a,
            len(KGB_affine.graphs(2,7)) + 1 : -prod_weight_b
        }, sparse=True)
        if plusminus_gh_idx in composite_vectors:
            # We already expressed this weight as a constant. let's make sure it's 
↳the same constant:
            assert v == composite_vectors[plusminus_gh_idx]
        else:
            composite_vectors[plusminus_gh_idx] = v
composites_matrix = matrix(QQ, composite_vectors.values())
```

```
CPU times: user 4.26 s, sys: 72.1 ms, total: 4.34 s
Wall time: 4.34 s
```

Now, by construction, every linear relation between the Kontsevich graph weights is a row with, firstly, rational coefficients of the unknown weights (for graphs which are ordered in a basis), followed by the last two columns (for the future right-hand side) with the rational coefficients of 1 and of  $\zeta(3)^2/\pi^6$  respectively. All these rows are combined into a matrix. We thus obtain a matrix with rational entries, which will be beneficial for performance when solving the linear system in the next section (e.g., a one-hour computation over  $\mathbb{Q}[z]$  becomes an equivalent one-second calculation over  $\mathbb{Q}$ ).

**Remark.** At orders  $\geq 12$ , the number  $(\zeta(3)^2/\pi^6)^2$  will show up (with a nonzero rational coefficient) in some of the weights of composite graphs. In general we need generators over  $\mathbb{Q}$  of multiple zeta values to express the multiplicativity of the weight as a relation over  $\mathbb{Q}$ .

### 3.7.3 From weight relations to master parameters

We put all the matrices together:

```
[14]: %%time
big_matrix = block_matrix([[flipping_matrix, zero_matrix(flipping_matrix.nrows(), 2)],
                           [cyclic_matrix, zero_matrix(cyclic_matrix.nrows(), 2)],
                           [eye_on_ground_matrix, zero_matrix(eye_on_ground_matrix.nrows(), 2)],
                           [disconnected_matrix, zero_matrix(disconnected_matrix.nrows(), 2)]]).
↳stack(composites_matrix)
big_matrix
```

CPU times: user 5.22 s, sys: 60 ms, total: 5.28 s  
Wall time: 5.28 s

```
[14]: 3936 x 1733 sparse matrix over Rational Field (use the '.str()' method to see the
entries)
```

In the above, 3936 is the total number of relations constraining the 1731 unknown weights of affine Kontsevich graphs on  $n = 7$  aerial vertices in  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$ , and 1733 is the width of each row of known coefficients (extended by the rational coefficients of 1 and of  $\zeta(3)^2/\pi^6$  in the right-hand side).

We compute the reduced *row* echelon form of this non-square matrix:

```
[15]: %time big_matrix_rref = big_matrix.rref() #algorithm='scaled_partial_pivoting')
```

CPU times: user 982 ms, sys: 12 ms, total: 994 ms  
Wall time: 996 ms

The general solution of the big  $\mathbb{Q}$ -linear system contains 78 parameters:

```
[16]: big_matrix_rref.right_nullity()
```

```
[16]: 78
```

By construction two of the parameters can be chosen to be 1 and  $\zeta(3)^2/\pi^6$ , so all the weights can be expressed as  $\mathbb{Q}$ -linear combinations of 1,  $\zeta(3)^2/\pi^6$ , and 76 parameters.

```
[17]: %time weight_directions = big_matrix_rref.right_kernel().basis() # fast because we
↳have the rref
```

CPU times: user 15.5 s, sys: 160 ms, total: 15.6 s  
Wall time: 15.6 s

We first obtain the parts of the solution expressed directly as a  $\mathbb{Q}$ -linear combinations of 1 and  $\zeta(3)^2/\pi^6$ :

```
[18]: inhomogeneouspart_directions = [v for v in weight_directions if v[-1] != 0 or v[-2] != 0]
      ↪= 0]
```

```
[19]: len(inhomogeneouspart_directions)
```

```
[19]: 2
```

Extract the coefficients of 1 and  $\zeta(3)^2/\pi^6$ :

```
[20]: inhomogeneouspart_submatrix = matrix(QQ, [v[-2:] for v in inhomogeneouspart_directions]); inhomogeneouspart_submatrix
      ↪= inhomogeneouspart_submatrix
```

```
[20]: [ 128 1216/6075]
      [ 0 512/15]
```

Make a  $\mathbb{Q}$ -linear transformation to obtain one part proportional to 1 and another part proportional to  $\zeta(3)^2/\pi^6$ :

```
[21]: inhomogeneouspart_new = [sum(scale*v for scale, v in zip(scales, inhomogeneouspart_submatrix.inverse().rows())
      ↪= inhomogeneouspart_submatrix.inverse().rows()) for scales in inhomogeneouspart_submatrix.inverse().rows()]
      ↪= rows()]
```

```
[22]: matrix(QQ, [v[-2:] for v in inhomogeneouspart_new])
```

```
[22]: [1 0]
      [0 1]
```

We now choose parameters for the homogeneous part of the solution:

```
[23]: homogeneouspart_directions = [v for v in weight_directions if v[-1] == 0 and v[-2] == 0]
      ↪= 0]
```

The linear solver is such that the first nonzero entry of each vector is a 1:

```
[24]: all([next(v_i for v_i in v if v_i != 0) == 1 for v in homogeneouspart_directions])
```

```
[24]: True
```

Hence the index of that first 1 is the index of a graph (in the basis) whose weight can be chosen as a master parameter.

```
[25]: master_indices = [list(v).index(1) for v in homogeneouspart_directions]
```

```
[26]: len(master_indices)
```

```
[26]: 76
```

### 3.7.4 Substitute master parameters into $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$ and its associator

Introduce undetermined weights for all affine graphs at  $\hbar^7$ :

```
[27]: w = [var('w{}'.format(k)) for k in range(len(KGB_affine.graphs(2,7)))]
```

```
[28]: master_weights = [w[i] for i in master_indices]
[29]: affine_star7w = FGC(affine_star6) + 1/factorial(7)*FGC([(g.multiplicity()*w[k], g)
↳ for (k,g) in enumerate(KGB_affine.graphs(2,7))]) #; affine_star7w
[30]: len(affine_star7w)
[30]: 2196
```

This is how many graphs we have with their weights known at orders  $\leq 6$  and none of the weights known yet at order 7.

Create a substitution in terms of the master parameters:

```
[31]: inhomogeneouspart_symbolic = vector(SR, inhomogeneouspart_new[0][: -2]) +
↳ var('z')*vector(SR, inhomogeneouspart_new[1][: -2])
[32]: homogeneouspart_symbolic = sum(w[idx]*vector(SR, v[: -2]) for idx, v in
↳ zip(master_indices, homogeneouspart_directions))
[33]: my_subs = dict(zip(w, inhomogeneouspart_symbolic + homogeneouspart_symbolic))
```

Apply the substitution:

```
[34]: affine_star7_master = affine_star7w.map_coefficients(lambda c: c.subs(my_subs))
[35]: len(affine_star7_master)
[35]: 1458
```

So, by taking into account all the so far known relations we already reduce the number of nonzero coefficients of the topmost graphs. The coefficients of the remaining topmost graphs will be fixed in the near future, and many of them will vanish as well.

Compute the associator mod  $\bar{o}(\hbar^7)$ :

```
[36]: %time affine_assoc7_master = affine_star7_master.insertion(0, affine_star7_master,
↳ max_num_aerial=7, max_aerial_in_degree=1) - affine_star7_master.insertion(1,
↳ affine_star7_master, max_num_aerial=7, max_aerial_in_degree=1)
```

```
CPU times: user 2min, sys: 681 ms, total: 2min
Wall time: 2min
```

```
[37]: affine_assoc7_master7 = affine_assoc7_master.homogeneous_part(3,7,14)
[38]: len(affine_assoc7_master7)
[38]: 50429
```

This is how many Kontsevich graphs, with their coefficients expressed in terms of the 76 master parameters, currently survive into the associator at  $\hbar^7$  (if all the topmost graph coefficients are undetermined, the associator contains 69843 terms at  $\hbar^7$ ).

```
[39]: len(list(affine_assoc7_master7.differential_orders()))
```

[39]: 161

Likewise, this is how many tri-differential orders survive into the associator at  $\hbar^7$  so far — out of 203.

### 3.7.5 Restrict onto generic affine Poisson bivector on $\mathbb{R}^2$

We consider a generic affine Poisson bivector  $P = (Ax + By + C) \partial_x \wedge \partial_y$  on  $\mathbb{R}^2$ :

[40]: `R.<x,y,A,B,C> = SR[]`

[41]: `SA.<xi1,xi2> = SuperfunctionAlgebra(R, [x,y])`

[42]: `PA.<ddx,ddy> = PolyDifferentialOperatorAlgebra(R, [x,y])`

[43]: `P = (A*x+B*y+C)*xi1*xi2; P`

[43]: `(x*A + y*B + C)*xi1*xi2`

The part at order  $\hbar^7$  of the associator  $(f \star g) \star h - f \star (g \star h)$  for Kontsevich's  $\star$  will be evaluated at this  $P$ . This results in a tri-differential operator  $A^{I_1 I_2 I_3} \partial_{I_1} \otimes \partial_{I_2} \otimes \partial_{I_3}$  acting on  $f \otimes g \otimes h \in C^\infty(\mathbb{R}^2)^{\otimes 3}$  with coefficients  $A^{I_1 I_2 I_3}$  which are polynomials in  $x, y$  with coefficients in  $SR[A, B, C]$ . The vanishing of this tri-differential operator and hence of all these polynomials yields relations between the master parameters.

[44]: `from gcaops.graph.formality_graph_operator import FormalityGraphOperator`

[45]: 

```

#%time
#assoc2_eqns = []
#for diff_order in affine_assoc7_master7.differential_orders():
#    part = affine_assoc7_master7.part_of_differential_order(diff_order)
#    print(diff_order, ':', len(part))
#    operat = FormalityGraphOperator(SA, PA, part)
#    op = operat.value_at_copies_of(P)
#    assoc2_eqns.extend(sum([op[m].coefficients() for m in op.multi_indices()], []))

```

[46]: `#solve(assoc2_eqns, master_weights)`

Thus we would reduce the number of master parameters from 76 down to 74. Since this is only a small improvement, we ignore it.

### 3.7.6 Restrict onto affine rescaled Nambu–Poisson bivector on $\mathbb{R}^3$

On  $\mathbb{R}^3$  there are two families of affine (rescaled) Nambu–Poisson brackets  $\{f, g\} = \rho \left| \frac{\partial(\varphi, f, g)}{\partial(x, y, z)} \right|$ , with  $(\rho, \varphi)$  polynomial of degrees  $(\deg(\rho), \deg(\varphi))$  equal to  $(1, 1)$  or  $(0, 2)$ . Since  $\varphi$  is always differentiated once we can omit its constant term in either case.

First let  $\rho = ax + by + cz + d$  and  $\varphi = Ax + By + Cz$ :

[47]: `D.<ddx,ddy,ddz> = PolyDifferentialOperatorAlgebra(SR, var('x,y,z'))`  
`S.<xi1,xi2,xi3> = SuperfunctionAlgebra(SR, var('x,y,z'), simplify='expand', ↵`  
`↪is_zero='is_trivial_zero')`

```
phi = var('A')*x + var('B')*y + var('C')*z
P_1_1 = (var('a')*x+var('b')*y+var('c')*z+var('d'))*(diff(phi,x)*xi2*xi3 +
↪diff(phi,y)*xi3*xi1 + diff(phi,z)*xi1*xi2)
```

```
[48]: %%time
#assoc3_1_1_eqns = []
#for diff_order in star7.differential_orders():
#    print(diff_order)
#    part = star7.part_of_differential_order(diff_order)
#    operat = FormalityGraphOperator(S, D, part)
#    op = operat.value_at_copies_of(P_1_1)
#    assoc3_1_1_eqns.extend(sum([op[m].coefficients() for m in op.multi_indices()],
↪[]))
```

```
[49]: #solve(assoc3_1_1_eqns, master_weights)
```

Now let  $\rho = 1$  and  $\varphi = Ax^2 + By^2 + Cz^2 + Dxy + Exz + Fyz$ :

```
[50]: phi = var('A')*x^2 + var('B')*y^2 + var('C')*z^2 + var('D')*x*y + var('E')*x*z +
↪var('F')*y*z
P_0_2 = (diff(phi,x)*xi2*xi3 + diff(phi,y)*xi3*xi1 + diff(phi,z)*xi1*xi2)
```

```
[51]: %%time
#assoc3_0_2_eqns = []
#for diff_order in star7.differential_orders():
#    print(diff_order)
#    part = star7.part_of_differential_order(diff_order)
#    operat = FormalityGraphOperator(S, D, part)
#    op = operat.value_at_copies_of(P_0_2)
#    assoc3_0_2_eqns.extend(sum([op[m].coefficients() for m in op.multi_indices()],
↪[]))
```

```
[52]: #solve(assoc3_0_2_eqns, master_weights)
```

It would be interesting to see how far the number of parameters drops in these cases.

### 3.7.7 Direct calculation of master parameter values

We calculate the values of the 76 master parameters directly using Panzer's `kontsevint` program:

```
[53]: #master_graphs = [KGB_affine.graphs(2,7)[i] for i in master_indices]
#print('[' + ', '.join('weight({})'.format(g.kontsevint_encoding().replace('p1','L')).
↪replace('p2','R')) for g in master_graphs) + '];')
```

```
[54]:
```

```

master_values = [0, 3/2048*zeta(3)^2/pi^6 + 943/46448640, 1/2048*zeta(3)^2/pi^6 + 257/
↳46448640, 65/2048*zeta(3)^2/pi^6 - 437/9289728, 149/2048*zeta(3)^2/pi^6 - 5239/
↳46448640, -99/2048*zeta(3)^2/pi^6 + 53/737280, 27/2048*zeta(3)^2/pi^6 - 289/
↳15482880, -31/2048*zeta(3)^2/pi^6 + 131/6635520, -35/1024*zeta(3)^2/pi^6 + 121/
↳1935360, -5/512*zeta(3)^2/pi^6 + 131/9289728, -27/2048*zeta(3)^2/pi^6 + 43/
↳15482880, 1/256*zeta(3)^2/pi^6 + 1/573440, -17/2048*zeta(3)^2/pi^6 + 9/573440, 1/
↳512*zeta(3)^2/pi^6 + 1/1658880, -49/2048*zeta(3)^2/pi^6 + 1849/46448640, -31/
↳1024*zeta(3)^2/pi^6 + 893/11612160, 143/2048*zeta(3)^2/pi^6 - 53/516096, 71/
↳2048*zeta(3)^2/pi^6 - 1/20480, 53/1024*zeta(3)^2/pi^6 - 163/2322432, 29/
↳1024*zeta(3)^2/pi^6 - 41/1161216, 5/512*zeta(3)^2/pi^6 - 1019/46448640, -1/
↳128*zeta(3)^2/pi^6 + 293/23224320, 95/1024*zeta(3)^2/pi^6 - 1301/9289728, -5/
↳128*zeta(3)^2/pi^6 + 71/1290240, 179/2048*zeta(3)^2/pi^6 - 2029/15482880, -3/
↳1024*zeta(3)^2/pi^6 - 109/7741440, -57/2048*zeta(3)^2/pi^6 + 85/2322432, 17/
↳512*zeta(3)^2/pi^6 - 121/3317760, -7/512*zeta(3)^2/pi^6 + 17/860160, -1/241920, 0,
↳1/256*zeta(3)^2/pi^6 - 89/3870720, 13/1024*zeta(3)^2/pi^6 - 1/5806080, 0, -55/
↳1024*zeta(3)^2/pi^6 + 1021/13271040, 57/2048*zeta(3)^2/pi^6 - 3659/92897280, 5/
↳1024*zeta(3)^2/pi^6 + 199/11612160, 1/2048*zeta(3)^2/pi^6 + 41/23224320, -7/
↳2048*zeta(3)^2/pi^6 + 263/46448640, 1/64*zeta(3)^2/pi^6 - 37/2903040, -25/
↳1024*zeta(3)^2/pi^6 + 25/663552, 53/2048*zeta(3)^2/pi^6 - 853/15482880, 3/
↳128*zeta(3)^2/pi^6 - 3131/92897280, -47/2048*zeta(3)^2/pi^6 + 3229/92897280, 67/
↳5806080, 1/256*zeta(3)^2/pi^6 - 157/11612160, -19/1024*zeta(3)^2/pi^6 + 199/
↳5806080, 3/128*zeta(3)^2/pi^6 - 7/221184, -7/2048*zeta(3)^2/pi^6 + 31/5806080, -27/
↳1024*zeta(3)^2/pi^6 + 3961/92897280, -65/2048*zeta(3)^2/pi^6 + 2123/46448640, -37/
↳1024*zeta(3)^2/pi^6 + 5011/92897280, -35/2048*zeta(3)^2/pi^6 + 37/2211840, -31/
↳2048*zeta(3)^2/pi^6 + 461/18579456, -29/1024*zeta(3)^2/pi^6 + 733/18579456, 3/
↳512*zeta(3)^2/pi^6 + 1/241920, -7/512*zeta(3)^2/pi^6 + 13/573440, 5/512*zeta(3)^2/
↳pi^6 - 17/1161216, 13/2048*zeta(3)^2/pi^6 - 149/5806080, -5/1024*zeta(3)^2/pi^6 + 7/
↳1105920, 3/2048*zeta(3)^2/pi^6 - 71/46448640, -87/4096*zeta(3)^2/pi^6 + 751/
↳26542080, -39/512*zeta(3)^2/pi^6 + 1807/15482880, 1/241920, 1/512*zeta(3)^2/pi^6 -
↳89/7741440, -1/512*zeta(3)^2/pi^6 + 1/11612160, -83/4096*zeta(3)^2/pi^6 + 1427/
↳46448640, 0, 1/241920, 0, -1/241920, 0, 1/241920, 5/2048*zeta(3)^2/pi^6 - 7/
↳2211840, 15/2048*zeta(3)^2/pi^6 - 31/3870720, 11/2048*zeta(3)^2/pi^6 - 179/23224320]

```

```

[55]: #master_values_subs = dict({w[idx] : master_values[k] for k, idx in
↳enumerate(master_indices)})
#affine_star7 = affine_star7_master.map_coefficients(lambda c: c.
↳subs(master_values_subs))

```

We have finally obtained the affine star-product  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$  with the harmonic weights! It is also contained in Appendix C.1 of the dissertation.

We write the result to the file `data/affine_star7.txt`:

```

[56]: #with open('data/affine_star7.txt', 'w') as f:
#     f.write(affine_star7.kgs_encoding())

```

After the file is saved (or imported from elsewhere) the result can be read back:

```

[57]: affine_star7 = FGC.element_from_kgs_encoding(open('data/affine_star7.txt').read().
↳rstrip())

```

**Remark.** In Section 3.7.9 below we compare the formula  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$  with an earlier result of Ben Amar (2003) about the rationality of coefficients in it.

### 3.7.8 Certificate of associativity

We calculate the associator for Kontsevich's affine  $\star$ -product up to  $\bar{o}(\hbar^7)$ :



```
[58]: %time affine_assoc7 = affine_star7.insertion(0, affine_star7, max_num_aerial=7,
↳max_aerial_in_degree=1) - affine_star7.insertion(1, affine_star7, max_num_aerial=7,
↳max_aerial_in_degree=1)
```

CPU times: user 1min 54s, sys: 733 ms, total: 1min 55s  
Wall time: 1min 54s

```
[59]: affine_assoc7_7 = affine_assoc7.homogeneous_part(3,7,14)
```

```
[60]: len(affine_assoc7_7)
```

```
[60]: 49621
```

This is how many Kontsevich graphs truly survive into the associator at order  $\hbar^7$  for the genuine affine  $\star$ -product modulo  $\bar{o}(\hbar^7)$  with harmonic propagators in the graph weights.

We now obtain *some* Leibniz graph factorization of (each tri-differential component of) the associator at  $\hbar^7$  for  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$ , as seen before in Section 3.5. Namely, we contract an edge between aerial vertices (in all possible ways) in the Kontsevich graphs to obtain Leibniz graphs, and we expand the resulting Leibniz graphs to Kontsevich graphs again (as explained in [10]). (Each time we expand Leibniz graphs we necessarily reproduce the Kontsevich graphs which were seen previously, but we possibly also meet new Kontsevich graphs.) At each stage where Leibniz graphs are obtained, we check if they are enough to factor the respective tri-differential component  $A_{d_1 d_2 d_3}$  of the associator  $A$  at  $\hbar^7$ , by equating  $A_{d_1 d_2 d_3}$  to the Kontsevich graph expansion of the obtained Leibniz graphs with undetermined coefficients, and trying to solve the linear system. If the so far available set of Leibniz graphs is not enough, the program builds the next layer of (neighbor) Leibniz graphs by contracting edges in *all* the Kontsevich graphs from the expansions of previously available Leibniz graphs. This is repeated until a solution appears for the factorization problem in that specific tri-differential order.

To obtain and solve the respective linear system over  $\mathbb{Q}$ , we use the following helper functions.

```
[63]: from gcaops.graph.leibniz_graph_expansion import
↳kontsevich_graph_sum_to_leibniz_graph_sum

def coefficient_to_vector(c):
    c_poly = QQ['zzz'](str(SR(c).expand()).replace('zeta(3)^2/pi^6', 'zzz'))
    return vector(QQ, [c_poly.constant_coefficient(), c_poly.
↳monomial_coefficient(QQ['zzz'].gen())])
def vector_to_coefficient(v):
    return v[0] + v[1]*zeta(3)^2/pi^6
```

The following lines of output contain the grading of a tri-differential component at of the associator at  $\hbar^7$  for  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$ , followed by the number of Kontsevich graphs in that component, followed by the number of new Leibniz graphs and Kontsevich graphs obtained in each step (usually one step, exceptionally two steps in four cases).

```
[64]: f_leibniz = open('data/affine_assoc7_leibniz.txt', 'w')
f_leibniz_coeffs = open('data/affine_assoc7_leibniz_coeffs.txt', 'w')
for diff_order in affine_assoc7_7.differential_orders():
    print(diff_order, end=': ', flush=True)
```

```

part = affine_assoc7_7.part_of_differential_order(diff_order)
part_Leibniz = kontsevich_graph_sum_to_leibniz_graph_sum(part,
↳coefficient_to_vector=coefficient_to_vector,
↳vector_to_coefficient=vector_to_coefficient, max_aerial_in_degree=1, verbose=True)
    for c,L in part_Leibniz:
        f_leibniz.write(str(L.edges()).replace(' ','') + '\n')
        f_leibniz_coefs.write(str(c).replace(' ','') + '\n')
f_leibniz.close()
f_leibniz_coefs.close()

```

```

(4, 3, 4): 236K -> +164L -> +21K
(3, 4, 4): 232K -> +164L -> +25K
(3, 3, 4): 835K -> +835L -> +116K
(2, 5, 4): 111K -> +80L -> +16K
(2, 4, 4): 539K -> +535L -> +79K
(5, 2, 4): 117K -> +81L -> +10K
(4, 2, 4): 552K -> +541L -> +66K
(5, 3, 4): 32K -> +16L -> +4K
(4, 4, 4): 38K -> +18L -> +4K
(3, 5, 4): 32K -> +16L -> +4K
(5, 4, 3): 32K -> +16L -> +4K
(5, 3, 3): 143K -> +98L -> +14K
(4, 5, 3): 32K -> +16L -> +4K
(4, 4, 3): 232K -> +164L -> +25K
(4, 3, 3): 835K -> +835L -> +116K
(3, 6, 3): 10K -> +5L -> +2K
(3, 5, 3): 135K -> +97L -> +22K
(3, 4, 3): 811K -> +827L -> +139K
(2, 6, 3): 23K -> +14L -> +5K
(2, 5, 3): 286K -> +274L -> +60K
(6, 3, 3): 12K -> +5L -> +0K
(6, 2, 3): 27K -> +14L -> +1K
(5, 2, 3): 320K -> +280L -> +27K
(6, 4, 3): 3K -> +1L -> +0K
(5, 5, 3): 3K -> +1L -> +0K
(4, 6, 3): 3K -> +1L -> +0K
(2, 6, 4): 10K -> +5L -> +2K
(6, 2, 4): 12K -> +5L -> +0K
(6, 3, 4): 3K -> +1L -> +0K
(5, 4, 4): 3K -> +1L -> +0K
(4, 5, 4): 3K -> +1L -> +0K
(3, 6, 4): 3K -> +1L -> +0K
(2, 3, 4): 1387K -> +1721L -> +148K
(3, 2, 4): 1411K -> +1730L -> +125K
(5, 4, 2): 115K -> +81L -> +12K
(5, 3, 2): 319K -> +280L -> +28K
(4, 5, 2): 111K -> +80L -> +16K
(4, 4, 2): 539K -> +535L -> +79K
(4, 3, 2): 1387K -> +1721L -> +148K
(3, 6, 2): 23K -> +14L -> +5K
(3, 5, 2): 286K -> +274L -> +60K
(3, 4, 2): 1300K -> +1691L -> +229K
(2, 6, 2): 36K -> +27L -> +11K
(2, 5, 2): 414K -> +508L -> +112K
(6, 3, 2): 27K -> +14L -> +1K
(6, 2, 2): 40K -> +27L -> +7K
(5, 2, 2): 479K -> +522L -> +51K

```

(6, 4, 2): 12K  $\rightarrow$  +5L  $\rightarrow$  +0K  
 (5, 5, 2): 26K  $\rightarrow$  +14L  $\rightarrow$  +4K  
 (4, 6, 2): 10K  $\rightarrow$  +5L  $\rightarrow$  +2K  
 (2, 2, 4): 1431K  $\rightarrow$  +2111L  $\rightarrow$  +98K  
 (3, 3, 3): 2216K  $\rightarrow$  +2897L  $\rightarrow$  +296K  
 (2, 4, 3): 1300K  $\rightarrow$  +1691L  $\rightarrow$  +229K  
 (4, 2, 3): 1411K  $\rightarrow$  +1730L  $\rightarrow$  +125K  
 (5, 4, 1): 135K  $\rightarrow$  +118L  $\rightarrow$  +9K  
 (5, 3, 1): 268K  $\rightarrow$  +291L  $\rightarrow$  +28K  
 (4, 5, 1): 122K  $\rightarrow$  +116L  $\rightarrow$  +22K  
 (4, 4, 1): 473K  $\rightarrow$  +558L  $\rightarrow$  +47K  
 (4, 3, 1): 790K  $\rightarrow$  +1140L  $\rightarrow$  +57K  
 (3, 6, 1): 24K  $\rightarrow$  +20L  $\rightarrow$  +10K  
 (3, 5, 1): 230K  $\rightarrow$  +280L  $\rightarrow$  +63K  
 (3, 4, 1): 758K  $\rightarrow$  +1124L  $\rightarrow$  +86K  
 (2, 6, 1): 30K  $\rightarrow$  +28L  $\rightarrow$  +12K  
 (2, 5, 1): 207K  $\rightarrow$  +329L  $\rightarrow$  +84K  
 (6, 3, 1): 28K  $\rightarrow$  +20L  $\rightarrow$  +6K  
 (6, 2, 1): 34K  $\rightarrow$  +28L  $\rightarrow$  +8K  
 (5, 2, 1): 274K  $\rightarrow$  +346L  $\rightarrow$  +20K  
 (6, 4, 1): 18K  $\rightarrow$  +10L  $\rightarrow$  +1K  
 (5, 5, 1): 41K  $\rightarrow$  +30L  $\rightarrow$  +6K  
 (4, 6, 1): 16K  $\rightarrow$  +10L  $\rightarrow$  +3K  
 (2, 3, 3): 2294K  $\rightarrow$  +3584L  $\rightarrow$  +221K  $\rightarrow$  +123L  $\rightarrow$  +35K  
 (3, 2, 3): 2331K  $\rightarrow$  +3603L  $\rightarrow$  +191K  $\rightarrow$  +106L  $\rightarrow$  +30K  
 (3, 3, 2): 2294K  $\rightarrow$  +3584L  $\rightarrow$  +221K  $\rightarrow$  +123L  $\rightarrow$  +35K  
 (2, 4, 2): 1246K  $\rightarrow$  +2041L  $\rightarrow$  +273K  $\rightarrow$  +111L  $\rightarrow$  +23K  
 (4, 2, 2): 1431K  $\rightarrow$  +2111L  $\rightarrow$  +98K  
 (2, 2, 3): 1095K  $\rightarrow$  +1967L  $\rightarrow$  +47K  
 (3, 3, 1): 616K  $\rightarrow$  +1091L  $\rightarrow$  +32K  
 (2, 4, 1): 353K  $\rightarrow$  +636L  $\rightarrow$  +49K  
 (4, 2, 1): 389K  $\rightarrow$  +652L  $\rightarrow$  +17K  
 (2, 3, 2): 1056K  $\rightarrow$  +1950L  $\rightarrow$  +81K  
 (3, 2, 2): 1095K  $\rightarrow$  +1967L  $\rightarrow$  +47K  
 (1, 5, 3): 230K  $\rightarrow$  +280L  $\rightarrow$  +63K  
 (5, 1, 3): 273K  $\rightarrow$  +291L  $\rightarrow$  +23K  
 (6, 5, 2): 3K  $\rightarrow$  +1L  $\rightarrow$  +0K  
 (5, 6, 2): 3K  $\rightarrow$  +1L  $\rightarrow$  +0K  
 (1, 6, 2): 30K  $\rightarrow$  +28L  $\rightarrow$  +12K  
 (6, 1, 2): 34K  $\rightarrow$  +28L  $\rightarrow$  +8K  
 (1, 6, 3): 24K  $\rightarrow$  +20L  $\rightarrow$  +10K  
 (6, 1, 3): 28K  $\rightarrow$  +20L  $\rightarrow$  +6K  
 (1, 5, 2): 207K  $\rightarrow$  +329L  $\rightarrow$  +84K  
 (5, 1, 2): 276K  $\rightarrow$  +346L  $\rightarrow$  +18K  
 (6, 1, 6): 3K  $\rightarrow$  +1L  $\rightarrow$  +0K  
 (5, 2, 6): 3K  $\rightarrow$  +1L  $\rightarrow$  +0K  
 (4, 3, 6): 3K  $\rightarrow$  +1L  $\rightarrow$  +0K  
 (3, 4, 6): 3K  $\rightarrow$  +1L  $\rightarrow$  +0K  
 (2, 5, 6): 3K  $\rightarrow$  +1L  $\rightarrow$  +0K  
 (1, 6, 6): 3K  $\rightarrow$  +1L  $\rightarrow$  +0K  
 (5, 1, 6): 9K  $\rightarrow$  +4L  $\rightarrow$  +0K  
 (4, 2, 6): 12K  $\rightarrow$  +5L  $\rightarrow$  +0K  
 (3, 3, 6): 12K  $\rightarrow$  +5L  $\rightarrow$  +0K  
 (2, 4, 6): 12K  $\rightarrow$  +5L  $\rightarrow$  +0K  
 (1, 5, 6): 9K  $\rightarrow$  +4L  $\rightarrow$  +0K  
 (6, 1, 5): 9K  $\rightarrow$  +4L  $\rightarrow$  +0K  
 (5, 2, 5): 26K  $\rightarrow$  +14L  $\rightarrow$  +4K  
 (4, 3, 5): 32K  $\rightarrow$  +16L  $\rightarrow$  +4K

(3, 4, 5): 32K -> +16L -> +4K  
(2, 5, 5): 26K -> +14L -> +4K  
(1, 6, 5): 7K -> +4L -> +2K  
(6, 1, 4): 18K -> +10L -> +1K  
(1, 6, 4): 16K -> +10L -> +3K  
(4, 1, 6): 18K -> +10L -> +1K  
(3, 2, 6): 27K -> +14L -> +1K  
(2, 3, 6): 27K -> +14L -> +1K  
(1, 4, 6): 18K -> +10L -> +1K  
(5, 1, 5): 44K -> +30L -> +3K  
(4, 2, 5): 117K -> +81L -> +10K  
(3, 3, 5): 143K -> +98L -> +14K  
(2, 4, 5): 115K -> +81L -> +12K  
(1, 5, 5): 41K -> +30L -> +6K  
(3, 1, 6): 28K -> +20L -> +6K  
(2, 2, 6): 40K -> +27L -> +7K  
(1, 3, 6): 28K -> +20L -> +6K  
(5, 1, 4): 139K -> +118L -> +5K  
(1, 5, 4): 122K -> +116L -> +22K  
(4, 1, 5): 139K -> +118L -> +5K  
(3, 2, 5): 320K -> +280L -> +27K  
(2, 3, 5): 319K -> +280L -> +28K  
(1, 4, 5): 135K -> +118L -> +9K  
(3, 1, 5): 273K -> +291L -> +23K  
(2, 2, 5): 479K -> +522L -> +51K  
(1, 3, 5): 268K -> +291L -> +28K  
(4, 1, 4): 505K -> +564L -> +15K  
(1, 4, 4): 473K -> +558L -> +47K  
(3, 1, 4): 825K -> +1148L -> +22K  
(1, 3, 4): 790K -> +1140L -> +57K  
(2, 1, 5): 276K -> +346L -> +18K  
(1, 2, 5): 274K -> +346L -> +20K  
(4, 1, 3): 825K -> +1148L -> +22K  
(1, 4, 3): 758K -> +1124L -> +86K  
(2, 1, 6): 34K -> +28L -> +8K  
(1, 2, 6): 34K -> +28L -> +8K  
(1, 1, 6): 14K -> +18L -> +13K  
(1, 1, 5): 78K -> +119L -> +13K  
(4, 1, 2): 397K -> +657L -> +9K  
(1, 4, 2): 353K -> +636L -> +49K  
(2, 1, 4): 397K -> +657L -> +9K  
(1, 2, 4): 389K -> +652L -> +17K  
(3, 1, 3): 626K -> +1095L -> +22K  
(1, 3, 3): 616K -> +1091L -> +32K  
(6, 2, 5): 3K -> +1L -> +0K  
(5, 3, 5): 3K -> +1L -> +0K  
(4, 4, 5): 3K -> +1L -> +0K  
(3, 5, 5): 3K -> +1L -> +0K  
(2, 6, 5): 3K -> +1L -> +0K  
(6, 1, 1): 14K -> +18L -> +13K  
(1, 6, 1): 12K -> +18L -> +15K  
(5, 1, 1): 78K -> +119L -> +13K  
(1, 5, 1): 40K -> +91L -> +48K  
(6, 6, 1): 3K -> +1L -> +0K  
(6, 5, 1): 9K -> +4L -> +0K  
(5, 6, 1): 7K -> +4L -> +2K

We observe that four tri-differential orders, namely  $(2, 4, 2)$ ,  $(3, 3, 2)$ ,  $(2, 3, 3)$ ,  $(3, 2, 3)$ , require the use of Leibniz graphs from the 1st layer of neighbors for a solution to the factorization problem to appear.

The Leibniz graphs are now stored in `data/affine_assoc7_leibniz.txt` and their found coefficients are in `data/affine_assoc7_leibniz_coeffs.txt`; together, these form a *certificate* of the affine  $\star$ -product associativity up to  $\bar{o}(\hbar^7)$ .

Given the data files with the coefficients and graphs in the Leibniz graph factorization, the associativity up to  $\bar{o}(\hbar^7)$  can be verified instantly and directly:

```
[65]: affine_assoc7_leibniz_coeffs = vector(SR, open('data/affine_assoc7_leibniz_coeffs.
      ↪txt').readlines())
```

```
[68]: affine_assoc7_leibniz_graphs = [FormalityGraph(3,6,sage_eval(line)) for line in
      ↪open('data/affine_assoc7_leibniz.txt').readlines()]
```

```
[69]: affine_assoc7_leibniz = FGC([(c,L) for c, L in zip(affine_assoc7_leibniz_coeffs,
      ↪affine_assoc7_leibniz_graphs)])
```

```
[70]: from gcaops.graph.leibniz_graph_expansion import
      ↪leibniz_graph_sum_to_kontsevich_graph_sum
      affine_assoc7_leibniz_expanded =
      ↪leibniz_graph_sum_to_kontsevich_graph_sum(affine_assoc7_leibniz,
      ↪max_aerial_in_degree=1)
```

```
[71]: affine_assoc7_leibniz_expanded == affine_assoc7_7
```

```
[71]: True
```

### 3.7.9 Rationality of $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$

It is known from Ben Amar [3] that the Kontsevich  $\star$ -product formula for the duals of Lie algebras  $\mathfrak{g}^*$  contains only *rational* coefficients which are known explicitly, expressed in terms of Bernoulli numbers and factorials. Specifically, all coefficients are determined by the weights of the following graphs [4]:

- The Bernoulli graphs:

```
[1]: def bernoulli_graph(n):
      ↪return FormalityGraph(2, n, [(2,0),(2,1)] + sum([(k+3, k+2),(k+3, 1)] for k in
      ↪range(n-1)], [])
```

The weights of the Bernoulli graphs are  $B_k/(k!)^2$ :

```
[4]: for n in range(1,8):
      ↪B = bernoulli_graph(n)
      ↪B_sign, B_normal = list(FGC(B))[0]
      ↪B_coeff = sum([B_sign*c for c,g in affine_star7 if g == B_normal], 0)
      ↪W = B_coeff/B.multiplicity()
      ↪print(W, bernoulli(n)/factorial(n)^2)
```

```

1/2 -1/2
1/24 1/24
0 0
-1/17280 -1/17280
0 0
1/21772800 1/21772800
0 0

```

- The wheel graphs:

```

[5]: def wheel_graph(n):
      return FormalityGraph(2, n, [(2,0),(2,1+n)] + sum([(k+3,k+2),(k+3,1)] for k in
      ↪range(n-1)], [])

```

The weights of the wheel graphs are  $\frac{1}{2}B_k/(k!)^2$ :

```

[6]: for n in range(2,8):
      W = wheel_graph(n)
      W_sign, W_normal = list(FGC(W))[0]
      W_coeff = sum([W_sign*c for c,g in affine_star7 if g == W_normal], 0)
      Wt = W_coeff/W.multiplicity()
      print(Wt, 1/2*bernoulli(n)/factorial(n)^2)

```

```

1/48 1/48
0 0
-1/34560 -1/34560
0 0
1/43545600 1/43545600
0 0

```

---

We now observe that the Kontsevich weights of some graphs in  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$  contain the conjecturally irrational number  $\zeta(3)^2/\pi^6$  with a nonzero rational coefficient.

While this may seem at odds with the fact of rationality of coefficients appearing in the expansion, we prove here that these claims are in fact consistent. We express the part of  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$  proportional to  $\zeta(3)^2/\pi^6$  as a sum of Leibniz graphs, proving that this part makes a zero contribution when evaluated at a generic affine Poisson structure.

```

[72]: def zetapart(c):
      return QQ['zzz'](str(SR(c).expand()).replace('zeta(3)^2/pi^6', 'zzz')).
      ↪monomial_coefficient(QQ['zzz'].gen())

```

```

[73]: affine_star7_zetapart = affine_star7.map_coefficients(zetapart)

```

```

[74]: for diff_order in affine_star7_zetapart.differential_orders():
      print(diff_order, end=': ', flush=True)
      part = affine_star7_zetapart.part_of_differential_order(diff_order)
      part_Leibniz = kontsevich_graph_sum_to_leibniz_graph_sum(part,
      ↪max_aerial_in_degree=1, verbose=True)
      print(leibniz_graph_sum_to_kontsevich_graph_sum(part_Leibniz,
      ↪max_aerial_in_degree=1) == part)

```

```

(3, 4): 119K -> +179L -> +3K
True
(4, 3): 119K -> +179L -> +3K
True
(2, 4): 19K -> +24L -> +4K
True
(4, 2): 19K -> +24L -> +4K
True
(3, 3): 40K -> +54L -> +1K
True
(4, 5): 22K -> +24L -> +2K
True
(5, 4): 22K -> +24L -> +2K
True
(4, 4): 106K -> +141L -> +4K
True
(3, 5): 62K -> +76L -> +7K
True
(5, 3): 62K -> +76L -> +7K
True
(2, 5): 28K -> +41L -> +6K
True
(5, 2): 28K -> +41L -> +6K
True

```

In conclusion, the number  $\zeta(3)^2/\pi^6$  showed up at  $\hbar^6$  and  $\hbar^7$  in the coefficients of Kontsevich's graphs at 12 bi-differential orders; in all the cases those Kontsevich graphs with their rational coefficients standing near  $\zeta(3)^2/\pi^6$  themselves—i.e. not needing higher layers of Leibniz graph neighbors—assimilate into Leibniz graphs, so that whenever the graph formula  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$  is restricted to any affine Poisson bracket, the entire analytic expression near  $\zeta(3)^2/\pi^6$  vanishes identically, by force of the Jacobi identity and its differential consequences.

---

In fact there are more terms in the affine  $\star$ -product that can be eliminated, by assimilating them into Leibniz graphs. This process of elimination results in a reduced graph expansion  $\star_{\text{aff}}^{\text{red}} \bmod \bar{o}(\hbar^7)$  with only rational coefficients, which is reported in Appendix C.2.

```

[75]: reduced_affine_star7 = FGC.zero()
for diff_order in affine_star7.differential_orders():
    #print(diff_order, end=': ', flush=True)
    part = affine_star7.part_of_differential_order(diff_order)
    part_Leibniz = kontsevich_graph_sum_to_leibniz_graph_sum(part,
    ↪max_aerial_in_degree=1, force_saturation=True, exact=False, verbose=True)
    if part_Leibniz is not None:
        # NOTE: by having passed exact=False, we get a (least squares?) "solution"
        part_Leibniz_expanded =
    ↪leibniz_graph_sum_to_kontsevich_graph_sum(part_Leibniz, max_aerial_in_degree=1)
        new_part = part - part_Leibniz_expanded
        reduced_affine_star7 += new_part
        #print(len(part), 'down to', len(new_part))
    else:
        #print('no reduction')

```

```
reduced_affine_star7 += part
```

We write the reduced star product to a file:

```
[ ]: #with open('data/affine_star7_reduced.txt', 'w') as f:
#     f.write(reduced_affine_star7.kgs_encoding())
```

We see at once that the reduced affine star product formula (in particular, having its  $\zeta(3)^2/\pi^6$ -slice equal to zero) is much shorter: at all orders up to  $\hbar^7$ , there remain only 326 terms:

```
[7]: affine_star7_reduced = FGC.element_from_kgs_encoding(open('data/affine_star7_reduced.
↳txt').read().rstrip())
```

```
[8]: len(affine_star7_reduced)
```

```
[8]: 326
```

We calculate the associator:

```
[9]: %time affine_assoc7_reduced = affine_star7_reduced.insertion(0, affine_star7_reduced,
↳max_num_aerial=7, max_aerial_in_degree=1) - affine_star7_reduced.insertion(1,
↳affine_star7_reduced, max_num_aerial=7, max_aerial_in_degree=1)
```

```
CPU times: user 49.7 s, sys: 0 ns, total: 49.7 s
```

```
Wall time: 49.3 s
```

```
[10]: len(affine_assoc7_reduced)
```

```
[10]: 29371
```

Again, the associator becomes twice smaller: there were 59905 Kontsevich graphs showing up at all orders in the associator for  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$ .

```
[11]: len(list(affine_assoc7_reduced.differential_orders()))
```

```
[11]: 181
```

This is an overall count of tri-differential orders; specifically at  $\hbar^7$ , there are 161 (that is, discarding the  $\zeta(3)^2/\pi^6$ -slice does not decrease the number).

In Appendix C.3 we inspect that the reduced star product  $\star_{\text{aff}}^{\text{red}} \bmod \bar{o}(\hbar^7)$  remains associative.

We conclude that the Kontsevich graph expansion of the reduced affine  $\star$ -product itself is associative up to  $\bar{o}(\hbar^7)$ ; the analytic formula which one writes for  $f \star_{\text{aff}}^{\text{red}} g \bmod \bar{o}(\hbar^7)$  with arbitrary arguments  $f, g \in C^\infty(M)$  and any affine Poisson structure  $P$  in  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$  is, we establish in this section, identically equal to the formula  $f \star_{\text{aff}} g \bmod \bar{o}(\hbar^7)$ : all the coefficients of differential polynomials in  $f, g$  and  $P^{ij}$  are rational numbers, and  $\zeta(3)^2/\pi^6$  is not met at all.



# Chapter 4

## Implementation of the graph complex

This chapter is an introduction to the graph operad and graph complex. By using examples we illustrate how to manipulate all the objects and structures (such as graphs, sums of graphs, and operadic insertions of graphs) with the new software. We now learn that graphs with a wedge ordering of edges form a vector space. Some graphs are zero graphs, whenever they admit an automorphism which induces an odd permutation of edges. In the quotient space modulo zero graphs, we (can) choose a basis so that (non)zero graphs are conveniently represented as vectors. We refer to [26, 27] or [17], [32] and [33] for general theory and more details.

### 4.1 Graphs

We import the implementation of undirected graphs:

```
[1]: from gcaops.graph.undirected_graph import UndirectedGraph
```

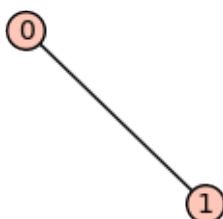
We consider undirected graphs with  $n$  vertices labeled  $0, \dots, n - 1$ , and an *ordered* list of edges between vertices. An often convenient/preferred ordering of the edges is the lexicographic ordering.

The stick on  $n = 2$  vertices labeled 0 and 1, and one edge  $(0, 1)$ :

```
[2]: stick_graph = UndirectedGraph(2, [(0,1)]); stick_graph
```

```
[2]: UndirectedGraph(2, [(0, 1)])
```

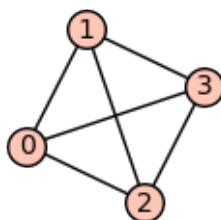
```
[3]: stick_graph.show(figsize=2)
```



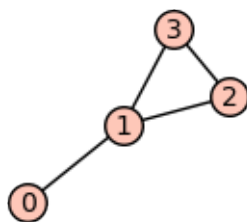
```
[4]: tetrahedron_graph = UndirectedGraph(4, [(0,1),(0,2),(0,3),(1,2),(1,3),(2,3)]);
      ↪ tetrahedron_graph
```

```
[4]: UndirectedGraph(4, [(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)])
```

```
[5]: tetrahedron_graph.show(figsize=2)
```



```
[6]: g = UndirectedGraph(4, [(0,1),(1,2),(1,3),(2,3)]); g.show(figsize=2)
```



Relabeling of the graph's vertices by a graph automorphism induces a permutation on the set of edges.

For example, let us relabel two vertices in the above graph  $g$ .

```
[7]: g.relabeled({0:0, 1:1, 2:3, 3:2}).edges()
```

```
[7]: [(0, 1), (1, 3), (1, 2), (2, 3)]
```

This permutation on the set of edges has a parity:

```
[8]: g.relabeled({0:0, 1:1, 2:3, 3:2}).canonicalize_edges()
```

```
[8]: -1
```

Indeed, the edge permutation swaps the two edges  $(1, 2)$  and  $(1, 3)$ ; hence it is parity odd.

## 4.2 Graph operad

We import the implementation of the operad of undirected graphs:

```
[9]: from gcaops.graph.undirected_graph_operad import UndirectedGraphOperad
```

The operad is a graded vector space spanned by undirected graphs, quotiented by graded edge permutations.

```
[10]: Gra = UndirectedGraphOperad(QQ); Gra
```

```
[10]: Operad of undirected graphs over Rational Field
```

Convert a graph into an element of `Gra`:

```
[11]: stick = Gra(stick_graph); stick
```

```
[11]: 1*UndirectedGraph(2, [(0, 1)])
```

Create a sum of graphs from a list of (coefficient, graph) pairs:

```
[12]: Gra([(1, stick_graph), (4, tetrahedron_graph)])
```

```
[12]: 1*UndirectedGraph(2, [(0, 1)]) + (4)*UndirectedGraph(4, [(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)])
```

We can insert a given graph into a vertex of another graph; the result is a sum of terms — to get all of them, we reattach the edges which were incident to that vertex, now attaching them (independently one from another) to the vertices of the inserted graph. We do this in all possible ways, and take the sum. In every resulting term, the edges of the inserted graph go last; first go the edges of the graph into which the former was inserted.

Here is an example.

```
[13]: stick.insertion(0, stick)
```

```
[13]: 0
```

Indeed, all terms in the result are proportional to the following graph:

```
[14]: stickstick = UndirectedGraph(3,[(0,1),(1,2)]); stickstick
```

```
[14]: UndirectedGraph(3, [(0, 1), (1, 2)])
```

But this graph is zero:

```
[15]: Gra(stickstick)
```

```
[15]: 0
```

Indeed, this is because the graph `stickstick` has an automorphism that induces an odd permutation on the set of edges.

```
[16]: stickstick.relabeled({0:2, 1:1, 2:0}).canonicalize_edges()
```

[16]: -1

Equal to minus itself, the graph is zero.

### 4.3 Full undirected graph complex

We import the implementation of the undirected graph complex:

```
[17]: from gcaops.graph.undirected_graph_complex import UndirectedGraphComplex
```

Define the full undirected graph complex over  $\mathbb{Q}$ :

```
[18]: fGC = UndirectedGraphComplex(QQ); fGC
```

[18]: Undirected graph complex over Rational Field with Basis consisting of representatives of isomorphism classes of undirected graphs with no automorphisms that induce an odd permutation on edges

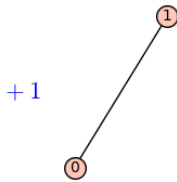
For the time being, we do not impose any restrictions on the graphs (such as “connected”, “biconnected”, etc.) or on the vertex degrees (e.g. “at least 3”).

Convert the stick graph to an element of the graph complex:

```
[19]: stick = fGC(stick_graph); stick
```

```
[19]: 1*UndirectedGraph(2, [(0, 1)])
```

```
[20]: stick.show()
```



The graded Lie bracket of graphs is the graded commutator of insertions  $[a, b] = a \overleftarrow{\circ} b - (-)^{\#E(a) \cdot E(b)} a \overrightarrow{\circ} b$ . We recall that the reverse order of terms ( $a \overrightarrow{\circ} b$  preceding  $a \overleftarrow{\circ} b$ ) is also used in the literature.

```
[21]: stick.bracket(stick)
```

```
[21]: 0
```

We now define the vertex-expanding differential: this operator is the graded Lie bracket with the stick graph as the first argument,  $d = [\bullet\text{---}\bullet, \cdot]$ .

```
[22]: stick.differential()
```

```
[22]: 0
```

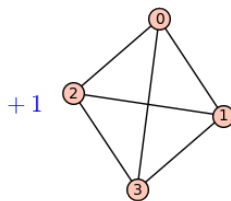
We now construct some graph cocycles manually:

- The tetrahedron (3-wheel) cocycle  $\gamma_3$  (on 4 vertices and 6 edges):

```
[23]: tetrahedron_cocycle = fGC(tetrahedron_graph); tetrahedron_cocycle
```

```
[23]: 1*UndirectedGraph(4, [(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)])
```

```
[24]: tetrahedron_cocycle.show()
```



```
[25]: tetrahedron_cocycle.differential()
```

```
[25]: 0
```

- The 5-wheel cocycle  $\gamma_5$  (on 6 vertices and 10 edges in each term):

```
[26]: fivewheel_graph = UndirectedGraph(6,
↳ [(0,1), (1,2), (2,3), (3,4), (0,4), (0,5), (1,5), (2,5), (3,5), (4,5)])
fivewheel = fGC(fivewheel_graph)
fivewheel.differential()
```

```
[26]: (-10)*UndirectedGraph(7, [(0, 1), (0, 4), (0, 5), (1, 3), (1, 5), (2, 3), (2, 4), (2,
6), (3, 6), (4, 6), (5, 6)])
```

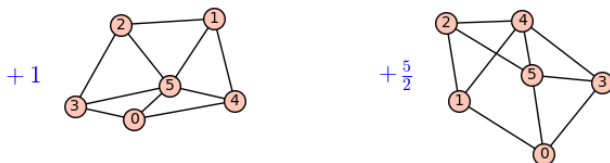
```
[27]: roof_graph = UndirectedGraph(6,
↳ [(0,1), (1,2), (2,3), (0,3), (3,4), (0,4), (4,5), (2,5), (1,5), (0,2)])
roof = fGC(roof_graph)
roof.differential()
```

```
[27]: (4)*UndirectedGraph(7, [(0, 1), (0, 4), (0, 5), (1, 3), (1, 5), (2, 3), (2, 4), (2,
6), (3, 6), (4, 6), (5, 6)])
```

```
[28]: fivewheel_cocycle = fivewheel + (5/2)*roof
fivewheel_cocycle
```

```
[28]: 1*UndirectedGraph(6, [(0, 3), (0, 4), (0, 5), (1, 2), (1, 4), (1, 5), (2, 3), (2, 5),
(3, 5), (4, 5)]) + (5/2)*UndirectedGraph(6, [(0, 1), (0, 3), (0, 5), (1, 2), (1, 4),
(2, 4), (2, 5), (3, 4), (3, 5), (4, 5)])
```

```
[29]: fivewheel_cocycle.show()
```



```
[30]: fivewheel_cocycle.differential()
```

```
[30]: 0
```

We can also create a graph cochain from a list of (coefficient, graph) tuples:

```
[31]: fGC([(1, fivewheel_graph), (5/2, roof_graph)])
```

```
[31]: 1*UndirectedGraph(6, [(0, 3), (0, 4), (0, 5), (1, 2), (1, 4), (1, 5), (2, 3), (2, 5),
(3, 5), (4, 5)]) + (5/2)*UndirectedGraph(6, [(0, 1), (0, 3), (0, 5), (1, 2), (1, 4),
(2, 4), (2, 5), (3, 4), (3, 5), (4, 5)])
```

Whenever we have a graph cochain (no matter how it was constructed) we can iterate over its terms:

```
[32]: for (c,g) in fivewheel_cocycle:
      print(c, g)
```

```
1 UndirectedGraph(6, [(0, 3), (0, 4), (0, 5), (1, 2), (1, 4), (1, 5), (2, 3), (2, 5),
(3, 5), (4, 5)])
5/2 UndirectedGraph(6, [(0, 1), (0, 3), (0, 5), (1, 2), (1, 4), (2, 4), (2, 5), (3,
4), (3, 5), (4, 5)])
```

```
[33]: set([c for (c,g) in fivewheel_cocycle])
```

```
[33]: {1, 5/2}
```

## 4.4 Graph bases: storing them in cache

The graph complex implementation stores a choice of basis internally. The basis can be accessed as follows:

```
[34]: list(fGC.basis().graphs(4,6))
```

```
[34]: [UndirectedGraph(4, [(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)])]
```

```
[35]: list(fGC.basis().graphs(6,10))
```

```
[35]: [UndirectedGraph(6, [(0, 4), (0, 5), (1, 4), (1, 5), (2, 3), (2, 4), (2, 5), (3, 4),
(3, 5), (4, 5)]),
UndirectedGraph(6, [(0, 4), (0, 5), (1, 3), (1, 5), (2, 3), (2, 4), (2, 5), (3, 4),
(3, 5), (4, 5)])]
```

```

UndirectedGraph(6, [(0, 3), (0, 5), (1, 2), (1, 4), (1, 5), (2, 4), (2, 5), (3, 4),
(3, 5), (4, 5)]),
UndirectedGraph(6, [(0, 3), (0, 4), (0, 5), (1, 2), (1, 4), (1, 5), (2, 3), (2, 5),
(3, 5), (4, 5)]),
UndirectedGraph(6, [(0, 1), (0, 3), (0, 5), (1, 2), (1, 4), (2, 4), (2, 5), (3, 4),
(3, 5), (4, 5)]),
UndirectedGraph(6, [(0, 4), (0, 5), (1, 2), (1, 3), (1, 5), (2, 3), (2, 4), (3, 4),
(3, 5), (4, 5)])]

```

Instead of listing too many graphs, let us now just count the dimensions of vector spaces. It can be noticed that we deal primarily with the spaces of undirected graphs on  $n$  vertices and  $2n - 2$  edges; they will be used to construct Poisson 2-cocycles.

```
[36]: len(fGC.basis().graphs(7,12))
```

```
[36]: 48
```

```
[37]: len(fGC.basis().graphs(8,14))
```

```
[37]: 1006
```

By default, these bases are generated anew (whenever needed) in each SageMath or Python session. To avoid generating these bases over and over, it is possible to use a cache on disk. Making use of this functionality is achieved by specifying a directory:

```
[38]: #from gcaops.graph import graph_cache
#graph_cache.GCAOPS_DATA_DIR = '/home/rburing/src/gcaops_data/'
```

Whenever a graph basis is accessed (e.g. as above) while this directory is specified, the program inspects first whether the needed basis is already stored in the directory. If so, it simply returns a reference to it. If not, the program generates the basis, stores it there for future (re)use, and returns a reference to it.

## 4.5 Undirected graph complex

To find interesting cohomology classes it suffices to restrict to a subcomplex spanned by graphs which are connected, biconnected, and in which each vertex has degree  $\geq 3$ . Graphs are stored as collections of vectors (one for each bi-graded component) and the differentials (restricted to each bi-graded component) are stored as matrices. This will allow to find a basis of cohomology automatically.

```
[39]: GC = UndirectedGraphComplex(QQ, connected=True, biconnected=True, min_degree=3,
->implementation='vector'); GC
```

```
[39]: Undirected graph complex over Rational Field with Basis consisting of representatives
of isomorphism classes of undirected graphs (connected, biconnected, of degree at
least 3) with no automorphisms that induce an odd permutation on edges
```

We can convert graph cocycles from the full graph complex `fGC` to this restricted graph complex `GC`.

```
[40]: GC(tetrahedron_cocycle)
```

```
[40]: 1*UndirectedGraph(4, [(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)])
```

Now, with this implementation, we can test whether a cocycle is a coboundary:

```
[41]: GC(tetrahedron_cocycle).is_coboundary()
```

```
[41]: False
```

```
[42]: GC(fivewheel_cocycle).is_coboundary()
```

```
[42]: False
```

## 4.6 Directed graph complex

We also have an implementation of directed graphs:

```
[43]: from gcaops.graph.directed_graph import DirectedGraph
```

```
[44]: directed_tetra = DirectedGraph(4, [(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)]);
      ↪ directed_tetra
```

```
[44]: DirectedGraph(4, [(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)])
```

We import the implementation of the directed graph complex:

```
[45]: from gcaops.graph.directed_graph_complex import DirectedGraphComplex
```

Define the directed graph complex over  $\mathbb{Q}$ ; the graphs are conveniently restricted to get rid of graphs with 2-cycles, etc.

```
[46]: dGC = DirectedGraphComplex(QQ, connected=True, biconnected=True, min_degree=3,
      ↪ loops=False, implementation='vector', sparse=True); dGC
```

```
[46]: Directed graph complex over Rational Field with Basis consisting of representatives of
isomorphism classes of directed graphs (connected, biconnected, of degree at least 3,
without loops) with no automorphisms that induce an odd permutation on edges
```

A cochain of `dGC` can be defined by a list of (coefficient, graph) pairs:

```
[47]: dGC([(1, directed_tetra)])
```

```
[47]: 1*DirectedGraph(4, [(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)])
```

The canonical map from the undirected graph complex to the directed graph complex works as follows: every edge, independently from all others (if any), is directed consecutively in two opposite ways (this creates a sum of directed graphs). The edge ordering is inherited from the undirected graph complex; this is important when we collect similar terms.

This map is implemented as conversion (e.g. here from `fGC` to `dGC`):

```
[48]: tetrahedron_cocycle
```

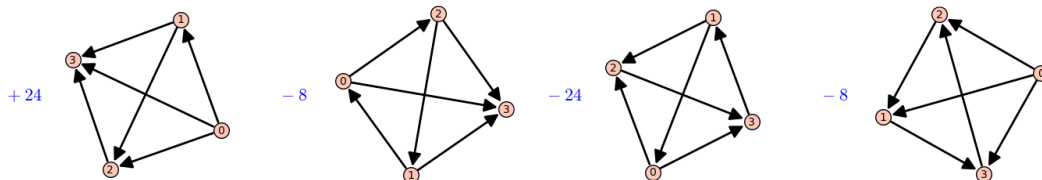
```
[48]: 1*UndirectedGraph(4, [(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)])
```



```
[49]: tetrahedron_cocycle_directed = dGC(tetrahedron_cocycle); tetrahedron_cocycle_directed
```

```
[49]: (24)*DirectedGraph(4, [(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)]) +
(-8)*DirectedGraph(4, [(0, 2), (0, 3), (1, 0), (1, 3), (2, 1), (2, 3)]) +
(-24)*DirectedGraph(4, [(0, 2), (0, 3), (1, 0), (1, 2), (2, 3), (3, 1)]) +
(-8)*DirectedGraph(4, [(0, 1), (0, 2), (0, 3), (1, 3), (2, 1), (3, 2)])
```

```
[50]: tetrahedron_cocycle_directed.show(ncols=4)
```



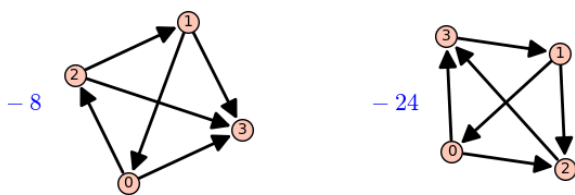
The differential in the directed graph complex is the graded Lie bracket with the sum of two ways to direct the stick:  $d = [\leftarrow \bullet + \bullet \rightarrow, \cdot]$ :

```
[51]: tetrahedron_cocycle_directed.differential()
```

```
[51]: 0
```

For the purpose of future use, we can filter directed graphs by prescribing a bound upon the outgoing degree of vertices.

```
[52]: tetrahedron_cocycle_directed_filtered = tetrahedron_cocycle_directed.
↳filter(max_out_degree=2)
tetrahedron_cocycle_directed_filtered.show()
```



As usual, let us count the dimensions; because the suitable bases are already stored on the disk, it takes milliseconds, microseconds, and even nanoseconds to produce them!

```
[53]: %time len(dGC.basis().graphs(6,10))
```

```
CPU times: user 0 ns, sys: 1.8 ms, total: 1.8 ms
Wall time: 4.83 ms
```

```
[53]: 732
```

```
[54]: %time len(dGC.basis().graphs(7,12))
```

```
CPU times: user 1.55 ms, sys: 156 µs, total: 1.7 ms
Wall time: 10.2 ms
```

```
[54]: 25638
```

```
[55]: %time len(dGC.basis().graphs(8,14))
```

```
CPU times: user 6.23 ms, sys: 20.1 ms, total: 26.3 ms
Wall time: 57.5 ms
```

```
[55]: 1126232
```

```
[56]: %time len(dGC.basis().graphs(9,16))
```

```
CPU times: user 243 ms, sys: 1.1 s, total: 1.34 s
Wall time: 2.5 s
```

```
[56]: 59381077
```

```
[57]: %time len(dGC.basis().graphs(9,15))
```

```
CPU times: user 18.3 ms, sys: 110 ms, total: 128 ms
Wall time: 273 ms
```

```
[57]: 5247208
```

## 4.7 Undirected graph operations

From Chapter 2 we recall the construction of the superfunction algebra.

```
[58]: from gcaops.algebra.superfunction_algebra import SuperfunctionAlgebra
SA.<xi1,xi2,xi3> = SuperfunctionAlgebra(SR, var('x1,x2,x3'), simplify='expand',
↳is_zero='is_trivial_zero'); SA
```

```
[58]: Superfunction algebra over Symbolic Ring with even coordinates (x1, x2, x3) and odd
coordinates (xi1, xi2, xi3)
```

Take some bivector field:

```
[59]: P = (x1^2-x2*x3)*(x1*xi2*xi3 + x2*xi3*xi1 + x3*xi1*xi2); P
```

```
[59]: (x1^3 - x1*x2*x3)*xi2*xi3 + (-x1^2*x2 + x2^2*x3)*xi1*xi3 + (x1^2*x3 - x2*x3^2)*xi1*xi2
```

And let there be a vector field:

```
[60]: X = x1*xi1
```

Let us calculate their Schouten bracket:

```
[61]: P.bracket(X)
```

```
[61]: (x1^2*x2 + x2^2*x3)*xi1*xi3 + (-x1^2*x3 - x2*x3^2)*xi1*xi2 + (-3*x1^3 +
x1*x2*x3)*xi2*xi3
```

Kontsevich's graph calculus tells us that the Schouten bracket  $\pi_S(A, B) = (-)^{|A|-1}[[A, B]]$  comes from the stick  $\bullet \dashrightarrow \bullet$ .

Let us illustrate this (see the seminal paper [27] for key facts and statements):

```
[62]: stick
```

```
[62]: 1*UndirectedGraph(2, [(0, 1)])
```

```
[63]: schouten = SA.graph_operation(stick); schouten
```

```
[63]: Symmetric operation of arity 2 and degree -1 on Superfunction algebra over Symbolic
Ring with even coordinates (x1, x2, x3) and odd coordinates (xi1, xi2, xi3)
```

```
[64]: schouten(P, X) == -P.bracket(X)
```

```
[64]: True
```

```
[65]: schouten(X, P) == schouten(P,X)
```

```
[65]: True
```

The tetrahedral flow on the space of Poisson bi-vectors comes from the tetrahedron graph cocycle. Namely, when directed (and filtered for the outgoing vertex degrees  $\leq 2$ ) the tetrahedral graph becomes a polylinear operation on the space of multivectors (in particular, of bivectors). This is how the Kontsevich graph flow is built:  $\dot{P} = \text{Or}(\gamma_3)(P^{\otimes n})$ ; see Chapter 17. This is precisely the formula of Kontsevich's tetrahedral graph flow which we use in the entire dissertation: for instance we have seen it in Chapter 2, and we shall use it again in Chapters 5, 6, 7, 8, 9.

```
[66]: tetrahedron_operation = SA.graph_operation(tetrahedron_cocycle); tetrahedron_operation
```

```
[66]: Operation of arity 4 and degree -6 on Superfunction algebra over Symbolic Ring with
even coordinates (x1, x2, x3) and odd coordinates (xi1, xi2, xi3)
```

```
[67]: %time Q_tetra = tetrahedron_operation(P,P,P,P); Q_tetra
```

```
CPU times: user 2.01 s, sys: 21.5 ms, total: 2.03 s
```

```
Wall time: 2.03 s
```

```
[67]: (288*x1^4*x2^2 - 288*x1^2*x2^3*x3 - 288*x1^4*x3^2 + 288*x1^2*x2*x3^3)*xi2*xi3 +
(-288*x1^3*x2^3 + 288*x1*x2^4*x3 + 288*x1^3*x2*x3^2 - 288*x1*x2^2*x3^3)*xi1*xi3 +
(288*x1^3*x2^2*x3 - 288*x1*x2^3*x3^2 - 288*x1^3*x3^3 + 288*x1*x2*x3^4)*xi1*xi2
```

Operations on the space of multivectors can also be defined using directed graph cocycles:

```
[68]: tetrahedron_operation_directed = SA.graph_operation(tetrahedron_cocycle_directed)
tetrahedron_operation_directed(P,P,P,P)
```

$$\begin{aligned}
[68]: & (-288*x1^3*x2^3 + 288*x1*x2^4*x3 + 288*x1^3*x2*x3^2 - 288*x1*x2^2*x3^3)*xi1*xi3 + \\
& (288*x1^3*x2^2*x3 - 288*x1*x2^3*x3^2 - 288*x1^3*x3^3 + 288*x1*x2*x3^4)*xi1*xi2 + \\
& (288*x1^4*x2^2 - 288*x1^2*x2^3*x3 - 288*x1^4*x3^2 + 288*x1^2*x2*x3^3)*xi2*xi3
\end{aligned}$$

The result is the same as in the previous cell; this is the tetrahedral graph flow on the space of bivectors.

# Chapter 5

## Examples of graph cocycles

In this chapter we find explicit representatives of some graph cocycles in low degrees, which will be necessary to compute the action on Poisson structures in the following chapters. A theorem of Willwacher [35] shows that the degree 0 cohomology of the graph complex is isomorphic to the Grothendieck–Teichmüller Lie algebra `grt`, and in particular for each odd  $n$  there exists a nontrivial graph cohomology class containing the wheel graph with  $n$  spokes with a nonzero coefficient. Moreover, the isomorphism to `grt` combined with a theorem of F. Brown [7] shows that these graph cohomology classes generate a *free* Lie subalgebra in the cohomology of the graph complex. It is an open problem whether these cohomology classes are all of the graph cohomology classes (the Deligne–Drinfeld conjecture states that they are). The dimensions of (finite-dimensional) graded parts of graph cohomology can be found using rank computations (Willwacher–Živković [38], partly numerical). Using linear algebra, we find bases of the graded parts of graph cohomology. We note that the coefficients of these graph cocycles can also be expressed using integral formulas, as shown by Willwacher–Rossi [37]. A closed form or combinatorial interpretation of these coefficients is an open problem, which provides another reason for listing them explicitly. In addition to finding explicit formulas for graph cocycles, we prove the factorization of the Poisson cocycle condition via the Jacobi identity in each case, by providing the necessary Leibniz graphs.

**Remark.** Of course a cocycle plus a coboundary is again a cocycle, so the bases we find are generally not canonical. But in low degrees there are also not that many coboundaries, so we list some of them as well. This will also be instructive when calculating the action on Poisson structures in the next chapters.

We import the relevant functionality from the `gcaops` package:

```
[1]: from gcaops.graph.undirected_graph import UndirectedGraph
      from gcaops.graph.undirected_graph_complex import UndirectedGraphComplex
      from gcaops.graph.directed_graph import DirectedGraph
      from gcaops.graph.directed_graph_complex import DirectedGraphComplex
      from gcaops.graph.formality_graph import FormalityGraph
      from gcaops.graph.formality_graph_complex import FormalityGraphComplex
```

To find interesting cohomology classes it suffices to restrict to a subcomplex spanned by graphs which are connected, biconnected (i.e. which remain connected even when any single vertex is deleted), and in which each vertex has degree  $\geq 3$ . We also switch to a different implementation: first we fix a basis in the bi-graded space of graphs, so that graphs are stored as collections of vectors (one vector for each bi-graded component);

likewise, the differentials (restricted to each bi-graded component) are stored as matrices. This will allow us to find a basis of cohomology classes automatically.

```
[2]: GC = UndirectedGraphComplex(QQ, connected=True, biconnected=True, min_degree=3,
↳ implementation='vector', sparse=True); GC
```

[2]: Undirected graph complex over Rational Field with Basis consisting of representatives of isomorphism classes of undirected graphs (connected, biconnected, of degree at least 3) with no automorphisms that induce an odd permutation on edges

```
[3]: dGC = DirectedGraphComplex(QQ, connected=True, biconnected=True, min_degree=3,
↳ loops=False, implementation='vector', sparse=True); dGC
```

[3]: Directed graph complex over Rational Field with Basis consisting of representatives of isomorphism classes of directed graphs (connected, biconnected, of degree at least 3, without loops) with no automorphisms that induce an odd permutation on edges

```
[4]: FGC = FormalityGraphComplex(QQ, lazy=True)
```

We recall that there is a Lie algebra generated by the wheel graph cocycles; every such generator is nontrivial, it contains with nonzero coefficient a wheel graph with an odd number of spokes.

## 5.1 The tetrahedron cocycle $\gamma_3$

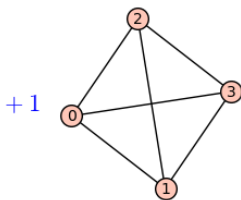
The graph cohomology in the vertex-edge bi-grading (4, 6) is one-dimensional:

```
[5]: len(GC.cohomology_basis(4,6))
```

[5]: 1

The generator is the tetrahedron (3-wheel) cocycle  $\gamma_3$ , the full graph on 4 vertices and 6 edges:

```
[6]: tetrahedron_cocycle = GC.cohomology_basis(4,6)[0]; tetrahedron_cocycle.show()
```



This was the first nontrivial graph cocycle ever found; it was introduced by Kontsevich in his paper [27].

The graph differential which we study here is the vertex-expanding differential; another differential (edge contracting) is also studied by others.

```
[7]: tetrahedron_cocycle.differential()
```

```
[7]: 0
```

```
[8]: tetrahedron_cocycle.is_coboundary()
```

```
[8]: False
```

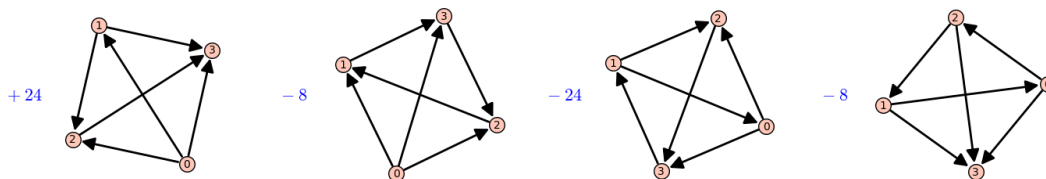
Now we study the directed graph complex with the differential which is the Lie bracket of a graph with the sum of two directed sticks  $\bullet \rightarrow \bullet + \bullet \leftarrow \bullet$ . The edges of tetrahedron oriented in all possible ways, it becomes a directed graph cocycle. The overall ordering of edges is inherited from the undirected graphs.

```
[9]: tetrahedron_cocycle_directed = dGC(tetrahedron_cocycle); tetrahedron_cocycle_directed
```

```
[9]: (24)*DirectedGraph(4, [(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)]) +
(-8)*DirectedGraph(4, [(0, 1), (0, 2), (0, 3), (1, 3), (2, 1), (3, 2)]) +
(-24)*DirectedGraph(4, [(0, 2), (0, 3), (1, 0), (1, 2), (2, 3), (3, 1)]) +
(-8)*DirectedGraph(4, [(0, 2), (0, 3), (1, 0), (1, 3), (2, 1), (2, 3)])
```

In the following picture, the ordering of edges is lexicographic.

```
[10]: tetrahedron_cocycle_directed.show(ncols=4)
```



```
[11]: tetrahedron_cocycle_directed.differential()
```

```
[11]: 0
```

Next, we prove the factorization of the Poisson cocycle condition via the Jacobi identity for the tetrahedral flow.

```
[12]: wedge = FGC(FormalityGraph(2,1,[(2,0),(2,1)])); wedge
```

```
[12]: 1*FormalityGraph(2, 1, [(2, 0), (2, 1)])
```

```
[13]: formality_stick = FGC(DirectedGraph(2,[(0,1)])); formality_stick
```

```
[13]: 1*FormalityGraph(0, 2, [(0, 1)])
```

This is the tetrahedral flow:

```
[14]: Q_tetrahedron = FGC(tetrahedron_cocycle_directed.filter(max_out_degree=2)).
↪attach_to_ground((2,2,2,2))
```

```
[15]: len(Q_tetrahedron)
```

```
[15]: 3
```

The necessary Leibniz graphs are these:

```
[16]: Q_tetrahedron_Leibniz = FGC(tetrahedron_cocycle_directed).attach_to_ground((2,2,2,3))
```

```
[17]: len(Q_tetrahedron_Leibniz)
```

```
[17]: 27
```

Now, the Poisson cocycle condition is verified:

```
[18]: P_Q_tetrahedron = wedge.schouten_bracket(Q_tetrahedron)
```

```
[19]: len(P_Q_tetrahedron)
```

```
[19]: 39
```

```
[20]: Q_tetrahedron_Leibniz_expanded = sum(Q_tetrahedron_Leibniz.insertion(k,
↳formality_stick, max_out_degree=2) for k in [3,4,5,6])
```

```
[21]: len(Q_tetrahedron_Leibniz_expanded)
```

```
[21]: 39
```

```
[22]: P_Q_tetrahedron == -16 * Q_tetrahedron_Leibniz_expanded
```

```
[22]: True
```

## 5.2 The five-wheel cocycle $\gamma_5$

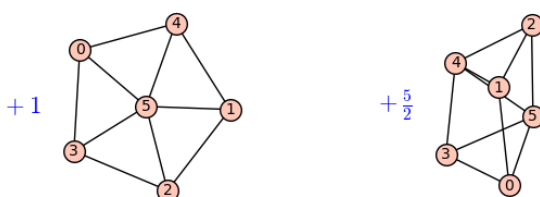
It was found independently by Kontsevich and by Willwacher in the years 1996–2010.

The 5-wheel cocycle  $\gamma_5$  is a sum of two graphs; each of them on 6 vertices and 10 edges:

```
[23]: fivewheel_cocycle = GC.cohomology_basis(6,10)[0]; fivewheel_cocycle
```

```
[23]: 1*UndirectedGraph(6, [(0, 3), (0, 4), (0, 5), (1, 2), (1, 4), (1, 5), (2, 3), (2, 5),
(3, 5), (4, 5)]) + (5/2)*UndirectedGraph(6, [(0, 1), (0, 3), (0, 5), (1, 2), (1, 4),
(2, 4), (2, 5), (3, 4), (3, 5), (4, 5)])
```

```
[24]: fivewheel_cocycle.show()
```





```
[25]: set([c for (c,g) in fivewheel_cocycle])
```

```
[25]: {1, 5/2}
```

```
[26]: fivewheel_cocycle.differential()
```

```
[26]: 0
```

```
[27]: fivewheel_cocycle.is_coboundary()
```

```
[27]: False
```

Let us orient the 5-wheel cocycle:

```
[28]: fivewheel_cocycle_directed = dGC(fivewheel_cocycle) #; fivewheel_cocycle_directed
```

```
[29]: len(fivewheel_cocycle_directed)
```

```
[29]: 616
```

Next, we prove the factorization of the Poisson cocycle condition via the Jacobi identity for the 5-wheel flow.

```
[30]: Q_fivewheel = FGC(fivewheel_cocycle_directed.filter(max_out_degree=2)).
      ↪attach_to_ground((2,2,2,2,2,2)); len(Q_fivewheel)
```

```
[30]: 167
```

```
[31]: Q_fivewheel_Leibniz = FGC(fivewheel_cocycle_directed.filter(max_out_degree=3)).
      ↪attach_to_ground((2,2,2,2,2,3))
```

```
[32]: len(Q_fivewheel_Leibniz)
```

```
[32]: 3876
```

```
[33]: P_Q_fivewheel = wedge.schouten_bracket(Q_fivewheel)
```

```
[34]: len(P_Q_fivewheel)
```

```
[34]: 3495
```

```
[35]: Q_fivewheel_Leibniz_expanded = sum(Q_fivewheel_Leibniz.insertion(k, formality_stick,
      ↪max_out_degree=2) for k in [3,4,5,6,7,8])
```

```
[36]: P_Q_fivewheel == -24 * Q_fivewheel_Leibniz_expanded
```

```
[36]: True
```

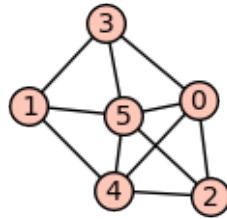
### 5.3 The coboundary $\delta_6 = d(\beta_6)$

The coboundary  $\delta_6$  is a sum of four graphs on 7 vertices and 12 edges:

```
[37]: beta6_graph = UndirectedGraph(6,
↳ [(0,2),(0,3),(0,4),(0,5),(1,3),(1,4),(1,5),(2,4),(2,5),(3,5),(4,5)])
beta6_graph
```

```
[37]: UndirectedGraph(6, [(0, 2), (0, 3), (0, 4), (0, 5), (1, 3), (1, 4), (1, 5), (2, 4),
(2, 5), (3, 5), (4, 5)])
```

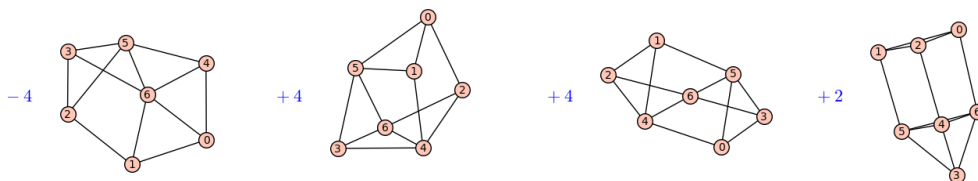
```
[38]: beta6_graph.show(figsize=2)
```



```
[39]: beta6 = GC(beta6_graph)
delta6_cocycle = beta6.differential()
delta6_cocycle
```

```
[39]: (-4)*UndirectedGraph(7, [(0, 1), (0, 4), (0, 6), (1, 2), (1, 6), (2, 3), (2, 5), (3,
5), (3, 6), (4, 5), (4, 6), (5, 6)]) + (4)*UndirectedGraph(7, [(0, 1), (0, 2), (0, 5),
(1, 4), (1, 5), (2, 4), (2, 6), (3, 4), (3, 5), (3, 6), (4, 6), (5, 6)]) +
(4)*UndirectedGraph(7, [(0, 3), (0, 4), (0, 5), (1, 2), (1, 4), (1, 5), (2, 4), (2,
6), (3, 5), (3, 6), (4, 6), (5, 6)]) + (2)*UndirectedGraph(7, [(0, 1), (0, 2), (0, 6),
(1, 2), (1, 5), (2, 4), (3, 4), (3, 5), (3, 6), (4, 5), (4, 6), (5, 6)])
```

```
[40]: delta6_cocycle.show(ncols=4)
```



Let us orient the  $\delta_6$  coboundary:

```
[41]: delta6_cocycle_directed = dGC(delta6_cocycle) #; delta6_cocycle_directed
```

```
[42]: len(delta6_cocycle_directed)
```

```
[42]: 10960
```

Next, we prove the factorization of the Poisson cocycle condition via the Jacobi identity for the  $\delta_6$  flow.

```
[43]: Q_delta6 = FGC(delta6_cocycle_directed.filter(max_out_degree=2)).
      ↪attach_to_ground((2,2,2,2,2,2,2)); len(Q_delta6)
```

```
[43]: 1500
```

```
[44]: Q_delta6_Leibniz = FGC(delta6_cocycle_directed.filter(max_out_degree=3)).
      ↪attach_to_ground((2,2,2,2,2,2,3)); len(Q_delta6_Leibniz)
```

```
[44]: 45965
```

```
[45]: P_Q_delta6 = wedge.schouten_bracket(Q_delta6)
```

```
[46]: len(P_Q_delta6)
```

```
[46]: 35949
```

```
[47]: Q_delta6_Leibniz_expanded = sum(Q_delta6_Leibniz.insertion(k, formality_stick,
      ↪max_out_degree=2) for k in [3,4,5,6,7,8,9])
```

```
[48]: P_Q_delta6 == -28 * Q_delta6_Leibniz_expanded
```

```
[48]: True
```

In fact, the coboundary  $\delta_6$  in the graph complex yields Poisson trivial flows on the spaces of Poisson brackets:

```
[49]: beta6_directed = dGC(beta6); len(beta6_directed)
```

```
[49]: 1024
```

```
[50]: X_beta6 = FGC(beta6_directed.filter(max_out_degree=2)).
      ↪attach_to_ground((2,2,2,2,2,2)); len(X_beta6)
```

```
[50]: 46
```

```
[51]: P_X_beta6 = wedge.schouten_bracket(X_beta6); len(P_X_beta6)
```

```
[51]: 598
```

```
[52]: X_beta6_Leibniz = FGC(beta6_directed.filter(max_out_degree=3)).
      ↪attach_to_ground((2,2,2,2,2,3)); len(X_beta6_Leibniz)
```

```
[52]: 1068
```

```
[53]: X_beta6_Leibniz_expanded = sum(X_beta6_Leibniz.insertion(k, formality_stick,
      ↪max_out_degree=2) for k in [2,3,4,5,6,7])
```

```
[54]: len(X_beta6_Leibniz_expanded)
```

```
[54]: 2098
```

```
[55]: Q_delta6 == -7*P_X_beta6 + 84*X_beta6_Leibniz_expanded
```



```
[60]: %time heptagon_cocycle_maybe_directed = dGC(heptagon_cocycle_maybe)
```

```
CPU times: user 3.03 s, sys: 32.1 ms, total: 3.06 s
Wall time: 3.06 s
```

```
[61]: len(heptagon_cocycle_maybe_directed)
```

```
[61]: 595476
```

Next, we prove the factorization of the Poisson cocycle condition via the Jacobi identity for the  $\gamma_7$  flow.

```
[62]: %time Q_heptagon_maybe = FGC(heptagon_cocycle_maybe_directed.
↳filter(max_out_degree=2)).attach_to_ground((2,2,2,2,2,2,2,2))
```

```
CPU times: user 2min 12s, sys: 3.66 s, total: 2min 16s
Wall time: 2min 15s
```

```
[63]: len(Q_heptagon_maybe)
```

```
[63]: 38538
```

```
[64]: %time Q_heptagon_maybe_Leibniz = FGC(heptagon_cocycle_maybe_directed.
↳filter(max_out_degree=3)).attach_to_ground((2,2,2,2,2,2,2,3))
```

```
CPU times: user 56min 36s, sys: 20.8 s, total: 56min 57s
Wall time: 56min 42s
```

```
[65]: len(Q_heptagon_maybe_Leibniz)
```

```
[65]: 1511359
```

```
[66]: %time P_Q_heptagon_maybe = wedge.schouten_bracket(Q_heptagon_maybe)
```

```
CPU times: user 1h 38min 59s, sys: 30.7 s, total: 1h 39min 30s
Wall time: 1h 38min 57s
```

```
[67]: len(P_Q_heptagon_maybe)
```

```
[67]: 1040373
```

```
[68]: %time Q_heptagon_maybe_Leibniz_expanded = sum(Q_heptagon_maybe_Leibniz.insertion(k,
↳formality_stick, max_out_degree=2) for k in [3,4,5,6,7,8,9,10])
```

```
CPU times: user 5h 6min 45s, sys: 1min 9s, total: 5h 7min 55s
Wall time: 5h 6min 34s
```

```
[69]: len(Q_heptagon_maybe_Leibniz_expanded)
```

[69]: 1040373

```
[70]: P_Q_heptagon_maybe == -32*Q_heptagon_maybe_Leibniz_expanded
```

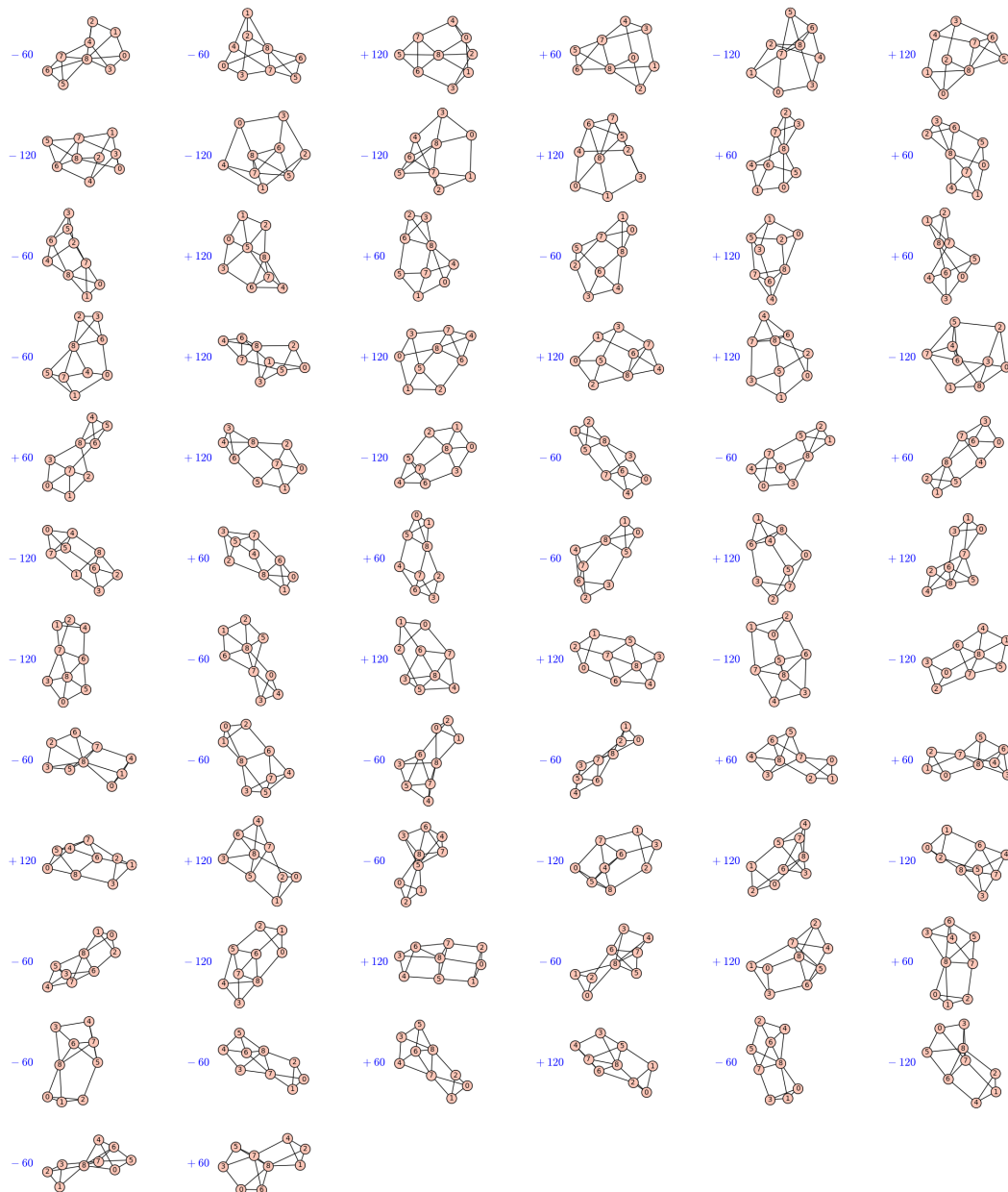
[70]: True

## 5.5 The commutator $[\gamma_3, \gamma_5]$

The commutator  $[\gamma_3, \gamma_5]$  of graph cocycles  $\gamma_3$  and  $\gamma_5$  is a sum of graphs, each on  $4+6-1 = 9$  vertices and  $6 + 10 = 16$  edges:

```
[71]: bracket_cocycle = tetrahedron_cocycle.bracket(fivewheel_cocycle)
      #bracket_cocycle
```

```
[72]: bracket_cocycle.show(ncols=6)
```



Count the number of terms:

```
[73]: len(bracket_cocycle)
```

```
[73]: 68
```

List the different values of coefficients:

```
[74]: set([c for (c,g) in bracket_cocycle])
```

```
[74]: {-120, -60, 60, 120}
```

Check that  $[\gamma_3, \gamma_5]$  is not a coboundary:

```
[75]: bracket_cocycle.is_coboundary()
```

```
[75]: False
```

Confirm that the cohomology in vertex-edge bi-grading (9, 16) is one-dimensional:

```
[76]: %time B_9_16 = GC.cohomology_basis(9,16)
```

```
CPU times: user 1min 18s, sys: 818 ms, total: 1min 19s
Wall time: 1min 19s
```

```
[77]: len(B_9_16)
```

```
[77]: 1
```

The basis element which was found automatically does not equal  $[\gamma_3, \gamma_5]$  exactly:

```
[78]: B_9_16[0] == bracket_cocycle
```

```
[78]: False
```

However, they are proportional in the cohomology:

```
[79]: (bracket_cocycle - 40*B_9_16[0]).is_coboundary()
```

```
[79]: True
```

Though not as cochains:

```
[80]: bracket_cocycle == 40*B_9_16[0]
```

```
[80]: False
```

Orient the graph cocycle  $[\gamma_3, \gamma_5]$ :

```
[81]: %time bracket_cocycle_directed = dGC(bracket_cocycle)
```

```
CPU times: user 10.7 s, sys: 1.55 s, total: 12.3 s
Wall time: 13.4 s
```

```
[82]: len(bracket_cocycle_directed)
```

```
[82]: 2752512
```

In a different way:

```
[83]: #bracket_directed = dGC(tetrahedron_cocycle).bracket(dGC(fivewheel_cocycle))
```

The outputs of the calculations in this section are stored externally at <https://rburing.nl/gcaops>: for each  $\gamma \in \{\gamma_3, \gamma_5, \delta_6, \gamma_7\}$  there is a triple of files, consisting of the flow  $Q_\gamma$  built of Kontsevich graphs, the Poisson differential  $[[P, Q_\gamma]]$  built of Kontsevich graphs, and the right-hand side of the Leibniz graph factorization  $[[P, Q_\gamma]] = \diamond_\gamma(P, [[P, P]])$ :

- [Q\\_gamma\\_3.txt](#) and [\[P,Q\\_gamma\\_3\].txt](#) and [\[P,Q\\_gamma\\_3\]\\_leibniz.txt](#)



- `Q_gamma_5.txt` and `[P,Q_gamma_5].txt` and `[P,Q_gamma_5]_leibniz.txt`
- `Q_delta_6.txt` and `[P,Q_delta_6].txt` and `[P,Q_delta_6]_leibniz.txt`
- `Q_gamma_7_maybe.txt` and `[P,Q_gamma_7_maybe].txt` and `[P,Q_gamma_7_maybe]_leibniz.txt`



# Chapter 6

## Graph complex action on Poisson structures in dimension two

We investigate the action of Kontsevich’s graph complex on Poisson structures in a very particular case, namely that of Poisson structures on  $\mathbb{R}^2$ . In the following we demonstrate how to correlate several things: undirected graphs will become directed, (un)directed graph cocycles will finally be evaluated at copies of Poisson structures and Jacobiators, trivializing vector fields will be shown to be Hamiltonian with respect to the standard symplectic structure, the respective Hamiltonian functions will be realized as sums of graphs, and coboundaries in the graph complex will be mapped to Poisson coboundaries thanks to identities for the Richardson–Nijenhuis bracket.

First import all necessary functionality from the `gcaops` package:

```
[1]: from gcaops.algebra.differential_polynomial_ring import DifferentialPolynomialRing
from gcaops.algebra.superfunction_algebra import SuperfunctionAlgebra
from gcaops.algebra.homogeneous_polynomial_poisson_complex import PoissonComplex
from gcaops.graph.undirected_graph import UndirectedGraph
from gcaops.graph.undirected_graph_complex import UndirectedGraphComplex
from gcaops.graph.directed_graph import DirectedGraph
from gcaops.graph.directed_graph_complex import DirectedGraphComplex
```

Let  $x, y$  be Cartesian coordinates on  $\mathbb{R}^2$ ,  $u$  be the coefficient of bi-vector  $P = u\partial_x \wedge \partial_y$ ;  $V_0, V_1$  be components of vector fields on  $\mathbb{R}^2$ , and  $H$  be a scalar function (the Hamiltonian) on  $\mathbb{R}^2$ . Define the superfunction algebra with these coordinates:

```
[2]: D2 = DifferentialPolynomialRing(QQ, ('u', 'V0', 'V1', 'H'), ('x', 'y'),
↳max_differential_orders=[8,1,1,1])
x, y = D2.base_variables()
u, V0, V1, H = D2.fibre_variables()
S2 = SuperfunctionAlgebra(D2, [x,y]); S2
```

```
[3]: Superfunction algebra over Differential Polynomial Ring in x, y, u, V0, V1, H, u_x,
u_y, u_xx, u_xy, u_yy, u_xxx, u_xxy, u_xyy, u_yyy, u_xxxx, u_xxxy, u_xxyy, u_xyyy,
u_yyyy, u_xxxxx, u_xxxxxy, u_xxxxyy, u_xxyyyy, u_xyyyyy, u_yyyyyy, u_xxxxxxx, u_xxxxxxy,
u_xxxxxyy, u_xxxxyyy, u_xxyyyyy, u_xyyyyyy, u_yyyyyyy, u_xxxxxxxx, u_xxxxxxxy, u_xxxxxyy,
u_xxxxxyy, u_xxxxxyy, u_xxxxxyy, u_xxxxxyy, u_xxxxxyy, u_xxxxxyy, u_xxxxxyy,
u_xxxxxyy, u_xxxxxyy, u_xxxxxyy, u_xxxxxyy, u_xxxxxyy, u_xxxxxyy, u_xxxxxyy,
V0_x, V0_y, V1_x, V1_y, H_x, H_y over Rational Field with even coordinates [x, y] and
odd coordinates (xi0, xi1)
```

```
[3]: xi = S2.odd_coordinates(); xi
```

```
[3]: (xi0, xi1)
```

Consider a generic bi-vector field:

```
[4]: P2 = u*xi[0]*xi[1]; P2
```

```
[4]: (u)*xi0*xi1
```

The Jacobi identity is always satisfied (because the only tri-vector field is zero), so every bi-vector field on  $\mathbb{R}^2$  is Poisson:

```
[5]: P2.bracket(P2)
```

```
[5]: 0
```

Define the graph complexes:

```
[6]: GC = UndirectedGraphComplex(QQ, connected=True, biconnected=True, min_degree=3,
↳ implementation='vector'); GC
```

```
[6]: Undirected graph complex over Rational Field with Basis consisting of representatives of isomorphism classes of undirected graphs (connected, biconnected, of degree at least 3) with no automorphisms that induce an odd permutation on edges
```

```
[7]: dGC = DirectedGraphComplex(QQ, connected=True, biconnected=True, min_degree=3,
↳ loops=False, implementation='vector', sparse=True); dGC
```

```
[7]: Directed graph complex over Rational Field with Basis consisting of representatives of isomorphism classes of directed graphs (connected, biconnected, of degree at least 3, without loops) with no automorphisms that induce an odd permutation on edges
```

```
[8]: dfGC = DirectedGraphComplex(QQ, connected=True, implementation='vector', sparse=True);
↳ dfGC
```

```
[8]: Directed graph complex over Rational Field with Basis consisting of representatives of isomorphism classes of directed graphs (connected) with no automorphisms that induce an odd permutation on edges
```

Now at our disposal we have the generic bi-vector field **P2**, and the graph complexes **GC** and **dGC**. Let us act on the bi-vector field with the graph complexes.

## 6.1 Tetrahedral $\gamma_3$ flow

This was the first example by Kontsevich (1996).

Define the tetrahedral flow:

```
[9]: tetrahedron_graph = UndirectedGraph(4, [(0,1),(0,2),(0,3),(1,2),(1,3),(2,3)]);
↳ tetrahedron_graph
tetrahedron_cocycle = GC(tetrahedron_graph); tetrahedron_cocycle
```

```
[9]: 1*UndirectedGraph(4, [(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)])
```

The tetrahedral flow is generally nonzero:

```
[10]: tetrahedron_operation2 = S2.graph_operation(tetrahedron_cocycle)
      Q_tetra2 = tetrahedron_operation2(P2,P2,P2,P2); Q_tetra2
```

```
[10]: (8*u_y^3*u_xxx - 24*u_x*u_y^2*u_xxy + 24*u_x^2*u_y*u_xyy - 8*u_x^3*u_yyy)*xi0*xi1
```

The tetrahedral flow indeed defines a Poisson 2-cocycle (again because the only tri-vector field is zero):

```
[11]: P2.bracket(Q_tetra2)
```

```
[11]: 0
```

We recall that the Poisson cohomology of  $\mathbb{R}^2$  can be non-trivial, e.g. already for  $P_2 = xy\partial_x \wedge \partial_y$  and an arbitrary (smooth or polynomial) vector field  $V$  we have that the bi-vector field  $[[P_2, V]]$  vanishes at the origin, while e.g. the bi-vector  $\partial_x \wedge \partial_y$  does not, hence this Poisson cocycle is not a coboundary.

However, from [27], the tetrahedral flow is known to be Poisson-trivial,  $Q_{\text{tetra}}(P_2, P_2, P_2, P_2) = [[P_2, X_{\text{tetra}}(P_2)]]$  in dimension two.

We find such a vector field. First, let  $V$  be an arbitrary vector field:

```
[12]: V = V0*xi[0] + V1*xi[1]; V
```

```
[12]: (V0)*xi0 + (V1)*xi1
```

Take the Poisson differential:

```
[13]: PbracketV = P2.bracket(V); PbracketV
```

```
[13]: (-V0*u_x - V1*u_y + u*V0_x + u*V1_y)*xi0*xi1
```

Solve  $Q_{\text{tetra}}(P_2, P_2, P_2, P_2) = [[P_2, V]]$  for  $V$  by using homogeneity of degree and differential weight of differential polynomials to generate an ansatz and solving the arising linear system:

```
[14]: from gcaops.algebra.differential_polynomial_solver import solve_homogeneous_diffpoly
```

```
[15]: cX_tetra2 = solve_homogeneous_diffpoly(Q_tetra2[0,1], PbracketV[0,1], [V0, V1]);
      ↪ cX_tetra2
```

```
[15]: {V0: -16*u_y*u_xy^2 + 16*u_y*u_xx*u_yy + 8*u_y^2*u_xxy - 16*u_x*u_y*u_xyy +
      8*u_x^2*u_yyy,
      V1: 16*u_x*u_xy^2 - 16*u_x*u_xx*u_yy - 8*u_y^2*u_xxx + 16*u_x*u_y*u_xxy -
      8*u_x^2*u_xyy}
```

```
[16]: X_tetra2 = cX_tetra2[V0]*xi[0] + cX_tetra2[V1]*xi[1]; X_tetra2
```

```
[16]: (-16*u_y*u_xy^2 + 16*u_y*u_xx*u_yy + 8*u_y^2*u_xxy - 16*u_x*u_y*u_xyy +
      8*u_x^2*u_yyy)*xi0 + (16*u_x*u_xy^2 - 16*u_x*u_xx*u_yy - 8*u_y^2*u_xxx +
      16*u_x*u_y*u_xxy - 8*u_x^2*u_xyy)*xi1
```

```
[17]: Q_tetra2 == P2.bracket(X_tetra2)
```

```
[17]: True
```

In fact, the vector field  $X_{\text{tetra}}(P_2)$  is known to be Hamiltonian with respect to the standard symplectic structure on  $\mathbb{R}^2$  [5]. Let us reproduce this result:

```
[18]: cH_tetra2 = solve_homogeneous_diffpoly(cX_tetra2[V0], diff(H,y), [H]); cH_tetra2
```

```
[18]: {H: 8*u_y^2*u_xx - 16*u_x*u_y*u_xy + 8*u_x^2*u_yy}
```

```
[19]: cH_tetra2 = solve_homogeneous_diffpoly(cX_tetra2[V1], -diff(H,x), [H]); cH_tetra2
```

```
[19]: {H: 8*u_y^2*u_xx - 16*u_x*u_y*u_xy + 8*u_x^2*u_yy}
```

```
[20]: H_tetra2 = cH_tetra2[H]; H_tetra2
```

```
[20]: 8*u_y^2*u_xx - 16*u_x*u_y*u_xy + 8*u_x^2*u_yy
```

```
[21]: diff(H_tetra2, y) == X_tetra2[0]
```

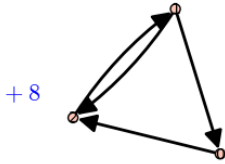
```
[21]: True
```

```
[22]: -diff(H_tetra2, x) == X_tetra2[1]
```

```
[22]: True
```

The Hamiltonian  $H$  itself can be written as a sum of *directed* graphs:

```
[23]: ham3 = 8*dfGC(DirectedGraph(3, [(0, 2), (1, 0), (1, 2), (2, 1)])); ham3.show()
```



```
[24]: S2.graph_operation(ham3)(P2,P2,P2)[0,1] == H_tetra2
```

```
[24]: True
```

## 6.2 Five-wheel $\gamma_5$ flow

Define the five-wheel cocycle:

```
[25]: fivewheel_graph = UndirectedGraph(6, □
  ↪ [(0,1), (1,2), (2,3), (3,4), (0,4), (0,5), (1,5), (2,5), (3,5), (4,5)])
roof_graph = UndirectedGraph(6, □
  ↪ [(0,1), (1,2), (2,3), (0,3), (3,4), (0,4), (4,5), (2,5), (1,5), (0,2)])
fivewheel_cocycle = GC(fivewheel_graph) + (5/2)*GC(roof_graph)
```

Take the 5-wheel cocycle and apply Kontsevich's formula to evaluate it at six copies of the Poisson structure:

```
[26]: fivewheel_operation2 = S2.graph_operation(fivewheel_cocycle)
      %time Q_fivewheel2 = fivewheel_operation2(P2,P2,P2,P2,P2,P2); Q_fivewheel2
```

CPU times: user 7.66 s, sys: 46.9 ms, total: 7.7 s  
Wall time: 7.7 s

```
[26]: (-10*u_y^3*u_xx*u_yy*u_xxx + 20*u_x*u_y^2*u_xy*u_yy*u_xxx - 10*u_x^2*u_y*u_yy^2*u_xxx
+ 20*u_y^3*u_xx*u_xy*u_xxy - 40*u_x*u_y^2*u_xy^2*u_xxy + 10*u_x*u_y^2*u_xx*u_yy*u_xxy
+ 10*u_x^3*u_yy^2*u_xxy - 10*u_y^3*u_xx^2*u_xyy + 40*u_x^2*u_y*u_xy^2*u_xyy -
10*u_x^2*u_y*u_xx*u_yy*u_xyy - 20*u_x^3*u_xy*u_yy*u_xyy + 10*u_x*u_y^2*u_xx^2*u_yyy -
20*u_x^2*u_y*u_xx*u_xy*u_yyy + 10*u_x^3*u_xx*u_yy*u_yyy - 10*u_y^4*u_xy*u_xxxx +
10*u_x*u_y^3*u_yy*u_xxxx + 10*u_y^4*u_xx*u_xxy + 20*u_x*u_y^3*u_xy*u_xxy -
30*u_x^2*u_y^2*u_yy*u_xxy - 30*u_x*u_y^3*u_xx*u_xxy + 30*u_x^3*u_y*u_yy*u_xxy +
30*u_x^2*u_y^2*u_xx*u_xyy - 20*u_x^3*u_y*u_xy*u_xyy - 10*u_x^4*u_yy*u_xyy -
10*u_x^3*u_y*u_xx*u_yyy + 10*u_x^4*u_xy*u_yyy - 2*u_y^5*u_xxxx +
10*u_x*u_y^4*u_xxy - 20*u_x^2*u_y^3*u_xxy + 20*u_x^3*u_y^2*u_xxy -
10*u_x^4*u_y*u_xyy + 2*u_x^5*u_yyy)*xi0*xi1
```

Let us do the same in a different way. First let us direct the edges of graphs in all possible ways; then, filter out all directed graphs with vertices of outgoing degrees greater than two; finally we place a copy of the Poisson bi-vector into every vertex. The outgoing edges stand for derivatives of a bi-vector.

```
[27]: %%time
      fivewheel_cocycle_directed = dGC(fivewheel_cocycle)
      fivewheel_cocycle_directed_filtered = fivewheel_cocycle_directed.
      ↪filter(max_out_degree=2)
      fivewheel_operation2_directed = S2.
      ↪graph_operation(fivewheel_cocycle_directed_filtered)
      Q_fivewheel2 = fivewheel_operation2_directed(P2,P2,P2,P2,P2,P2); Q_fivewheel2
```

CPU times: user 1.81 s, sys: 3.91 ms, total: 1.81 s  
Wall time: 1.82 s

```
[27]: (-10*u_y^3*u_xx*u_yy*u_xxx + 20*u_x*u_y^2*u_xy*u_yy*u_xxx - 10*u_x^2*u_y*u_yy^2*u_xxx
+ 20*u_y^3*u_xx*u_xy*u_xxy - 40*u_x*u_y^2*u_xy^2*u_xxy + 10*u_x*u_y^2*u_xx*u_yy*u_xxy
+ 10*u_x^3*u_yy^2*u_xxy - 10*u_y^3*u_xx^2*u_xyy + 40*u_x^2*u_y*u_xy^2*u_xyy -
10*u_x^2*u_y*u_xx*u_yy*u_xyy - 20*u_x^3*u_xy*u_yy*u_xyy + 10*u_x*u_y^2*u_xx^2*u_yyy -
20*u_x^2*u_y*u_xx*u_xy*u_yyy + 10*u_x^3*u_xx*u_yy*u_yyy - 10*u_y^4*u_xy*u_xxxx +
10*u_x*u_y^3*u_yy*u_xxxx + 10*u_y^4*u_xx*u_xxy + 20*u_x*u_y^3*u_xy*u_xxy -
30*u_x^2*u_y^2*u_yy*u_xxy - 30*u_x*u_y^3*u_xx*u_xxy + 30*u_x^3*u_y*u_yy*u_xxy +
30*u_x^2*u_y^2*u_xx*u_xyy - 20*u_x^3*u_y*u_xy*u_xyy - 10*u_x^4*u_yy*u_xyy -
10*u_x^3*u_y*u_xx*u_yyy + 10*u_x^4*u_xy*u_yyy - 2*u_y^5*u_xxxx +
10*u_x*u_y^4*u_xxy - 20*u_x^2*u_y^3*u_xxy + 20*u_x^3*u_y^2*u_xxy -
10*u_x^4*u_y*u_xyy + 2*u_x^5*u_yyy)*xi0*xi1
```

Quite naturally, the outputs of the two cells immediately above coincide, as expected.

The five-wheel flow is a cocycle:

```
[28]: P2.bracket(Q_fivewheel2)
```

```
[28]: 0
```

The graph flow  $\gamma_5$  is also Poisson-trivial on  $\mathbb{R}^2$ :

```
[29]: cX_fivewheel2 = solve_homogeneous_diffpoly(Q_fivewheel2[0,1], PbracketV[0,1], [V0, u
↪V1]); cX_fivewheel2
```

```
[29]: {V0: -12*u_y*u_xy^4 + 24*u_y*u_xx*u_xy^2*u_yy - 12*u_y*u_xx^2*u_yy^2 -
4*u_y^2*u_xy*u_yy*u_xxx + 4*u_x*u_y*u_yy^2*u_xxx + 8*u_y^2*u_xy^2*u_xxy -
6*u_y^2*u_xx*u_yy*u_xxy + 8*u_x*u_y*u_xy*u_yy*u_xxy - 10*u_x^2*u_yy^2*u_xxy +
2*u_y^3*u_xxy^2 + 8*u_y^2*u_xx*u_xy*u_xyy - 32*u_x*u_y*u_xy^2*u_xyy +
4*u_x*u_y*u_xx*u_yy*u_xyy + 20*u_x^2*u_xy*u_yy*u_xyy - 2*u_y^3*u_xxx*u_xyy -
2*u_x*u_y^2*u_xxy*u_xyy + 2*u_x^2*u_y*u_xyy^2 - 6*u_y^2*u_xx^2*u_yyy +
16*u_x*u_y*u_xx*u_xy*u_yyy - 10*u_x^2*u_xx*u_yy*u_yyy + 2*u_x*u_y^2*u_xxx*u_yyy -
2*u_x^2*u_y*u_xxy*u_yyy - 8*u_y^3*u_yy*u_xxxx + 6*u_y^3*u_xy*u_xxy +
26*u_x*u_y^2*u_yy*u_xxy + 2*u_y^3*u_xx*u_xxy - 22*u_x*u_y^2*u_xy*u_xxy -
28*u_x^2*u_y*u_yy*u_xxy - 4*u_x*u_y^2*u_xx*u_xxy + 26*u_x^2*u_y*u_xy*u_xxy +
10*u_x^3*u_yy*u_xxy + 2*u_x^2*u_y*u_xx*u_xxy - 10*u_x^3*u_xy*u_xxy -
2*u_y^4*u_xxy + 8*u_x*u_y^3*u_xxy - 12*u_x^2*u_y^2*u_xxy + 8*u_x^3*u_y*u_xxy -
2*u_x^4*u_xxy},
V1: 12*u_x*u_xy^4 - 24*u_x*u_xx*u_xy^2*u_yy + 12*u_x*u_xx^2*u_yy^2 +
10*u_y^2*u_xx*u_yy*u_xxx - 16*u_x*u_y*u_xy*u_yy*u_xxx + 6*u_x^2*u_yy^2*u_xxx -
20*u_y^2*u_xx*u_xy*u_xxy + 32*u_x*u_y*u_xy^2*u_xxy - 4*u_x*u_y*u_xx*u_yy*u_xxy -
8*u_x^2*u_xy*u_yy*u_xxy - 2*u_x*u_y^2*u_xxy^2 + 10*u_y^2*u_xx^2*u_xxy -
8*u_x*u_y*u_xx*u_xy*u_xyy - 8*u_x^2*u_xy^2*u_xyy + 6*u_x^2*u_xx*u_yy*u_xyy +
2*u_x*u_y^2*u_xxx*u_xyy + 2*u_x^2*u_y*u_xxy*u_xyy - 2*u_x^3*u_xxy^2 -
4*u_x*u_y*u_xx^2*u_yyy + 4*u_x^2*u_xx*u_xy*u_yyy - 2*u_x^2*u_y*u_xxx*u_yyy +
2*u_x^3*u_xxy*u_yyy + 10*u_y^3*u_xy*u_xxxx - 2*u_x*u_y^2*u_yy*u_xxxx -
10*u_y^3*u_xx*u_xxy - 26*u_x*u_y^2*u_xy*u_xxy + 4*u_x^2*u_y*u_yy*u_xxy +
28*u_x*u_y^2*u_xx*u_xxy + 22*u_x^2*u_y*u_xy*u_xxy - 2*u_x^3*u_yy*u_xxy -
26*u_x^2*u_y*u_xx*u_xxy - 6*u_x^3*u_xy*u_xxy + 8*u_x^3*u_xx*u_yyy + 2*u_y^4*u_xxxx -
8*u_x*u_y^3*u_xxy + 12*u_x^2*u_y^2*u_xxy - 8*u_x^3*u_y*u_xxy +
2*u_x^4*u_xxy}
```

```
[30]: X_fivewheel2 = cX_fivewheel2[V0]*xi[0] + cX_fivewheel2[V1]*xi[1]
```

```
[31]: Q_fivewheel2 == P2.bracket(X_fivewheel2)
```

```
[31]: True
```

This vector field is also Hamiltonian with respect to the standard symplectic structure on  $\mathbb{R}^2$ .

```
[32]: cH_fivewheel2 = solve_homogeneous_diffpoly(cX_fivewheel2[V0], diff(H,y), [H]); u
↪cH_fivewheel2
```

```
[32]: {H: 6*u_y^2*u_xx*u_xy^2 - 12*u_x*u_y*u_xy^3 - 6*u_y^2*u_xx^2*u_yy +
12*u_x*u_y*u_xx*u_xy*u_yy + 6*u_x^2*u_xy^2*u_yy - 6*u_x^2*u_xx*u_yy^2 -
2*u_y^3*u_xy*u_xxx + 2*u_x*u_y^2*u_yy*u_xxx + 2*u_y^3*u_xx*u_xxy +
2*u_x*u_y^2*u_xy*u_xxy - 4*u_x^2*u_y*u_yy*u_xxy - 4*u_x*u_y^2*u_xx*u_xxy +
2*u_x^2*u_y*u_xy*u_xyy + 2*u_x^3*u_yy*u_xyy + 2*u_x^2*u_y*u_xx*u_yyy -
2*u_x^3*u_xy*u_yyy - 2*u_y^4*u_xxy + 8*u_x*u_y^3*u_xxy - 12*u_x^2*u_y^2*u_xxy +
8*u_x^3*u_y*u_xxy - 2*u_x^4*u_xxy}
```

```
[33]: cH_fivewheel2 = solve_homogeneous_diffpoly(cX_fivewheel2[V1], -diff(H,x), [H]); u
↪cH_fivewheel2
```

```
[33]: {H: 6*u_y^2*u_xx*u_xy^2 - 12*u_x*u_y*u_xy^3 - 6*u_y^2*u_xx^2*u_yy +
12*u_x*u_y*u_xx*u_xy*u_yy + 6*u_x^2*u_xy^2*u_yy - 6*u_x^2*u_xx*u_yy^2 -
2*u_y^3*u_xy*u_xxx + 2*u_x*u_y^2*u_yy*u_xxx + 2*u_y^3*u_xx*u_xxy +
```



$$2*u_x*u_y^2*u_{xy}u_{xxy} - 4*u_x^2*u_y*u_{yy}u_{xxy} - 4*u_x*u_y^2*u_{xx}u_{xy} + 2*u_x^2*u_y*u_{xy}u_{xy} + 2*u_x^3*u_{yy}u_{xy} + 2*u_x^2*u_y*u_{xx}u_{yyy} - 2*u_x^3*u_{xy}u_{yyy} - 2*u_y^4*u_{xxxx} + 8*u_x*u_y^3*u_{xxy} - 12*u_x^2*u_y^2*u_{xxy} + 8*u_x^3*u_y*u_{xyy} - 2*u_x^4*u_{yyyy}$$

```
[34]: H_fivewheel2 = cH_fivewheel2[H]; H_fivewheel2
```

```
[34]: 6*u_y^2*u_xx*u_xy^2 - 12*u_x*u_y*u_xy^3 - 6*u_y^2*u_xx^2*u_yy + 12*u_x*u_y*u_xx*u_xy*u_yy + 6*u_x^2*u_xy^2*u_yy - 6*u_x^2*u_xx*u_yy^2 - 2*u_y^3*u_xy*u_xxx + 2*u_x*u_y^2*u_yy*u_xxx + 2*u_y^3*u_xx*u_xxy + 2*u_x*u_y^2*u_xy*u_xxy - 4*u_x^2*u_y*u_yy*u_xxy - 4*u_x*u_y^2*u_xx*u_xxy + 2*u_x^2*u_y*u_xy*u_xyy + 2*u_x^3*u_yy*u_xyy + 2*u_x^2*u_y*u_xx*u_yyy - 2*u_x^3*u_xy*u_yyy - 2*u_y^4*u_xxxx + 8*u_x*u_y^3*u_xxy - 12*u_x^2*u_y^2*u_xxy + 8*u_x^3*u_y*u_xyy - 2*u_x^4*u_yyyy
```

```
[35]: diff(H_fivewheel2, y) == X_fivewheel2[0]
```

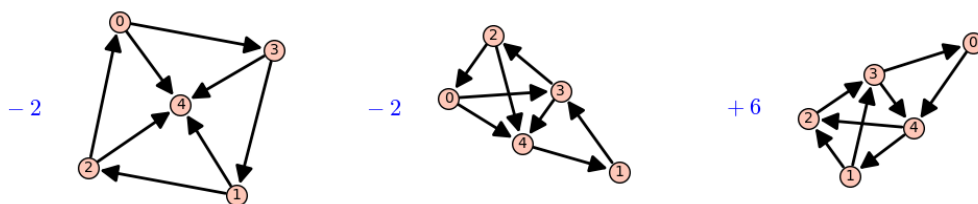
```
[35]: True
```

```
[36]: -diff(H_fivewheel2, x) == X_fivewheel2[1]
```

```
[36]: True
```

The Hamiltonian of the found trivializing vector field for the  $\gamma_5$  flow can be written as a sum of directed graphs:

```
[37]: ham5 = (6)*dfGC(DirectedGraph(5, [(0, 4), (1, 2), (1, 3), (2, 3), (3, 0), (3, 4), (4, 1), (4, 2)])) + \
      (-2)*dfGC(DirectedGraph(5, [(0, 3), (0, 4), (1, 3), (2, 0), (2, 4), (3, 2), (3, 4), (4, 1)])) + \
      (-2)*dfGC(DirectedGraph(5, [(0, 3), (0, 4), (1, 2), (1, 4), (2, 0), (2, 4), (3, 1), (3, 4)]))
      ham5.show()
```



Actually you get a bi-vector on  $\mathbb{R}^2$ , and the Hamiltonian is its coefficient.

```
[38]: S2.graph_operation(ham5)(P2,P2,P2,P2,P2)[0,1] == H_fivewheel2
```

```
[38]: True
```



```

+ 20*u_y^2*u_xx*u_xy^2*u_xxy - 8*u_x*u_y*u_xy^3*u_xxy - 8*u_y^2*u_xx^2*u_yy*u_xxy -
16*u_x*u_y*u_xx*u_xy*u_yy*u_xxy + 12*u_x^2*u_xx*u_yy^2*u_xxy + 8*u_y^3*u_xx*u_xxy^2 -
8*u_x*u_y^2*u_xy*u_xxy^2 - 12*u_y^2*u_xx^2*u_xy*u_xxy + 8*u_x*u_y*u_xx*u_xy^2*u_xxy +
4*u_x^2*u_xy^3*u_xxy + 16*u_x*u_y*u_xx^2*u_yy*u_xxy - 16*u_x^2*u_xx*u_xy*u_yy*u_xxy -
8*u_y^3*u_xx*u_xxx*u_xxy + 8*u_x*u_y^2*u_xy*u_xxx*u_xxy - 8*u_x*u_y^2*u_xx*u_xxy*u_xxy
+ 8*u_x^2*u_y*u_xy*u_xxy*u_xxy + 8*u_x^2*u_y*u_xx*u_xxy^2 - 8*u_x^3*u_xy*u_xxy^2 +
4*u_y^2*u_xx^3*u_yyy - 8*u_x*u_y*u_xx^2*u_xy*u_yyy + 4*u_x^2*u_xx*u_xy^2*u_yyy +
8*u_x*u_y^2*u_xx*u_xxx*u_yyy - 8*u_x^2*u_y*u_xy*u_xxx*u_yyy -
8*u_x^2*u_y*u_xx*u_xxy*u_yyy + 8*u_x^3*u_xy*u_xxy*u_yyy - 4*u_y^3*u_xy^2*u_xxxx +
8*u_x*u_y^2*u_xy*u_yy*u_xxxx - 4*u_x^2*u_y*u_yy^2*u_xxxx + 8*u_y^3*u_xx*u_xy*u_xxxx -
4*u_x*u_y^2*u_xy^2*u_xxxx - 8*u_x*u_y^2*u_xx*u_yy*u_xxxx + 4*u_x^3*u_yy^2*u_xxxx -
4*u_y^3*u_xx^2*u_xxxx + 4*u_x^2*u_y*u_xy^2*u_xxxx + 8*u_x^2*u_y*u_xx*u_yy*u_xxxx -
8*u_x^3*u_xy*u_yy*u_xxxx + 4*u_x*u_y^2*u_xx^2*u_xxxx - 8*u_x^2*u_y*u_xx*u_xy*u_xxxx +
4*u_x^3*u_xy^2*u_xxxx)*xi1

```

```
[42]: delta6_operation2 = S2.graph_operation(delta6_cocycle); delta6_operation2
```

[42]: Symmetric operation of arity 7 and degree -12 on Superfunction algebra over Differential Polynomial Ring in  $x, y, u, V_0, V_1, H, u_x, u_y, u_{xx}, u_{xy}, u_{yy}, u_{xxx}, u_{xxy}, u_{xyy}, u_{yyy}, u_{xxxx}, u_{xxxxy}, u_{xxyy}, u_{xyyy}, u_{yyyy}, u_{xxxxx}, u_{xxxxy}, u_{xxxyy}, u_{xxyyy}, u_{xyyyy}, u_{yyyyy}, u_{xxxxxx}, u_{xxxxyy}, u_{xxxyyy}, u_{xxyyyy}, u_{xyyyyy}, u_{yyyyyy}, u_{xxxxxxx}, u_{xxxxyy}, u_{xxxyyy}, u_{xxyyyy}, u_{xyyyyy}, u_{yyyyyy}, u_{xxxxxxx}, u_{xxxxyy}, u_{xxxyyy}, u_{xxyyyy}, u_{xyyyyy}, u_{yyyyyy}, u_{xxxxxxx}, u_{xxxxyy}, u_{xxxyyy}, u_{xxyyyy}, u_{xyyyyy}, u_{yyyyyy}, u_{xxxxxxx}, u_{xxxxyy}, u_{xxxyyy}, u_{xxyyyy}, u_{xyyyyy}, u_{yyyyyy}, u_{xxxxxxx}, u_{xxxxyy}, u_{xxxyyy}, u_{xxyyyy}, u_{xyyyyy}, u_{yyyyyy},  $V_0_x, V_0_y, V_1_x, V_1_y, H_x, H_y$  over Rational Field with even coordinates  $[x, y]$  and odd coordinates  $(xi_0, xi_1)$$

Here is the flow  $\dot{P}_2 = \text{Or}(\delta_6)(P_2^{\otimes 7})$ :

```
[43]: %time Q_delta6_2 = delta6_operation2(P2,P2,P2,P2,P2,P2,P2); Q_delta6_2
```

CPU times: user 2min 9s, sys: 752 ms, total: 2min 10s

Wall time: 2min 10s

```

[43]: (24*u_y^3*u_xy^3*u_xxx - 16*u_y^3*u_xx*u_xy*u_yy*u_xxx -
40*u_x*u_y^2*u_xy^2*u_yy*u_xxx + 16*u_x*u_y^2*u_xx*u_yy^2*u_xxx +
24*u_x^2*u_y*u_xy*u_yy^2*u_xxx - 8*u_x^3*u_yy^3*u_xxx - 40*u_y^3*u_xx*u_xy^2*u_xxy +
8*u_x*u_y^2*u_xy^3*u_xxy + 16*u_y^3*u_xx^2*u_yy*u_xxy +
64*u_x*u_y^2*u_xx*u_xy*u_yy*u_xxy - 16*u_x^2*u_y*u_xy^2*u_yy*u_xxy -
56*u_x^2*u_y*u_xx*u_yy^2*u_xxy + 24*u_x^3*u_xy*u_yy^2*u_xxy - 16*u_y^4*u_xx*u_xxy^2 +
32*u_x*u_y^3*u_xy*u_xxy^2 - 16*u_x^2*u_y^2*u_yy*u_xxy^2 + 24*u_y^3*u_xx^2*u_xy*u_xxy -
16*u_x*u_y^2*u_xx*u_xy^2*u_xxy + 8*u_x^2*u_y*u_xy^3*u_xxy -
56*u_x*u_y^2*u_xx^2*u_yy*u_xxy + 64*u_x^2*u_y*u_xx*u_xy*u_yy*u_xxy -
40*u_x^3*u_xy^2*u_yy*u_xxy + 16*u_x^3*u_xx*u_yy^2*u_xxy + 16*u_y^4*u_xx*u_xxx*u_xxy -
32*u_x*u_y^3*u_xy*u_xxx*u_xxy + 16*u_x^2*u_y^2*u_yy*u_xxx*u_xxy +
16*u_x*u_y^3*u_xx*u_xxy*u_xxy - 32*u_x^2*u_y^2*u_xy*u_xxy*u_xxy +
16*u_x^3*u_y*u_yy*u_xxy*u_xxy - 16*u_x^2*u_y^2*u_xx*u_xxy^2 +
32*u_x^3*u_y*u_xy*u_xxy^2 - 16*u_x^4*u_yy*u_xxy^2 - 8*u_y^3*u_xx^3*u_yyy +
24*u_x*u_y^2*u_xx^2*u_xy*u_yyy - 40*u_x^2*u_y*u_xx*u_xy^2*u_yyy +
24*u_x^3*u_xy^3*u_yyy + 16*u_x^2*u_y*u_xx^2*u_yy*u_yyy - 16*u_x^3*u_xx*u_xy*u_yy*u_yyy
- 16*u_x*u_y^3*u_xx*u_xxx*u_yyy + 32*u_x^2*u_y^2*u_xy*u_xxx*u_yyy -
16*u_x^3*u_y*u_yy*u_xxx*u_yyy + 16*u_x^2*u_y^2*u_xx*u_xxy*u_yyy -
32*u_x^3*u_y*u_xy*u_xxy*u_yyy + 16*u_x^4*u_yy*u_xxy*u_yyy + 8*u_y^4*u_xy^2*u_xxxx -
16*u_x*u_y^3*u_xy*u_yy*u_xxxx + 8*u_x^2*u_y^2*u_yy^2*u_xxxx -
16*u_y^4*u_xx*u_xy*u_xxxx + 16*u_x*u_y^3*u_xx*u_yy*u_xxxx +
16*u_x^2*u_y^2*u_xy*u_yy*u_xxxx - 16*u_x^3*u_y*u_yy^2*u_xxxx + 8*u_y^4*u_xx^2*u_xxxx +
16*u_x*u_y^3*u_xx*u_xy*u_xxxx - 16*u_x^2*u_y^2*u_xy^2*u_xxxx -

```



$$8*u_x^2*u_y*u_{xx}*u_{yy}*u_{xyy} + 8*u_x^3*u_{xy}*u_{yy}*u_{xyy} - 4*u_x*u_y^2*u_{xx}^2*u_{yyy} + 8*u_x^2*u_y*u_{xx}*u_{xy}*u_{yyy} - 4*u_x^3*u_{xy}^2*u_{yyy}$$

```
[50]: solve_homogeneous_diffpoly(X_delta6_2[1], -diff(H,x), [H])
```

```
[50]: {H: 4*u_y^3*u_xy^2*u_xxx - 8*u_x*u_y^2*u_xy*u_yy*u_xxx + 4*u_x^2*u_y*u_yy^2*u_xxx -
8*u_y^3*u_xx*u_xy*u_xxy + 4*u_x*u_y^2*u_xy^2*u_xxy + 8*u_x*u_y^2*u_xx*u_yy*u_xxy -
4*u_x^3*u_yy^2*u_xxy + 4*u_y^3*u_xx^2*u_xyy - 4*u_x^2*u_y*u_xy^2*u_xyy -
8*u_x^2*u_y*u_xx*u_yy*u_xyy + 8*u_x^3*u_xy*u_yy*u_xyy - 4*u_x*u_y^2*u_xx^2*u_yyy +
8*u_x^2*u_y*u_xx*u_xy*u_yyy - 4*u_x^3*u_xy^2*u_yyy}
```

```
[51]: H_delta6_2 = cH_delta6_2[H]
```

```
[52]: diff(H_delta6_2, y) == X_delta6_2[0]
```

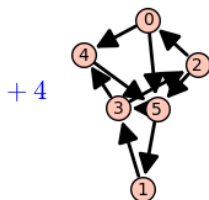
```
[52]: True
```

```
[53]: -diff(H_delta6_2, x) == X_delta6_2[1]
```

```
[53]: True
```

The Hamiltonian can be realized as a sum of graphs:

```
[54]: ham6_1 = (4)*dfGC(DirectedGraph(6, [(0, 4), (0, 5), (1, 3), (2, 0), (2, 5), (3, 2),
↪(3, 4), (4, 5), (5, 1), (5, 3)])); ham6_1.show()
```



```
[55]: S2.graph_operation(ham6_1)(P2,P2,P2,P2,P2,P2)[0,1] == H_delta6_2
```

```
[55]: True
```

## 6.4 Heptagon-wheel $\gamma_7$ flow

Get a representative of the cohomology class for  $\gamma_7$ :

```
[56]: heptagon_cocycle_maybe = GC.cohomology_basis(8,14)[0]
```

Orient it:

```
[57]: %time heptagon_cocycle_maybe_directed = dGC(heptagon_cocycle_maybe)
```



[67]: True

The trivializing vector field is Hamiltonian with respect to the standard symplectic structure:

```
[68]: cH_hepta2 = solve_homogeneous_diffpoly(cX_hepta2[V0], diff(H,y), [H]); cH_hepta2
```

```
[68]: {H: 199/4*u_y^2*u_xx*u_xy^4 - 199/2*u_x*u_y*u_xy^5 - 199/2*u_y^2*u_xx^2*u_xy^2*u_yy +
199*u_x*u_y*u_xx*u_xy^3*u_yy + 199/4*u_x^2*u_xy^4*u_yy + 199/4*u_y^2*u_xx^3*u_yy^2 -
199/2*u_x*u_y*u_xx^2*u_xy*u_yy^2 - 199/2*u_x^2*u_xx*u_xy^2*u_yy^2 +
199/4*u_x^2*u_xx^2*u_yy^3 - 9*u_y^3*u_xy^3*u_xxx + 23/4*u_y^3*u_xx*u_xy*u_yy*u_xxx +
31/2*u_x*u_y^2*u_xy^2*u_yy*u_xxx - 23/4*u_x*u_y^2*u_xx*u_yy^2*u_xxx -
39/4*u_x^2*u_y*u_xy*u_yy^2*u_xxx + 13/4*u_x^3*u_yy^3*u_xxx + 6*u_y^4*u_yy*u_xxx^2 +
31/2*u_y^3*u_xx*u_xy^2*u_xxy - 4*u_x*u_y^2*u_xy^3*u_xxy - 23/4*u_y^3*u_xx^2*u_yy*u_xxy
- 101/4*u_x*u_y^2*u_xx*u_xy*u_yy*u_xxy + 8*u_x^2*u_y*u_xy^2*u_yy*u_xxy +
85/4*u_x^2*u_y*u_xx*u_yy^2*u_xxy - 39/4*u_x^3*u_xy*u_yy^2*u_xxy -
12*u_y^4*u_xy*u_xxx*u_xxy - 24*u_x*u_y^3*u_yy*u_xxx*u_xxy - 9*u_y^4*u_xx*u_xxy^2 +
54*u_x*u_y^3*u_xy*u_xxy^2 + 9*u_x^2*u_y^2*u_yy*u_xxy^2 - 39/4*u_y^3*u_xx^2*u_xy*u_xxy
+ 8*u_x*u_y^2*u_xx*u_xy^2*u_xyy - 4*u_x^2*u_y*u_xy^3*u_xyy +
85/4*u_x*u_y^2*u_xx^2*u_yy*u_xyy - 101/4*u_x^2*u_y*u_xx*u_xy*u_yy*u_xyy +
31/2*u_x^3*u_xy^2*u_yy*u_xyy - 23/4*u_x^3*u_xx*u_yy^2*u_xyy +
15*u_y^4*u_xx*u_xxx*u_xyy - 6*u_x*u_y^3*u_xy*u_xxx*u_xyy +
27*u_x^2*u_y^2*u_yy*u_xxx*u_xyy - 9*u_x*u_y^3*u_xx*u_xxy*u_xyy -
90*u_x^2*u_y^2*u_xy*u_xxy*u_xyy - 9*u_x^3*u_y*u_yy*u_xxy*u_xyy +
9*u_x^2*u_y^2*u_xx*u_xyy^2 + 54*u_x^3*u_y*u_xy*u_xyy^2 - 9*u_x^4*u_yy*u_xyy^2 +
13/4*u_y^3*u_xx^3*u_yyy - 39/4*u_x*u_y^2*u_xx^2*u_xy*u_yyy +
31/2*u_x^2*u_y*u_xx*u_xy^2*u_yyy - 9*u_x^3*u_xy^3*u_yyy -
23/4*u_x^2*u_y*u_xx^2*u_yy*u_yyy + 23/4*u_x^3*u_xx*u_xy*u_yy*u_yyy -
15*u_x*u_y^3*u_xx*u_xxx*u_yyy + 18*u_x^2*u_y^2*u_xy*u_xxx*u_yyy -
15*u_x^3*u_y*u_yy*u_xxx*u_yyy + 27*u_x^2*u_y^2*u_xx*u_xxy*u_yyy -
6*u_x^3*u_y*u_xy*u_xxy*u_yyy + 15*u_x^4*u_yy*u_xxy*u_yyy -
24*u_x^3*u_y*u_xx*u_xyy*u_yyy - 12*u_x^4*u_xy*u_xyy*u_yyy + 6*u_x^4*u_xx*u_yyy^2 +
18*u_y^4*u_xy^2*u_xxxx + 16*u_y^4*u_xx*u_yy*u_xxxx - 68*u_x*u_y^3*u_xy*u_yy*u_xxxx +
34*u_x^2*u_y^2*u_yy^2*u_xxxx + 12*u_y^5*u_xxy*u_xxxx - 24*u_x*u_y^4*u_xyy*u_xxxx +
12*u_x^2*u_y^3*u_yyy*u_xxxx - 68*u_y^4*u_xx*u_xy*u_xxy + 64*u_x*u_y^3*u_xy^2*u_xxy +
4*u_x*u_y^3*u_xx*u_yy*u_xxy + 68*u_x^2*u_y^2*u_xy*u_yy*u_xxy -
68*u_x^3*u_y*u_yy^2*u_xxy - 12*u_y^5*u_xxx*u_xxy - 12*u_x*u_y^4*u_xxy*u_xxy +
60*u_x^2*u_y^3*u_xxy*u_xxy - 36*u_x^3*u_y^2*u_yyy*u_xxy + 34*u_y^4*u_xx^2*u_xxy +
68*u_x*u_y^3*u_xx*u_xy*u_xxy - 164*u_x^2*u_y^2*u_xy^2*u_xxy -
40*u_x^2*u_y^2*u_xx*u_yy*u_xxy + 68*u_x^3*u_y*u_xy*u_yy*u_xxy +
34*u_x^4*u_yy^2*u_xxy + 36*u_x*u_y^4*u_xxx*u_xxy - 36*u_x^2*u_y^3*u_xxy*u_xxy -
36*u_x^3*u_y^2*u_xxy*u_xxy + 36*u_x^4*u_y*u_yyy*u_xxy - 68*u_x*u_y^3*u_xx^2*u_xxy +
68*u_x^2*u_y^2*u_xx*u_xy*u_xxy + 64*u_x^3*u_y*u_xy^2*u_xxy +
4*u_x^3*u_y*u_xx*u_yy*u_xxy - 68*u_x^4*u_xy*u_yy*u_xxy - 36*u_x^2*u_y^3*u_xxx*u_xxy
+ 60*u_x^3*u_y^2*u_xxy*u_xxy - 12*u_x^4*u_y*u_xxy*u_xxy - 12*u_x^5*u_yyy*u_xxy +
34*u_x^2*u_y^2*u_xx^2*u_yyy - 68*u_x^3*u_y*u_xx*u_xy*u_yyy + 18*u_x^4*u_xy^2*u_yyy
+ 16*u_x^4*u_xx*u_yy*u_yyy + 12*u_x^3*u_y^2*u_xxx*u_yyy - 24*u_x^4*u_y*u_xxy*u_yyy
+ 12*u_x^5*u_xxy*u_yyy + 16*u_y^5*u_xy*u_xxxx - 16*u_x*u_y^4*u_yy*u_xxxx -
16*u_y^5*u_xx*u_xxxx - 48*u_x*u_y^4*u_xy*u_xxxx + 64*u_x^2*u_y^3*u_yy*u_xxxx +
64*u_x*u_y^4*u_xx*u_xxy + 32*u_x^2*u_y^3*u_xy*u_xxy - 96*u_x^3*u_y^2*u_yy*u_xxy
- 96*u_x^2*u_y^3*u_xx*u_xxy + 32*u_x^3*u_y^2*u_xy*u_xxy +
64*u_x^4*u_y*u_yy*u_xxy + 64*u_x^3*u_y^2*u_xx*u_xxy - 48*u_x^4*u_y*u_xy*u_xxy -
16*u_x^5*u_yy*u_xxy - 16*u_x^4*u_y*u_xx*u_yyy + 16*u_x^5*u_xy*u_yyy +
2*u_y^6*u_xxxx - 12*u_x*u_y^5*u_xxxx + 30*u_x^2*u_y^4*u_xxxx -
40*u_x^3*u_y^3*u_xxxx + 30*u_x^4*u_y^2*u_xxxx - 12*u_x^5*u_y*u_xxxx +
2*u_x^6*u_yyyy}
```

```
[69]: H_hepta2 = cH_hepta2[H]
```

```
[70]: diff(H_hepta2,y) == X_hepta2[0]
```

```
[70]: True
```

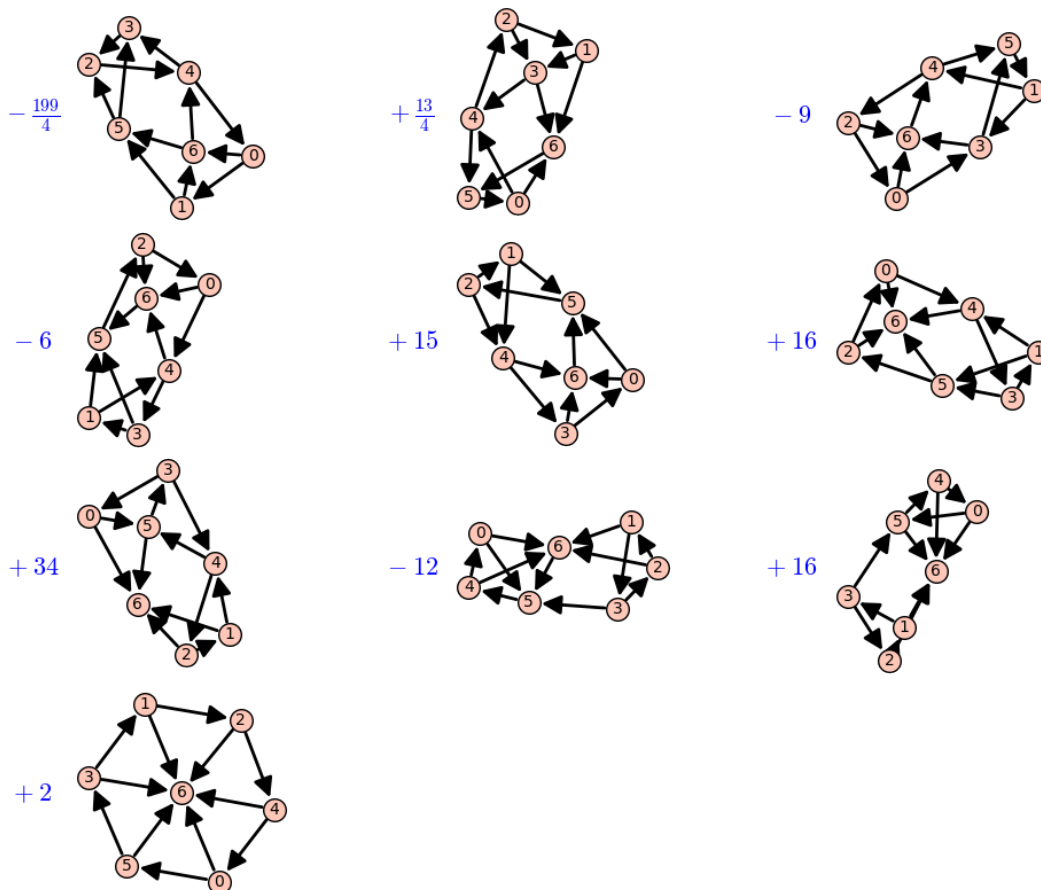
```
[71]: -diff(H_hepta2,x) == X_hepta2[1]
```

```
[71]: True
```

The Hamiltonian  $H$  of the trivializing vector field for the  $\gamma_7$  flow on  $\mathbb{R}^2$  can be realized as a sum of graphs:

```
[72]: ham7_1 = (2)*dfGC(DirectedGraph(7, [(0, 5), (0, 6), (1, 2), (1, 6), (2, 4), (2, 6),
↳(3, 1), (3, 6), (4, 0), (4, 6), (5, 3), (5, 6)])) + (16)*dfGC(DirectedGraph(7, [(0,
↳5), (0, 6), (1, 3), (1, 6), (2, 1), (2, 6), (3, 2), (3, 5), (4, 0), (4, 6), (5, 4),
↳(5, 6)])) + (-12)*dfGC(DirectedGraph(7, [(0, 5), (0, 6), (1, 3), (1, 6), (2, 1),
↳(2, 6), (3, 2), (3, 5), (4, 0), (4, 6), (5, 4), (6, 5)])) +
↳(34)*dfGC(DirectedGraph(7, [(0, 5), (0, 6), (1, 4), (1, 6), (2, 1), (2, 6), (3, 0),
↳(3, 4), (4, 2), (4, 5), (5, 3), (5, 6)])) + (16)*dfGC(DirectedGraph(7, [(0, 4), (0,
↳6), (1, 4), (1, 5), (2, 0), (2, 6), (3, 1), (3, 5), (4, 3), (4, 6), (5, 2), (5,
↳6)])) + (15)*dfGC(DirectedGraph(7, [(0, 5), (0, 6), (1, 4), (1, 5), (2, 1), (2, 4),
↳(3, 0), (3, 6), (4, 3), (4, 6), (5, 2), (6, 5)])) + (-6)*dfGC(DirectedGraph(7, [(0,
↳4), (0, 6), (1, 4), (1, 5), (2, 0), (2, 6), (3, 1), (3, 5), (4, 3), (4, 6), (5, 2),
↳(6, 5)])) + (-9)*dfGC(DirectedGraph(7, [(0, 3), (0, 6), (1, 3), (1, 4), (2, 0), (2,
↳6), (3, 5), (3, 6), (4, 2), (4, 5), (5, 1), (6, 4)])) + (13/
↳4)*dfGC(DirectedGraph(7, [(0, 4), (0, 6), (1, 3), (1, 6), (2, 1), (2, 3), (3, 4),
↳(3, 6), (4, 2), (4, 5), (5, 0), (6, 5)])) + (-199/4)*dfGC(DirectedGraph(7, [(0, 1),
↳(0, 6), (1, 5), (1, 6), (2, 4), (3, 2), (4, 0), (4, 3), (5, 2), (5, 3), (6, 4), (6,
↳5)])); ham7_1.show()
```





```
[73]: S2.graph_operation(ham7_1)(P2,P2,P2,P2,P2,P2,P2)[0,1] == H_hepta2
```

[73]: True

But we discover an alternative realization of the same Hamiltonian on  $\mathbb{R}^2$  by using Konsevich's directed graphs.

```
[74]: ham7_2 = (2)*dfGC(DirectedGraph(7, [(0, 5), (0, 6), (1, 3), (1, 6), (2, 1), (2, 6),
↳(3, 2), (3, 6), (4, 0), (4, 6), (5, 4), (5, 6)])) + (16)*dfGC(DirectedGraph(7, [(0,
↳5), (0, 6), (1, 3), (1, 6), (2, 1), (2, 6), (3, 2), (3, 5), (4, 0), (4, 6), (5, 4),
↳(5, 6)])) + (-12)*dfGC(DirectedGraph(7, [(0, 5), (0, 6), (1, 3), (1, 6), (2, 1),
↳(2, 6), (3, 2), (3, 5), (4, 0), (4, 6), (5, 4), (6, 5)])) +
↳(34)*dfGC(DirectedGraph(7, [(0, 5), (0, 6), (1, 4), (1, 6), (2, 1), (2, 6), (3, 0),
↳(3, 4), (4, 2), (4, 5), (5, 3), (5, 6)])) + (16)*dfGC(DirectedGraph(7, [(0, 4), (0,
↳6), (1, 4), (1, 5), (2, 0), (2, 6), (3, 1), (3, 5), (4, 3), (4, 6), (5, 2), (5,
↳6)])) + (15)*dfGC(DirectedGraph(7, [(0, 5), (0, 6), (1, 4), (1, 5), (2, 1), (2, 4),
↳(3, 0), (3, 6), (4, 3), (4, 6), (5, 2), (6, 5)])) + (-6)*dfGC(DirectedGraph(7, [(0,
↳4), (0, 6), (1, 4), (1, 5), (2, 0), (2, 6), (3, 1), (3, 5), (4, 3), (4, 6), (5, 2),
↳(6, 5)])) + (-9)*dfGC(DirectedGraph(7, [(0, 3), (0, 6), (1, 3), (1, 4), (2, 0), (2,
↳6), (3, 5), (3, 6), (4, 2), (4, 5), (5, 1), (6, 4)])) + (13/
↳4)*dfGC(DirectedGraph(7, [(0, 4), (0, 6), (1, 3), (1, 6), (2, 1), (2, 3), (3, 4),
↳(3, 6), (4, 2), (4, 5), (5, 0), (6, 5)])) + (-199/4)*dfGC(DirectedGraph(7, [(0, 1),
↳(0, 6), (1, 5), (1, 6), (2, 4), (3, 2), (4, 0), (4, 3), (5, 2), (5, 3), (6, 4), (6,
↳5)]))
```

```
[75]: S2.graph_operation(ham7_2)(P2,P2,P2,P2,P2,P2,P2)[0,1] == H_hepta2
```

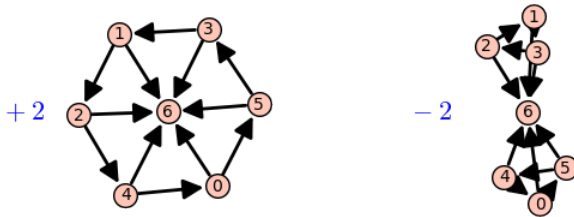
```
[75]: True
```

Let us inspect the difference of two graph realizations for the Hamiltonian  $H_{\gamma_7}$ .

```
[76]: ham7_1 - ham7_2
```

```
[76]: (2)*DirectedGraph(7, [(0, 5), (0, 6), (1, 2), (1, 6), (2, 4), (2, 6), (3, 1), (3, 6),
(4, 0), (4, 6), (5, 3), (5, 6)]) + (-2)*DirectedGraph(7, [(0, 5), (0, 6), (1, 3), (1,
6), (2, 1), (2, 6), (3, 2), (3, 6), (4, 0), (4, 6), (5, 4), (5, 6)])
```

```
[77]: (ham7_1 - ham7_2).show()
```



Let us keep in mind that however that for generic Poisson structures all of these formulas are valid only in dimension two: the trivializing vector field  $X$ , its Hamiltonian  $H$ , and its graph realization.

## 6.5 Commutator $[\gamma_3, \gamma_5]$ flow

```
[39]: bracket_cocycle = tetrahedron_cocycle.bracket(fivewheel_cocycle) #; bracket_cocycle.
↪ show()
```

```
[40]: len(bracket_cocycle)
```

```
[40]: 68
```

```
[ ]: %time bracket_cocycle_directed = dGC(bracket_cocycle)
```

```
[81]: len(bracket_cocycle_directed)
```

```
[81]: 2752512
```

```
[82]: %time bracket_cocycle_directed_filtered = bracket_cocycle_directed.
↪ filter(max_out_degree=2)
```

```
CPU times: user 4min 43s, sys: 14.4 s, total: 4min 58s
Wall time: 4min 58s
```

```
[83]: len(bracket_cocycle_directed_filtered)
```

[83]: 42252

Let us inspect the maximal in-degree of vertices in the directed graphs.

```
[84]: %time max(max(DiGraph(g.edges()).in_degree(k) for k in range(9)) for (c,g) in
↳bracket_cocycle_directed_filtered)
```

```
CPU times: user 12.5 s, sys: 252 ms, total: 12.8 s
Wall time: 12.8 s
```

[84]: 7

Now, with sufficient computational power one could in principle continue, to calculate the oriented graphs on two sinks for the Poisson flow:

```
[ ]: #FGC = FormalityGraphComplex(QQ, lazy=True)
```

```
[ ]: #Q_bracket = FGC(bracket_cocycle_directed_filtered).
↳attach_to_ground((2,2,2,2,2,2,2,2,2))
```

```
[ ]: #len(Q_bracket)
```

One would evaluate it at  $P_2^{\otimes 9}$ :

```
[85]: #bracket_cocycle_operation2 = S2.graph_operation(bracket_cocycle_directed_filtered)
```

```
[86]: #%time Q_bracket2 = bracket_cocycle_operation2(P2,P2,P2,P2,P2,P2,P2,P2,P2)
```

And one would try to find a trivializing vector field:

```
[87]: #cX_bracket2 = solve_homogeneous_diffpoly(Q_bracket2[0,1], PbracketV[0,1], [V0, V1]);
↳cX_bracket2
```

The trivializing vector field could then probably be expressed in terms of graphs, like the others above. In fact, perhaps it would be more efficient to try to find the graphs first. Finally, it would be interesting to compare  $[\text{Or}(\gamma_3), \text{Or}(\gamma_5)]$  with  $\text{Or}([\gamma_3, \gamma_5])$ , also in terms of graphs.



# Chapter 7

## Graph complex action on rank two rescaled Nambu–Poisson structures

We consider a class of Poisson brackets whose coefficients are differential polynomial in the functional parameters  $\rho(x, y, z)$  and  $a(x, y, z)$  on  $\mathbb{R}^3$  or  $\rho(x, y, z, w)$  and  $a_0(x, y, z, w), a_1(x, y, z, w)$  on  $\mathbb{R}^4$ .

### 7.1 Tetrahedral flow on rescaled Nambu–Poisson structures on $\mathbb{R}^3$

By construction,  $\{f, g\} = \rho(x, y, z) \cdot \det \begin{pmatrix} a_x & a_y & a_z \\ f_x & f_y & f_z \\ g_x & g_y & g_z \end{pmatrix}$ , where the smooth function  $a$  is the global Casimir of the Poisson bracket.

We import the relevant functionality from the `gcaops` package:

```
[1]: from gcaops.graph.undirected_graph import UndirectedGraph
     from gcaops.graph.undirected_graph_complex import UndirectedGraphComplex
     from gcaops.graph.directed_graph_complex import DirectedGraphComplex
```

Define the graph complexes:

```
[2]: GC = UndirectedGraphComplex(QQ); GC
```

[2]: Undirected graph complex over Rational Field with Basis consisting of representatives of isomorphism classes of undirected graphs with no automorphisms that induce an odd permutation on edges

```
[3]: dGC = DirectedGraphComplex(QQ, implementation='vector'); dGC
```

[3]: Directed graph complex over Rational Field with Basis consisting of representatives of isomorphism classes of directed graphs with no automorphisms that induce an odd permutation on edges

Define the tetrahedron cocycle  $\gamma_3$ :

```
[4]: tetrahedron_graph = UndirectedGraph(4, [(0,1),(0,2),(0,3),(1,2),(1,3),(2,3)])
     tetrahedron = GC(tetrahedron_graph)
```

```
[5]: tetrahedron_oriented = dGC(tetrahedron); tetrahedron_oriented
```

```
[5]: (24)*DirectedGraph(4, [(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)]) +
(-8)*DirectedGraph(4, [(0, 1), (0, 2), (0, 3), (1, 3), (2, 1), (3, 2)]) +
(-24)*DirectedGraph(4, [(0, 2), (0, 3), (1, 0), (1, 2), (2, 3), (3, 1)]) +
(-8)*DirectedGraph(4, [(0, 2), (0, 3), (1, 0), (1, 3), (2, 1), (2, 3)])
```

```
[6]: tetrahedron_oriented_filtered = tetrahedron_oriented.filter(max_out_degree=2);
↳ tetrahedron_oriented_filtered
```

```
[6]: (-24)*DirectedGraph(4, [(0, 2), (0, 3), (1, 0), (1, 2), (2, 3), (3, 1)]) +
(-8)*DirectedGraph(4, [(0, 2), (0, 3), (1, 0), (1, 3), (2, 1), (2, 3)])
```

### 7.1.1 Superfunction algebra

Define the superfunction algebra:

```
[7]: from gcaops.algebra.differential_polynomial_ring import DifferentialPolynomialRing
```

```
[8]: D3 = DifferentialPolynomialRing(QQ, ('rho', 'a'), ('x', 'y', 'z'),
↳ max_differential_orders=[3+1, 4+1]) #; D3
```

```
[9]: from gcaops.algebra.superfunction_algebra import SuperfunctionAlgebra
```

```
[10]: S3.<xi0,xi1,xi2> = SuperfunctionAlgebra(D3, D3.base_variables()) #; S3
```

```
[11]: rho, a = D3.fibre_variables()
```

```
[12]: xi = S3.gens()
```

```
[13]: X = S3.even_coordinates()
```

Consider a rescaled Nambu–Poisson bi-vector field:

```
[14]: P = rho*sum(sigma.sign()*diff(a,X[sigma(1)-1])*xi[sigma(2)-1]*xi[sigma(3)-1] for
↳ sigma in Permutations(3))/2; P
```

```
[14]: (rho*a_x)*xi1*xi2 + (-rho*a_y)*xi0*xi2 + (rho*a_z)*xi0*xi1
```

The Jacobi identity is satisfied:

```
[15]: P.bracket(P)
```

```
[15]: 0
```

The bracket is a “derived” bracket  $P = \llbracket \rho \xi_1 \xi_2 \xi_3, a \rrbracket$ :

```
[16]: P == (rho*xi[0]*xi[1]*xi[2]).bracket(a)
```

```
[16]: True
```

### 7.1.2 Tetrahedral flow

First example by Kontsevich (1996, revised 2017). Define the tetrahedral flow:

```
[17]: # NOTE: filtered out graphs with out degree > 2, because we won't pass in any
      ↪ 3-vectors
      tetrahedron_operation3 = S3.graph_operation(tetrahedron_oriented_filtered);
      ↪ tetrahedron_operation3
```

[17]: Symmetric operation of arity 4 and degree -6 on Superfunction algebra over Differential Polynomial Ring in  $x, y, z, \rho, a, \rho_x, \rho_y, \rho_z, \rho_{xx}, \rho_{xy}, \rho_{xz}, \rho_{yy}, \rho_{yz}, \rho_{zz}, \rho_{xxx}, \rho_{xxy}, \rho_{xxz}, \rho_{xyy}, \rho_{xyz}, \rho_{xzz}, \rho_{yyy}, \rho_{yyz}, \rho_{yzz}, \rho_{zzz}, \rho_{xxxx}, \rho_{xxxy}, \rho_{xxxz}, \rho_{xxyy}, \rho_{xxyz}, \rho_{xxzz}, \rho_{xyyy}, \rho_{xyyz}, \rho_{xyzz}, \rho_{xzzz}, \rho_{yyyy}, \rho_{yyyz}, \rho_{yyzz}, \rho_{yzzz}, \rho_{zzzz}, a_x, a_y, a_z, a_{xx}, a_{xy}, a_{xz}, a_{yy}, a_{yz}, a_{zz}, a_{xxx}, a_{xxy}, a_{xxz}, a_{xyy}, a_{xyz}, a_{xzz}, a_{yyy}, a_{yyz}, a_{yzz}, a_{zzz}, a_{xxxx}, a_{xxxy}, a_{xxxz}, a_{xxyy}, a_{xxyz}, a_{xxzz}, a_{xyyy}, a_{xyyz}, a_{xyzz}, a_{xzzz}, a_{yyyy}, a_{yyyz}, a_{yyzz}, a_{yzzz}, a_{zzzz}, a_{xxxxx}, a_{xxxxy}, a_{xxxzx}, a_{xxxxy}, a_{xxxzy}, a_{xxxzz}, a_{xxyyy}, a_{xxyyz}, a_{xxyzz}, a_{xyyyy}, a_{xyyyz}, a_{xyyzz}, a_{xyzzz}, a_{xyzzz}, a_{yyyyy}, a_{yyyyz}, a_{yyyzz}, a_{yyzzz}, a_{yzzzz}, a_{zzzzz}$  over Rational Field with even coordinates  $(x, y, z)$  and odd coordinates  $(xi_0, xi_1, xi_2)$

```
[18]: %time Q_tetra3 = tetrahedron_operation3(P,P,P,P)
```

```
CPU times: user 297 ms, sys: 63 μs, total: 297 ms
Wall time: 296 ms
```

Both terms in the tetrahedral flow are generally nonzero:

```
[19]: #Q_tetra3
```

```
[20]: len(Q_tetra3[0,1].monomials()), len(Q_tetra3[0,2].monomials()), len(Q_tetra3[1,2]).
      ↪ monomials()
```

```
[20]: (1504, 1504, 1504)
```

The tetrahedral flow indeed defines a Poisson 2-cocycle:

```
[21]: P.bracket(Q_tetra3)
```

```
[21]: 0
```

### 7.1.3 The induced flow

We have  $Q_{\text{tetra}}(P[\rho, a]^{\otimes 4}) = P[\dot{\rho}, a] + P[\rho, \dot{a}]$  for differential polynomials  $\dot{\rho}$  and  $\dot{a}$ .

In fact  $\dot{a} = 4 \cdot Q_{\text{tetra}}(P, P, P, a)$ .

```
[22]: %time adot = 4 * tetrahedron_operation3(P,P,P,a)
```

```
CPU times: user 95 ms, sys: 3.91 ms, total: 98.9 ms
Wall time: 97.9 ms
```

```
[23]: #adot
```

```
[24]: len(adot[tuple()].monomials())
```

[24]: 228

```
[25]: P0 = rho*sum(sigma.sign()*diff(adot,X[sigma(1)-1])*xi[sigma(2)-1]*xi[sigma(3)-1] for sigma in Permutations(3))/2
```

```
[26]: Q_remainder = Q_tetra3 - P0
```

```
[27]: len(Q_remainder[0,1].monomials()), len(Q_remainder[0,2].monomials()), len(Q_remainder[1,2].monomials())
```

[27]: (426, 426, 426)

```
[28]: P_withoutprefactor = sum(sigma.sign()*diff(a,X[sigma(1)-1])*xi[sigma(2)-1]*xi[sigma(3)-1] for sigma in Permutations(3))/2; P_withoutprefactor
```

[28]: (a\_x)\*xi1\*xi2 + (-a\_y)\*xi0\*xi2 + (a\_z)\*xi0\*xi1

Get  $\rho$  by (differential) polynomial division:

```
[29]: Q_remainder[0,1] % P_withoutprefactor[0,1] == 0
```

[29]: True

```
[30]: rhodot = Q_remainder[0,1] // P_withoutprefactor[0,1]
```

```
[31]: len(rhodot.monomials())
```

[31]: 426

```
[32]: rhodot * P_withoutprefactor == Q_remainder
```

[32]: True

```
[33]: Q_tetra3 == rhodot * P_withoutprefactor + P0
```

[33]: True

```
[34]: #rhodot
```

```
[35]: #%time sol = solve_homogeneous_diffpoly(D4('rhodot')*D4(P_withoutprefactor[0,1]), jQ_remainder01, [rhodot])
```

### 7.1.4 Symmetry

Sum over permutations:

```
[36]: adot_maybe = 0
from itertools import product
for sigma, tau, zeta in product(SymmetricGroup(3), repeat=3):
    u1,v1,w1 = X[sigma(1)-1], X[sigma(2)-1], X[sigma(3)-1]
    u2,v2,w2 = X[tau(1)-1], X[tau(2)-1], X[tau(3)-1]
    u3,v3,w3 = X[zeta(1)-1], X[zeta(2)-1], X[zeta(3)-1]
    adot_maybe += sigma.sign() * tau.sign() * zeta.sign() * (\
```



```

2*diff(a,u1) * diff(a,u2) * diff(a,u3) * diff(rho,w1) * diff(rho,w2) *
↪diff(rho,w3) * diff(a,v1,v2,v3) + \
-6*rho * diff(a,u1,v2) * diff(a,u2) * diff(a,u3) * diff(rho,w1) *
↪diff(rho,w3) * diff(a,v1,v3,w2) + \
-6*rho * rho*diff(a,u1) * diff(a,u2,u3) * diff(a,v1,v2) * diff(rho,w3) *
↪diff(a,v3,w1,w2)
)
adot_maybe == adot/4

```

[36]: True

```

[37]: rhodot_maybe = 0
from itertools import product
for sigma, tau, zeta in product(SymmetricGroup(3), repeat=3):
    u1,v1,w1 = X[sigma(1)-1], X[sigma(2)-1], X[sigma(3)-1]
    u2,v2,w2 = X[tau(1)-1], X[tau(2)-1], X[tau(3)-1]
    u3,v3,w3 = X[zeta(1)-1], X[zeta(2)-1], X[zeta(3)-1]
    rhodot_maybe += sigma.sign() * tau.sign() * zeta.sign() * (
-2*diff(a,u1) * diff(a,u2) * diff(a,u3) * diff(rho,v1) * diff(rho,v2) *
↪diff(rho,v3) * diff(rho,w1,w2,w3) + \
6 * diff(a,u1,v2) * diff(a,u2) * diff(a,u3) * diff(rho,v1) * diff(rho,v3) *
↪diff(rho,w2) * diff(rho,w1,w3) + \
-12*rho * diff(a,u1) * diff(a,u2,u3) * diff(a,v1,v2) * diff(rho,v3) *
↪diff(rho,w1) * diff(rho,w2,w3) + \
-6*rho * diff(a,u1,v2) * diff(a,u2) * diff(a,u3) * diff(rho,v1) *
↪diff(rho,v3) * diff(rho,w1,w2,w3) + \
6*rho * rho*diff(a,u1) * diff(a,u2,u3) * diff(a,v1,v2) * diff(rho,v3) *
↪diff(rho,w1,w2,w3)
)
rhodot_maybe == rhodot/4

```

[37]: True

### 7.1.5 Differential polynomial triviality

The Poisson cohomology of  $\mathbb{R}^3$  is generally non-trivial.

Is the tetrahedral flow non-trivial, for the rescaled Nambu–Poisson bracket  $P[\rho, a]$  on  $\mathbb{R}^3$ ?

First, let  $V$  be an arbitrary vector field:

```

[38]: D3V = DifferentialPolynomialRing(QQ, ('rho', 'a', 'V0', 'V1', 'V2', 'H'), ('x', 'y', 'z'),
↪max_differential_orders=[3+1,4+1,1,1,1,1]) #; D3V

```

```

[39]: V0, V1, V2, H = D3V.fibre_variables()[2:]

```

```

[40]: S3V = SuperfunctionAlgebra(D3V, D3V.base_variables(), names='xi0,xi1,xi2') #; S3V

```

```

[41]: V = V0*S3V(xi0) + V1*S3V(xi1) + V2*S3V(xi2); V

```

```

[41]: (V0)*xi0 + (V1)*xi1 + (V2)*xi2

```

Take the Poisson differential:

```

[42]: PbracketV = S3V(P).bracket(V); PbracketV

```

```
[42]: (V0*rho_x*a_y + V1*rho_y*a_y + V2*rho_z*a_y + rho*V0*a_xy + rho*V1*a_yy + rho*V2*a_yz
- rho*a_y*V0_x + rho*a_x*V0_y + rho*a_z*V2_y - rho*a_y*V2_z)*xi0*xi2 + (-V0*rho_x*a_x
- V1*rho_y*a_x - V2*rho_z*a_x - rho*V0*a_xx - rho*V1*a_xy - rho*V2*a_xz - rho*a_y*V1_x
+ rho*a_x*V1_y - rho*a_z*V2_x + rho*a_x*V2_z)*xi1*xi2 + (-V0*rho_x*a_z - V1*rho_y*a_z
- V2*rho_z*a_z - rho*V0*a_xz - rho*V1*a_yz - rho*V2*a_zz + rho*a_z*V0_x - rho*a_x*V0_z
+ rho*a_z*V1_y - rho*a_y*V1_z)*xi0*xi1
```

Solve  $Q_{\text{tetra}}(P, P, P, P) = \llbracket P, V \rrbracket$  for  $V$  with differential polynomial coefficients by using homogeneity:

```
[43]: from gcaops.algebra.differential_polynomial_solver import solve_homogeneous_diffpoly
```

```
[44]: set_verbose(1)
```

```
[45]: %time sol = solve_homogeneous_diffpoly(S3V(Q_tetra3)[0,1], PbracketV[0,1], [V0,V1,V2])
```

```
verbose 1 (12: differential_polynomial_solver.py, solve_homogeneous_diffpoly) target
degrees: (4, 4, 0, 0, 0, 0)
verbose 1 (12: differential_polynomial_solver.py, solve_homogeneous_diffpoly) target
weights: (3, 3, 4)
verbose 1 (12: differential_polynomial_solver.py, solve_homogeneous_diffpoly) ansatz
degrees: {V1: {(3, 3, 0, 0, 0, 0)}, V0: {(3, 3, 0, 0, 0, 0)}, V2: {(3, 3, 0, 0, 0,
0)}}
verbose 1 (12: differential_polynomial_solver.py, solve_homogeneous_diffpoly) ansatz
weights: {V1: {(3, 2, 3)}, V0: {(2, 3, 3)}, V2: {(3, 3, 2)}}
verbose 1 (12: differential_polynomial_solver.py, solve_homogeneous_diffpoly) ansatz
#monomials: {V0: 2843, V1: 2843, V2: 2843}
verbose 1 (12: differential_polynomial_solver.py, solve_homogeneous_diffpoly)
len(target_basis) == 17085
verbose 1 (12: differential_polynomial_solver.py, solve_homogeneous_diffpoly)
len(ansatz_basis) == 7477
verbose 1 (12: differential_polynomial_solver.py, multimod echelon) Multimodular
echelon algorithm on 17085 x 7477 matrix
verbose 1 (12: differential_polynomial_solver.py, sparse_matrix_pyx matrix_modint
echelon)
verbose 1 (12: differential_polynomial_solver.py, sparse_matrix_pyx matrix_modint
echelon)
verbose 1 (12: differential_polynomial_solver.py, sparse_matrix_pyx matrix_modint
echelon)
verbose 1 (12: differential_polynomial_solver.py, multimod echelon) Multimodular
echelon algorithm on 7332 x 17085 matrix
verbose 1 (12: differential_polynomial_solver.py, sparse_matrix_pyx matrix_modint
echelon)
verbose 1 (12: differential_polynomial_solver.py, sparse_matrix_pyx matrix_modint
echelon)
verbose 1 (12: differential_polynomial_solver.py, sparse_matrix_pyx matrix_modint
echelon)
verbose 1 (12: differential_polynomial_solver.py, sparse_matrix_pyx matrix_modint
echelon)
verbose 1 (12: differential_polynomial_solver.py, sparse_matrix_pyx matrix_modint
echelon)
verbose 1 (12: differential_polynomial_solver.py, sparse_matrix_pyx matrix_modint
echelon)
verbose 1 (12: differential_polynomial_solver.py, multimod echelon) Multimodular
echelon algorithm on 7332 x 7332 matrix
```

```

verbose 1 (12: differential_polynomial_solver.py, sparse_matrix_pyx matrix_modint
echelon)
verbose 1 (12: differential_polynomial_solver.py, multimod echelon) done: the echelon
form mod p is the identity matrix and possibly some 0 rows
verbose 1 (12: differential_polynomial_solver.py, multimod echelon) Multimodular
echelon algorithm on 7332 x 7333 matrix
verbose 1 (12: differential_polynomial_solver.py, sparse_matrix_pyx matrix_modint
echelon)
verbose 1 (12: differential_polynomial_solver.py, sparse_matrix_pyx matrix_modint
echelon)
verbose 1 (12: differential_polynomial_solver.py, sparse_matrix_pyx matrix_modint
echelon)
verbose 1 (12: differential_polynomial_solver.py, sparse_matrix_pyx matrix_modint
echelon)
CPU times: user 25min 29s, sys: 1.93 s, total: 25min 30s
Wall time: 25min 29s

```

```
[46]: V_tetra3 = D3(sol[V0])*xi0 + D3(sol[V1])*xi1 + D3(sol[V2])*xi2 #; V_tetra3
```

```
[47]: tuple(len(V_tetra3[i].monomials()) for i in range(3))
```

```
[47]: (352, 347, 339)
```

```
[48]: P.bracket(V_tetra3) == Q_tetra3
```

```
[48]: True
```

So, the tetrahedral flow of a rescaled Nambu–Poisson structure on  $\mathbb{R}^3$  is Poisson-trivial!

Moreover, by running the script several times one can get many different such trivializing vector fields — evidently, they are defined modulo Poisson coboundaries  $\llbracket P, H \rrbracket$  for arbitrary Hamiltonian functions  $H$  (in particular homogeneous differential polynomial ones).

### 7.1.6 Total skew-symmetry of the trivializing vector field

The trivializing vector field is expressed as a sum (of 11 terms) over permutations with signs plus a Poisson exact-term given by a Hamiltonian function:

$$V = \sum_{\sigma, \tau, \zeta \in S_3} (-)^\sigma (-)^\tau (-)^\zeta (\sigma \otimes \tau \otimes \zeta) (11 \text{ terms in vector field}) + \llbracket P, H_{\gamma_3} \rrbracket.$$

Each of the permutations  $\sigma, \tau, \zeta$  acts on its own triple of base variables,  $\sigma: (x, y, z) \mapsto (\sigma(x), \sigma(y), \sigma(z))$  etc., thus reproducing three Civita symbols and three sums  $\sum_{\vec{i}} \varepsilon^{\vec{i}} \partial_{i_1} \otimes \partial_{i_2} \otimes \partial_{i_3}$ .

```
[49]: V_skew = 0
from itertools import product
for sigma, tau, zeta in product(SymmetricGroup(3), repeat=3):
    i_0, i_1, i_2 = sigma(1)-1, sigma(2)-1, sigma(3)-1
    i_3, i_4, i_5 = tau(1)-1, tau(2)-1, tau(3)-1
    i_6, i_7, i_8 = zeta(1)-1, zeta(2)-1, zeta(3)-1
    V_skew += sigma.sign() * tau.sign() * zeta.sign() * (\
```

```

12 * diff(rho, X[i_0], X[i_3]) * diff(rho, X[i_7]) * rho * diff(a, X[i_1],
↪X[i_4]) * diff(a, X[i_2], X[i_5]) * diff(a, X[i_8]) * xi[i_6] + \
48 * diff(rho, X[i_0], X[i_3]) * diff(rho, X[i_5]) * rho * diff(a, X[i_1],
↪X[i_4]) * diff(a, X[i_2], X[i_6]) * diff(a, X[i_8]) * xi[i_7] + \
8 * diff(rho, X[i_0], X[i_6]) * diff(rho, X[i_1], X[i_7]) * diff(rho, X[i_4])
↪* diff(a, X[i_2]) * diff(a, X[i_5]) * diff(a, X[i_8]) * xi[i_3] + \
-40 * diff(rho, X[i_0], X[i_6]) * diff(rho, X[i_2]) * diff(rho, X[i_4]) *
↪diff(a, X[i_1], X[i_7]) * diff(a, X[i_5]) * diff(a, X[i_8]) * xi[i_3] + \
8 * diff(rho, X[i_2]) * diff(rho, X[i_4]) * diff(rho, X[i_8]) * diff(a,
↪X[i_0], X[i_6]) * diff(a, X[i_1], X[i_7]) * diff(a, X[i_5]) * xi[i_3] + \
24 * diff(rho, X[i_0], X[i_6]) * diff(rho, X[i_4]) * diff(rho, X[i_8]) *
↪diff(a, X[i_2]) * diff(a, X[i_3], X[i_7]) * diff(a, X[i_5]) * xi[i_1] + \
-12 * diff(rho, X[i_7]) * rho * rho * diff(a, X[i_0], X[i_3]) * diff(a,
↪X[i_1], X[i_4]) * diff(a, X[i_2], X[i_5], X[i_8]) * xi[i_6] + \
24 * diff(rho, X[i_4]) * diff(rho, X[i_6]) * rho * diff(a, X[i_0], X[i_3]) *
↪diff(a, X[i_7]) * diff(a, X[i_2], X[i_5], X[i_8]) * xi[i_1] + \
-36 * diff(rho, X[i_1]) * diff(rho, X[i_4]) * rho * diff(a, X[i_0], X[i_3]) *
↪diff(a, X[i_7]) * diff(a, X[i_2], X[i_5], X[i_8]) * xi[i_6] + \
8 * diff(rho, X[i_1]) * diff(rho, X[i_3]) * diff(rho, X[i_6]) * diff(a,
↪X[i_4]) * diff(a, X[i_7]) * diff(a, X[i_2], X[i_5], X[i_8]) * xi[i_0] + \
-8 * diff(rho, X[i_3]) * diff(rho, X[i_6]) * diff(rho, X[i_2], X[i_5],
↪X[i_8]) * diff(a, X[i_1]) * diff(a, X[i_4]) * diff(a, X[i_7]) * xi[i_0]
)
#V_skew

```

```
[50]: tuple(len(V_skew[i].monomials()) for i in range(3))
```

```
[50]: (324, 324, 324)
```

```
[51]: Q_tetra3 == P.bracket(V_skew)
```

```
[51]: True
```

Here is an example of the Hamiltonian  $H_{\gamma_3}$  (consisting of 20 monomials) which was obtained for the trivializing vector field  $V$  from long ago:  $V - V_{skew} = \llbracket P, H_{\gamma_3} \rrbracket$ :

```

[52]: #H_tetra3 = D3('12*rho_z^2*a_xy^2 - 24*rho*rho_zz*a_xy^2 - 48*rho_y*rho_xz*a_y*a_xz -
↪48*rho_y*rho_z*a_xy*a_xz + 24*rho_y^2*a_xz^2 - 12*rho_z^2*a_xx*a_yy +
↪48*rho_x*rho_z*a_xz*a_yy + 48*rho_y*rho_z*a_xx*a_yz - 48*rho_x*rho_z*a_xy*a_yz -
↪48*rho_x*rho_y*a_xz*a_yz + 36*rho_x^2*a_yz^2 - 24*rho_y^2*a_xx*a_zz +
↪24*rho*rho_yy*a_xx*a_zz + 48*rho_x*rho_y*a_xy*a_zz - 24*rho_x^2*a_yy*a_zz -
↪12*rho*rho_y*a_zz*a_xxy + 24*rho*rho_yz*a_y*a_xxz - 24*rho*rho_z*a_yy*a_xxz -
↪24*rho_y^2*a_x*a_xzz - 24*rho*rho_z*a_xx*a_yyz')

```

But as the representative of  $[V \bmod \llbracket P, \cdot \rrbracket]$  typically changes every time the solver is invoked, we shall find another  $H_{\gamma_3}$  in the next step, for the representative  $V$  which we currently have.

```
[53]: set_verbose(0)
```

```
[97]: H_tetra3 = solve_homogeneous_diffpoly(S3V(V_tetra3 - V_skew)[0], S3V(P).
↪bracket(H)[0], [H])[H]
```

```
[98]: H_tetra3
```

```
[98]: 24*rho_z^2*a_xy^2 - 96*rho_y*rho_xy*a_z*a_xz - 48*rho_y*rho_z*a_xy*a_xz +
48*rho*rho_yz*a_xy*a_xz + 12*rho_y^2*a_xz^2 - 24*rho_z^2*a_xx*a_yy +
48*rho_y*rho_z*a_xx*a_yz - 48*rho_x*rho_z*a_xy*a_yz + 48*rho*rho_xz*a_xy*a_yz +
96*rho_x*rho_y*a_xz*a_yz + 24*rho_x^2*a_yz^2 - 12*rho_y^2*a_xx*a_zz +
48*rho_x*rho_y*a_xy*a_zz - 12*rho_x^2*a_yy*a_zz + 12*rho_z^2*a_y*a_xxy +
48*rho_y^2*a_z*a_xxz + 24*rho*rho_z*a_yy*a_xxz + 12*rho_z^2*a_x*a_xyy +
48*rho_x*rho_z*a_z*a_xyy - 48*rho*rho_xz*a_z*a_xyy + 24*rho_x^2*a_z*a_yyz -
24*rho*rho_z*a_xx*a_yyz
```

```
[99]: len(H_tetra3.monomials())
```

```
[99]: 22
```

In contrast with the previous run  $Q_{\text{tetra}} \rightarrow V \rightarrow H$  (see the `#H_tetra3 = ...` cell above), this new Hamiltonian –for the new representative  $V$ – of trivializing vector field contains only 22 monomials.

```
[57]: V_tetra3 == V_skew + S3V(P).bracket(H_tetra3)
```

```
[57]: True
```

Our crucial remark is that the marker monomials for the velocity  $\dot{a}$  of the Casimir, now induced by  $\dot{a} = -V(a)$  with a totally skew-symmetric  $V_{\text{skew-part}}$ , cannot change for different representatives of  $[V \bmod \llbracket P, \cdot \rrbracket]$ .

```
[58]: -V_skew.bracket(a) == adot
```

```
[58]: True
```

```
[59]: -V_tetra3.bracket(a) == adot
```

```
[59]: True
```

We see that the expression  $-V_{\text{skew}}(a)$  equals  $\dot{a}$ .

Moreover,  $\rho \xi_1 \xi_2 \xi_3 = -\llbracket V_{\text{skew}}, \rho \xi_1 \xi_2 \xi_3 \rrbracket$ :

```
[60]: -V_skew.bracket(rho*xi[0]*xi[1]*xi[2])[0,1,2] == rhodot
```

```
[60]: True
```

## 7.2 Tetrahedral flow on Nambu–Poisson structures on $\mathbb{R}^4$

We have

$$\{f, g\} = \det \left( \frac{\partial(a_0, a_1, f, g)}{\partial(x, y, z, w)} \right).$$

We import the relevant functionality from the `gcaops` package:

```
[1]: from gcaops.graph.undirected_graph import UndirectedGraph
from gcaops.graph.undirected_graph_complex import UndirectedGraphComplex
```

```
[2]: GC = UndirectedGraphComplex(QQ); GC
```

[2]: Undirected graph complex over Rational Field with Basis consisting of representatives of isomorphism classes of undirected graphs with no automorphisms that induce an odd permutation on edges

Define the tetrahedron cocycle  $\gamma_3$ :

```
[3]: tetrahedron_graph = UndirectedGraph(4, [(0,1),(0,2),(0,3),(1,2),(1,3),(2,3)]);
↳ tetrahedron_graph
```

```
[3]: UndirectedGraph(4, [(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)])
```

```
[4]: tetrahedron = GC(tetrahedron_graph); tetrahedron
```

```
[4]: 1*UndirectedGraph(4, [(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)])
```

```
[5]: from gcaops.graph.directed_graph_complex import DirectedGraphComplex
```

```
[6]: dGC = DirectedGraphComplex(QQ, implementation='vector')
```

```
[7]: tetrahedron_oriented = dGC(tetrahedron); tetrahedron_oriented
```

```
[7]: (24)*DirectedGraph(4, [(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)]) +
(-8)*DirectedGraph(4, [(0, 1), (0, 2), (0, 3), (1, 3), (2, 1), (3, 2)]) +
(-24)*DirectedGraph(4, [(0, 2), (0, 3), (1, 0), (1, 2), (2, 3), (3, 1)]) +
(-8)*DirectedGraph(4, [(0, 2), (0, 3), (1, 0), (1, 3), (2, 1), (2, 3)])
```

```
[8]: tetrahedron_oriented_filtered = tetrahedron_oriented.filter(max_out_degree=2);
↳ tetrahedron_oriented_filtered
```

```
[8]: (-24)*DirectedGraph(4, [(0, 2), (0, 3), (1, 0), (1, 2), (2, 3), (3, 1)]) +
(-8)*DirectedGraph(4, [(0, 2), (0, 3), (1, 0), (1, 3), (2, 1), (2, 3)])
```

The differential polynomial ring with fibre variables  $a_0, a_1$  and base variables  $x, y, z, w$ :

```
[9]: from gcaops.algebra.differential_polynomial_ring import DifferentialPolynomialRing
```

```
[10]: D4 = DifferentialPolynomialRing(QQ, ('a0','a1'), ('x','y','z','w')),
↳ max_differential_orders=[4+1,4+1]) #; D4
```

```
[13]: a0, a1 = D4.fibre_variables()
```

```
[14]: a = [a0,a1]
```

The superfunction algebra:

```
[11]: from gcaops.algebra.superfunction_algebra import SuperfunctionAlgebra
```

```
[12]: S4.<xi0,xi1,xi2,xi3> = SuperfunctionAlgebra(D4, D4.base_variables()) #; S4
```

```
[15]: xi = S4.gens()
```

```
[16]: X = S4.even_coordinates()
```

The Nambu–Poisson structure:

```
[17]: P = sum(sigma.sign()*diff(a[0],X[sigma(1)-1])*diff(a[1],
↳X[sigma(2)-1])*xi[sigma(3)-1]*xi[sigma(4)-1] for sigma in Permutations(4))/2; P
```

```
[17]: (-a0_y*a1_x + a0_x*a1_y)*xi2*xi3 + (a0_z*a1_x - a0_x*a1_z)*xi1*xi3 + (-a0_w*a1_x +
a0_x*a1_w)*xi1*xi2 + (-a0_z*a1_y + a0_y*a1_z)*xi0*xi3 + (a0_w*a1_y -
a0_y*a1_w)*xi0*xi2 + (-a0_w*a1_z + a0_z*a1_w)*xi0*xi1
```

In fact the Poisson bracket is a “derived” bracket  $P = -[[[\partial_x \wedge \partial_y \wedge \partial_z \wedge \partial_w, a_0], a_1]]$ :

```
[18]: P == -(xi0*xi1*xi2*xi3).bracket(a0).bracket(a1)
```

```
[18]: True
```

```
[19]: P.bracket(P)
```

```
[19]: 0
```

```
[20]: # NOTE: filtered out graphs with out degree > 2, because we won't pass in any
↳3-vectors
tetrahedron_operation4 = S4.graph_operation(tetrahedron_oriented_filtered) #;
↳tetrahedron_operation4
```

The evolutions  $\dot{a}_0$  and  $\dot{a}_1$  of  $a_0$  and  $a_1$ :

```
[21]: %time adot = [4*tetrahedron_operation4(P,P,P,a[i]) for i in range(2)]
```

```
CPU times: user 12.8 s, sys: 58 ms, total: 12.9 s
Wall time: 12.9 s
```

```
[22]: #adot[0]
```

```
[23]: #adot[1]
```

```
[24]: len(adot[0][tuple()].monomials()), len(adot[1][tuple()].monomials())
```

```
[24]: (9024, 9024)
```

The flow  $Q_{\text{tetra}}(P[a_0, a_1])$ :

```
[25]: %time Q_tetra4 = tetrahedron_operation4(P,P,P,P)
```

```
CPU times: user 7min 45s, sys: 1min 20s, total: 9min 6s
Wall time: 9min 6s
```

```
[26]: [len(Q_tetra4[i,j].monomials()) for i,j in [(0,1),(0,2),(0,3),(1,2),(1,3),(2,3)]]
```

```
[26]: [161040, 161040, 161040, 161040, 161040, 161040]
```

We have the equality  $Q_{\text{tetra}}(P[a_0, a_1]) = P[\dot{a}_0, a_1] + P[a_0, \dot{a}_1]$ :

```
[27]: P0 = sum(sigma.sign()*diff(adot[0],X[sigma(1)-1])*diff(a[1],
↳X[sigma(2)-1])*xi[sigma(3)-1]*xi[sigma(4)-1] for sigma in Permutations(4))/2
```

```
[28]: P1 = sum(sigma.sign()*diff(a[0],X[sigma(1)-1])*diff(adot[1],
↳X[sigma(2)-1])*xi[sigma(3)-1]*xi[sigma(4)-1] for sigma in Permutations(4))/2
```

```
[29]: Q_remainder = Q_tetra4 - P0 - P1; Q_remainder
```

```
[29]: 0
```

```
[30]: Q_tetra4 == P0 + P1
```

```
[30]: True
```

## 7.2.1 Symmetry

The evolutions  $\dot{a}_0$  and  $\dot{a}_1$  are induced by the tetrahedral flow  $Q_{\gamma_3}(P^{\otimes 4})$  on the class  $P[a_0, a_1]$  of Nambu–Poisson brackets on  $\mathbb{R}^4$  with the pre-factor  $\rho = 1$ . We represent either velocity by using three Civita symbols (with four indices each).

```
[31]: a0dot_maybe = 0
from itertools import product
for sigma, tau, zeta in product(SymmetricGroup(4), repeat=3):
    s1,t1,u1,v1 = X[sigma(1)-1], X[sigma(2)-1], X[sigma(3)-1], X[sigma(4)-1]
    s2,t2,u2,v2 = X[tau(1)-1], X[tau(2)-1], X[tau(3)-1], X[tau(4)-1]
    s3,t3,u3,v3 = X[zeta(1)-1], X[zeta(2)-1], X[zeta(3)-1], X[zeta(4)-1]
    a0dot_maybe += sigma.sign() * tau.sign() * zeta.sign() * (
        3*diff(a0,s1,u2,u3) * diff(a0,t1,t2) * diff(a1,s2) * diff(a1,s3,v1) *
↳diff(a1,t3,u1) * diff(a0,v2) * diff(a0,v3) + \
        +6*diff(a0,s1,u2) * diff(a0,t1) * diff(a0,t2,v3) * diff(a0,u3,v1,v2) *
↳diff(a1,t3,u1) * diff(a1,s2) * diff(a1,s3)
    )
a0dot_maybe == adot[0]/4
```

```
[31]: True
```

```
[32]: a1dot_maybe = 0
from itertools import product
for sigma, tau, zeta in product(SymmetricGroup(4), repeat=3):
    s1,t1,u1,v1 = X[sigma(1)-1], X[sigma(2)-1], X[sigma(3)-1], X[sigma(4)-1]
    s2,t2,u2,v2 = X[tau(1)-1], X[tau(2)-1], X[tau(3)-1], X[tau(4)-1]
    s3,t3,u3,v3 = X[zeta(1)-1], X[zeta(2)-1], X[zeta(3)-1], X[zeta(4)-1]
    a1dot_maybe += sigma.sign() * tau.sign() * zeta.sign() * (
        3*diff(a0,s1) * diff(a1,t1,u2) * diff(a1,u1,u3,v2) * diff(a0,s2,t3) *
↳diff(a0,s3,t2) * diff(a1,v1) * diff(a1,v3) + \
        -3*diff(a0,s1,t2) * diff(a1,u1) * diff(a1,u2,u3,v1) * diff(a0,t1) *
↳diff(a0,t3) * diff(a1,s2,v3) * diff(a1,s3,v2)
    )
a1dot_maybe == adot[1]/4
```

```
[32]: True
```

## 7.2.2 Differential polynomial (non)triviality

One could try to solve the equation  $Q_{\text{tetra}}(P[a_0, a_1]) = \llbracket P, V[a_0, a_1] \rrbracket$  for a vector field  $V[a_0, a_1]$  as follows:



```
[33]: D4V = DifferentialPolynomialRing(QQ, ('a0', 'a1', 'V0', 'V1', 'V2', 'V3'),
    ↪ ('x', 'y', 'z', 'w'), max_differential_orders=[4+1,4+1,1,1,1,1]) #; D4V

[34]: V0, V1, V2, V3 = D4V.fibre_variables()[2:]

[35]: S4V = SuperfunctionAlgebra(D4V, D4V.base_variables(), names='xi0,xi1,xi2,xi3') #; S4V

[36]: V = V0*S4V(xi0) + V1*S4V(xi1) + V2*S4V(xi2) + V3*S4V(xi3); V

[36]: (V0)*xi0 + (V1)*xi1 + (V2)*xi2 + (V3)*xi3

[37]: PbracketV = S4V(P).bracket(V)

[38]: from gcaops.algebra.differential_polynomial_solver import solve_homogeneous_diffpoly

[39]: set_verbose(1)

[ ]: %time sol = solve_homogeneous_diffpoly(S4V(Q_tetra4)[0,1], PbracketV[0,1],
    ↪ [V0,V1,V2,V3])
```

So far, the (non)triviality of the flow  $Q_{\text{tetra}}(P[a_0, a_1])$  remains an open problem.

## 7.3 Tetrahedral flow on rescaled Nambu–Poisson structures on $\mathbb{R}^4$

We have

$$\{f, g\} = \rho(x, y, z, w) \cdot \det \left( \frac{\partial(a_0, a_1, f, g)}{\partial(x, y, z, w)} \right).$$

We import the relevant functionality from the `gcaops` package:

```
[1]: from gcaops.graph.undirected_graph import UndirectedGraph
    from gcaops.graph.undirected_graph_complex import UndirectedGraphComplex

[2]: GC = UndirectedGraphComplex(QQ); GC

[2]: Undirected graph complex over Rational Field with Basis consisting of representatives
of isomorphism classes of undirected graphs with no automorphisms that induce an odd
permutation on edges

Define the tetrahedron cocycle  $\gamma_3$ :

[3]: tetrahedron_graph = UndirectedGraph(4, [(0,1),(0,2),(0,3),(1,2),(1,3),(2,3)]);
    ↪ tetrahedron_graph

[3]: UndirectedGraph(4, [(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)])

[4]: tetrahedron = GC(tetrahedron_graph); tetrahedron

[4]: 1*UndirectedGraph(4, [(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)])

[5]: from gcaops.graph.directed_graph_complex import DirectedGraphComplex
```

```
[6]: dGC = DirectedGraphComplex(QQ, implementation='vector')
[7]: tetrahedron_oriented = dGC(tetrahedron); tetrahedron_oriented
[7]: (24)*DirectedGraph(4, [(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)]) +
(-8)*DirectedGraph(4, [(0, 1), (0, 2), (0, 3), (1, 3), (2, 1), (3, 2)]) +
(-24)*DirectedGraph(4, [(0, 2), (0, 3), (1, 0), (1, 2), (2, 3), (3, 1)]) +
(-8)*DirectedGraph(4, [(0, 2), (0, 3), (1, 0), (1, 3), (2, 1), (2, 3)])
[8]: tetrahedron_oriented_filtered = tetrahedron_oriented.filter(max_out_degree=2);
↳ tetrahedron_oriented_filtered
[8]: (-24)*DirectedGraph(4, [(0, 2), (0, 3), (1, 0), (1, 2), (2, 3), (3, 1)]) +
(-8)*DirectedGraph(4, [(0, 2), (0, 3), (1, 0), (1, 3), (2, 1), (2, 3)])
```

The differential polynomial ring with fibre variables  $\rho, a_0, a_1$  and base variables  $x, y, z, w$ :

```
[9]: from gcaops.algebra.differential_polynomial_ring import DifferentialPolynomialRing
[10]: D4 = DifferentialPolynomialRing(QQ, ('rho', 'a0', 'a1'), ('x', 'y', 'z', 'w'),
↳ max_differential_orders=[3+1, 4+1, 4+1]) #; D4
[13]: rho, a0, a1 = D4.fibre_variables()
[14]: a = [a0, a1]
```

The superfunction algebra:

```
[11]: from gcaops.algebra.superfunction_algebra import SuperfunctionAlgebra
[12]: S4.<xi0, xi1, xi2, xi3> = SuperfunctionAlgebra(D4, D4.base_variables()) #; S4
[15]: xi = S4.gens()
[16]: X = S4.even_coordinates()
```

The Nambu–Poisson structure:

```
[17]: P = rho*sum(sigma.sign()*diff(a[0], X[sigma(1)-1])*diff(a[1],
↳ X[sigma(2)-1])*xi[sigma(3)-1]*xi[sigma(4)-1] for sigma in Permutations(4))/2; P
[17]: (-rho*a0_y*a1_x + rho*a0_x*a1_y)*xi2*xi3 + (rho*a0_z*a1_x - rho*a0_x*a1_z)*xi1*xi3 +
(-rho*a0_w*a1_x + rho*a0_x*a1_w)*xi1*xi2 + (-rho*a0_z*a1_y + rho*a0_y*a1_z)*xi0*xi3 +
(rho*a0_w*a1_y - rho*a0_y*a1_w)*xi0*xi2 + (-rho*a0_w*a1_z + rho*a0_z*a1_w)*xi0*xi1
```

In fact the Poisson bracket is a “derived” bracket:  $P = -[[[\rho \partial_x \wedge \partial_y \wedge \partial_z \wedge \partial_w], a_0], a_1]$ :

```
[18]: P == -(rho*xi0*xi1*xi2*xi3).bracket(a0).bracket(a1)
[18]: True
[19]: P.bracket(P)
[19]: 0
```

```
[20]: # NOTE: filtered out graphs with out degree > 2, because we won't pass in any
      ↪ 3-vectors
      tetrahedron_operation4 = S4.graph_operation(tetrahedron_oriented_filtered) #;
      ↪ tetrahedron_operation4
```

The evolutions  $\dot{a}_0$  and  $\dot{a}_1$  of  $a_0$  and  $a_1$ :

```
[21]: %time adot = [4*tetrahedron_operation4(P,P,P,a[i]) for i in range(2)]
```

```
CPU times: user 57.3 s, sys: 71.3 ms, total: 57.4 s
Wall time: 57.4 s
```

```
[22]: #adot[0]
```

```
[23]: #len(adot[0].monomials())
```

```
[24]: #adot[1]
```

```
[25]: #len(adot[1].monomials())
```

The tetrahedral flow  $Q_{\text{tetra}}(P[\rho, a_0, a_1])$ :

```
[26]: %time Q_tetra4 = tetrahedron_operation4(P,P,P,P)
```

```
CPU times: user 1h 23min 10s, sys: 45min 6s, total: 2h 8min 17s
Wall time: 2h 8min 16s
```

```
[27]: len(str(Q_tetra4))
```

```
[27]: 318982343
```

We verify the equality  $Q_{\text{tetra}}(P[\rho, a_0, a_1]) = P[\dot{\rho}, a_0, a_1] + P[\rho, \dot{a}_0, a_1] + P[\rho, a_0, \dot{a}_1]$ :

```
[28]: P0 = rho*sum(sigma.sign()*diff(adot[0],X[sigma(1)-1])*diff(a[1],
      ↪ X[sigma(2)-1])*xi[sigma(3)-1]*xi[sigma(4)-1] for sigma in Permutations(4))/2
```

```
[29]: P1 = rho*sum(sigma.sign()*diff(a[0],X[sigma(1)-1])*diff(adot[1],
      ↪ X[sigma(2)-1])*xi[sigma(3)-1]*xi[sigma(4)-1] for sigma in Permutations(4))/2
```

```
[30]: Q_remainder = Q_tetra4 - P0 - P1
```

```
[31]: len(str(Q_remainder))
```

```
[31]: 62481479
```

```
[32]: P_withoutprefactor = sum(sigma.sign()*diff(a[0],X[sigma(1)-1])*diff(a[1],
      ↪ X[sigma(2)-1])*xi[sigma(3)-1]*xi[sigma(4)-1] for sigma in Permutations(4))/2;
      ↪ P_withoutprefactor
```

```
[32]: (-a0_w*a1_z + a0_z*a1_w)*xi0*xi1 + (a0_w*a1_y - a0_y*a1_w)*xi0*xi2 + (-a0_z*a1_y +
      a0_y*a1_z)*xi0*xi3 + (-a0_w*a1_x + a0_x*a1_w)*xi1*xi2 + (a0_z*a1_x -
      a0_x*a1_z)*xi1*xi3 + (-a0_y*a1_x + a0_x*a1_y)*xi2*xi3
```

```
[33]: Q_remainder01 = Q_remainder[0,1]
```

```
[34]: Q_remainder01 % P_withoutprefactor[0,1] == 0
```

```
[34]: True
```

```
[35]: rhodot = Q_remainder01 // P_withoutprefactor[0,1] #; rhodot
```

```
[40]: len(rhodot.monomials())
```

```
[40]: 90024
```

```
[37]: rhodot * P_withoutprefactor == Q_remainder
```

```
[37]: True
```

```
[38]: Q_tetra4 == rhodot*P_withoutprefactor + P0 + P1
```

```
[38]: True
```

```
[39]: from itertools import combinations
      for (i,j) in combinations(range(4),2):
          print((i,j),
                Q_remainder[i,j] % P_withoutprefactor[i,j] == 0,
                Q_remainder[i,j] // P_withoutprefactor[i,j] == rhodot)
```

```
(0, 1) True True
(0, 2) True True
(0, 3) True True
(1, 2) True True
(1, 3) True True
(2, 3) True True
```

### 7.3.1 Symmetry

Now the evolutions  $\dot{a}_0$  and  $\dot{a}_1$  are induced by the tetrahedral flow  $Q_{\gamma_3}(P^{\otimes 4})$  on the class  $P[\rho, a_0, a_1]$  of generalized Nambu–Poisson brackets on  $\mathbb{R}^4$  with generic inverse density  $\rho(x, y, z, w)$ . We collapse the known formulas of  $\dot{a}_0$  and  $\dot{a}_1$  by using three Civita symbols (with four indices each). This expression is joint work with D. Lipper (2021). We note that by setting  $\rho = 1$  one recovers the formulas which were found earlier in that special case.

```
[41]: a0dot_maybe = 0
      from itertools import product
      for sigma, tau, zeta in product(SymmetricGroup(4), repeat=3):
          s1,t1,u1,v1 = X[sigma(1)-1], X[sigma(2)-1], X[sigma(3)-1], X[sigma(4)-1]
          s2,t2,u2,v2 = X[tau(1)-1], X[tau(2)-1], X[tau(3)-1], X[tau(4)-1]
          s3,t3,u3,v3 = X[zeta(1)-1], X[zeta(2)-1], X[zeta(3)-1], X[zeta(4)-1]
          a0dot_maybe += sigma.sign() * tau.sign() * zeta.sign() * ( \
              3 * diff(a0,s1,u2,u3) * diff(a0,t1,t2) * diff(a1,s2) * diff(a1,s3,v1) * \
              ↪ diff(a1,t3,u1) * diff(a0,v2) * diff(a0,v3) * rho^3 + \
              +6 * diff(a0,s1,u2) * diff(a0,t1) * diff(a0,t2,v3) * diff(a0,u3,v1,v2) * \
              ↪ diff(a1,t3,u1) * diff(a1,s2) * diff(a1,s3) * rho^3 + \
```

```

+3 * diff(a0,v2) * diff(a0,t1,u2,v3) * diff(a1,v1) * diff(rho,s1) *
↪diff(a0,u1) * diff(a0,u3) * diff(a1,s2,t3) * diff(a1,s3,t2) * rho^2 + \
-6 * diff(a0,s1,v3) * diff(a1,s2,t1) * diff(rho,v1) * diff(a0,s3,u1,v2) *
↪diff(a0,u2) * diff(a0,u3) * diff(a1,t2) * diff(a1,t3) * rho^2 + \
-6 * diff(a0,s1,v2,v3) * diff(a0,t1) * diff(a0,u2) * diff(a0,u3,v1) *
↪diff(a1,s2) * diff(a1,s3,u1) * diff(a1,t3) * diff(rho,t2) * rho^2 + \
+6 * diff(a0,s1) * diff(a0,s2,v3) * diff(a0,s3,u1) * diff(a0,t1,t2,t3) *
↪diff(a1,v1) * diff(rho,v2) * diff(a1,u2) * diff(a1,u3) * rho^2 + \
-6 * diff(a0,s1) * diff(a0,t1,u2,u3) * diff(a1,u1,v2) * diff(a0,t2) *
↪diff(a0,t3) * diff(a1,s2) * diff(a1,s3) * diff(rho,v1) * diff(rho,v3) * rho + \
+6 * diff(a0,v2) * diff(a0,s1) * diff(a0,s2,s3,t1) * diff(a0,t2,t3) *
↪diff(rho,v1) * diff(rho,v3) * diff(a1,u1) * diff(a1,u2) * diff(a1,u3) * rho + \
-2 * diff(a0,s1) * diff(a0,t2) * diff(a0,t3,u1,u2) * diff(a0,u3) *
↪diff(a1,t1) * diff(a1,s2) * diff(a1,s3) * diff(rho,v1) * diff(rho,v2) * diff(rho,v3)
)
aldot_maybe == adot[0]/4

```

[41]: True

```

[42]: aldot_maybe = 0
from itertools import product
for sigma, tau, zeta in product(SymmetricGroup(4), repeat=3):
    s1,t1,u1,v1 = X[sigma(1)-1], X[sigma(2)-1], X[sigma(3)-1], X[sigma(4)-1]
    s2,t2,u2,v2 = X[tau(1)-1], X[tau(2)-1], X[tau(3)-1], X[tau(4)-1]
    s3,t3,u3,v3 = X[zeta(1)-1], X[zeta(2)-1], X[zeta(3)-1], X[zeta(4)-1]
    aldot_maybe += sigma.sign() * tau.sign() * zeta.sign() * (
+3 * diff(a0,s1) * diff(a1,t1,u2) * diff(a1,u1,u3,v2) * diff(a0,s2,t3) *
↪diff(a0,s3,t2) * diff(a1,v1) * diff(a1,v3) * rho^3 + \
-3 * diff(a0,s1,t2) * diff(a1,u1) * diff(a1,u2,u3,v1) * diff(a0,t1) *
↪diff(a0,t3) * diff(a1,s2,v3) * diff(a1,s3,v2) * rho^3 + \
-6 * diff(a0,u1) * diff(a1,t1,t2,v3) * diff(rho,t3) * diff(a0,u2,v1) *
↪diff(a0,u3,v2) * diff(a1,s1) * diff(a1,s2) * diff(a1,s3) * rho^2 + \
+6 * diff(a0,s1) * diff(a0,t1,t3) * diff(a0,u2) * diff(a1,v2) *
↪diff(a1,s2,v1,v3) * diff(a1,s3,t2) * diff(a1,u1) * diff(rho,u3) * rho^2 + \
+6 * diff(a0,t1,u2) * diff(a1,u1,v2) * diff(rho,u3) * diff(a1,s1,s2,s3) *
↪diff(a0,v1) * diff(a0,v3) * diff(a1,t2) * diff(a1,t3) * rho^2 + \
+3 * diff(a1,v1) * diff(a1,s1,s2,s3) * diff(rho,t1) * diff(a1,t2,v3) *
↪diff(a1,t3,v2) * diff(a0,u1) * diff(a0,u2) * diff(a0,u3) * rho^2 + \
-6 * diff(a0,t1,u2) * diff(a1,u1,u3,v2) * diff(a0,v1) * diff(a0,v3) *
↪diff(rho,t2) * diff(rho,t3) * diff(a1,s1) * diff(a1,s2) * diff(a1,s3) * rho + \
-6 * diff(a1,t1,u2,v3) * diff(a1,u1,u3) * diff(a1,v1) * diff(a1,v2) *
↪diff(rho,t2) * diff(rho,t3) * diff(a0,s1) * diff(a0,s2) * diff(a0,s3) * rho + \
+2 * diff(a1,s1) * diff(a1,s2,s3,t1) * diff(rho,v1) * diff(a1,v2) *
↪diff(a1,v3) * diff(rho,t2) * diff(rho,t3) * diff(a0,u1) * diff(a0,u2) * diff(a0,u3)
)
aldot_maybe == adot[1]/4

```

[42]: True

[43]: # TODO: rhodot

### 7.3.2 Differential polynomial (non)triviality

One could try to solve the equation  $Q_{\text{tetra}}(P[\rho, a_0, a_1]) = [[P, V[\rho, a_0, a_1]]]$  for a vector field  $V[\rho, a_0, a_1]$  as follows:

```
[ ]: D4V = DifferentialPolynomialRing(QQ, ('rho', 'a0', 'a1', 'V0', 'V1', 'V2', 'V3'),
↳ ('x', 'y', 'z', 'w'), max_differential_orders=[3+1,4+1,4+1,1,1,1,1]) #; D4V
[42]: V0, V1, V2, V3 = D4V.fibre_variables()[3:]
[ ]: S4V = SuperfunctionAlgebra(D4V, D4V.base_variables(), names='xi0,xi1,xi2,xi3') #; S4V
[44]: V = V0*S4V(xi0) + V1*S4V(xi1) + V2*S4V(xi2) + V3*S4V(xi3); V
[44]: (V0)*xi0 + (V1)*xi1 + (V2)*xi2 + (V3)*xi3
[46]: PbracketV = S4V(P).bracket(V)
[47]: from gcaops.algebra.differential_polynomial_solver import solve_homogeneous_diffpoly
[48]: set_verbose(1)
[ ]: %time sol = solve_homogeneous_diffpoly(S4V(Q_tetra4)[0,1], PbracketV[0,1],
↳ [V0,V1,V2,V3])
```

```
verbose 1 (9: differential_polynomial_solver.py, solve_homogeneous_diffpoly) target
degrees: (4, 4, 4, 0, 0, 0, 0)
verbose 1 (9: differential_polynomial_solver.py, solve_homogeneous_diffpoly) target
weights: (3, 3, 4, 4)
verbose 1 (9: differential_polynomial_solver.py, solve_homogeneous_diffpoly) ansatz
degrees: {V0: {(3, 3, 3, 0, 0, 0, 0)}, V3: {(3, 3, 3, 0, 0, 0, 0)}, V1: {(3, 3, 3, 0,
0, 0, 0)}, V2: {(3, 3, 3, 0, 0, 0, 0)}}
verbose 1 (9: differential_polynomial_solver.py, solve_homogeneous_diffpoly) ansatz
weights: {V0: {(2, 3, 3, 3)}, V3: {(3, 3, 3, 2)}, V1: {(3, 2, 3, 3)}, V2: {(3, 3, 2,
3)}}
```

So far, the (non)triviality of the flow  $Q_{\text{tetra}}(P[\rho, a_0, a_1])$  remains an open problem.

## 7.4 Five-wheel flow on rescaled Nambu–Poisson structures on $\mathbb{R}^3$

Define the five-wheel graph cocycle  $\gamma_5$ :

```
[2]: from gcaops.graph.undirected_graph import UndirectedGraph
from gcaops.graph.undirected_graph_complex import UndirectedGraphComplex
[3]: GC = UndirectedGraphComplex(QQ); GC
[3]: Undirected graph complex over Rational Field with Basis consisting of representatives
of isomorphism classes of undirected graphs with no automorphisms that induce an odd
permutation on edges
[4]: fivewheel_graph = UndirectedGraph(6,
↳ [(0,1),(1,2),(2,3),(3,4),(0,4),(0,5),(1,5),(2,5),(3,5),(4,5)])
[5]: roof_graph = UndirectedGraph(6,
↳ [(0,1),(1,2),(2,3),(0,3),(3,4),(0,4),(4,5),(2,5),(1,5),(0,2)])
```

```
[6]: fivewheel_cocycle = GC(fivewheel_graph) + (5/2)*GC(roof_graph)
[7]: from gcaops.graph.directed_graph_complex import DirectedGraphComplex
[8]: dGC = DirectedGraphComplex(QQ, implementation='vector')
[23]: fivewheel_cocycle_oriented = dGC(fivewheel_cocycle) #; fivewheel_cocycle_oriented
[24]: fivewheel_cocycle_oriented_filtered = fivewheel_cocycle_oriented.
      ↪ filter(max_out_degree=2) #; fivewheel_cocycle_oriented_filtered
[11]: len(fivewheel_cocycle_oriented_filtered)
[11]: 91
```

The differential polynomial ring with fibre variables  $\rho, a$  and base variables  $x, y, z$ :

```
[12]: from gcaops.algebra.differential_polynomial_ring import DifferentialPolynomialRing
[13]: D3 = DifferentialPolynomialRing(QQ, ('rho', 'a'), ('x', 'y', 'z'),
      ↪ max_differential_orders=[5,6]) #; D3
[16]: rho, a = D3.fibre_variables()
```

The superfunction algebra:

```
[14]: from gcaops.algebra.superfunction_algebra import SuperfunctionAlgebra
[15]: S3.<xi0,xi1,xi2> = SuperfunctionAlgebra(D3, D3.base_variables()) #; S3
[17]: xi = S3.gens()
[18]: X = S3.even_coordinates()
```

The Nambu-Poisson structure:

```
[19]: P = rho*sum(sigma.sign()*diff(a,X[sigma(1)-1])*xi[sigma(2)-1]*xi[sigma(3)-1] for
      ↪ sigma in Permutations(3))/2; P
[19]: (rho*a_z)*xi0*xi1 + (-rho*a_y)*xi0*xi2 + (rho*a_x)*xi1*xi2
[20]: P.bracket(P)
[20]: 0
```

```
[21]: # NOTE: filtered out graphs with out degree > 2, because we won't pass in any
      ↪ 3-vectors
      fivewheel_operation3 = S3.graph_operation(fivewheel_cocycle_oriented_filtered) #;
      ↪ fivewheel_operation3
```

The evolution  $\dot{a}$  of  $a$ :

```
[22]: %time adot = 6 * fivewheel_operation3(P,P,P,P,P,a)
```

CPU times: user 27min 12s, sys: 1.51 s, total: 27min 13s  
Wall time: 27min 13s

The five-wheel flow  $Q_{\gamma_5}(P[\rho, a])$ :

```
[46]: %time Q_fivewheel3 = fivewheel_operation3(P,P,P,P,P,P)
```

```
CPU times: user 14h 27min 1s, sys: 9.58 s, total: 14h 27min 10s
Wall time: 14h 27min 8s
```

```
[47]: len(str(Q_fivewheel3))
```

```
[47]: 63344567
```

We have the equality  $Q_{\gamma_5}(P[\rho, a]) = P[\dot{\rho}, a] + P[\rho, \dot{a}]$ :

```
[66]: P0 = rho*sum(sigma.sign()*diff(adot,X[sigma(1)-1])*xi[sigma(2)-1]*xi[sigma(3)-1] for
↳sigma in Permutations(3))/2
```

```
[67]: Q_remainder = Q_fivewheel3 - P0
```

```
[68]: len(str(Q_remainder))
```

```
[68]: 24690209
```

```
[52]: P_withoutprefactor = sum(sigma.
↳sign()*diff(a[0],X[sigma(1)-1])*xi[sigma(2)-1]*xi[sigma(3)-1] for sigma in
↳Permutations(3))/2; P_withoutprefactor
```

```
[52]: (a_z)*xi0*xi1 + (-a_y)*xi0*xi2 + (a_x)*xi1*xi2
```

```
[54]: Q_remainder[0,1] % P_withoutprefactor[0,1] == 0
```

```
[54]: True
```

```
[55]: rhodot = Q_remainder[0,1] // P_withoutprefactor[0,1]
```

```
[69]: len(str(rhodot))
```

```
[69]: 7863550
```

```
[70]: #with open('data/Q_5_3d_rank2_adot_rhodot.txt', 'w') as f:
#     f.write('adot = ' + str(adot) + '\n')
#     f.write('rhodot = ' + str(rhodot) + '\n')
```

```
[71]: rhodot * P_withoutprefactor == Q_remainder
```

```
[71]: True
```

```
[73]: Q_fivewheel3 == rhodot * P_withoutprefactor + P0
```

```
[73]: True
```

```
[60]: len(rhodot.monomials())
```

```
[60]: 146340
```



```
[77]: len(list(adot._monomial_coefficients.values())[0][0].monomials())
```

```
[77]: 79212
```

```
[ ]: from itertools import combinations
     for (i,j) in combinations(range(3),2):
         print((i,j),
               Q_remainder[i,j] % P_withoutprefactor[i,j] == 0,
               Q_remainder[i,j] // P_withoutprefactor[i,j] == rhodot)
```

```
[ ]: # TODO: differential polynomial (non)triviality
```



# Chapter 8

## Graph complex action on $R$ -matrix Poisson structures

The theory which we use in this chapter originates from the article [31] by Li and Parmentier. Those authors recall a method to obtain quadratic and introduce a method to construct cubic Poisson brackets associated with Lie brackets and  $R$ -matrices on associative algebras, such as  $\mathfrak{gl}_n(\mathbb{R})$ . Their formulae express Poisson structures in terms of a Lie bracket, an associative product, a nondegenerate symmetric bilinear form, and an  $R$ -matrix.

We shall evaluate the formulae at specific choices of the arguments: for the Lie algebras  $\mathfrak{gl}_n(\mathbb{R})$  with  $n = 2, 3$ , for the usual matrix product, for the nondegenerate symmetric bilinear form  $\langle A, B \rangle = \text{tr}(AB)$ , and for various  $R$ -matrices. A store of  $R$ -matrices is available from the Appendix within *loc. cit.*

(By the way, the formula by Li–Parmentier gives us a valid Poisson tensor also for  $\mathfrak{sl}_n(\mathbb{R})$ , which is not an algebra at all, as it is not closed under the usual product. We demonstrate this in what follows, by producing the Poisson bracket.)

**Definition.** The gradient  $\nabla_x F \in \mathfrak{g}$  of  $F \in C^\infty(\mathfrak{g})$  at  $x \in \mathfrak{g}$  is the unique element such that  $\langle \nabla_x F, y \rangle = (d_x F)(y)$ , where  $d_x F$  is the de Rham differential of  $F$  at  $x$ .

The gradient exists because the bilinear form is non-degenerate.

**Example 1.** For  $\mathfrak{g} = \mathfrak{gl}_2(\mathbb{R})$  and  $F : \begin{pmatrix} x_0 & x_1 \\ x_2 & x_3 \end{pmatrix} \mapsto x_0^2$  we have  $d_x F = 2x_0 dx_0$  and hence

$$(d_x F)(y) = (d_x F) \begin{pmatrix} y_0 & y_1 \\ y_2 & y_3 \end{pmatrix} = 2x_0 y_0.$$

The nondegenerate bilinear form is

$$\begin{aligned} \langle x, y \rangle &= \text{tr}(xy) = \text{tr} \left( \begin{pmatrix} x_0 & x_1 \\ x_2 & x_3 \end{pmatrix} \cdot \begin{pmatrix} y_0 & y_1 \\ y_2 & y_3 \end{pmatrix} \right) \\ &= \text{tr} \begin{pmatrix} x_0 y_0 + x_1 y_2 & x_0 y_1 + x_1 y_3 \\ x_2 y_0 + x_3 y_2 & x_2 y_1 + x_3 y_3 \end{pmatrix} \\ &= x_0 y_0 + x_1 y_2 + x_2 y_1 + x_3 y_3, \end{aligned}$$

and it follows that  $\nabla_x F = \begin{pmatrix} 2x_0 & 0 \\ 0 & 0 \end{pmatrix}$ .

**Example 2.** The formulas for  $R$ -matrix Poisson brackets require only the gradients of linear coordinate functions. For these we have a specialized implementation:

```
[1]: from gcaops.algebra.r_matrix_poisson import gradients_of_linear_coordinates
```

```
[2]: import itertools
gl2_basis = [matrix(2, lambda i,j: 1 if (i,j) == (a,b) else 0) for (a,b) in itertools.
↳product(range(2), repeat=2)]
gl2_basis
```

```
[2]: [
[1 0] [0 1] [0 0] [0 0]
[0 0], [0 0], [1 0], [0 1]
]
```

```
[3]: gradients_of_linear_coordinates(gl2_basis)
```

```
[3]: [
[1 0] [0 0] [0 1] [0 0]
[0 0], [1 0], [0 0], [0 1]
]
```

Indeed, we have e.g.  $d_x x_1 = dx_1$ ;  $dx_1(y) = y_1$  and hence  $\nabla_x x_1 = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}$ , as can be seen from the formula for  $\langle x, y \rangle$  above.

We now import the constructor of  $R$ -matrix Poisson structures. For the quadratic and cubic Poisson structures, it is required that the bilinear form is moreover “associative” in the sense that  $\langle X \cdot Y, Z \rangle = \langle X, Y \cdot Z \rangle$  for all  $X, Y, Z$ , and it is assumed that the  $R$ -matrix satisfies the modified Yang–Baxter equation. For the quadratic Poisson structure, it is further required that the skew part  $R_-$  of  $R$  satisfies the modified Yang-Baxter equation with the same constant as  $R$  itself.

```
[4]: from gcaops.algebra.r_matrix_poisson import r_matrix_poisson_bivector
```

To prepare for using this constructor, we define some bases of Lie algebras:

```
[5]: def gl_basis(field, matrix_dimension):
import itertools
return [matrix(field, matrix_dimension, lambda i,j: 1 if (i,j) == (a,b) else 0)
↳for (a,b) in itertools.product(range(matrix_dimension), repeat=2)]
```

```
[6]: gl_basis(QQ, 2)
```

```
[6]: [
[1 0] [0 1] [0 0] [0 0]
[0 0], [0 0], [1 0], [0 1]
]
```

```
[7]: def so_basis(field, matrix_dimension):
```

```

    return [matrix(field, matrix_dimension, lambda i,j: 1 if (i,j) == (a,b) else -1,
↳if (i,j) == (b,a) else 0) for (a,b) in itertools.
↳combinations(range(matrix_dimension),2)]

```

```
[8]: so_basis(QQ, 3)
```

```
[8]: [
[ 0  1  0] [ 0  0  1] [ 0  0  0]
[-1  0  0] [ 0  0  0] [ 0  0  1]
[ 0  0  0], [-1  0  0], [ 0 -1  0]
]
```

```
[9]: def sl_basis(field, matrix_dimension):
    return list(m.matrix() for m in lie_algebras.sl(QQ, matrix_dimension,
↳representation='matrix').basis())
```

```
[10]: sl_basis(QQ, 3)
```

```
[10]: [
[ 1  0  0] [0 1 0] [0 0 1] [0 0 0] [ 0  0  0] [0 0 0] [0 0 0]
[ 0  0  0] [0 0 0] [0 0 0] [1 0 0] [ 0  1  0] [0 0 1] [0 0 0]
[ 0  0 -1], [0 0 0], [0 0 0], [0 0 0], [ 0  0 -1], [0 0 0], [1 0 0],

[0 0 0]
[0 0 0]
[0 1 0]
]
```

Moreover, we define some typical  $R$ -matrices:

```
[11]: R_id = lambda X: X
# differences of projections
R_strict = lambda X: matrix(X.nrows(), lambda i,j: X[i,j] if i < j else -X[i,j] if i
↳> j else 0)
R_weak = lambda X: matrix(X.nrows(), lambda i,j: X[i,j] if i <= j else -X[i,j]) # ???
```

We also define a shorthand function for the tetrahedral flow, which will be used in the next sections.

```
[12]: from gcaops.graph.undirected_graph import UndirectedGraph
from gcaops.graph.undirected_graph_complex import UndirectedGraphComplex
GC = UndirectedGraphComplex(QQ)
tetrahedron_graph = UndirectedGraph(4, [(0,1),(0,2),(0,3),(1,2),(1,3),(2,3)])
tetrahedron = GC(tetrahedron_graph)

def tetrahedral_flow(P):
    S = P.parent()
    tetrahedron_operation = S.graph_operation(tetrahedron)
    return tetrahedron_operation(P,P,P,P)
```

In each case in the upcoming sections, we will calculate the tetrahedral flow and check its (non-)triviality in the respective Poisson cohomology.

## 8.1 The Lie algebra of $2 \times 2$ matrices

$\mathfrak{gl}_2(\mathbb{R})$  with  $R = \text{id}$ , quadratic Poisson structure

[13]: `P = r_matrix_poisson_bivector(gl_basis(QQ, 2), 2, R_matrix=R_id); P`

[13]:  $(-x_0*x_1 - x_1*x_3)*xi_0*xi_1 + (x_0*x_2 + x_2*x_3)*xi_0*xi_2 + (-x_0^2 + x_3^2)*xi_1*xi_2 + (-x_0*x_1 - x_1*x_3)*xi_1*xi_3 + (x_0*x_2 + x_2*x_3)*xi_2*xi_3$

[14]: `P.bracket(P)`

[14]: 0

[15]: `Q = tetrahedral_flow(P); Q`

[15]: 0

$\mathfrak{gl}_2(\mathbb{R})$  with  $R = \text{id}$ , cubic Poisson structure

[16]: `P = r_matrix_poisson_bivector(gl_basis(QQ, 2), 3, R_matrix=R_id); P`

[16]:  $(x_1^2*x_2 - x_0*x_1*x_3)*xi_0*xi_1 + (-x_1*x_2^2 + x_0*x_2*x_3)*xi_0*xi_2 + (x_0*x_1*x_2 - x_0^2*x_3 - x_1*x_2*x_3 + x_0*x_3^2)*xi_1*xi_2 + (x_1^2*x_2 - x_0*x_1*x_3)*xi_1*xi_3 + (-x_1*x_2^2 + x_0*x_2*x_3)*xi_2*xi_3$

[17]: `P.bracket(P)`

[17]: 0

[18]: `Q = tetrahedral_flow(P); Q`

[18]: 0

$\mathfrak{gl}_2(\mathbb{R})$  with  $R = \begin{pmatrix} x_0 & x_1 \\ x_2 & x_3 \end{pmatrix} \mapsto \begin{pmatrix} 0 & x_1 \\ -x_2 & 0 \end{pmatrix}$ , quadratic Poisson structure

[19]: `P = r_matrix_poisson_bivector(gl_basis(QQ, 2), 2, R_matrix=R_strict); P`

[19]:  $(x_0*x_1)*xi_0*xi_1 + (x_0*x_2)*xi_0*xi_2 + (2*x_1*x_2)*xi_0*xi_3 + (x_1*x_3)*xi_1*xi_3 + (x_2*x_3)*xi_2*xi_3$

[20]: `P.bracket(P)`

[20]: 0

[21]: `Q = tetrahedral_flow(P); Q`

[21]: 0

$\mathfrak{gl}_2(\mathbb{R})$  with  $R = \begin{pmatrix} x_0 & x_1 \\ x_2 & x_3 \end{pmatrix} \mapsto \begin{pmatrix} 0 & x_1 \\ -x_2 & 0 \end{pmatrix}$ , cubic Poisson structure

[22]: `P = r_matrix_poisson_bivector(gl_basis(QQ, 2), 3, R_matrix=R_strict); P`

[22]:  $(1/2*x_0^2*x_1 + 1/2*x_1^2*x_2)*xi_0*xi_1 + (1/2*x_0^2*x_2 + 1/2*x_1*x_2^2)*xi_0*xi_2 + (x_0*x_1*x_2 + x_1*x_2*x_3)*xi_0*xi_3 + (1/2*x_1^2*x_2 + 1/2*x_1*x_3^2)*xi_1*xi_3 + (1/2*x_1*x_2^2 + 1/2*x_2*x_3^2)*xi_2*xi_3$

```
[23]: P.bracket(P)
```

```
[23]: 0
```

```
[24]: Q = tetrahedral_flow(P); Q
```

```
[24]: (48*x0^3*x1^2*x2 - 48*x0*x1^3*x2^2 + 96*x0^2*x1^2*x2*x3 + 60*x1^3*x2^2*x3 +
72*x1^2*x2*x3^3 + 12*x1*x3^5)*xi1*xi3 + (48*x0^3*x1*x2^2 - 48*x0*x1^2*x2^3 +
96*x0^2*x1*x2^2*x3 + 60*x1^2*x2^3*x3 + 72*x1*x2^2*x3^3 + 12*x2*x3^5)*xi2*xi3 +
(-12*x0^5*x1 - 72*x0^3*x1^2*x2 - 60*x0*x1^3*x2^2 + 48*x1^3*x2^2*x3 -
96*x0*x1^2*x2*x3^2 - 48*x1^2*x2*x3^3)*xi0*xi1 + (-12*x0^5*x2 - 72*x0^3*x1*x2^2 -
60*x0*x1^2*x2^3 + 48*x1^2*x2^3*x3 - 96*x0*x1*x2^2*x3^2 - 48*x1*x2^2*x3^3)*xi0*xi2 +
(-84*x0^4*x1*x2 - 120*x0^2*x1^2*x2^2 - 144*x0^3*x1*x2*x3 + 120*x1^2*x2^2*x3^2 +
144*x0*x1*x2*x3^3 + 84*x1*x2*x3^4)*xi0*xi3
```

```
[25]: from gcaops.algebra.homogeneous_polynomial_poisson_complex import PoissonComplex
```

```
[26]: PC = PoissonComplex(P)
```

```
[27]: PC(Q).is_coboundary(certificate=True)
```

```
[27]: (True,
Poisson cochain (-12*x0^4 + 156*x0*x1*x2*x3 + 48*x1*x2*x3^2)*xi0 + (78*x0^3*x1 +
60*x0*x1^2*x2 - 30*x0^2*x1*x3 + 18*x1^2*x2*x3 + 30*x0*x1*x3^2)*xi1 + (78*x0^3*x2 +
60*x0*x1*x2^2 - 30*x0^2*x2*x3 + 18*x1*x2^2*x3 + 30*x0*x2*x3^2)*xi2 + (108*x0^2*x1*x2 +
12*x3^4)*xi3)
```

$\mathfrak{gl}_2(\mathbb{R})$  with  $R = \begin{pmatrix} x_0 & x_1 \\ x_2 & x_3 \end{pmatrix} \mapsto \begin{pmatrix} x_0 & x_1 \\ -x_2 & x_3 \end{pmatrix}$ , quadratic Poisson structure

```
[28]: P = r_matrix_poisson_bivector(gl_basis(QQ, 2), 2, R_matrix=R_weak); P
```

```
[28]: (2*x0*x2)*xi0*xi2 + (2*x1*x2)*xi0*xi3 + (2*x2*x3)*xi2*xi3
```

```
[29]: P.bracket(P)
```

```
[29]: 0
```

```
[30]: Q = tetrahedral_flow(P); Q
```

```
[30]: 0
```

$\mathfrak{gl}_2(\mathbb{R})$  with  $R = \begin{pmatrix} x_0 & x_1 \\ x_2 & x_3 \end{pmatrix} \mapsto \begin{pmatrix} x_0 & x_1 \\ -x_2 & x_3 \end{pmatrix}$ , cubic Poisson structure

```
[31]: P = r_matrix_poisson_bivector(gl_basis(QQ, 2), 3, R_matrix=R_weak); P
```

```
[31]: (x1^2*x2)*xi0*xi1 + (x0^2*x2)*xi0*xi2 + (x0*x1*x2 + x1*x2*x3)*xi0*xi3 + (x0*x1*x2 -
x1*x2*x3)*xi1*xi2 + (x1^2*x2)*xi1*xi3 + (x2*x3^2)*xi2*xi3
```

```
[32]: P.bracket(P)
```

```
[32]: 0
```

```
[33]: Q = tetrahedral_flow(P); Q
```

```
[33]: (384*x0*x1^2*x2^3 + 384*x0^2*x1*x2^2*x3 - 96*x0*x1*x2^2*x3^2 + 96*x1*x2^2*x3^3 +
96*x2*x3^5)*xi2*xi3 + (-96*x0^4*x1*x2 - 96*x0^2*x1^2*x2^2 - 384*x1^3*x2^3 +
96*x0^3*x1*x2*x3 - 192*x0*x1^2*x2^2*x3 - 96*x0^2*x1*x2*x3^2 - 96*x1^2*x2^2*x3^2 +
96*x0*x1*x2*x3^3 - 96*x1*x2*x3^4)*xi1*xi2 + (-96*x0^5*x2 - 96*x0^3*x1*x2^2 +
96*x0^2*x1*x2^2*x3 - 384*x1^2*x2^3*x3 - 384*x0*x1*x2^2*x3^2)*xi0*xi2 + (-96*x0^4*x1*x2
- 672*x0^2*x1^2*x2^2 - 384*x0^3*x1*x2*x3 + 672*x1^2*x2^2*x3^2 + 384*x0*x1*x2*x3^3 +
96*x1*x2*x3^4)*xi0*xi3 + (-96*x0^3*x1^2*x2 - 672*x0*x1^3*x2^2 - 288*x0^2*x1^2*x2*x3 +
672*x1^3*x2^2*x3 + 288*x0*x1^2*x2*x3^2 + 96*x1^2*x2*x3^3)*xi0*xi1 + (-96*x0^3*x1^2*x2
- 672*x0*x1^3*x2^2 - 288*x0^2*x1^2*x2*x3 + 672*x1^3*x2^2*x3 + 288*x0*x1^2*x2*x3^2 +
96*x1^2*x2*x3^3)*xi1*xi3
```

```
[34]: PC = PoissonComplex(P)
```

```
[35]: PC(Q).is_coboundary(certificate=True)
```

```
[35]: (True,
Poisson cochain (-48*x0^4 + 480*x0*x1*x2*x3)*xi0 + (-48*x0^3*x1 + 480*x0*x1^2*x2 +
48*x0^2*x1*x3 - 48*x0*x1*x3^2 + 48*x1*x3^3)*xi1 + (480*x0^3*x2 + 192*x0*x1*x2^2 -
288*x0^2*x2*x3 - 192*x1*x2^2*x3 + 288*x0*x2*x3^2)*xi2 + (480*x0^2*x1*x2 +
48*x3^4)*xi3)
```

## 8.2 The Lie algebra of $3 \times 3$ matrices

$\mathfrak{gl}_3(\mathbb{R})$  with  $R = \text{id}$ , quadratic Poisson structure

Skew part  $R_-$  of  $R$  is zero, does not satisfy YBE.

$\mathfrak{gl}_3(\mathbb{R})$  with  $R = \text{id}$ , cubic Poisson structure

```
[36]: P = r_matrix_poisson_bivector(gl_basis(QQ, 3), 3, R_matrix=R_id); P
```

```
[36]: (x1^2*x3 - x0*x1*x4 + x1*x2*x6 - x0*x2*x7)*xi0*xi1 + (x1*x2*x3 - x0*x1*x5 + x2^2*x6 -
x0*x2*x8)*xi0*xi2 + (-x1*x3^2 + x0*x3*x4 - x2*x3*x6 + x0*x5*x6)*xi0*xi3 + (x1*x5*x6 -
x2*x3*x7)*xi0*xi4 + (x2*x3*x4 - x1*x3*x5 + x2*x5*x6 - x2*x3*x8)*xi0*xi5 + (-x1*x3*x6 -
x2*x6^2 + x0*x3*x7 + x0*x6*x8)*xi0*xi6 + (-x1*x4*x6 + x1*x3*x7 - x2*x6*x7 +
x1*x6*x8)*xi0*xi7 + (-x1*x5*x6 + x2*x3*x7)*xi0*xi8 + (x1*x2*x4 - x1^2*x5 + x2^2*x7 -
x1*x2*x8)*xi1*xi2 + (x0*x1*x3 - x0^2*x4 - x1*x3*x4 + x0*x4^2 - x2*x4*x6 +
x0*x5*x7)*xi1*xi3 + (x1^2*x3 - x0*x1*x4 - x2*x4*x7 + x1*x5*x7)*xi1*xi4 + (x1*x2*x3 -
x0*x2*x4 + x2*x4^2 - x1*x4*x5 + x2*x5*x7 - x2*x4*x8)*xi1*xi5 + (x0*x1*x6 - x0^2*x7 -
x1*x3*x7 + x0*x4*x7 - x2*x6*x7 + x0*x7*x8)*xi1*xi6 + (x1^2*x6 - x0*x1*x7 - x2*x7^2 +
x1*x7*x8)*xi1*xi7 + (x1*x2*x6 - x0*x2*x7 + x2*x4*x7 - x1*x5*x7)*xi1*xi8 + (x0*x2*x3 -
x0^2*x5 - x1*x3*x5 + x0*x4*x5 - x2*x5*x6 + x0*x5*x8)*xi2*xi3 + (x1*x2*x3 - x0*x1*x5 -
x2*x5*x7 + x1*x5*x8)*xi2*xi4 + (x2^2*x3 - x0*x2*x5 + x2*x4*x5 - x1*x5^2)*xi2*xi5 +
(x0*x2*x6 + x0*x5*x7 - x0^2*x8 - x1*x3*x8 - x2*x6*x8 + x0*x8^2)*xi2*xi6 + (x1*x2*x6 +
x1*x5*x7 - x0*x1*x8 - x1*x4*x8 - x2*x7*x8 + x1*x8^2)*xi2*xi7 + (x2^2*x6 + x2*x5*x7 -
x0*x2*x8 - x1*x5*x8)*xi2*xi8 + (-x1*x3^2 + x0*x3*x4 + x4*x5*x6 - x3*x5*x7)*xi3*xi4 +
(-x2*x3^2 + x0*x3*x5 + x5^2*x6 - x3*x5*x8)*xi3*xi5 + (-x3*x4*x6 - x5*x6^2 + x3^2*x7 +
x3*x6*x8)*xi3*xi6 + (-x1*x3*x6 + x0*x4*x6 - x4^2*x6 + x3*x4*x7 - x5*x6*x7 +
x4*x6*x8)*xi3*xi7 + (-x2*x3*x6 + x0*x5*x6 - x4*x5*x6 + x3*x5*x7)*xi3*xi8 + (-x2*x3*x4
+ x1*x3*x5 + x5^2*x7 - x4*x5*x8)*xi4*xi5 + (x1*x3*x6 - x0*x3*x7 - x5*x6*x7 +
x3*x7*x8)*xi4*xi6 + (x1*x4*x6 - x1*x3*x7 - x5*x7^2 + x4*x7*x8)*xi4*xi7 + (x1*x5*x6 -
x2*x3*x7)*xi4*xi8 + (x2*x3*x6 + x3*x5*x7 - x0*x3*x8 - x3*x4*x8 - x5*x6*x8 +
x3*x8^2)*xi5*xi6 + (x2*x4*x6 + x4*x5*x7 - x1*x3*x8 - x4^2*x8 - x5*x7*x8 +
```



$x_4*x_8^2)*x_{i5}*x_{i7} + (x_2*x_5*x_6 + x_5^2*x_7 - x_2*x_3*x_8 - x_4*x_5*x_8)*x_{i5}*x_{i8} + (-x_1*x_6^2 + x_0*x_6*x_7 - x_4*x_6*x_7 + x_3*x_7^2)*x_{i6}*x_{i7} + (-x_2*x_6^2 - x_5*x_6*x_7 + x_0*x_6*x_8 + x_3*x_7*x_8)*x_{i6}*x_{i8} + (-x_2*x_6*x_7 - x_5*x_7^2 + x_1*x_6*x_8 + x_4*x_7*x_8)*x_{i7}*x_{i8}$

```
[37]: P.bracket(P)
```

```
[37]: 0
```

```
[38]: #time Q = tetrahedral_flow(P)
```

```
[39]: #PC = PoissonComplex(P)
```

```
[40]: #time PC(Q).is_coboundary(certificate=True)
```

$\mathfrak{gl}_3(\mathbb{R})$  with  $R = \begin{pmatrix} x_0 & x_1 & x_2 \\ x_3 & x_4 & x_5 \\ x_6 & x_7 & x_8 \end{pmatrix} \mapsto \begin{pmatrix} 0 & x_1 & x_2 \\ -x_3 & 0 & x_5 \\ -x_6 & -x_7 & 0 \end{pmatrix}$ , quadratic Poisson structure

```
[41]: P = r_matrix_poisson_bivector(gl_basis(QQ, 3), 2, R_matrix=R_strict); P
```

```
[41]: (x0*x1)*xi0*x_i1 + (x0*x2)*xi0*x_i2 + (x0*x3)*xi0*x_i3 + (2*x1*x3)*xi0*x_i4 +
(2*x2*x3)*xi0*x_i5 + (x0*x6)*xi0*x_i6 + (2*x1*x6)*xi0*x_i7 + (2*x2*x6)*xi0*x_i8 +
(x1*x2)*x_i1*x_i2 + (x1*x4)*x_i1*x_i4 + (2*x2*x4)*x_i1*x_i5 + (x1*x7)*x_i1*x_i7 +
(2*x2*x7)*x_i1*x_i8 + (x2*x5)*x_i2*x_i5 + (x2*x8)*x_i2*x_i8 + (x3*x4)*x_i3*x_i4 +
(x3*x5)*x_i3*x_i5 + (x3*x6)*x_i3*x_i6 + (2*x4*x6)*x_i3*x_i7 + (2*x5*x6)*x_i3*x_i8 +
(x4*x5)*x_i4*x_i5 + (x4*x7)*x_i4*x_i7 + (2*x5*x7)*x_i4*x_i8 + (x5*x8)*x_i5*x_i8 +
(x6*x7)*x_i6*x_i7 + (x6*x8)*x_i6*x_i8 + (x7*x8)*x_i7*x_i8
```

```
[42]: P.bracket(P)
```

```
[42]: 0
```

```
[43]: Q = tetrahedral_flow(P); Q
```

```
[43]: 0
```

$\mathfrak{gl}_3(\mathbb{R})$  with  $R = \begin{pmatrix} x_0 & x_1 & x_2 \\ x_3 & x_4 & x_5 \\ x_6 & x_7 & x_8 \end{pmatrix} \mapsto \begin{pmatrix} 0 & x_1 & x_2 \\ -x_3 & 0 & x_5 \\ -x_6 & -x_7 & 0 \end{pmatrix}$ , cubic Poisson structure

```
[44]: P = r_matrix_poisson_bivector(gl_basis(QQ, 3), 3, R_matrix=R_strict); P
```

```
[44]: (1/2*x0^2*x1 + 1/2*x1^2*x3)*xi0*x_i1 + (1/2*x0^2*x2 + x1*x2*x3 + 1/2*x2^2*x6)*xi0*x_i2 +
(1/2*x0^2*x3 + 1/2*x1*x3^2)*xi0*x_i3 + (x0*x1*x3 + x1*x3*x4)*xi0*x_i4 + (x0*x2*x3 +
x2*x3*x4 + 1/2*x1*x3*x5 + 1/2*x2*x5*x6)*xi0*x_i5 + (1/2*x0^2*x6 + x1*x3*x6 +
1/2*x2*x6^2)*xi0*x_i6 + (x0*x1*x6 + x1*x4*x6 + 1/2*x1*x3*x7 + 1/2*x2*x6*x7)*xi0*x_i7 +
(x0*x2*x6 + x1*x5*x6 + x2*x3*x7 + x2*x6*x8)*xi0*x_i8 + (x1*x2*x4 - 1/2*x1^2*x5 +
1/2*x2^2*x7)*x_i1*x_i2 + (1/2*x1^2*x3 + 1/2*x1*x4^2)*x_i1*x_i4 + (1/2*x1*x2*x3 + x2*x4^2 +
1/2*x2^2*x5*x7)*x_i1*x_i5 + (1/2*x1*x3*x7 + 1/2*x2*x6*x7)*x_i1*x_i6 + (1/2*x1^2*x6 +
x1*x4*x7 + 1/2*x2*x7^2)*x_i1*x_i7 + (1/2*x1*x2*x6 + x2*x4*x7 + 1/2*x1*x5*x7 +
x2*x7*x8)*x_i1*x_i8 + (-1/2*x1*x3*x5 - 1/2*x2^2*x5*x6)*x_i2*x_i3 + (1/2*x1*x2*x3 -
1/2*x2^2*x5*x7)*x_i2*x_i4 + (1/2*x2^2*x3 + x2*x4*x5 - 1/2*x1*x5^2)*x_i2*x_i5 + (1/2*x1*x2*x6 +
1/2*x1*x5*x7)*x_i2*x_i7 + (1/2*x2^2*x6 + x2*x5*x7 + 1/2*x2*x8^2)*x_i2*x_i8 +
(1/2*x1*x3^2 + 1/2*x3*x4^2)*x_i3*x_i4 + (1/2*x2*x3^2 + x3*x4*x5 + 1/2*x5^2*x6)*x_i3*x_i5 +
```

$(x_3*x_4*x_6 + 1/2*x_5*x_6^2 - 1/2*x_3^2*x_7)*xi_3*xi_6 + (1/2*x_1*x_3*x_6 + x_4^2*x_6 + 1/2*x_5*x_6*x_7)*xi_3*xi_7 + (1/2*x_2*x_3*x_6 + x_4*x_5*x_6 + 1/2*x_3*x_5*x_7 + x_5*x_6*x_8)*xi_3*xi_8 + (1/2*x_4^2*x_5 + 1/2*x_5^2*x_7)*xi_4*xi_5 + (-1/2*x_1*x_3*x_6 + 1/2*x_5*x_6*x_7)*xi_4*xi_6 + (1/2*x_4^2*x_7 + 1/2*x_5*x_7^2)*xi_4*xi_7 + (x_4*x_5*x_7 + x_5*x_7*x_8)*xi_4*xi_8 + (-1/2*x_2*x_3*x_6 - 1/2*x_3*x_5*x_7)*xi_5*xi_6 + (1/2*x_5^2*x_7 + 1/2*x_5*x_8^2)*xi_5*xi_8 + (1/2*x_1*x_6^2 + x_4*x_6*x_7 - 1/2*x_3*x_7^2)*xi_6*xi_7 + (1/2*x_2*x_6^2 + x_5*x_6*x_7 + 1/2*x_6*x_8^2)*xi_6*xi_8 + (1/2*x_5*x_7^2 + 1/2*x_7*x_8^2)*xi_7*xi_8$

[45]: P.bracket(P)

[45]: 0

[46]: `time Q = tetrahedral_flow(P); Q`

[47]: `PC = PoissonComplex(P)`

[48]: `PC(Q).is_coboundary(certificate=True)`

$\mathfrak{gl}_3(\mathbb{R})$  with  $R = \begin{pmatrix} x_0 & x_1 & x_2 \\ x_3 & x_4 & x_5 \\ x_6 & x_7 & x_8 \end{pmatrix} \mapsto \begin{pmatrix} x_0 & x_1 & x_2 \\ -x_3 & x_4 & x_5 \\ -x_6 & -x_7 & x_8 \end{pmatrix}$ , quadratic Poisson structure

[49]: `P = r_matrix_poisson_bivector(gl_basis(QQ, 3), 2, R_matrix=R_weak); P`

[49]:  $(2*x_0*x_3)*xi_0*xi_3 + (2*x_1*x_3)*xi_0*xi_4 + (2*x_2*x_3)*xi_0*xi_5 + (2*x_0*x_6)*xi_0*xi_6 + (2*x_1*x_6)*xi_0*xi_7 + (2*x_2*x_6)*xi_0*xi_8 + (x_1*x_2)*xi_1*xi_2 + (2*x_2*x_4 - x_1*x_5)*xi_1*xi_5 + (x_1*x_6)*xi_1*xi_6 + (x_1*x_7)*xi_1*xi_7 + (2*x_2*x_7)*xi_1*xi_8 + (x_2*x_3)*xi_2*xi_3 + (x_2*x_5)*xi_2*xi_5 + (-x_2*x_7)*xi_2*xi_7 + (2*x_3*x_4)*xi_3*xi_4 + (x_3*x_5)*xi_3*xi_5 + (x_3*x_6)*xi_3*xi_6 + (2*x_4*x_6 + x_3*x_7)*xi_3*xi_7 + (2*x_5*x_6)*xi_3*xi_8 + (2*x_4*x_7)*xi_4*xi_7 + (2*x_5*x_7)*xi_4*xi_8 + (-x_5*x_6)*xi_5*xi_6 + (x_6*x_7)*xi_6*xi_7 + (2*x_6*x_8)*xi_6*xi_8 + (2*x_7*x_8)*xi_7*xi_8$

[50]: P.bracket(P)

[50]: 0

[51]: `time Q = tetrahedral_flow(P); Q`

CPU times: user 15.7 s, sys: 51.9 ms, total: 15.8 s  
Wall time: 15.8 s

[51]:  $(-384*x_2*x_3)*xi_0*xi_5 + (-384*x_1*x_3)*xi_0*xi_4 + (-384*x_1*x_6)*xi_0*xi_7 + (384*x_2*x_7)*xi_1*xi_8 + (384*x_5*x_7)*xi_4*xi_8 + (384*x_5*x_6)*xi_3*xi_8$

[52]: `PC = PoissonComplex(P)`

[53]: `PC(Q).is_coboundary(certificate=True)`

[53]: (True, Poisson cochain  $(192*x_1)*xi_1 + (192*x_3)*xi_3 + (192*x_4)*xi_4$ )

$\mathfrak{gl}_3(\mathbb{R})$  with  $R = \begin{pmatrix} x_0 & x_1 & x_2 \\ x_3 & x_4 & x_5 \\ x_6 & x_7 & x_8 \end{pmatrix} \mapsto \begin{pmatrix} x_0 & x_1 & x_2 \\ -x_3 & x_4 & x_5 \\ -x_6 & -x_7 & x_8 \end{pmatrix}$ , cubic Poisson structure

```
[54]: P = r_matrix_poisson_bivector(gl_basis(QQ, 3), 3, R_matrix=R_weak); P
```

```
[54]: (x1^2*x3)*xi0*x11 + (x1*x2*x3 + x2^2*x6)*xi0*x12 + (x0^2*x3)*xi0*x13 + (x0*x1*x3 +
x1*x3*x4)*xi0*x14 + (x0*x2*x3 + x2*x3*x4 + x2*x5*x6)*xi0*x15 + (x0^2*x6 +
x1*x3*x6)*xi0*x16 + (x0*x1*x6 + x1*x4*x6 + x1*x3*x7)*xi0*x17 + (x0*x2*x6 + x1*x5*x6 +
x2*x3*x7 + x2*x6*x8)*xi0*x18 + (x1*x2*x4 - x1^2*x5 + x2^2*x7)*xi1*x12 + (x0*x1*x3 -
x1*x3*x4)*xi1*x13 + (x1^2*x3)*xi1*x14 + (x1*x2*x3 + x2*x4^2 - x1*x4*x5 +
x2*x5*x7)*xi1*x15 + (x0*x1*x6)*xi1*x16 + (x1^2*x6 + x1*x4*x7)*xi1*x17 + (x1*x2*x6 +
x2*x4*x7 + x2*x7*x8)*xi1*x18 + (x0*x2*x3 - x1*x3*x5 - x2*x5*x6)*xi2*x13 + (x1*x2*x3 -
x2*x5*x7)*xi2*x14 + (x2^2*x3 + x2*x4*x5 - x1*x5^2)*xi2*x15 + (x0*x2*x6 -
x2*x6*x8)*xi2*x16 + (x1*x2*x6 + x1*x5*x7 - x2*x7*x8)*xi2*x17 + (x2^2*x6 +
x2*x5*x7)*xi2*x18 + (x3*x4^2)*xi3*x14 + (x3*x4*x5 + x5^2*x6)*xi3*x15 +
(x3*x4*x6)*xi3*x16 + (x4^2*x6 + x3*x4*x7)*xi3*x17 + (x4*x5*x6 + x3*x5*x7 +
x5*x6*x8)*xi3*x18 + (x5^2*x7)*xi4*x15 + (x4^2*x7)*xi4*x17 + (x4*x5*x7 +
x5*x7*x8)*xi4*x18 + (-x5*x6*x8)*xi5*x16 + (x4*x5*x7 - x5*x7*x8)*xi5*x17 +
(x5^2*x7)*xi5*x18 + (x4*x6*x7)*xi6*x17 + (x5*x6*x7 + x6*x8^2)*xi6*x18 +
(x7*x8^2)*xi7*x18
```

```
[55]: P.bracket(P)
```

```
[55]: 0
```

```
[56]: #%time Q = tetrahedral_flow(P); Q
```

### 8.3 The Lie algebra of traceless $2 \times 2$ matrices

**N.B.** Not an associative algebra, nevertheless the construction can work:

$\mathfrak{sl}_2(\mathbb{R})$  with  $R = \begin{pmatrix} x_0 & x_1 \\ x_2 & x_3 \end{pmatrix} \mapsto \begin{pmatrix} 0 & x_1 \\ -x_2 & 0 \end{pmatrix}$ , quadratic Poisson structure  $\equiv 0$

```
[57]: P = r_matrix_poisson_bivector(sl_basis(QQ, 2), 2, R_matrix=R_strict); P
```

```
[57]: 0
```

$\mathfrak{sl}_2(\mathbb{R})$  with  $R = \begin{pmatrix} x_0 & x_1 \\ x_2 & x_3 \end{pmatrix} \mapsto \begin{pmatrix} 0 & x_1 \\ -x_2 & 0 \end{pmatrix}$ , cubic Poisson structure

```
[58]: P = r_matrix_poisson_bivector(sl_basis(QQ, 2), 3, R_matrix=R_strict); P
```

```
[58]: (-1/2*x0^2*x1 - 1/2*x0*x2^2)*xi0*x12 + (-1/2*x0*x1^2 - 1/2*x1*x2^2)*xi1*x12
```

```
[59]: P.bracket(P)
```

```
[59]: 0
```

```
[60]: Q = tetrahedral_flow(P); Q
```

```
[60]: (60*x0^3*x1^2*x2 + 72*x0^2*x1*x2^3 + 12*x0*x2^5)*xi0*x12 + (60*x0^2*x1^3*x2 +
72*x0*x1^2*x2^3 + 12*x1*x2^5)*xi1*x12
```

```
[61]: PC = PoissonComplex(P)
```

[62]: `PC(Q).is_coboundary(certificate=True)`

[62]: (True,  
Poisson cochain  $(-48*x_0^2*x_1*x_2)*x_{i0} + (-48*x_0*x_1^2*x_2)*x_{i1} + (60*x_0^2*x_1^2 - 12*x_2^4)*x_{i2}$ )

$\mathfrak{sl}_2(\mathbb{R})$  with  $R = \begin{pmatrix} x_0 & x_1 \\ x_2 & x_3 \end{pmatrix} \mapsto \begin{pmatrix} x_0 & x_1 \\ -x_2 & x_3 \end{pmatrix}$ , quadratic Poisson structure  $\equiv 0$

[63]: `P = r_matrix_poisson_bivector(sl_basis(QQ, 2), 2, R_matrix=R_weak); P`

[63]: 0

$\mathfrak{sl}_2(\mathbb{R})$  with  $R = \begin{pmatrix} x_0 & x_1 \\ x_2 & x_3 \end{pmatrix} \mapsto \begin{pmatrix} x_0 & x_1 \\ -x_2 & x_3 \end{pmatrix}$ , cubic Poisson structure

[64]: `P = r_matrix_poisson_bivector(sl_basis(QQ, 2), 3, R_matrix=R_weak); P`

[64]:  $(2*x_0*x_1*x_2)*x_{i0}*x_{i1} + (-x_0^2*x_1)*x_{i0}*x_{i2} + (-x_1*x_2^2)*x_{i1}*x_{i2}$

[65]: `P.bracket(P)`

[65]: 0

[66]: `Q = tetrahedral_flow(P); Q`

[66]:  $(-384*x_0^2*x_1^3*x_2 + 384*x_0*x_1^2*x_2^3 + 96*x_1*x_2^5)*x_{i1}*x_{i2} + (1152*x_0^3*x_1^2*x_2 - 384*x_0^2*x_1*x_2^3)*x_{i0}*x_{i2} + (-384*x_0^3*x_1^3 + 384*x_0^2*x_1^2*x_2^2 - 480*x_0*x_1*x_2^4)*x_{i0}*x_{i1}$

[67]: `PC = PoissonComplex(P)`

[68]: `PC(Q).is_coboundary(certificate=True)`

[68]: (True,  
Poisson cochain  $(384*x_0^2*x_1*x_2 - 192*x_0*x_2^3)*x_{i0} + (-384*x_0*x_1^2*x_2)*x_{i1} + (384*x_0^2*x_1^2 - 48*x_2^4)*x_{i2}$ )

## 8.4 The Lie algebra of traceless $3 \times 3$ matrices

$\mathfrak{sl}_3(\mathbb{R})$  with  $R = \text{id}$ , cubic Poisson structure

[69]: `P = r_matrix_poisson_bivector(sl_basis(QQ, 3), 3, R_matrix=R_id); P`

[69]:  $(x_1^2*x_3 - x_0*x_1*x_4 + x_1*x_2*x_6 - x_0*x_2*x_7)*x_{i0}*x_{i1} + (x_0^2*x_2 + x_1*x_2*x_3 + x_0*x_2*x_4 - x_0*x_1*x_5 + x_2^2*x_6)*x_{i0}*x_{i2} + (-x_1*x_3^2 + x_0*x_3*x_4 - x_2*x_3*x_6 + x_0*x_5*x_6)*x_{i0}*x_{i3} + (x_1*x_5*x_6 - x_2*x_3*x_7)*x_{i0}*x_{i4} + (x_0*x_2*x_3 + 2*x_2*x_3*x_4 - x_1*x_3*x_5 + x_2*x_5*x_6)*x_{i0}*x_{i5} + (-x_0^2*x_6 - x_1*x_3*x_6 - x_0*x_4*x_6 - x_2*x_6^2 + x_0*x_3*x_7)*x_{i0}*x_{i6} + (-x_0*x_1*x_6 - 2*x_1*x_4*x_6 + x_1*x_3*x_7 - x_2*x_6*x_7)*x_{i0}*x_{i7} + (x_0*x_1*x_2 + 2*x_1*x_2*x_4 - x_1^2*x_5 + x_2^2*x_7)*x_{i1}*x_{i2} + (x_0*x_1*x_3 - x_0^2*x_4 - x_1*x_3*x_4 + x_0*x_4^2 - x_2*x_4*x_6 + x_0*x_5*x_7)*x_{i1}*x_{i3} + (x_1^2*x_3 - x_0*x_1*x_4 - x_2*x_4*x_7 + x_1*x_5*x_7)*x_{i1}*x_{i4} + (x_1*x_2*x_3 + 2*x_2*x_4^2 - x_1*x_4*x_5 + x_2*x_5*x_7)*x_{i1}*x_{i5} + (x_0*x_1*x_6 - 2*x_0^2*x_7 - x_1*x_3*x_7 - x_2*x_6*x_7)*x_{i1}*x_{i6} + (x_1^2*x_6 - 2*x_0*x_1*x_7 - x_1*x_4*x_7 - x_2*x_7^2)*x_{i1}*x_{i7} + (x_0*x_2*x_3 -$

$$\begin{aligned}
& 2*x0^2*x5 - x1*x3*x5 - x2*x5*x6)*xi2*xi3 + (x1*x2*x3 - 2*x0*x1*x5 - x1*x4*x5 - \\
& x2*x5*x7)*xi2*xi4 + (x2^2*x3 - x0*x2*x5 + x2*x4*x5 - x1*x5^2)*xi2*xi5 + (2*x0^3 + \\
& x0*x1*x3 + 3*x0^2*x4 + x1*x3*x4 + x0*x4^2 + 2*x0*x2*x6 + x2*x4*x6 + x0*x5*x7)*xi2*xi6 \\
& + (2*x0^2*x1 + 4*x0*x1*x4 + 2*x1*x4^2 + x1*x2*x6 + x0*x2*x7 + x2*x4*x7 + \\
& x1*x5*x7)*xi2*xi7 + (-x1*x3^2 + x0*x3*x4 + x4*x5*x6 - x3*x5*x7)*xi3*xi4 + (-x2*x3^2 + \\
& 2*x0*x3*x5 + x3*x4*x5 + x5^2*x6)*xi3*xi5 + (-x0*x3*x6 - 2*x3*x4*x6 - x5*x6^2 + \\
& x3^2*x7)*xi3*xi6 + (-x1*x3*x6 - 2*x4^2*x6 + x3*x4*x7 - x5*x6*x7)*xi3*xi7 + (-x2*x3*x4 \\
& + x1*x3*x5 + x0*x4*x5 + x4^2*x5 + x5^2*x7)*xi4*xi5 + (x1*x3*x6 - 2*x0*x3*x7 - x3*x4*x7 \\
& - x5*x6*x7)*xi4*xi6 + (x1*x4*x6 - x1*x3*x7 - x0*x4*x7 - x4^2*x7 - x5*x7^2)*xi4*xi7 + \\
& (2*x0^2*x3 + 4*x0*x3*x4 + 2*x3*x4^2 + x2*x3*x6 + x0*x5*x6 + x4*x5*x6 + \\
& x3*x5*x7)*xi5*xi6 + (x0*x1*x3 + x0^2*x4 + x1*x3*x4 + 3*x0*x4^2 + 2*x4^3 + x2*x4*x6 + \\
& x0*x5*x7 + 2*x4*x5*x7)*xi5*xi7 + (-x1*x6^2 + x0*x6*x7 - x4*x6*x7 + x3*x7^2)*xi6*xi7
\end{aligned}$$

```
[70]: P.bracket(P)
```

```
[70]: 0
```

```
[71]: #%time Q = tetrahedral_flow(P); Q
```



# Chapter 9

## Graph complex action on star products

In this chapter we bring everything together. We use star products and gauge transformations from Chapters 1 and 3, Poisson structures from Chapter 2, graph cocycles from Chapters 4 and 5, and flows from Chapters 6, 7, and 8.

Finally, let us bring together, compare and contrast two types of deformations: Kontsevich’s universal construction of deformations of Poisson brackets (i.e. Licherowicz–Poisson classes) and gauge transformations of star products.

```
[1]: from gcaops.graph.formality_graph import FormalityGraph
      from gcaops.graph.formality_graph_basis import KontsevichGraphBasis
      from gcaops.graph.formality_graph_complex import FormalityGraphComplex
      from gcaops.graph.formality_graph_operator import FormalityGraphOperator
      set_verbose(-1)
```

Define the quotient ring  $SR[\hbar, \varepsilon]/(\hbar^5, \varepsilon^2)$ , i.e. the symbolic ring  $SR$  with the variables  $\hbar$  and  $\varepsilon$  —conveniently satisfying  $\hbar^5 = 0$  and  $\varepsilon^2 = 0$ , respectively— adjoined to it:

```
[2]: R = PolynomialRing(SR, 2, names='hbar,eps')
      Q.<hbar,eps> = R.quotient(R.ideal(R.gen(0)^5, R.gen(1)^2))
      Q.element_class.derivative = lambda self, x: self.parent()(self.lift().derivative(x))
```

We recall the Proposition from §1.3: gauge transformations  $T = \text{id} + \hbar^k T_k \bmod \bar{o}(\hbar^k)$  concentrated in degree  $k$  act on  $\star$ -products modulo  $\bar{o}(\hbar^k)$  by adding the Hochschild coboundaries  $\hbar d_H(T_k) \bmod \bar{o}(\hbar^k)$ .

### 9.1 Poisson-trivial deformations and gauge transformations

Now let us take a *Poisson-trivial* deformation  $P \mapsto P + \varepsilon[[P, X]] \bmod \bar{o}(\varepsilon)$  of a given Poisson structure, and induce a deformation of the Kontsevich star product. This results not in adding a Hochschild coboundary (that is, not a Gerstenhaber bracket with the usual multiplication) but this results in adding to  $\star$  the Gerstenhaber bracket  $[\hbar P, X]_G$  of the Poisson bi-vector  $\hbar P$  (viewed as a bi-differential operator) with the vector field  $X$  in the trivial deformation. Hence, the deformation of the star product is produced by a gauge transformation  $T = \text{id} + \hbar^k \cdot \text{const} \cdot X$ ; here the derivation  $X$  is given by the

sunflower graph,  $k = 3$ , the star-product is affected at order 4, and all structures are taken modulo  $\bar{o}(\hbar^4)$ .

**Example.** Let  $P = x^2y\xi_1\xi_2$  be a Poisson structure (on  $\mathbb{R}^2$  with coordinates  $x, y$ ):

```
[3]: from gcaops.algebra.superfunction_algebra import SuperfunctionAlgebra
      from gcaops.algebra.polydifferential_operator import PolyDifferentialOperatorAlgebra
      SA.<xi1,xi2> = SuperfunctionAlgebra(Q, var('x,y'))
      PA.<ddx,ddy> = PolyDifferentialOperatorAlgebra(Q, var('x,y'))
```

```
[4]: P = y*x^2*xi1*xi2; P
```

```
[4]: (x^2*y)*xi1*xi2
```

Calculate the Kontsevich star product expansion mod  $\bar{o}(\hbar^4)$  for the Poisson structure  $P$ :

```
[5]: FGC = FormalityGraphComplex(SR, lazy=True); FGC
```

```
[5]: Formality graph complex over Symbolic Ring with Basis consisting of representatives of isomorphism classes of formality graphs with no automorphisms that induce an odd permutation on edges
```

```
[6]: star4_txt = open('data/star4.txt').read().rstrip()
      star4 = FGC.element_from_kgs_encoding(star4_txt) #; star4
```

```
[7]: star4_operator = FormalityGraphOperator(SA, PA, star4)
      %time star4_op = star4_operator.value_at_copies_of(hbar*P) #; star4_op
```

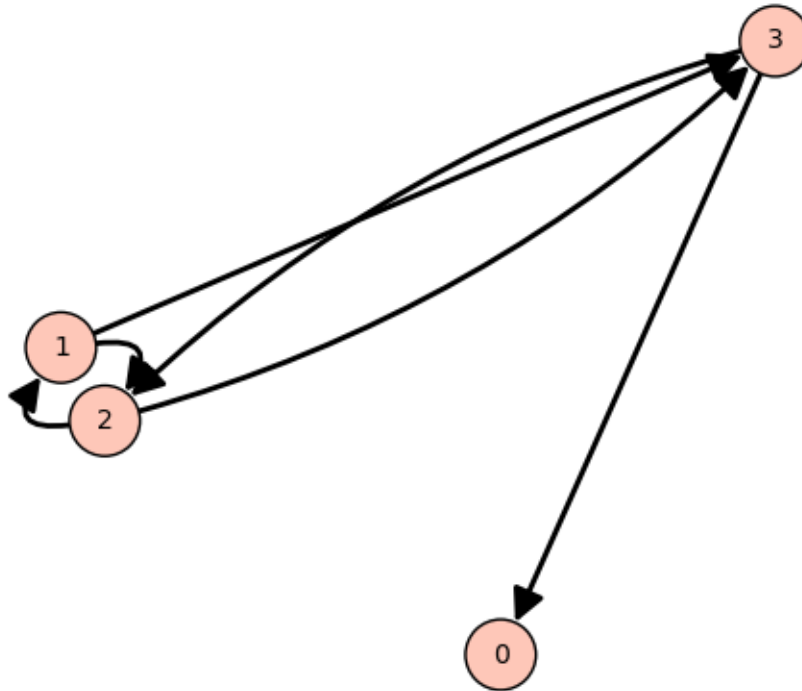
```
CPU times: user 1min 52s, sys: 74.9 ms, total: 1min 52s
Wall time: 1min 52s
```

Deform the Poisson structure  $P$  by using the sunflower graph at  $\varepsilon$  (cf. [6]):

```
[8]: KGB = KontsevichGraphBasis()
```

```
[9]: sunflower = list(KGB.graphs(1,3))[-1]; sunflower.show()
```





```
[10]: X = FGC(sunflower); X
```

```
[10]: 1*FormalityGraph(1, 3, [(1, 2), (1, 3), (2, 1), (2, 3), (3, 0), (3, 2)])
```

The value of this graph at three copies of  $\hbar P$  is a unary differential operator:

```
[11]: X_operator = FormalityGraphOperator(SA, PA, X)
X_op = X_operator.value_at_copies_of(hbar*P); X_op
```

```
[11]: ((-4*x^4)*hbar^3)*ddx + (8*x^3*y*hbar^3)*ddy
```

This unary first-order differential operator is naturally a vector field (i.e. a superfunction linear in  $\xi$ 's):

```
[12]: X_vec = SA(X_op); X_vec
```

```
[12]: ((-4*x^4)*hbar^3)*xi1 + (8*x^3*y*hbar^3)*xi2
```

So, let us construct a Poisson-trivial deformation  $P \mapsto P + \varepsilon[[P, X]] + \bar{o}(\varepsilon) + \bar{o}(\hbar^3)$ .

```
[13]: P2 = P + eps*P.bracket(X_vec); P2
```

```
[13]: ((-8*x^5*y)*hbar^3*eps + x^2*y)*xi1*xi2
```

Calculate the star product expansion  $\star' \bmod \bar{o}(\hbar^4)$  and  $\bmod \bar{o}(\varepsilon)$  for the deformed Poisson structure  $P_2$ :

```
[14]: %time star4_deformed_op = star4_operator.value_at_copies_of(hbar*P2) #;
↪star4_deformed_op
```

CPU times: user 3min 38s, sys: 216 ms, total: 3min 39s  
Wall time: 3min 39s

The difference  $\star' - \star$  between the two star product expansions modulo  $\bar{o}(\hbar^4)$  and  $\bar{o}(\varepsilon)$  is the following bi-differential operator, naturally given by the bi-vector  $\hbar\varepsilon\llbracket P, X \rrbracket$ :

```
[15]: star4_deformed_op - star4_op
```

```
[15]: ((-8*x^5*y)*hbar^4*eps)*(ddx @ ddy) + (8*x^5*y*hbar^4*eps)*(ddy @ ddx)
```

The difference is a Hochschild cocycle because it is a derivation in each argument:

```
[16]: (star4_deformed_op - star4_op).hochschild_differential()
```

```
[16]: 0
```

**Remark.** The difference is not a Hochschild coboundary, yet the difference is equal to  $\varepsilon$  times the Gerstenhaber bracket of the Poisson bi-vector  $\hbar P$  and the “sunflower” vector field  $X$ :

```
[17]: hP_op = PA(hbar*P); hP_op
```

```
[17]: (x^2*y*hbar)*(ddx @ ddy) + ((-x^2*y)*hbar)*(ddy @ ddx)
```

The above is a bi-derivation. Let us take its Gerstenhaber bracket with the derivation  $X$ .

```
[18]: eps*hP_op.bracket(X_op)
```

```
[18]: ((-8*x^5*y)*hbar^4*eps)*(ddx @ ddy) + (8*x^5*y*hbar^4*eps)*(ddy @ ddx)
```

This justifies our remark.

The sunflower graph is a derivation, hence its Hochschild differential vanishes and a gauge transformation  $T = \text{id} + \varepsilon X$  with this graph at  $\hbar^3$  does not affect the star-product at  $\hbar^3$ , but it will affect  $\star$  at order  $\hbar^4$  — indeed, by adding the Gerstenhaber bracket of  $X$  not with the usual multiplication at  $\hbar^0$  but with the Poisson structure  $P$  in the next order  $\hbar^1$ . This is how the deformation is realized by a gauge transformation:

```
[19]: T_op = PA.identity_operator() + eps*X_op
```

```
[20]: T_inverse_op = PA.identity_operator() - eps*X_op
```

```
[21]: %time star4_gauged_op = T_inverse_op.insertion(0, star4_op.insertion(0, T_op).
      ↪insertion(1, T_op))
```

CPU times: user 15.7 s, sys: 40 ms, total: 15.7 s  
Wall time: 15.7 s

```
[22]: star4_gauged_op - star4_op
```

```
[22]: ((-8*x^5*y)*hbar^4*eps)*(ddx @ ddy) + (8*x^5*y*hbar^4*eps)*(ddy @ ddx)
```

Three outputs in this subsection are identical.

## 9.2 Poisson-trivial deformation and the gauge transform in terms of graphs

**Claim.** The gauge transformation of the star-product from the previous subsection (in the above, restricted to a particular Poisson structure  $P$ ) is produced by using Kontsevich graphs (likewise, containing a copy of this Poisson structure  $P$  in every aerial vertex).

Now, the task is to solve —for the gauge and Leibniz graph coefficients— the equation,

$$\star' - \star \stackrel{\text{def}}{=} \varepsilon \llbracket \hbar P, X((\hbar P)^{\otimes 3}) \rrbracket = \varepsilon \llbracket \hbar P, T_3((\hbar P)^{\otimes 3}) \rrbracket_G + \diamond(P, \llbracket P, P \rrbracket) \pmod{\bar{o}(\hbar^4)} \pmod{\bar{o}(\varepsilon)}.$$

First, let us generate the gauge graphs:

```
[23]: len(KGB.graphs(1,3))
```

```
[23]: 4
```

Gauge transformation in terms of graphs:

```
[24]: G = FGC(FormalityGraph(1,0,[])) + FGC([(var('g13_{}'.format(k)),g) for k,g in
↳ enumerate(KGB.graphs(1,3))])
```

```
[25]: #G.show()
```

```
[26]: G_inverse = FGC(FormalityGraph(1,0,[])) - G.homogeneous_part(1, 3, 6)
```

Let us check that  $G\_inverse$  is the inverse of gauge transformation  $G$  modulo  $\bar{o}(\hbar^4)$ .

```
[27]: G.insertion(0, G_inverse, max_num_aerial=4)
```

```
[27]: 1*FormalityGraph(1, 0, [])
```

```
[28]: G_inverse.insertion(0, G, max_num_aerial=4)
```

```
[28]: 1*FormalityGraph(1, 0, [])
```

```
[29]: %time star4_gauged = G_inverse.insertion(0, star4.insertion(0, G, max_num_aerial=4).
↳ insertion(1, G, max_num_aerial=4), max_num_aerial=4)
```

```
CPU times: user 1.25 s, sys: 4 ms, total: 1.26 s
Wall time: 1.26 s
```

The gauge transformation adds 72 terms to the star-product modulo  $\bar{o}(\hbar^4)$ :

```
[30]: len(star4_gauged - star4)
```

```
[30]: 72
```

At  $\hbar^3$  the gauge transformation amounts to adding the Hochschild differential of the terms in  $G$  at  $\hbar^3$ :

```
[31]: (star4_gauged - star4).homogeneous_part(2,3,6) == G.homogeneous_part(1,3,6).
↳ hochschild_differential()
```

[31]: True

At  $\hbar^4$  the gauge transformation amounts to adding the Hochschild differential of the terms in  $G$  at  $\hbar^4$  and the Gerstenhaber bracket [wedge, terms in  $G$  at  $\hbar^3$ ], where wedge stands at  $\hbar^1$  in  $\star$ :

[32]: `wedge = star4.homogeneous_part(2,1,2); wedge`

[32]: `1*FormalityGraph(2, 1, [(2, 0), (2, 1)])`

[33]: `(star4_gauged - star4).homogeneous_part(2,4,8) == G.homogeneous_part(1,4,8).  
↪hochschild_differential() + wedge.gerstenhaber_bracket(G.homogeneous_part(1,3,6))`

[33]: True

Now, before generating any Leibniz graphs for  $\diamond$ , and actually instead of doing that, let us inspect that the equation  $[[P, X]] = [P, T_3]_G$  has a solution  $T_3$  in terms of gauge graphs:

[34]: `obstruction = wedge.schouten_bracket(FGC(sunflower)) - (star4_gauged - star4)  
eqns = [c == 0 for c,g in obstruction]  
solve(eqns, [g for (g,) in G.homogeneous_part(1,3,6)])`

[34]: `[[g13_0 == 0, g13_1 == 0, g13_2 == 0, g13_3 == -2]]`

But what is the gauge graph (on 3 aerial vertices and 1 sink) whose coefficient is `g13_3`?

[35]: `KGB.graphs(1,3)[3]`

[35]: `FormalityGraph(1, 3, [(1, 2), (1, 3), (2, 1), (2, 3), (3, 0), (3, 2)])`

We observe that the sought-for gauge transformation  $T = \text{id} + \hbar^3 T_3 \text{ mod } \bar{o}(\hbar^4)$  of the star-product is completely determined by the “sunflower” graph that gave us the vector field  $X$  for a Poisson-trivial deformation of the bracket  $P$  inside the star-product.

In particular, the formula of gauge transformation for the specific Poisson bi-vector  $P = x^2 y \xi_1 \xi_2$  (see the Example in §9.1) is obtained by evaluating the sunflower graph at a copy of  $P$  in each aerial vertex.

### 9.3 How the tetrahedral flow deforms the star-product

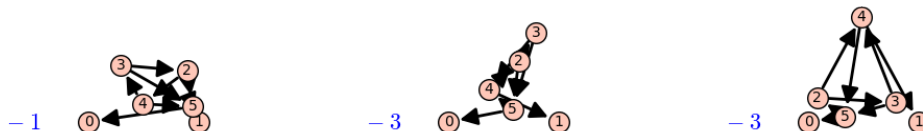
Last but not least, we recall the gauge construction from the previous paragraph, now taking the Kontsevich tetrahedral flow to deform the Poisson bracket. We keep in mind that there is no universal mechanism (over all affine manifolds in all dimensions at once) for the Kontsevich tetrahedral flow to be Poisson-trivial in terms of graphs. So the graph equation to solve (for the gauge and Leibniz graph coefficients) is this:

$$\star' - \star \stackrel{\text{def}}{=} \varepsilon Q_{\text{tetra}}((\hbar P)^{\otimes 4}) = \varepsilon \hbar^3 d_H(T_3) + \varepsilon \hbar^3 [\hbar P, T_3]_G + \varepsilon \hbar^4 d_H(T_4) + \diamond(P, [[P, P]]) \text{ mod } \bar{o}(\hbar^4) \text{ mod } \bar{o}(\varepsilon).$$

First let us define the tetrahedral flow in terms of Kontsevich graphs (see [27] and [6]):

```
[36]: Q_tetra = FGC.element_from_kgs_encoding("""h^4:
2 4 1  0 1 2 4 2 5 2 3   1
2 4 1  0 3 1 4 2 5 2 3  -3
2 4 1  0 3 4 5 1 2 2 4  -3""")
```

```
[37]: Q_tetra.show()
```



```
[38]: Q_tetra
```

```
[38]: (-1)*FormalityGraph(2, 4, [(2, 4), (2, 5), (3, 2), (3, 5), (4, 3), (4, 5), (5, 0), (5,
1)]) + (-3)*FormalityGraph(2, 4, [(2, 3), (2, 5), (3, 4), (3, 5), (4, 1), (4, 2), (5,
0), (5, 4)]) + (-3)*FormalityGraph(2, 4, [(2, 3), (2, 4), (3, 4), (3, 5), (4, 1), (4,
5), (5, 0), (5, 2)])
```

```
[39]: star4_tetra_deformed = star4 + Q_tetra
```

To generate the gauge transformation, we remember the four gauge graphs for  $T_3$  on three aerial vertices, and we generate 60 new gauge graphs on four aerial vertices:

```
[40]: len(KGB.graphs(1,4))
```

```
[40]: 60
```

```
[41]: G4 = FGC(FormalityGraph(1,0,[])) + FGC([(var('g13_{}'.format(k)),g) for k,g in
↳ enumerate(KGB.graphs(1,3))]) + FGC([(var('g14_{}'.format(k)),g) for k,g in
↳ enumerate(KGB.graphs(1,4))])
```

```
[42]: G4_inverse = FGC(FormalityGraph(1,0,[])) - G.homogeneous_part(1,3,6) - G.
↳ homogeneous_part(1,4,8)
```

```
[43]: %time star4_gauged4 = G4_inverse.insertion(0, star4.insertion(0, G4,
↳ max_num_aerial=4).insertion(1, G4, max_num_aerial=4), max_num_aerial=4)
```

```
CPU times: user 1.66 s, sys: 4 ms, total: 1.66 s
Wall time: 1.66 s
```

Generate Leibniz graphs on 2 sinks and 3 aerial vertices:

```
[44]: from gcaops.graph.formality_graph_basis import LeibnizGraphBasis
LGB = LeibnizGraphBasis(positive_differential_order=True)
```

```
[45]: L23 = FGC([(var('l{}'.format(k)),g) for k, g in enumerate(LGB.graphs(2,3))])
```

```
[46]: len(L23)
```

```
[46]: 60
```

Expand Leibniz graphs into Kontsevich graphs built of wedges:

```
[47]: stick = FGC(FormalityGraph(0,2,[(0,1)])); stick
```

```
[47]: 1*FormalityGraph(0, 2, [(0, 1)])
```

```
[48]: L23_expanded = sum(L23.insertion(k,stick,max_out_degree=2) for k in [2,3,4])
```

```
[49]: len(L23_expanded)
```

```
[49]: 235
```

Try solving the system of linear algebraic equations for the coefficients of all the gauge and Leibniz graphs:

```
[50]: obstruction = Q_tetra - ((star4_gauged4 - star4) + L23_expanded)
eqns3 = [c for c,g in obstruction.homogeneous_part(2,3,6)]
eqns4 = [c for c,g in obstruction.homogeneous_part(2,4,8)]
solve(eqns3 + eqns4,
      [l for (l,_) in L23] + \
      [g for (g,_) in G.homogeneous_part(1,3,6)] + \
      [g for (g,_) in G.homogeneous_part(1,4,8)])
```

```
[50]: []
```

No solution. Indeed, on the one hand the “tetrahedron on top of wedge” graph  $\Gamma$  in  $Q_{\text{tetra}}$  can only be affected by the “tetrahedron with one sink” gauge graph  $\gamma$  at  $\hbar^3$  with coefficient  $\mathbf{g13\_0}$  (and  $\Gamma$  cannot be affected by any Leibniz graph, because the contraction of any edge in  $\Gamma$  results in a zero graph); hence the coefficient  $\mathbf{g13\_0}$  of  $\gamma$  must be  $-1$ :

```
[51]: eqns4[0]
```

```
[51]: -g13_0 - 1
```

But on the other hand the Hochschild differential of the “tetrahedron with one sink” graph  $\gamma$  is nonzero, and shows up at  $\hbar^3$ , which forces  $\mathbf{g13\_0}$  to be zero:

```
[52]: solve(eqns3, [g for (g,_) in G.homogeneous_part(1,3,6)])
```

```
[52]: [[g13_0 == 0, g13_1 == 0, g13_2 == 0, g13_3 == r1]]
```

So the deformation of the star product modulo  $\bar{o}(\hbar^4)$  given by the tetrahedron cannot be realized as a gauge transformation in terms of graphs.

# List of references

- [1] Peter Banks, Erik Panzer, and Brent Pym. Multiple zeta values in deformation quantization. *Invent. Math.*, 222(1):79–159, 2020.
- [2] François Bayen, Moshé Flato, Christian Frønsdal, André Lichnerowicz, and Daniel Sternheimer. Deformation theory and quantization. I. Deformations of symplectic structures. *Ann. Physics*, 111(1):61–110, 1978.
- [3] Nabih Ben Amar.  $K$ -star products on dual of Lie algebras. *J. Lie Theory*, 13(2):329–357, 2003.
- [4] Nabih Ben Amar. A comparison between Rieffel’s and Kontsevich’s deformation quantizations for linear Poisson tensors. *Pacific J. Math.*, 229(1):1–24, 2007.
- [5] Anass Bouisaghouane. The Kontsevich tetrahedral flow in 2D: a toy model. Preprint [arXiv:1702.06044](#) [math.DG] — 6 -p., 2017.
- [6] Anass Bouisaghouane, Ricardo Buring, and Arthemy V. Kiselev. The Kontsevich tetrahedral flow revisited. *J. Geom. Phys.*, 119:272–285, 2017. Preprint [arXiv:1608.01710](#) [q-alg] — 29 p.
- [7] Francis Brown. Mixed Tate motives over  $\mathbb{Z}$ . *Ann. of Math. (2)*, 175(2):949–976, 2012.
- [8] Ricardo Buring and Arthemy V. Kiselev. Universal cocycles and the graph complex action on homogeneous Poisson brackets by diffeomorphisms. *Physics of Particles and Nuclei Letters*, 17(5):707–713, 2020. (Proc. International workshop SQS’19 on Supersymmetries and Quantum Symmetries, 26–31 August 2019, Yerevan, Armenia) Preprint [arXiv:1912.12664](#) [math.SG] — 8 p.
- [9] Ricardo Buring and Arthemy V. Kiselev. The expansion  $\star \bmod \bar{o}(\hbar^4)$  and computer-assisted proof schemes in the Kontsevich deformation quantization. *Exp. Math. (in press)*, 31(3 or 4), 2022. Preprint [arXiv:1702.00681](#) [math.CO] — 77 p.
- [10] Ricardo Buring, Arthemy V. Kiselev, and Nina J. Rutten. Infinitesimal deformations of Poisson bi-vectors using the Kontsevich graph calculus. *J. Phys.: Conf. Ser.*, 965, Paper 012010, 2018. Proc. XXV Int. conf. ‘Integrable Systems & Quantum Symmetries’ (6–10 June 2017, CVUT Prague, Czech Republic). Preprint [arXiv:1710.02405](#) [math.CO] — 12 p.
- [11] Ricardo Buring, Arthemy V. Kiselev, and Nina J. Rutten. Poisson brackets symmetry from the pentagon-wheel cocycle in the graph complex. *Physics of Particles and*

- Nuclei*, 49(5):924–928, 2018. (Proc. International workshop SQS'17 on Supersymmetries and quantum symmetries, July 31 – August 5, 2017, JINR Dubna, Russia) Preprint [arXiv:1712.05259](https://arxiv.org/abs/1712.05259) [math-ph] — 4 p.
- [12] Alberto S. Cattaneo. Formality and star products. In *Poisson geometry, deformation quantisation and group representations*, volume 323 of *London Math. Soc. Lecture Note Ser.*, pages 79–144. Cambridge Univ. Press, Cambridge, 2005. Lecture notes taken by D. Indelicato.
- [13] Willem A. de Kok. The relation between Feynman diagrams and Kontsevich graphs in non-commutative star products. Master's thesis, University of Groningen, 2021.
- [14] Marc De Wilde and Pierre B.A. Lecomte. Formal deformations of the Poisson Lie algebra of a symplectic manifold and star-products. Existence, equivalence, derivations. In *Deformation theory of algebras and structures and applications (Il Ciocco, 1986)*, volume 247 of *NATO Adv. Sci. Inst. Ser. C: Math. Phys. Sci.*, pages 897–960. Kluwer Acad. Publ., Dordrecht, 1988.
- [15] The Sage developers. *SageMath, the Sage Mathematics Software System (Version 9.2)*, 2021. <https://www.sagemath.org>.
- [16] Vasily A. Dolgushev. Stable formality quasi-isomorphisms for Hochschild cochains. *Mém. Soc. Math. Fr. (N.S.)*, (168):vi + 108 pages, 2021.
- [17] Vasily A. Dolgushev, Christopher L. Rogers, and Thomas H. Willwacher. Kontsevich's graph complex, GRT, and the deformation complex of the sheaf of polyvector fields. *Ann. of Math. (2)*, 182(3):855–943, 2015.
- [18] Chiara Esposito. *Formality theory: From Poisson structures to deformation quantization*, volume 2 of *SpringerBriefs in Mathematical Physics*. Springer, Cham, 2015.
- [19] Boris V. Fedosov. A simple geometrical construction of deformation quantization. *J. Differential Geom.*, 40(2):213–238, 1994.
- [20] Giovanni Felder and Boris Shoikhet. Deformation quantization with traces. *Lett. Math. Phys.*, 53(1):75–86, 2000. Preprint [arXiv:math/0002057](https://arxiv.org/abs/math/0002057) [math.QA] — 14 p.
- [21] Giovanni Felder and Thomas Willwacher. On the (ir)rationality of Kontsevich weights. *Int. Math. Res. Not. IMRN*, (4):701–716, 2010. Preprint [arXiv:0808.2762](https://arxiv.org/abs/0808.2762) [math.QA] — 11 p.
- [22] Murray Gerstenhaber. The cohomology structure of an associative ring. *Ann. of Math. (2)*, 78:267–288, 1963.
- [23] Murray Gerstenhaber. On the deformation of rings and algebras. *Ann. of Math. (2)*, 79:59–103, 1964.
- [24] Hilbrand J. Groenewold. On the principles of elementary quantum mechanics. *Physica*, 12:405–460, 1946.
- [25] Gerhard P. Hochschild. On the cohomology groups of an associative algebra. *Ann. of Math. (2)*, 46:58–67, 1945.



- [26] Maxim Kontsevich. Formal (non)commutative symplectic geometry. In *The Gelfand Mathematical Seminars, 1990–1992*, pages 173–187. Birkhäuser Boston, Boston, MA, 1993.
- [27] Maxim Kontsevich. Formality conjecture. In *Deformation theory and symplectic geometry (Ascona, 1996)*, volume 20 of *Math. Phys. Stud.*, pages 139–156. Kluwer Acad. Publ., Dordrecht, 1997.
- [28] Maxim Kontsevich. Deformation quantization of Poisson manifolds. *Lett. Math. Phys.*, 66(3):157–216, 2003.
- [29] Maxim Kontsevich. XI Solomon Lefschetz Memorial Lecture Series: Hodge structures in non-commutative geometry. *Morfismos*, 11(2):1–32, 2007. Notes by Ernesto Lupercio.
- [30] Camille Laurent-Gengoux, Anne Pichereau, and Pol Vanhaecke. *Poisson structures*, volume 347 of *Grundlehren der mathematischen Wissenschaften*. Springer, Heidelberg, 2013.
- [31] Luen-Chau Li and Serge Parmentier. Nonlinear Poisson structures and  $r$ -matrices. *Comm. Math. Phys.*, 125(4):545–563, 1989.
- [32] Brendan McKay and Dror Bar-Natan. Graph Cohomology - An Overview and Some Computations. Preprint <https://www.math.toronto.edu/~drorbn/papers/GCOC/GCOC.ps> — 13 p., 2001.
- [33] Nina Rutten and Arthemy V. Kiselev. The defining properties of the Kontsevich unoriented graph complex. *Journal of Physics: Conference Series*, 1194:1–10, 2019. Paper 012095.
- [34] Hermann Weyl. *Gruppentheorie und Quantenmechanik*. Wissenschaftliche Buchgesellschaft, Darmstadt, second edition, 1977.
- [35] Thomas Willwacher. M. Kontsevich’s graph complex and the Grothendieck-Teichmüller Lie algebra. *Invent. Math.*, 200(3):671–760, 2015.
- [36] Thomas Willwacher. The homotopy braces formality morphism. *Duke Math. J.*, 165(10):1815–1964, 2016.
- [37] Thomas Willwacher and Carlo A. Rossi. P. Etingof’s conjecture about Drinfeld associators. Preprint [arXiv:1404.2047](https://arxiv.org/abs/1404.2047) [math.QA] — 47 p., 2014.
- [38] Thomas Willwacher and Marko Živković. Multiple edges in M. Kontsevich’s graph complexes and computations of the dimensions and Euler characteristics. *Adv. Math.*, 272:553–578, 2015.



# Part II

## Research articles

### Abstract

This part consists of eleven articles that can be read independently. Each article is introduced with a commentary that discusses its relation to Part I of the dissertation.

### Contents of Part II

10	On the Kontsevich $\star$ -product associativity mechanism .....	217
11	The expansion $\star \bmod \bar{o}(\hbar^4)$ and computer-assisted proof schemes in the Kontsevich deformation quantization .....	223
12	Formality morphism as the mechanism of $\star$ -product associativity: how it works	303
13	The heptagon-wheel cocycle in the Kontsevich graph complex .....	321
14	Infinitesimal deformations of Poisson bi-vectors using the Kontsevich graph calculus .....	345
15	The Kontsevich tetrahedral flow revisited .....	359
16	Poisson brackets symmetry from the pentagon-wheel cocycle in the graph complex .....	389
17	The orientation morphism: from graph cocycles to deformations of Poisson structures .....	401
18	The Kontsevich graph orientation morphism revisited .....	415
19	Universal cocycles and the graph complex action on homogeneous Poisson brackets by diffeomorphisms .....	435
20	The hidden symmetry of Kontsevich's graph flows on the spaces of Nambu-determinant Poisson brackets .....	445



# Chapter 10

## On the Kontsevich $\star$ -product associativity mechanism

This chapter is based on the peer-reviewed journal publication *R. Buring and A.V. Kiselev, Physics of Particles and Nuclei Letters*, **14**(2), 403–407, 2017. (Preprint [arXiv:1602.09036](https://arxiv.org/abs/1602.09036) [q-alg] – 4 p.)

*Commentary.* In reference to Part **I** of the dissertation, the material of this chapter is used in Chapter **1**. The typo in the definition of  $I_f$  is corrected in Chapter **12**. The associativity mechanism from this chapter is explained in Chapter **12**.

# On the Kontsevich $\star$ -product associativity mechanism

R. Buring\*, A. V. Kiselev\*,<sup>§</sup>

## Abstract

The deformation quantization by Kontsevich is a way to construct an associative non-commutative star-product  $\star = \times + \hbar \{ , \}_\mathcal{P} + \bar{o}(\hbar)$  in the algebra of formal power series in  $\hbar$  on a given finite-dimensional affine Poisson manifold: here  $\times$  is the usual multiplication,  $\{ , \}_\mathcal{P} \neq 0$  is the Poisson bracket, and  $\hbar$  is the deformation parameter. The product  $\star$  is assembled at all powers  $\hbar^{k \geq 0}$  via summation over a certain set of weighted graphs with  $k+2$  vertices; for each  $k > 0$ , every such graph connects the two co-multiples of  $\star$  using  $k$  copies of  $\{ , \}_\mathcal{P}$ . Cattaneo and Felder interpreted these topological portraits as genuine Feynman diagrams in the Ikeda–Izawa model for quantum gravity.

By expanding the star-product up to  $\bar{o}(\hbar^3)$ , i.e., with respect to graphs with at most five vertices but possibly containing loops, we illustrate the mechanism  $\text{Assoc} = \diamond (\text{Poisson})$  that converts the Jacobi identity for the bracket  $\{ , \}_\mathcal{P}$  into the associativity of  $\star$ .

Denote by  $\times$  the multiplication in the commutative associative unital algebra  $C^\infty(N^n \rightarrow \mathbb{R})$  of scalar functions on a smooth  $n$ -dimensional real manifold  $N^n$ . Suppose first that a non-commutative deformation  $\star = \times + O(\hbar)$  of  $\times$  is still unital ( $f \star 1 = f = 1 \star f$ ) and associative,  $(f \star g) \star h = f \star (g \star h)$  for  $f, g, h \in C^\infty(N^n)[[\hbar]]$ . By taking  $3! = 6$  copies of the associativity equation for the star-product  $\star$ , we infer that the skew-symmetric part of the leading deformation term,  $\{f, g\}_\star := \frac{1}{\hbar}(f \star g - g \star f)|_{\hbar=0}$ , is a Poisson bracket.<sup>1</sup>

Now the other way round: can the multiplication  $\times$  on a Poisson manifold  $N^n$  be deformed using the bracket  $\{ , \}_\mathcal{P}$  such that the  $\mathbb{k}[[\hbar]]$ -linear star-product  $\star = \times + \hbar \{ , \}_\mathcal{P} + \bar{o}(\hbar)$  stays associative? Kontsevich proved [1] that on finite-dimensional affine<sup>2</sup> Poisson manifolds, this is always possible: from  $\{ , \}_\mathcal{P}$  one obtains the bi-differential terms  $B_k(\cdot, \cdot)$  at all powers of  $\hbar^{k \geq 0}$  in the formal series for  $\star$ . This associative unital  $\star$ -product was constructed in [1] using a pictorial language: the operators  $B_k = \sum_{\{\Gamma\}} w(\Gamma) \times B_k^\Gamma(\cdot, \cdot)$  are encoded by the weighted oriented graphs  $\Gamma$  with  $k+2$  vertices and  $2k$  edges but without tadpoles or multiple edges; in every such  $\Gamma$ , there are  $k$  internal vertices (each of them is a tail for two edges) and 2 sinks (no issued edges). The Poisson bracket  $\{ , \}_\mathcal{P}$  with coefficients  $\mathcal{P}^{ij}(\mathbf{u})$  at  $\mathbf{u} \in N^n$  provides the “building block”  $\Lambda = \leftarrow \begin{matrix} i \\ \text{Left} \end{matrix} \bullet \begin{matrix} j \\ \text{Right} \end{matrix} \rightarrow$  in which  $\sum_{i,j=1}^n$  is implicit and the vertex

\*Johann Bernoulli Institute for Mathematics & Computer Science, University of Groningen, P.O. Box 407, 9700 AK Groningen, The Netherlands. <sup>§</sup> Partially supported by JBI RUG project 103511 (Groningen).

<sup>1</sup>The left-hand side of the Jacobi identity  $\sum_{\mathcal{C}} \{\{f, g\}_\star, h\}_\star = 0$  is an obstruction to the associativity of the star-product: whenever the Jacobi identity is violated, one cannot have that  $(f \star g) \star h = f \star (g \star h)$ .

<sup>2</sup>On affine manifolds  $N^n$ , the only shape of coordinate changes is  $\tilde{\mathbf{u}} = A \cdot \mathbf{u} + \tilde{\mathbf{c}}$ . Yet no loss of generality occurs if the space  $N^n$  is the fibre in an affine bundle  $\pi$  of physical fields  $\{\mathbf{u} = \phi(\mathbf{x})\}$  over the space-time  $M^m \ni \mathbf{x}$ ; the Jacobians  $\partial \tilde{\mathbf{u}} / \partial \mathbf{u} = A(\mathbf{x})$  are then constant over  $N^n$ . (The arguments of  $\star$  are local functionals of sections,  $\phi \in \Gamma(\pi) \rightarrow \mathbb{k}$ ; the  $\star$ -product is marked by the variational Poisson brackets  $\{ , \}_\mathcal{P}$  on the jet space  $J^\infty(\pi)$ .) The deformation quantization from [1] is lifted to the gauge field set-up in [2].

contains  $\mathcal{P}^{ij}(\mathbf{u})$ . To indicate the ordering of indexes in  $\mathcal{P}^{ij} = -\mathcal{P}^{ji}$ , the out-going edges are ordered by Left  $\prec$  Right. The edges carry the derivatives  $\partial_i \equiv \partial/\partial u^i$  and  $\partial_j \equiv \partial/\partial u^j$ , respectively. Every such derivation acts on the content of the vertex at the arrowhead via the Leibniz rule (and it does so independently from the other in-coming arrows, if any).<sup>3</sup>

The weights<sup>4</sup>  $w(\Gamma) \in \mathbb{R}$  of such graphs  $\Gamma$  are given by the integrals over configuration spaces of  $k$  distinct points in the hyperbolic plane  $\mathbb{H}^2$  (e.g., in its upper half-plane model).<sup>5</sup>

The associativity postulate for  $\star$  yields the infinite system of quadratic algebraic equations for the weights  $w(\Gamma)$  of graphs.<sup>6</sup> Kontsevich shows [1] that the left-hand side  $\text{Jac}_{\mathcal{P}}(\cdot, \cdot, \cdot) := \sum_{\circlearrowleft} \{\{\cdot, \cdot\}_{\mathcal{P}}, \cdot\}_{\mathcal{P}}$  of the Jacobi identity for  $\{\cdot, \cdot\}_{\mathcal{P}}$  is the *only* obstruction to the balance  $\text{Assoc}(f, g, h) := (f \star g) \star h - f \star (g \star h) = 0$  at all powers  $\hbar^k$  of the deformation parameter at once.<sup>7</sup> The core question that we address in this note is how the mechanism  $\text{Assoc} = \diamond(\text{Poisson})$  works explicitly, making the star-product  $\star = \times + \hbar \{\cdot, \cdot\}_{\mathcal{P}} + \bar{o}(\hbar)$  associative by virtue of Jacobi identity for the Poisson bracket  $\{\cdot, \cdot\}_{\mathcal{P}}$ . Expanding the Kontsevich  $\star$ -product in  $\hbar$  up to  $\bar{o}(\hbar^3)$  and with respect to all the graphs  $\Gamma_i$  such that  $w(\Gamma_i) \neq 0$ , we obtain<sup>8</sup>

<sup>3</sup>For example,  $\{f, g\}_{\mathcal{P}}(\mathbf{u}) = f \xleftarrow{\text{Left } i} \bullet \xrightarrow{\text{Right } j} g = (f) \overleftarrow{\partial}_i|_{\mathbf{u}} \cdot \mathcal{P}^{ij}(\mathbf{u}) \cdot \overrightarrow{\partial}_j|_{\mathbf{u}}(g)$ , see (1) above.

<sup>4</sup>Willwacher and Felder (2010) conjecture that the weights can be *irrational* numbers for some graphs.

<sup>5</sup>The wedge factors within the integrand in the formula for  $w(\Gamma)$  are copies of the kernel of the singular linear integral operator  $(d \star d)^{-1}$  in the hyperbolic geometry of  $\mathbb{H}^2$ , see [3]. Cattaneo and Felder also showed that the  $\star$ -product of two functions  $f, g \in C^\infty(N^n \rightarrow \mathbb{C})$  amounts to the Feynman path integral calculation of the correlation function,  $(f \star g)(\mathbf{u}) = \int_{\mathcal{X}(\infty)=\mathbf{u}} D\mathcal{X} D\eta f(\mathcal{X}(0)) \times g(\mathcal{X}(1)) \times \exp(\frac{i}{\hbar} S(\mathcal{P}, [\mathcal{X}, \eta]))$ , in the Ikeda–Izawa topological open string model on a disk  $D \simeq \mathbb{H}^2$  with boundary  $\partial D \ni 0, 1, \infty$ ; here  $\mathcal{X}: D \rightarrow N^n$  and  $\eta: D \rightarrow T^*D \otimes \mathcal{X}^*(T^*N^n)$ . All details and further references are found in [3, 4]; still let us remember that within the Ikeda–Izawa model, the perturbative expansions in  $\hbar$  run, in particular, over the graphs with tadpoles (which must be regularized by hand) but at the same time, those path integral calculations reproduce only the weighted oriented graphs without “eyes” (e.g., as in  $\leftarrow \cdot \rightleftarrows \cdot \rightarrow$ , see Eq. (1) above). Because, to the best of our knowledge, the eye-containing graphs  $\Gamma_i$  such that  $w(\Gamma_i) \neq 0$  cannot all at once be eliminated from the star-product  $\star$  via gauge transformations of its arguments and of its output, see Remark 1 on p. 221 and [1], many graphs in the original construction of  $\star$  were not recovered in [3]. Hence there is an open problem to extend or modify the Ikeda–Izawa Poisson  $\sigma$ -model such that in the new set-up, the correlation functions would expand with respect to all the Kontsevich graphs  $\Gamma_i$  with  $w(\Gamma_i) \neq 0$ .

<sup>6</sup>That system solution is not claimed unique: one is provided by the Kontsevich integrals. Number-theoretic properties of those weights were explored by Kontsevich in the context of motives and by Willwacher–Felder and Garay–van Straten in the context of Riemann  $\zeta$ -function and Euler  $\Gamma$ -function, respectively.

<sup>7</sup>Ensuring the associativity  $\text{Assoc}(f, g, h) = 0$ , the tri-vector  $\text{Jac}_{\mathcal{P}}(\cdot, \cdot, \cdot)$  is not necessarily (indeed, far not always!) evaluated at the three arguments  $f, g, h$  of the associator for  $\star$ .

<sup>8</sup>Balancing the associativity of a star-product order-by-order up to  $\bar{o}(\hbar^3)$ , Penkava and Vanhaecke (1998) derived a set of weights for the  $(k+2)$ -vertex Kontsevich graphs *without* loops. Yet no loops are destroyed in either of the copies of  $\star$  when the composition  $\star \circ \star$  is taken; the associativity of loopless star-products is only a part of the full claim for  $\star$ . So, we integrate over the configuration spaces of  $k \leq 3$  points in  $\mathbb{H}^2$  for *all* the Kontsevich graphs (e.g., with loops).

$$\begin{aligned}
f \star g &= f \text{---} g + \frac{\hbar^1}{1!} \left( \begin{array}{c} \bullet \\ / \quad \backslash \\ f \quad g \end{array} \right) + \frac{\hbar^2}{2!} \left( \begin{array}{c} \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ f \quad g \end{array} \right) + \frac{\hbar^2}{3} \left( \begin{array}{c} \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ f \quad g \end{array} + \begin{array}{c} \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ f \quad g \end{array} \right) + \frac{\hbar^2}{6} \left( \begin{array}{c} \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ f \quad g \end{array} \text{ "eye" } \right) + \\
&+ \frac{\hbar^3}{6} \left( \begin{array}{c} \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ f \quad g \end{array} + \begin{array}{c} \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ f \quad g \end{array} \text{ "eye" } + \begin{array}{c} \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ f \quad g \end{array} \text{ "eye" } + \begin{array}{c} \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ f \quad g \end{array} + \begin{array}{c} \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ f \quad g \end{array} + \begin{array}{c} \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ f \quad g \end{array} + \begin{array}{c} \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ f \quad g \end{array} \right) + \\
&+ \frac{\hbar^3}{3} \left( \begin{array}{c} \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ f \quad g \end{array} + \begin{array}{c} \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ f \quad g \end{array} \right) + \frac{\hbar^3}{6} \left( \begin{array}{c} \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ f \quad g \end{array} \text{ "eye" } + \begin{array}{c} \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ f \quad g \end{array} \text{ "eye" } + \begin{array}{c} \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ f \quad g \end{array} + \begin{array}{c} \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ f \quad g \end{array} \right) + \bar{o}(\hbar^3). \quad (1)
\end{aligned}$$

In every composition  $\star \circ \star$  the sums of graphs act on sums of graphs by linearity; each incoming edge acts via the Leibniz rule (see above). The mechanism for  $\text{Assoc}(f, g, h)$  to vanish is two-step: first, the sums in  $\star \circ \star$  are reduced using the antisymmetry of the Poisson bi-vector  $\mathcal{P}$ . The output is then reduced modulo the (consequences of) Jacobi identity,<sup>9</sup>

$$\text{Jac}_{\mathcal{P}}(f, g, h) = \begin{array}{c} \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ f \quad g \quad h \end{array} - \begin{array}{c} \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ f \quad g \quad h \end{array} \text{ "eye" } - \begin{array}{c} \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ f \quad g \quad h \end{array} = 0. \quad (2)$$

For  $\star$  given by (1), the associator contains 6 terms at  $\hbar$ , 38 terms  $\sim \hbar^2$ , and 218 terms  $\sim \hbar^3$ . After the use of  $\mathcal{P}^{ij} = -\mathcal{P}^{ji}$ , we infer that  $\text{Assoc}(f, g, h)$  starts at  $\hbar^2$  with  $2/3$  times (2). Next, there are 39 terms at  $\hbar^3$ ; we now examine how their sum  $A$  vanishes by virtue of (2) and its differential consequences.<sup>10</sup> Of them, three which are the easiest to recognize are<sup>11</sup>

$$\frac{2}{3} \mathcal{P}^{ij} \text{Jac}_{\mathcal{P}}(\partial_i f, \partial_j g, h) = \frac{2}{3} \cdot \left( \begin{array}{c} \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ f \quad g \quad h \end{array} \text{ "eye" } - \begin{array}{c} \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ f \quad g \quad h \end{array} \text{ "eye" } - \begin{array}{c} \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ f \quad g \quad h \end{array} \right) = 0, \quad (3)$$

as well as  $\frac{2}{3} \mathcal{P}^{ij} \text{Jac}_{\mathcal{P}}(f, \partial_i g, \partial_j h) = 0$  and  $\frac{2}{3} \mathcal{P}^{ij} \text{Jac}_{\mathcal{P}}(\partial_i f, g, \partial_j h) = 0$ . So, there remain 30 terms which vanish via (2) in a way more intricate than (3). It is clear that

$$S_f := \mathcal{P}^{ij} \partial_j \text{Jac}_{\mathcal{P}}(\partial_i f, g, h) = \begin{array}{c} \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ f \quad g \quad h \end{array} \text{ "eye" } - \begin{array}{c} \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ f \quad g \quad h \end{array} \text{ "eye" } - \begin{array}{c} \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ \bullet \\ / \quad \backslash \\ f \quad g \quad h \end{array} \text{ "eye" } = 0. \quad (4)$$

Working out the Leibniz rule in (4), we collect the graphs according to the number of derivatives falling on each of  $(f, g, h)$ . The edge  $-j \rightarrow$  provides the differential orders<sup>12</sup>  $(3, 1, 1)$ ,  $(2, 2, 1)$ ,  $(2, 1, 2)$ , and  $(2, 1, 1)$  twice. Likewise, we see  $(1, 1, 1)$  in (2) and  $(2, 2, 1)$  in (3).

**Lemma.** A tri-differential operator  $\sum_{|I|, |J|, |K| \geq 0} c^{IJK} \partial_I \otimes \partial_J \otimes \partial_K$  vanishes identically iff all its coefficients vanish:  $c^{IJK} = 0$  for every triple  $(I, J, K)$  of multi-indices; here  $\partial_L = \partial_1^{\alpha_1} \circ \dots \circ \partial_n^{\alpha_n}$  for a multi-index  $L = (\alpha_1, \dots, \alpha_n)$ . Moreover, the sums  $\sum_{|I|=i, |J|=j, |K|=k} c^{IJK} \partial_I \otimes \partial_J \otimes \partial_K$  are then zero for all  $(i, j, k)$ ; in a vanishing sum  $X$  of graphs, we denote by  $X_{ijk}$  its vanishing restriction<sup>13</sup> to a fixed differential order  $(i, j, k)$ .

<sup>9</sup>By default, the  $L \prec R$  edge ordering equals the left  $\prec$  right direction in which edges start on these pages.

<sup>10</sup>Within the variational geometry of Poisson field models (cf. [2]), a tiny leak of the associativity for  $\star$  may occur, if it does at all, only at orders  $\hbar^{\geq 4}$  because at most *one* arrow falls on  $\text{Jac}_{\mathcal{P}}(\cdot, \cdot, \cdot)$  in the balance  $\text{Assoc}(f, g, h) = \bar{o}(\hbar^3)$ . But unlike the always vanishing first variation of a homologically trivial functional  $\text{Jac}_{\mathcal{P}}(\cdot, \cdot, \cdot) \cong 0$ , its higher-order variations can be nonzero.

<sup>11</sup>We use the Einstein summation convention; a sum over all indices is also implicit in the graph notation.

<sup>12</sup>In fact, the double edge to  $f$  contributes with zero at  $(3, 1, 1)$  due to the skew-symmetry  $\mathcal{P}^{ij} = -\mathcal{P}^{ji}$ .

<sup>13</sup>For example, relation (3) is the consequence of (4) at order  $(2, 2, 1)$ ; restriction of (4) to  $(2, 1, 1)$  yields



The Poisson bi-vector components  $\mathcal{P}^{ij}$  can also serve as arguments of the Jacobiator:<sup>14</sup>

$$I_f := \partial_j(\text{Jac}_{\mathcal{P}}(\mathcal{P}^{ij}, g, h)) \partial_i f = \begin{array}{c} \text{graph 1} \\ - \text{graph 2} \\ - \text{graph 3} \end{array} = 0.$$

Likewise,  $I_g := \partial_i(\text{Jac}_{\mathcal{P}}(f, \mathcal{P}^{ij}, h)) \partial_j g = 0$  and  $I_h := \partial_i(\text{Jac}_{\mathcal{P}}(f, g, \mathcal{P}^{ij})) \partial_j h = 0$ . It is the expansion of  $I_f, I_g, I_h$  via the Leibniz rule that produces the graphs with “eyes”. It also yields an order  $(1, 1, 1)$  differential operator on  $(f, g, h)$  which cannot be obtained from (4).

**Claim.** The sum  $\mathbf{A}$  of 39 terms at  $\hbar^3$  in  $\text{Assoc}(f, g, h)$  vanishes by virtue of restriction of  $S_f, S_g, S_h$  and  $I_f, I_g, I_h$  to the orders  $(i, j, k)$  that are present in  $\mathbf{A}$ . Indeed, we have<sup>15</sup>  $\mathbf{A}_{221} \stackrel{[3]}{=} \frac{2}{3}(S_f)_{221}$ ,  $\mathbf{A}_{122} \stackrel{[3]}{=} \frac{2}{3}(S_g)_{122}$ , and  $\mathbf{A}_{212} \stackrel{[3]}{=} -\frac{2}{3}(S_h)_{212}$ , see (3). Finally, we deduce that  $\mathbf{A}_{111} \stackrel{[8]}{=} \frac{1}{6}(I_f - I_h)_{111}$ ,  $\mathbf{A}_{112} \stackrel{[9]}{=} (\frac{1}{6}I_f + \frac{1}{6}I_g - \frac{1}{3}S_h)_{112}$ ,  $\mathbf{A}_{121} \stackrel{[4]}{=} \frac{1}{3}(I_f - I_h)_{121}$ , and  $\mathbf{A}_{211} \stackrel{[9]}{=} (\frac{1}{3}S_f - \frac{1}{6}I_g - \frac{1}{6}I_h)_{211}$ . The total number of terms which we thus eliminate equals  $(3 + 3 + 3) + 8 + 9 + 4 + 9 = 39$ .  $\square$

**Remark 1.** The deformation quantization is a gauge theory: each argument  $\bullet$  of  $\star$  marks its gauge class  $[\bullet]$  under the linear maps  $\mathfrak{t}: \bullet \mapsto [\bullet] = \bullet + \hbar(I^\emptyset \partial_i \partial_j (\mathcal{P}^{ij})^{\equiv 0} \times \bullet + I^\circ \partial_i \mathcal{P}^{ij} \partial_j(\bullet)) +$

$$\hbar^2 I^{(0)} \begin{array}{c} \text{graph} \\ \bullet \end{array} + \hbar^3 \left[ \begin{array}{c} (1) \\ I \end{array} \begin{array}{c} \text{graph} \\ \bullet \end{array} \stackrel{\equiv 0}{=} \begin{array}{c} (2) \\ I \end{array} \begin{array}{c} \text{graph} \\ \bullet \end{array} + \begin{array}{c} (3) \\ I \end{array} \begin{array}{c} \text{graph} \\ \bullet \end{array} \stackrel{\equiv 0}{=} \begin{array}{c} (4) \\ I \end{array} \begin{array}{c} \text{graph} \\ \bullet \end{array} + \begin{array}{c} (5) \\ I \end{array} \begin{array}{c} \text{graph} \\ \bullet \end{array} \stackrel{\equiv 0}{=} \begin{array}{c} (6) \\ I \end{array} \begin{array}{c} \text{graph} \\ \bullet \end{array} + \begin{array}{c} (7) \\ I \end{array} \begin{array}{c} \text{graph} \\ \bullet \end{array} \right]$$

$+ \bar{o}(\hbar^3)$ , where the constants  $I \in \mathbb{k}$  can be arbitrary<sup>16</sup> and  $\mathfrak{t}$  is formally invertible over  $\mathbb{k}[[\hbar]]$ .

In turn, the star-products are gauged<sup>17</sup> by using  $\mathfrak{t}: f \star' g := \mathfrak{t}^{-1}(\mathfrak{t}(f) \star \mathfrak{t}(g))$ . This degree of freedom extends the uniqueness problem for Kontsevich’s solution  $\star$  of  $\text{Assoc}(f, g, h) = 0$ . Namely, not the exact balance of power series but an equivalence  $[=]$  of gauge classes (up to unrelated transformations at all steps) can be sought in  $[[f] \star [g]] \star [h] [=] [f] \star ([g] \star [h])$ .

**Remark 2.** Each graph  $\Gamma$  in (1) encodes the polydifferential operator of scalar arguments in a coordinate-free way. The Jacobians  $\partial \mathbf{u} / \partial \tilde{\mathbf{u}}$  of affine mappings appear on the edges but then they join the content  $\hbar \mathcal{P}^{ij}$  of internal vertices at the arrowtails,<sup>18</sup> forming  $\tilde{\mathcal{P}}^{\alpha\beta}$  from  $\mathcal{P}^{ij}$ . Independent from  $\mathbf{u} \in N^n$ , these Jacobians stay invisible to all in-coming arrows (if any). So, the operator given by a graph  $\Gamma$  with  $\hbar \mathcal{P}(\mathbf{u})$  in its vertices is equal to the one for  $\hbar \tilde{\mathcal{P}}(\tilde{\mathbf{u}}(\mathbf{u}))$  there. This reasoning works for the variational Poisson brackets  $\{, \}_{\mathcal{P}}$  on  $J^\infty(\pi)$  for affine bundles  $\pi$  with fibre  $N^n$  over points  $\mathbf{x} \in M^m$ , see [2]. The graphs  $\Gamma$  then yield local variational polydifferential operators yet the pictorial language of [1] is the same.<sup>19</sup>

$$\left( \begin{array}{c} \text{graph 1} \\ + \\ \text{graph 2} \end{array} \right) - \left( \begin{array}{c} \text{graph 3} \\ + \\ \text{graph 4} \end{array} \right) - \left( \begin{array}{c} \text{graph 5} \\ + \\ \text{graph 6} \end{array} \right) = 0.$$

Similarly, we have  $S_g := \mathcal{P}^{ij} \partial_j \text{Jac}_{\mathcal{P}}(f, \partial_i g, h) = 0$  and  $S_h := \mathcal{P}^{ij} \partial_j \text{Jac}_{\mathcal{P}}(f, g, \partial_i h) = 0$ .

<sup>14</sup>The three tadpoles produce  $\text{Jac}_{\mathcal{P}}(\partial_i \mathcal{P}^{ij}, g, h) \partial_j f = 0$ , which plays its rôle in  $\mathbf{A}_{111}$  (see the claim below).

<sup>15</sup>By using the symbol  $\stackrel{[m]}{=}$  we indicate the number  $m$  of terms that are eliminated at each step.

<sup>16</sup>The view [3] on  $\star$ -products as  $\hbar$ -expansions of path integrals shows that the graphs  $\Gamma_i$  in (1) are genuine Feynman diagrams for the channel marked by  $\mathcal{P}$ . The weights  $w(\Gamma_i)$  integrate over the energy of each intermediate vertex. Quite naturally, a particle  $\bullet$  shares its energy-mass with the interaction carriers  $\mathcal{P}$  as it gets coated by them. But no object  $\bullet$  can spend more energy on growing its gauge tail than the amount it actually has; hence every set  $[\bullet]$  is bounded in the space of parameters  $\mathbf{I}$ .

<sup>17</sup>For example, the loop graph at  $\hbar^2/6$  in (1) is gauged out by  $\mathfrak{t}(\bullet) = \bullet + \frac{\hbar^2}{12} \begin{array}{c} \text{loop} \\ \bullet \end{array}$ , see [1] for further details.

<sup>18</sup>E.g.,  $\partial'_\alpha \tilde{\mathcal{P}}^{\alpha\beta} \Big|_{\tilde{\mathbf{u}}} \partial'_\beta = \partial_i \frac{\partial u^i}{\partial \tilde{u}^\alpha} \tilde{\mathcal{P}}^{\alpha\beta} \Big|_{\tilde{\mathbf{u}}(\mathbf{u})} \frac{\partial u^j}{\partial \tilde{u}^\beta} \partial'_j = \partial_i \cdot \mathcal{P}^{ij} \Big|_{\mathbf{u}} \cdot \partial'_j$  so that  $\{f, g\}_{\mathcal{P}(\mathbf{u})} = \{f, g\}_{\tilde{\mathcal{P}}(\tilde{\mathbf{u}}(\mathbf{u}))}(\tilde{\mathbf{u}}(\mathbf{u}))$ .

<sup>19</sup>A sought-for extension of the Ikeda–Izawa topological open string geometry – namely, its lift from the Poisson manifolds  $(N^n, \{, \}_{\mathcal{P}})$  in [3, 4] to the variational set-up  $(J^\infty(\pi), \{, \}_{\mathcal{P}})$  of jet spaces in [2] – is a mechanism to quantize Poisson field models. This will be the object of another paper.

**Acknowledgements.** A. V. K. thanks the organizers of international workshop SQS'15 (August 3–8, 2015 at JINR Dubna, Russia) for stimulating discussions and partial financial support.

## References

- [1] Kontsevich M. Deformation quantization of Poisson manifolds. I // *Lett. Math. Phys.* 2003. V. 66, n. 3. P. 157–216. [arXiv:q-alg/9709040](#)
- [2] Kiselev A. V. Deformation approach to quantisation of field models. Preprint IHÉS/M/15/13. Bures-sur-Yvette: IHÉS, 2015. P. 1–37.
- [3] Cattaneo A. S., Felder G. A path integral approach to the Kontsevich quantization formula // *Comm. Math. Phys.* 2000. V. 212, n. 3. P. 591–611. [arXiv:q-alg/9902090](#)
- [4] Ikeda N. Two-dimensional gravity and nonlinear gauge theory // *Ann. Phys.* 1994. V. 235, n. 2. P. 435–464. [arXiv:hep-th/9312059](#)

# Chapter 11

## The expansion $\star \bmod \bar{o}(\hbar^4)$ and computer-assisted proof schemes in the Kontsevich deformation quantization

This chapter is based on the peer-reviewed journal publication *R. Buring and A.V. Kiselev, Experimental Math.* **31**(3) or (4), 54 p., 2022 (in press). (doi:10.1080/10586458.2019.168046) (Preprint arXiv:1702.00681 [math.CO] – 77 p.) Appendix A.1 in that paper follows the talk given by the dissertant at the workshop *Symmetries of Discrete Systems and Processes III* (3–7 August 2015, Děčín, Czech Republic); the entire work has been presented by the dissertant at seminars and conferences multiple times (Oxford, Utrecht, Larnaca, Bedlewo, etc.).

*Commentary.* The theory in this chapter is used in Chapters 1, 3, and 9 of Part I. The theory of this chapter is explained in more detail in the next chapter. The graph encoding of Kontsevich's  $\star \bmod \bar{o}(\hbar^4)$  is contained in Appendix B.1.

# THE EXPANSION $\star$ MOD $\bar{o}(\hbar^4)$ AND COMPUTER-ASSISTED PROOF SCHEMES IN THE KONTSEVICH DEFORMATION QUANTIZATION

R. BURING\* AND A. V. KISELEV<sup>§</sup>

ABSTRACT. The Kontsevich deformation quantization combines Poisson dynamics, noncommutative geometry, number theory, and calculus of oriented graphs. To manage the algebra and differential calculus of series of weighted graphs, we present software modules: these allow generating the Kontsevich graphs, expanding the noncommutative  $\star$ -product by using *a priori* undetermined coefficients, and deriving linear relations between the weights of graphs. Throughout this text we illustrate the assembly of the Kontsevich  $\star$ -product up to order 4 in the deformation parameter  $\hbar$ . Already at this stage, the  $\star$ -product involves hundreds of graphs; expressing all their coefficients via 149 weights of basic graphs (of which 67 weights are now known exactly), we express the remaining 82 weights in terms of only 10 parameters (more specifically, in terms of only 6 parameters modulo gauge-equivalence). Finally, we outline a scheme for computer-assisted proof of the associativity, modulo  $\bar{o}(\hbar^4)$ , for the newly built  $\star$ -product expansion.

## CONTENTS

Introduction	225
1. Weighted graphs	229
2. The Kontsevich $\star$ -product	237
3. Associativity of the Kontsevich $\star$ -product	246
4. Discussion	259
Conclusion	267
References	273
Appendix A. Approximations and conjectured values of weight integrals	276
Appendix B. C++ classes and methods	i
Appendix C. Encoding of the entire $\star$ -product modulo $\bar{o}(\hbar^4)$	v
Appendix D. Encoding of the associator of the $\star$ -product modulo $\bar{o}(\hbar^4)$	x
Appendix E. Gauge transformation that removes 4 master-parameters out of 10xvii	

---

*Date:* 14 March 2018; in original form 20 December 2017, in final form 5 October 2019.

2010 *Mathematics Subject Classification.* 05C22, 53D55, 68R10, also 05C31, 16Z05, 53D17, 81R60, 81Q30.

*Key words and phrases.* Associative algebra, noncommutative geometry, deformation quantization, Kontsevich graph complex, computer-assisted proof scheme, software module, template library.

\**Address:* Institut für Mathematik, Johannes Gutenberg–Universität, Staudingerweg 9, D-55128 Mainz, Germany. \**E-mail:* [rburing@uni-mainz.de](mailto:rburing@uni-mainz.de).

<sup>§</sup>*Address:* Johann Bernoulli Institute for Mathematics and Computer Science, University of Groningen, P.O. Box 407, 9700 AK Groningen, The Netherlands. <sup>§</sup>*E-mail:* [A.V.Kiselev@rug.nl](mailto:A.V.Kiselev@rug.nl).

**Introduction.** On every finite-dimensional affine (i.e. piecewise-linear) manifold  $N^n$ , the Kontsevich star-product  $\star$  [32] is an associative but not necessarily commutative deformation of the usual product  $\times$  in the algebra of functions  $C^\infty(N^n)$  towards a given Poisson bracket  $\{\cdot, \cdot\}_{\mathcal{P}}$  on  $N^n$  (see also [19, 4, 5]). Specifically, whenever  $\star = \times + \hbar \{\cdot, \cdot\}_{\mathcal{P}} + \bar{o}(\hbar)$  is an infinitesimal deformation, it can always be completed to an associative star-product  $\star = \times + \hbar \{\cdot, \cdot\}_{\mathcal{P}} + \sum_{k \geq 2} \hbar^k B_k(\cdot, \cdot)$  in the space of formal power series  $C^\infty(N^n)[[\hbar]]$ ; this was proven in [32]. An explicit calculation of the bi-linear bi-differential terms  $B_k(\cdot, \cdot)$  at high orders  $\hbar^k$  is a computationally hard problem. In this paper we reach the order  $k = 4$  in expansion of  $\star$  by using software modules for the Kontsevich graph calculus, which we presently discuss.

Convenient in practice, the idea from [32] (see also [28, 29, 31]) is to draw every derivation  $\partial_i \equiv \partial/\partial x^i$  (with respect to a local coordinate  $x^i$  on a chart in the Poisson manifold  $N^n$  at hand) as decorated edge  $\xrightarrow{i}$ , so that large differential expressions become oriented graphs. For example, the Poisson bracket  $\{f, g\}_{\mathcal{P}}(\mathbf{x}) = \sum_{i,j=1}^n (f) \overleftarrow{\partial}_i|_{\mathbf{x}} \cdot \mathcal{P}^{ij}(\mathbf{x}) \cdot \overrightarrow{\partial}_j|_{\mathbf{x}}(g)$  of two functions  $f, g \in C^\infty(N^n)$  is depicted by the graph  $(f) \xleftarrow{i} \mathcal{P}^{ij} \xrightarrow{j} (g)$ ; here  $\mathcal{P}^{ij}$  is the skew-symmetric matrix of Poisson bracket coefficients and the summation over  $i, j$  running from 1 to the dimension  $n$  of  $N^n$  is implicit. In these terms, the known – from [12] – expansion of the Kontsevich star-product<sup>1</sup> looks as follows:<sup>2</sup>

$$\begin{aligned}
 f \star g &= f \cdot g + \frac{\hbar^1}{1!} \text{graph}_1 + \frac{\hbar^2}{2!} \text{graph}_2 + \frac{\hbar^2}{3} \left( \text{graph}_3 + \text{graph}_4 \right) + \frac{\hbar^2}{6} \text{graph}_5 + \\
 &+ \frac{\hbar^3}{6} \left( \text{graph}_6 + \text{graph}_7 + \text{graph}_8 + \text{graph}_9 + \text{graph}_{10} + \text{graph}_{11} + \text{graph}_{12} \right) + \\
 &+ \frac{\hbar^3}{3} \left( \text{graph}_{13} + \text{graph}_{14} \right) + \frac{\hbar^3}{6} \left( \text{graph}_{15} + \text{graph}_{16} + \text{graph}_{17} + \text{graph}_{18} \right) + \bar{o}(\hbar^3). \quad (1)
 \end{aligned}$$

By construction, every oriented edge carries its own index and every *internal* vertex (not containing the arguments  $f$  or  $g$ ) is inhabited by a copy of the coefficient matrix  $\mathcal{P} = (\mathcal{P}^{ij})$  of the Poisson bracket  $\{\cdot, \cdot\}_{\mathcal{P}}$ . This means that expansion (1) encodes the analytic formula

$$\begin{aligned}
 f \star g &= f \times g + \hbar \mathcal{P}^{ij} \partial_i f \partial_j g + \hbar^2 \left( \frac{1}{2} \mathcal{P}^{ij} \mathcal{P}^{k\ell} \partial_k \partial_i f \partial_\ell \partial_j g + \frac{1}{3} \partial_\ell \mathcal{P}^{ij} \mathcal{P}^{k\ell} \partial_k \partial_i f \partial_j g \right. \\
 &\quad \left. - \frac{1}{3} \partial_\ell \mathcal{P}^{ij} \mathcal{P}^{k\ell} \partial_i f \partial_k \partial_j g - \frac{1}{6} \partial_\ell \mathcal{P}^{ij} \partial_j \mathcal{P}^{k\ell} \partial_i f \partial_k g \right) + \hbar^3 \left( \frac{1}{6} \mathcal{P}^{ij} \mathcal{P}^{k\ell} \mathcal{P}^{mn} \partial_m \partial_k \partial_i f \partial_n \partial_\ell \partial_j g \right.
 \end{aligned}$$

<sup>1</sup>We recall that the expansion  $\star \bmod \bar{o}(\hbar^2)$  in [32] was gauge-equivalent to the genuine one so that the two-cycle graph at  $\hbar^2/6$  in the first line of above formula (1) was gauged out: see Example 25 on p. 246 where we explain how this is done.

<sup>2</sup>The indication  $L$  and  $R$  for Left  $\prec$  Right, respectively, matches the indices – which the pairs of edges carry – with the ordering of indices in the coefficients of the Poisson structure contained in the arrowtail vertex. Note that exactly *two* edges are issued from every internal vertex in every graph in formula (1); not everywhere displayed in (1), the ordering  $L \prec R$  in each term is determined from same object’s expansion (2).

$$\begin{aligned}
& -\frac{1}{6}\partial_m\partial_\ell\mathcal{P}^{ij}\partial_n\partial_j\mathcal{P}^{kl}\mathcal{P}^{mn}\partial_i f\partial_k g - \frac{1}{6}\mathcal{P}^{ij}\partial_n\mathcal{P}^{kl}\partial_\ell\mathcal{P}^{mn}\partial_k\partial_i f\partial_m\partial_j g \\
& \quad - \frac{1}{6}\partial_m\partial_\ell\mathcal{P}^{ij}\partial_n\mathcal{P}^{kl}\mathcal{P}^{mn}\partial_k\partial_i f\partial_j g - \frac{1}{6}\partial_m\partial_\ell\mathcal{P}^{ij}\partial_n\mathcal{P}^{kl}\mathcal{P}^{mn}\partial_i f\partial_k\partial_j g \\
& \quad + \frac{1}{6}\partial_n\partial_\ell\mathcal{P}^{ij}\mathcal{P}^{kl}\mathcal{P}^{mn}\partial_m\partial_k\partial_i f\partial_j g + \frac{1}{6}\partial_n\partial_\ell\mathcal{P}^{ij}\mathcal{P}^{kl}\mathcal{P}^{mn}\partial_i f\partial_m\partial_k\partial_j g \\
& \quad + \frac{1}{3}\partial_n\mathcal{P}^{ij}\mathcal{P}^{kl}\mathcal{P}^{mn}\partial_m\partial_k\partial_i f\partial_\ell\partial_j g - \frac{1}{3}\partial_n\mathcal{P}^{ij}\mathcal{P}^{kl}\mathcal{P}^{mn}\partial_k\partial_i f\partial_m\partial_\ell\partial_j g \\
& \quad - \frac{1}{6}\partial_\ell\mathcal{P}^{ij}\partial_n\partial_j\mathcal{P}^{kl}\mathcal{P}^{mn}\partial_m\partial_i f\partial_k g + \frac{1}{6}\partial_n\partial_\ell\mathcal{P}^{ij}\partial_j\mathcal{P}^{kl}\mathcal{P}^{mn}\partial_i f\partial_m\partial_k g \\
& \quad - \frac{1}{6}\partial_n\mathcal{P}^{ij}\mathcal{P}^{kl}\partial_\ell\mathcal{P}^{mn}\partial_k\partial_i f\partial_m\partial_j g - \frac{1}{6}\partial_\ell\mathcal{P}^{ij}\partial_n\mathcal{P}^{kl}\mathcal{P}^{mn}\partial_k\partial_i f\partial_m\partial_j g) + \bar{o}(\hbar^3). \quad (2)
\end{aligned}$$

We now see that the language of Kontsevich graphs is more intuitive and easier to percept than writing formulae. The calculation of the associator  $\text{Assoc}_*(f, g, h) = (f \star g) \star h - f \star (g \star h)$  can also be done in a pictorial way (see section 2.4 on p. 243). The coefficients of graphs at  $\hbar^k$  in a star-product expansion are given by the Kontsevich integrals over the configuration spaces of  $k$  distinct points in the Lobachevsky plane  $\mathbb{H}$ , see [32] and [15]. Although proven to exist, such weights of graphs are very hard to obtain in practice.<sup>3</sup> Much research has been done on deriving helpful relations between the weights in order to facilitate their calculation [17, 36, 21, 18, 6]. In Example 26 on p. 250 we explain how expansion (1) modulo  $\bar{o}(\hbar^3)$  was obtained in [12]. The techniques which were then sufficient are no longer enough to build the Kontsevich  $\star$ -product beyond the order  $\hbar^3$ ; clearly, extra mathematical concepts and computational tools must be developed. In this paper we present the software in which several known relations between the Kontsevich graph weights are taken into account; we express the weights of all graphs at  $\hbar^4$  in terms of 10 master-parameters. (To be more precise, the ten master-parameters are reduced to just 6 by taking the quotient over certain four degrees of gauge freedom in the associative star-product expansions mod  $\bar{o}(\hbar^4)$ .) This paper is aimed to provide much more than a reference to computer programs: it also contains a synopsis of the proofs for the ideas in the construction, as well as an explanation of the parts which require computer implementation. Now, the values of Kontsevich graph weights and, with more input from the work in progress [2, 34], *all* the values which specify  $\star \bmod \bar{o}(\hbar^4)$  are the main result of this paper. These weights (as well as the ones of higher-order expansion terms) are subject to conjectures and open problems (see [17, 2]).

This paper contains four chapters. In chapter 1 we introduce the software to encode and generate the Kontsevich graphs and operate with series of such graphs. In particular, the coefficients of graphs in series can be undetermined variables. The series are then reduced modulo the skew-symmetry of graphs (under the swapping of Left  $\rightleftharpoons$  Right in their construction). Thirdly, a series can be evaluated at a given Poisson structure: that is, a copy of the bracket is placed at every internal vertex.

Chapter 2 is devoted to the construction of Kontsevich's  $\star$ -product: containing a given Poisson structure in its leading deformation term, this bi-linear operation is not necessarily commutative but it is required to be associative; hence the coefficients of a power series for  $\star$  must be specified. For example, at order  $k = 4$  of the deformation parameter  $\hbar$  there are 149 parameters to be found. (The actual number of graphs at  $\hbar^4$  is much greater; we here count the "basic" graphs only.) We review a number of methods

<sup>3</sup>In fact, there are many other admissible graphs, not shown in (1), in which every internal vertex is a tail for two oriented edges, but the weights of those graphs are found to be zero.

to obtain the weights of Kontsevich graphs; the spectrum of techniques employed ranges from complex analysis and direct numeric integration [14] to finding linear relations between such weights by using abstract geometric reasonings. The associativity of Kontsevich's  $\star$ -product is a major source of relations between the graph weights; at  $\hbar^4$  such relations are *linear* because everything is known about the weights up to order three. We obtain these relations at order four in chapter 3 and we solve that system of linear algebraic equations for 149 unknowns. The solution is expressed in terms of only 10 master-parameters, see formula (11) on pp. 267–272.<sup>4</sup> It is readily seen that the final formula (13) on pp. 280–284, in which the ten parameters are assigned specific real values so that *all* the coefficients in  $\star \bmod \bar{o}(\hbar^4)$  are the values of Kontsevich's integrals, is the genuine formula of the Kontsevich  $\star$ -product. Indeed, our formula  $\star \bmod \bar{o}(\hbar^4)$  is obtained according to the Proof scheme for Theorem 9 on p. 251 below. We compute a big system of equations which is satisfied by Kontsevich's formula: it is constrained by Lemmas 1–5 (basic identities), Proposition 7 (cyclic weight relations), Methods 1–3 (associativity), and vanishing of some integrands (cf. Appendix A.1). Having solved this system, we incorporate external input [34, 2] in the form of direct calculation of Kontsevich's integrals.

The algebraic system constructed in section 3.1 was obtained by restricting the associativity for  $\star$  to (a class of) specific Poisson structures. We want however to prove that for the newly found collection of graph weights, the  $\star$ -product is associative for *every* Poisson structure on *all* finite-dimensional affine manifolds. For that, in section 3.2 we design a computer-assisted proof scheme that is independent of the bracket (and of a manifold at hand). Specifically, in Theorem 12 on p. 256 we reveal how the associator for Kontsevich's  $\star$ -product, taken modulo  $\bar{o}(\hbar^4)$ , is factorised via the Jacobiator  $\text{Jac}(\mathcal{P})$  or via its differential consequences that all vanish identically for Poisson structures  $\mathcal{P}$  on the manifolds  $N^n$ . We discover in particular that such factorisation,

$$\text{Assoc}_\star(f, g, h) = \diamond(\mathcal{P}, \text{Jac}(\mathcal{P}), \text{Jac}(\mathcal{P})) \bmod \bar{o}(\hbar^4),$$

is quadratic and has differential order two with respect to the Jacobiator. For all Poisson brackets  $\{\cdot, \cdot\}_{\mathcal{P}}$  on finite-dimensional affine manifolds  $N^n$  our ten-parameter expression of the  $\star$ -product does agree up to  $\bar{o}(\hbar^4)$  with previously known results about the values of Kontsevich graph weights at some fixed values of the ten master-parameters and about the linear relations between those weights at all values of the master-parameters.<sup>5</sup> In an extensive Discussion on pp. 259–266, we compare (and, again, verify) our result with other work, namely by Gutt et al [1], Ben Amar [6], Kathotia [21], Willwacher [38], and Penkava–Vanhaecke [35]. Further discussion of our result is contained in section

<sup>4</sup>The values of all these ten master-parameters have recently been claimed by Panzer and Pym [34] as a result of implementation of another technique to calculate the Kontsevich weights: see Table 4 on p. 280 in Appendix A.2. In particular, the values which we conjecture in Table 3 fully agree with the exact values suggested in [34]. Based on this external input, the expansion of the Kontsevich  $\star$ -product becomes (13) on pp. 280–284.

<sup>5</sup>From Theorem 12 we also assert that the associativity of Kontsevich's  $\star$ -product does not carry on but it can leak at orders  $\hbar^{\geq 4}$  of the deformation parameter, should one enlarge the construction of  $\star$  to an affine bundle set-up of  $N^n$ -valued fields over a given affine manifold  $M^m$  and of variational Poisson brackets  $\{\cdot, \cdot\}_{\mathcal{P}}$  for local functionals  $F, G, H: C^\infty(M^m \rightarrow N^n) \rightarrow \mathbb{k}$ , see [23, 24, 25, 26] and [27].

6.2 on p. 61 in the most recent preprint [2]. The following list of insights is gained as a byproduct of our approach:

- Relations between the Kontsevich graph weights can be obtained by viewing the  $\star$ -product associator  $\text{Assoc}_\star(\mathcal{P})(f, g, h) = 0$  for a Poisson structure  $\mathcal{P} = \mathcal{P}(\psi)$  as a polydifferential operator on  $f, g, h, \psi$  (see §3.1, Method 3). This new technique (effective by virtue of computer implementation) yields many new relations. In particular:
- All the weights of graphs at order 3 in the  $\star$ -product are uniquely determined (see Example 26 on p. 250) by the associativity equation up to order 4 for Poisson structure (10) on  $\mathbb{R}^3$ , the elementary Lemmas 1–5, and the cyclic weight relations up to order 4. This is one instance of:
- Linear relations between *only* weights of graphs at order  $n$  can be obtained (in an effective, predictable way) from the associativity equation at orders greater than  $n$  (see Remark 11 on p. 247). This is explained using the decomposition of a polydifferential operator into homogeneous components and the notion of “prime” Kontsevich graphs.
- The proof of associativity of the  $\star$ -product at order 4 must involve a *second-order* differential consequence of the Jacobi identity (see the second part of Theorem 12 on p. 256). In particular, a naive jet space extension of Kontsevich’s star product, where derivatives are replaced by variations, is in general not associative at order 4 (see Corollary 13 on p. 257).
- The mechanism of vanishing via differential consequences of the Jacobi identity may start working for the  $\star$ -product expansion itself (see Theorem 15 on p. 259). In fact, the order 4 is the first where this may happen. (It could have happened at order 3, if the weights of graphs were different.)
- So far, from the work of Willwacher (see [38]) it was known that graphs with two-cycles, or loops, cannot be eliminated all at once from the star-product by using gauge transformations. At  $\hbar^2$ , the only such graph can be removed indeed (see Example 24); at  $\hbar^3$  there are four loopful graphs out of 13 graphs with nonzero coefficients. We discovered a totally unexpected fact (see p. 284 below): at  $\hbar^4$ , the graphs with loops are dominant: 138 out of 247.

---

The software implementation [9] consists of a C++ library and a set of command-line programs. The latter are specified in what follows; a full list of new C++ subroutines and their call syntax is contained in Appendix B. Whenever a command-line program refers to just one particular function in C++, we indicate that in the text. The current text refers to version 0.66 of the software. This and future versions are available from

[https://github.com/rburing/kontsevich\\_graph\\_series-cpp](https://github.com/rburing/kontsevich_graph_series-cpp)

All data files constructed and referred to in this article (in plain text format, which can be appreciated independently of the software) are available in the `data` subdirectory:

[https://github.com/rburing/kontsevich\\_graph\\_series-cpp/tree/master/data](https://github.com/rburing/kontsevich_graph_series-cpp/tree/master/data)

© The copyright for all newly designed software modules which are presented in this paper is retained by R. Buring; provisions of the MIT free software license apply.



1. WEIGHTED GRAPHS

In this section we introduce the software to operate with series of oriented graphs.

1.1. **Normal forms of graphs and their machine-readable format.** As it was explained in the introduction, we consider graphs whose vertices contain Poisson structures and whose edges represent derivatives. To be precise, the class of graphs to deal with is as follows.

**Definition 1.** Let us consider a class of oriented graphs on  $m + n$  vertices labelled  $0, \dots, m + n - 1$  such that the consecutively ordered vertices  $0, \dots, m - 1$  are sinks, and each of the internal vertices  $m, \dots, m + n - 1$  is a source for two edges. For every internal vertex, the two outgoing edges are ordered using  $L \prec R$ : the preceding edge is labeled  $L$  (Left) and the other is  $R$  (Right). An oriented graph on  $m$  sinks and  $n$  internal vertices is a *Kontsevich graph* of type  $(m, n)$ . We denote by  $G_{m,n}$  the set of all Kontsevich graphs of type  $(m, n)$ , and by  $\tilde{G}_{m,n}$  the subset of  $G_{m,n}$  consisting of all those graphs having neither double edges nor tadpoles.

**Example 1.** The star-product expansion (1) contains graphs in  $\tilde{G}_{2,k}$  for  $0 \leq k \leq 3$ .

*Remark 1.* The class of graphs which we consider is not the most general type considered by Kontsevich in [32]. In the construction of the Formality morphism there also appear graphs with sources for more or fewer (than two) arrows. However, in our approach to the problem at hand, which is the construction of a  $\star$ -product expansion that would be associative modulo  $\hbar^k$  for some  $k \gg 0$ , we shall only meet graphs from the class of Definition 1. Actually, to be more accurate, the Leibniz graphs in Definition 6 on p. 254 are Kontsevich graphs where some vertices have *three* outgoing edges; these are expanded into ordinary Kontsevich graphs (built of wedges) by inserting the Jacobiator at the tri-valent vertex; see [11] for more details.

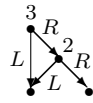
*Remark 2.* There can be tadpoles or cycles in a graph  $\Gamma \in G_{m,n}$ , see Fig. 1.

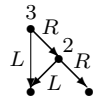


FIGURE 1. A tadpole and an “eye”.

A Kontsevich graph  $\Gamma \in G_{m,n}$  is uniquely determined by the numbers  $n$  and  $m$  together with the list of ordered pairs of targets for the internal vertices. For reasons which will become clear immediately below, we now consider a Kontsevich graph  $\Gamma$  together with a *sign*  $s \in \{0, \pm 1\}$ , denoted by concatenation of the symbols:  $s\Gamma$ .

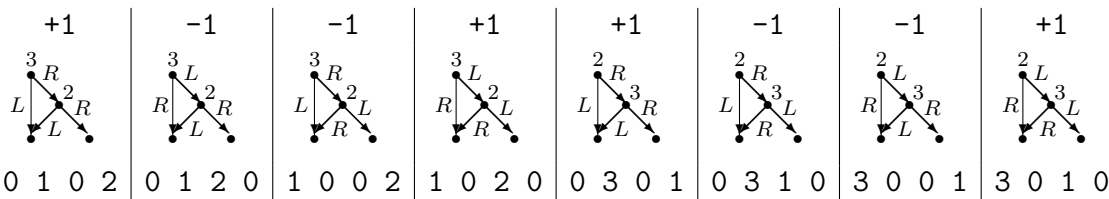
**Implementation 1** (encoding). The format to store a signed graph  $s\Gamma$  with  $\Gamma \in G_{m,n}$  is the integer number  $m > 0$ , the integer  $n \geq 0$ , the sign  $s$ , followed by the (possibly empty, when  $n = 0$ ) list of  $n$  ordered pairs of targets for edges issued from the internal vertices  $m, \dots, m + n - 1$ , respectively. The full format is then  $(m, n, s; \text{list of ordered pairs})$ ; in plain text we also write  $\mathbf{m\ n\ s\ \langle list of ordered pairs \rangle}$ . In the software, the class `KontsevichGraph` represents these signed Kontsevich graphs.



**Example 2.** The graph  has encoding  $2\ 2\ 1\ 0\ 1\ 0\ 2$ .

We recall that to every Kontsevich graph one associates a polydifferential operator by placing a copy of the Poisson bracket at each vertex. To a signed graph one associates the polydifferential operator of the graph multiplied by the sign. The skew-symmetry of the Poisson bracket implies that the same polydifferential operator may be represented by several different signed graphs, all having different encodings.

**Example 3.** Taken with the signs in the first row, the graphs in the second row all represent the same polydifferential operator:



In the third row the target list (for internal vertices 2 and 3, respectively) is written.

We would like to know whether two (encodings of) signed graphs specify the same topological portrait — up to a permutation of internal vertices and/or a possible swap  $L \rightleftharpoons R$  for some pair(s) of outgoing edges. To compare two given encodings of a signed graph, let us define its normal form. Such normal form is a way to pick the representative modulo the action of group  $S_n \times (\mathbb{Z}_2)^n$  on the space  $G_{m,n}$ .

**Definition 2** (normal form). The list of targets of a graph  $\Gamma \in G_{m,n}$  can be considered as a  $2n$ -digit integer written in base- $(n+m)$  notation. By running over the entire group  $S_n \times (\mathbb{Z}_2)^n$ , and by this over all the different re-labelings of  $\Gamma$ , we obtain many different integers written in base- $(n+m)$ . The *absolute value*  $|\Gamma|$  of  $\Gamma$  is the re-labeling of  $\Gamma$  such that its list of targets is *minimal* as a nonnegative base- $(n+m)$  integer. For a signed graph  $s\Gamma$ , the *normal form* is the signed graph  $t|\Gamma|$  which represents the same polydifferential operator as  $s\Gamma$ . Here we let  $t = 0$  if the graph is zero (see Remark 3 below).

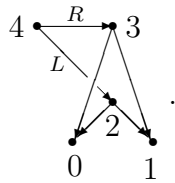
**Example 4.** The minimal base-4 number in the third column of Example 3 is  $0\ 1\ 0\ 2$ . Hence the absolute value of each of the graphs in Example 3 is the first graph. The normal form of each of the signed graphs in Example 3 is the first graph taken with the appropriate sign  $\pm 1$ ; the encodings of the normal forms are then  $2\ 2\ \pm 1\ 0\ 1\ 0\ 2$ .

This normal form is implemented in software as the method `normalize()` of the class `KontsevichGraph`. By running over the entire symmetry group, it will be inefficient when the number of vertices is large. In the future this method could be replaced by a more efficient one, without requiring changes to the rest of the code.

*Remark 3.* The graphs  $\Gamma \in G_{m,n}$  for which the associated polydifferential operator vanishes, by being equal to minus itself, are called *zero*. This property can be detected during the calculation of the normal form of a signed graph. One starts with the encoding of a signed graph. Obtain a “sorted” encoding (representing the same polydifferential operator) by sorting the outgoing edges in every pair in nondecreasing

order: each swap  $L \rightleftharpoons R$  entails a reversion of the sign. Now run over the group  $S_n$  of permutations of the internal vertices in the graph at hand, relabeling those vertices. Should the list of targets in the sorted encoding of a relabeling be equal to the list of targets in the original sorted encoding, but the sign be opposite, then the graph is zero. We will see in Chapter 2 (specifically, in Lemma 2 on p. 238) that the weights of these graphs also vanish, this time by the anticommutativity of certain differentials under the wedge product.

**Example 5.** Consider the graph



with the encoding 2 3 1 0 1 0 1 2 3. For the identity permutation we obtain the initial sorted encoding 2 3 1 0 1 0 1 2 3 (it was already sorted). For the permutation  $2 \rightleftharpoons 3$  we obtain the encoding 2 3 1 0 1 0 1 3 2; upon sorting the pairs it becomes 2 3 -1 0 1 0 1 2 3. The list of pairs coincides with the initial sorted encoding but the sign is opposite; hence the graph is zero.

The notion of normal form of graphs allows one to generate lists of graphs with different topological portraits (e.g., Kontsevich graph series, see section 1.2 below) by using the following algorithm. Initially, the list of generated graphs is empty. For every possible encoding (according to Implementation 1) in a run-through, its normal form with sign +1 or 0 is added to the list if it is not contained there (otherwise, the offered encoding is skipped).

**Implementation 2.** To generate all the Kontsevich graphs with  $m$  sinks and  $n$  internal vertices in  $\tilde{G}_{m,n}$  (without tadpoles or double edges), the command is

```
> generate_graphs n m
```

The procedure lists all such graphs (one per line) in the standard output. The second argument  $m$  may be omitted: the default value is  $m = 2$ .

Similarly, to generate only normal forms (with sign +1 or 0), the call is

```
> generate_graphs n m --normal-forms=yes
```

The optional argument `--with-coefficients=yes` indicates that (numbered) undetermined coefficients should be listed alongside the graphs (the default is `no`); see §1.2. (Accordingly, see `KontsevichGraph::graphs` in Appendix B.)

**Example 6.** The Kontsevich graphs in  $\tilde{G}_{m,n}$  with *one* internal vertex

```
> generate_graphs 1
2 1 1 0 1
2 1 1 1 0
```

consist of the wedge with its two different labellings. We can verify that the number of Kontsevich graphs on  $n$  internal vertices and two sinks is  $(n(n + 1))^n$ :

```

> generate_graphs 2 | wc -l
36
> generate_graphs 3 | wc -l
1728
> generate_graphs 4 | wc -l
160000
> generate_graphs 5 | wc -l
24300000

```

Here, “| wc -l” counts the number of lines in the output (wc is from GNU coreutils).

Let us remember that while a list of graphs is generated, more options can be chosen to restrict the graphs: e.g., only *prime* graphs can be taken into account, graphs of which the mirror-reflection is already on the list can be skipped, and/or only those graphs in which each sink receives at least one arrow can be taken. The purpose and implementation of these options will be explained in the next chapter (see p. 239 below).

**1.2. Series of graphs: file format.** We now specify how formal power series expansions of graphs are implemented in software. Denote by  $\hbar$  the formal parameter; in machine-readable format, a power series expansion in  $\hbar$  is a list of coefficients of  $\hbar^k$ ,  $k \geq 0$ . The coefficients are formal sums of signed graphs (see `KontsevichGraphSum` in Appendix B) in which the coefficients can be of any type, e.g.,

- integer or floating point numbers (e.g., 0.333),
- rational numbers (e.g., 1/3),
- undetermined variables (resp., `OneThird`).

To be precise, the library [9] contains the class `KontsevichGraphSeries` which depends on a template parameter `T`; it specifies the type of all the coefficients of graphs in the series. In the command-line programs, the external type `GiNaC::ex`, which is the expression type of the `GiNaC` library [3], allows all of the above values (and combinations of them). Hence a series under study can contain coefficients of all types at once; the coefficient of a graph itself can be a sum of different types of objects (e.g., `p16 + 0.25`).

**Implementation 3** (series encoding). In the file format for formal power series expansions, two kinds of lines are possible: either

$\hbar^k$ :

or (separated by whitespace)

<encoding of a graph>    <coefficient>

The precision of the formal power series expansion is indicated by the highest  $k$  occurring in lines of the form “ $\hbar^k$ :". Hence one can control this bound by adding such a line with a high  $k$  at the end of the file.

**Example 7.** The Kontsevich  $\star$ -product (see §2) is a graph series given up to the second order in the deformation parameter  $\hbar$  in the file `star2w.txt` which reads

```

h^0:
2 0 1                    1
h^1:
2 1 1    0 1            1

```

```

h^2:
2 2 1   0 1 0 1   1/2
2 2 1   0 1 0 2   w_2_1
2 2 1   0 1 1 2   w_2_2
2 2 1   0 3 1 2   w_2_3

```

**Implementation 4.** The substitution of undetermined coefficients by their actual values, as well as re-expression of indeterminates via other such objects, is done by using the program

```
> substitute_relations <graph-series-file> <substitutions-file>
```

Its command line arguments are two file names: the first file contains the series and the second file consists of a list of substitutions (one per line), each substitution written in the form

```
<variable>==<what it is set equal to>
```

The command line program sends the series with all those substitutions to the standard output.

**Example 8.** The values of the unknowns in Example 7 are written in `weights2.txt`:

```

w_2_1==1/3
w_2_2==-1/3
w_2_3==-1/6

```

Whence the star-product is given modulo  $\bar{o}(\hbar^2)$  as follows:

```

$ substitute_relations star2w.txt weights2.txt > star2.txt
$ cat star2.txt
h^0:
2 0 1           1
h^1:
2 1 1   0 1     1
h^2:
2 2 1   0 1 0 1   1/2
2 2 1   0 1 0 2   1/3
2 2 1   0 1 1 2   -1/3
2 2 1   0 3 1 2   -1/6

```

Here `cat` from GNU `coreutils` is used to display the file.

In practice one may encounter graph series containing many graphs and undetermined coefficients. To split a graph series into parts, the following command is helpful.

**Implementation 5.** To extract the part of a graph series proportional to a given expression, use the call

```
> extract_coefficient <graph-series-file> <expression>
```

In the standard output one obtains a modification of the original graph series: each graph coefficient `c` is now replaced by the coefficient of `<expression>` in `c`. If the coefficient of `<expression>` in `c` is identically zero, then the graph is skipped. The special value `<expression> = 1` yields the constant part of the graph series (all the undetermined variables in the input are set to zero).

**Example 9.** From the file in Example 8, we extract the part proportional to  $w_{2_1}$ :

```
> extract_coefficient star2w.txt w_2_1
h^0:
h^1:
h^2:
2 2 1  0 1 0 2  1
```

It is just one graph.

**1.3. Reduction modulo skew-symmetry.** Let us recall that for every internal vertex in a Kontsevich graph, the pair of out-going edges is ordered by the relation Left  $\prec$  Right and by a mark-up of those two edges using  $L$  and  $R$ . By construction, the coefficients of a graph series are sums of *signed* graphs; each signed graph is specified by its encoding, see Implementation 1 on p. 229 above. Starting from the vector space of formal sums of signed graphs with real coefficients, we pass to its quotient. Namely, we postulate that graphs which differ only by their internal vertex labeling are equal. Further, we proclaim that every reversal of the edge order in any pair (from the same internal vertex) entails the reversion of the graph sign. Lastly, we introduce the relations

$$\langle \text{coeff} \rangle \cdot (\text{sign})\text{Graph} = \langle \text{sign} \cdot \text{coeff} \rangle \cdot (+1)\text{Graph},$$

for each signed graph  $(\text{sign})\text{Graph}$  with any coefficient  $\langle \text{coeff} \rangle$ .

The combined effect of these relations is that each sum of signed graphs may be reduced to a sum of normal forms (see Definition 2) with sign  $+1$ . Recall that the ordering mechanism Left  $\prec$  Right creates graphs that equal zero because they are equal to minus themselves (see Remark 3 and Example 5).

*Remark 4.* To avoid such comparison of graphs with zero all the time and so, to increase efficiency, every signed graph is brought to its normal form as soon as it is constructed. It is this moment when zero graphs acquire zero signs.

The algorithm to reduce a sum of graphs modulo skew-symmetry runs as follows. For the starting graph or every next graph in the list, its sign (if nonzero) is set equal to  $+1$  and its coefficient is modified, if necessary, by using the rule

$$\langle \text{coeff} \rangle \cdot \langle \text{sign} \rangle = \langle \text{sign} \cdot \text{coeff} \rangle \cdot (+1). \quad (3)$$

Every graph with sign 0 is removed. Then the graph at hand (in its normal form, times a coefficient) is compared, disregarding signs, with all the graphs which follow in the list. A match found, its coefficient is added – using relation (3) – to the coefficient of the graph we started with; the match itself is removed. By this reduction procedure for graph sums, all vanishing graphs with zero signs are excluded from the list.

**Implementation 6.** To reduce a graph series expansion modulo skew-symmetry, call

```
> reduce_mod_skew <graph-series-file> [--print-differential-orders]
```

The resulting graph series is sent to the standard output. The optional argument `--print-differential-orders` controls whether the differential orders of the graphs (as operators acting on the sinks) are included in the output, with lines such as

```
# 2 1
```

indicating subsequent graphs have differential order (2, 1). (The corresponding methods are `KontsevichGraphSeries<T>::reduce_mod_skew()` and `KontsevichGraphSum<T>::reduce_mod_skew()` in Appendix B.)

**Example 10.** We put the zero graph from Example 5 with the coefficient +1 into a file `zerograph3.txt`:

```
h^3:
2 3 1 0 1 0 1 2 3 1
```

We confirm that `reduce_mod_skew` kills it:

```
> reduce_mod_skew zerograph3.txt
h^3:
```

The output is an empty list of graphs.

*Remark 5.* An alternative for the implementation of `reduce_mod_skew` is to make use of the plain text file format, in three passes. In the first pass, put all graphs in normal form with sign +1 (updating the coefficients). Recall that graphs are listed *first* in the file format for graph series, so the problem of collecting terms is the same as sorting the file. In the second pass, use `sort` from GNU `coreutils` to sort the file (this uses the very efficient “external *R*-way merge” algorithm). In the third pass: for each normal form, add up the coefficients of every copy in the list (since the list is sorted, one need not look far). The implementation of this algorithm is left as an exercise to the reader.

*Remark 6.* Sums of graphs may also be reduced modulo the (graphical) Jacobi identity and its (pictorial) differential consequences; this is the subject of section 3.2.

**1.4. Evaluate a given graph series at a given Poisson structure.** Let us recall that every Kontsevich graph contains at least one sink. Every edge (decorated with an index, say  $i$ , over which the summation runs from 1 to  $n = \dim N^n$ ) denotes the derivation with respect to a local coordinate  $x^i$  at a given point  $\mathbf{x}$  of the affine manifold  $N^n$  (hence the edge denotes  $\partial/\partial x^i|_{\mathbf{x}}$ ). Every internal vertex (if any) encodes a copy of a given Poisson structure  $\mathcal{P}$ . Should the labellings of two outgoing edges be  $\rightarrow i$  and  $\rightarrow j$  so that the edge with  $i$  precedes that with  $j$ , the Poisson structure in that vertex is  $\mathcal{P}^{ij}(\mathbf{x})$  (that is, the ordering  $i \prec j$  is preserved; moreover, the reference to a point  $\mathbf{x}$  is common to all vertices). Now, every Kontsevich graph (with a coefficient after it) represents a (poly)differential operator with respect to the content of sink(s); to build that operator, we apply the derivations (at  $\mathbf{x} \in N^n$ ) to objects in the arrowhead vertices, multiply the content of all vertices at a fixed set of index values, and then sum over all the indices.

**Example 11** (Jacobi identity). For all Poisson structures  $\mathcal{P}$  and all triples of arguments from the algebra  $C^\infty(N^n)$  of functions on the Poisson manifold at hand, we have that

$$\begin{array}{c} \boxed{\bullet \bullet} \\ \downarrow \quad \downarrow \quad \downarrow \\ 1 \quad 2 \quad 3 \end{array} := \begin{array}{c} \bullet \\ \swarrow \quad \searrow \\ i \quad j \\ \swarrow \quad \searrow \\ 1 \quad 2 \quad 3 \end{array} - \begin{array}{c} \bullet \\ \swarrow \quad \searrow \\ i \quad j \\ \swarrow \quad \searrow \\ 1 \quad 2 \quad 3 \end{array} \begin{array}{c} L \\ \swarrow \\ \bullet \\ \searrow \\ R \end{array} - \begin{array}{c} \bullet \\ \swarrow \quad \searrow \\ i \quad j \\ \swarrow \quad \searrow \\ 1 \quad 2 \quad 3 \end{array} = 0. \tag{4}$$

In formulae, by ascribing the index  $\ell$  to the unlabeled edge, the identity reads

$$(\partial_\ell \mathcal{P}^{ij} \mathcal{P}^{lk} + \partial_\ell \mathcal{P}^{jk} \mathcal{P}^{\ell i} + \partial_\ell \mathcal{P}^{ki} \mathcal{P}^{\ell j}) \partial_i(1) \partial_j(2) \partial_k(3) = 0.$$

Indeed, the coefficient of  $\partial_i \otimes \partial_j \otimes \partial_k$  is the familiar form of the Jacobi identity.

In fact, the graph itself is the most convenient way to transcribe the formulae which one constructs from it, see [25, §2.1] for more details.<sup>6</sup> The computer implementation is straightforward. We acknowledge however that it is one of the most needed instruments.

**Implementation 7.** The call is

```
> poisson_evaluate <graph-series-filename> <poisson-structure>
```

and options for <poisson-structure> are<sup>7</sup>

- 2d-polar,
- 3d-generic,
- 3d-polynomial,
- 4d-determinant,
- 4d-rank2,
- 9d-rank6.

The output is a list of coefficients of the differential operator that the graph series represents, filtered by (a) powers of  $\hbar$ , (b) the differential order as an operator acting on the sinks, and (c) the actual derivatives falling on the sinks.

**Example 12.** Put the graph sum for the Jacobiator  $\text{Jac}(\mathcal{P})$  in `jacobiator.txt`:

```
3 2 1  0 1 2 3  -1
3 2 1  0 2 1 3   1
3 2 1  0 4 1 2  -1
```

We evaluate it at a Poisson structure:

```
> poisson_evaluate jacobiator.txt 2d-polar
Coordinates: r t
Poisson structure matrix:
[[0, r^(-1)]
[-r^(-1), 0]]

h^0:
```

<sup>6</sup>In the variational set-up of Poisson field models, the affine manifold  $N^n$  is realised as fibre in an affine bundle  $\pi$  over another affine manifold  $M^m$  equipped with a volume element. The variational Poisson brackets  $\{\cdot, \cdot\}_{\mathcal{P}}$  are then defined for integral functionals that take sections of such bundle  $\pi$  to numbers. The encoding of variational polydifferential operators by the Kontsevich graphs now reads as follows. Decorated by an index  $i$ , every edge denotes the variation with respect to the  $i$ th coordinate along the fibre. By construction, the variations act by first differentiating their argument with respect to the fibre variables (or their derivatives along the base  $M^m$ ); secondly, the integrations by parts over the underlying space  $M^m$  are performed. Whenever two or more arrows arrive at a graph vertex, its content is first differentiated the corresponding number of times with respect to the jet fibre variables in  $J^\infty(\pi)$  and only then it can be differentiated with respect to local coordinates on the base manifold  $M^m$ . The assumption that both the manifolds  $M^m$  and  $N^n$  be affine makes the construction coordinate-free, see [23, 27] and [24, 26].

<sup>7</sup>The current version of the software does not allow specification of an arbitrary Poisson structure at runtime (e.g. input as a matrix of functions); however, in the source file `util/poisson_structure_examples.hpp` the list of Poisson structures (as matrices) can be extended to one's heart's desire.



```
# 1 1 1
# [ r ] [ r ] [ r ]
0
# [ r ] [ r ] [ t ]
0
# [ r ] [ t ] [ r ]
0
# [ r ] [ t ] [ t ]
0
# [ t ] [ r ] [ r ]
0
# [ t ] [ r ] [ t ]
0
# [ t ] [ t ] [ r ]
0
# [ t ] [ t ] [ t ]
0
```

For example, the pair of lines

```
# [ r ] [ t ] [ r ]
0
```

indicates that the coefficient of  $\partial_r \otimes \partial_t \otimes \partial_r$  is zero in the polydifferential operator.

Restriction of graph series to Poisson structures will be essential in section 3.1 below where systems of linear algebraic equations between the Kontsevich graph weights in  $\star$  will be obtained by restricting the associativity equation  $\text{Assoc}_\star(f, g, h) = 0$  to a given Poisson bracket.

## 2. THE KONTSEVICH $\star$ -PRODUCT

The star-product  $\star = \times + \hbar\{\cdot, \cdot\}_{\mathcal{P}} + \bar{o}(\hbar)$  in  $C^\infty(N^n)[[\hbar]]$  is an associative unital noncommutative deformation of the associative unital commutative product  $\times$  in the algebra of functions  $C^\infty(N^n)$  on a given affine manifold  $N^n$  of dimension  $n < \infty$ . The bi-linear bi-differential  $\star$ -product is realized as a formal power series in  $\hbar$  by using the weighted Kontsevich graphs. In fact, the bi-differential operator at  $\hbar^k$  is a sum of *all* Kontsevich graphs  $\Gamma \in \tilde{G}_{2,k}$  without tadpoles, with  $k$  internal vertices (and two sinks) taken with some weights  $w(\Gamma)$ . Let us recall their original definition [32].

**Definition 3.** Every Kontsevich graph  $\Gamma \in \tilde{G}_{2,k}$  can be embedded in the closed upper half-plane  $\mathbb{H} \cup \mathbb{R} \subset \mathbb{C}$  by placing the internal vertices at pairwise distinct points in  $\mathbb{H}$  and the external vertices at 0 and 1; the edges are drawn as geodesics with respect to the hyperbolic metric, i.e. as vertical lines and circular segments. The angle  $\varphi(p, q)$  between two distinct points  $p, q \in \mathbb{H}$  is the angle between the geodesic from  $p$  to  $q$  and the geodesic from  $p$  to  $\infty$  (measured counterclockwise from the latter):

$$\varphi(p, q) = \text{Arg} \left( \frac{q - p}{q - \bar{p}} \right),$$

and it can be extended to  $\mathbb{H} \cup \mathbb{R}$  by continuity. The *weight* of a Kontsevich graph  $\Gamma \in \tilde{G}_{2,k}$  is given by the integral<sup>8</sup>

$$w(\Gamma) = \frac{1}{(2\pi)^{2k}} \int_{C_k(\mathbb{H})} \bigwedge_{j=1}^k d\varphi(p_j, p_{\text{Left}(j)}) \wedge d\varphi(p_j, p_{\text{Right}(j)}), \tag{5}$$

over the *configuration space* of  $k$  points in the upper half-plane  $\mathbb{H} \subset \mathbb{C}$ ,

$$C_k(\mathbb{H}) = \{(p_1, \dots, p_k) \in \mathbb{H}^k : p_i \text{ pairwise distinct}\};$$

the integrand is defined pointwise at  $(p_1, \dots, p_k)$  by considering the embedding of  $\Gamma$  in  $\mathbb{H}$  that sends the  $j$ th internal vertex to  $p_j$ ; the numbers  $\text{Left}(j)$  and  $\text{Right}(j)$  are the left and right targets of  $j$ th vertex, respectively. (If  $\text{Left}(j)$  is the first or the second sink, put  $p_{\text{Left}(j)} = 0$  or  $1$  respectively; the same goes for  $p_{\text{Right}(j)}$  if  $\text{Right}(j)$  is a sink.)

**Theorem** (Kontsevich [32]). For every Poisson bi-vector  $\mathcal{P}$  on  $N^n$  and an infinitesimal deformation  $\times \mapsto \times + \hbar\{\cdot, \cdot\}_{\mathcal{P}} + \bar{o}(\hbar)$  towards the respective Poisson bracket, the  $\hbar$ -linear star-product

$$\star = \times + \sum_{k \geq 1} \frac{\hbar^k}{k!} \sum_{\Gamma \in \tilde{G}_{2,k}} w(\Gamma) \Gamma(\mathcal{P})(\cdot, \cdot): C^\infty(N^n)[[\hbar]] \times C^\infty(N^n)[[\hbar]] \rightarrow C^\infty(N^n)[[\hbar]] \tag{6}$$

is associative.

**Lemma 1.** Permuting the internal vertex labels of a Kontsevich graph leaves the weight unchanged.

*Proof.* Such a permutation re-orders the factors in a wedge product of two-forms.  $\square$

**Lemma 2.** Swapping  $L \rightleftharpoons R$  at an internal vertex of a Kontsevich graph  $\Gamma \in \tilde{G}_{2,k}$  implies the reversal of the sign of its weight.

*Proof.* Anticommutativity of wedge product of two differentials in formula (5).  $\square$

**Lemma 3.** The weight of a graph  $\Gamma \in \tilde{G}_{2,k}$  and its mirror-reflection  $\bar{\Gamma}$  are related by  $w(\bar{\Gamma}) = (-)^k w(\Gamma)$ .

*Proof.* Taking the reflection of a graph (with respect to the vertical line  $\Re(z) = 1/2$ ) is an orientation-reversing coordinate change on each of the  $k$  “factors”  $\mathbb{H}$  in  $C_k(\mathbb{H})$ .  $\square$

**Lemma 4** ([16]). For a Kontsevich graph such that at least one sink receives no edge(s), its weight is zero.<sup>9</sup>

**Lemma 5.** The map  $w: \sqcup_k \tilde{G}_{2,k} \rightarrow \mathbb{R}$  that assigns weights to graphs is multiplicative,

$$w(\Gamma_i \bar{\times} \Gamma_j) = w(\Gamma_i) \times w(\Gamma_j), \tag{7}$$

with respect to the product  $\bar{\times}$  of graphs,

$$\Gamma_i \bar{\times} \Gamma_j = (\Gamma_i \sqcup \Gamma_j) / \{a^{\text{th}} \text{ sink in } \Gamma_i = a^{\text{th}} \text{ sink in } \Gamma_j, \quad 0 \leq a \leq 1\},$$

which identifies the respective sinks.

<sup>8</sup>We omit the factor  $1/k!$  that was written in [32], to make the weight multiplicative (see Lemma 5).

<sup>9</sup>The fact that the differential order of  $\star$  is positive with respect to either of its arguments should be expected, in view of the required property of the  $\star$ -product to be unital:  $f \star 1 = f = 1 \star f$ .

*Proof.* The integrals converge absolutely [32]; apply Fubini’s theorem and linearity.  $\square$

**Example 13.** Some weight relations obtained from the lemmas above:

$$w\left(\begin{array}{c} \bullet \\ \swarrow \searrow \\ \bullet \end{array}\right) = w\left(\begin{array}{c} \bullet \\ \swarrow \searrow \\ \bullet \end{array}\right)^2; \quad w\left(\begin{array}{c} \bullet \\ \swarrow \searrow \\ \bullet \end{array}\right) = -w\left(\begin{array}{c} \bullet \\ \searrow \swarrow \\ \bullet \end{array}\right); \quad w\left(\begin{array}{c} \bullet \\ \swarrow \searrow \\ \bullet \end{array}\right) = w\left(\begin{array}{c} \bullet \\ \swarrow \searrow \\ \bullet \end{array}\right).$$

Lemma 5 motivates the following definition.




**Definition 4.** A Kontsevich graph  $\Gamma \in \tilde{G}_{2,k}$  is called *composite* if  $\Gamma$  is equal to the  $\bar{\times}$ -product of some Kontsevich graphs on two sinks and positive number of internal vertices in both of the co-factors. Otherwise (if such a realization is not possible), the graph is called *prime*.

Using Lemma 5 and induction, we obtain that the weight of a composite graph  $\Gamma = \Gamma_1 \bar{\times} \dots \bar{\times} \Gamma_t$  is the product of the weights of its factors:  $w(\Gamma) = w(\Gamma_1) \times \dots \times w(\Gamma_t)$ .

**2.1. Basic set of graphs.** We identify a set of graphs such that the weights of those graphs would suffice to determine all the other weights.

**Definition 5.** A *basic* set of graphs on  $k$  internal vertices is a set of pairwise distinct normal forms (the signs of which are discarded) of only those Kontsevich graphs  $\Gamma \in \tilde{G}_{2,k}$  which are prime, and in which every sink receives at least one edge. By definition, the basic set contains the normal form of a graph but not its mirror reflection if it differs from the graph at hand. To decide whether a graph or its mirror-reflection  $\bar{\Gamma} \neq \Gamma$  is included into a basic set, we take the graph whose absolute value is *minimal* as a base- $(k + 2)$  number. Note that a basic set on  $k \geq 3$  vertices *does* contain zero graphs.

**Corollary 6.** To build  $\star$ -product (6) up to  $\bar{o}(\hbar^k)$  for some power  $k \geq 1$ , knowing the Kontsevich weights  $w(\Gamma_i)$  only for a *basic* set of graphs  $\Gamma_i \in \tilde{G}_{2,\ell}$  at all  $\ell \leq k$  is enough. Indeed, the weights of all other graphs with  $\ell$  internal vertices are calculated from Lemmas 1, 2, 3, 4, and 5.

**Example 14.** Consider the prime graph  and its mirror-reflection . The encodings of their normal forms are 2 2 1 0 1 0 2 and 2 2 1 0 1 1 2 respectively. Since 0 1 0 2 < 0 1 1 2 as base-4 numbers, only the first graph is included in the basic set. The fork graph  is mirror-symmetric hence it is included anyway.

The basic set at order 3 is displayed in Figure 2.

**2.2. “All” graphs in  $\star \bmod \bar{o}(\hbar^4)$ .** In Table 1 we list the number of basic graphs at every order  $k \leq 6$  in the Kontsevich  $\star$ -product. The actual number of graphs with respect to which the sums in formula (6) expand is of course much greater.

**Implementation 8.** To obtain the list of normal forms for graphs from a basic set at order  $k$ , the following command is available:

```
> generate_graphs k --basic=yes
```

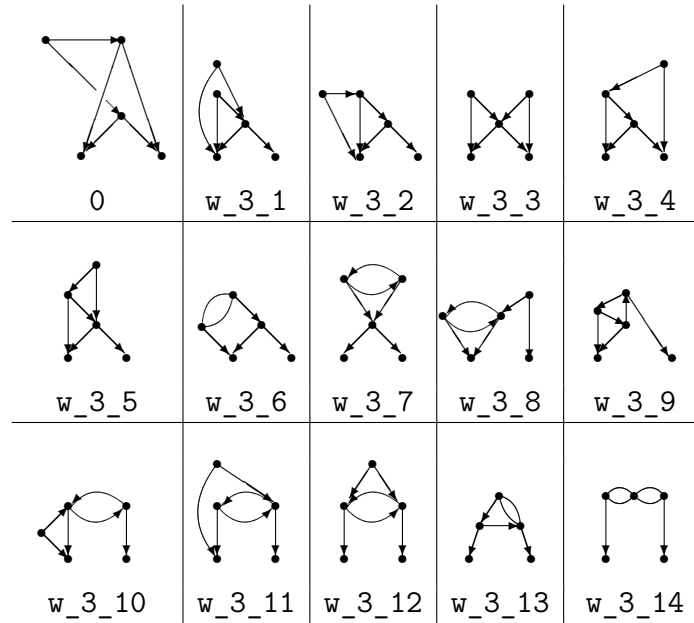


FIGURE 2. Basic set at order 3, with undetermined weights for nonzero graphs. (The weights are determined in Example 26 on p. 250 below.)

TABLE 1. How many basic graphs there are at low orders  $k$ .

Order = $k$	0	1	2	3	4	5	6
#(Basic set)	0	1	2	15	156	2307	43231
#(Nonzero in basic set)	0	1	2	14	149	2218	42050

The list of normal forms is then sent to the standard output. This command is equivalent to

```
> generate_graphs k --prime=yes --normal-forms=yes \
  --postive-differential-order=yes --modulo-mirror-images=yes
```

**Example 15.** The list of basic graphs with  $\leq 3$  internal vertices – with undetermined coefficients at orders 1, 2, 3 – is constructed using the following commands:

```
$ cat > basic3w.txt
h^0:
2 0 1      1
h^1:
^D (press Ctrl+D)
$ generate_graphs 1 --basic=yes --with-coefficients=yes \
  >> basic3w.txt
$ echo 'h^2:' >> basic3w.txt
$ generate_graphs 2 --basic=yes --with-coefficients=yes \
  >> basic3w.txt
$ echo 'h^3:' >> basic3w.txt
$ generate_graphs 3 --basic=yes --with-coefficients=yes \
  >> basic3w.txt
```

The file `basic3w.txt` now contains the basic set.

Starting from a basic set, the  $\star$ -product is built up to a certain order  $k \geq 0$  in  $\hbar$ .

**Implementation 9.** The program

```
> star_product <basic-set-filename>
```

takes as its input a graph series with a basic set of graphs at each order; the graphs go with coefficients of any nature (i.e. number or indeterminate). The program's output is an expansion of the  $\star$ -product up to the order that was specified by the input. In other words, all the graphs which are produced from the ones contained in a given basic set are generated and their coefficients are (re)calculated from the ones in the input (using Lemmas 2, 3, and 5).

**Example 16.** To generate the star-product up to order 3 with all weights of nonzero basic graphs undetermined (from Example 15), one proceeds as follows:

```
$ star_product basic3w.txt > star3w_unreduced.txt
$ reduce_mod_skew --print-differential-orders star3w_unreduced.txt \
  > star3w.txt
```

The file `star3w.txt` now contains the desired star-product.

**2.3. Methods to obtain the weights of basic graphs.** We deduce that to build the  $\star$ -product modulo  $\bar{o}(\hbar^4)$  as many as 149 weights of nonzero basic graphs  $\Gamma_i \in \tilde{G}_{2,4}$  at  $k = 4$  must be found (or at least expressed in terms of as few master-parameters as possible). In fact, direct calculation of all of the 149 Kontsevich integrals is not needed to solve the problem in full because there exist more algebraic relations between the weights of basic graphs. In the following proposition we recall a class of such relations.<sup>10</sup>

**Proposition 7** (cyclic weight relations [17]). Let  $\Gamma$  be a Kontsevich graph on  $m = 2$  ground vertices. Let  $E \subset \text{Edge}(\Gamma)$  be a subset of edges in  $\Gamma$  such that for every  $e \in E$ ,  $\text{target}(e) \neq 0$ . (That is, every edge from the subset  $E$  lands on the sink 1 or an internal vertex.) For every such subset  $E$ , define the graph  $\Gamma_E$  as follows: let its vertices be the same as in  $\Gamma$  and for every edge  $e \in \text{Edge}(\Gamma)$ , preserve it in  $\Gamma_E$  if  $e \notin E$ , but if  $e \in E$  replace that edge by a new edge in  $\Gamma_E$  going from  $\text{source}(e)$  to the sink 0. By definition, the ordering  $L \prec R$  of outgoing edges is inherited in  $\Gamma_E$  from  $E$  even if the targets of any of those edges are new. Thirdly, denote by  $N_0(\Gamma_E)$  the number of edges in  $\Gamma_E$  such that their target is the sink 0. Then the Kontsevich weight of a graph  $\Gamma$  is related to the weights of all such graphs  $\Gamma_E$  obtained from  $\Gamma$  by the formula

$$w(\Gamma) = (-)^n \sum_{\substack{E \subset \text{Edge}(\Gamma) \\ \forall e \in E, \text{target}(e) \neq 0}} (-)^{N_0(\Gamma_E)} w(\Gamma_E). \tag{8}$$

Note that this relation is linear in the weights of all graphs.

---

<sup>10</sup>A convenient approach to calculation of Kontsevich weights (5) at order 3 by using direct integration (and for that, using methods of complex analysis such as the Cauchy residue theorem) was developed in [14], see Appendix A.1 on p. 276 below. However, we note that most successful at  $k = 3$ , this method is no longer effective for all graphs at  $k \geq 4$ . More progress is badly needed to allow  $k \geq 5$ .

If the graph  $\Gamma$  or, in practice, some of the new graphs  $\Gamma_E$  in (8) is composite, Lemma 5 provides a further, nonlinear reduction of  $w(\Gamma)$  by using graphs with fewer internal vertices.

**Example 17.** Consider the graph  $\Gamma_{3,8}$  in Figure 2 with weight  $w(\Gamma_{3,8}) = w_{3,8}$ . For every non-empty subset  $E$  (with  $\text{target}(e) \neq 0$  for every  $e \in E$ ) the graph  $(\Gamma_{3,8})_E$  is a zero-weight graph by virtue of one of the Lemmas at the beginning of this chapter. Hence the only term in the sum on the right-hand side in (8) is the weight of the graph corresponding to the empty set:  $w((\Gamma_{3,8})_\emptyset) = w(\Gamma_{3,8})$ . Since  $n = 3$  and  $N_0(\Gamma_{3,8}) = 2$  we get the cyclic relation  $w(\Gamma_{3,8}) = -w(\Gamma_{3,8})$ ; whence  $w(\Gamma_{3,8}) = 0$ .

*Remark 7.* It is readily seen that only *prime*, that is, non-composite graphs  $\Gamma$  need be used to generate *all* relations (8). Indeed, every subset  $E$  of edges for a composite graph  $\Gamma = \Gamma^1 \bar{\times} \Gamma^2$  splits to a disjoint union  $E^1 \sqcup E^2$  of such subsets for the graphs  $\Gamma^1$  and  $\Gamma^2$  separately. Therefore the re-direction of edges in a composite graph would inevitably yield the composite graph  $\Gamma_{E^1}^1 \bar{\times} \Gamma_{E^2}^2$ . Now, the multiplicativity of Kontsevich weights and the additivity of the count  $N_0(\Gamma_E) = N_0(\Gamma_{E^1}^1) + N_0(\Gamma_{E^2}^2)$  can be used to conclude that the relations obtained from composite graphs are redundant.

**Implementation 10.** The command

```
> cyclic_weight_relations <star-product-file>
```

treats the input  $\star$ -product as a clothesline for graphs and their weights. For each graph  $\Gamma$  in the  $\star$ -product, it outputs the relation (8) between the weights of the respective graphs in the form  $\text{LHS} - \text{RHS} == 0$ .

**Example 18.** At the order three with the  $\star$ -product from Example 16:

```
> cyclic_weight_relations star3w.txt
...
1/3*w_3_6+1/6*w_2_3==0
1/3*w_3_8==0
...
```

*Remark 8.* For some (basic) graphs it happens that the weight integrand in (5), as a differential  $2k$ -form, vanishes identically, even if the graph is not zero due to skew-symmetry. This is the case for 21 out of 149 nonzero basic graphs at  $k = 4$ ; see also Appendix A.1.

For calculations of particular weight integrals we refer to the literature in section 4.

*Remark 9* (rationality). Willwacher and Felder [17] express the weight of a graph in  $\tilde{G}_{2,7}$  as  $p \cdot \zeta(3)^2/\pi^6 + q$  where  $p$  and  $q$  are rational numbers and  $\zeta$  is the Riemann  $\zeta$ -function. Whether  $\zeta(3)^2/\pi^6$  is rational or not is an open problem. The software which we presently discuss supports – through GiNaC [3] – the input of  $\zeta$ -values as coefficients, e.g. the expression  $\zeta(3)^2/\pi^6$  can be input as `zeta(3)^2/Pi^6`. This can be used e.g. to express other weights in terms of such values. According to Banks–Panzer–Pym in [2],  $\mathbb{Q}[(2\pi)^{-1}]$ -linear combinations of multiple zeta values actually start to appear in the harmonic weight coefficients in the  $\star$ -product, but at order  $\hbar^6$ . They do not yet appear at order 4 (or rather, they are all seen to be rational at order 4).

Is any of the weights transcendental? This has not been proved, so that it remains unknown whether any of them is or none of them are.

All the above being said about methods to obtain the values  $w(\Gamma)$  for Kontsevich graph weights and about the schemes to generate linear relations between these numbers, we observe that the requirement of associativity for the  $\star$ -product modulo  $\bar{o}(\hbar^k)$ , whenever that structure is completely known at all orders up to  $\hbar^k$ , is an ample source of relations of that kind. This will be used intensively in chapter 3 from p. 246 onwards. In particular, we mention here that the values of weights of graphs at order  $\ell$  may be restricted by the associativity requirement at orders  $> \ell$ , by restriction to fixed differential orders  $(i, j, k)$  (see Lemma 10 on p. 253).

**2.4. How graphs act on graphs.** Let us have a closer look at the equation of associativity for the sought-for  $\star$ -product:

$$\text{Assoc}_\star(f, g, h) = (f \star g) \star h - f \star (g \star h) = 0.$$

We see that the graph series  $f \star g$  and  $g \star h$  serve as the left- and right co-multiples of  $h$  and  $f$ , respectively, in yet another copy of the star-product. To realize the associator by using the Kontsevich graphs, we now explain how graphs act on graphs (here, in every composition  $\star \circ \star$  the graph series acts on a graph series by linearity).

We postulate that the action of graph series on graph series is  $\mathbb{k}[[\hbar]]$ -linear and  $\mathbb{k}[G_{\star, \star}]$ -linear with respect to both the graphs that act and that become the arguments.

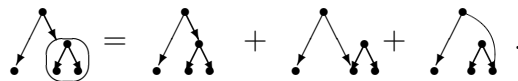
Recall that every Poisson bracket is a derivation in each of its arguments. In consequence, every derivation falling on a sink – in a graph  $\Gamma_1$  that acts on a given graph  $\Gamma_2$  taken as the new content of that sink – acts on the sink’s content via the Leibniz rule; all the Leibniz rules for the derivations in-coming to that sink work independently from each other. Recall that the vertices of a graph represent factors in an expression.

**Example 19.** Consider the action of a wedge graph  $\Lambda$  on the two-sinks graph  $(\bullet\bullet) \in G_{2,0}$ , taken as its second argument. We have that



The result is a sum of Kontsevich graphs of type  $(3, 1)$ . Let us remember that the sinks are distinguished by their ordering; in particular the two Kontsevich graphs on the right-hand side are not equal.

**Example 20.** Now let the wedge graph act on a wedge graph (again, as the former’s second argument):



**Example 21.** Finally, consider a graph in which two arrows fall on the first sink and let its content be  $(\bullet\bullet) \in G_{2,0}$ :



These three examples basically cover all the situations; we shall refer to them again, namely, from the next chapter where the restrictions by using the total differential orders are discussed.

So far, we have focused on graphs; under the action of a graph on a graph, their coefficients are multiplied. (This is why the associativity of the  $\star$ -product is an infinite system of quadratic equations for the coefficients of all the graphs).

**Implementation 11.** In the class `KontsevichGraphSeries<T>`, the method

```
KontsevichGraphSeries<T>::operator()
```

allows function-call syntax for the insertions described above. As its argument it takes a `std::vector` (that is, a list) of the Kontsevich graph series in  $\hbar$ ; these are the  $m$  respective arguments for a Kontsevich graph series. It returns a `KontsevichGraphSeries<T>`. The method is called for the object of the class, that is, for the graph series which is to be evaluated at the  $m$  specified arguments.

For example, this allows the realization of Examples 19 and 20 in C++ expressions as `wedge({ dot, twodots })` and `wedge({ dot, wedge })` respectively.

**Implementation 12.** To calculate the associator  $\text{Assoc}_\star(f, g, h)$  for a given  $\star$ -product and ordered objects  $f, g, h$ , the call is

```
> star_product_associator <star-product-filename>
```

where the input file `<star-product-filename>` contains the (truncated) power series for the  $\star$ -product. In the standard output one obtains a (truncated at the same order in  $\hbar$  as in the input) power series containing, at each power  $\hbar^k$ , the sums of graphs from  $G_{3,k}$  with coefficients (their admissible types were introduced in §1.2 above).

**Example 22.** The associator for the  $\star$ -product up to order 2 (from Example 8):

```
$ star_product_associator star2.txt
h^0:
h^1:
h^2:
# 1 1 1
3 2 1 0 1 2 3 -2/3
3 2 1 0 2 1 3 2/3
3 2 1 0 4 1 2 -2/3
```

It is  $\frac{2}{3}\hbar^2$  times the Jacobiator (4), whose encoding we saw before in Example 12.

**2.5. Gauge transformations.** At first glance, the concept of gauge transformations for (graphs in the)  $\star$ -products is an extreme opposite of plugging a list of graph series as arguments of a given graph series. Namely, the idea of a gauge transformation is that a graph series (possibly of finite length) is towered over a single vertex  $\bullet \in G_{1,0}$ . By definition, a gauge transformation of a vertex  $\bullet$  is a map of the form  $\bullet \mapsto [\bullet] = \bullet + \hbar \cdot (\dots)$  taking  $G_{1,0} \rightarrow \mathbb{k}[G_{1,\star}][[\hbar]]$ .

**Example 23.** The map  $\bullet \mapsto \bullet + \frac{\hbar^2}{12} \text{loop}$  is a gauge transformation of  $\bullet \in G_{1,0}$ . This graph series is encoded in the following file `gaugeloop.txt`:



$\hbar^0$ :  
 1 0 1                    1  
 $\hbar^2$ :  
 1 2 1   0 2 1 0    1/12

The construction of gauge transformations is extended from  $G_{1,0}$  by  $\mathbb{k}[[\hbar]]$ - and  $\mathbb{k}[G_{*,*}]$ -linearity. This effectively means that in the course of action by a gauge transformation  $\mathbf{t}$  on a graph series  $f \in \mathbb{k}[G_{*,*}][[\hbar]]$ , all the arrows work over the vertices in every graph in  $f$  via the Leibniz rule (as it has been explained in the previous section). This is how one expands  $[f] \star [g]$ , that is, the Kontsevich  $\star$ -product (6) of two gauged arguments  $[f]$  and  $[g]$ . Let us recall further that the shape  $[\bullet] = \bullet + \hbar \cdot (\dots)$ , where the gauge tail of  $\bullet$  is given by some graphs from  $\mathbb{k}[G_{1,*}][[\hbar]]$ , guarantees the existence of a formal left inverse  $\mathbf{t}^{-1}$  to the original transformation  $\mathbf{t}$ , so that  $(\mathbf{t}^{-1} \circ \mathbf{t})(\bullet) = \bullet$ .

**Lemma 8.** If  $\bullet \mapsto \blacksquare = \mathbf{t}(\bullet) = \bullet + \hbar\Gamma_1(\bullet) + \dots + \hbar^\ell\Gamma_\ell(\bullet) + \bar{o}(\hbar^\ell)$  is a gauge transformation, let

$$\mathbf{t}^{-1}(\blacksquare) = \blacksquare + \hbar\gamma_1(\blacksquare) + \dots + \hbar^\ell\gamma_\ell(\blacksquare) + \bar{o}(\hbar^\ell)$$

by setting

$$\gamma_0 = \text{id}, \quad \gamma_m(\blacksquare) := - \sum_{k=0}^{m-1} \gamma_k(\Gamma_{m-k}(\blacksquare)).$$


Then  $\mathbf{t}^{-1}(\mathbf{t}(\bullet)) = \bullet$ , that is, the transformation  $\mathbf{t}^{-1}: \mathbb{k}[G_{1,*}][[\hbar]] \rightarrow \mathbb{k}[G_{1,*}][[\hbar]]$  is the left inverse of  $\mathbf{t}$  up to  $\bar{o}(\hbar)$ .

It is readily seen that the assembly of the entire  $\mathbf{t}^{-1}$  can require infinitely many operations even if the direct transformation  $\mathbf{t}$  took only finitely many of them, e.g., as in Example 23.

In these terms, for the Kontsevich  $\star$ -product (6) we obtain, by operating with gauge transformations and their formal inverses, a class of star products  $\star'$  which are defined by the relation

$$\mathbf{t}(f \star' g) = \mathbf{t}(f) \star \mathbf{t}(g), \quad f, g \in C^\infty(N^n)[[\hbar]]. \tag{9}$$

Clearly, all these gauged star-products  $\star'$  remain associative (because  $\star$  was) but the coefficients of graphs at an order  $k \geq 2$  in  $\hbar$  are no longer necessarily equal to the respective values in (6). The use of gauge transformations for products allows to gauge out *some* graphs, often at a certain order  $\hbar^k$  in the star-product expansion.

**Example 24.** The graph  with a loop is gauged out from the Kontsevich  $\star$ -product (6) by using the gauge transformation  $\mathbf{t}: \bullet \mapsto \bullet + \frac{\hbar^2}{12} \text{graph}$ , see Example 23. Note that taking the formal inverse  $\mathbf{t}^{-1}$  does create loop-containing graphs at higher orders  $\hbar^{\geq 3}$  in the gauged star-product  $\star'$  which is specified by (9).

*Remark 10.* Not every graph taken in the Kontsevich star-product  $\star$  at a particular order  $\hbar^k$  can be gauged out. For example, such are the graphs  $\Gamma \in \tilde{G}_{2,*}$  containing an internal vertex  $v$  with edges running from it to both the ground vertices.

**Implementation 13.** The command for gauge transformation is

> gauge <star-product-filename> <gauge-transformation-filename>

where

- the file `<star-product-filename>` contains a machine-format graph encoding of star-product  $\star$  truncated modulo  $\bar{o}(\hbar^k)$  for some  $k \geq 0$ ;
- the content of `<gauge-transformation-filename>` is a gauge transformation  $\mathbf{t}(\bullet)$ , that is, a truncated modulo  $\bar{o}(\hbar^{\ell \geq 0})$  series in  $\hbar$  consisting of the Kontsevich graphs built over one sink vertex  $\bullet$ .

In the standard output one obtains the truncation, modulo  $\bar{o}(\hbar^{\min(k,\ell)})$ , of the graph series for the gauged star-product  $\star'$  defined by  $f \star' g = \mathbf{t}^{-1}(\mathbf{t}(f) \star \mathbf{t}(g))$ .

(The corresponding method is `KontsevichGraphSeries<T>::gauge_transform()` in Appendix B.)

**Example 25.** Let the gauge transformation from Example 24 be stored in the file `gaugeloop.txt`, and recall the  $\star$ -product up to order two from Example 8 in the file `star2.txt`. The gauge transformation kills the loop graph:

```
$ gauge star2.txt gaugeloop.txt > star2gauged_unreduced.txt
$ reduce_mod_skew star2gauged_unreduced.txt > star2gauged.txt
$ cat star2gauged.txt
h^0:
2 0 1          1
h^1:
2 1 1  0 1    1
h^2:
2 2 1  0 1 0 1  1/2
2 2 1  0 1 0 2  1/3
2 2 1  0 1 1 2  -1/3
```

Indeed, we see that the line

```
2 2 1  0 3 1 2  -1/6
```

containing the loop graph has disappeared.

Let us note at once that every gauge transformation  $\mathbf{t}$  given by a Kontsevich graph polynomial in  $\hbar$  of degree  $\ell$  can clearly be viewed formally as a polynomial transformation of any degree greater or equal than  $\ell$ . This is why by using the same software we can actually obtain the gauged star-product  $\star'$  modulo  $\bar{o}(\hbar^4)$  starting with the Kontsevich star-product  $\star$  modulo  $\bar{o}(\hbar^4)$  and applying the gauge transformation of nominal degree  $\ell = 2$  from Example 23. In other words, the precision in  $\star'$  with respect to  $\hbar$  is the same as in  $\star$  even though the degree of the polynomial gauge transformation  $\mathbf{t}$  is smaller. In practice, this is achieved by adding an empty list of graphs at the power  $\hbar^k$  to a given gauge transformation of degree  $\ell < k$ .

### 3. ASSOCIATIVITY OF THE KONTSEVICH $\star$ -PRODUCT

In the final section of this paper we explore two complementary matters. On the one hand, we analyse how the associativity postulate for the Kontsevich  $\star$ -product contributes to finding the values of weights  $w(\Gamma)$  for graphs  $\Gamma$  in  $\star$ . On the other hand, a point is soon reached when no new information can be obtained about the values of  $w(\Gamma)$ : specifically, neither from the fact of associativity of the  $\star$ -product nor

from any proven properties of the Kontsevich weights. We outline a computer-assisted scheme of reasoning that, working uniformly over the set of all Poisson structures under study, reveals the associativity of  $\star$ -product on the basis of our actual knowledge about the weights  $w(\Gamma)$  of graphs  $\Gamma$  in it.

In [12] we reported an exhaustive description of the Kontsevich  $\star$ -product up to  $\bar{o}(\hbar^3)$ . At the next expansion order  $\bar{o}(\hbar^4)$  in  $\star$ , we now express the weights of all the 160 000 =  $(5 \cdot 4)^4$  graphs  $\Gamma \in \tilde{G}_{2,4}$  (of which up to 10 000 =  $(5 \cdot 4/2)^4$  are different modulo signs) in terms of only 10 parameters; those ten master-parameters themselves are the (still unknown) Kontsevich weights of the four internal vertex graphs portrayed in Fig. 3. By following the second strategy we prove that for any values of those ten parameters the  $\star$ -product expansion modulo  $\bar{o}(\hbar^4)$  is associative, also up to  $\bar{o}(\hbar^4)$ .

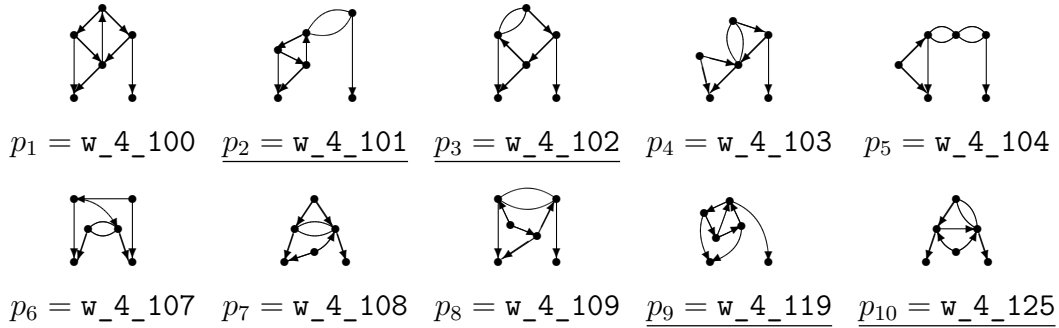


FIGURE 3. The ten graphs whose unknown weights<sup>11</sup> are taken as the master-parameters  $p_i$ ; in fact, the four graphs whose weights are underlined can be gauged out from  $\star$  so that there remain only 6 parameters that determine it modulo  $\bar{o}(\hbar^4)$ .

**3.1. Restriction of the  $\star$ -product associativity equation  $\text{Assoc}_\star(f, g, h) = 0$  to a Poisson structure  $\mathcal{P}$ .** We now view the postulate of associativity for the Kontsevich  $\star$ -product as an equation for coefficients in the graph expansion of  $\star$ . Whenever an expansion modulo  $\bar{o}(\hbar^\ell)$  is known for the  $\star$ -product, one passes to the next order  $\bar{o}(\hbar^{\ell+1})$  by taking all the graphs  $\Gamma \in \tilde{G}_{2,\ell+1}$  with undetermined coefficients, and then expands (with respect to graphs) the associator  $\text{Assoc}_\star(f, g, h)$  up to the order  $\bar{o}(\hbar^{\ell+1})$ . This expansion now runs over all the graphs with at most  $\ell + 1$  internal vertices. It is readily seen that by construction this associativity equation  $\text{Assoc}_\star(f, g, h) = \bar{o}(\hbar^{\ell+1})$  is always *linear*<sup>12</sup> with respect to the coefficients of graphs from  $\tilde{G}_{2,\ell+1}$ .

*Remark 11.* One can still get *linear* relations between the weights  $w(\Gamma)$  of graphs  $\Gamma \in \tilde{G}_{2,\ell+1}$  at order  $\hbar^{\ell+1}$  in  $\star$  by inspecting the associativity of  $\star$  at *higher* orders – ranging from  $\ell + 2$  till  $2\ell + 1$  – in  $\hbar$ . Indeed, a linear relation containing the unknown weights (and the already known lower-order part of  $\star$  as coefficients) but not the weights of graphs with  $\geq \ell + 2$  internal vertices can appear whenever a properly chosen homogeneous

<sup>11</sup>Numerical approximations of two of these weights are listed in Table 3 in Appendix A.1.

<sup>12</sup>Should a graph  $\Gamma \in \tilde{G}_{2,\ell+1}$  be composite so that its Kontsevich weight is factorized using formula (7), the resulting nonlinearity with respect to the weights would actually involve only the graphs with at most  $\ell$  internal vertices.

component of the tri-differential operator  $\text{Assoc}_\star(f, g, h)$  does not contain any weights from higher orders. For instance, this is the component at homogeneity orders  $(i, j, k)$  such that prime graphs  $\Gamma \in \tilde{G}_{2, \geq \ell+2}$  of homogeneity orders  $(i+j, k)$  and  $(i, j+k)$  (when viewed as bi-differential operators) do not exist or if the weights of all such graphs are known in advance.

3.1.1. Let us also note that in the graph equation  $\text{Assoc}_\star(f, g, h) = 0$  that holds by virtue of the Jacobi identity  $\text{Jac}(\mathcal{P}) = 0$ , not every coefficient of every graph in the expansion should be expected to vanish. Indeed, the Jacobiator is a vanishing sum of three graphs that evaluates to zero at every Poisson structure  $\mathcal{P}$  which we put into every internal vertex. This is why the restriction of associativity equation to a given Poisson structure (or to a class of Poisson structures) is a practical way to proceed in solution of the problem of finding the coefficients of graphs in  $\star$ . More specifically, after the restriction of associator  $\text{Assoc}_\star(f, g, h)$  to a structure  $\mathcal{P}$  which is known to be Poisson so that all the instances and all derivatives of the Jacobiator  $\text{Jac}(\mathcal{P})$  are automatically trivialized, the left-hand side of the associativity equation  $\text{Assoc}_\star(f, g, h)|_{\mathcal{P}} = 0 \pmod{\bar{o}(\hbar^{\ell+1})}$  becomes an analytic expression (*linear* with respect to the unknowns  $w(\Gamma)$  for  $\Gamma \in \tilde{G}_{2, \ell+1}$ ). At this point one can proceed in several ways.

We now outline three methods to obtain systems of linear equations upon the unknown weights  $w(\Gamma)$  of basic graphs  $\Gamma \in \tilde{G}_{2, \ell+1}$ . Working in local coordinates, we ensure that the unknowns' coefficients in the equations which we derive are *real* numbers.<sup>13</sup>

**Method 1.** Let the associator's arguments be given functions  $f, g, h \in C^\infty(N^n)$ . Restrict the analytic expression  $\text{Assoc}_\star(f, g, h)|_{\mathcal{P}}$  to a point  $\mathbf{x}$  of the manifold  $N^n$  equipped with a Poisson structure  $\mathcal{P}$ . For every choice of  $f, g, h \in C^\infty(N^n)$  and of a point  $\mathbf{x} \in N^n$ , the restriction  $\text{Assoc}_\star(f, g, h)|_{\mathcal{P}}(\mathbf{x}) = 0 \pmod{\bar{o}(\hbar^{\ell+1})}$  yields *one* linear relation between the weights of graphs at order  $\hbar^{\ell+1}$ . Taking the restriction at several points  $\mathbf{x}_1, \dots, \mathbf{x}_k \in N^n$ , one obtains a system of such equations, the rank of which does not exceed the number  $k$  of such points in  $N^n$ . Bounded by the number of unknowns  $w(\Gamma)$ , the rank would always stabilize as  $k \rightarrow \infty$ .

Examples of Poisson structures  $\mathcal{P}$  – for instance, on the manifolds  $\mathbb{R}^n$  – are available from [20] (here  $n \geq 3$ ) and [37]; from Proposition 2.1 on p. 74 in the latter one obtains a class of Poisson (in fact, symplectic) structures with polynomial coefficients on even-dimensional affine spaces  $\mathbb{R}^{2k}$ . Besides, there is a regular construction (by using the R-matrix formalism, see [33, p. 287]) of Poisson brackets on the vector space of square matrices  $\text{Mat}(\mathbb{R}, k \times k) \cong \mathbb{R}^{k^2}$  (e.g., in this way one has a rank-six Poisson structure on  $\mathbb{R}^9$ ).

Method 1 is the least computationally expensive, so it can be used effectively at the initial stage, e.g., to detect the zero values of certain graph weights: once found, such trivial values allow to decrease the number of unknowns in the further reasoning.

<sup>13</sup>From the factorization of associator for  $\star$  via differential consequences of the Jacobi identity for a Poisson structure  $\mathcal{P}$ , which will be revealed in section 3.2 below, it will be seen in hindsight that the construction of linear relations between the graph weights is overall insensitive to a choice of local coordinates in a chart within a given Poisson manifold. Indeed, the factorization will have been achieved simultaneously for all Poisson structures on all the manifolds at once, irrespective of any local coordinates.

**Method 2.** Now let  $f, g, h \in \mathbb{k}[x^1, \dots, x^n]$  be polynomials referred to local coordinates  $x^1, \dots, x^n$  on  $N^n$ . On that coordinate chart  $U_\alpha \subset N^n$ , take a Poisson structure the coefficients  $\mathcal{P}^{ij}(\mathbf{x})$  of which would also be polynomial. In consequence, the left-hand side of the equation  $\text{Assoc}_\star(f, g, h)|_{\mathcal{P}} = 0 \pmod{\bar{o}(\hbar^{\ell+1})}$  then becomes polynomial as well. Linear in the unknowns  $w(\Gamma)$ , all the coefficients of this polynomial equation vanish (independently from each other). Again, this yields a system of linear algebraic equations for the unknown weights  $w(\Gamma)$  of the Kontsevich graphs  $\Gamma \in \tilde{G}_{2,\ell+1}$  in the  $\star$ -product.

We observe that the linear equations obtained by using Method 2 better constrain the set of unknowns  $w(\Gamma)$ , that is, the rank of this system is typically higher than in Method 1. Intuitively, this is because the polynomials at hand are not collapsed to their values at points  $\mathbf{x} \in N$ .

**Method 3.** Keep the associator's arguments  $f, g, h$  unspecified and consider a class of Poisson structures  $\mathcal{P}[\psi_1, \dots, \psi_m]$  depending in a differential polynomial way on functional parameters  $\psi_\alpha$ , that is, on arbitrary functions, whenever  $\mathcal{P}$  is referred to local coordinates. (For example, let  $n = 3$  and on  $\mathbb{R}^3$  with Cartesian coordinates  $x, y, z$  introduce the class of Poisson brackets using the Jacobian determinants,

$$\{u, v\}_{\mathcal{P}} = p \cdot \det(\partial(q, u, v)/\partial(x, y, z)), \quad q \in C^\infty(\mathbb{R}^3), \tag{10}$$

supposing that the density  $p(x, y, z)$  is also smooth on  $\mathbb{R}^3$ .) Now view the associator  $\text{Assoc}_\star(f, g, h)|_{\mathcal{P}[\psi_1, \dots, \psi_m]}$  as a polydifferential operator in the parameters  $f, g, h$  (with respect to which it is linear) and in  $\psi_1, \dots, \psi_m$  from  $\mathcal{P}$ . By splitting the associator, which is postulated to vanish modulo  $\bar{o}(\hbar^{\ell+1})$ , into homogeneous differential-polynomial components, we obtain a system of linear algebraic equations upon the graph weights.

It is readily seen that, whenever the parameters  $\psi_1, \dots, \psi_m$  are chosen to be polynomials (here let us suppose for definition that the resulting Poisson structure  $\mathcal{P}(\mathbf{x})$  itself is polynomial), the rank of the algebraic system obtained by Method 3 can be greater than the rank of an analogous system from Method 2. This is because the analytic expression  $\text{Assoc}_\star(f, g, h)|_{\mathcal{P}[\psi_1, \dots, \psi_m]}$  keeps track of all the parameters, whereas in Method 2 they are merged to a single polynomial.

We finally note that the linear algebraic systems which are produced by each method should be merged. Indeed, the goal is to maximize the rank and by this, reduce the number of free parameters in the solution.<sup>14</sup>

It has been seen in §2.4, Implementation 12 how the associator is calculated in terms of graphs. The next step – namely, restriction of the associator to a given Poisson structure – can be performed by using a call `poisson_evaluate` as it has been explained in §1.4. However, the further restriction as described in the Methods has been implemented in a separate program (similar to `poisson_evaluate`) which directly outputs the desired relations, as follows.

**Implementation 14.** The command

---

<sup>14</sup>If the rank of the resulting linear algebraic system is equal to the number of unknowns – and if all the coefficients coming from lower orders  $\leq \ell$  within the  $\star$ -product expansion with respect to  $\hbar$  are also rational – then all the solution components are rational numbers as well, cf. [17].

```
> poisson_make_vanish <graph-series-file> <poisson-structure>
```

sends to the standard output relations such as

$$-1/24+w_3_1+4*w_3_2=0$$

between the undetermined coefficients in the input, which must hold if the input graph series is to vanish as a consequence of the Jacobi identity for the specified Poisson structure. The implementation is described in the Methods above. The choice of Poisson structure is made in the same way as in Implementation 7. If the optional extra argument `--linear-solve` is specified, the program will assume that the relations which will be obtained are linear, and attempt to solve the linear system.

**Example 26.** To obtain all the weights of basic graphs  $\Gamma \in \tilde{G}_{2,3}$  at  $\hbar^3$  in the Kontsevich star-product  $\star$ , it was enough to build the linear system of algebraic equations that combined (i) cyclic relations (8), (ii) the relations which Method 3 produces for generic Poisson structure (10), and (iii) those linear relations between the weights of  $\Gamma \in \tilde{G}_{2,3}$  which –in view of Remark 11 on p. 247– still do appear at the next power  $\hbar^4$  in  $\text{Assoc}_\star(f, g, h) = 0$ , by using the same generic Poisson structure (10). The resulting expansion of  $\star$ -product modulo  $\bar{o}(\hbar^3)$  is shown in formula (1) on p. 225. This result is achieved by using the software as follows. Starting from the sets of basic graphs up to the order 2 (with known weights) in the file `basic2.txt`, generate lists of basic graphs (with undetermined weights) up to the order four:

```
$ cp basic2.txt basic3+4w.txt
$ echo 'h^3:' >> basic3+4w.txt
$ generate_graphs 3 --basic=yes --with-coefficients=yes \
  >> basic3+4w.txt
$ echo 'h^4:' >> basic3+4w.txt
$ generate_graphs 4 --basic=yes --with-coefficients=yes \
  >> basic3+4w.txt
```

Build the  $\star$ -product expansion up to the order 4 from these basic sets:

```
$ star_product basic3+4w.txt > star3+4w_unreduced.txt
$ reduce_mod_skew star3+4w_unreduced.txt > star3+4w.txt
```

Generate cyclic weight relations:

```
$ cyclic_weight_relations star3+4w_unreduced.txt \
  > weight_relations_3+4w-cyclic.txt
```

Build the associator expansion up to the order 4 from the  $\star$ -product expansion:

```
$ star_product_associator star3+4w.txt > assoc3+4w.txt
```

Obtain relations from the requirement of associativity for the Poisson structure (10):

```
$ poisson_make_vanish assoc3+4w.txt 3d-generic \
  > weight_relations_3+4w-3d.txt
```

Merge the systems of linear relations:

```
$ cat weight_relations_3+4w-* > weight_relations_3+4w_all.txt
```

Solving the linear system in `weight_relations_3+4w_all.txt` yields the solution

$$\begin{array}{llll} w_{3_1}=1/24, & w_{3_2}=0, & w_{3_3}=0, & w_{3_4}=-1/48, & w_{3_5}=-1/48 \\ w_{3_6}=0, & w_{3_7}=0, & w_{3_8}=0, & w_{3_9}=0, & w_{3_{10}}=0 \end{array}$$

$$w_{3\_11}=-1/48, w_{3\_12}=-1/48, w_{3\_13}=0, w_{3\_14}=0.$$

Store the set of basic graphs at  $\hbar^3$  with their true weights in the file `basic3.txt` (not removing graphs with zero weights); store the true Kontsevich  $\star$ -product up to  $\hbar^3$  in the file `star3.txt` and its associator in the file `assoc3.txt`.

Instead of evaluating the associator in full, we could also have selected (e.g. by reading the file `assoc3+4w.txt`, which also contains lines of the form “# i j k”) those differential orders  $(i, j, k)$  at  $\hbar^4$  at which only weights from order 3 appear, in view of Remark 11: such are  $(1, 3, 2), (2, 3, 1), (2, 1, 3), (3, 2, 1), (3, 1, 2), (1, 2, 3)$  and  $(2, 2, 2)$ .

*Remark 12.* A substitution of the values of certain graph weights expressed via other weights is tempting but not always effective. Namely, we do not advise repeated running of any of the three methods with such expressions taken into account in the input. Usually, the gain is disproportional to the time consumed; for instead of a coefficient to express the program now has to handle what typically is a linear combination of several coefficients. This shows that the only types of substitutions which are effective are either setting the coefficients to fixed numeric values (e.g., to zero) or the shortest possible assignments of a weight value via a single other weight value (like  $w(\Gamma_1) = -w(\Gamma_2)$  for some graphs  $\Gamma_1$  and  $\Gamma_2$ ).

3.1.2. *The  $\star$ -product expansion at order four.* At order four in the expansion of the Kontsevich  $\star$ -product with respect to  $\hbar$ , there are 149 basic graphs  $\Gamma \in \tilde{G}_{2,4}$ . The knowledge of their coefficients would completely determine the  $\star$ -product modulo  $\bar{o}(\hbar^4)$ . By using Methods 1–3 from §3.1, we found the exact values of 67 basic graphs and we expressed the remaining 82 weights in terms of the 10 master-parameters (themselves the weights of certain graphs from  $\tilde{G}_{2,4}$ ; the other 72 weights are linear functions of these ten).

**Theorem 9.** *The weights of basic Kontsevich graphs at order 4 are subdivided as follows. The weights of 27 basic graphs are equal to zero. Of these 27, the integrands of 21 weights are identically zero, and the other 6 weight values were found to be equal to zero. The remaining 122 weights of basic graphs  $\Gamma \in \tilde{G}_{2,4}$  are arranged as follows:*

- 40 nonzero weights are known explicitly;
- the values of the remaining 82 weights are expressed linearly in terms of the weights of those ten graphs which are shown in Fig. 3.
- *The encoding of entire  $\star$ -product modulo  $\bar{o}(\hbar^4)$ , that is, its part up to  $\bar{o}(\hbar^3)$  known from formula (1) plus  $\hbar^4$  times the sum of all the prime and composite weighted graphs with four internal vertices, is given in Appendix C. (In that table the weights of composite graphs are numbers; for they are expressed via the known coefficients of graphs from  $\tilde{G}_{2,\leq 3}$ .) The weights of basic graphs at  $\hbar^4$  are expressed in Table 7 in terms of the ten master-parameters, see p. viii in Appendix C.*

Moreover (as stated in Theorem 12 on p. 256 below), the associativity  $\text{Assoc}_\star(f, g, h) = 0 \pmod{\bar{o}(\hbar^4)}$  is established (up to order four) for the star product  $\star \pmod{\bar{o}(\hbar^4)}$  at all values of the ten master-parameters.

*Proof scheme (for Theorem 9).* We run the software as follows. First one generates the sets of basic graphs up to order 4, with undetermined weights at order 4 (the weights at order 2 and 3 are known from e.g. Example 8 and Example 26):

```
$ cp basic3.txt basic4w.txt
$ echo 'h^4:' >> basic4w.txt
$ generate_graphs 4 --basic=yes --with-coefficients=yes \
  >> basic4w.txt
```

(The output is listed in Table 5 of Appendix C.)

Build the  $\star$ -product expansion up to order 4:

```
$ star_product basic4w.txt > star4w_unreduced.txt
$ reduce_mod_skew --print-differential-orders star4w_unreduced.txt \
  > star4w.txt
```

(The output is listed in Table 6 of Appendix C.)

Generate the linear cyclic weight relations at order 4:

```
$ cyclic_weight_relations star4w_unreduced.txt \
  > weight_relations_4w-cyclic.txt
```

Find 21 relations of the form  $w_4\text{xxx}=0$  which hold by virtue of the weight integrand vanishing in formula (5), by using Implementation 17 in Appendix A.1, and place these relations in the file `weight_relations_4w-integrandvanishes.txt`.

Build the expansion of the associator for the  $\star$ -product up to the order 4:

```
$ star_product_associator star4w.txt > assoc4w.txt
```

(The output is listed in Table 8 of Appendix C.)

Obtain relations from the requirement of associativity for the Poisson structure (10):

```
$ poisson_make_vanish assoc4w.txt 3d-generic \
  > weight_relations_4w-3d.txt
```

Merge the systems of linear equations:

```
$ cat weight_relations_4w-* > weight_relations_4w_total.txt
```

Solve the resulting system (contained in `weight_relations_4w_total.txt`) by using any relevant software. One obtains the relations listed in Table 7 in Appendix C, e.g. in the file `weight_relations_4w_intermsof10.txt`. To express the star-product (respectively, the associator for the  $\star$ -product) in terms of the 10 parameters, run

```
$ substitute_relations star4w.txt \
  weight_relations_4w_intermsof10.txt \
  > star4_intermsof10_unreduced.txt
$ reduce_mod_skew star4_intermsof10_unreduced.txt \
  > star4_intermsof10.txt
```

(respectively, substitute into `assoc4w.txt` to obtain `assoc4_intermsof10.txt`); see Implementation 4. □



*Remark 13.* Numerical approximations of weights are listed in Tables 2 and 3 in Appendix A.1. In particular, we have the approximate values of the master-parameters  $p_4 = w_{4\_103} \approx -1/11520$  and  $p_5 = w_{4\_104} \approx 1/2880$ .<sup>15</sup>

*Remark 14.* Out of the 149 weights of basic graphs in the Kontsevich  $\star$ -product, as many as 28 weights do not appear in the equation  $\text{Assoc}_\star(f, g, h) = 0$  at  $\hbar^4$ . A mechanism which works towards such disappearance is that some graphs  $\Gamma \in \tilde{G}_{2,4}$  which do not show up are bi-derivations with respect to the sinks. Combined at order four in the associator with only the original undeformed product  $\times$ , every such graph is cancelled out from  $(f \star g) \star h - f \star (g \star h)$  according to the mechanism which we illustrate here:

$$\left[ \begin{array}{c} \blacksquare \\ \swarrow \quad \searrow \\ \bullet \quad \bullet \end{array} , \bullet \bullet \right] = \begin{array}{c} \blacksquare \\ \swarrow \quad \searrow \\ \circ \bullet \quad \bullet \end{array} + \begin{array}{c} \blacksquare \\ \swarrow \quad \searrow \\ \bullet \quad \bullet \end{array} - \begin{array}{c} \blacksquare \\ \swarrow \quad \searrow \\ \bullet \quad \circ \bullet \end{array} - \begin{array}{c} \blacksquare \\ \swarrow \quad \searrow \\ \bullet \bullet \quad \bullet \end{array} = 0.$$

In this way the ten master-parameters are split into the six which do show up in the associativity equation and the four weights which do not show up in  $\text{Assoc}_\star(f, g, h) = 0$  at  $\hbar^4$  but which do appear through the cyclic weight relations (see formula (8) on p. 241).

**3.2. Computer-assisted proof scheme for associativity of  $\star$  for all  $\{\cdot, \cdot\}_{\mathcal{P}}$ .** In practice, the methods from §3.1 stop producing linear relations that would be new with respect to the already known constraints for the graph weights. As soon as such “saturation” is achieved, the number of master-parameters in  $\star$ -product expansion may in effect be minimal. That is, the  $\star$ -product, known so far up to a certain order  $\bar{o}(\hbar^k)$ , may in fact be always associative – modulo  $\bar{o}(\hbar^k)$  – irrespective of a choice of the Poisson structure(s)  $\mathcal{P}$ .

In this section we outline a scheme of computer-assisted reasoning that allows to reveal the factorization  $\text{Assoc}_\star(f, g, h) = \diamond(\mathcal{P}, \text{Jac}(\mathcal{P}))(f, g, h)$  of associator for  $\star$  via the Jacobiator  $\text{Jac}(\mathcal{P})$  that vanishes by definition for every Poisson structure  $\mathcal{P}$ . At order  $k = 2$  the factorization  $\diamond(\text{Jac}(\mathcal{P}))$  is readily seen; the factorizing operator  $\diamond(\text{Jac}(\mathcal{P})) = \frac{2}{3}\hbar^2 \text{Jac}(\mathcal{P}) + \bar{o}(\hbar^2)$  is a differential operator of order zero, acting on its argument  $\text{Jac}(\mathcal{P})$  by multiplication. Involving the Jacobi identity and only seven differential consequences from it at the next expansion order  $k = 3$ , the factorization  $\text{Assoc}_\star(f, g, h) = \diamond(\mathcal{P}, \text{Jac}(\mathcal{P}))(f, g, h)$  was established by hand in [12]. For higher orders  $k \geq 4$  the use of software allows to extend this line of reasoning; the scheme which we now provide works uniformly at all orders  $\geq 2$ .

Let us first inspect how sums of graphs can vanish by virtue of differential consequences of the Jacobi identity  $\text{Jac}(\mathcal{P}) = 0$  for Poisson structures  $\mathcal{P}$  on finite-dimensional affine real manifolds  $N^n$ .

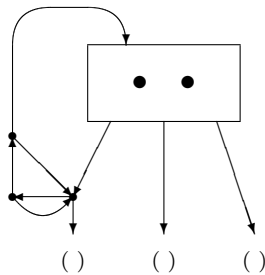
**Lemma 10** ([12]). A tri-differential operator  $C = \sum_{|I|,|J|,|K| \geq 0} c^{IJK} \partial_I \otimes \partial_J \otimes \partial_K$  with coefficients  $c^{IJK} \in C^\infty(N^n)$  vanishes identically if and only if all its homogeneous components  $C_{ijk} = \sum_{|I|=i,|J|=j,|K|=k} c^{IJK} \partial_I \otimes \partial_J \otimes \partial_K$  vanish for all differential orders  $(i, j, k)$  of the respective multi-indices  $(I, J, K)$ ; here  $\partial_L = \partial_1^{\alpha_1} \circ \dots \circ \partial_n^{\alpha_n}$  for a multi-index  $L = (\alpha_1, \dots, \alpha_n)$ .

<sup>15</sup>The values of ten master-parameters have been suggested by Pym and Panzer [34], see Table 4 on p. 280 in Appendix A.2 below. Their prediction completely agrees with our numeric data.

Lemma 10 states in practice that for every arrow falling on the Jacobiator (for which, in turn, a triple of arguments is specified), the expansion of the Leibniz rule yields four fragments which vanish separately. Namely, there is the fragment such that the derivation acts on the content  $\mathcal{P}$  of the Jacobiator’s two internal vertices, and there are three fragments such that the arrow falls on the first, second, or third argument of the Jacobiator. It is readily seen that the action of a derivative on an argument of the Jacobiator effectively amounts to an appropriate redefinition of its respective argument (cf. Examples 19–21 on p. 243). Therefore, a restriction to the order  $(1, 1, 1)$  is enough in the run-through over all the graphs which contain Jacobiator (4) and which stand on the three arguments  $f, g, h$  of the operator  $\diamond(\mathcal{P}, \text{Jac}(\mathcal{P}))$  at hand.

**Definition 6.** A *Leibniz graph* is a graph whose vertices are either sinks, or the sources for two arrows, or the Jacobiator (which is a source for three arrows). There must be at least one Jacobiator vertex. The three arrows originating from a Jacobiator vertex must land on three distinct vertices (and not on the Jacobiator itself). Each edge falling on a Jacobiator works by the Leibniz rule on the two internal vertices in it.

An example of a Leibniz graph is given in Fig. 4. Every Leibniz graph can be expanded to a sum of Kontsevich graphs, by expanding both the Leibniz rule(s) and all copies of the Jacobiator. In this way (sums of) Leibniz graphs also encode (poly)differential operators  $\diamond(\mathcal{P}, \text{Jac}(\mathcal{P}))$ , depending on the bi-vector  $\mathcal{P}$  and the tri-vector  $\text{Jac}(\mathcal{P})$ .



- There is a cycle,
- there is a loop,
- there are no tadpoles in this graph,
- an arrow falls back on  $\text{Jac}(\mathcal{P})$ ,
- and  $\text{Jac}(\mathcal{P})$  does not stand on all of the three sinks.

FIGURE 4. A nontrivial example of Leibniz graph.

By design, we have

**Proposition 11.** For every Poisson bi-vector  $\mathcal{P}$  the value – at the Jacobiator  $\text{Jac}(\mathcal{P})$  – of every (poly)differential operator encoded by the Leibniz graph(s) is zero.

*Proof.* By induction on the number of arrows falling on the Jacobiator. In case of zero arrows, the operator is a multiple of a Jacobiator and hence zero. In general, the operator associated to a Leibniz graph is of the form

$$(\partial_L \text{Jac}(\mathcal{P}))(A, B, C) \cdot D,$$

where  $\partial_L$  are the incoming arrows on the Jacobiator. Now,  $\text{Jac}(\mathcal{P})(A, B, C) = 0$  implies

$$\begin{aligned} 0 &= \partial_L(\text{Jac}(\mathcal{P})(A, B, C)) \cdot D \\ &= (\partial_L \text{Jac}(\mathcal{P}))(A, B, C) \cdot D + \sum_{\substack{H+I+J+K=L \\ H \neq L}} (\partial_H \text{Jac}(\mathcal{P}))(\partial_I A, \partial_J B, \partial_K C) \cdot D \end{aligned}$$

where the terms in the sum on the right are Leibniz graphs with *fewer* arrows falling on the Jacobiator, hence they are zero by induction. The same proof works for a Leibniz graph with more than one Jacobiator (the extraneous ones – in  $D$  – are irrelevant).  $\square$

Hence, to show that a sum of Kontsevich graphs vanishes at every Poisson structure, it suffices to write it as a sum of Leibniz graphs.

In particular, the mechanism of factorization of the associator for the Kontsevich  $\star$ -product is known from [32]; it has been discussed in [11]. Namely, by [32] the Jacobi identity is the only obstruction to the Kontsevich  $\star$ -product associativity. This is because an expression of the  $\star$ -product associator as a (possibly, non-unique) sum of Leibniz graphs can be predicted in advance, based on the graphs in the Formality morphism (see [11] for more details).

**Example 27.** Consider the associator  $\text{Assoc}_\star(f, g, h) \bmod \bar{o}(\hbar^3)$  for the  $\star$ -product which is fully known up to order 3. The assembly of factorizing operator  $\diamond(\mathcal{P}, \cdot)$  acting on  $\text{Jac}(\mathcal{P})$  is explained in [12]; linear in its argument, the operator  $\diamond$  has differential order one with respect to the Jacobiator.

*Remark 15.* The same technique, showing the vanishing of a sum of Kontsevich graphs by writing it as a sum of Leibniz graphs, has been used in [7]. The underlying mechanism from [31] is analyzed in detail in [13].

**Implementation 15** (Encoding of Leibniz graphs). For a Leibniz graph with  $\ell$  Jacobiators and  $n - 2\ell$  remaining bi-vector vertices, an encoding is defined in terms of the encoding of a Kontsevich graph in its expansion, plus the data which tells where the Jacobiators are. The full encoding is the integer  $\ell$ , followed by the Kontsevich graph encoding with  $n$  internal vertices, followed by the  $\ell$  pairs of Jacobiator vertices  $(j_1, j_2)$ , where the internal Jacobiator edge is  $j_1 \leftarrow j_2$ . Each target in the Kontsevich graph encoding which is a Jacobiator vertex  $j_i$  from a Jacobiator  $(j_1, j_2)$  (except for the target of the internal Jacobiator edge  $j_1 \leftarrow j_2$ ) should be interpreted as a placeholder for a Leibniz rule acting on both  $j_1$  and  $j_2$ .

**Example 28.** The Leibniz graph from Fig. 4 (with  $n = 5$  and  $\ell = 1$ ) has the encoding

1   3 5 1   0 5 3 6 3 4 3 1 6 2   6 7

Here the first 6 should be interpreted as a placeholder for the Jacobiator containing the last two vertices 6 and 7; the three arguments of the Jacobiator are 3, 1, 2. To expand this encoding into Kontsevich graph encodings, cyclically permute the arguments of the Jacobiator and replace the placeholder by 6 or 7 (in all possible ways):

3 5 1   0 5 3 6 3 4 3 1 6 2  
 3 5 1   0 5 3 7 3 4 3 1 6 2  
 3 5 1   0 5 3 6 3 4 1 2 6 3  
 3 5 1   0 5 3 7 3 4 1 2 6 3  
 3 5 1   0 5 3 6 3 4 2 3 6 1  
 3 5 1   0 5 3 7 3 4 2 3 6 1

One obtains six terms.

**Implementation 16.** Let the input file `<graph-series-filename>` contain a graph series  $S$  with constant (e. g., rational, real or complex) coefficients; here  $S$  is supposed

to vanish by virtue of the Jacobi identity and its differential consequences. Now run the command

```
> reduce_mod_jacobi <graph-series-filename>
```

The program finds a particular solution  $\diamond$  of the factorization problem

$$S(f, g, h) = \diamond(\mathcal{P}, \text{Jac}(\mathcal{P}), \dots, \text{Jac}(\mathcal{P}))(f, g, h).$$

In the standard output one obtains the list of encodings of Leibniz graphs in  $\diamond$  that specify differential consequences of the Jacobi identity; every such graph encoding is followed in the output by its sought-for nonzero coefficient.<sup>16</sup> Two extra options can be set equal to nonnegative integer values, by passing these two numbers as extra command-line arguments. Namely,

- the parameter `max-jacobiators` restricts the number of Jacobiators in each Leibniz graph, so that by the assignment `max-jacobiators = 1` the right-hand side  $\diamond(\mathcal{P}, \text{Jac}(\mathcal{P}))$  is linear in the Jacobiator, whereas if `max-jacobiators = 2`, the right-hand side  $\diamond(\mathcal{P}, \text{Jac}(\mathcal{P}), \text{Jac}(\mathcal{P}))$  can be quadratic in  $\text{Jac}(\mathcal{P})$ , and so on;
- independently, the parameter `max-jac-indegree` restricts (from above) the number of arrows falling on the Jacobiator(s) in each of the Leibniz graphs that constitute the factorizing operator  $\diamond$ .

Furthermore, if `--solve` is specified as the third extra argument, the input graph series is allowed to contain undetermined coefficients; these are then added as variables to solve-for in the linear system.

**Theorem 12.** *For every component  $S^{(i)}$  of the associator (for  $\star$  from Theorem 9)*

$$\text{Assoc}_\star(f, g, h) \pmod{\bar{o}(\hbar^4)} =: S^{(0)} + p_1 S^{(1)} + \dots + p_{10} S^{(10)},$$

*there exists a factorizing operator  $\diamond^{(i)}$  such that*

$$S^{(i)}(f, g, h) = \diamond^{(i)}(\mathcal{P}, \text{Jac}(\mathcal{P}))(f, g, h), \quad 0 \leq i \leq 10.$$

- *At no values of the master-parameters  $p_i$  would the solution  $\diamond = \sum_i \diamond^{(i)}$  of factorization problem be a first-order differential operator acting on the Jacobiator.*

*Proof scheme.* Take the associator  $\text{Assoc}_\star(f, g, h) \pmod{\bar{o}(\hbar^4)}$  for the  $\star$ -product expansion modulo  $\bar{o}(\hbar^4)$ , in the file `assoc4_intermsof10.txt` which was obtained in Theorem 9. The associator is linear in the ten master-parameters. Let us split it into the constant term (e.g., at the zero value of every parameter) plus the ten respective components  $S^{(i)}$ :

```
$ extract_coefficient assoc4_intermsof10.txt 1 \
  > assoc4_intermsof10_constantpart.txt
$ extract_coefficient assoc4_intermsof10.txt w_4_100 \
  > assoc4_intermsof10_part100.txt
$ extract_coefficient assoc4_intermsof10.txt w_4_101 \
  > assoc4_intermsof10_part101.txt
```

<sup>16</sup>Sample outputs of specified type are contained in Table 9 in Appendix D.

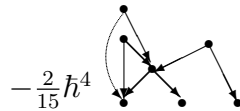
(and so on, for each parameter  $p_i$ ). In fact, four of the parameters do not show up in the associator (see Remark 14): the corresponding files do not contain any graphs. Now run the command `reduce_mod_jacobi` for each input file with  $S^{(i)}$ , e.g., for  $S^{(1)}$ :

```
$ reduce_mod_jacobi assoc4_intermsof10_part100.txt
```

For each  $S^{(i)}$  a solution is found: the series vanishes modulo the Jacobi identity. The output for  $S^{(1)}$  is written in Table 9 in Appendix D. For the second part of the theorem, we run `reduce_mod_jacobi` with the options `max-jac-indegree = 1` and `--solve`:

```
$ reduce_mod_jacobi assoc4_intermsof10.txt 1 1 --solve
```

(Our setting of `max-jacobiators = 1` here makes no difference.) No solution is found. Inspecting the output, we find that the following term in the associator cannot be produced by a first-order differential consequence of the Jacobi identity:



Indeed one can show this graph arises only in a differential consequence of order two. □

**Corollary 13** ( $\star$ -product non-extendability from  $\{\cdot, \cdot\}_{\mathcal{P}}$  to  $\{\cdot, \cdot\}_{\mathcal{P}}$  at order  $\hbar^4$ ). Because there are at least two arrows falling on the object  $\text{Jac}(\mathcal{P})$  in  $\diamond$  at every value of the ten master-parameters  $p_i$ , the associativity can be broken at order  $\hbar^4$  for extensions of the  $\star$ -product to infinite-dimensional set-up<sup>6 on p. 236</sup> of  $N^n$ -valued fields  $\phi \in C^\infty(M^m \rightarrow N^n)$  over a given affine manifold  $M^m$ , of local functionals  $F, G, H$  taking such fields to numbers, and of variational Poisson brackets  $\{\cdot, \cdot\}_{\mathcal{P}}$  on the algebra of local functionals.

Indeed, the Jacobiator  $\text{Jac}(\mathcal{P}) \cong 0$  for a variational Poisson bi-vector  $\mathcal{P}$  is a cohomologically trivial variational tri-vector on the jet space  $J^\infty(M^m \rightarrow N^n)$ , whence the first variation of  $\text{Jac}(\mathcal{P})$  brought on it by a unique arrow would of course be vanishing identically. Nevertheless, that variational tri-vector's density is not necessarily equal to zero on  $J^\infty(M^m \rightarrow N^n)$  over  $M^m$  for those variational Poisson structures whose coefficients  $\mathcal{P}^{ij}$  explicitly depend on the fields  $\phi$  or their derivatives along  $M^m$ . This is why the second and higher variations of the Jacobiator  $\text{Jac}(\mathcal{P})$  would not always vanish. (Such higher-order variations of functionals are calculated by using the techniques from [23, 27].) We know from [12] that  $\text{Assoc}_\star(F, G, H) \cong 0 \pmod{\bar{o}(\hbar^3)}$ , i.e. the associator is trivial up to order  $\hbar^3$  for all variational Poisson brackets  $\{\cdot, \cdot\}_{\mathcal{P}}$  but we now see that it can contain cohomologically nontrivial terms proportional to  $\hbar^4$ . Consequently, it is the order four at which the associativity of  $\star$ -products can start to leak in the course of deformation quantization of Poisson field models.

We now claim that four master-parameters can simultaneously be gauged out of the star-product. (That is, either some of the four or all of them at once can be set equal to zero, although this may not necessarily be their true value given by formula (5).)<sup>17</sup>

<sup>17</sup>Let us recall that the property of a parameter in a family of star-products to be removable by some gauge transformation is not the same as setting such parameter to zero (or any other value). Indeed, other graph coefficients, not depending on the parameter at hand, might get modified by that gauge transformation. However – and similarly to the removal of the loop graph at  $\hbar^2$  in the Kontsevich

**Theorem 14.** *For each  $j \in \{2, 3, 9, 10\}$  there exists a gauge transformation  $\text{id} + \hbar^4 p_j Z_j$  (listed in Table 10 in Appendix E) such that the master-parameter  $p_j$  is reset to zero in the deformed star-product  $\star'$ . This is achieved in such a way that no graph coefficients which initially did not contain the parameter to gauge out would change at all.*

- *Moreover, the gauge transformation  $\text{id} + \hbar^4 \cdot (\sum_j p_j Z_j)$  removes at once all the four master-parameters, still preserving those coefficients of graphs in  $\star$  which did not depend on any of them.*

*Proof scheme.* Let the  $\star$ -product expansion in terms of 10 parameters (obtained in Theorem 9) be contained in `star4_intermsof10.txt`. Construct a gauge transformation of the form  $\text{id} + \hbar^4 G$ , where  $G$  is the sum over all possible graphs with four internal vertices over one sink which are nonzero, without double edges, without tadpoles, and with positive differential order, taken with undetermined coefficients  $g_i$ :

```
$ cat > gauge4.txt
1 0 1          1
h^4:
^D (press Ctrl+D)
$ generate_graphs 4 1 --normal-forms=yes --zero=no \
  --positive-differential-order=yes \
  --with-coefficients=yes >> gauge4.txt
$ sed -i 's/w/g/' gauge4.txt # replace coefficient prefix 'w' by 'g'
```

Obtain gauged star-product expansion  $\star'$  by applying the gauge transformation to  $\star$ :

```
$ gauge star4_intermsof10.txt gauge4.txt \
> star4_intermsof10_gauged_unreduced.txt
```

Reduce the graph series for  $\star'$  modulo skew-symmetry:

```
$ reduce_mod_skew star4_intermsof10_gauged_unreduced.txt \
> star4_intermsof10_gauged.txt
```

Inspect which of the 10 parameters  $p_j$  cannot be gauged out, by checking for the existence of graph coefficients containing  $p_j$  but not any  $g_i$ . For example, for  $p_1 = w\_4\_100$ :

```
$ grep w_4_100 star4_intermsof10_gauged.txt \
| grep -v g | wc -l
```

17

There are 17 graphs with such coefficients, so  $p_1 = w\_4\_100$  cannot be gauged out. Following this procedure for all the 10 parameters, we find that the only candidates to be gauged out are  $p_2 = w\_4\_101$ ,  $p_3 = w\_4\_102$ ,  $p_9 = w\_4\_119$ , and  $p_{10} = w\_4\_125$ . Now inspect the file `star4_intermsof10_gauged.txt` for the lines containing these  $p_j$  and (necessarily, some)  $g_i$ . For each  $p_j$ , find a choice of  $g_i$  so that  $p_j$  is completely removed from the file. (The  $g_i$  will be of the form  $g_i = \alpha_{ij} p_j$  for  $\alpha_{ij} \in \mathbb{R}$ .) It turns out that this is always possible. Hence this choice of  $g_i$  defines the sought-for gauge transformation  $\text{id} + \hbar^4 p_j Z_j$  which gauges out the parameter  $p_j$ . The gauge-transformations which

---

$\star$ -product (see Examples 24 and 25) – the trivialization of four parameters at no extra cost is the case which Theorem 14 states.

kill the (four) parameters separately may be combined into the gauge-transformation  $\text{id} + \hbar^4(\sum_j p_j Z_j)$  that kills all (four) of them simultaneously.  $\square$

*Remark 16.* The master-parameters which we can gauge out are exactly the ones which do not show up in the associativity equation (see Remark 14).

Let us finally address a possible origin of so ample a freedom in the ten-parameter family of star-products (now known up to  $\bar{o}(\hbar^4)$ ). We claim that the mechanism of vanishing via differential consequences of the Jacobi identity, which was recalled in Lemma 10 and used in Theorem 12, starts working not only for the associator built over  $\star$ , but it may even start working for the  $\star$ -product expansion itself.

**Theorem 15.** *The ten-parameter family of star-product expansions  $\star = \dots + \hbar^4(\star^{(0)} + \sum_{i=1}^{10} p_i \star^{(i)}) + \bar{o}(\hbar^4)$  does contain, in the ten-dimensional affine subspace parametrized by  $p_1, \dots, p_{10}$  in  $\mathbb{k}[G_{2,4}]$ , a unique one-dimensional (null or ‘improper’) subspace such that every point  $\alpha \cdot (\star^{(9)} - 2\star^{(6)}) = \alpha \cdot \star^{(9/6)}$  in it admits a Leibniz graph factorization (via the Jacobiator)  $\star^{(9/6)} = \nabla(\mathcal{P}, \text{Jac}(\mathcal{P})) \in \mathbb{k}[G_{2,4}]$ . This null space is the span of the direction  $\mathbf{w\_4\_119} : \mathbf{w\_4\_107} : \dots = 1 : (-2) : 0 : \dots : 0 \in \mathbb{RP}^9$ , that is, the master-parameters  $p_9$  and  $p_6$  occur in proportion  $1 : (-2)$  and all the other  $p_i$ ’s are zero.*

In effect, the respective part of the star-product always cancels out for every given Poisson structure  $\mathcal{P}$ . This factorization and uniqueness of the direction  $\star^{(9/6)}$  is established by using the same computer-assisted scheme of reasoning which worked in the proof of Theorem 12.

#### 4. DISCUSSION

The coefficients of (sometimes different, sometimes gauge-inequivalent) star-product expansions up to low orders were previously obtained in the papers [21, 35, 1, 6, 38]. Let us compare the result in this paper with those publications, and let us use the software described in this paper to verify some results about other star-products.

**4.1. Previously known weights.** The values of some (families of) Kontsevich graph weights are given in the literature. The graphs in the Bernoulli family have scaled Bernoulli numbers as weights (see [6, Corollary 6.3] or [21, Proposition 4.4.1]), e.g.  $\mathbf{w\_3\_2} = B_3/3! = 0$  and  $\mathbf{w\_4\_12} = B_4/4! = -1/720$ . The weights of a family of graphs containing cycles are obtained in [6, Corollary 6.3], e.g.  $\mathbf{w\_3\_9} = \pm B_3/(2 \cdot 3!) = 0$  and  $\mathbf{w\_4\_72} = -B_4/(2 \cdot 4!) = 1/1440$ . Willwacher states in [38] the vanishing of three graph weights at the order 3 (they are  $\mathbf{w\_3\_7}$ ,  $\mathbf{w\_3\_13}$ ,  $\mathbf{w\_3\_14}$  in Figure 2) and the non-vanishing one other (it is  $\mathbf{w\_3\_12}$  in Figure 2); this agrees with our calculation in Example 26.

**4.2. Numerical approximation.** In Tables 2 and 3 in Appendix A.1 we list numerical approximations of several weights. These approximations are consistent with the exact weights (and relations) obtained in this paper.

**4.3. Independent symbolic calculation.** The values for the weights found in this paper agree with a symbolic calculation of the graph weights reported by Pym and Panzer [34] and reproduced in Table 4 on p. 280 in Appendix A.2.

**4.4. The obstruction to the existence of a loopless star product.** In [38], Willwacher establishes that any universal star-product (defined by the Kontsevich graphs, possibly with different coefficients) which is gauge-equivalent to Kontsevich's  $\star$ -product *must* contain graphs with 2-cycles. To obtain the same result using our software, we proceed as follows.

**Example 29.** We start with Kontsevich's  $\star$ -product up to  $\hbar^3$  in `star3.txt`. The unique graph with a loop at order 2 can be removed by extending the gauge transformation from Example 23 which was stored in `gaugeloop.txt`:

```
$ cp gaugeloop.txt gaugeloop3.txt
$ echo "h^3:" >> gaugeloop3.txt
$ gauge star3.txt gaugeloop3.txt > star3_gauge2_unreduced.txt
$ reduce_mod_skew star3_gauge2_unreduced.txt > star3_gauge2.txt
```

The gauged  $\star$ -product is obtained in `star3_gauge2.txt`:

```
h^0:
2 0 1      1
h^1:
2 1 1  0 1  1
h^2:
2 2 1  0 1 1 2  -1/3
2 2 1  0 1 0 2   1/3
2 2 1  0 1 0 1   1/2
h^3:
2 3 1  0 1 1 2 1 2   1/6
2 3 1  0 1 0 1 1 2  -1/3
2 3 1  0 1 0 2 0 2   1/6
2 3 1  0 1 0 1 0 2   1/3
2 3 1  0 1 0 1 0 1   1/6
2 3 1  0 3 1 2 2 3  -1/6
2 3 1  0 1 2 4 2 3   1/12
2 3 1  0 1 0 2 1 3  -1/6
2 3 1  0 1 0 4 1 2  -1/6
2 3 1  0 3 1 4 1 3  -1/6
2 3 1  0 1 1 2 2 3  -1/6
2 3 1  0 3 1 2 1 2   1/6
2 3 1  0 1 1 4 2 3   1/6
2 3 1  0 3 0 2 1 2   1/6
2 3 1  0 1 0 2 2 3  -1/6
2 3 1  0 3 1 2 0 3  -1/6
2 3 1  0 1 0 4 2 3   1/6
```

Willwacher denotes this  $\star$ -product by  $a = a_0 + a_1 + a_2 + a_3 + \dots$ , and supposes that our (desirably loopless)  $\star$ -product reads  $b = a_0 + a_1 + a_2 + (a_3 + b_3) + (a_4 + b_4) + \dots$ . The Maurer-Cartan associativity equation  $[b, b] = 0$  implies in particular  $[a_0, b_3] = 0$  and  $[a_1, b_3] + [a_0, b_4] = 0$  (here  $[-, -]$  is the Gerstenhaber bracket).

**Claim.** *For any solution of  $[b, b] = 0$ , the sum  $a_3 + b_3$  contains graphs with cycles.*



We carry out Willwacher's proof, with some minor corrections. Since  $b_3$  is a Hochschild cocycle it suffices to assume  $b_3$  is in the image of the (graphical) Hochschild-Kostant-Rosenberg map, so it is a skew-symmetric bi-derivation.<sup>18</sup> There are two terms  $-\alpha A$  and  $-\beta B$  in  $a_3$  ( $\alpha, \beta \neq 0$ ) which are skew-symmetric bi-derivations and graphs with cycles. So for  $a_3 + b_3$  to have no cycles,  $b_3$  must be the linear combination  $b_3 = \alpha A + \beta B$ . The proof proceeds by showing that  $[a_1, b_3] + [a_0, b_4] = 0$  cannot hold: indeed, simplifying the 3-cochain  $[a_1, b_3]$  modulo the image of  $[a_0, -]$  and the Jacobi identity results in a sum of graphs (called Shoikhet's obstruction) which does not vanish. Let us illustrate all of this in detail.

**Example 30.** The skew bi-derivation terms in  $a_3$  are:

```
2 3 1  0 1 2 4 2 3    1/12
2 3 1  0 3 1 2 2 3    -1/6
```

(We have  $\alpha = -1/12$  and  $\beta = 1/6$ .) We store the correction term  $b_3$  in the file `b3.txt`:

```
2 3 1  0 1 2 4 2 3    -1/12
2 3 1  0 3 1 2 2 3     1/6
```

Calculate the Gerstenhaber bracket  $[a_1, b_3]$  (using  $a_1$  in `wedge.txt`)<sup>19</sup>:

```
$ echo '2 1 1  0 1  1' > wedge.txt
$ gerstenhaber_bracket wedge.txt b3.txt > \[wedge,b3\]_unreduced.txt
$ reduce_mod_skew \[wedge,b3\]_unreduced.txt > \[wedge,b3\].txt
```

Generate an ansatz for  $b_4$ :

```
$ generate_graphs 4 --normal-forms=yes --with-coefficients=yes \
  --positive-differential-order=yes --zero=no > b4.txt
```

Calculate  $[a_0, b_4]$  (using  $a_0$  in `dotdot.txt`):

```
$ echo '2 0 1  1' > dotdot.txt
$ gerstenhaber_bracket dotdot.txt b4.txt > \[dotdot,b4\]_unreduced.txt
$ reduce_mod_skew \[dotdot,b4\]_unreduced.txt > \[dotdot,b4\].txt
```

Store Shoikhet's obstruction with undetermined coefficients  $A, B$  in `shoikhet_obs.txt`:

```
3 4 1  0 1 2 3 3 4 3 4  A
3 4 1  2 1 0 3 3 4 3 4  -A
3 4 1  0 1 2 3 3 4 4 5  B
3 4 1  2 1 0 3 3 4 4 5  -B
```

Add  $[a_0, b_4]$  and Shoikhet's obstruction to  $[a_1, b_3]$ :

```
$ cat \[wedge,b3\].txt \[dotdot,b4\].txt shoikhet_obs.txt \
  > \[wedge,b3\]+\[dotdot,b4\]+shoikhet_obs_unreduced.txt
$ reduce_mod_skew \
  \[wedge,b3\]+\[dotdot,b4\]+shoikhet_obs_unreduced.txt \
  > \[wedge,b3\]+\[dotdot,b4\]+shoikhet_obs.txt
```

<sup>18</sup>However, this does not imply that each individual graph in it is a skew-symmetric bi-derivation. Rather, each graph which is a bi-derivation can be skew-symmetrized, which yields either the original graph or the sum of two graphs which are mirror-reflections of each other. It is clear that this is what Willwacher intended, e.g. because the mirror-reflection of his graph  $D$  is not drawn.

<sup>19</sup>The graphical calculation of  $[a_1, b_3]$  in [38] contains errors, e.g. the first graph has a vertex with three outgoing edges and the term with coefficient  $\beta$  has arrows in the wrong direction.

Reduce modulo the Jacobi identity and solve (for the expression to be equal to zero):

```
$ reduce_mod_jacobi \[wedge,b3\]+\[dotdot,b4\]+shoikhet_obs.txt \
1 10 --solve
```

Indeed, there is a solution  $A = \beta = 1/6$  and  $B = -4(\alpha + \beta) = -1/3$ . So, modulo the image of  $[a_0, -]$  and the Jacobi identity,  $[a_1, b_3]$  is equal to Shoikhet’s obstruction with  $A = -\beta = -1/6$  and  $B = 4(\alpha + \beta) = 1/3$  (the sign changed because we added Shoikhet’s obstruction instead of subtracting it).<sup>20</sup>

**Example 31.** An example of a Poisson structure for which Shoikhet’s obstruction doesn’t vanish is given by 3d-polynomial:

```
$ poisson_evaluate shoikhet_obs.txt 3d-polynomial
...
# [ x ] [ x ] [ y ]
-4*A*y^3*z^2*x^2+2*y^3*B*z*x^3-2*y*B*z^3*x^3+y*B*z^4*x^2+...
...
```

For example, the coefficient of the differential monomial  $x^2y^3z^2\partial_x \otimes \partial_x \otimes \partial_y$  is  $-4A \neq 0$ .

In this section we traced Willwacher’s steps. There is a much simpler proof of the claim when all the coefficients of graphs in  $a_3$  are known (which was not the case in [38]): there are loopful graphs with nonzero coefficients in  $a_3$  which cannot be gauged out.

**4.5. Penkava–Vanhaecke deformations.** In [35] M. Penkava and P. Vanhaecke give (among other things) deformations  $\pi_\star = \pi + h\pi_1 + h^2\pi_2 + \dots + h^n\pi_n + \bar{o}(h^n)$  where  $\pi$  is the pointwise product,  $\pi_1$  is the Poisson bracket,  $h$  is the formal parameter, and associativity holds modulo  $\bar{o}(h^n)$  for arbitrary *polynomial* Poisson algebras. Note that every  $\star$ -product (which is associative as a formal power series in  $h$ ) induces such an expansion modulo  $\bar{o}(h^n)$  for every  $n$ , but not every deformation modulo  $\bar{o}(h^n)$  can be extended to higher orders.<sup>21</sup> Indeed, Penkava–Vanhaecke exhibit deformations which can be extended and some which cannot be extended. (Namely, already at order 3 there exist formulas which do not extend to higher orders — although such formulas are clearly not the genuine Kontsevich star-product.) In the following sequence of examples, we verify some of their results and compare them with ours.

**Example 32.** Proposition 5.1 in [35] gives a deformation  $\pi + h\pi_1 + h^2\pi_2 + \bar{o}(h^2)$ , and in fact it coincides with Kontsevich’s  $\star$ -product modulo  $\bar{o}(h^2)$  with the loop graph gauged out; see Example 25 in this text.

**Example 33.** Theorem 5.6 in [35] provides a deformation  $\pi_\star = \pi + h\pi_1 + h^2\pi_2 + h^3\pi_3 + \bar{o}(h^3)$ . The differential polynomials in it can be viewed as Kontsevich graphs; we store their encodings with their numerical coefficients in `star3pv5.6.txt`:

```
h^0:
2 0 1                               1
```

<sup>20</sup>The solution reported here differs from Willwacher’s, not in sign (which is left ambiguous in [38]) but in proportion: he claims  $A = \pm 2\beta$  and  $B = \pm 2(\alpha + \beta)$ .

<sup>21</sup>Note that this problem is different from the computation of obstructions to Kontsevich’s Formality [32, 31]. Specifically, in “Formality Conjecture” [31], Kontsevich reports the absence of obstructions to Formality up to  $n \leq 6$ . Formality is now a theorem: Kontsevich’s  $\star$ -product exists at all orders.

```

h^1:
2 1 1 0 1 1
h^2:
2 2 1 0 1 1 2 -1/3
2 2 1 0 1 0 2 1/3
2 2 1 0 1 0 1 1/2
h^3:
2 3 1 0 1 1 2 2 3 -1/3
2 3 1 0 1 0 2 2 3 -1/3
2 3 1 0 1 0 1 0 1 1/6
2 3 1 0 1 0 2 1 3 -1/6
2 3 1 0 1 0 4 1 2 -1/6
2 3 1 0 1 1 2 1 2 1/6
2 3 1 0 1 0 2 0 2 1/6
2 3 1 0 1 0 1 0 2 1/3
2 3 1 0 1 0 1 1 2 -1/3

```

Calculate the associator in terms of graphs (see Implementation 12):

```
$ star_product_associator star3pv5.6.txt > assoc3pv5.6.txt
```

It vanishes as a consequence of the Jacobi identity (see Implementation 16):

```
$ reduce_mod_jacobi assoc3pv5.6.txt
```

Hence  $\pi_*$  is associative modulo  $\bar{o}(h^3)$  (for arbitrary Poisson structures on  $\mathbb{R}^d$ ). This deformation is not equal to Kontsevich's  $\star$ -product modulo  $\bar{o}(h^3)$ , nor is it Kontsevich's  $\star$ -product with the loop graph gauged out, but the next example gives the explicit relation between this product and Kontsevich's.

**Example 34.** Theorem 5.6 in [35] further relates arbitrary deformations to  $\pi_* = \pi + h\pi_1 + h^2\pi_2 + h^3\pi_3 + \bar{o}(h^3)$  from Example 33. Namely, every deformation modulo  $\bar{o}(h^3)$  is gauge-equivalent to  $\tilde{\pi}_* = \pi + h\pi_1 + h^2(\pi_2 + \varphi_2) + h^3(\pi_3 + \varphi_3 + \psi_3)$  for some choice of  $(\varphi_2, \varphi_3, \psi_3)$ , where  $\varphi_2$  and  $\varphi_3$  are antisymmetric biderivations and  $\psi_3$  is a symmetric 2-cochain satisfying  $\partial\psi_3 = [\pi_1, \varphi_2]$ . Let us show that this holds for Kontsevich's  $\star$ -product expansion  $\pi_*^K \bmod \bar{o}(h^3)$ . In Section 4.4 we obtained Kontsevich's  $\star$ -product with the loop graph gauged out; let us denote it by  $\tilde{\pi}_*^K \bmod \bar{o}(h^3)$ . Up to  $\bar{o}(h^2)$  the deformations  $\tilde{\pi}_*^K$  and  $\pi_*$  are equal (as we observed in Example 32), so we choose  $\varphi_2 = 0$ . Subtracting  $\pi_*$  from  $\tilde{\pi}_*^K$  yields the file `star3_gauge2_minus_pv5.6.txt`:

```

h^3:
# 1 1
2 3 1 0 3 1 2 2 3 -1/6
2 3 1 0 1 2 4 2 3 1/12
# 1 2
2 3 1 0 3 1 4 1 3 -1/6
2 3 1 0 1 1 2 2 3 1/6
2 3 1 0 3 1 2 1 2 1/6
2 3 1 0 1 1 4 2 3 1/6
# 2 1
2 3 1 0 3 0 2 1 2 1/6

```

```

2 3 1  0 1 0 2 2 3   1/6
2 3 1  0 3 1 2 0 3  -1/6
2 3 1  0 1 0 4 2 3   1/6

```

and it can be seen that it is antisymmetric, so this must be  $h^3\varphi_3$  and hence  $\psi_3 = 0$ . But the terms are not all of differential order  $(1, 1)$ , so how can  $\varphi_3$  be a bi-derivation? The answer is that all other terms vanish due to two first-order differential consequences of the Jacobi identity for the Poisson structure. In other words, there is a Leibniz graph – with one arrow incoming on the Jacobiator – which expands to the homogeneous component of order  $(1, 2)$ , and naturally the mirror-reflection of that Leibniz graph expands to the order  $(2, 1)$  component. This can be verified using `reduce_mod_jacobi`.

**Example 35.** Theorem 6.1 in [35] gives the obstruction to extending  $\pi_*$  from Example 33 to the fourth order. The proof shows that this obstruction is the skew-symmetrization of the degree- $(1, 1, 1)$  homogeneous component of the associator at  $h^4$ . We reproduce this as follows. First create a file representing  $\pi_* \bmod \bar{o}(h^4)$ :

```

$ cp star3pv5.6.txt star4pv6.1.txt
$ echo "h^4:" >> star4pv6.1.txt

```

Calculate the associator:

```

$ star_product_associator star4pv6.1.txt > assoc4pv6.1.txt

```

Skew-symmetrize (see §4.7 below):

```

$ skew_symmetrize assoc4pv6.1.txt > obs4pv6.1_unreduced.txt

```

Reduce modulo skew-symmetry:

```

$ reduce_mod_skew obs4pv6.1_unreduced.txt \
  --print-differential-orders > obs4pv6.1.txt

```

Finally, we see that the degree- $(1, 1, 1)$  homogeneous component at  $h^4$  is

```

3 4 1  0 1 2 3 3 4 4 5   4/3
3 4 1  0 2 1 3 3 4 4 5  -4/3
3 4 1  0 4 1 2 3 4 3 5  -4/3
3 4 1  0 1 2 3 3 4 3 4  -2/3
3 4 1  0 2 1 3 3 4 3 4   2/3
3 4 1  0 4 1 2 3 4 3 4  -2/3

```

which is (up to an irrelevant constant factor) the sum of six terms written in Theorem 6.1. We store this component in the file `obs4pv6.1_111.txt` and the others in `obs4pv6.1_rest.txt`. The latter vanish as a consequence of the Jacobi identity:

```

$ reduce_mod_jacobi obs4pv6.1_rest.txt

```

as claimed in Theorem 6.1. The  $(1, 1, 1)$ -component does not vanish in general: indeed,

```

$ reduce_mod_jacobi obs4pv6.1_111.txt

```

does not find any solution. An explicit Poisson structure for which the obstruction does not vanish is given at the end of [35, §9]. We can do the same computation in our software: the respective Poisson structure was added under the name `4d-pv`, so that

```

$ poisson_evaluate obs4pv6.1_111.txt 4d-pv

```

shows a multi-vector field which is not identically zero.

**Example 36.** Finally Lemma 8.2 in [35] gives the correction term  $\varphi_3$  to  $\pi_3$  for the deformation to extend to the fourth order. In terms of graph encodings with coefficients,  $\varphi_3$  is:

$$\begin{array}{rcl} 2 & 3 & 1 \quad 0 \quad 3 \quad 1 \quad 2 \quad 2 \quad 3 \quad -1/6 \\ 2 & 3 & 1 \quad 0 \quad 1 \quad 2 \quad 4 \quad 2 \quad 3 \quad 1/12 \end{array}$$

which is exactly the correction term we found in Example 34 to make the deformation equal to Kontsevich’s  $\star$ -product modulo  $\bar{o}(\hbar^3)$  with the loop graph gauged out. This proves that the deformation extends to the fourth order. Alternatively, we can store the full expansion in `star3pv8.2.txt` and confirm that it extends to the order 4 as follows. First, add graphs with undetermined coefficients at  $\hbar^4$ :

```
$ cp star3pv8.2.txt star4pv8.2.txt
$ echo 'h^4:' >> star4pv8.2.txt
$ generate_graphs 4 --normal-forms=yes --with-coefficients=yes \
  >> star4pv8.2.txt
```

Calculate the graphical associator:

```
$ star_product_associator star4pv8.2.txt > assoc4pv8.2.txt
```

Finally, run

```
$ reduce_mod_jacobi assoc4pv8.2.txt 1 2 --solve
```

and observe that there is a solution.

**4.6. Universal star-products.** We do work on affine Poisson manifolds, so that formulae are coordinate-independent because of the contraction of upper versus lower indices in all tensor objects *and* because all the Jacobians are constant in the course of affine coordinate reparametrizations. S. Gutt et al in [1] provide star-products modulo  $\bar{o}(\hbar^3)$  which are universal with respect to all Poisson structures  $\mathcal{P}$  on all smooth finite-dimensional manifolds  $\mathcal{M}^d$  equipped with a torsion-free, not necessarily flat, linear connection  $\nabla$ . Then the formula of  $\star \bmod \bar{o}(\hbar^3)$  is expressed in terms of differential polynomials in not only the bi-vector  $\mathcal{P}$  – clearly, our  $\partial_i$  replaced by  $\nabla_i$  in every instance – but also in the *curvature*  $R$  of  $\nabla$ .

**Example 37.** To compare with Kontsevich’s formula up to  $\bar{o}(\hbar^3)$  which is given in the present paper (also in [12]), we can put  $R = 0$  in the formula by Gutt et al. The terms up to  $\bar{o}(\hbar^2)$  clearly match. A-priori there are  $(5 - 1) \times (5 - 1) = 16$  terms with coefficient  $-1/6$  at  $\hbar^3$ . Two pairs of terms double, and they become two terms with coefficients  $\pm 1/3$ . One term vanishes identically, because it is the zero graph from Example 5. The resulting 13 terms are exactly those in Kontsevich’s formula (1) at  $\hbar^3$ . Hence the formula obtained by Gutt et al. restricted to  $R = 0$  coincides with Kontsevich’s  $\star$ -product up to  $\bar{o}(\hbar^3)$ .

It would be interesting to recover such a univereal formula  $\star(\mathcal{P}, R)$  – depending also on the curvature  $R$  – modulo  $\bar{o}(\hbar^4)$  and beyond.

**4.7. Universal flows on spaces of Poisson structures.** The software presented in this paper has been extended to operate on first-order differential operators which represent (skew-symmetric) multi-vector fields. In particular skew-symmetrization was implemented in `skew_symmetrize` and the graphical Schouten bracket was implemented

in `schouten_bracket`. This has been applied by the authors jointly with A. Bouis-aghoulane in [7] to confirm the existence of a universal flow on the spaces of Poisson structures, which was suggested by Kontsevich. The explicit mechanism that explains why these universal flows exist, based on work by Kontsevich, Willwacher, and Jost, is given by the authors in [13].

**4.8. Open problems.** The following two questions, posed by M. Kontsevich (private communication) can be approached up to finite orders in  $\hbar$  by using the software modules which we have presented:

- Which quadratic weight relations are determined by the associativity alone? (We refer to the preprint [2, p. 61] for discussion.)
- How many degrees of freedom in the graph weights (at a fixed order in  $\hbar$ ) are due to gauge transformations?

Independently (Kevin Morand, private communication), an open problem is to describe the action of the graph complex (with suitable cocycles  $\gamma \in \ker[\bullet\bullet, -]$ ) on the  $\star$ -product under the infinitesimal symmetries  $\hbar\mathcal{P} \mapsto \hbar\mathcal{P} + \varepsilon \text{Or}(\gamma)(\hbar\mathcal{P}) + \bar{o}(\varepsilon)$  of Poisson structures (see [7, 8, 13] and [29, 30, 31]).

For a long time, the third and fourth order expansion of Kontsevich's  $\star$ -product was unknown to the physics community, which may have delayed the implementation of deformation quantization in the study of Nature. No model of that theory could be tested approximately because it could not be known what any such model actually was. This is why we present the formula  $\star \bmod \bar{o}(\hbar^4)$  in Eq. (13) on pp. 280–284 in this paper.











$$\begin{aligned}
& + \left(-\frac{1}{90} - 8p_6 - 16p_4 - 24p_5 + 8p_1 - 8p_7\right) \partial_m \partial_\ell \mathcal{P}^{ij} \partial_p \mathcal{P}^{kl} \partial_q \mathcal{P}^{mn} \partial_n \mathcal{P}^{pq} \partial_i f \partial_k \partial_j g \\
& + \left(-\frac{1}{15} + 8p_8 + 8p_4 + 8p_2 + 16p_{10} - 16p_7\right) \partial_m \mathcal{P}^{ij} \partial_p \partial_j \mathcal{P}^{kl} \partial_q \partial_\ell \mathcal{P}^{mn} \partial_n \mathcal{P}^{pq} \partial_i f \partial_k g \\
& + \left(-\frac{1}{15} + 8p_8 + 8p_4 + 8p_2 + 16p_{10} - 16p_7\right) \partial_p \partial_n \mathcal{P}^{ij} \partial_q \partial_j \mathcal{P}^{kl} \partial_k \mathcal{P}^{mn} \partial_\ell \mathcal{P}^{pq} \partial_i f \partial_m g \\
& + \left(\frac{1}{20} - 8p_8 + 24p_4 - 16p_5 - 8p_2 + 8p_7\right) \partial_k \mathcal{P}^{ij} \partial_q \partial_m \partial_j \mathcal{P}^{kl} \partial_\ell \mathcal{P}^{mn} \partial_n \mathcal{P}^{pq} \partial_i f \partial_p g \\
& + \left(-\frac{1}{20} + 8p_8 - 24p_4 + 16p_5 + 8p_2 - 8p_7\right) \partial_p \mathcal{P}^{ij} \partial_q \partial_n \partial_j \mathcal{P}^{kl} \partial_k \mathcal{P}^{mn} \partial_\ell \mathcal{P}^{pq} \partial_i f \partial_m g \\
& + \left(-\frac{1}{40} + 8p_8 + 16p_4 + 8p_5 + 16p_{10} - 12p_7\right) \partial_m \mathcal{P}^{ij} \partial_p \partial_n \partial_j \mathcal{P}^{kl} \partial_q \partial_\ell \mathcal{P}^{mn} \mathcal{P}^{pq} \partial_i f \partial_k g \\
& + \left(\frac{1}{40} - 8p_8 - 16p_4 - 8p_5 - 16p_{10} + 12p_7\right) \partial_p \partial_n \partial_k \mathcal{P}^{ij} \partial_q \partial_j \mathcal{P}^{kl} \partial_\ell \mathcal{P}^{mn} \mathcal{P}^{pq} \partial_i f \partial_m g \\
& + \left(\frac{11}{90} + 8p_6 - 16p_4 + 40p_5 - 8p_1 + 24p_7\right) \partial_p \partial_m \mathcal{P}^{ij} \partial_q \partial_n \mathcal{P}^{kl} \partial_\ell \mathcal{P}^{mn} \mathcal{P}^{pq} \partial_k \partial_i f \partial_j g \\
& + \left(-\frac{11}{90} - 8p_6 + 16p_4 - 40p_5 + 8p_1 - 24p_7\right) \partial_p \partial_m \mathcal{P}^{ij} \partial_q \partial_n \mathcal{P}^{kl} \partial_\ell \mathcal{P}^{mn} \mathcal{P}^{pq} \partial_i f \partial_k \partial_j g \\
& + \left(\frac{1}{5} - 32p_8 - 48p_5 - 32p_{10} + 16p_1 + 48p_7\right) \partial_p \partial_n \mathcal{P}^{ij} \partial_q \partial_j \mathcal{P}^{kl} \partial_\ell \mathcal{P}^{mn} \mathcal{P}^{pq} \partial_k \partial_i f \partial_m g \\
& + \left(\frac{1}{5} - 32p_8 - 48p_5 - 32p_{10} + 16p_1 + 48p_7\right) \partial_n \mathcal{P}^{ij} \partial_p \partial_j \mathcal{P}^{kl} \partial_q \partial_\ell \mathcal{P}^{mn} \mathcal{P}^{pq} \partial_i f \partial_m \partial_k g \\
& + \left(-\frac{1}{6} + 16p_8 - 16p_3 + 32p_4 - 16p_1 - 32p_7\right) \partial_p \mathcal{P}^{ij} \partial_j \mathcal{P}^{kl} \partial_q \partial_\ell \mathcal{P}^{mn} \partial_n \mathcal{P}^{pq} \partial_m \partial_i f \partial_k g \\
& + \left(\frac{1}{6} - 16p_8 + 16p_3 - 32p_4 + 16p_1 + 32p_7\right) \partial_q \mathcal{P}^{ij} \partial_m \partial_j \mathcal{P}^{kl} \partial_\ell \mathcal{P}^{mn} \partial_n \mathcal{P}^{pq} \partial_i f \partial_p \partial_k g \\
& + \left(\frac{1}{9} - 16p_8 + 16p_4 - 32p_5 + 16p_1 + 16p_7\right) \partial_m \mathcal{P}^{ij} \partial_n \partial_j \mathcal{P}^{kl} \partial_q \partial_\ell \mathcal{P}^{mn} \mathcal{P}^{pq} \partial_i f \partial_p \partial_k g \\
& + \left(-\frac{1}{9} + 16p_8 - 16p_4 + 32p_5 - 16p_1 - 16p_7\right) \partial_n \partial_k \mathcal{P}^{ij} \partial_q \partial_j \mathcal{P}^{kl} \partial_\ell \mathcal{P}^{mn} \mathcal{P}^{pq} \partial_p \partial_i f \partial_m g \\
& + \left(-\frac{1}{9} + 16p_8 - 32p_4 + 48p_5 + 16p_2 - 16p_7\right) \partial_m \mathcal{P}^{ij} \partial_j \mathcal{P}^{kl} \partial_q \partial_\ell \mathcal{P}^{mn} \partial_n \mathcal{P}^{pq} \partial_p \partial_i f \partial_k g \\
& + \left(-\frac{1}{9} + 16p_8 - 32p_4 + 48p_5 + 16p_2 - 16p_7\right) \partial_n \mathcal{P}^{ij} \partial_q \partial_j \mathcal{P}^{kl} \partial_k \mathcal{P}^{mn} \partial_\ell \mathcal{P}^{pq} \partial_i f \partial_p \partial_m g \\
& + \left(\frac{1}{9} - 16p_8 + 32p_4 - 48p_5 + 16p_2 + 16p_7\right) \partial_m \mathcal{P}^{ij} \partial_j \mathcal{P}^{kl} \partial_q \partial_\ell \mathcal{P}^{mn} \partial_n \mathcal{P}^{pq} \partial_i f \partial_p \partial_k g \\
& + \left(-\frac{1}{9} + 16p_8 - 32p_4 + 48p_5 - 16p_2 - 16p_7\right) \partial_k \mathcal{P}^{ij} \partial_q \partial_j \mathcal{P}^{kl} \partial_\ell \mathcal{P}^{mn} \partial_n \mathcal{P}^{pq} \partial_p \partial_i f \partial_m g \\
& + \left(\frac{7}{90} - 16p_8 + 40p_4 - 40p_5 + 8p_2 + 12p_7\right) \partial_k \mathcal{P}^{ij} \partial_p \partial_j \mathcal{P}^{kl} \partial_q \partial_\ell \mathcal{P}^{mn} \partial_n \mathcal{P}^{pq} \partial_i f \partial_m g \\
& + \left(\frac{7}{90} - 16p_8 + 40p_4 - 40p_5 + 8p_2 + 12p_7\right) \partial_m \partial_k \mathcal{P}^{ij} \partial_j \mathcal{P}^{kl} \partial_q \partial_\ell \mathcal{P}^{mn} \partial_n \mathcal{P}^{pq} \partial_i f \partial_p g \\
& + \left(\frac{1}{180} - 16p_8 - 16p_4 - 16p_5 - 32p_{10} + 8p_7\right) \partial_n \mathcal{P}^{ij} \partial_p \partial_j \mathcal{P}^{kl} \partial_q \partial_\ell \mathcal{P}^{mn} \mathcal{P}^{pq} \partial_k \partial_i f \partial_m g \\
& + \left(-\frac{1}{180} + 16p_8 + 16p_4 + 16p_5 + 32p_{10} - 8p_7\right) \partial_p \partial_n \mathcal{P}^{ij} \partial_j \mathcal{P}^{kl} \partial_q \partial_\ell \mathcal{P}^{mn} \mathcal{P}^{pq} \partial_i f \partial_m \partial_k g \\
& + \left(\frac{37}{90} - 48p_8 - 16p_6 + 96p_4 - 96p_5 + 48p_7\right) \partial_p \mathcal{P}^{ij} \partial_q \partial_n \mathcal{P}^{kl} \partial_\ell \mathcal{P}^{mn} \mathcal{P}^{pq} \partial_k \partial_i f \partial_m \partial_j g \\
& + \left(-\frac{37}{90} + 48p_8 + 16p_6 - 96p_4 + 96p_5 - 48p_7\right) \partial_p \mathcal{P}^{ij} \partial_n \mathcal{P}^{kl} \partial_q \partial_\ell \mathcal{P}^{mn} \mathcal{P}^{pq} \partial_k \partial_i f \partial_m \partial_j g \\
& + \left(\frac{29}{360} - 16p_8 - 16p_4 - 16p_{10} + 8p_1 + 20p_7\right) \partial_p \partial_m \mathcal{P}^{ij} \partial_n \partial_j \mathcal{P}^{kl} \partial_q \partial_\ell \mathcal{P}^{mn} \mathcal{P}^{pq} \partial_i f \partial_k g \\
& + \left(\frac{29}{360} - 16p_8 - 16p_4 - 16p_{10} + 8p_1 + 20p_7\right) \partial_n \partial_k \mathcal{P}^{ij} \partial_p \partial_j \mathcal{P}^{kl} \partial_q \partial_\ell \mathcal{P}^{mn} \mathcal{P}^{pq} \partial_i f \partial_m g \\
& + \left(\frac{34}{45} - 96p_8 - 32p_6 + 240p_4 - 288p_5 + 96p_7\right) \partial_p \mathcal{P}^{ij} \partial_q \mathcal{P}^{kl} \partial_\ell \mathcal{P}^{mn} \partial_n \mathcal{P}^{pq} \partial_k \partial_i f \partial_m \partial_j g \\
& + \left(-\frac{34}{45} + 96p_8 + 32p_6 - 240p_4 + 288p_5 - 96p_7\right) \partial_m \mathcal{P}^{ij} \partial_q \mathcal{P}^{kl} \partial_\ell \mathcal{P}^{mn} \partial_n \mathcal{P}^{pq} \partial_k \partial_i f \partial_p \partial_j g \\
& + \left(-\frac{2}{45} + 8p_9 + 4p_6 - 8p_3 + 4p_5 - 4p_1 - 4p_7\right) \partial_p \mathcal{P}^{ij} \partial_q \partial_m \partial_j \mathcal{P}^{kl} \partial_\ell \mathcal{P}^{mn} \partial_n \mathcal{P}^{pq} \partial_i f \partial_k g \\
& + \left(\frac{2}{45} - 8p_9 - 4p_6 + 8p_3 - 4p_5 + 4p_1 + 4p_7\right) \partial_q \partial_m \partial_k \mathcal{P}^{ij} \partial_j \mathcal{P}^{kl} \partial_\ell \mathcal{P}^{mn} \partial_n \mathcal{P}^{pq} \partial_i f \partial_p g \\
& + \left(\frac{1}{30} + 8p_6 + 32p_4 + 8p_5 + 32p_{10} - 8p_1 + 8p_7\right) \partial_p \partial_n \mathcal{P}^{ij} \partial_j \mathcal{P}^{kl} \partial_q \partial_\ell \mathcal{P}^{mn} \mathcal{P}^{pq} \partial_k \partial_i f \partial_m g \\
& + \left(-\frac{1}{30} - 8p_6 - 32p_4 - 8p_5 - 32p_{10} + 8p_1 - 8p_7\right) \partial_p \partial_n \mathcal{P}^{ij} \partial_q \partial_j \mathcal{P}^{kl} \partial_\ell \mathcal{P}^{mn} \mathcal{P}^{pq} \partial_i f \partial_m \partial_k g
\end{aligned}$$

$$\begin{aligned}
& + \left(-\frac{7}{90} + 8p_8 - 16p_3 + 16p_4 - 8p_5 - 8p_1 - 16p_7\right) \partial_p \mathcal{P}^{ij} \partial_m \partial_j \mathcal{P}^{kl} \partial_q \partial_\ell \mathcal{P}^{mn} \partial_n \mathcal{P}^{pq} \partial_i f \partial_k g \\
& \quad + \left(\frac{7}{90} - 8p_8 + 16p_3 - 16p_4 + 8p_5 + 8p_1 + 16p_7\right) \partial_q \partial_k \mathcal{P}^{ij} \partial_m \partial_j \mathcal{P}^{kl} \partial_\ell \mathcal{P}^{mn} \partial_n \mathcal{P}^{pq} \partial_i f \partial_p g \\
& + \left(-\frac{7}{180} + 16p_8 - 8p_6 - 16p_4 + 8p_5 + 8p_1 - 16p_7\right) \partial_m \mathcal{P}^{ij} \partial_n \partial_j \mathcal{P}^{kl} \partial_q \partial_\ell \mathcal{P}^{mn} \mathcal{P}^{pq} \partial_p \partial_i f \partial_k g \\
& \quad + \left(\frac{7}{180} - 16p_8 + 8p_6 + 16p_4 - 8p_5 - 8p_1 + 16p_7\right) \partial_n \partial_k \mathcal{P}^{ij} \partial_q \partial_j \mathcal{P}^{kl} \partial_\ell \mathcal{P}^{mn} \mathcal{P}^{pq} \partial_i f \partial_p \partial_m g \\
& \quad + \left(\frac{13}{360} - 8p_8 + 24p_4 - 32p_5 - 8p_2 + 8p_1 + 4p_7\right) \partial_m \partial_k \mathcal{P}^{ij} \partial_q \partial_j \mathcal{P}^{kl} \partial_\ell \mathcal{P}^{mn} \partial_n \mathcal{P}^{pq} \partial_i f \partial_p g \\
& \quad + \left(-\frac{13}{360} + 8p_8 - 24p_4 + 32p_5 + 8p_2 - 8p_1 - 4p_7\right) \partial_p \mathcal{P}^{ij} \partial_n \partial_j \mathcal{P}^{kl} \partial_q \partial_k \mathcal{P}^{mn} \partial_\ell \mathcal{P}^{pq} \partial_i f \partial_m g \\
& \quad + \left(\frac{1}{15} - 32p_8 + 8p_6 + 48p_4 - 72p_5 + 24p_1 + 24p_7\right) \partial_p \mathcal{P}^{ij} \partial_n \partial_j \mathcal{P}^{kl} \partial_q \partial_\ell \mathcal{P}^{mn} \mathcal{P}^{pq} \partial_k \partial_i f \partial_m g \\
& \quad + \left(-\frac{1}{15} + 32p_8 - 8p_6 - 48p_4 + 72p_5 - 24p_1 - 24p_7\right) \partial_p \partial_\ell \mathcal{P}^{ij} \partial_n \partial_j \mathcal{P}^{kl} \partial_q \mathcal{P}^{mn} \mathcal{P}^{pq} \partial_i f \partial_m \partial_k g \\
& \quad + \left(-\frac{11}{180} - 16p_9 + 16p_8 - 8p_6 + 8p_5 + 8p_1 - 16p_7\right) \partial_p \mathcal{P}^{ij} \partial_q \partial_j \mathcal{P}^{kl} \partial_k \mathcal{P}^{mn} \partial_n \partial_\ell \mathcal{P}^{pq} \partial_i f \partial_m g \\
& \quad + \left(-\frac{17}{180} + 16p_8 - 8p_6 - 32p_4 + 40p_5 - 8p_1 - 16p_7\right) \partial_p \partial_\ell \mathcal{P}^{ij} \partial_q \partial_j \mathcal{P}^{kl} \mathcal{P}^{mn} \partial_n \mathcal{P}^{pq} \partial_m \partial_i f \partial_k g \\
& \quad + \left(\frac{17}{180} - 16p_8 + 8p_6 + 32p_4 - 40p_5 + 8p_1 + 16p_7\right) \partial_p \partial_\ell \mathcal{P}^{ij} \partial_q \partial_j \mathcal{P}^{kl} \mathcal{P}^{mn} \partial_n \mathcal{P}^{pq} \partial_i f \partial_m \partial_k g \\
& \quad + \left(\frac{61}{180} - 48p_8 - 8p_6 + 96p_4 - 120p_5 + 24p_1 + 48p_7\right) \partial_p \partial_\ell \mathcal{P}^{ij} \partial_q \mathcal{P}^{kl} \mathcal{P}^{mn} \partial_n \mathcal{P}^{pq} \partial_k \partial_i f \partial_m \partial_j g \\
& \quad + \left(-\frac{61}{180} + 48p_8 + 8p_6 - 96p_4 + 120p_5 - 24p_1 - 48p_7\right) \partial_q \partial_m \mathcal{P}^{ij} \mathcal{P}^{kl} \partial_\ell \mathcal{P}^{mn} \partial_n \mathcal{P}^{pq} \partial_k \partial_i f \partial_p \partial_j g \\
& \quad + \left(\frac{53}{90} - 96p_8 - 16p_6 + 192p_4 - 240p_5 + 48p_1 + 96p_7\right) \partial_n \partial_\ell \mathcal{P}^{ij} \partial_p \mathcal{P}^{kl} \partial_q \mathcal{P}^{mn} \mathcal{P}^{pq} \partial_k \partial_i f \partial_m \partial_j g \\
& \quad + \left(-\frac{49}{90} + 48p_8 + 24p_6 - 144p_4 + 168p_5 - 24p_1 - 72p_7\right) \partial_p \partial_\ell \mathcal{P}^{ij} \partial_n \mathcal{P}^{kl} \partial_q \mathcal{P}^{mn} \mathcal{P}^{pq} \partial_k \partial_i f \partial_m \partial_j g \\
& \quad + \left(\frac{49}{90} - 48p_8 - 24p_6 + 144p_4 - 168p_5 + 24p_1 + 72p_7\right) \partial_p \partial_n \mathcal{P}^{ij} \partial_q \mathcal{P}^{kl} \partial_\ell \mathcal{P}^{mn} \mathcal{P}^{pq} \partial_k \partial_i f \partial_m \partial_j g \\
& \quad + \left(\frac{1}{90} - 16p_8 + 8p_6 - 16p_3 + 16p_4 - 24p_5 - 8p_1 + 8p_7\right) \partial_p \mathcal{P}^{ij} \partial_j \mathcal{P}^{kl} \partial_q \partial_\ell \mathcal{P}^{mn} \partial_n \mathcal{P}^{pq} \partial_k \partial_i f \partial_m g \\
& \quad + \left(-\frac{1}{90} + 16p_8 - 8p_6 + 16p_3 - 16p_4 + 24p_5 + 8p_1 - 8p_7\right) \partial_q \partial_k \mathcal{P}^{ij} \partial_j \mathcal{P}^{kl} \partial_\ell \mathcal{P}^{mn} \partial_n \mathcal{P}^{pq} \partial_i f \partial_p \partial_m g \\
& \quad + \left(\frac{3}{20} + 16p_9 - 32p_8 + 8p_6 + 16p_4 - 40p_5 - 8p_1 + 32p_7\right) \partial_p \mathcal{P}^{ij} \partial_q \partial_j \mathcal{P}^{kl} \partial_\ell \mathcal{P}^{mn} \partial_n \mathcal{P}^{pq} \partial_k \partial_i f \partial_m g \\
& \quad + \left(-\frac{3}{20} - 16p_9 + 32p_8 - 8p_6 - 16p_4 + 40p_5 + 8p_1 - 32p_7\right) \partial_n \mathcal{P}^{ij} \partial_j \mathcal{P}^{kl} \partial_q \partial_k \mathcal{P}^{mn} \partial_\ell \mathcal{P}^{pq} \partial_i f \partial_p \partial_m g \\
& \quad + \left(-\frac{7}{180} - 16p_9 + 16p_8 - 8p_6 + 16p_4 + 8p_5 - 8p_1 - 16p_7\right) \partial_q \partial_m \mathcal{P}^{ij} \partial_j \mathcal{P}^{kl} \partial_\ell \mathcal{P}^{mn} \partial_n \mathcal{P}^{pq} \partial_k \partial_i f \partial_p g \\
& \quad + \left(\frac{7}{180} + 16p_9 - 16p_8 + 8p_6 - 16p_4 - 8p_5 + 8p_1 + 16p_7\right) \partial_p \mathcal{P}^{ij} \partial_q \partial_j \mathcal{P}^{kl} \partial_\ell \mathcal{P}^{mn} \partial_n \mathcal{P}^{pq} \partial_i f \partial_m \partial_k g \\
& \quad + \left(\frac{7}{120} + 16p_9 - 8p_8 + 8p_6 + 16p_4 + 16p_{10} - 8p_1 + 12p_7\right) \partial_p \partial_m \mathcal{P}^{ij} \partial_q \partial_j \mathcal{P}^{kl} \partial_\ell \mathcal{P}^{mn} \partial_n \mathcal{P}^{pq} \partial_i f \partial_k g \\
& \quad + \left(-\frac{7}{120} - 16p_9 + 8p_8 - 8p_6 - 16p_4 - 16p_{10} + 8p_1 - 12p_7\right) \partial_p \partial_n \mathcal{P}^{ij} \partial_j \mathcal{P}^{kl} \partial_q \partial_k \mathcal{P}^{mn} \partial_\ell \mathcal{P}^{pq} \partial_i f \partial_m g \\
& \quad + (8p_9 - 8p_8 + 4p_6 - 8p_3 - 8p_4 + 4p_5 - 8p_2 - 4p_1 + 4p_7) \partial_p \partial_k \mathcal{P}^{ij} \partial_q \partial_j \mathcal{P}^{kl} \partial_\ell \mathcal{P}^{mn} \partial_n \mathcal{P}^{pq} \partial_i f \partial_m g \\
& \quad + (-8p_9 + 8p_8 - 4p_6 + 8p_3 + 8p_4 - 4p_5 + 8p_2 + 4p_1 - 4p_7) \partial_p \mathcal{P}^{ij} \partial_j \mathcal{P}^{kl} \partial_q \partial_k \mathcal{P}^{mn} \partial_n \partial_\ell \mathcal{P}^{pq} \partial_i f \partial_m g \\
& \quad \quad + \left(\frac{23}{360} + 8p_9 - 16p_8 + 4p_6 - 8p_3 + 8p_4 - 20p_5 + 8p_2 - 16p_{10} - 4p_1 + 16p_7\right) \\
& \quad \quad \quad \partial_p \partial_m \mathcal{P}^{ij} \partial_j \mathcal{P}^{kl} \partial_q \partial_\ell \mathcal{P}^{mn} \partial_n \mathcal{P}^{pq} \partial_i f \partial_k g \\
& \quad \quad + \left(-\frac{23}{360} - 8p_9 + 16p_8 - 4p_6 + 8p_3 - 8p_4 + 20p_5 - 8p_2 + 16p_{10} + 4p_1 - 16p_7\right) \\
& \quad \quad \quad \partial_n \mathcal{P}^{ij} \partial_p \partial_j \mathcal{P}^{kl} \partial_q \partial_k \mathcal{P}^{mn} \partial_\ell \mathcal{P}^{pq} \partial_i f \partial_m g) + \bar{o}(\hbar^4). \quad (11)
\end{aligned}$$

The ten master-parameters in (11) are the still unknown weights of the prime graphs which are portrayed in Fig. 3 on p. 247. The four underlined parameters can be gauged out (*without* modifying the coefficients of any other Kontsevich graphs with four internal vertices), see Theorem 14 on p. 258. At all values of the ten master-parameters, that

is, irrespective of their true values given by formula (5), the  $\star$ -product is proven in Theorem 12 to be associative modulo  $\bar{o}(\hbar^4)$ .

*Acknowledgements.* The authors are grateful to the anonymous referees for their critical comments and suggestions which helped us improve this text, to prof. S. Tabachnikov (Editor-in-Chief) for persistence and constructive criticism, and B. Pym and E. Panzer for communicating the values of ten master-parameters obtained via a different technique [34]. We thank prof. M. Gerstenhaber and M. Kontsevich for their attention to our work.

This research was supported in part by JBI RUG project 106552 (Groningen, The Netherlands) and IM JGU project 5020 (Mainz, Germany). The authors also thank the Center for Information Technology of the University of Groningen for providing access to Peregrine high performance computing cluster. A part of this research was done while the authors were visiting at the IHÉS in Bures-sur-Yvette, France and AVK was visiting at the MPIM Bonn, Germany; warm hospitality and partial financial support by these institutions are gratefully acknowledged.

#### REFERENCES

- [1] Ammar M., Chloup V., Gutt S. (2008) Universal Star Products, *Lett. Math. Phys.* **84**:2–3, 199–215.
- [2] Banks P., Panzer E., Pym B. (2018) Multiple zeta values in deformation quantization, *Preprint arXiv:1812.11649* [math.QA].
- [3] Bauer C., Frink A., Kreckel R. (2002) Introduction to the GiNaC Framework for Symbolic Computation within the C++ Programming Language, *J. Symb. Comp.* **33**, 1–12. See also <http://www.ginac.de>.
- [4] Bayen F., Flato M., Frønsdal C., Lichnerowicz A., Sternheimer D. (1978) Deformation theory and quantization. I. Deformations of symplectic structures, *Ann. Phys. (N. Y.)* **111**:1, 61–110.
- [5] Bayen F., Flato M., Frønsdal C., Lichnerowicz A., Sternheimer D. (1978) Deformation theory and quantization. II. Physical applications, *Ann. Phys. (N. Y.)* **111**:1, 111–151.
- [6] Ben Amar N. (2007) A comparison between Rieffel’s and Kontsevich’s deformation quantizations for linear Poisson tensors, *Pac. J. Math.* **229**:1, 1–24.
- [7] Bouisaghouane A., Buring R., Kiselev A. V. (2017) The Kontsevich tetrahedral flow revisited, *J. Geom. Phys.* **119**, 272–285. (*Preprint arXiv:1608.01710* [q-alg]).
- [8] Bouisaghouane A., Kiselev A. V. (2017) Do the Kontsevich tetrahedral flows preserve or destroy the space of Poisson bi-vectors? *J. Phys.: Conf. Ser.* **804**, Paper 012008, 1–10. (*Preprint arXiv:1609.06677* [q-alg])
- [9] Buring R. Software package `kontsevich-graph-series-cpp`, see link: [https://github.com/rburing/kontsevich\\_graph\\_series\\_cpp](https://github.com/rburing/kontsevich_graph_series_cpp)
- [10] Buring R. List of integrands for the 149 basic Kontsevich graphs at  $\hbar^4$ , see link: [http://rburing.nl/kontsevich\\_graph\\_weight\\_integrands4.txt](http://rburing.nl/kontsevich_graph_weight_integrands4.txt)
- [11] Buring R., Kiselev A. V. (2019) Formality morphism as the mechanism of  $\star$ -product associativity: how it works. *Collected Works Inst. Math. Kiev* **16**:1, 22–43. (*Preprint arXiv:1907.00639* [math.QA])

- [12] *Buring R., Kiselev A. V.* (2017) On the Kontsevich  $\star$ -product associativity mechanism, *Physics of Particles and Nuclei Letters* **14**:2, 403–407. (*Preprint arXiv:1602.09036* [q-alg])
- [13] *Buring R., Kiselev A. V.* (2019) The orientation morphism: from graph cocycles to deformations of Poisson structures, *J. Phys.: Conf. Ser.* **1194** Proc. 32nd Int. colloquium on Group-theoretical methods in Physics: GROUP32 (9–13 July 2018, CVUT Prague, Czech Republic), Paper 012017, 10 p. (*Preprint arXiv:1811.07878* [math.CO])
- [14] *Buring R., Kiselev A. V.* (2015) The table of weights for graphs with  $\leq 3$  internal vertices in Kontsevich’s deformation quantization formula. (3rd International workshop on symmetries of discrete systems & processes, 3–7 August 2015, CVUT Děčín, Czech Republic), see Appendix A.1 in this paper.
- [15] *Cattaneo A. S., Felder G.* (2000) A path integral approach to the Kontsevich quantization formula, *Comm. Math. Phys.* **212**:3, 591–611.
- [16] *Dito G.* (1999) Kontsevich star product on the dual of a Lie algebra, *Lett. Math. Phys.* **4**, 291–306.
- [17] *Felder G., Willwacher T.* (2010) On the (ir)rationality of Kontsevich weights, *Int. Math. Res. Not.* **2010**:4, 701–716.
- [18] *Felder G., Shoikhet B.* (2000) Deformation quantization with traces, *Lett. Math. Phys.* **53**, 75–86.
- [19] *Gerstenhaber M.* (1964) On the deformation of rings and algebras, *Ann. Math.* **79**, 59–103.
- [20] *Grabowski J., Marmo G., Perelomov A. M.* (1993) Poisson structures: towards a classification, *Mod. Phys. Lett.* **A8**:18, 1719–1733.
- [21] *Kathotia V.* (1998) Kontsevich’s universal formula for deformation quantization and the Campbell–Baker–Hausdorff formula, I. *Preprint arXiv:9811174* (v2) [math.QA]
- [22] *Kiselev A. V.* (2012) The twelve lectures in the (non)commutative geometry of differential equations, *Preprint IHÉS/M/12/13* (Bures-sur-Yvette, France), 140 p.
- [23] *Kiselev A. V.* (2013) The geometry of variations in Batalin–Vilkovisky formalism, *J. Phys.: Conf. Ser.* **474**, Paper 012024, 1–51. (*Preprint arXiv:1312.1262* [math-ph])
- [24] *Kiselev A. V.* (2014) The Jacobi identity for graded-commutative variational Schouten bracket revisited, *Physics of Particles and Nuclei Letters* **11**:7, 950–953. (*Preprint arXiv:1312.4140* [math-ph])
- [25] *Kiselev A. V.* (2017) The deformation quantization mapping of Poisson to associative structures in field theory, *Banach Center Publ.* **113** 50th Sophus Lie Seminar 219–242. (*Preprint arXiv:1705.01777* [q-alg])
- [26] *Kiselev A. V.* (2016) The right-hand side of the Jacobi identity: to be naught or not to be? *J. Phys.: Conf. Ser.* **670** Proc. XXIII Int. conf. ‘Integrable Systems and Quantum Symmetries’ (23–27 June 2015, CVUT Prague, Czech Republic), Paper 012030, 1–17. (*Preprint arXiv:1410.0173* [math-ph])
- [27] *Kiselev A. V.* (2017) The calculus of multivectors on noncommutative jet spaces, *J. Geom. Phys.* **130**, 130–167. (*Preprint arXiv:1210.0726* [math.DG])

- [28] *Kontsevich M.* (1993) Formal (non)commutative symplectic geometry, The Gel'fand Mathematical Seminars, 1990-1992 (L. Corwin, I. Gelfand, and J. Lepowsky, eds), Birkhäuser, Boston MA, 173–187.
- [29] *Kontsevich M.* (1994) Feynman diagrams and low-dimensional topology. First Europ. Congr. of Math. **2** (Paris, 1992), Progr. Math. **120**, Birkhäuser, Basel, 97–121.
- [30] *Kontsevich M.* (1995) Homological algebra of mirror symmetry. Proc. Intern. Congr. Math. **1** (Zürich, 1994), Birkhäuser, Basel, 120–139.
- [31] *Kontsevich M.* (1997) Formality conjecture. Deformation theory and symplectic geometry (Ascona 1996, D. Sternheimer, J. Rawnsley and S. Gutt, eds), Math. Phys. Stud. **20**, Kluwer Acad. Publ., Dordrecht, 139–156.
- [32] *Kontsevich M.* (2003) Deformation quantization of Poisson manifolds, *Lett. Math. Phys.* **66**:3, 157–216. (*Preprint* [q-alg/9709040](https://arxiv.org/abs/q-alg/9709040))
- [33] *Laurent–Gengoux C., Picherau A., Vanhaecke P.* (2013) Poisson structures. Grundlehren der mathematischen Wissenschaften **347**, Springer–Verlag, Berlin.
- [34] *Panzer E., Pym B.* (2017) Private communication. See also: <https://www.mathematics.uni-bonn.de/veranstaltungskalender/17251>
- [35] *Penkava M., Vanhaecke P.* (2000) Deformation Quantization of Polynomial Poisson Algebras, *J. Algebra* **227**:1, 365–393. (*Preprint* [arXiv:math/9804022](https://arxiv.org/abs/math/9804022))
- [36] *Polyak M.* (2003) Quantization of linear Poisson structures and degrees of maps, *Lett. Math. Phys.* **66**:1, 15–35.
- [37] *Vanhaecke P.* (1996) Integrable systems in the realm of algebraic geometry, Lect. Notes Math. **1638**, Springer–Verlag, Berlin.
- [38] *Willwacher T.* (2014) The obstruction to the existence of a loopless star product, *C. R. Math. Acad. Sci. Paris* **352**:11, 881–883.

*This text was submitted in its original form on 20 December 2017.*

APPENDIX A. APPROXIMATIONS AND CONJECTURED VALUES OF WEIGHT  
INTEGRALS

The material presented here is an expanded version of section 3 of the note [14] by the authors.

**A.1. The weight integral in Cartesian coordinates.** Recall the integral formula for the weight of a graph  $\Gamma \in \tilde{G}_{2,k}$  (see section 2):

$$w(\Gamma) = \frac{1}{(2\pi)^{2k}} \int_{C_k(\mathbb{H})} \bigwedge_{j=1}^k d\varphi(p_j, p_{\text{Left}(j)}) \wedge d\varphi(p_j, p_{\text{Right}(j)}), \quad (5)$$

such that the integral is taken over the configuration space of  $k$  points in the upper half-plane  $\mathbb{H} \subset \mathbb{C}$ ,

$$C_k(\mathbb{H}) = \{(p_1, \dots, p_k) \in \mathbb{H}^k : p_i \text{ pairwise distinct}\},$$

and where  $\varphi: C_2(\mathbb{H}) \rightarrow [0, 2\pi)$  was defined by  $\varphi(p, q) = \text{Arg}\left(\frac{q-p}{q-\bar{p}}\right)$ .

For nonzero  $z = x + iy$  in  $\mathbb{H}$  we have  $\text{Arg}(x + iy) \cong \arctan(y/x)$ , where  $\cong$  denotes equality of functions up to a constant. Since  $\frac{d}{dt} \arctan(t) = 1/(1+t^2)$ , the weight integrand is a rational function of the Cartesian coordinates: for  $p = a+ib$  and  $q = x+iy$ ,

$$\varphi(p, q) \cong \arctan\left(\frac{2b(a-x)}{(a-x)^2 + (y+b)(y-b)}\right). \quad (12)$$

In Cartesian coordinates  $(x_1, y_1, \dots, x_k, y_k)$ , the weight integrand can now be written as the Jacobian determinant of the map  $\Phi_\Gamma: C_k(\mathbb{H}) \rightarrow [0, 2\pi)^{2k}$  defined by<sup>22</sup>

$$\Phi_\Gamma(p_1, \dots, p_k) = (\varphi(p_1, p_{\text{Left}(1)}), \varphi(p_1, p_{\text{Right}(1)}), \dots, \varphi(p_k, p_{\text{Left}(k)}), \varphi(p_k, p_{\text{Right}(k)}))$$

considered as a function of the  $(x_j, y_j)$  through  $p_j = x_j + iy_j$ .

**Implementation 17.** The command

```
> weight_integrands <graph-series-file>
```

takes as input a list of graphs  $\Gamma \in \tilde{G}_{2,k}$  with (possibly undetermined) coefficients, and sends to the standard output lines of the following form:

```
(* <graph encoding>    <coefficient> *)
<weight integrand of the graph above>
```

where the weight integrands are written in **Mathematica** format, as `Det[...]`.

We can take integration domain to be  $\mathbb{H}^k$ , since for any  $i \neq j$  the set  $\{(p_1, \dots, p_k) \in \mathbb{C}^k : p_i = p_j\}$  is a strict linear subspace of  $\mathbb{C}^k$ , which has measure zero. The weight integral is absolutely convergent [32], so by the Fubini–Tonelli theorem we may evaluate it as an iterated integral in any order. We can use the residue theorem<sup>23</sup> to integrate out the Cartesian coordinates corresponding to the  $k$  real parts, halving the dimension. It then remains to integrate the result (a function of the  $k$  imaginary parts) over  $\mathbb{R}^k$ .

<sup>22</sup>Called a Gauss map by M. Polyak [36].

<sup>23</sup>G. Dito used the residue method for one graph [16] at  $k = 2$ , and remarked that that it becomes unpractical for  $k \geq 3$ .



**Example 38.** For the wedge graph  $\Lambda$  we have the Cartesian coordinates  $x + iy$  in the upper half-plane and the integrand (obtained using Implementation 17)

$$f(x, y) = \frac{4y}{((x-1)^2 + y^2)(x^2 + y^2)}.$$

To apply the residue theorem we interpret  $f(x, y)$  as a rational function in a single *complex* variable  $x$ . Its poles are then  $\pm iy$  and  $1 \pm iy$ , so the poles in the upper half-plane are  $iy$  and  $1 + iy$  (since  $y > 0$ ). The residues at these poles are  $r_1 = 2/(i + 2y)$  and  $r_2 = -2/(2y - i)$  respectively. Hence the residue theorem yields that the integral of  $f(x, y)$  with respect to  $x$  over the real line is  $2\pi i(r_1 + r_2) = 8\pi/(1 + 4y^2)$ . When we integrate this over  $y > 0$  and divide by  $(2\pi)^2$  we obtain  $1/2$ , as desired.

This is of course a toy example. For higher  $k$  the expressions become larger, but also one has to consider more carefully which poles are in the upper half-plane. From the expression (12) for  $\varphi$  one can see that this issue depends on the relative position of the coordinates on the imaginary axis ( $y$  and  $b$  in that formula).

For  $k = 3$  with coordinates on  $\mathbb{H}^3$  given by

$$a + bi, \quad c + di, \quad e + fi,$$

let us agree to call  $a, c, e$  the real coordinates and  $b, d, f$  the imaginary coordinates. We now split the integral into a sum of integrals over  $3! = 6$  regions, one for each possible ordering of the imaginary coordinates:

$$b < d < f; \quad b < f < d; \quad d < b < f; \quad d < f < b; \quad f < b < d; \quad f < d < b.$$

In each such region it is known for every (complexified) real coordinate which poles are in the upper half-plane, so we can apply the residue theorem three times. The result can be numerically integrated more effectively than the original expression, for one because we have halved the dimension of the integration domain.

*Remark 17.* To integrate over the region of  $\mathbb{H}^3$  defined by  $b < d < f$ , one can choose integration bounds as follows:  $\int_0^\infty db \int_b^\infty df \int_b^f dd$  (and similarly for the other permutations). For the region of  $\mathbb{H}^4$  defined by  $b < d < f < h$  one can choose the integration bounds  $\int_0^\infty db \int_b^\infty dh \int_b^h dd \int_d^h df$ , and so on.

**Implementation 18.** The strategy above is implemented by the following *Mathematica* code (for the order 4, but it can be adapted for others), where  $W$  is the weight integrand.

`W = an integrand, e.g. from list [10];`

`integrationvariables = {a, b, c, d, e, f, g, h};`

`imaginaryvariables =`

`integrationvariables[[2 #1]] & /@`

`Range[1, Length[integrationvariables]/2];`

`realvariables =`

`integrationvariables[[2 #1 - 1]] & /@`

`Range[1, Length[integrationvariables]/2];`

`basicAssumptions =`

```

Element[a, Reals] && Element[c, Reals] && Element[e, Reals] &&
Element[g, Reals] && b > 0 && d > 0 && f > 0 && h > 0;

ContourIntegrate[function_, variable_, assumptions_] :=
2*Pi*I*Total[
Map[
Function[{p}, (Numerator[Together[function]]/
D[Denominator[Together[function]], variable]) /. {variable ->
p}],
Select[
ReplaceList[variable,
Assuming[assumptions,
Flatten[FullSimplify[
Solve[Denominator[Together[function]] == 0, variable,
Complexes]]]],
Function[{r}, Simplify[ComplexExpand[Im[r]] > 0, assumptions]]]]]]

IteratedContourIntegrate[function_, variables_, assumptions_] :=
Fold[ContourIntegrate[Together[#1], #2, assumptions] &, function,
variables]

integrals = Map[
NIntegrate[
Simplify[
IteratedContourIntegrate[W, realvariables,
basicAssumptions && #1[[1]] < #1[[2]] < #1[[3]] < #1[[4]]]
TimeConstraint -> Infinity],
Evaluate[
Sequence @@
{{#1[[1]], 0, Infinity}, {#1[[3]], #1[[1]],
Infinity}, {#1[[2]], #1[[1]], #1[[3]]}, {#1[[4]], #1[[3]],
Infinity}}
],
Method -> {GlobalAdaptive, MaxErrorIncreases -> 10^4}
] &, Permutations[imaginaryvariables]]

Print[integrals]
Print[Total[integrals]]
Print[Total[integrals]/N[(2 Pi)^8]]

```

*Remark 18.* This strategy allows effective numerical integration of all weights up to order 3. At the order 4, it works for some weights but not others: see Tables 2 and 3. The call(s) to Map may be replaced by ParallelMap to parallelize the computation.

**Example 39.** The second Bernoulli graph  [17] has the weight integrand

$$\frac{64bfd(c((a-c)^2+b^2)+d^2(c-2a))(f^2(e-2c)+e((e-c)^2+d^2))}{(a^2+b^2)(f^2+(e-1)^2)(f^2+e^2)(c^2+d^2)((a-c)^2+(b-d)^2)((a-c)^2+(b+d)^2)((f+d)^2+(e-c)^2)}$$

The residue calculation followed by the numerical integration leads to the estimate  $5.71871 \times 10^{-9} - 5.92495 \times 10^{-21}i$  of the weight; this leads to the guess that it is zero and in fact it is true.

TABLE 2. Verified values

Weight	Approximation	True value
w_4_1	$-0.00694444401170 \pm 0.000000906189$	$-1/144 \approx -0.00694444$

TABLE 3. Conjectured values

Weight	Approximation	Conjectured true value
w_4_103	$-0.000086894703 \pm 0.000000681076$	$-1/11520 \approx 0.000086805$
w_4_104	$0.000347214860 \pm 0.000000371598$	$1/2880 \approx 0.000347222$
w_4_112	$-0.000347219933 \pm 0.000000042901$	$-1/2880 \approx -0.000347222$
w_4_113	$0.000694441623 \pm 0.000000093136$	$1/1440 \approx 0.000694444$
w_4_133	$0.000694443060 \pm 0.000000078774$	$1/1440 \approx 0.000694444$
w_4_138	$-0.001041664533 \pm 0.000000095465$	$-1/960 \approx -0.001041666$
w_4_147	$-0.000043376821 \pm 0.000000095465$	$-1/23040 \approx -0.000043402$
w_4_148	$0.000173611294 \pm 0.000000015063$	$1/5760 \approx 0.000173611$

In particular, this table lists the approximate value of the master-parameters  $p_4 = w_4_103$  and  $p_5 = w_4_104$ . The relation  $w_4_133 = 2 \cdot w_4_104$  which was found in Theorem 9 and listed in Table 7 of Appendix C is satisfied approximately. Furthermore, the relation  $w_4_103 = 2 \cdot w_4_147$  seems to hold approximately.

**A.2. Claimed values of the 10 master-parameters.** By using a different technique B. Pym and E. Panzer have obtained the exact values of the ten master-parameters.

**Claim** ([34]). *The values of ten master-parameters (which are the weights of ten graphs in Figure 3 on p. 247) are given in Table 4 below.*

TABLE 4. Recently suggested values of the master-parameters [34].

Master-parameter	Value
$p_1 = w\_4\_100$	1/1440
$p_2 = w\_4\_101$	1/2880
$p_3 = w\_4\_102$	1/5760
$p_4 = w\_4\_103$	-1/11520
$p_5 = w\_4\_104$	1/2880
$p_6 = w\_4\_107$	13/2880
$p_7 = w\_4\_108$	-17/2880
$p_8 = w\_4\_109$	-1/1152
$p_9 = w\_4\_119$	-1/1280
$p_{10} = w\_4\_125$	-1/960

Let it be emphasized that these ten values are conjectured via a use of software which is currently under development.

*Remark 19.* The exact values of two master-parameters  $w\_4\_103$  and  $w\_4\_104$  reproduce the values which had been conjectured in Table 3. We also note that all the weights of graphs in  $\star \bmod \bar{o}(\hbar^4)$  are rational numbers. Thirdly, the values of non-master parameters (namely,  $w\_4\_112$ ,  $w\_4\_113$ ,  $w\_4\_133$ ,  $w\_4\_138$ ,  $w\_4\_147$ , and  $w\_4\_148$ ) in Table 3, whenever recalculated on the basis of conjectured values from Table 4, do all match the numerical approximations in Table 3, reproducing our conjectured rational values in its rightmost column.

In conclusion, provided that all the ten values in Table 4 are true, this is the authentic Kontsevich star-product up to  $\bar{o}(\hbar^4)$ :

$$\begin{aligned}
 f \star g = & f \times g + \hbar \mathcal{P}^{ij} \partial_i f \partial_j g + \hbar^2 \left( -\frac{1}{6} \partial_\ell \mathcal{P}^{ij} \partial_j \mathcal{P}^{k\ell} \partial_i f \partial_k g - \frac{1}{3} \partial_\ell \mathcal{P}^{ij} \mathcal{P}^{k\ell} \partial_i f \partial_k \partial_j g \right. \\
 & + \frac{1}{3} \partial_\ell \mathcal{P}^{ij} \mathcal{P}^{k\ell} \partial_k \partial_i f \partial_j g + \frac{1}{2} \mathcal{P}^{ij} \mathcal{P}^{k\ell} \partial_k \partial_i f \partial_\ell \partial_j g \left. \right) + \hbar^3 \left( -\frac{1}{6} \partial_m \partial_\ell \mathcal{P}^{ij} \partial_n \partial_j \mathcal{P}^{k\ell} \mathcal{P}^{mn} \partial_i f \partial_k g \right. \\
 & + \frac{1}{6} \partial_n \partial_\ell \mathcal{P}^{ij} \mathcal{P}^{k\ell} \mathcal{P}^{mn} \partial_i f \partial_m \partial_k \partial_j g - \frac{1}{3} \partial_n \mathcal{P}^{ij} \mathcal{P}^{k\ell} \mathcal{P}^{mn} \partial_k \partial_i f \partial_m \partial_\ell \partial_j g \\
 & + \frac{1}{6} \partial_n \partial_\ell \mathcal{P}^{ij} \mathcal{P}^{k\ell} \mathcal{P}^{mn} \partial_m \partial_k \partial_i f \partial_j g + \frac{1}{3} \partial_n \mathcal{P}^{ij} \mathcal{P}^{k\ell} \mathcal{P}^{mn} \partial_m \partial_k \partial_i f \partial_\ell \partial_j g \\
 & + \frac{1}{6} \mathcal{P}^{ij} \mathcal{P}^{k\ell} \mathcal{P}^{mn} \partial_m \partial_k \partial_i f \partial_n \partial_\ell \partial_j g - \frac{1}{6} \partial_m \partial_\ell \mathcal{P}^{ij} \partial_n \mathcal{P}^{k\ell} \mathcal{P}^{mn} \partial_i f \partial_k \partial_j g \\
 & + \frac{1}{6} \partial_n \partial_\ell \mathcal{P}^{ij} \partial_j \mathcal{P}^{k\ell} \mathcal{P}^{mn} \partial_i f \partial_m \partial_k g - \frac{1}{6} \partial_m \partial_\ell \mathcal{P}^{ij} \partial_n \mathcal{P}^{k\ell} \mathcal{P}^{mn} \partial_k \partial_i f \partial_j g \\
 & - \frac{1}{6} \partial_\ell \mathcal{P}^{ij} \partial_n \partial_j \mathcal{P}^{k\ell} \mathcal{P}^{mn} \partial_m \partial_i f \partial_k g - \frac{1}{6} \mathcal{P}^{ij} \partial_n \mathcal{P}^{k\ell} \partial_\ell \mathcal{P}^{mn} \partial_k \partial_i f \partial_m \partial_j g \\
 & \left. - \frac{1}{6} \partial_n \mathcal{P}^{ij} \mathcal{P}^{k\ell} \partial_\ell \mathcal{P}^{mn} \partial_k \partial_i f \partial_m \partial_j g - \frac{1}{6} \partial_\ell \mathcal{P}^{ij} \partial_n \mathcal{P}^{k\ell} \mathcal{P}^{mn} \partial_k \partial_i f \partial_m \partial_j g \right) +
 \end{aligned}$$











## APPENDIX B. C++ CLASSES AND METHODS

CLASS `KontsevichGraph`

Summary: a (signed) Kontsevich graph.

Data members (private):

```
size_t d_internal = 0;
size_t d_external = 0;
std::vector< std::pair<char, char> > d_targets;
int d_sign = 1;
```

Public typedefs:

```
typedef char Vertex;
typedef std::pair<Vertex, Vertex> VertexPair;
```

Constructors:

```
KontsevichGraph() = default;
KontsevichGraph(size_t internal, size_t external,
                 std::vector<VertexPair> targets,
                 int sign = 1, bool normalized = false);
```

Accessor methods:

```
std::vector<VertexPair> targets() const;
VertexPair targets(Vertex internal_vertex) const;
int sign() const;
int sign(int new_sign);
size_t internal() const;
size_t external() const;
```

Methods to obtain numerical information:

```
size_t vertices() const;
std::vector<Vertex> internal_vertices() const;
std::pair< size_t, std::vector<VertexPair> > abs() const;
size_t multiplicity() const;
size_t in_degree(KontsevichGraph::Vertex vertex) const;
std::vector<size_t> in_degrees() const;
std::vector<Vertex> neighbors_in(Vertex vertex) const;
KontsevichGraph mirror_image() const;
std::string as_sage_expression() const;
std::string encoding() const;
std::vector< std::tuple<KontsevichGraph, int, int> > permutations() const;
```

Methods that modify the graph:

```
void normalize();
KontsevichGraph& operator*=(const KontsevichGraph& rhs);
```

Methods that test for graph properties:

```
bool operator<(const KontsevichGraph& rhs) const;
bool is_zero() const;
```

```

bool is_prime() const;
bool positive_differential_order() const;
bool has_cycles() const;
bool has_tadpoles() const;
bool has_multiple_edges() const;
bool has_max_internal_indegree(size_t max_indegree) const;

```

Static methods:

```

static std::set<KontsevichGraph> graphs(size_t internal,
    size_t external = 2, bool modulo_signs = false,
    bool modulo_mirror_images = false,
    std::function<void(KontsevichGraph)> const& callback = nullptr,
    std::function<bool(KontsevichGraph)> const& filter = nullptr);

```

Private methods:

```

friend std::ostream& operator<<(std::ostream &os, const KontsevichGraph& g);
friend std::istream& operator>>(std::istream& is, KontsevichGraph& g);
friend bool operator==(const KontsevichGraph &lhs, const KontsevichGraph& rhs);
friend bool operator!=(const KontsevichGraph &lhs, const KontsevichGraph& rhs);

```

Functions defined outside the class:

```

KontsevichGraph operator*(KontsevichGraph lhs, const KontsevichGraph& rhs);
std::ostream& operator<<(std::ostream &os, const KontsevichGraph::Vertex v);

```

#### CLASS KontsevichGraphSum<T>

- Template parameter T: type of the coefficients (e.g. `KontsevichGraphSum<int>`).
- Publically extends: `std::vector< std::pair<T, KontsevichGraph> >`.

Summary: a sum of Kontsevich graphs, with method to reduce modulo skew-symmetry.

Data members: inherited.

Public typedefs:

```

typedef std::pair<T, KontsevichGraph> Term;

```

Constructors (inherited):

```

using std::vector< std::pair<T, KontsevichGraph> >::vector;

```

Accessor methods:

```

using std::vector< std::pair<T, KontsevichGraph> >::operator[];
KontsevichGraphSum<T> operator[](std::vector<size_t> indegrees) const;
T operator[](KontsevichGraph) const;

```

Arithmetic operators:

```

KontsevichGraphSum<T> operator()(std::vector< KontsevichGraphSum<T> >) const;
KontsevichGraphSum<T>& operator+=(const KontsevichGraphSum<T>& rhs);
KontsevichGraphSum<T>& operator-=(const KontsevichGraphSum<T>& rhs);
KontsevichGraphSum<T>& operator=(const KontsevichGraphSum<T>&) = default;

```

Methods:

```
std::vector< std::vector<size_t> > in_degrees(bool ascending = false) const;
KontsevichGraphSum<T> skew_symmetrization() const;
```

Methods that modify the graph sum:

```
void reduce_mod_skew();
```

Comparison operators:

```
bool operator==(const KontsevichGraphSum<T>& other) const;
bool operator==(int other) const;
bool operator!=(const KontsevichGraphSum<T>& other) const;
bool operator!=(int other) const;
```

Friend operators:

```
friend std::ostream& operator<< <>(std::ostream& os,
                                   const KontsevichGraphSum<T>::Term& term);
friend std::ostream& operator<< <>(std::ostream& os,
                                   const KontsevichGraphSum<T>& gs);
friend std::istream& operator>> <>(std::istream& is,
                                   KontsevichGraphSum<T>& sum);
```

Functions defined outside the class:

```
KontsevichGraphSum<T> operator+(KontsevichGraphSum<T> lhs,
                                const KontsevichGraphSum<T>& rhs);
KontsevichGraphSum<T> operator-(KontsevichGraphSum<T> lhs,
                                const KontsevichGraphSum<T>& rhs);
KontsevichGraphSum<T> operator*(T lhs,
                                KontsevichGraphSum<T> rhs);
std::ostream& operator<<(std::ostream&, const std::pair<T, KontsevichGraph>&);
std::ostream& operator<<(std::ostream&, const KontsevichGraphSum<T>&);
std::istream& operator>>(std::istream&, KontsevichGraphSum<T>&);
```

#### CLASS KontsevichGraphSeries<T>

- Template parameter T: type of the coefficients (e.g. `KontsevichGraphSeries<int>`).
- Publically extends: `std::map< size_t, KontsevichGraphSum<T> >`

Summary: a formal power series expansion; sums of Kontsevich graphs as coefficients.

Data members: inherited, plus (private):

```
size_t d_precision = std::numeric_limits<std::size_t>::max();
```

Constructors (inherited):

```
using std::map< size_t, KontsevichGraphSum<T> >::map;
```

Accessor methods:

```
size_t precision() const;
size_t precision(size_t new_precision);
```

Arithmetic operators:

```

KontsevichGraphSeries<T> operator()(std::vector< KontsevichGraphSeries<T> >)
                                     const;
KontsevichGraphSeries<T>& operator+=(const KontsevichGraphSeries<T>& rhs);
KontsevichGraphSeries<T>& operator-=(const KontsevichGraphSeries<T>& rhs);

```

Methods:

```

KontsevichGraphSeries<T> skew_symmetrization() const;
KontsevichGraphSeries<T> inverse() const;
KontsevichGraphSeries<T> gauge_transform(const KontsevichGraphSeries<T>& gauge);

```

Comparison operators:

```

bool operator==(int other) const;
bool operator!=(int other) const;

```

Methods that modify the graph series:

```

void reduce_mod_skew();

```

Static methods:

```

static KontsevichGraphSeries<T> from_istream(std::istream& is,
        std::function<T(std::string)> const& parser,
        std::function<bool(KontsevichGraph, size_t)> const& filter = nullptr);

```

Friend methods:

```

friend std::ostream& operator<< <>(std::ostream& os,
        const KontsevichGraphSeries<T>& series);

```

Functions defined outside the class:

```

KontsevichGraphSeries<T> operator+(KontsevichGraphSeries<T> lhs,
        const KontsevichGraphSeries<T>& rhs);
KontsevichGraphSeries<T> operator-(KontsevichGraphSeries<T> lhs,
        const KontsevichGraphSeries<T>& rhs);
std::ostream& operator<<(std::ostream&, const KontsevichGraphSeries<T>&);

```

APPENDIX C. ENCODING OF THE ENTIRE  $\star$ -PRODUCT MODULO  $\bar{o}(\hbar^4)$

In the following two tables, containing the sets of basic graphs and the  $\star$ -product expansion respectively, encodings of graphs (see Implementation 1 on p. 229) are followed by their coefficients.

TABLE 5. Basic sets of Kontsevich graphs, up to order 4, including zero graphs.

$\hbar^0$ :		2 4 1	0 1 0 2 2 3 2 4	w_4_35	2 4 1	0 3 0 4 1 5 1 2	w_4_92
2 0 1	1	2 4 1	0 1 0 2 2 3 3 4	w_4_36	2 4 1	0 3 0 4 1 5 2 3	w_4_93
$\hbar^1$ :		2 4 1	0 1 0 2 2 5 2 4	w_4_37	2 4 1	0 3 0 4 1 5 2 4	w_4_94
2 1 1	0 1	2 4 1	0 1 0 2 2 5 3 4	w_4_38	2 4 1	0 3 0 4 1 5 3 4	w_4_95
$\hbar^2$ :		2 4 1	0 1 0 2 3 5 3 4	w_4_39	2 4 1	0 3 0 4 2 3 1 2	w_4_96
2 2 1	0 1 0 2	2 4 1	0 1 0 4 0 3 2 3	w_4_40	2 4 1	0 3 0 4 2 3 1 3	w_4_97
2 2 1	0 3 1 2	2 4 1	0 1 0 4 0 5 2 3	w_4_41	2 4 1	0 3 0 4 2 3 1 4	w_4_98
$\hbar^3$ :		2 4 1	0 1 0 4 0 5 2 4	w_4_42	2 4 1	0 3 0 4 2 5 1 2	w_4_99
2 3 0	0 1 0 1 2 3	2 4 1	0 1 0 4 1 3 2 3	w_4_43	2 4 1	0 3 0 4 2 5 1 3	w_4_100
2 3 1	0 1 0 2 0 2	2 4 1	0 1 0 4 1 5 2 3	w_4_44	2 4 1	0 3 0 4 2 5 1 4	w_4_101
2 3 1	0 1 0 2 0 3	2 4 1	0 1 0 4 1 5 2 4	w_4_45	2 4 1	0 3 0 4 3 5 1 2	w_4_102
2 3 1	0 1 0 2 1 2	2 4 1	0 1 0 4 2 3 0 4	w_4_46	2 4 1	0 3 0 4 3 5 1 3	w_4_103
2 3 1	0 1 0 2 1 3	2 4 1	0 1 0 4 2 3 1 4	w_4_47	2 4 1	0 3 0 4 3 5 1 4	w_4_104
2 3 1	0 1 0 2 2 3	2 4 1	0 1 0 4 2 3 2 3	w_4_48	2 4 1	0 3 1 2 0 3 0 3	w_4_105
2 3 1	0 1 0 4 2 3	2 4 1	0 1 0 4 2 3 2 4	w_4_49	2 4 1	0 3 1 2 0 3 1 2	w_4_106
2 3 1	0 1 2 4 2 3	2 4 1	0 1 0 4 2 3 3 4	w_4_50	2 4 1	0 3 1 2 0 3 1 4	w_4_107
2 3 1	0 3 0 2 1 2	2 4 1	0 1 0 4 2 5 2 3	w_4_51	2 4 1	0 3 1 2 0 3 2 3	w_4_108
2 3 1	0 3 0 4 1 2	2 4 1	0 1 0 4 2 5 2 4	w_4_52	2 4 1	0 3 1 2 0 3 2 4	w_4_109
2 3 1	0 3 0 4 1 3	2 4 1	0 1 0 4 2 5 3 4	w_4_53	2 4 1	0 3 1 2 0 3 3 4	w_4_110
2 3 1	0 3 1 2 0 3	2 4 1	0 1 0 4 3 5 2 3	w_4_54	2 4 1	0 3 1 2 0 5 2 3	w_4_111
2 3 1	0 3 1 2 2 3	2 4 1	0 1 0 4 3 5 2 4	w_4_55	2 4 1	0 3 1 2 0 5 2 4	w_4_112
2 3 1	0 3 1 4 2 3	2 4 1	0 1 2 4 2 3 2 3	w_4_56	2 4 1	0 3 1 2 0 5 3 4	w_4_113
2 3 1	0 3 2 4 1 3	2 4 0	0 1 2 4 2 3 3 4	0	2 4 1	0 3 1 2 2 3 2 3	w_4_114
$\hbar^4$ :		2 4 1	0 1 2 4 2 5 2 3	w_4_57	2 4 1	0 3 1 2 2 3 2 4	w_4_115
2 4 1	0 1 0 1 0 2 2 3	2 4 1	0 1 2 4 2 5 3 4	w_4_58	2 4 1	0 3 1 2 2 5 2 4	w_4_116
2 4 1	0 1 0 1 0 2 3 4	2 4 0	0 1 2 4 3 5 2 4	0	2 4 1	0 3 1 2 2 5 3 4	w_4_117
2 4 0	0 1 0 1 0 5 2 3	2 4 1	0 1 2 4 3 5 3 4	w_4_59	2 4 1	0 3 1 4 0 5 1 2	w_4_118
2 4 1	0 1 0 1 2 3 2 3	2 4 1	0 3 0 2 0 2 1 2	w_4_60	2 4 1	0 3 1 4 0 5 2 3	w_4_119
2 4 1	0 1 0 1 2 3 2 4	2 4 1	0 3 0 2 0 2 1 3	w_4_61	2 4 1	0 3 1 4 0 5 2 4	w_4_120
2 4 1	0 1 0 1 2 5 3 4	2 4 1	0 3 0 2 0 2 1 4	w_4_62	2 4 1	0 3 1 4 0 5 3 4	w_4_121
2 4 1	0 1 0 2 0 2 0 2	2 4 1	0 3 0 2 0 5 1 2	w_4_63	2 4 1	0 3 1 4 2 3 0 3	w_4_122
2 4 1	0 1 0 2 0 2 0 3	2 4 1	0 3 0 2 1 2 1 2	w_4_64	2 4 1	0 3 1 4 2 3 0 4	w_4_123
2 4 1	0 1 0 2 0 2 1 2	2 4 1	0 3 0 2 1 2 1 3	w_4_65	2 4 1	0 3 1 4 2 3 1 4	w_4_124
2 4 1	0 1 0 2 0 2 1 3	2 4 1	0 3 0 2 1 2 1 4	w_4_66	2 4 1	0 3 1 4 2 3 2 3	w_4_125
2 4 1	0 1 0 2 0 2 2 3	2 4 1	0 3 0 2 1 2 2 3	w_4_67	2 4 1	0 3 1 4 2 3 2 4	w_4_126
2 4 0	0 1 0 2 0 2 3 4	2 4 1	0 3 0 2 1 2 2 4	w_4_68	2 4 1	0 3 1 4 2 3 3 4	w_4_127
2 4 1	0 1 0 2 0 3 0 3	2 4 1	0 3 0 2 1 2 3 4	w_4_69	2 4 0	0 3 1 4 2 5 0 3	0
2 4 1	0 1 0 2 0 3 0 4	2 4 0	0 3 0 2 1 5 2 3	0	2 4 1	0 3 1 4 2 5 0 4	w_4_128
2 4 1	0 1 0 2 0 3 1 2	2 4 1	0 3 0 2 1 5 2 4	w_4_70	2 4 1	0 3 1 4 2 5 1 4	w_4_129
2 4 1	0 1 0 2 0 3 1 3	2 4 1	0 3 0 4 0 2 1 2	w_4_71	2 4 1	0 3 1 4 2 5 2 3	w_4_130
2 4 1	0 1 0 2 0 3 1 4	2 4 1	0 3 0 4 0 5 1 2	w_4_72	2 4 1	0 3 1 4 2 5 2 4	w_4_131
2 4 1	0 1 0 2 0 3 2 3	2 4 1	0 3 0 4 0 5 1 3	w_4_73	2 4 1	0 3 1 4 2 5 3 4	w_4_132
2 4 1	0 1 0 2 0 3 2 4	2 4 1	0 3 0 4 0 5 1 4	w_4_74	2 4 1	0 3 1 4 3 5 0 4	w_4_133
2 4 1	0 1 0 2 0 3 3 4	2 4 1	0 3 0 4 1 2 0 3	w_4_75	2 4 1	0 3 1 4 3 5 1 4	w_4_134
2 4 1	0 1 0 2 0 5 1 2	2 4 1	0 3 0 4 1 2 0 4	w_4_76	2 4 1	0 3 1 4 3 5 2 3	w_4_135
2 4 1	0 1 0 2 0 5 1 3	2 4 1	0 3 0 4 1 2 1 2	w_4_77	2 4 1	0 3 1 4 3 5 2 4	w_4_136
2 4 1	0 1 0 2 0 5 2 3	2 4 1	0 3 0 4 1 2 1 3	w_4_78	2 4 1	0 3 1 4 3 5 3 4	w_4_137
2 4 1	0 1 0 2 0 5 2 4	2 4 1	0 3 0 4 1 2 1 4	w_4_79	2 4 0	0 3 2 4 0 3 1 3	0
2 4 1	0 1 0 2 0 5 3 4	2 4 1	0 3 0 4 1 2 2 3	w_4_80	2 4 1	0 3 2 4 1 3 0 3	w_4_138
2 4 1	0 1 0 2 1 2 2 3	2 4 1	0 3 0 4 1 2 2 4	w_4_81	2 4 1	0 3 2 4 1 3 2 3	w_4_139
2 4 1	0 1 0 2 1 2 3 4	2 4 1	0 3 0 4 1 2 3 4	w_4_82	2 4 1	0 3 2 4 1 3 2 4	w_4_140
2 4 1	0 1 0 2 1 3 1 3	2 4 1	0 3 0 4 1 3 0 3	w_4_83	2 4 1	0 3 2 4 1 5 2 3	w_4_141
2 4 1	0 1 0 2 1 3 1 4	2 4 1	0 3 0 4 1 3 0 4	w_4_84	2 4 1	0 3 2 4 1 5 2 4	w_4_142
2 4 1	0 1 0 2 1 3 2 3	2 4 1	0 3 0 4 1 3 1 2	w_4_85	2 4 1	0 3 2 4 1 5 3 4	w_4_143
2 4 1	0 1 0 2 1 3 2 4	2 4 1	0 3 0 4 1 3 1 3	w_4_86	2 4 1	0 3 2 4 1 5 3 1 3	w_4_144
2 4 1	0 1 0 2 1 3 3 4	2 4 1	0 3 0 4 1 3 1 4	w_4_87	2 4 1	0 3 2 4 2 3 1 4	w_4_145
2 4 1	0 1 0 2 1 5 2 3	2 4 1	0 3 0 4 1 3 2 3	w_4_88	2 4 1	0 3 2 4 2 5 1 3	w_4_146
2 4 1	0 1 0 2 1 5 2 4	2 4 1	0 3 0 4 1 3 2 4	w_4_89	2 4 1	0 3 2 4 3 5 1 3	w_4_147
2 4 1	0 1 0 2 1 5 3 4	2 4 1	0 3 0 4 1 3 3 4	w_4_90	2 4 1	0 3 2 4 3 5 1 4	w_4_148
2 4 1	0 1 0 2 2 3 2 3	2 4 1	0 3 0 4 1 5 0 4	w_4_91	2 4 1	0 3 4 5 1 5 2 3	w_4_149



TABLE 6 (continued).

2 4 1	0 3 0 2 1 2 1 3	8*w_4_65	2 4 1	0 3 1 2 0 5 1 2	16*w_4_107
2 4 1	0 3 1 4 1 3 0 4	8*w_4_65	2 4 1	0 3 1 2 0 3 2 3	16*w_4_108
2 4 1	0 3 0 2 1 2 1 4	16*w_4_66	2 4 1	0 3 1 2 1 2 2 3	-16*w_4_108
2 4 1	0 3 0 4 1 5 1 4	16*w_4_66	2 4 1	0 3 1 2 0 3 2 4	16*w_4_109
2 4 1	0 3 0 2 1 2 2 3	16*w_4_67	2 4 1	0 3 1 2 1 2 3 4	16*w_4_109
2 4 1	0 3 1 4 1 3 3 4	16*w_4_67	2 4 1	0 3 1 2 0 3 3 4	16*w_4_110
2 4 1	0 3 0 2 1 2 2 4	16*w_4_68	2 4 1	0 3 1 2 1 2 2 4	16*w_4_110
2 4 1	0 3 1 4 1 3 2 3	-16*w_4_68	2 4 1	0 3 1 2 0 5 2 3	16*w_4_111
2 4 1	0 3 0 2 1 2 3 4	16*w_4_69	2 4 1	0 3 1 2 1 5 2 3	-16*w_4_111
2 4 1	0 3 1 4 1 3 2 4	-16*w_4_69	2 4 1	0 3 1 2 0 5 2 4	16*w_4_112
2 4 1	0 3 0 2 1 5 2 4	16*w_4_70	2 4 1	0 3 1 2 1 5 3 4	16*w_4_112
2 4 1	0 3 2 4 1 5 1 4	-16*w_4_70	2 4 1	0 3 1 2 0 5 3 4	16*w_4_113
2 4 1	0 3 0 4 0 2 1 2	16*w_4_71	2 4 1	0 3 1 2 1 5 2 4	16*w_4_113
2 4 1	0 3 1 4 1 5 1 3	16*w_4_71	2 4 1	0 3 1 2 2 3 2 3	8*w_4_114
2 4 1	0 3 0 4 0 5 1 2	16*w_4_72	2 4 1	0 3 1 2 2 3 2 4	16*w_4_115
2 4 1	0 3 1 4 1 5 1 2	16*w_4_72	2 4 1	0 3 1 2 2 3 3 4	-16*w_4_115
2 4 1	0 3 0 4 0 5 1 3	16*w_4_73	2 4 1	0 3 1 2 2 5 2 4	8*w_4_116
2 4 1	0 3 1 4 1 2 1 3	16*w_4_73	2 4 1	0 3 1 2 3 5 3 4	8*w_4_116
2 4 1	0 3 0 4 0 5 1 4	16*w_4_74	2 4 1	0 3 1 2 2 5 3 4	16*w_4_117
2 4 1	0 3 1 2 1 3 1 4	16*w_4_74	2 4 1	0 3 1 4 0 5 1 2	8*w_4_118
2 4 1	0 3 0 4 1 2 0 3	16*w_4_75	2 4 1	0 3 1 4 0 5 2 3	16*w_4_119
2 4 1	0 3 1 4 1 2 1 4	16*w_4_75	2 4 1	0 3 1 4 2 5 1 2	-16*w_4_119
2 4 1	0 3 0 4 1 2 0 4	16*w_4_76	2 4 1	0 3 1 4 0 5 2 4	16*w_4_120
2 4 1	0 3 1 4 1 2 1 2	16*w_4_76	2 4 1	0 3 1 4 3 5 1 2	-16*w_4_120
2 4 1	0 3 0 4 1 2 1 2	16*w_4_77	2 4 1	0 3 1 4 0 5 3 4	16*w_4_121
2 4 1	0 3 1 4 1 2 0 3	16*w_4_77	2 4 1	0 3 1 4 2 3 1 2	16*w_4_121
2 4 1	0 3 0 4 1 2 1 3	16*w_4_78	2 4 1	0 3 1 4 2 3 0 3	16*w_4_122
2 4 1	0 3 1 4 0 5 1 3	16*w_4_78	2 4 1	0 3 2 4 1 2 1 2	-16*w_4_122
2 4 1	0 3 0 4 1 2 1 4	16*w_4_79	2 4 1	0 3 1 4 2 3 0 4	16*w_4_123
2 4 1	0 3 0 4 1 5 1 3	16*w_4_79	2 4 1	0 3 2 4 1 2 1 3	-16*w_4_123
2 4 1	0 3 0 4 1 2 2 3	16*w_4_80	2 4 1	0 3 1 4 2 3 1 4	16*w_4_124
2 4 1	0 3 1 4 1 2 3 4	16*w_4_80	2 4 1	0 3 2 4 1 2 0 3	-16*w_4_124
2 4 1	0 3 0 4 1 2 2 4	16*w_4_81	2 4 1	0 3 1 4 2 3 2 3	16*w_4_125
2 4 1	0 3 1 4 1 2 2 3	-16*w_4_81	2 4 1	0 3 2 4 1 2 2 4	16*w_4_125
2 4 1	0 3 0 4 1 2 3 4	16*w_4_82	2 4 1	0 3 1 4 2 3 2 4	16*w_4_126
2 4 1	0 3 1 4 1 2 2 4	-16*w_4_82	2 4 1	0 3 2 4 1 2 3 4	16*w_4_126
2 4 1	0 3 0 4 1 3 0 3	8*w_4_83	2 4 1	0 3 1 4 2 3 3 4	16*w_4_127
2 4 1	0 3 1 2 1 3 1 3	8*w_4_83	2 4 1	0 3 2 4 1 2 2 3	-16*w_4_127
2 4 1	0 3 0 4 1 3 0 4	16*w_4_84	2 4 1	0 3 1 4 2 5 0 4	16*w_4_128
2 4 1	0 3 1 2 1 2 1 3	16*w_4_84	2 4 1	0 3 4 5 1 2 1 3	16*w_4_128
2 4 1	0 3 0 4 1 3 1 2	16*w_4_85	2 4 1	0 3 1 4 2 5 1 4	16*w_4_129
2 4 1	0 3 1 2 0 5 1 3	16*w_4_85	2 4 1	0 3 2 4 1 5 0 3	-16*w_4_129
2 4 1	0 3 0 4 1 3 1 3	16*w_4_86	2 4 1	0 3 1 4 2 5 2 3	16*w_4_130
2 4 1	0 3 1 2 0 3 1 3	16*w_4_86	2 4 1	0 3 4 5 1 2 2 4	-16*w_4_130
2 4 1	0 3 0 4 1 3 1 4	16*w_4_87	2 4 1	0 3 1 4 2 5 2 4	16*w_4_131
2 4 1	0 3 0 4 1 3 2 3	16*w_4_88	2 4 1	0 3 4 5 1 2 3 4	-16*w_4_131
2 4 1	0 3 1 2 1 3 3 4	-16*w_4_88	2 4 1	0 3 1 4 2 5 3 4	16*w_4_132
2 4 1	0 3 0 4 1 3 2 4	16*w_4_89	2 4 1	0 3 4 5 1 2 2 3	16*w_4_132
2 4 1	0 3 1 2 1 3 2 4	-16*w_4_89	2 4 1	0 3 1 4 3 5 0 4	16*w_4_133
2 4 1	0 3 0 4 1 3 3 4	16*w_4_90	2 4 1	0 3 2 4 1 3 1 2	16*w_4_133
2 4 1	0 3 1 2 1 3 2 3	-16*w_4_90	2 4 1	0 3 1 4 3 5 1 4	16*w_4_134
2 4 1	0 3 0 4 1 5 0 4	16*w_4_91	2 4 1	0 3 2 4 0 3 1 2	16*w_4_134
2 4 1	0 3 1 2 1 2 1 4	16*w_4_91	2 4 1	0 3 1 4 3 5 2 3	16*w_4_135
2 4 1	0 3 0 4 1 5 1 2	16*w_4_92	2 4 1	0 3 2 4 2 5 1 2	-16*w_4_135
2 4 1	0 3 0 4 1 5 2 3	16*w_4_93	2 4 1	0 3 1 4 3 5 2 4	16*w_4_136
2 4 1	0 3 4 5 1 2 1 4	-16*w_4_93	2 4 1	0 3 2 4 3 5 1 2	-16*w_4_136
2 4 1	0 3 0 4 1 5 2 4	16*w_4_94	2 4 1	0 3 1 4 3 5 3 4	16*w_4_137
2 4 1	0 3 2 4 1 5 1 2	-16*w_4_94	2 4 1	0 3 2 4 2 3 1 2	16*w_4_137
2 4 1	0 3 0 4 1 5 3 4	16*w_4_95	2 4 1	0 3 2 4 1 3 0 3	16*w_4_138
2 4 1	0 3 2 4 1 2 1 4	-16*w_4_95	2 4 1	0 3 2 4 1 3 1 3	-16*w_4_138
2 4 1	0 3 0 4 2 3 1 2	16*w_4_96	2 4 1	0 3 2 4 1 3 2 3	16*w_4_139
2 4 1	0 3 1 4 1 5 3 4	16*w_4_96	2 4 1	0 3 2 4 1 3 3 4	16*w_4_139
2 4 1	0 3 0 4 2 3 1 3	16*w_4_97	2 4 1	0 3 2 4 1 3 2 4	16*w_4_140
2 4 1	0 3 1 4 3 5 1 3	-16*w_4_97	2 4 1	0 3 2 4 1 5 2 3	16*w_4_141
2 4 1	0 3 0 4 2 3 1 4	16*w_4_98	2 4 1	0 3 4 5 1 5 2 4	-16*w_4_141
2 4 1	0 3 4 5 1 3 1 4	-16*w_4_98	2 4 1	0 3 2 4 1 5 2 4	16*w_4_142
2 4 1	0 3 0 4 2 5 1 2	16*w_4_99	2 4 1	0 3 2 4 1 5 3 4	16*w_4_143
2 4 1	0 3 1 4 1 5 2 3	-16*w_4_99	2 4 1	0 3 2 4 2 5 1 4	16*w_4_143
2 4 1	0 3 0 4 2 5 1 3	16*w_4_100	2 4 1	0 3 2 4 2 3 1 3	16*w_4_144
2 4 1	0 3 1 4 2 5 1 3	-16*w_4_100	2 4 1	0 3 4 5 1 3 3 4	-16*w_4_144
2 4 1	0 3 0 4 2 5 1 4	16*w_4_101	2 4 1	0 3 2 4 2 3 1 4	16*w_4_145
2 4 1	0 3 2 4 1 5 1 3	-16*w_4_101	2 4 1	0 3 4 5 1 5 3 4	-16*w_4_145
2 4 1	0 3 0 4 3 5 1 2	16*w_4_102	2 4 1	0 3 2 4 2 5 1 3	16*w_4_146
2 4 1	0 3 1 4 1 5 2 4	-16*w_4_102	2 4 1	0 3 4 5 1 3 2 4	-16*w_4_146
2 4 1	0 3 0 4 3 5 1 3	16*w_4_103	2 4 1	0 3 2 4 3 5 1 3	16*w_4_147
2 4 1	0 3 1 4 2 3 1 3	-16*w_4_103	2 4 1	0 3 4 5 1 3 2 3	-16*w_4_147
2 4 1	0 3 0 4 3 5 1 4	16*w_4_104	2 4 1	0 3 2 4 3 5 1 4	16*w_4_148
2 4 1	0 3 2 4 1 3 1 4	-16*w_4_104	2 4 1	0 3 2 4 3 5 1 4	16*w_4_149
2 4 1	0 3 1 2 0 3 0 3	8*w_4_105			
2 4 1	0 3 1 2 1 2 1 2	8*w_4_105			
2 4 1	0 3 1 2 0 3 1 2	16*w_4_106			
2 4 1	0 3 1 2 0 3 1 4	16*w_4_107			

TABLE 7. Relations between weights of  $\hbar^4$ -basic graphs: 149 via 10.

$$\begin{aligned}
w_{4_1} &= -1/144 \\
w_{4_2} &= -1/288 \\
w_{4_3} &= 17/360 + 6w_{4_{108}} \\
w_{4_4} &= 49/2880 - 3w_{4_{104}} - w_{4_{107}} + (3w_{4_{108}})/2 \\
w_{4_5} &= -1/96 + 6w_{4_{104}} + 2w_{4_{107}} \\
w_{4_6} &= 1/80 \\
w_{4_7} &= 1/360 \\
w_{4_8} &= -1/240 \\
w_{4_9} &= -13/1440 \\
w_{4_{10}} &= -7/1440 \\
w_{4_{11}} &= 1/240 \\
w_{4_{12}} &= -1/720 \\
w_{4_{13}} &= 1/720 \\
w_{4_{14}} &= 1/480 \\
w_{4_{15}} &= -1/1440 \\
w_{4_{16}} &= 1/1440 \\
w_{4_{17}} &= -1/480 \\
w_{4_{18}} &= -1/360 \\
w_{4_{19}} &= -1/480 \\
w_{4_{20}} &= -1/240 \\
w_{4_{21}} &= -1/480 \\
w_{4_{22}} &= -1/720 \\
w_{4_{23}} &= 1/1440 \\
w_{4_{24}} &= 1/360 \\
w_{4_{25}} &= 53/1440 + 3w_{4_{100}} + 12w_{4_{103}} - 15w_{4_{104}} - w_{4_{107}} + 6w_{4_{108}} - 6w_{4_{109}} \\
w_{4_{26}} &= 1/120 \\
w_{4_{27}} &= 1/1440 \\
w_{4_{28}} &= -1/960 - (3w_{4_{108}})/2 \\
w_{4_{29}} &= -49/1440 - (3w_{4_{100}})/2 - 9w_{4_{103}} + (21w_{4_{104}})/2 + (3w_{4_{107}})/2 - (9w_{4_{108}})/2 + 3w_{4_{109}} \\
w_{4_{30}} &= 1/72 + 6w_{4_{103}} - 6w_{4_{104}} + 3w_{4_{108}} - 3w_{4_{109}} \\
w_{4_{31}} &= 61/2880 + (3w_{4_{100}})/2 + 6w_{4_{103}} - (15w_{4_{104}})/2 - w_{4_{107}}/2 + 3w_{4_{108}} - 3w_{4_{109}} \\
w_{4_{32}} &= 1/1440 \\
w_{4_{33}} &= 5/288 + 6w_{4_{103}} - 6w_{4_{104}} + 3w_{4_{108}} - 3w_{4_{109}} \\
w_{4_{34}} &= 1/96 + w_{4_{108}} \\
w_{4_{35}} &= -w_{4_{103}} \\
w_{4_{36}} &= -13/2880 - w_{4_{100}}/2 + (3w_{4_{104}})/2 + w_{4_{107}}/2 \\
w_{4_{37}} &= 0 \\
w_{4_{38}} &= 1/1440 - w_{4_{100}}/2 + w_{4_{103}} + (3w_{4_{104}})/2 + w_{4_{107}}/2 + w_{4_{108}}/2 \\
w_{4_{39}} &= 0 \\
w_{4_{40}} &= 0 \\
w_{4_{41}} &= 1/1440 \\
w_{4_{42}} &= 1/1440 \\
w_{4_{43}} &= 37/1440 + 6w_{4_{103}} - 6w_{4_{104}} - w_{4_{107}} + 3w_{4_{108}} - 3w_{4_{109}} \\
w_{4_{44}} &= 17/360 + 15w_{4_{103}} - 18w_{4_{104}} - 2w_{4_{107}} + 6w_{4_{108}} - 6w_{4_{109}} \\
w_{4_{45}} &= 7/1440 - 3w_{4_{104}} - w_{4_{107}} \\
w_{4_{46}} &= -1/480 \\
w_{4_{47}} &= 1/60 + 6w_{4_{103}} - 6w_{4_{104}} + 3w_{4_{108}} - 3w_{4_{109}} \\
w_{4_{48}} &= 11/1440 - w_{4_{100}}/2 - w_{4_{103}} + (5w_{4_{104}})/2 + w_{4_{107}}/2 + (3w_{4_{108}})/2 \\
w_{4_{49}} &= -w_{4_{104}} \\
w_{4_{50}} &= -1/192 - w_{4_{108}}/2 \\
w_{4_{51}} &= -w_{4_{103}} \\
w_{4_{52}} &= -1/1440 + w_{4_{100}}/2 - (3w_{4_{104}})/2 - w_{4_{107}}/2 - w_{4_{108}}/2 \\
w_{4_{53}} &= w_{4_{103}} \\
w_{4_{54}} &= -1/576 + w_{4_{103}} - w_{4_{104}} - w_{4_{108}}/2 \\
w_{4_{55}} &= w_{4_{104}} \\
w_{4_{56}} &= 0 \\
w_{4_{57}} &= 0 \\
w_{4_{58}} &= 0 \\
w_{4_{59}} &= 0 \\
w_{4_{60}} &= 0 \\
w_{4_{61}} &= 0 \\
w_{4_{62}} &= 0 \\
w_{4_{63}} &= 0 \\
w_{4_{64}} &= 0 \\
w_{4_{65}} &= 0 \\
w_{4_{66}} &= 0 \\
w_{4_{67}} &= 0 \\
w_{4_{68}} &= 0 \\
w_{4_{69}} &= 0 \\
w_{4_{70}} &= 0 \\
w_{4_{71}} &= 0 \\
w_{4_{72}} &= 1/1440 \\
w_{4_{73}} &= 1/1440 \\
w_{4_{74}} &= 1/1440 \\
w_{4_{75}} &= -1/480 \\
w_{4_{76}} &= -1/720 \\
w_{4_{77}} &= 1/180 + 3w_{4_{103}} - 3w_{4_{104}} - w_{4_{107}} \\
w_{4_{78}} &= -1/144 - 3w_{4_{103}} + 3w_{4_{104}} + w_{4_{107}} \\
w_{4_{79}} &= -1/1440 \\
w_{4_{80}} &= 1/80 + w_{4_{100}} - 3w_{4_{104}} + 3w_{4_{108}} - 2w_{4_{109}} - 2w_{4_{125}} \\
w_{4_{81}} &= 1/480 - w_{4_{100}}/2 + 2w_{4_{103}} + w_{4_{104}}/2 + w_{4_{107}}/2 + w_{4_{108}}/2 + 2w_{4_{125}}
\end{aligned}$$



TABLE 7 (continued).

$w_{4\_82}==1/2880 - w_{4\_103} - w_{4\_104} + w_{4\_108}/2 - w_{4\_109} - 2*w_{4\_125}$   
 $w_{4\_83}==1/480$   
 $w_{4\_84}==1/720$   
 $w_{4\_85}==1/180 - w_{4\_107}$   
 $w_{4\_86}==1/480$   
 $w_{4\_87}==1/1440$   
 $w_{4\_88}==1/96 + 4*w_{4\_103} - 4*w_{4\_104} + 2*w_{4\_108} - 2*w_{4\_109}$   
 $w_{4\_89}==1/240 + (3*w_{4\_100})/2 + 3*w_{4\_103} - (9*w_{4\_104})/2 + w_{4\_107}/2 + (3*w_{4\_108})/2 - 2*w_{4\_109}$   
 $w_{4\_90}==1/192 - w_{4\_108}/2$   
 $w_{4\_91}==1/720$   
 $w_{4\_92}==17/1440 + 6*w_{4\_103} - 6*w_{4\_104} - 2*w_{4\_107}$   
 $w_{4\_93}==3/320 - w_{4\_100}/2 + w_{4\_103} - (5*w_{4\_104})/2 + w_{4\_107}/2 + 2*w_{4\_108} - 2*w_{4\_109} + w_{4\_119}$   
 $w_{4\_94}==1/1440 - w_{4\_100}/2 - w_{4\_102} + w_{4\_103} - (3*w_{4\_104})/2 + w_{4\_107}/2 + w_{4\_108}/2 - w_{4\_109}$   
 $w_{4\_95}==1/576 + w_{4\_103} - w_{4\_104} - w_{4\_108}/2$   
 $w_{4\_96}==0$   
 $w_{4\_97}==0$   
 $w_{4\_98}==0$   
 $w_{4\_99}==7/2880 - w_{4\_100}/2 + w_{4\_103} + w_{4\_104}/2 - w_{4\_107}/2 - w_{4\_108} + w_{4\_109} - w_{4\_119}$   
 $w_{4\_105}==1/160$   
 $w_{4\_106}==13/1440$   
 $w_{4\_110}==1/288 - 2*w_{4\_103} + 2*w_{4\_104} - w_{4\_108} + w_{4\_109}$   
 $w_{4\_111}==17/2880 - w_{4\_100}/2 - 2*w_{4\_103} + (5*w_{4\_104})/2 - w_{4\_107}/2 - w_{4\_108} + w_{4\_109}$   
 $w_{4\_112}==7/576 - 4*w_{4\_103} + 4*w_{4\_104} - 2*w_{4\_108} + 2*w_{4\_109}$   
 $w_{4\_113}==1/192 - 2*w_{4\_103} + 2*w_{4\_104} - w_{4\_108} + w_{4\_109}$   
 $w_{4\_114}==1/360 + w_{4\_108}$   
 $w_{4\_115}==23/5760 - w_{4\_100}/2 + w_{4\_103} + (3*w_{4\_108})/4$   
 $w_{4\_116}==0$   
 $w_{4\_117}==19/2880 + w_{4\_100} - 2*w_{4\_103} - w_{4\_108}$   
 $w_{4\_118}==31/1440 - 12*w_{4\_103} + 12*w_{4\_104} + 4*w_{4\_107}$   
 $w_{4\_120}==1/96 - w_{4\_100} - w_{4\_102} + 2*w_{4\_103} - 2*w_{4\_108} + w_{4\_109}$   
 $w_{4\_121}==1/288 + 2*w_{4\_103} - 2*w_{4\_104} - w_{4\_108}$   
 $w_{4\_122}==2*w_{4\_103}$   
 $w_{4\_123}==7/2880 + w_{4\_100}/2 - w_{4\_103} + w_{4\_104}/2 - w_{4\_107}/2 - w_{4\_108} + w_{4\_109}$   
 $w_{4\_124}==1/144 + w_{4\_100} + w_{4\_103} - 2*w_{4\_104} + w_{4\_108} - w_{4\_109}$   
 $w_{4\_126}==29/5760 + w_{4\_100}/2 - w_{4\_103} + (5*w_{4\_108})/4 - w_{4\_109} - w_{4\_125}$   
 $w_{4\_127}==1/640 + w_{4\_103} + w_{4\_104}/2 - (3*w_{4\_108})/4 + w_{4\_109}/2 + w_{4\_125}$   
 $w_{4\_128}==1/144 + w_{4\_101} - 2*w_{4\_103} + 3*w_{4\_104} - w_{4\_108} + w_{4\_109}$   
 $w_{4\_129}==1/144 + w_{4\_101} + 2*w_{4\_103} - 3*w_{4\_104} + w_{4\_108} - w_{4\_109}$   
 $w_{4\_130}==7/1920 - w_{4\_100}/2 + w_{4\_103} + w_{4\_107}/2 + (3*w_{4\_108})/4 - w_{4\_109}/2 + w_{4\_119} + w_{4\_125}$   
 $w_{4\_131}==23/5760 - w_{4\_100}/4 + w_{4\_101}/2 - w_{4\_102}/2 + w_{4\_103}/2 - (5*w_{4\_104})/4 + w_{4\_107}/4 + w_{4\_108} - w_{4\_109} + w_{4\_119}/2 - w_{4\_125}$   
 $w_{4\_132}==1/240 + w_{4\_101}/2 + w_{4\_103}/2 - w_{4\_108} + w_{4\_109}/2 + w_{4\_125}$   
 $w_{4\_133}==2*w_{4\_104}$   
 $w_{4\_134}==0$   
 $w_{4\_135}==1/360 - w_{4\_100}/4 - w_{4\_102}/2 + w_{4\_104}/4 + w_{4\_107}/4 - w_{4\_108}/4 + w_{4\_119}/2$   
 $w_{4\_136}==7/1440 - w_{4\_100}/2 - w_{4\_102} + w_{4\_103} - w_{4\_104}/2 - w_{4\_108} + w_{4\_109}/2$   
 $w_{4\_137}==0$   
 $w_{4\_138}==1/144 - 2*w_{4\_103} + 2*w_{4\_104} - w_{4\_108} + w_{4\_109}$   
 $w_{4\_139}==1/1920 + w_{4\_103} - (3*w_{4\_104})/2 + w_{4\_108}/4 - w_{4\_109}/2$   
 $w_{4\_140}==1/1440 + w_{4\_100} + 2*w_{4\_103} - 5*w_{4\_104} - w_{4\_109}$   
 $w_{4\_141}==w_{4\_100}/4 - w_{4\_101}/2 - w_{4\_102}/2 - w_{4\_103}/2 + w_{4\_104}/4 + w_{4\_107}/4 + w_{4\_108}/4 - w_{4\_109}/2 + w_{4\_119}/2$   
 $w_{4\_142}==1/5760 - w_{4\_102} + 2*w_{4\_103} - 3*w_{4\_104} - w_{4\_109}$   
 $w_{4\_143}==7/1440 + w_{4\_101}/2 + (5*w_{4\_103})/2 - (5*w_{4\_104})/2 + (3*w_{4\_108})/4 - w_{4\_109}$   
 $w_{4\_144}==0$   
 $w_{4\_145}==0$   
 $w_{4\_146}==13/5760 + w_{4\_100}/2 - w_{4\_101}/2 + (3*w_{4\_103})/2 - 2*w_{4\_104} + w_{4\_108}/4 - w_{4\_109}/2$   
 $w_{4\_147}==1/320 - w_{4\_101}/2 + (3*w_{4\_103})/2 - w_{4\_104} + w_{4\_108}/2 - w_{4\_109}/2$   
 $w_{4\_148}==11/1920 + 2*w_{4\_103} - w_{4\_104} + w_{4\_108} - w_{4\_109}$   
 $w_{4\_149}==11/2880 + w_{4\_100}/2 + w_{4\_104}/2 - w_{4\_107}/2 - w_{4\_108} + w_{4\_109} - w_{4\_119}$

APPENDIX D. ENCODING OF THE ASSOCIATOR OF THE  $\star$ -PRODUCT MODULO  $\bar{o}(\hbar^4)$ 

Encodings of graphs (see Implementation 1 on p. 229) are followed by their coefficients, in the following table containing the expansion of the associator  $(f \star g) \star h - f \star (g \star h)$ .

The table below contains the output of the command

```
$ reduce_mod_jacobi assoc4_intermsof10_part100.txt
```

as described in Implementation 16.

The first part of the output lists the graph series  $S^{(1)} - \diamond$ , reduced modulo skew-symmetry, wherein the coefficients of  $\diamond$  are still undetermined. The second part of the output (after the blank line) specifies the coefficients such that  $S^{(1)} = \diamond$ . Every coefficient in the second part is preceded by the encoding of the Leibniz graph that specifies a differential operator acting on the Jacobi identity. Such a differential operator expands into a sum of graphs that can be read in the first part of the output.



TABLE 8 (part 2).

3 4 1	0 2 0 3 0 4 1 5	-16*w_4_15+16*w_4_12	3 4 1	0 1 0 1 2 3 3 4	-16*w_4_1
3 4 1	0 2 0 3 0 6 1 4	16*w_4_12-16*w_4_20	3 4 1	0 1 0 2 1 4 3 5	-16*w_4_2
3 4 1	0 2 0 5 0 6 1 3	16*w_4_12+16*w_4_27	3 4 1	0 1 0 3 1 3 2 3	16*w_4_8
3 4 1	0 4 0 5 0 6 1 2	16*w_4_12	3 4 1	0 1 0 3 1 3 2 5	16*w_4_13
3 4 1	0 1 0 2 0 4 3 5	-16*w_4_2	3 4 1	0 1 0 5 1 6 2 3	16*w_4_15
3 4 1	0 1 0 3 0 4 2 3	-16*w_4_13	3 4 1	0 1 0 3 1 3 2 4	16*w_4_19
3 4 1	0 1 0 3 0 4 2 4	-16*w_4_14	3 4 1	0 1 0 5 2 3 1 4	16*w_4_20
3 4 1	0 1 0 3 0 4 2 5	-16*w_4_15	# 1 2 2		
3 4 1	0 1 0 5 0 6 2 3	16*w_4_27	3 4 1	0 1 1 3 2 3 2 3	-1/6+8*w_4_6
# 1 1 3			3 4 1	0 1 1 3 2 3 2 4	-1/3+16*w_4_7
3 4 1	0 1 2 3 2 3 2 3	-1/6+8/3*w_4_6	3 4 1	0 1 1 3 2 4 2 4	-1/6+8*w_4_11
3 4 1	0 1 2 3 2 3 2 4	-1/3+16*w_4_7	3 4 1	0 4 1 3 1 2 2 4	1/6
3 4 1	0 1 2 3 2 4 2 4	-1/6+8*w_4_11	3 4 1	0 1 1 3 2 3 2 5	-1/9+16*w_4_7
3 4 1	0 2 1 2 2 3 3 4	1/3+16*w_4_1	3 4 1	0 1 1 3 2 4 2 5	-1/9+16*w_4_12
3 4 1	0 2 1 2 2 4 3 4	1/9-16*w_4_1	3 4 1	0 1 1 2 2 3 3 5	-1/6
3 4 1	0 2 1 2 2 3 4 5	16*w_4_2	3 4 1	0 1 1 5 2 3 2 3	1/3+16*w_4_7
3 4 1	0 2 1 2 2 4 3 5	1/9+16*w_4_2	3 4 1	0 1 1 2 2 6 3 5	-1/6
3 4 1	0 2 1 3 2 3 2 3	-8*w_4_8+8*w_4_6	3 4 1	0 1 1 5 2 4 2 3	1/9
3 4 1	0 4 1 2 2 4 2 4	-1/6-8*w_4_8	3 4 1	0 1 1 2 2 4 3 5	-1/18
3 4 1	0 2 1 5 2 3 2 3	-16*w_4_9+16*w_4_7	3 4 1	0 1 1 5 2 6 2 3	1/6+16*w_4_12
3 4 1	0 4 2 5 1 2 2 5	-1/3-16*w_4_9	3 4 1	0 1 1 2 2 6 3 4	-1/6
3 4 1	0 2 1 3 2 3 2 5	-16*w_4_13+16*w_4_7	3 4 1	0 1 1 5 2 3 2 4	1/6+16*w_4_12
3 4 1	0 4 1 2 2 4 2 5	-16*w_4_13	3 4 1	0 4 1 2 1 2 3 4	-1/6
3 4 1	0 2 1 5 2 3 2 5	16*w_4_11-16*w_4_14	3 4 1	0 4 2 3 1 2 1 4	-1/6
3 4 1	0 4 2 5 1 2 2 4	-16*w_4_14	3 4 1	0 2 1 3 1 6 2 5	-1/9
3 4 1	0 2 1 5 2 6 2 3	-16*w_4_15+16*w_4_12	3 4 1	0 4 1 2 1 6 2 5	1/9
3 4 1	0 4 2 5 2 6 1 2	-16*w_4_15	3 4 1	0 2 1 2 1 3 3 4	1/3+16*w_4_1
3 4 1	0 2 1 3 2 3 2 4	-16*w_4_19+16*w_4_7	3 4 1	0 2 1 2 1 4 3 4	-1/9-16*w_4_1
3 4 1	0 4 1 2 2 3 2 4	-1/9-16*w_4_19	3 4 1	0 4 1 2 1 2 4 5	16*w_4_1
3 4 1	0 2 1 5 2 3 2 4	16*w_4_12-16*w_4_20	3 4 1	0 2 1 2 1 3 4 5	16*w_4_2
3 4 1	0 4 2 5 1 2 2 3	-1/9-16*w_4_20	3 4 1	0 2 1 2 1 4 3 5	-1/9+16*w_4_2
3 4 1	0 2 1 3 2 4 2 4	8*w_4_11+8*w_4_26	3 4 1	0 4 1 2 1 2 3 5	-1/3-16*w_4_2
3 4 1	0 4 1 2 2 3 2 3	-1/6+8*w_4_26	3 4 1	0 2 1 3 1 3 2 3	8*w_4_8+8*w_4_6
3 4 1	0 2 1 3 2 4 2 5	16*w_4_12+16*w_4_27	3 4 1	0 4 1 2 1 4 2 4	16*w_4_8
3 4 1	0 4 1 2 2 3 2 5	16*w_4_27	3 4 1	0 2 1 3 1 3 2 4	16*w_4_9+16*w_4_7
3 4 1	0 1 2 3 2 4 2 5	16*w_4_12	3 4 1	0 4 1 2 1 4 2 3	1/9+16*w_4_9
# 2 2 1			3 4 1	0 4 1 2 1 4 2 5	1/6+16*w_4_9
3 4 1	0 1 0 5 1 4 2 3	1/9	3 4 1	0 2 1 3 1 4 2 3	16*w_4_13+16*w_4_7
3 4 1	0 1 0 5 1 4 2 5	1/6	3 4 1	0 4 1 2 1 3 2 4	-1/9+16*w_4_13
3 4 1	0 1 0 5 1 3 2 3	1/6+16*w_4_9	3 4 1	0 4 1 5 1 2 2 5	16*w_4_13
3 4 1	0 1 0 5 1 3 2 4	1/6+16*w_4_20	3 4 1	0 2 1 3 1 4 2 4	16*w_4_11+16*w_4_14
3 4 1	0 1 0 5 1 3 2 5	1/6+16*w_4_14	3 4 1	0 4 1 2 1 3 2 3	-1/3+16*w_4_14
3 4 1	0 1 0 3 1 4 2 3	1/6+16*w_4_19	3 4 1	0 4 1 5 1 2 2 4	1/6+16*w_4_14
3 4 1	0 1 0 3 1 4 2 4	1/6-16*w_4_26	3 4 1	0 2 1 3 1 4 2 5	16*w_4_15+16*w_4_12
3 4 1	0 1 0 3 1 4 2 5	1/6-16*w_4_27	3 4 1	0 4 1 2 1 3 2 5	16*w_4_15
3 4 1	0 1 0 1 2 3 3 5	-1/6	3 4 1	0 4 1 5 1 2 2 3	1/9+16*w_4_15
3 4 1	0 1 0 1 2 3 4 5	-1/3-16*w_4_2	3 4 1	0 2 1 3 1 6 2 3	16*w_4_19+16*w_4_7
3 4 1	0 1 0 3 1 2 3 5	1/9	3 4 1	0 4 1 2 1 6 2 4	1/6+16*w_4_19
3 4 1	0 1 0 3 1 2 4 5	-1/18	3 4 1	0 4 2 5 1 2 1 5	16*w_4_19
3 4 1	0 1 0 3 1 6 2 3	-1/9+16*w_4_13	3 4 1	0 2 1 3 1 6 2 4	16*w_4_12+16*w_4_20
3 4 1	0 1 0 3 1 6 2 4	-1/9-16*w_4_27	3 4 1	0 4 1 2 1 6 2 3	16*w_4_20
3 4 1	0 1 0 2 1 3 3 4	-1/9-16*w_4_1	3 4 1	0 4 2 5 1 6 1 2	1/6+16*w_4_20
3 4 1	0 1 0 2 1 3 4 5	1/9-16*w_4_2	3 4 1	0 2 1 5 2 3 1 5	8*w_4_11-8*w_4_26
3 4 1	0 1 0 5 2 3 1 3	1/9+16*w_4_9	3 4 1	0 4 2 5 1 2 1 4	1/6-16*w_4_26
3 4 1	0 1 0 5 2 6 1 3	1/9+16*w_4_15	3 4 1	0 2 1 5 1 6 2 3	16*w_4_12-16*w_4_27
3 4 1	0 2 0 5 1 4 1 3	-1/9	3 4 1	0 4 1 5 2 6 1 2	1/6-16*w_4_27
3 4 1	0 4 1 2 0 6 1 5	1/9	3 4 1	0 4 2 5 1 2 1 3	-1/9-16*w_4_27
3 4 1	0 1 0 1 2 6 3 5	-1/6	3 4 1	0 4 1 5 2 4 1 2	1/6
3 4 1	0 1 0 2 1 4 3 4	1/3+16*w_4_1	3 4 1	0 1 1 2 2 4 3 4	1/9
3 4 1	0 1 0 5 2 3 1 5	-1/3+16*w_4_14	3 4 1	0 4 3 5 1 2 1 2	-1/6
3 4 1	0 1 0 5 1 2 4 5	-1/6	3 4 1	0 4 2 3 1 2 1 3	-1/6
3 4 1	0 1 0 5 2 4 1 5	-1/6	3 4 1	0 1 1 5 2 3 2 5	16*w_4_11
3 4 1	0 2 0 3 1 3 1 3	8*w_4_8+8*w_4_6	# 2 1 2		
3 4 1	0 4 1 2 0 4 1 4	-1/6+8*w_4_6	3 4 1	0 1 0 3 2 3 2 3	1/6+8*w_4_8
3 4 1	0 2 0 3 1 3 1 4	16*w_4_19+16*w_4_7	3 4 1	0 1 0 3 2 3 2 4	1/3+16*w_4_19
3 4 1	0 2 0 3 1 3 1 5	16*w_4_13+16*w_4_7	3 4 1	0 1 0 3 2 4 2 4	1/6-8*w_4_26
3 4 1	0 2 0 5 1 3 1 3	16*w_4_9+16*w_4_7	3 4 1	0 2 0 5 1 4 2 5	1/6
3 4 1	0 4 1 2 0 4 1 3	1/3+16*w_4_7	3 4 1	0 1 0 3 2 3 2 5	1/9+16*w_4_13
3 4 1	0 4 0 5 1 2 1 5	-1/9+16*w_4_7	3 4 1	0 1 0 3 2 4 2 5	1/9-16*w_4_27
3 4 1	0 4 1 2 0 6 1 4	-1/3+16*w_4_7	3 4 1	0 1 0 2 2 3 3 4	-1/3-16*w_4_1
3 4 1	0 2 0 3 1 4 1 4	8*w_4_11-8*w_4_26	3 4 1	0 1 0 2 2 3 3 5	-1/6
3 4 1	0 2 0 5 1 3 1 5	16*w_4_11+16*w_4_14	3 4 1	0 1 0 5 2 3 2 3	1/3+16*w_4_9
3 4 1	0 4 0 5 1 2 1 4	16*w_4_11	3 4 1	0 1 0 2 2 6 3 5	-1/6
3 4 1	0 4 1 5 1 2 0 4	-1/6+8*w_4_11	3 4 1	0 1 0 5 2 4 2 3	1/9
3 4 1	0 2 0 3 1 4 1 5	16*w_4_12-16*w_4_27	3 4 1	0 1 0 2 2 4 3 5	-1/6-16*w_4_2
3 4 1	0 2 0 5 1 3 1 4	16*w_4_12+16*w_4_20	3 4 1	0 1 0 5 2 6 2 3	1/6+16*w_4_15
3 4 1	0 2 0 5 1 6 1 3	16*w_4_15+16*w_4_12	3 4 1	0 1 0 2 2 6 3 4	-1/6
3 4 1	0 4 0 5 1 2 1 3	1/6+16*w_4_12	3 4 1	0 1 0 5 2 3 2 4	1/6+16*w_4_20
3 4 1	0 4 1 2 0 6 1 3	1/6+16*w_4_12	3 4 1	0 2 0 5 1 2 4 5	-1/6
3 4 1	0 4 0 5 1 6 1 2	-1/9+16*w_4_12	3 4 1	0 2 0 5 2 4 1 5	-1/6
3 4 1	0 1 0 5 1 6 2 5	1/6	3 4 1	0 2 0 5 2 4 1 3	-1/9
3 4 1	0 1 0 5 4 6 1 2	-1/6	3 4 1	0 4 1 2 0 6 2 5	1/9
3 4 1	0 1 0 5 3 6 1 2	-1/6	3 4 1	0 2 0 3 1 2 3 5	16*w_4_1
3 4 1	0 1 0 5 2 4 1 4	-1/6	3 4 1	0 2 0 5 1 2 3 5	-1/3-16*w_4_1

TABLE 8 (part 3).

3 4 1	0 2 0 3 1 2 4 5	-1/6-16*w_4_2	3 4 1	0 2 1 5 3 4 1 5	32*w_4_46
3 4 1	0 2 0 5 1 2 3 4	16*w_4_2	3 4 1	0 4 3 5 1 2 1 4	-16*w_4_46
3 4 1	0 2 0 3 1 3 2 3	32*w_4_8	3 4 1	0 4 5 6 1 2 1 4	16*w_4_46
3 4 1	0 4 1 2 0 4 2 4	1/6+8*w_4_8	3 4 1	0 4 1 3 1 4 2 4	16*w_4_60-16*w_4_83
3 4 1	0 2 0 3 1 3 2 4	16*w_4_9+16*w_4_19	3 4 1	0 4 1 3 1 4 2 3	16*w_4_61-16*w_4_84
3 4 1	0 2 0 3 1 3 2 5	16*w_4_9+16*w_4_13	3 4 1	0 4 1 3 1 4 2 5	-16*w_4_74+16*w_4_62
3 4 1	0 4 1 2 0 4 2 3	1/3+16*w_4_9	3 4 1	0 4 1 5 1 4 2 3	-16*w_4_63+16*w_4_62
3 4 1	0 2 0 3 1 4 2 3	16*w_4_19+16*w_4_13	3 4 1	0 4 2 5 1 6 1 5	16*w_4_63-16*w_4_62
3 4 1	0 2 0 5 1 3 2 3	16*w_4_9+16*w_4_13	3 4 1	0 4 1 5 1 3 2 3	-16*w_4_76+16*w_4_71
3 4 1	0 4 0 5 1 2 2 5	1/9+16*w_4_13	3 4 1	0 4 1 5 1 3 2 5	16*w_4_71-16*w_4_75
3 4 1	0 2 0 3 1 4 2 4	-16*w_4_26+16*w_4_14	3 4 1	0 4 1 5 1 3 2 4	-16*w_4_73+16*w_4_71
3 4 1	0 2 0 5 1 3 2 5	32*w_4_14	3 4 1	0 4 1 5 1 6 2 4	16*w_4_73-16*w_4_71
3 4 1	0 4 0 5 1 2 2 4	16*w_4_14	3 4 1	0 4 1 5 2 3 1 3	-16*w_4_76+16*w_4_73
3 4 1	0 2 0 3 1 4 2 5	16*w_4_15-16*w_4_27	3 4 1	0 4 2 5 1 3 1 5	16*w_4_73-16*w_4_75
3 4 1	0 2 0 5 1 3 2 4	16*w_4_15+16*w_4_20	3 4 1	0 4 1 5 1 6 2 5	16*w_4_74-16*w_4_62
3 4 1	0 4 0 5 1 2 2 3	1/6+16*w_4_15	3 4 1	0 4 1 5 2 4 1 3	-1/18-16*w_4_63+16*w_4_74
3 4 1	0 2 0 3 1 6 2 3	16*w_4_19+16*w_4_13	3 4 1	0 4 2 3 1 3 1 5	16*w_4_74-16*w_4_91
3 4 1	0 2 0 5 2 3 1 3	16*w_4_9+16*w_4_19	3 4 1	0 4 1 5 2 3 1 4	-16*w_4_73+16*w_4_75
3 4 1	0 4 1 2 0 6 2 4	1/3+16*w_4_19	3 4 1	0 4 2 5 1 3 1 3	-16*w_4_76+16*w_4_75
3 4 1	0 2 0 3 1 6 2 4	16*w_4_20-16*w_4_27	3 4 1	0 4 1 5 2 6 1 4	-16*w_4_71+16*w_4_75
3 4 1	0 2 0 5 2 6 1 3	16*w_4_15+16*w_4_20	3 4 1	0 4 1 5 2 3 1 5	16*w_4_76-16*w_4_75
3 4 1	0 4 1 2 0 6 2 3	1/6+16*w_4_20	3 4 1	0 4 2 5 1 3 1 4	16*w_4_76-16*w_4_73
3 4 1	0 2 0 5 2 3 1 5	-16*w_4_26+16*w_4_14	3 4 1	0 4 2 5 1 6 1 4	16*w_4_76-16*w_4_71
3 4 1	0 4 2 5 1 2 0 4	1/6-8*w_4_26	3 4 1	0 4 1 5 2 4 1 4	-16*w_4_60+16*w_4_83
3 4 1	0 2 0 5 1 6 2 3	16*w_4_15-16*w_4_27	3 4 1	0 4 2 3 1 3 1 3	-8*w_4_105+8*w_4_83
3 4 1	0 2 0 5 2 3 1 4	16*w_4_20-16*w_4_27	3 4 1	0 4 1 5 2 4 1 5	1/6-16*w_4_61+16*w_4_84
3 4 1	0 4 0 5 2 6 1 2	1/9-16*w_4_27	3 4 1	0 4 2 5 1 4 1 5	-16*w_4_61+16*w_4_84
3 4 1	0 2 0 5 1 6 2 5	1/6	3 4 1	0 4 1 5 2 6 1 5	1/6+16*w_4_91-16*w_4_62
3 4 1	0 2 0 5 4 6 1 2	-1/6	3 4 1	0 4 2 5 1 4 1 3	-1/18-16*w_4_63+16*w_4_91
3 4 1	0 2 0 5 3 6 1 2	-1/6	3 4 1	0 4 2 3 1 4 1 5	-16*w_4_74+16*w_4_91
3 4 1	0 2 0 5 2 4 1 4	-1/6	3 4 1	0 4 2 3 1 4 1 4	8*w_4_105-8*w_4_83
3 4 1	0 1 0 2 2 4 3 4	16*w_4_1	3 4 1	0 4 2 5 1 4 1 4	1/6+16*w_4_105-16*w_4_60
3 4 1	0 1 0 2 2 3 4 5	-16*w_4_2	3 4 1	0 4 1 2 1 4 3 5	-1/9
3 4 1	0 1 0 2 2 3 4 5	16*w_4_14	3 4 1	0 1 1 5 2 4 3 4	1/18+16*w_4_40
# 1 2 1			3 4 1	0 1 1 2 3 4 3 5	-1/6
3 4 1	0 1 1 3 2 3 3 4	1/6+16*w_4_10	3 4 1	0 1 1 5 2 3 3 4	16*w_4_17
3 4 1	0 1 1 3 2 4 3 4	1/6+16*w_4_16	3 4 1	0 1 1 5 3 6 2 3	16*w_4_21
3 4 1	0 1 1 3 2 6 3 4	1/6+16*w_4_21	3 4 1	0 1 1 5 3 4 2 3	16*w_4_22
3 4 1	0 4 1 3 1 3 2 3	-1/6-16*w_4_105+16*w_4_60	3 4 1	0 1 1 5 4 6 2 3	-16*w_4_23
3 4 1	0 4 1 3 1 3 2 4	-1/6+16*w_4_61-16*w_4_84	3 4 1	0 1 1 5 2 6 3 4	16*w_4_41
3 4 1	0 4 1 3 1 3 2 5	-1/6-16*w_4_91+16*w_4_62	3 4 1	0 1 1 5 3 6 2 4	16*w_4_41
3 4 1	0 1 1 3 2 3 3 5	1/9+16*w_4_10	3 4 1	0 1 1 5 3 4 2 4	16*w_4_42
3 4 1	0 1 1 3 2 3 4 5	1/9	3 4 1	0 1 1 5 3 4 2 5	16*w_4_46
3 4 1	0 1 1 3 2 4 3 5	1/9+16*w_4_17	3 4 1	0 1 1 5 3 6 2 5	16*w_4_46
3 4 1	0 1 1 3 2 4 4 5	1/9+16*w_4_18	# 2 1 1		
3 4 1	0 4 1 3 1 2 3 5	1/9-16*w_4_40	3 4 1	0 1 0 3 2 3 3 4	1/6-16*w_4_24
3 4 1	0 4 1 3 1 2 4 5	-1/18-16*w_4_40	3 4 1	0 1 0 3 2 4 3 4	1/6-16*w_4_28
3 4 1	0 4 1 3 1 6 2 3	1/18+16*w_4_63-16*w_4_91	3 4 1	0 1 0 3 2 6 3 4	1/6-16*w_4_31
3 4 1	0 4 1 3 1 6 2 4	1/18+16*w_4_63-16*w_4_74	3 4 1	0 4 1 3 0 4 2 3	1/6-16*w_4_106+16*w_4_61
3 4 1	0 1 1 3 2 6 3 5	1/18+16*w_4_22	3 4 1	0 4 1 3 0 4 2 4	1/6+16*w_4_60-16*w_4_86
3 4 1	0 1 1 3 2 6 4 5	1/18+16*w_4_23	3 4 1	0 4 1 3 0 4 2 5	1/6-16*w_4_107+16*w_4_62
3 4 1	0 1 1 5 2 3 3 5	-1/3+16*w_4_16	3 4 1	0 1 0 3 2 3 3 5	-1/9+16*w_4_24
3 4 1	0 1 1 2 3 4 4 5	-1/6	3 4 1	0 1 0 3 2 3 4 5	-1/9-16*w_4_25
3 4 1	0 1 1 5 2 3 4 5	-1/6-16*w_4_18	3 4 1	0 1 0 3 2 4 3 5	-1/9-16*w_4_29
3 4 1	0 1 1 5 2 4 3 5	-1/9+16*w_4_40	3 4 1	0 1 0 3 2 4 4 5	-1/9-16*w_4_30
3 4 1	0 1 1 5 2 6 3 5	-1/6+16*w_4_42	3 4 1	0 2 0 5 1 4 3 4	1/18+16*w_4_40-16*w_4_43
3 4 1	0 2 1 3 1 3 3 4	32*w_4_10	3 4 1	0 2 0 5 1 4 3 5	1/18+16*w_4_40+16*w_4_43
3 4 1	0 4 1 2 1 4 3 4	-1/9-16*w_4_10	3 4 1	0 4 1 3 0 6 2 3	1/18+16*w_4_63-16*w_4_107
3 4 1	0 4 1 2 1 4 4 5	1/6+16*w_4_10	3 4 1	0 4 1 3 0 6 2 4	1/18+16*w_4_63-16*w_4_85
3 4 1	0 2 1 3 1 4 3 4	32*w_4_16	3 4 1	0 1 0 3 2 6 3 5	-1/18-16*w_4_32
3 4 1	0 4 1 2 1 3 3 4	1/3-16*w_4_16	3 4 1	0 1 0 3 2 6 4 5	-1/18-16*w_4_33
3 4 1	0 4 1 5 1 2 4 5	-1/6-16*w_4_16	3 4 1	0 1 0 2 3 4 3 4	1/6-8*w_4_3
3 4 1	0 2 1 3 1 4 3 5	32*w_4_17	3 4 1	0 1 0 5 2 3 3 5	-1/3+16*w_4_28
3 4 1	0 4 1 2 1 3 4 5	16*w_4_17	3 4 1	0 1 0 2 3 4 4 5	-1/6+16*w_4_4
3 4 1	0 4 1 5 1 2 3 5	-1/9-16*w_4_17	3 4 1	0 1 0 5 2 3 4 5	-1/6-16*w_4_30
3 4 1	0 2 1 3 1 4 4 5	32*w_4_18	3 4 1	0 1 0 5 2 4 3 5	-1/6+16*w_4_43
3 4 1	0 4 1 2 1 3 3 5	1/6+16*w_4_18	3 4 1	0 1 0 5 2 6 3 5	-1/6-16*w_4_45
3 4 1	0 4 1 5 1 2 3 4	-1/9-16*w_4_18	3 4 1	0 2 0 3 1 3 3 4	-16*w_4_24+16*w_4_10
3 4 1	0 2 1 3 1 6 3 4	32*w_4_21	3 4 1	0 2 0 3 1 3 3 5	16*w_4_24+16*w_4_10
3 4 1	0 4 1 2 1 6 3 4	-16*w_4_21	3 4 1	0 4 1 2 0 4 3 4	-1/3-16*w_4_10
3 4 1	0 4 5 6 1 2 1 5	1/6+16*w_4_21	3 4 1	0 2 0 3 1 4 3 4	-16*w_4_28+16*w_4_16
3 4 1	0 2 1 3 1 6 3 5	32*w_4_22	3 4 1	0 2 0 5 1 3 3 5	16*w_4_28+16*w_4_16
3 4 1	0 4 1 2 1 6 4 5	16*w_4_22	3 4 1	0 4 0 5 1 2 4 5	-16*w_4_16
3 4 1	0 4 3 5 1 2 1 5	-1/18-16*w_4_22	3 4 1	0 2 0 3 1 4 3 5	-16*w_4_29+16*w_4_17
3 4 1	0 2 1 3 1 6 4 5	32*w_4_23	3 4 1	0 2 0 5 1 3 3 4	16*w_4_29+16*w_4_17
3 4 1	0 4 1 2 1 6 3 5	16*w_4_23	3 4 1	0 4 0 5 1 2 3 5	-1/6-16*w_4_17
3 4 1	0 4 3 5 1 6 1 2	-1/18-16*w_4_23	3 4 1	0 2 0 3 1 4 4 5	16*w_4_18-16*w_4_30
3 4 1	0 2 1 5 1 4 3 4	32*w_4_40	3 4 1	0 2 0 5 1 3 4 5	-16*w_4_18-16*w_4_30
3 4 1	0 2 1 5 1 6 3 4	32*w_4_41	3 4 1	0 4 0 5 1 2 3 4	-1/6-16*w_4_18
3 4 1	0 4 1 5 3 6 1 2	-16*w_4_41	3 4 1	0 2 0 3 1 6 3 4	16*w_4_21-16*w_4_31
3 4 1	0 4 5 6 1 2 1 3	16*w_4_41	3 4 1	0 2 0 5 3 6 1 3	16*w_4_21+16*w_4_31
3 4 1	0 2 1 5 1 6 3 5	32*w_4_42	3 4 1	0 4 1 2 0 6 3 4	-1/6-16*w_4_21
3 4 1	0 4 1 5 4 6 1 2	-16*w_4_42	3 4 1	0 2 0 3 1 6 3 5	16*w_4_22-16*w_4_32
3 4 1	0 4 3 5 1 2 1 3	1/6-16*w_4_42	3 4 1	0 2 0 5 3 4 1 3	16*w_4_22+16*w_4_32
			3 4 1	0 4 1 2 0 6 4 5	1/6+16*w_4_22

TABLE 8 (part 4).

3 4 1	0 2 0 3 1 6 4 5	16*w_4_23-16*w_4_33	3 4 1	0 2 1 5 2 3 3 5	-16*w_4_28+16*w_4_16
3 4 1	0 2 0 5 4 6 1 3	-16*w_4_23-16*w_4_33	3 4 1	0 4 2 5 1 2 4 5	-1/6+16*w_4_28
3 4 1	0 4 1 2 0 6 3 5	16*w_4_23	3 4 1	0 2 1 3 2 4 3 5	16*w_4_29+16*w_4_17
3 4 1	0 4 0 3 1 2 3 5	-16*w_4_40	3 4 1	0 4 1 2 2 3 4 5	16*w_4_29
3 4 1	0 2 0 5 1 6 3 4	-16*w_4_44+16*w_4_41	3 4 1	0 2 1 5 2 3 3 4	-16*w_4_29+16*w_4_17
3 4 1	0 2 0 5 3 6 1 4	16*w_4_44+16*w_4_41	3 4 1	0 4 2 5 1 2 3 5	1/9+16*w_4_29
3 4 1	0 4 0 5 3 6 1 2	-16*w_4_41	3 4 1	0 2 1 3 2 4 4 5	16*w_4_18+16*w_4_30
3 4 1	0 2 0 5 1 6 3 5	-16*w_4_45+16*w_4_42	3 4 1	0 4 1 2 2 3 3 5	1/6+16*w_4_30
3 4 1	0 2 0 5 3 4 1 4	16*w_4_45+16*w_4_42	3 4 1	0 2 1 5 2 3 4 5	-16*w_4_18+16*w_4_30
3 4 1	0 4 0 5 4 6 1 2	-16*w_4_42	3 4 1	0 4 2 5 1 2 3 4	1/9+16*w_4_30
3 4 1	0 2 0 5 3 4 1 5	-16*w_4_47+16*w_4_46	3 4 1	0 2 1 3 2 6 3 4	16*w_4_21+16*w_4_31
3 4 1	0 2 0 5 3 6 1 5	16*w_4_47+16*w_4_46	3 4 1	0 4 1 2 2 6 3 4	-16*w_4_31
3 4 1	0 4 3 5 1 2 0 4	-1/6-16*w_4_46	3 4 1	0 2 1 5 3 6 2 3	16*w_4_21-16*w_4_31
3 4 1	0 4 0 3 1 3 2 3	16*w_4_60-16*w_4_64	3 4 1	0 4 5 6 1 2 2 5	1/6-16*w_4_31
3 4 1	0 4 0 5 1 4 2 4	16*w_4_60-16*w_4_86	3 4 1	0 2 1 3 2 6 3 5	16*w_4_22+16*w_4_32
3 4 1	0 4 0 3 1 3 2 4	16*w_4_61-16*w_4_65	3 4 1	0 4 1 2 2 6 4 5	16*w_4_32
3 4 1	0 4 0 5 1 4 2 5	16*w_4_61-16*w_4_87	3 4 1	0 2 1 5 3 4 2 3	16*w_4_22-16*w_4_32
3 4 1	0 4 0 3 1 3 2 5	-16*w_4_66+16*w_4_62	3 4 1	0 4 3 5 1 2 2 5	1/18+16*w_4_32
3 4 1	0 4 0 5 1 4 2 3	16*w_4_62-16*w_4_85	3 4 1	0 2 1 3 2 6 4 5	16*w_4_23+16*w_4_33
3 4 1	0 4 0 3 1 6 2 3	-16*w_4_66+16*w_4_63	3 4 1	0 4 1 2 2 6 3 5	16*w_4_33
3 4 1	0 4 0 5 1 3 2 3	-16*w_4_77+16*w_4_71	3 4 1	0 2 1 5 4 6 2 3	-16*w_4_23+16*w_4_33
3 4 1	0 4 0 5 1 3 2 4	16*w_4_71-16*w_4_78	3 4 1	0 4 3 5 2 6 1 2	1/18+16*w_4_33
3 4 1	0 4 0 5 1 3 2 5	-16*w_4_79+16*w_4_71	3 4 1	0 2 1 5 2 4 3 4	1/18+16*w_4_40+16*w_4_43
3 4 1	0 4 0 5 1 6 2 3	-16*w_4_92+16*w_4_72	3 4 1	0 4 2 3 1 2 3 5	1/6-16*w_4_43
3 4 1	0 4 1 5 0 6 2 3	-16*w_4_118+16*w_4_72	3 4 1	0 2 1 5 2 4 3 5	1/18+16*w_4_40-16*w_4_43
3 4 1	0 4 0 5 2 6 1 3	-16*w_4_92+16*w_4_72	3 4 1	0 4 2 3 1 2 4 5	16*w_4_43
3 4 1	0 4 0 5 1 6 2 4	-16*w_4_79+16*w_4_73	3 4 1	0 2 1 5 2 6 3 4	16*w_4_44+16*w_4_41
3 4 1	0 4 1 5 0 6 2 4	16*w_4_73-16*w_4_78	3 4 1	0 4 2 5 3 6 1 2	-16*w_4_44
3 4 1	0 4 0 5 2 3 1 3	16*w_4_73-16*w_4_77	3 4 1	0 2 1 5 3 6 2 4	-16*w_4_44+16*w_4_41
3 4 1	0 4 0 5 1 6 2 5	1/18-16*w_4_66+16*w_4_74	3 4 1	0 4 5 6 1 2 2 3	-16*w_4_44
3 4 1	0 4 1 5 0 6 2 5	16*w_4_74-16*w_4_107	3 4 1	0 2 1 5 2 6 3 5	16*w_4_45+16*w_4_42
3 4 1	0 4 0 5 2 4 1 3	16*w_4_74-16*w_4_85	3 4 1	0 4 2 5 4 6 1 2	-16*w_4_45
3 4 1	0 4 0 5 2 3 1 4	-16*w_4_78+16*w_4_75	3 4 1	0 2 1 5 3 4 2 4	-16*w_4_45+16*w_4_42
3 4 1	0 4 1 5 2 3 0 4	-16*w_4_77+16*w_4_75	3 4 1	0 4 3 5 1 2 2 3	1/6+16*w_4_45
3 4 1	0 4 0 5 2 6 1 4	-16*w_4_79+16*w_4_75	3 4 1	0 2 1 5 3 4 2 5	16*w_4_47+16*w_4_46
3 4 1	0 4 0 5 2 3 1 5	-16*w_4_79+16*w_4_76	3 4 1	0 4 3 5 1 2 2 4	-16*w_4_47
3 4 1	0 4 1 5 2 3 0 5	16*w_4_76-16*w_4_78	3 4 1	0 2 1 5 3 6 2 5	-16*w_4_47+16*w_4_46
3 4 1	0 4 2 5 1 3 0 4	16*w_4_76-16*w_4_77	3 4 1	0 4 5 6 1 2 2 4	-16*w_4_47
3 4 1	0 4 0 5 2 4 1 4	-16*w_4_86+16*w_4_83	3 4 1	0 4 2 5 2 4 1 4	-16*w_4_60+16*w_4_64
3 4 1	0 4 1 5 2 4 0 4	1/12+8*w_4_83-8*w_4_64	3 4 1	0 4 2 5 2 4 1 5	-16*w_4_61+16*w_4_65
3 4 1	0 4 0 5 2 4 1 5	-16*w_4_87+16*w_4_84	3 4 1	0 4 1 5 2 6 2 5	16*w_4_66-16*w_4_62
3 4 1	0 4 1 5 2 4 0 5	1/6+16*w_4_84-16*w_4_65	3 4 1	0 4 2 5 2 4 1 3	16*w_4_66-16*w_4_63
3 4 1	0 4 2 3 0 4 1 3	-16*w_4_106+16*w_4_84	3 4 1	0 4 1 5 2 3 2 3	-16*w_4_76+16*w_4_77
3 4 1	0 4 0 5 2 6 1 5	1/18-16*w_4_66+16*w_4_91	3 4 1	0 4 2 5 1 3 2 5	16*w_4_77-16*w_4_75
3 4 1	0 4 1 5 2 6 0 5	16*w_4_91-16*w_4_85	3 4 1	0 4 2 5 2 3 1 4	-16*w_4_73+16*w_4_77
3 4 1	0 4 2 3 0 4 1 5	-16*w_4_107+16*w_4_91	3 4 1	0 4 2 5 2 6 1 4	16*w_4_77-16*w_4_71
3 4 1	0 4 2 3 0 4 1 4	16*w_4_105-16*w_4_86	3 4 1	0 4 1 5 2 3 2 4	-16*w_4_73+16*w_4_78
3 4 1	0 4 2 5 1 4 0 4	1/12+8*w_4_105-8*w_4_64	3 4 1	0 4 2 5 1 3 2 3	-16*w_4_76+16*w_4_78
3 4 1	0 1 0 2 3 4 3 5	-16*w_4_4	3 4 1	0 4 2 5 1 6 2 4	-16*w_4_71+16*w_4_78
3 4 1	0 1 0 2 3 6 4 5	-16*w_4_5	3 4 1	0 4 2 5 2 3 1 5	16*w_4_78-16*w_4_75
3 4 1	0 2 0 3 1 3 4 5	-16*w_4_25	3 4 1	0 4 1 5 2 3 2 5	16*w_4_79-16*w_4_75
3 4 1	0 1 0 5 2 3 3 4	16*w_4_29	3 4 1	0 4 2 5 1 3 2 4	16*w_4_79-16*w_4_73
3 4 1	0 1 0 5 3 6 2 3	16*w_4_31	3 4 1	0 4 1 5 2 6 2 4	16*w_4_79-16*w_4_71
3 4 1	0 1 0 5 3 4 2 3	16*w_4_32	3 4 1	0 4 2 5 2 3 1 3	16*w_4_79-16*w_4_76
3 4 1	0 1 0 5 4 6 2 3	-16*w_4_33	3 4 1	0 4 1 5 2 4 2 3	-1/18-16*w_4_63+16*w_4_85
3 4 1	0 1 0 5 2 4 3 4	-16*w_4_43	3 4 1	0 4 2 3 1 3 2 5	-16*w_4_91+16*w_4_85
3 4 1	0 1 0 5 2 6 3 4	-16*w_4_44	3 4 1	0 4 2 3 1 6 2 4	-16*w_4_74+16*w_4_85
3 4 1	0 1 0 5 3 6 2 4	16*w_4_44	3 4 1	0 4 2 5 2 6 1 5	-16*w_4_62+16*w_4_85
3 4 1	0 1 0 5 3 4 2 4	16*w_4_45	3 4 1	0 4 1 5 2 4 2 4	-1/6-16*w_4_60+16*w_4_86
3 4 1	0 1 0 5 3 4 2 5	-16*w_4_47	3 4 1	0 4 2 3 1 3 2 3	-16*w_4_105+16*w_4_86
3 4 1	0 1 0 5 3 6 2 5	16*w_4_47	3 4 1	0 4 2 3 1 4 2 4	16*w_4_86-16*w_4_83
# 1 1 2			3 4 1	0 4 2 5 1 4 2 4	-16*w_4_60+16*w_4_86
3 4 1	0 4 1 3 2 3 2 3	-1/12-8*w_4_105+8*w_4_64	3 4 1	0 4 1 5 2 4 2 5	-16*w_4_61+16*w_4_87
3 4 1	0 4 1 3 2 3 2 4	-1/6-16*w_4_84+16*w_4_65	3 4 1	0 4 2 3 1 3 2 4	16*w_4_87-16*w_4_84
3 4 1	0 4 1 3 2 4 2 4	-1/12-8*w_4_83+8*w_4_64	3 4 1	0 4 1 5 2 6 2 3	16*w_4_92-16*w_4_72
3 4 1	0 4 1 3 2 3 2 5	-1/18+16*w_4_66-16*w_4_91	3 4 1	0 4 2 5 2 6 1 3	16*w_4_92-16*w_4_72
3 4 1	0 4 1 3 2 4 2 5	-1/18+16*w_4_66-16*w_4_74	3 4 1	0 4 2 3 1 4 2 3	16*w_4_106-16*w_4_84
3 4 1	0 1 2 3 2 3 3 4	1/3+16*w_4_10	3 4 1	0 4 2 5 1 4 2 5	-1/6+16*w_4_106-16*w_4_61
3 4 1	0 1 2 3 2 6 3 5	1/6+16*w_4_22	3 4 1	0 4 2 3 1 4 2 5	-16*w_4_74+16*w_4_107
3 4 1	0 1 2 3 2 4 4 5	1/6+16*w_4_18	3 4 1	0 4 2 5 1 4 2 3	-1/18-16*w_4_63+16*w_4_107
3 4 1	0 1 2 3 2 4 3 5	1/6+16*w_4_17	3 4 1	0 4 2 3 1 6 2 3	16*w_4_107-16*w_4_91
3 4 1	0 1 2 3 2 6 3 4	1/6+16*w_4_21	3 4 1	0 4 2 5 1 6 2 5	-1/6+16*w_4_107-16*w_4_62
3 4 1	0 1 2 5 3 4 2 5	1/6+16*w_4_46	3 4 1	0 4 2 5 1 6 2 3	16*w_4_118-16*w_4_72
3 4 1	0 2 1 2 3 4 3 4	-1/6+8*w_4_3	3 4 1	0 1 2 3 2 4 3 4	16*w_4_16
3 4 1	0 2 1 2 3 4 3 5	-1/6+16*w_4_4	3 4 1	0 1 2 3 2 6 4 5	16*w_4_23
3 4 1	0 2 1 2 3 4 4 5	-16*w_4_4	3 4 1	0 1 2 5 2 4 3 4	16*w_4_40
3 4 1	0 2 1 2 3 6 4 5	16*w_4_5	3 4 1	0 1 2 5 2 6 3 4	16*w_4_41
3 4 1	0 2 1 3 2 3 3 4	16*w_4_24+16*w_4_10	3 4 1	0 1 2 5 2 6 3 5	16*w_4_42
3 4 1	0 4 1 2 2 4 3 4	1/9-16*w_4_24	# 1 1 1		
3 4 1	0 2 1 3 2 3 3 5	-16*w_4_24+16*w_4_10	3 4 1	0 4 1 3 2 3 3 4	1/6+16*w_4_108+16*w_4_67
3 4 1	0 4 1 2 2 4 4 5	1/6-16*w_4_24	3 4 1	0 4 1 3 2 4 3 4	1/6-16*w_4_67+16*w_4_90
3 4 1	0 2 1 3 2 3 4 5	16*w_4_25	3 4 1	0 4 1 3 2 6 3 4	1/6+16*w_4_111
3 4 1	0 4 1 2 2 4 3 5	1/9+16*w_4_25	3 4 1	0 4 1 3 2 3 3 5	1/18-16*w_4_110+16*w_4_68
3 4 1	0 2 1 3 2 4 3 4	16*w_4_28+16*w_4_16	3 4 1	0 4 1 3 2 3 4 5	1/18-16*w_4_109+16*w_4_69
3 4 1	0 4 1 2 2 3 3 4	1/3-16*w_4_28	3 4 1	0 4 1 3 2 4 3 5	1/18+16*w_4_89+16*w_4_69
			3 4 1	0 4 1 3 2 4 4 5	1/18+16*w_4_68+16*w_4_88

TABLE 8 (part 5).

3 4 1	0 4 1 3 2 6 3 5	1/36+16*w_4_70-16*w_4_113	3 4 1	0 4 3 5 1 6 2 3	16*w_4_94-16*w_4_120
3 4 1	0 4 1 3 2 6 4 5	1/36-16*w_4_112+16*w_4_70	3 4 1	0 4 2 5 1 6 4 5	16*w_4_121-16*w_4_96
3 4 1	0 1 2 3 3 4 3 4	-1/6+8*w_4_34	3 4 1	0 4 3 5 2 3 1 5	16*w_4_95-16*w_4_121
3 4 1	0 1 2 3 3 4 4 5	1/6+16*w_4_36	3 4 1	0 4 2 5 3 4 1 4	16*w_4_122+16*w_4_103
3 4 1	0 1 2 5 3 4 3 4	-1/6+16*w_4_48	3 4 1	0 4 2 5 4 6 1 4	-16*w_4_122+16*w_4_97
3 4 1	0 1 2 5 3 4 4 5	1/6+16*w_4_50	3 4 1	0 4 2 5 3 4 1 5	16*w_4_123-16*w_4_124
3 4 1	0 2 1 3 3 4 3 4	16*w_4_34	3 4 1	0 4 5 6 1 6 2 4	16*w_4_123-16*w_4_98
3 4 1	0 4 1 2 3 4 3 4	-1/6+8*w_4_34	3 4 1	0 4 3 5 2 3 1 4	16*w_4_123-16*w_4_124
3 4 1	0 2 1 3 3 4 3 5	32*w_4_35	3 4 1	0 4 5 6 1 4 2 5	16*w_4_98-16*w_4_124
3 4 1	0 4 1 2 3 4 4 5	-16*w_4_35	3 4 1	0 4 2 5 3 6 1 5	16*w_4_128-16*w_4_129
3 4 1	0 2 1 3 3 4 4 5	32*w_4_36	3 4 1	0 4 3 5 1 6 2 4	-16*w_4_128+16*w_4_101
3 4 1	0 4 1 2 3 4 3 5	-1/6-16*w_4_36	3 4 1	0 4 3 5 2 6 1 4	-16*w_4_129+16*w_4_101
3 4 1	0 2 1 3 3 6 3 5	16*w_4_37	3 4 1	0 4 5 6 1 4 2 3	16*w_4_128-16*w_4_129
3 4 1	0 4 1 2 4 6 4 5	8*w_4_37	3 4 1	0 4 2 5 4 6 1 5	-16*w_4_134+16*w_4_133
3 4 1	0 2 1 3 3 6 4 5	32*w_4_38	3 4 1	0 4 3 5 2 4 1 5	-16*w_4_133+16*w_4_104
3 4 1	0 4 1 2 3 6 4 5	16*w_4_38	3 4 1	0 4 3 5 1 4 2 3	16*w_4_134-16*w_4_133
3 4 1	0 2 1 3 4 6 4 5	16*w_4_39	3 4 1	0 4 3 5 1 4 2 5	-16*w_4_134+16*w_4_104
3 4 1	0 4 1 2 3 6 3 5	8*w_4_39	3 4 1	0 4 3 5 2 4 1 4	32*w_4_138
3 4 1	0 2 1 5 3 4 3 4	32*w_4_48	3 4 1	0 4 5 6 1 4 2 4	16*w_4_138
3 4 1	0 4 3 5 1 2 3 5	-1/6+16*w_4_48	3 4 1	0 1 2 3 3 4 3 5	16*w_4_35
3 4 1	0 2 1 5 3 4 3 5	32*w_4_49	3 4 1	0 1 2 3 3 6 3 5	8*w_4_37
3 4 1	0 4 3 5 1 2 4 5	16*w_4_49	3 4 1	0 1 2 3 3 6 4 5	16*w_4_38
3 4 1	0 2 1 5 3 4 4 5	32*w_4_50	3 4 1	0 1 2 3 4 6 4 5	8*w_4_39
3 4 1	0 4 3 5 1 2 3 4	-1/6-16*w_4_50	3 4 1	0 1 2 5 3 4 3 5	16*w_4_49
3 4 1	0 2 1 5 3 6 3 4	32*w_4_51	3 4 1	0 1 2 5 3 6 3 4	16*w_4_51
3 4 1	0 4 5 6 1 2 3 5	-16*w_4_51	3 4 1	0 1 2 5 3 6 3 5	16*w_4_52
3 4 1	0 2 1 5 3 6 3 5	32*w_4_52	3 4 1	0 1 2 5 3 6 4 5	16*w_4_53
3 4 1	0 4 5 6 1 2 4 5	-16*w_4_52	3 4 1	0 1 2 5 4 6 3 4	16*w_4_54
3 4 1	0 2 1 5 3 6 4 5	32*w_4_53	3 4 1	0 1 2 5 4 6 3 5	16*w_4_55
3 4 1	0 4 5 6 1 2 3 4	16*w_4_53	3 4 1	0 4 2 5 3 6 1 4	16*w_4_100
3 4 1	0 2 1 5 4 6 3 4	32*w_4_54	3 4 1	0 4 3 5 1 4 2 4	16*w_4_138
3 4 1	0 4 3 5 3 6 1 2	-16*w_4_54			
3 4 1	0 2 1 5 4 6 3 5	32*w_4_55			
3 4 1	0 4 3 5 4 6 1 2	-16*w_4_55			
3 4 1	0 4 1 5 2 3 3 4	16*w_4_80+16*w_4_81			
3 4 1	0 4 2 5 1 3 3 5	-16*w_4_80+16*w_4_82			
3 4 1	0 4 1 5 2 3 3 5	16*w_4_81+16*w_4_82			
3 4 1	0 4 2 5 1 3 4 5	-16*w_4_80-16*w_4_81			
3 4 1	0 4 1 5 2 3 4 5	-16*w_4_80+16*w_4_82			
3 4 1	0 4 2 5 1 3 3 4	16*w_4_81+16*w_4_82			
3 4 1	0 4 1 5 2 4 3 4	1/18+16*w_4_68+16*w_4_88			
3 4 1	0 4 2 3 1 3 3 5	-16*w_4_110-16*w_4_88			
3 4 1	0 4 1 5 2 4 3 5	1/18+16*w_4_89+16*w_4_69			
3 4 1	0 4 2 3 1 3 4 5	-16*w_4_89-16*w_4_109			
3 4 1	0 4 1 5 2 4 4 5	1/6-16*w_4_67+16*w_4_90			
3 4 1	0 4 2 3 1 3 3 4	16*w_4_108+16*w_4_90			
3 4 1	0 4 1 5 2 6 3 4	16*w_4_99+16*w_4_93			
3 4 1	0 4 2 5 3 6 1 3	-16*w_4_93+16*w_4_119			
3 4 1	0 4 1 5 2 6 3 5	16*w_4_102+16*w_4_94			
3 4 1	0 4 2 5 4 6 1 3	-16*w_4_94+16*w_4_120			
3 4 1	0 4 1 5 2 6 4 5	16*w_4_95-16*w_4_96			
3 4 1	0 4 2 5 3 4 1 3	16*w_4_95-16*w_4_121			
3 4 1	0 4 1 5 3 4 2 3	-16*w_4_121+16*w_4_96			
3 4 1	0 4 3 5 1 3 2 5	16*w_4_95-16*w_4_96			
3 4 1	0 4 1 5 3 4 2 4	16*w_4_103+16*w_4_97			
3 4 1	0 4 3 5 1 3 2 3	16*w_4_122-16*w_4_97			
3 4 1	0 4 1 5 3 4 2 5	16*w_4_98-16*w_4_124			
3 4 1	0 4 3 5 1 3 2 4	16*w_4_123-16*w_4_98			
3 4 1	0 4 1 5 3 6 2 3	16*w_4_99+16*w_4_119			
3 4 1	0 4 5 6 1 3 2 5	16*w_4_99+16*w_4_93			
3 4 1	0 4 1 5 3 6 2 4	32*w_4_100			
3 4 1	0 4 5 6 1 3 2 3	16*w_4_100			
3 4 1	0 4 1 5 3 6 2 5	-16*w_4_129+16*w_4_101			
3 4 1	0 4 5 6 1 3 2 4	-16*w_4_128+16*w_4_101			
3 4 1	0 4 1 5 4 6 2 3	16*w_4_102+16*w_4_120			
3 4 1	0 4 3 5 2 6 1 3	16*w_4_102+16*w_4_94			
3 4 1	0 4 1 5 4 6 2 4	16*w_4_103+16*w_4_97			
3 4 1	0 4 3 5 2 3 1 3	16*w_4_122+16*w_4_103			
3 4 1	0 4 1 5 4 6 2 5	-16*w_4_134+16*w_4_104			
3 4 1	0 4 3 5 2 4 1 3	-16*w_4_133+16*w_4_104			
3 4 1	0 4 2 3 1 4 3 4	16*w_4_108+16*w_4_90			
3 4 1	0 4 2 5 1 4 4 5	-1/6-16*w_4_108-16*w_4_67			
3 4 1	0 4 2 3 1 4 3 5	16*w_4_89+16*w_4_109			
3 4 1	0 4 2 5 1 4 3 5	1/18-16*w_4_109+16*w_4_69			
3 4 1	0 4 2 3 1 4 4 5	16*w_4_110+16*w_4_88			
3 4 1	0 4 2 5 1 4 3 4	1/18-16*w_4_110+16*w_4_68			
3 4 1	0 4 2 3 1 6 3 4	32*w_4_111			
3 4 1	0 4 5 6 1 6 2 5	1/6+16*w_4_111			
3 4 1	0 4 2 3 1 6 3 5	16*w_4_112-16*w_4_113			
3 4 1	0 4 3 5 1 6 2 5	1/36-16*w_4_112+16*w_4_70			
3 4 1	0 4 2 3 1 6 4 5	-16*w_4_112+16*w_4_113			
3 4 1	0 4 3 5 2 6 1 5	1/36+16*w_4_70-16*w_4_113			
3 4 1	0 4 2 5 1 6 3 4	16*w_4_99+16*w_4_119			
3 4 1	0 4 5 6 1 6 2 3	-16*w_4_93+16*w_4_119			
3 4 1	0 4 2 5 1 6 3 5	16*w_4_102+16*w_4_120			

TABLE 9. Sample output of `reduce_mod_jacobi`.

3 4 1	0 1 0 3 2 6 3 4	-24+c_1_1221_211	3 4 1	0 4 1 5 3 6 2 3	-8+c_1_1023_111
3 4 1	0 4 1 3 2 6 3 4	-8+c_1_1240_111	3 4 1	0 4 5 6 1 3 2 5	-16+c_1_540_111
3 4 1	0 1 0 3 2 3 4 5	-48+c_1_1221_211	3 4 1	0 4 1 5 3 6 2 4	32-c_1_1242_111
		-c_1_513_211			-c_1_540_111
3 4 1	0 1 0 3 2 4 3 5	24-c_1_1221_211	3 4 1	0 4 5 6 1 3 2 3	16-c_1_1023_111
3 4 1	0 4 1 3 2 4 3 5	24-c_1_1240_111			+c_1_1245_111
		-c_1_540_111	3 4 1	0 4 1 5 4 6 2 3	-16+c_1_1242_111
3 4 1	0 1 2 3 3 4 4 5	-8-c_1_1228_111	3 4 1	0 4 3 5 2 6 1 3	-8-c_1_1245_111
3 4 1	0 1 2 5 3 4 3 4	-8-c_1_1228_111	3 4 1	0 4 2 3 1 4 3 5	24+c_1_538_111
3 4 1	0 2 0 3 1 4 3 5	24-c_1_1005_211			-c_1_1019_111
3 4 1	0 2 0 5 1 3 3 4	-24-c_1_516_211	3 4 1	0 4 2 3 1 6 3 4	-16+c_1_1019_111
3 4 1	0 2 0 3 1 6 3 4	-24+c_1_1005_211	3 4 1	0 4 5 6 1 6 2 5	-8+c_1_536_111
3 4 1	0 2 0 5 3 6 1 3	24+c_1_516_211	3 4 1	0 4 2 5 1 6 3 4	-8+c_1_1021_111
3 4 1	0 2 1 3 2 3 4 5	48+c_1_1230_112	3 4 1	0 4 5 6 1 6 2 3	8+c_1_538_111
		-c_1_1008_112	3 4 1	0 4 2 5 1 6 3 5	-16+c_1_540_111
3 4 1	0 4 1 2 2 4 3 5	48-c_1_525_112	3 4 1	0 4 3 5 1 6 2 3	8-c_1_1023_111
		-c_1_1239_112	3 4 1	0 4 2 5 3 4 1 5	-8+c_1_1021_111
3 4 1	0 2 1 3 2 4 3 5	-24-c_1_1230_112	3 4 1	0 4 5 6 1 6 2 4	8+c_1_538_111
3 4 1	0 4 1 2 2 3 4 5	-24+c_1_1239_112	3 4 1	0 4 3 5 2 3 1 4	-8+c_1_1023_111
3 4 1	0 2 1 5 2 3 3 4	24-c_1_1008_112	3 4 1	0 4 5 6 1 4 2 5	-16+c_1_540_111
3 4 1	0 4 2 5 1 2 3 5	-24+c_1_525_112	3 4 1	0 2 0 3 1 3 4 5	-48+c_1_1005_211-c_1_516_211
3 4 1	0 2 1 3 2 6 3 4	24+c_1_1230_112	3 4 1	0 1 0 5 2 3 3 4	-24-c_1_513_211
3 4 1	0 4 1 2 2 6 3 4	-24+c_1_1239_112	3 4 1	0 1 0 5 3 6 2 3	24+c_1_513_211
3 4 1	0 2 1 5 3 6 2 3	-24+c_1_1008_112	3 4 1	0 1 2 3 3 6 4 5	-8-c_1_1228_111
3 4 1	0 4 5 6 1 2 2 5	-24+c_1_525_112	3 4 1	0 1 2 5 3 6 3 5	8+c_1_1228_111
3 4 1	0 2 1 3 3 4 4 5	-16-c_1_1012_111	3 4 1	0 4 2 5 3 6 1 4	16+c_1_538_111-c_1_1021_111
3 4 1	0 4 1 2 3 4 3 5	8+c_1_529_111	3 4 1	0 4 1 3 2 3 4 5	-c_1_1023_111+c_1_1240_111
3 4 1	0 2 1 3 3 6 4 5	-16-c_1_1012_111	3 4 1	0 4 2 5 1 4 3 5	c_1_536_111-c_1_1021_111
3 4 1	0 4 1 2 3 6 4 5	-8-c_1_529_111			
3 4 1	0 2 1 5 3 4 3 4	-16-c_1_1012_111	3 4 1	0 1 2 3 0 3 5 4	c_1_513_211===24
3 4 1	0 4 3 5 1 2 3 5	-8-c_1_529_111	3 4 1	0 2 1 3 0 3 5 4	c_1_516_211===24
3 4 1	0 2 1 5 3 6 3 5	16+c_1_1012_111	3 4 1	1 2 2 3 0 3 5 4	c_1_525_112==24
3 4 1	0 4 5 6 1 2 4 5	-8-c_1_529_111	3 4 1	1 2 3 5 0 3 5 4	c_1_529_111===8
3 4 1	0 4 1 5 2 3 3 4	8-c_1_1023_111	3 4 1	1 4 2 3 0 3 5 4	c_1_536_111==8
3 4 1	0 4 2 5 1 3 3 5	-16+c_1_540_111	3 4 1	1 4 2 5 0 3 5 4	c_1_538_111===8
3 4 1	0 4 1 5 2 3 3 5	-8-c_1_538_111	3 4 1	1 5 2 3 0 3 5 4	c_1_540_111==16
3 4 1	0 4 2 5 1 3 4 5	-8+c_1_1021_111	3 4 1	0 2 0 3 1 3 5 4	c_1_1005_211==24
3 4 1	0 4 1 5 2 3 4 5	-16+c_1_1242_111	3 4 1	0 2 2 3 1 3 5 4	c_1_1008_112==24
3 4 1	0 4 2 5 1 3 3 4	-8-c_1_1245_111	3 4 1	0 2 3 5 1 3 5 4	c_1_1012_111===16
3 4 1	0 4 1 5 2 4 3 5	24-c_1_536_111	3 4 1	0 4 2 3 1 3 5 4	c_1_1019_111==16
		-c_1_1242_111	3 4 1	0 4 2 5 1 3 5 4	c_1_1021_111==8
3 4 1	0 4 2 3 1 3 4 5	-24-c_1_1245_111	3 4 1	0 5 2 3 1 3 5 4	c_1_1023_111==8
		+c_1_1019_111	3 4 1	0 1 0 3 2 3 5 4	c_1_1221_211==24
3 4 1	0 4 1 5 2 6 3 4	-16+c_1_1242_111	3 4 1	0 1 3 5 2 3 5 4	c_1_1228_111==8
3 4 1	0 4 2 5 3 6 1 3	8+c_1_1245_111	3 4 1	0 2 1 3 2 3 5 4	c_1_1230_112==24
3 4 1	0 4 1 5 2 6 3 5	-8-c_1_538_111	3 4 1	0 4 1 2 2 3 5 4	c_1_1239_112==24
3 4 1	0 4 2 5 4 6 1 3	-8+c_1_1021_111	3 4 1	0 4 1 3 2 3 5 4	c_1_1240_111==8
3 4 1	0 4 1 5 3 4 2 5	-16+c_1_1242_111	3 4 1	0 4 1 5 2 3 5 4	c_1_1242_111==16
3 4 1	0 4 3 5 1 3 2 4	8+c_1_1245_111	3 4 1	0 5 1 3 2 3 5 4	c_1_1245_111==8



APPENDIX E. GAUGE TRANSFORMATION THAT REMOVES  
4 MASTER-PARAMETERS OUT OF 10

Encodings of graphs (see Implementation 1 on p. 229) built over one sink vertex are followed by their coefficients, in the following table containing the gauge transformation which was claimed to exist in Theorem 14.

TABLE 10. Gauge transformation that removes 4 master-parameters out of 10.

h <sup>0</sup> :		
1 0 1		1
h <sup>4</sup> :		
1 4 1	0 2 0 3 1 4 0 3	16*w_4_101
1 4 1	0 2 0 3 1 4 1 3	8*w_4_101
1 4 1	0 2 0 3 1 4 2 3	8*w_4_101
1 4 1	0 2 1 3 0 4 1 2	-8*w_4_101
1 4 1	0 2 1 3 0 4 2 3	8*w_4_101
1 4 1	0 2 1 3 1 4 0 2	-8*w_4_101
1 4 1	0 2 1 3 2 4 0 2	-8*w_4_101
1 4 1	0 2 0 3 0 4 1 3	-16*w_4_102
1 4 1	0 2 0 3 1 4 1 3	-8*w_4_102
1 4 1	0 2 0 3 2 4 1 2	-8*w_4_102
1 4 1	0 2 0 3 2 4 1 3	-16*w_4_102
1 4 1	0 2 1 3 0 4 1 2	-8*w_4_102
1 4 1	0 2 1 3 0 4 1 3	-8*w_4_102
1 4 1	0 2 0 3 0 4 1 2	16*w_4_119
1 4 1	0 2 0 3 1 4 1 2	16*w_4_119
1 4 1	0 2 0 3 1 4 1 3	8*w_4_119
1 4 1	0 2 0 3 2 4 1 2	8*w_4_119
1 4 1	0 2 1 3 0 4 1 2	8*w_4_119
1 4 1	0 2 3 4 0 4 1 2	-8*w_4_119
1 4 1	0 2 0 3 0 1 1 2	-32*w_4_125
1 4 1	0 2 0 3 1 2 1 2	16*w_4_125
1 4 1	0 2 0 3 1 2 1 3	-16*w_4_125
1 4 1	0 2 0 3 1 2 2 3	16*w_4_125
1 4 1	0 2 0 3 1 4 1 2	16*w_4_125
1 4 1	0 2 0 3 1 4 1 3	-16*w_4_125
1 4 1	0 2 0 3 1 4 2 3	16*w_4_125



## Chapter 12

# Formality morphism as the mechanism of $\star$ -product associativity: how it works

This chapter is based on the peer-reviewed journal publication *R. Buring and A.V. Kiselev, Collected works Inst. Math., Kyiv* **16**:1, 22–43, 2019. (Preprint [arXiv:1907.00639](https://arxiv.org/abs/1907.00639) [q-alg] – 16 p.) This paper follows the talk given by the dissertant at conference *Symmetry & Integrability of Equations of Mathematical Physics* (December 22–23, IM NASU Kyiv, Ukraine, 2018).

*Commentary.* In reference to Part **I** of the dissertation, the material of this chapter is used in Chapter **2** (§**2.5**) and Chapter **3**. This chapter concludes the exposition about star products from Chapters **10** and **11**.

# FORMALITY MORPHISM AS THE MECHANISM OF ★-PRODUCT ASSOCIATIVITY: HOW IT WORKS

RICARDO BURING<sup>1)</sup> AND ARTHEMY V. KISELEV<sup>2)</sup>

‘Symmetries & integrability of equations of mathematical physics’,  
(22–24 December 2018, IM NASU Kiev, Ukraine)

Abstract. The formality morphism  $\mathcal{F} = \{\mathcal{F}_n, n \geq 1\}$  in Kontsevich’s deformation quantization is a collection of maps from tensor powers of the differential graded Lie algebra (dgLa) of multivector fields to the dgLa of polydifferential operators on finite-dimensional affine manifolds. Not a Lie algebra morphism by its term  $\mathcal{F}_1$  alone, the entire set  $\mathcal{F}$  is an  $L_\infty$ -morphism instead. It induces a map of the Maurer–Cartan elements, taking Poisson bi-vectors to deformations  $\mu_A \mapsto \star_{A[[\hbar]]}$  of the usual multiplication of functions into associative noncommutative  $\star$ -products of power series in  $\hbar$ . The associativity of  $\star$ -products is then realized, in terms of the Kontsevich graphs which encode polydifferential operators, by differential consequences of the Jacobi identity. The aim of this paper is to illustrate the work of this algebraic mechanism for the Kontsevich  $\star$ -products (in particular, with harmonic propagators). We inspect how the Kontsevich weights are correlated for the orgraphs which occur in the associator for  $\star$  and in its expansion using Leibniz graphs with the Jacobi identity at a vertex.

Introduction. The Kontsevich formality morphism  $\mathcal{F}$  relates two differential graded Lie algebras (dgLa). Its domain of definition is the shifted-graded vector space  $T_{\text{poly}}^{\downarrow[1]}(M^r)$  of multivectors on an affine real finite-dimensional manifold  $M^r$ ; the graded Lie algebra structure is the Schouten bracket  $[[, ]]$  and the differential is set to (the bracket with) zero by definition. On the other hand, the target space of the formality morphism  $\mathcal{F}$  is the graded vector space  $D_{\text{poly}}^{\downarrow[1]}(M^r)$  of polydifferential operators on  $M^r$ ; the graded Lie algebra structure is the Gerstenhaber bracket  $[, ]_G$  and the differential  $\mathbf{d}_H = [\mu_A, \cdot]$  is induced by using the multiplication  $\mu_A$  in the algebra  $A := C^\infty(M^r)$  of functions on  $M^r$ . It is readily seen that w.r.t. the above notation, Poisson bi-vectors  $\mathcal{P}$  satisfying the Jacobi identity  $[[\mathcal{P}, \mathcal{P}]] = 0$  on  $M^r$  are the Maurer–Cartan elements (indeed,  $(d \equiv 0)(\mathcal{P}) + \frac{1}{2}[[\mathcal{P}, \mathcal{P}]] = 0$ ). Likewise, for a (non)commutative star-product  $\star = \mu_{A[[\hbar]]} + \langle \text{tail} =: B \rangle$ , which deforms the usual multiplication  $\mu = \mu_{A[[\hbar]]}$  in  $A[[\hbar]] = C^\infty(M^r) \otimes_{\mathbb{R}} \mathbb{R}[[\hbar]]$  by a tail  $B$  w.r.t. a formal parameter  $\hbar$ , the requirement that  $\star$  be associative again is the

---

Date: 1 July 2019.

2010 Mathematics Subject Classification. 05C22, 16E45, 53D55, secondary 53D17, 68R10, 81R60.

<sup>1)</sup> Address: Institut für Mathematik, Johannes Gutenberg–Universität, Staudingerweg 9, D-55128 Mainz, Germany. E-mail (corresponding author): rburing@uni-mainz.de .

<sup>2)</sup> Address: Bernoulli Institute for Mathematics, Computer Science and Artificial Intelligence, University of Groningen, P.O.Box 407, 9700 AK Groningen, The Netherlands. E-mail: A.V.Kiselev@rug.nl .

Maurer–Cartan equation,

$$[\mu, B]_G + \frac{1}{2}[B, B]_G = 0 \iff \frac{1}{2}[\mu + B, \mu + B]_G = 0.$$

Here, the leading order equality  $[\mu, \mu]_G = 0$  expresses the given associativity of the product  $\mu$  itself.

The Kontsevich formality mapping  $\mathcal{F} = \{\mathcal{F}_n: T_{\text{poly}}^{\otimes n} \rightarrow D_{\text{poly}}, n \geq 1\}$  in [14, 15] is an  $L_\infty$ -morphism which induces a map that takes Maurer–Cartan elements  $\mathcal{P}$ , i.e. formal Poisson bi-vectors  $\tilde{\mathcal{P}} = \hbar\mathcal{P} + \bar{o}(\hbar)$  on  $M^r$ , to Maurer–Cartan elements<sup>1</sup>, i.e. the tails  $B$  in solutions  $\star$  of the associativity equation on  $A[[\hbar]]$ .

The theory required to build the Kontsevich map  $\mathcal{F}$  is standard, well reflected in the literature (see [14, 15], as well as [9, 11] and references therein); a proper choice of signs is analysed in [2, 18]. The framework of homotopy Lie algebras and  $L_\infty$ -morphisms, introduced by Schlessinger–Stasheff [17], is available from [16], cf. [10] in the context of present paper.

So, the general fact of (existence of) factorization,

$$\text{Assoc}(\star)(\mathcal{P})(f, g, h) = \diamond(\mathcal{P}, \llbracket \mathcal{P}, \mathcal{P} \rrbracket)(f, g, h), \quad f, g, h \in A[[\hbar]], \tag{1}$$

is known to the expert community. Indeed, this factorization is immediate from the construction of  $L_\infty$ -morphism in [15, §6.4]. We shall inspect how this mechanism works in practice, i.e. how precisely the  $\star$ -product is made associative in its perturbative expansion whenever the bi-vector  $\mathcal{P}$  is Poisson, thus satisfying the Jacobi identity  $\text{Jac}(\mathcal{P}) := \frac{1}{2}\llbracket \mathcal{P}, \mathcal{P} \rrbracket = 0$ . To the same extent as our paper [6] justifies a similar factorization,  $\llbracket \mathcal{P}, \mathcal{Q}(\mathcal{P}) \rrbracket = \diamond(\mathcal{P}, \llbracket \mathcal{P}, \mathcal{P} \rrbracket)$ , of the Poisson cocycle condition for universal deformations  $\dot{\mathcal{P}} = \mathcal{Q}(\mathcal{P})$  of Poisson structures<sup>2</sup>, we presently motivate the findings in [5] for  $\star \bmod \bar{o}(\hbar^3)$ , proceeding to the next order  $\star \bmod \bar{o}(\hbar^4)$  from [7] (and higher orders, recently available from [3]).<sup>3</sup> Let us emphasize that the theoretical constructions and algorithms (contained in the computer-assisted proof scheme under study and in the tools for graph weight calculation) would still work at arbitrarily high orders of expansion  $\star \bmod \bar{o}(\hbar^k)$  as  $k \rightarrow \infty$ . Explicit factorization (1) up to  $\bar{o}(\hbar^k)$  helps us build the star-product  $\star \bmod \bar{o}(\hbar^k)$  by using a self-starting iterative process, because the Jacobi identity for  $\mathcal{P}$  is the only obstruction to the associativity of  $\star$ . Specifically, the Kontsevich weights of graphs on fewer vertices (yet with a number of edges such that they do not show up in the perturbative expansion of  $\star$ ) dictate the coefficients of Leibniz orgraphs in operator  $\diamond$  at higher orders in  $\hbar$ . These weights in the r.-h.s. of (1) constrain the higher-order weights of the Kontsevich orgraphs in the expansion of  $\star$ -product itself. This is important also in the context of a number-theoretic open problem about the (ir)rational value  $(\text{const} \in \mathbb{Q} \setminus \{0\}) \cdot \zeta(3)^2/\pi^6 + (\text{const} \in \mathbb{Q})$  of a graph weight at  $\hbar^7$  in  $\star$  (see [12] and [3]).

Our paper is structured as follows. First, we fix notation and recall some basic facts from relevant theory. Secondly, we provide three examples which illustrate the work

<sup>1</sup>In fact, the morphism  $\mathcal{F}$  is a quasi-isomorphism (see [15, Th. 6.3]), inducing a bijection between the sets of gauge-equivalence classes of Maurer–Cartan elements.

<sup>2</sup>Universal w.r.t. all Poisson brackets on all finite-dimensional affine manifolds, such infinitesimal deformations were pioneered in [14]; explicit examples of these flows  $\dot{\mathcal{P}} = \mathcal{Q}(\mathcal{P})$  are given in [4, 8, 6].

<sup>3</sup>Note that both the approaches – to noncommutative associative  $\star$ -products and deformations of Poisson structures – rely on the same calculus of oriented graphs by Kontsevich [13, 14, 15].

of formality morphism in solving Eq. (1). Specifically, we read the operators  $\diamond_k = \diamond \pmod{\bar{o}(\hbar^k)}$  satisfying

$$\text{Assoc}(\star)(\mathcal{P})(f, g, h) \pmod{\bar{o}(\hbar^k)} = \diamond_k(\mathcal{P}, \llbracket \mathcal{P}, \mathcal{P} \rrbracket)(f, g, h) \tag{1'}$$

at  $k = 2, 3,$  and  $4$ . This corresponds to the expansions  $\star \pmod{\bar{o}(\hbar^k)}$  in [15], [5], and [7], respectively. One can then continue with  $k = 5, 6$ ; these expansions are in [3]. Independently, one can probe such factorizations using other stable formality morphisms: for instance, the ones which correspond to a different star-product, the weights in which are determined by a logarithmic propagator instead of the harmonic one (see [1]).

### 1. Two differential graded Lie algebra structures

Let  $M^r$  be an  $r$ -dimensional affine real manifold (we set  $\mathbb{k} = \mathbb{R}$  for simplicity). In the algebra  $A := C^\infty(M^r)$  of smooth functions, denote by  $\mu_A$  (or equivalently, by the dot  $\cdot$ ) the usual commutative, associative, bi-linear multiplication. The space of formal power series in  $\hbar$  over  $A$  will be  $A[[\hbar]]$  and the  $\hbar$ -linear multiplication in it is  $\mu$  (instead of  $\mu_{A[[\hbar]]}$ ). Consider two differential graded Lie algebra structures. First, we have that the shifted-graded space  $T_{\text{poly}}^{\downarrow[1]}(M^r)$  of multivector fields on  $M^r$  is equipped with the shifted-graded skew-symmetric Schouten bracket  $\llbracket \cdot, \cdot \rrbracket$  (itself bi-linear by construction and satisfying the shifted-graded Jacobi identity); the differential is set to zero. Secondly, the vector space  $D_{\text{poly}}^{\downarrow[1]}(M^r)$  of polydifferential operators (linear in each argument but not necessarily skew over the set of arguments or a derivation in any of them) is graded by using the number of arguments  $m$ : by definition, let  $\text{deg}(\theta(m \text{ arguments})) := m - 1$ . For instance,  $\text{deg}(\mu_A) = 1$ . The Lie algebra structure on  $D_{\text{poly}}^{\downarrow[1]}(M^r)$  is the Gerstenhaber bracket  $[\cdot, \cdot]_G$ ; for two homogeneous operators  $\Phi_1$  and  $\Phi_2$  it equals  $[\Phi_1, \Phi_2]_G = \Phi_1 \circ \Phi_2 - (-)^{\text{deg } \Phi_1 \cdot \text{deg } \Phi_2} \Phi_2 \circ \Phi_1$ , where the directed, non-associative insertion product is, by definition

$$(\Phi_1 \circ \Phi_2)(a_0, \dots, a_{k_1+k_2}) = \sum_{i=0}^{k_1} (-)^{ik_2} \Phi_1(a_0 \otimes \dots \otimes a_{i-1} \otimes \Phi_2(a_i \otimes \dots \otimes a_{i+k_2}) \otimes a_{i+k_2+1} \otimes \dots \otimes a_{k_1+k_2}).$$

In the above,  $\Phi_i: A^{\otimes(k_i+1)} \rightarrow A$  so that  $a_j \in A$ . Like  $\llbracket \cdot, \cdot \rrbracket$ , the Gerstenhaber bracket satisfies the shifted-graded Jacobi identity. The Hochschild differential on  $D_{\text{poly}}^{\downarrow[1]}(M^r)$  is  $d_H = [\mu_A, \cdot]_G$ ; indeed, its square vanishes,  $d_H^2 = 0$ , due to the Jacobi identity for  $[\cdot, \cdot]_G$  into which one plugs the equality  $[\mu_A, \mu_A]_G = 0$ .

Example 1. The associativity of the product  $\mu_A$  in the algebra of functions  $A = C^\infty(M^r)$  is the statement that

$$\begin{aligned} &\mu_A^{(1)}(\mu_A^{(2)}(a_0, a_1), a_2) + (-1)^{(i=1) \cdot (\text{deg } \mu_A=1)} \mu_A^{(1)}(a_0, \mu_A^{(2)}(a_1, a_2)) \\ &\quad - (-)^{(\text{deg } \mu_A^{(1)}=1) \cdot (\text{deg } \mu_A^{(2)}=1)} \{ \mu_A^{(1)}(\mu_A^{(1)}(a_0, a_1), a_2) - \mu_A^{(2)}(a_0, \mu_A^{(1)}(a_1, a_2)) \} \\ &\quad = 2\{(a_0 \cdot a_1) \cdot a_2 - a_0 \cdot (a_1 \cdot a_2)\} = 0. \end{aligned}$$

So, the associator  $\text{Assoc}(\mu_A)(a_0, a_1, a_2) = \frac{1}{2}[\mu_A, \mu_A]_G(a_0, a_1, a_2) = 0$  for any  $a_j \in A$ .

2. The Maurer–Cartan elements

In every differential graded Lie algebra with a Lie bracket  $[\cdot, \cdot]$ , the Maurer–Cartan (MC) elements are solutions of degree 1 for the Maurer–Cartan equation

$$d\alpha + \frac{1}{2}[\alpha, \alpha] = 0, \tag{2}$$

where  $d$  is the differential (equal, we recall, to zero identically on  $T_{\text{poly}}^{\downarrow[1]}(M^r)$  and  $d_H = [\mu_A, \cdot]_G$  on  $D_{\text{poly}}^{\downarrow[1]}(M^r)$ . Likewise, the Lie algebra structure  $[\cdot, \cdot]$  is the Schouten bracket  $[[\cdot, \cdot]]$  and Gerstenhaber bracket  $[\cdot, \cdot]_G$ , respectively.)

Now tensor the degree-one parts of both dgLa structures with  $\hbar \cdot \mathbb{k}[[\hbar]]$ , i.e. with formal power series starting at  $\hbar^1$ , and, preserving the notation (that is, extending the brackets and the differentials by  $\hbar$ -linearity), consider the same Maurer–Cartan equation (2). Let us study its formal power series solutions  $\alpha = \hbar^1 \alpha_1 + \dots$ .

So far, in the Poisson world we have that the Maurer–Cartan bi-vectors are formal Poisson structures  $0 + \hbar \mathcal{P}_1 + \bar{o}(\hbar)$  satisfying (2), which is  $[[\hbar \mathcal{P}_1 + \bar{o}(\hbar), \hbar \mathcal{P}_1 + \bar{o}(\hbar)]] = 0$  with zero differential. In the world of associative structures, the Maurer–Cartan elements are the tails  $B$  in expansions  $\star = \mu + B$ , so that the associativity equation  $[\star, \star]_G = 0$  reads (for  $[\mu, \mu]_G = 0$ )

$$[\mu, B]_G + \frac{1}{2}[B, B]_G = 0,$$

which is again (2).

3. The  $L_\infty$ -morphisms

Our goal is to have (and use) a morphism  $T_{\text{poly}}^{\downarrow[1]}(M^r) \rightarrow D_{\text{poly}}^{\downarrow[1]}(M^r)$  which would induce a map that takes Maurer–Cartan elements in the Poisson world to Maurer–Cartan elements in the associative world.

The leading term  $\mathcal{F}_1$ , i.e. the first approximation to the morphism which we consider, is the Hochschild–Kostant–Rosenberg (HKR) map (obviously, extended by linearity),

$$\mathcal{F}: \xi_1 \wedge \dots \wedge \xi_m \mapsto \frac{1}{m!} \sum_{\sigma \in \mathcal{S}_m} (-)^{\sigma} \xi_{\sigma(1)} \otimes \dots \otimes \xi_{\sigma(m)},$$

which takes a split multi-vector to a polydifferential operator (in fact, an  $m$ -vector). More explicitly, we have that

$$\mathcal{F}_1: (\xi_1 \wedge \dots \wedge \xi_m) \mapsto \left( a_1 \otimes \dots \otimes a_m \mapsto \frac{1}{m!} \sum_{\sigma \in \mathcal{S}_m} (-)^{\sigma} \prod_{i=1}^m \xi_{\sigma(i)}(a_i) \right), \tag{3}$$

here  $a_j \in A := C^\infty(M^r)$ . For zero-vectors  $h \in A$ , one has  $\mathcal{F}_1: h \mapsto (1 \mapsto h)$ .

Claim 1 ([15, §4.6.2]). The leading term, map  $\mathcal{F}_1$ , is not a Lie algebra morphism (which, if it were, would take the Schouten bracket of multivectors to the Gerstenhaber bracket of polydifferential operators).

Proof (by counterexample). Take two bi-vectors; their Schouten bracket is a tri-vector, but the Gerstenhaber bracket of two bi-vectors is a differential operator which has homogeneous components of differential orders (2,1,1) and (1,1,2). And in general, those components do not vanish.  $\square$

The construction of not a single map  $\mathcal{F}_1$  but of an entire collection  $\mathcal{F} = \{\mathcal{F}_n, n \geq 1\}$  of maps does nevertheless yield a well-defined mapping of the Maurer–Cartan elements from the two differential graded Lie algebras.<sup>4</sup>

Theorem 2 ([15, Main Theorem]). There exists a collection of linear maps  $\mathcal{F} = \{\mathcal{F}_n: T_{\text{poly}}^{\downarrow[1]}(M^r)^{\otimes n} \rightarrow D_{\text{poly}}^{\downarrow[1]}(M^r), n \geq 1\}$  such that  $\mathcal{F}_1$  is the HKR map (3) and  $\mathcal{F}$  is an  $L_\infty$ -morphism of the two differential graded Lie algebras:  $(T_{\text{poly}}^{\downarrow[1]}(M^r), \llbracket \cdot, \cdot \rrbracket, d = 0) \rightarrow (D_{\text{poly}}^{\downarrow[1]}(M^r), [\cdot, \cdot]_G, d_H = [\mu_A, \cdot]_G)$ . Namely,

- (1) each component  $\mathcal{F}_n$  is homogeneous of own grading  $1 - n$ ,
- (2) each morphism  $\mathcal{F}_n$  is graded skew-symmetric, i.e.

$$\mathcal{F}_n(\dots, \xi, \eta, \dots) = -(-)^{\text{deg}(\xi) \cdot \text{deg}(\eta)} \mathcal{F}_n(\dots, \eta, \xi, \dots)$$

for  $\xi, \eta$  homogeneous,

- (3) for each  $n \geq 1$  and (homogeneous) multivectors  $\xi_1, \dots, \xi_n \in T_{\text{poly}}^{\downarrow[1]}(M^r)$ , we have that (cf. [11, §3.6])

$$\begin{aligned} d_H(\mathcal{F}_n(\xi_1, \dots, \xi_n)) - (-)^{n-1} \sum_{i=1}^n (-)^u \mathcal{F}_n(\xi_1, \dots, d\xi_i, \dots, \xi_n) \\ + \frac{1}{2} \sum_{\substack{p+q=n \\ p,q>0}} \sum_{\sigma \in S_{p,q}} (-)^{pn+t} [\mathcal{F}_p(\xi_{\sigma(1)}, \dots, \xi_{\sigma(p)}), \mathcal{F}_q(\xi_{\sigma(p+1)}, \dots, \xi_{\sigma(n)})]_G \\ = (-)^n \sum_{i < j} (-)^s \mathcal{F}_{n-1}([\xi_i, \xi_j], \xi_1, \dots, \widehat{\xi}_i, \dots, \widehat{\xi}_j, \dots, \xi_n). \end{aligned} \quad (4)$$

In the above formula,  $\sigma$  runs through the set of  $(p, q)$ -shuffles, i.e. all permutations  $\sigma \in S_n$  such that  $\sigma(1) < \dots < \sigma(p)$  and independently  $\sigma(p+1) < \dots < \sigma(n)$ ; the exponents  $t$  and  $s$  are the numbers of transpositions of odd elements which we count when passing  $(t)$  from  $(\mathcal{F}_p, \mathcal{F}_q, \xi_1, \dots, \xi_n)$  to  $(\mathcal{F}_p, \xi_{\sigma(1)}, \dots, \xi_{\sigma(p)}, \mathcal{F}_q, \xi_{\sigma(p+1)}, \dots, \xi_{\sigma(n)})$ , and  $(s)$  from  $(\xi_1, \dots, \xi_n)$  to  $(\xi_i, \xi_j, \xi_1, \dots, \widehat{\xi}_i, \dots, \widehat{\xi}_j, \dots, \xi_n)$ .<sup>5</sup>

Remark 1. Let  $n := 1$ , then equality (4) in Theorem 2 is

$$d_H \circ \mathcal{F}_1 - (-)^{1-1} \cdot (-)^{u=0 \text{ from } (d, \xi_1) \mapsto (d, \xi_1)} \mathcal{F}_1 \circ d = 0 \iff d_H \circ \mathcal{F}_1 = \mathcal{F}_1 \circ d,$$

whence  $\mathcal{F}_1$  is a morphism of complexes.

- Let  $n := 2$ , then for any homogeneous multivectors  $\xi_1$  and  $\xi_2$ ,

$$\mathcal{F}_1(\llbracket \xi_1, \xi_2 \rrbracket) - [\mathcal{F}_1(\xi_1), \mathcal{F}_1(\xi_2)]_G = d_H(\mathcal{F}_2(\xi_1, \xi_2)) + \mathcal{F}_2((d = 0)(\xi_1), \xi_2) + (-)^{\text{deg} \xi_1} \mathcal{F}_2(\xi_1, (d = 0)(\xi_2)),$$

so that in our case  $\mathcal{F}_1$  is “almost” a Lie algebra morphism but for the discrepancy which is controlled by the differential of the (value of the) succeeding map  $\mathcal{F}_2$  in the sequence  $\mathcal{F} = \{\mathcal{F}_n, n \geq 1\}$ . Big formula (4) shows in precisely which sense this is also the case for higher homotopies  $\mathcal{F}_n, n \geq 2$  in the  $L_\infty$ -morphism  $\mathcal{F}$ . Indeed, an  $L_\infty$ -morphism is a map between dgLas which, in every term, almost preserves the bracket up to a homotopy  $d_H \circ \{\dots\}$  provided by the next term.

<sup>4</sup>The name ‘Formality’ for the collection  $\mathcal{F}$  of maps is motivated by Theorem 4.10 in [15] and by the main theorem in loc. cit.

<sup>5</sup>The exponent  $u$  is not essential for us now because the differential  $d$  on  $T_{\text{poly}}^{\downarrow[1]}(M^r)$  is set equal to zero identically, so that the entire term with  $u$  does not contribute (recall  $\mathcal{F}_n$  is linear).



Even though neither  $\mathcal{F}_1$  nor the entire collection  $\mathcal{F} = \{\mathcal{F}_n, n \geq 1\}$  is a dgLa morphism, their defining property (4) guarantees that  $\mathcal{F}$  gives us a well defined mapping of the Maurer–Cartan elements (which, we recall, are formal Poisson bi-vectors and tails  $B$  of associative (non)commutative multiplications  $\star = \mu + B$  on  $A[[\hbar]]$ , respectively).

Corollary 3. The natural  $\hbar$ -linear extension of  $\mathcal{F}$ , now acting on the space of formal power series in  $\hbar$  with coefficients in  $T_{\text{poly}}^{\llbracket 1 \rrbracket}(M^r)$  and with zero free term by the rule

$$\xi \mapsto \sum_{n \geq 1} \frac{1}{n!} \mathcal{F}_n(\xi, \dots, \xi),$$

takes the Maurer–Cartan elements  $\tilde{\mathcal{P}} = \hbar\mathcal{P} + \bar{o}(\hbar)$  to the Maurer–Cartan elements  $B = \sum_{n \geq 1} \frac{1}{n!} \mathcal{F}_n(\tilde{\mathcal{P}}, \dots, \tilde{\mathcal{P}}) = \hbar\tilde{\mathcal{P}} + \bar{o}(\hbar)$ . (Note that the HKR map  $\mathcal{F}_1$ , extended by  $\hbar$ -linearity, still is an identity mapping on multivectors, now viewed as special polydifferential operators.)

In plain terms, for a bivector  $\mathcal{P}$  itself Poisson, formal Poisson structures  $\tilde{\mathcal{P}} = \hbar\mathcal{P} + \bar{o}(\hbar)$  satisfying  $\llbracket \tilde{\mathcal{P}}, \tilde{\mathcal{P}} \rrbracket = 0$  are mapped by  $\mathcal{F}$  to the tails  $B = \hbar\mathcal{P} + \bar{o}(\hbar)$  such that  $\star = \mu + B$  is associative and its leading order deformation term is a given Poisson structure  $\mathcal{P}$ .

Proof (of Corollary 3). Let us presently consider the restricted case when  $\tilde{\mathcal{P}} = \hbar\mathcal{P}$ , without any higher order tail  $\bar{o}(\hbar)$ . The Maurer–Cartan equation in  $D_{\text{poly}}^{\llbracket 1 \rrbracket}(M^r) \otimes \hbar\mathbb{k}[[\hbar]]$  is  $[\mu, B]_G + \frac{1}{2}[B, B]_G = 0$ , where  $B = \sum_{n \geq 1} \frac{1}{n!} \mathcal{F}_n(\tilde{\mathcal{P}}, \dots, \tilde{\mathcal{P}})$  and we let  $\tilde{\mathcal{P}} = \hbar\mathcal{P}$ , so that  $B = \sum_{n \geq 1} \frac{\hbar^n}{n!} \mathcal{F}_n(\mathcal{P}, \dots, \mathcal{P})$ . Let us plug this formal power series in the l.h.s. of the above equation. Equating the coefficients at powers  $\hbar^n$  and multiplying by  $n!$ , we obtain the expression

$$[\mu, \mathcal{F}_n(\mathcal{P}, \dots, \mathcal{P})]_G + \frac{1}{2} \sum_{\substack{p+q=n \\ p, q > 0}} \frac{n!}{p!q!} [\mathcal{F}_p(\mathcal{P}, \dots, \mathcal{P}), \mathcal{F}_q(\mathcal{P}, \dots, \mathcal{P})]_G.$$

It is readily seen that now the sum  $\sum_{\sigma \in \mathcal{S}_{p,q}}$  in (4) over the set of  $(p, q)$ -shuffles of  $n = p + q$  identical copies of an object  $\mathcal{P}$  just counts the number of ways to pick  $p$  copies going first in an ordered string of length  $n$ . To balance the signs, we note at once that by item 2 in Theorem 2, see above,  $\mathcal{F}_p(\dots, \mathcal{P}^{(\alpha)}, \mathcal{P}^{(\alpha+1)}, \dots) = +\mathcal{F}_p(\dots, \mathcal{P}^{(\alpha+1)}, \mathcal{P}^{(\alpha)}, \dots)$  because bi-vector’s shifted degree is  $+1$ , so that no  $(p, q)$ -shuffles of  $(\mathcal{P}, \dots, \mathcal{P})$  contribute with any sign factor. The only sign contribution that remains stems from the symbol  $\mathcal{F}_q$  of grading  $1 - q$  transported along  $p$  copies of odd-degree bi-vector  $\mathcal{P}$ ; this yields  $t = (1 - p) \cdot q$  and  $(-)^{pm+t} = (-)^{p \cdot (p+q)} \cdot (-)^{(1-q) \cdot p} = (-)^{p \cdot (p+1)} = +$ .

The left-hand side of the Maurer–Cartan equation (2) is, by the above, expressed by the left-hand side of (4) which the  $L_\infty$ -morphism  $\mathcal{F}$  satisfies. In the right-hand side of (4), we now obtain (with, actually, whatever sign factors) the values of linear mappings  $\mathcal{F}_{n-1}$  at twice the Jacobiator  $\llbracket \tilde{\mathcal{P}}, \tilde{\mathcal{P}} \rrbracket$  as one of the arguments. All these values are therefore zero, which implies that the right-hand side of the Maurer–Cartan equation (2) vanishes, so that the tail  $B$  indeed is a Maurer–Cartan element in the Hochschild cochain complex (in other words, the star-product  $\star = \mu + B$  is associative).

This completes the proof in the restricted case when  $\tilde{\mathcal{P}} = \hbar\mathcal{P}$ . Formal power series bi-vectors  $\tilde{\mathcal{P}} = \hbar\mathcal{P} + \bar{o}(\hbar)$  refer to the same count of signs as above, yet the calculation of multiplicities at  $\hbar^n$  (for all possible lexicographically ordered  $p$ - and  $q$ -tuples of  $n$  arguments) is an extensive exercise in combinatorics.  $\square$

Corollary 4. Because the right-hand side of (2) in the above reasoning is determined by the right-hand side of (4), we read off an explicit formula of the operator  $\diamond$  that solves the factorization problem

$$\text{Assoc}(\star)(\mathcal{P})(f, g, h) = \diamond(\mathcal{P}, \llbracket \mathcal{P}, \mathcal{P} \rrbracket)(f, g, h), \quad f, g, h \in A[[\hbar]]. \quad (1)$$

Indeed, the operator is

$$\diamond = 2 \cdot \sum_{n \geq 1} \frac{\hbar^n}{n!} \cdot c_n \cdot \mathcal{F}_{n-1}(\llbracket \mathcal{P}, \mathcal{P} \rrbracket, \mathcal{P}, \dots, \mathcal{P}). \quad (5)$$

But what are the coefficients  $c_n \in \mathbb{R}$  equal to? Let us find it out.

#### 4. Explicit construction of the formality morphism $\mathcal{F}$

The first explicit formula for the formality morphism  $\mathcal{F}$  which we study in this paper was discovered by Kontsevich in [15, §6.4], providing an expansion of every term  $\mathcal{F}_n$  using weighted decorated graphs:

$$\mathcal{F} = \left\{ \mathcal{F}_n = \sum_{m \geq 0} \sum_{\Gamma \in G_{n,m}} W_\Gamma \cdot \mathcal{U}_\Gamma \right\}.$$

Here  $\Gamma$  belongs to the set  $G_{n,m}$  of oriented graphs on  $n$  internal vertices (i.e. arrowtails),  $m$  sinks (from which no arrows start), and  $2n + m - 2 \geq 0$  edges, such that at every internal vertex there is an ordering of outgoing edges. By decorating each edge with a summation index that runs from 1 to  $r$ , by viewing each edge as a derivation  $\partial/\partial x^\alpha$  of the arrowhead vertex content, by placing  $n$  multivectors from an ordered tuple of arguments of  $\mathcal{F}_n$  into the respective vertices, now taking the sum over all indices of the resulting products of the content of vertices, and skew-symmetrizing over the  $n$ -tuple of (shifted-)graded multivectors, we realize each graph at hand as a polydifferential operator  $T_{\text{poly}}^{\llbracket [1] \rrbracket}(M^r)^{\otimes n} \rightarrow D_{\text{poly}}^{\llbracket [1] \rrbracket}(M^r)$  whose arguments are multivectors. Note that the value  $\mathcal{F}_n(\xi_1, \dots, \xi_n)$  itself is, by construction, a differential operator w.r.t. the contents of sinks of the graph  $\Gamma$ . All of this is discussed in detail in [13, 14, 15] or [4, 5, 7].

The formula for the harmonic weights  $W_\Gamma \in \mathbb{R}$  is given in [15, §6.2]; it is

$$W_\Gamma = \left( \prod_{k=1}^n \frac{1}{\#\text{Star}(k)!} \right) \cdot \frac{1}{(2\pi)^{2n+m-2}} \int_{\bar{C}_{n,m}^+} \bigwedge_{e \in E_\Gamma} d\phi_e,$$

where  $\#\text{Star}(k)$  is the number of edges starting from vertex  $k$ ,  $d\phi_e$  is the “harmonic angle” differential 1-form associated to the edge  $e$ , and the integration domain  $\bar{C}_{n,m}^+$  is the connected component of  $\bar{C}_{n,m}$  which is the closure of configurations where points  $q_j$ ,  $1 \leq j \leq m$  on  $\mathbb{R}$  are placed in increasing order:  $q_1 < \dots < q_m$ . For convenience, let us also define

$$w_\Gamma = \left( \prod_{k=1}^n \#\text{Star}(k)! \right) \cdot W_\Gamma.$$

The convenience is that by summing over labelled graphs  $\Gamma$ , we actually sum over the equivalence classes  $[\Gamma]$  (i.e. over unlabeled graphs) with multiplicities  $(w_\Gamma/W_\Gamma) \cdot n!/\#\text{Aut}(\Gamma)$ . The division by the volume  $\#\text{Aut}(\Gamma)$  of the symmetry group eliminates the repetitions of graphs which differ only by a labeling of vertices but, modulo such, do not differ by the labeling of ordered edge tuples (issued from the vertices which are matched by a symmetry).

Let us remember that the integrand in the formula of  $W_\Gamma$  is defined in terms of the harmonic propagator; other propagators (e.g. logarithmic, or other members of the family interpolating between harmonic and logarithmic [1]) would give other formality morphisms. A path integral realization of the  $\star$ -product itself and of the components  $\mathcal{F}_n$  in the formality morphism is proposed in [10].

To calculate the graph weights  $W_\Gamma$  in practice, we employ methods which were outlined in [7], as well as [12, App. E] (about the cyclic weight relations), and [3] that puts those real values in the context of Riemann multiple zeta functions and polylogarithms.<sup>6</sup> Examples of such decorated oriented graphs  $\Gamma$  and their weights  $W_\Gamma$  will be given in the next section.

4.1. Sum over equivalence classes. The sum in Kontsevich’s formula is over labeled graphs: internal vertices are numbered from 1 to  $n$ , and the edges starting from each internal vertex  $k$  are numbered from 1 to  $\#\text{Star}(k)$ . Under a re-labeling  $\sigma : \Gamma \mapsto \Gamma^\sigma$  of internal vertices and edges it is seen from the definitions that the operator  $\mathcal{U}_\Gamma$  and the weight  $W_\Gamma$  enjoy the same skew-symmetry property (as remarked in [15, §6.5]), whence  $W_\Gamma \cdot \mathcal{U}_\Gamma = W_{\Gamma^\sigma} \cdot \mathcal{U}_{\Gamma^\sigma}$ . It follows that the sum over labeled graphs can be replaced by a sum over equivalence classes  $[\Gamma]$  of graphs, modulo labeling of internal vertices and edges. For this it remains to count the size of an equivalence class: the edges can be labeled in  $\prod_{k=1}^n \#\text{Star}(k)!$  ways, while the  $n$  internal vertices can be labeled in  $n!/\#\text{Aut}(\Gamma)$  ways.

Example 2. The double wedge on two ground vertices has only one possible labeling of vertices, due to the automorphism that interchanges the wedges.

We denote by  $M_\Gamma = (\prod_{k=1}^n \#\text{Star}(k)!) \cdot n!/\#\text{Aut}(\Gamma)$  the multiplicity of the graph  $\Gamma$ , and let  $\tilde{G}_{n,m}$  be the set of equivalence classes  $[\Gamma]$  modulo labeling of  $\Gamma \in G_{n,m}$ . The formula for the formality morphism can then be rewritten as

$$\mathcal{F} = \{ \mathcal{F}_n = \sum_{m \geq 0} \sum_{[\Gamma] \in \tilde{G}_{n,m}} M_\Gamma \cdot W_\Gamma \cdot \mathcal{U}_\Gamma \};$$

here the  $\Gamma$  in  $M_\Gamma \cdot W_\Gamma \cdot \mathcal{U}_\Gamma$  is any representative of  $[\Gamma]$ . Any ambiguity in signs (due to the choice of representative) in the latter two factors is cancelled in their product. Note that the factor  $(\prod_{k=1}^n \#\text{Star}(k)!)$  in  $M_\Gamma$  kills the corresponding factor in  $W_\Gamma$ , as remarked in [15, §6.5].

4.2. The coefficient of a graph in the  $\star$ -product. The  $\star$ -product associated to a Poisson structure  $\mathcal{P}$  is given by Corollary 3:

$$\star = \mu + \sum_{n \geq 1} \frac{\hbar^n}{n!} \mathcal{F}_n(\mathcal{P}, \dots, \mathcal{P}) = \mu + \sum_{n \geq 1} \frac{\hbar^n}{n!} \sum_{[\Gamma] \in \tilde{G}_{n,2}} M_\Gamma \cdot W_\Gamma \cdot \mathcal{U}_\Gamma(\mathcal{P}, \dots, \mathcal{P}).$$

For a graph  $\Gamma \in G_{n,2}$  such that each internal vertex has two outgoing edges (these are the only graphs that contribute, because we insert bi-vectors) we have  $M_\Gamma = 2^n \cdot n!/\#\text{Aut}(\Gamma)$ . In total, the coefficient of  $\mathcal{U}_\Gamma(\mathcal{P}, \dots, \mathcal{P})$  at  $\hbar^n$  is  $2^n/\#\text{Aut}(\Gamma) \cdot W_\Gamma = w_\Gamma/\#\text{Aut}(\Gamma)$ . The skew-symmetrization without prefactor of bi-vector coefficients in  $\mathcal{U}_\Gamma(\mathcal{P}, \dots, \mathcal{P})$  provides an extra factor  $2^n$ .

<sup>6</sup>It is the values  $w_\Gamma$  instead of  $W_\Gamma$  which are calculated by software [3].

Example 3 (at  $\hbar^1$ ). The coefficient of the wedge graph is  $1/2$  and the operator is  $2\mathcal{P}$ , hence we recover  $\mathcal{P}$ .

4.3. The coefficient of a Leibniz graph in the associator. The factorizing operator  $\diamond$  for  $\text{Assoc}(\star)$  is given by Corollary 4:

$$\begin{aligned} \diamond &= 2 \cdot \sum_{n \geq 1} \frac{\hbar^n}{n!} \cdot c_n \cdot \mathcal{F}_{n-1}(\llbracket \mathcal{P}, \mathcal{P} \rrbracket, \mathcal{P}, \dots, \mathcal{P}) \\ &= 2 \cdot \sum_{n \geq 1} \frac{\hbar^n}{n!} \cdot c_n \cdot \sum_{[\Gamma] \in \bar{G}_{n-1,3}} M_\Gamma \cdot W_\Gamma \cdot \mathcal{U}_\Gamma(\llbracket \mathcal{P}, \mathcal{P} \rrbracket, \mathcal{P}, \dots, \mathcal{P}). \end{aligned}$$

For a graph  $\Gamma \in G_{n-1,3}$  where one internal vertex has three outgoing edges and the rest have two, we have  $M_\Gamma = 3! \cdot 2^{n-2} \cdot (n-1)! / \#\text{Aut}(\Gamma)$ . In total, the coefficient of  $\mathcal{U}_\Gamma(\llbracket \mathcal{P}, \mathcal{P} \rrbracket, \mathcal{P}, \dots, \mathcal{P})$  at  $\hbar^n$  is

$$\left[ 2 \cdot \frac{1}{n!} \cdot c_n \cdot 3! \cdot 2^{n-2} \cdot (n-1)! \right] \cdot \frac{W_\Gamma}{\#\text{Aut}(\Gamma)} = \left[ 2 \cdot \frac{c_n}{n} \right] \cdot \frac{w_\Gamma}{\#\text{Aut}(\Gamma)}$$

The skew-symmetrization without prefactor of bi- and tri-vector coefficients in the operator  $\mathcal{U}_\Gamma(\llbracket \mathcal{P}, \mathcal{P} \rrbracket, \mathcal{P}, \dots, \mathcal{P})$  provides an extra factor  $3! \cdot 2^{n-2}$ .

Example 4 (at  $\hbar^2$ ). The coefficient of the tripod graph is  $c_2 \cdot \frac{1}{3!}$  and the operator is  $3! \cdot \llbracket \mathcal{P}, \mathcal{P} \rrbracket$ , hence we recover  $c_2 \llbracket \mathcal{P}, \mathcal{P} \rrbracket = \frac{2}{3} \text{Jac}(\mathcal{P})$ . (The right-hand side is known from the associator, e.g. from [5].) This yields  $c_2 = 1/3$ . In addition, we see that the HKR map  $\mathcal{F}_1$  acts here by the identity on  $\llbracket \mathcal{P}, \mathcal{P} \rrbracket$ .

In the next section, we shall find that at  $\hbar^n$ , the coefficients of our Leibniz graphs (with  $\text{Jac}(\mathcal{P})$  inserted instead of  $\llbracket \mathcal{P}, \mathcal{P} \rrbracket$ ) are

$$\frac{\llbracket \mathcal{P}, \mathcal{P} \rrbracket}{\text{Jac}(\mathcal{P})} \cdot \left[ 3! \cdot 2^{n-2} \right] \cdot \left[ 2 \cdot \frac{c_n}{n} \right] \cdot \frac{w_\Gamma}{\#\text{Aut}(\Gamma)} = 2^n \cdot \frac{w_\Gamma}{\#\text{Aut}(\Gamma)}, \quad \text{so} \quad 3! \cdot 2^n \cdot \frac{c_n}{n} = 2^n.$$

We deduce that  $c_n = n/3! = n/6$  in all our experiments.

Conjecture. For all  $n \geq 2$ , the coefficients in (5) are  $c_n = n/3! = n/6$  (hence, the coefficients of markers  $\Gamma$  for equivalence classes  $[\Gamma]$  of the Leibniz graphs in (5) are  $2^n \cdot w_\Gamma / \#\text{Aut}(\Gamma)$ ), although it still remains to be explained how exactly this follows from the  $L_\infty$  condition (4).

## 5. Examples

Let  $\mathcal{P}$  be a Poisson bi-vector on an affine manifold  $M^r$ . We inspect the associativity of the star-product  $\star = \mu + \sum_{n \geq 1} \frac{\hbar^n}{n!} \mathcal{F}_n(\mathcal{P}, \dots, \mathcal{P})$  given by Corollary 3 by illustrating the work of the factorization mechanism from Corollary 4. The powers of deformation parameter  $\hbar$  provide a natural filtration  $\hbar^2 \cdot \mathbf{A}^{(2)} + \hbar^3 \cdot \mathbf{A}^{(3)} + \hbar^4 \cdot \mathbf{A}^{(4)} + \bar{o}(\hbar^4)$  so that we verify the vanishing of  $\text{Assoc}(\star)(\mathcal{P})(\cdot, \cdot, \cdot) \bmod \bar{o}(\hbar^4)$  for  $\star \bmod \bar{o}(\hbar^4)$  order by order.

At  $\hbar^0$  there is nothing to do (indeed, the usual multiplication is associative). All contribution to the associator of  $\star$  at  $\hbar^1$  cancels out because the leading deformation term  $\hbar \mathcal{P}$  in the star-product  $\star = \mu + \hbar \mathcal{P} + \bar{o}(\hbar)$  is a bi-derivation. The order  $\hbar^2$  was discussed in Example 4 in §4.3.

Remark 2. In all our reasoning at any order  $\hbar^{n \geq 2}$ , the Jacobiator in Leibniz graphs is expanded (w.r.t. the three cyclic permutations of its arguments) into the Kontsevich graphs, built of wedges, in such a way that the internal edge, connecting two Poisson bi-vectors in  $\text{Jac}(\mathcal{P})$ , is proclaimed Left by construction. Specifically, the algorithm to expand each Leibniz graphs is as follows:

- (1) Split the trivalent vertex with ordered targets  $(a, b, c)$  into two wedges: the first wedge stands on  $a$  and  $b$  (in that order), and the second wedge stands on the first wedge-top and  $c$  (in that order), so that the internal edge of the Jacobiator is marked Left, preceding the Right edge towards  $c$ .
- (2) Re-direct the edges (if any) which had the tri-valent vertex as their target, to one of the wedge-tops; take the sum over all possible combinations (this is the iterated Leibniz rule).
- (3) Take the sum over cyclic permutations of the targets of the edges which (initially) have  $(a, b, c)$  as their targets (this is the expansion of the Jacobiator).

5.1. The order  $\hbar^3$ . To factorize the next order expansion of the associator,  $\text{Assoc}(\star)(\mathcal{P}) \bmod \bar{o}(\hbar^3) = \hbar^2 \cdot \mathbf{A}^{(2)} + \hbar^3 \cdot \mathbf{A}^{(3)} + \bar{o}(\hbar^3)$ , at  $\hbar^3$  in the operator  $\diamond$  in the right-hand side of (1), we use graphs on  $n - 1 = 2$  vertices,  $m = 3$  sinks, and  $2(n - 1) + m - 2 = 5$  edges.

At  $\hbar^3$ , two internal vertices in the Leibniz graphs in the r.-h.s. of factorization (1) are manifestly different: one vertex, containing the bi-vector  $\mathcal{P}$ , is a source of two outgoing edges, and the other, with  $[[\mathcal{P}, \mathcal{P}]]$ , of three. Therefore, the automorphism groups of such Leibniz graphs (under relabellings of internal vertices of the same valency but with the sinks fixed) can only be trivial, i.e. one-element. (This will not necessarily be the case of Leibniz graphs on  $(n - 2) + 1$  internal vertices at  $\hbar^{\geq 4}$ : compare Examples 8 vs 9 on p. 316 below, where the weight of a graph is divided further by the size of its automorphism group.)

The coefficient of  $\hbar^3$  in the factorizing operator  $\diamond$ ,

$$\text{coeff}(\diamond, \hbar^3) = 2 \cdot \frac{1}{3!} \cdot c_3 \cdot \sum_{[\Gamma] \in \bar{\mathcal{G}}_{2,3}} M_\Gamma \cdot W_\Gamma \cdot \mathcal{U}_\Gamma([[ \mathcal{P}, \mathcal{P} ]], \mathcal{P}, \dots, \mathcal{P}),$$

expands into a sum of  $\leq 24$  admissible oriented graphs. Indeed, there are six essentially different oriented graph topologies, filtered by the number of sinks on which the tri-vector  $[[\mathcal{P}, \mathcal{P}]]$  and bi-vector  $\mathcal{P}$  stand; the ordering of sinks in the associator then yields  $3 + 3 + 3 \times 2 + 3 \times 2 + 3 = 24$  oriented graphs. (None of them is a zero orgraph.) As we recall from [5], only thirteen of them actually occur with nonzero coefficients in the term  $\mathbf{A}^{(3)} \sim \hbar^3$  in  $\text{Assoc}(\star)(\mathcal{P})$ , the remaining eleven have zero weights.<sup>7</sup> The weights of 15 relevant oriented Leibniz graphs from [5] are listed in Table 1.<sup>8</sup>

<sup>7</sup>Yet, these seemingly ‘unnecessary’ graphs can contribute to the cyclic weight relations (see [12, App. E]): zero values of some of such graph weights can simplify the system of linear relations between nonzero weights.

<sup>8</sup>To get the values, one uses the software [3] by Banks–Panzer–Pym or, independently, exact symbolic or approximate numeric methods from [7], also taking into account the cyclic weight relations from [12, App. E].

Table 1. Weights  $w_\Gamma$  of oriented Leibniz graphs  $\Gamma$  in  $\text{coeff}(\diamond, \hbar^3)$ .

$(S_f)_{221}$	$=$	$[01; 012]$	$\frac{1}{12}$	$(S_g)_{122}$	$=$	$[12; 012]$	$\frac{1}{12}$	$(S_h)_{212}$	$=$	$[20; 012]$	$\frac{-1}{12}$
$(I_f)_{112}$	$=$	$[02; 312]$	$\frac{1}{48}$	$(I_g)_{112}$	$=$	$[12; 032]$	$\frac{1}{48}$	$(S_h)_{112}$	$=$	$[24; 012]$	$\frac{-1}{24}$
$(S_f)_{211}$	$=$	$[04; 012]$	$\frac{1}{24}$	$(I_g)_{211}$	$=$	$[10; 032]$	$\frac{-1}{48}$	$(I_h)_{211}$	$=$	$[20; 013]$	$\frac{-1}{48}$
$(I_f)_{111}$	$=$	$[04; 312]$	$\frac{1}{48}$	$(I_h)_{111}$	$=$	$[24; 013]$	$\frac{-1}{48}$	$(I_g)_{111}$	$=$	$[14; 032]$	$0$
$(S_g)_{111}$	$=$	$[14; 012]$	$0$	$(I_f)_{121}$	$=$	$[01; 312]$	$\frac{1}{24}$	$(I_h)_{121}$	$=$	$[21; 013]$	$\frac{-1}{24}$

Here we let by definition

$$I_f := \partial_j(\text{Jac}(\mathcal{P})(\mathcal{P}^{ij}, g, h)) \partial_i f = \text{Diagram 1} - \text{Diagram 2} - \text{Diagram 3} = 0.$$

Likewise,  $I_g := \partial_j(\text{Jac}(\mathcal{P})(f, \mathcal{P}^{ij}, h)) \cdot \partial_i g$  and  $I_h := \partial_j(\text{Jac}(\mathcal{P})(f, g, \mathcal{P}^{ij})) \cdot \partial_i h$ , respectively.<sup>9</sup>

We also set

$$S_f := \mathcal{P}^{ij} \partial_j \text{Jac}(\mathcal{P})(\partial_i f, g, h) = \text{Diagram 4} - \text{Diagram 5} - \text{Diagram 6} = 0.$$

Similarly, we let  $S_g := \mathcal{P}^{ij} \partial_j \text{Jac}(\mathcal{P})(f, \partial_i g, h) = 0$  and  $S_h := \mathcal{P}^{ij} \partial_j \text{Jac}(\mathcal{P})(f, g, \partial_i h) = 0$ . Note that after all the Leibniz rules are reworked, each of the six graphs  $I_f, \dots, S_h$  – with the Jacobiator  $\text{Jac}(\mathcal{P}) = \frac{1}{2}[[\mathcal{P}, \mathcal{P}]]$  at the tri-valent vertex – splits into several homogeneous components, like  $(I_f)_{111}$  or  $(S_h)_{212}$ ; taken alone, each of the components encodes a zero polydifferential operator of respective orders.

Claim 5. Multiplied by a common factor  $([[\mathcal{P}, \mathcal{P}]] / \text{Jac}(\mathcal{P})) \cdot 2^{k-1} = 2 \cdot 4 = 8$ , the Leibniz graph weights from Table 1 at  $\hbar^3$  fully reproduce the factorization which was found in the main Claim in [5], namely:

$$\begin{aligned} \mathbf{A}_{221}^{(3)} &= \frac{2}{3}(S_f)_{221}, & \mathbf{A}_{122}^{(3)} &= \frac{2}{3}(S_g)_{122}, & \mathbf{A}_{212}^{(3)} &= -\frac{2}{3}(S_h)_{212}, \\ \mathbf{A}_{111}^{(3)} &= \frac{1}{6}(I_f - I_h)_{111}, & \mathbf{A}_{112}^{(3)} &= (\frac{1}{6}I_f + \frac{1}{6}I_g - \frac{1}{3}S_h)_{112}, \\ \mathbf{A}_{121}^{(3)} &= \frac{1}{3}(I_f - I_h)_{121}, & \mathbf{A}_{211}^{(3)} &= (\frac{1}{3}S_f - \frac{1}{6}I_g - \frac{1}{6}I_h)_{211}. \end{aligned}$$

Otherwise speaking, the sum of these Leibniz oriented graphs with these weights (times  $2 \cdot 4 = 8$ ), when expanded into the sum of 39 weighted Kontsevich graphs (built only of wedges), equals identically the  $\hbar^3$ -proportional term in the associator  $\text{Assoc}(\star)(\mathcal{P})(f, g, h)$ .

Proof scheme. The encodings of weighted Kontsevich-graph expansions of the homogeneous components of the weighted Leibniz graphs  $I_f, \dots, S_h$ , which show up in the associator at  $\hbar^3$  and which are processed according to the algorithm in Remark 2, are listed in Appendix A. Reducing that collection modulo skew symmetry at internal vertices, we reproduce, as desired, the entire term  $\mathbf{A}^{(3)}$  in the expansion  $\hbar^2 \cdot \mathbf{A}^{(2)} + \hbar^3 \cdot \mathbf{A}^{(3)} + \bar{o}(\hbar^3)$  of the associator  $\text{Assoc}(\star)(\mathcal{P}) \bmod \bar{o}(\hbar^3)$ .  $\square$

<sup>9</sup>In [5], the indices  $i$  and  $j$  were interchanged in the definitions of both  $I_g$  and  $I_h$  (compare the expression of  $I_f$ ); that typo is now corrected in the above formulae.

Three examples, corresponding to the leftmost column of equalities in Claim 5, illustrate this scheme at order  $\hbar^3$ . The three cases differ in that for  $\mathbf{A}_{221}^{(3)}$  in Example 5, there is just one Leibniz graph without any arrows acting on the Jacobiator vertex. In the other Example 6 for  $\mathbf{A}_{121}^{(3)}$ , there are two Leibniz graphs still without Leibniz-rule actions on the Jacobiators in them, so that we aim to show how similar terms are collected.<sup>10</sup> Finally, in Example 7 about  $\mathbf{A}_{111}^{(3)}$  there are two Leibniz graphs with one Leibniz rule action per either graph: an arrow targets the two internal vertices in the Jacobiator.

Example 5. Take the Leibniz graph  $(S_f)_{221} = [01; 012]$ . Its weight is  $1/12$ . Multiplying the Leibniz graph by 8 times its weight and expanding the Jacobiator (there are no Leibniz rules to expand) yields the sum of three Kontsevich graphs:  $\frac{2}{3}([01; 01; 42] + [01; 12; 40] + [01; 20; 41])$ . This is identically equal to the differential order  $(2, 2, 1)$  homogeneous part  $\mathbf{A}_{221}^{(3)}$  of  $\text{Assoc}(\star)(\mathcal{P})$  at  $\hbar^3$ . For instance, these terms are listed in [7, App. D].

Example 6. Take the Leibniz graphs  $(I_f)_{121} = [01; 312]$  and  $(I_h)_{121} = [21; 013]$ . Their weights are  $1/24$  and  $-1/24$ , respectively; multiply them by 8. Expanding the Jacobiator in the linear combination  $\frac{1}{3}(I_f - I_h)_{121}$  yields the sum of Kontsevich graphs  $\frac{1}{3}([01; 31; 42] + [01; 12; 43] + [01; 23; 41] - [21; 01; 43] - [21; 13; 40] - [21; 30; 41])$ . The two Leibniz graphs have a Kontsevich graph in common:  $[01; 12; 43] = [21; 01; 43]$  (recall that internal vertex labels can be permuted at no cost and the swap  $L \rightleftharpoons R$  at a wedge costs a minus sign). This gives one cancellation; the remaining four terms equal  $\mathbf{A}_{121}^{(3)}$  as listed in [7, App. D].

Example 7. Take the Leibniz graphs  $(I_f)_{111} = [04; 312]$  and  $(I_h)_{111} = [24; 013]$ . Their weights are  $1/48$  and  $-1/48$ , respectively; multiply them by 8. Expanding the Jacobiator and the Leibniz rule in the linear combination  $\frac{1}{6}(I_f - I_h)_{111}$  yields the sum of Kontsevich graphs:

$$\begin{aligned} & \frac{1}{6}([04; 31; 42] + [04; 12; 43] + [04; 23; 41] + [05; 31; 42] + [05; 12; 43] + [05; 23; 41] \\ & - [24; 01; 43] - [24; 13; 40] - [24; 30; 41] - [25; 01; 43] - [25; 13; 40] - [25; 30; 41]). \end{aligned}$$

Two pairs of graphs cancel; namely  $[05; 31; 42] = [25; 30; 41]$  and  $[05; 23; 41] = [25; 13; 40]$ . The remaining eight terms equal  $\mathbf{A}_{111}^{(3)}$  as listed in [7, App. D].

5.2. The order  $\hbar^4$ . Let us proceed with the term  $\mathbf{A}^{(4)}$  at  $\hbar^4$  in the associator  $\text{Assoc}(\star)(\mathcal{P})(\cdot, \cdot, \cdot)$  mod  $\bar{o}(\hbar^4)$ . The numbers of Kontsevich oriented graphs in the star-product expansion grow as fast as

$$\begin{aligned} \star &= \hbar^0 \cdot (\#\text{graphs} = 1) + \hbar^1 \cdot (\# = 1) + \hbar^2 \cdot (\# = 4) + \hbar^3 \cdot (\# = 13) + \hbar^4 \cdot (\# = 247) + \\ &+ \hbar^5 \cdot (\# = 2356) + \hbar^6 \cdot (\# = 66041) + \bar{o}(\hbar^6); \end{aligned}$$

here we report the count of all nonzero-weight Kontsevich oriented graphs. Counting them modulo automorphisms (which may also swap the sinks), Banks, Panzer, and Pym

<sup>10</sup>To collect and compare the Kontsevich orgraphs (built of wedges, i.e. ordered edge pairs issued from internal vertices), we can bring every such graph to its normal form, that is, represent it using the minimal base-(#sinks + #internal vertices) number, encoding the graph as the list of ordered pairs of target vertices, by running over all the relabellings of internal vertices. (The labelling of ordered sinks is always  $0 < 1 < \dots < m - 1$ .)

obtain the numbers ( $\hbar^0 : 1, \hbar^1 : 1, \hbar^2 : 3, \hbar^3 : 8, \hbar^4 : 133, \hbar^5 : 1209, \hbar^6 : 33268$ ). This shows that at orders  $\hbar^{k \geq 4}$ , the use of graph-processing software is indispensable in the task of verifying factorization (1) using weighted graph expansion (5) of the operator  $\diamond$ .

Specifically, the number of Kontsevich oriented graphs at  $\hbar^k$  in the left-hand side of the factorization problem  $\text{Assoc}(\star)(\mathcal{P})(\cdot, \cdot, \cdot) = \diamond(\mathcal{P}, \llbracket \mathcal{P}, \mathcal{P} \rrbracket)(\cdot, \cdot, \cdot)$ , and the number of Leibniz graphs which assemble with nonzero coefficients to a solution  $\diamond$  in the right-hand side is presented in Table 2. At  $\hbar^4$ , the expansion of  $\text{Assoc}(\star)(\mathcal{P}) \bmod \bar{o}(\hbar^4)$  requires 241

Table 2. Number of graphs in either side of the factorization.

$k$	2	3	4	5	6	7
LHS: # K. orgraphs	3 (Jac)	39	740	12464	290305	?
RHS: # L. orgraphs, coeff $\neq 0$	1 (Jac)	13	241	?	?	?
Reference	§4.3, [15]	§5.1, [5]	§5.2, [7]	[3]		

nonzero coefficients of Leibniz graphs on 3 sinks,  $2 = n - 1$  internal vertices for bi-vectors  $\mathcal{P}$  and one internal vertex for the tri-vector  $\llbracket \mathcal{P}, \mathcal{P} \rrbracket$ , and therefore,  $2(n - 1) + 3 = 2n + 3 - 2 = 7$  oriented edges.

Remark 3. Again, this set of Leibniz graphs is well structured. Indeed, it is a disjoint union of homogeneous differential operators arranged according to their differential orders w.r.t. the sinks, e.g., (1, 1, 1), (2, 1, 1), (1, 2, 1), (1, 1, 2), etc., up to (3, 3, 1).

Example 8. The Leibniz graph  $L_{331} := [01; 01; 012]$  of differential orders (3, 3, 1) has the weight  $1/24$  according to [3]. Multiplied by a universal (for all graphs at  $\hbar^4$ ) factor  $2^4 = 16$  and the factor  $1/(\# \text{Aut}(L_{331})) = 1/2$  due to this graph's symmetry ( $3 \rightleftharpoons 4$ ), it expands to  $\frac{1}{3}([01; 01; 01; 52] + [01; 01; 12; 50] + [01; 01; 20; 51])$  by the definition of Jacobi's identity. This sum of three weighted Kontsevich orgraphs reproduces exactly  $\mathbf{A}_{331}^{(4)}$ , which is known from [7, Table 8 in App. D].

Example 9. The Leibniz graph  $L_{322} := [01; 02; 012]$  of differential orders (3, 2, 2) has the weight  $1/24$  according to [3]. Multiplied now by a universal (for all graphs at  $\hbar^4$ ) factor  $2^4 = 16$  and the factor  $1/(\# \text{Aut}(L_{322})) = 1$ , it expands to  $\frac{2}{3}([01; 02; 01; 52] + [01; 02; 12; 50] + [01; 02; 20; 51])$ . This sum reproduces  $\mathbf{A}_{322}^{(4)}$  (again, see [7, Table 8 in App. D]).

Example 10. Consider at the differential order (1, 3, 2) at  $\hbar^4$  the three Leibniz graphs  $L_{132}^{(1)} := [12; 13; 012]$ ,  $L_{132}^{(2)} := [12; 12; 014]$ , and  $L_{132}^{(3)} := [12; 01; 412]$ . They have no symmetries, i.e. their automorphism groups are one-element, and their weights are  $W(L_{132}^{(1)}) = 1/72$ ,  $W(L_{132}^{(2)}) = 1/48$ , and  $W(L_{132}^{(3)}) = 1/48$ , respectively. Pre-multiplied by their weights and universal factor  $2^4 = 16$ , these Leibniz graphs expand to

$$\begin{aligned} & \frac{2}{9}([12; 13; 01; 52] + [12; 13; 12; 50] + [12; 13; 20; 51]) \\ & + \frac{1}{3}([12; 12; 01; 54] + [12; 12; 14; 50] + [12; 12; 40; 51]) \\ & + \frac{1}{3}([12; 01; 41; 52] + [12; 01; 12; 54] + [12; 01; 24; 51]). \end{aligned}$$

There is one cancellation, since  $[12; 01; 12; 54] = -[12; 12; 01; 54]$ . The remaining seven terms reproduce exactly  $\mathbf{A}_{132}^{(4)}$ ; that component is known from [7, Table 8 in App. D].



Actually, there was another Leibniz graph at this homogeneity order,  $L_{132}^{(4)} := [12; 15; 012]$ , but its weight is zero and hence it does not contribute. (Indeed, we get an independent verification of this by having already balanced the entire homogeneous component at differential orders (1, 3, 2) in the associator.)

Intermediate conclusion. We have experimentally found the constants  $c_k$  in Corollary 4 which balance the Kontsevich graph expansion of the  $\hbar^k$ -term  $\mathbf{A}^{(k)}$  in the associator against an expansion of the respective term at  $\hbar^k$  in the r.-h.s. of (1) using the weighted Leibniz graphs. Namely, we conjecture  $c_k = k/6$  in §4.3. The origin of these constants, in particular how they arise from the sum over  $i < j$  in the  $L_\infty$  condition (4) (perhaps, in combination with different normalizations of the objects which we consider) still remains to be explained, similar to the reasoning in [2, 18] where the signs are fixed. Note that both in the associator, which is quadratic w.r.t. the weights of Kontsevich graphs in  $\star$ , and in the operator  $\diamond$ , which is linear in the Kontsevich weights of Leibniz graphs, the weight values are provided simultaneously, by using identical techniques (for instance, from [3]). Indeed, the weights are provided by the integral formula which is universal with respect to all the graphs under study [15].

Appendix A. Encodings of weighted Kontsevich-graph expansions for  $(p, q, r)$ -homogeneous components  $(I_f, \dots, S_h)_{pqr}$

- # 2/3 (S\_f)\_\_{221}
- 3 3 1 0 1 0 1 4 2 2/3
- 3 3 1 0 1 1 2 4 0 2/3
- 3 3 1 0 1 2 0 4 1 2/3
- # 2/3 (S\_g)\_\_{122}
- 3 3 1 1 2 0 1 4 2 2/3
- 3 3 1 1 2 1 2 4 0 2/3
- 3 3 1 1 2 2 0 4 1 2/3
- # -2/3 (S\_h)\_\_{212}
- 3 3 1 2 0 0 1 4 2 -2/3
- 3 3 1 2 0 1 2 4 0 -2/3
- 3 3 1 2 0 2 0 4 1 -2/3
- # 1/6 (I\_f)\_\_{111}
- 3 3 1 0 4 3 1 4 2 1/6
- 3 3 1 0 4 1 2 4 3 1/6
- 3 3 1 0 4 2 3 4 1 1/6
- 3 3 1 0 5 3 1 4 2 1/6
- 3 3 1 0 5 1 2 4 3 1/6
- 3 3 1 0 5 2 3 4 1 1/6
- # -1/6 (I\_h)\_\_{111}
- 3 3 1 2 4 0 1 4 3 -1/6
- 3 3 1 2 4 1 3 4 0 -1/6
- 3 3 1 2 4 3 0 4 1 -1/6
- 3 3 1 2 5 0 1 4 3 -1/6
- 3 3 1 2 5 1 3 4 0 -1/6

$3\ 3\ 1\ 2\ 5\ 3\ 0\ 4\ 1\ -1/6$   
 $\# 1/6 (I_f)_{\{112\}}$   
 $3\ 3\ 1\ 0\ 2\ 3\ 1\ 4\ 2\ 1/6$   
 $3\ 3\ 1\ 0\ 2\ 1\ 2\ 4\ 3\ 1/6$   
 $3\ 3\ 1\ 0\ 2\ 2\ 3\ 4\ 1\ 1/6$   
 $\# 1/6 (I_g)_{\{112\}}$   
 $3\ 3\ 1\ 1\ 2\ 0\ 3\ 4\ 2\ 1/6$   
 $3\ 3\ 1\ 1\ 2\ 3\ 2\ 4\ 0\ 1/6$   
 $3\ 3\ 1\ 1\ 2\ 2\ 0\ 4\ 3\ 1/6$   
 $\# -1/3 (S_h)_{\{112\}}$   
 $3\ 3\ 1\ 2\ 4\ 0\ 1\ 4\ 2\ -1/3$   
 $3\ 3\ 1\ 2\ 4\ 1\ 2\ 4\ 0\ -1/3$   
 $3\ 3\ 1\ 2\ 4\ 2\ 0\ 4\ 1\ -1/3$   
 $3\ 3\ 1\ 2\ 5\ 0\ 1\ 4\ 2\ -1/3$   
 $3\ 3\ 1\ 2\ 5\ 1\ 2\ 4\ 0\ -1/3$   
 $3\ 3\ 1\ 2\ 5\ 2\ 0\ 4\ 1\ -1/3$   
 $\# 1/3 (I_f)_{\{121\}}$   
 $3\ 3\ 1\ 0\ 1\ 3\ 1\ 4\ 2\ 1/3$   
 $3\ 3\ 1\ 0\ 1\ 1\ 2\ 4\ 3\ 1/3$   
 $3\ 3\ 1\ 0\ 1\ 2\ 3\ 4\ 1\ 1/3$   
 $\# -1/3 (I_h)_{\{121\}}$   
 $3\ 3\ 1\ 2\ 1\ 0\ 1\ 4\ 3\ -1/3$   
 $3\ 3\ 1\ 2\ 1\ 1\ 3\ 4\ 0\ -1/3$   
 $3\ 3\ 1\ 2\ 1\ 3\ 0\ 4\ 1\ -1/3$   
 $\# 1/3 (S_f)_{\{211\}}$   
 $3\ 3\ 1\ 0\ 4\ 0\ 1\ 4\ 2\ 1/3$   
 $3\ 3\ 1\ 0\ 4\ 1\ 2\ 4\ 0\ 1/3$   
 $3\ 3\ 1\ 0\ 4\ 2\ 0\ 4\ 1\ 1/3$   
 $3\ 3\ 1\ 0\ 5\ 0\ 1\ 4\ 2\ 1/3$   
 $3\ 3\ 1\ 0\ 5\ 1\ 2\ 4\ 0\ 1/3$   
 $3\ 3\ 1\ 0\ 5\ 2\ 0\ 4\ 1\ 1/3$   
 $\# -1/6 (I_g)_{\{211\}}$   
 $3\ 3\ 1\ 1\ 0\ 0\ 3\ 4\ 2\ -1/6$   
 $3\ 3\ 1\ 1\ 0\ 3\ 2\ 4\ 0\ -1/6$   
 $3\ 3\ 1\ 1\ 0\ 2\ 0\ 4\ 3\ -1/6$   
 $\# -1/6 (I_h)_{\{211\}}$   
 $3\ 3\ 1\ 2\ 0\ 0\ 1\ 4\ 3\ -1/6$   
 $3\ 3\ 1\ 2\ 0\ 1\ 3\ 4\ 0\ -1/6$   
 $3\ 3\ 1\ 2\ 0\ 3\ 0\ 4\ 1\ -1/6$

Acknowledgements. The first author thanks the Organisers of international workshop ‘Symmetries & integrability of equations of Mathematical Physics’ (22–24 December 2018, IM NASU Kiev, Ukraine) for helpful discussions and warm atmosphere during the meeting. A part of this research was done while RB was visiting at RUG and AVK was visiting at JGU Mainz (supported by IM JGU via project 5020 and JBI RUG

project 106552). The research of AVK is supported by the IHÉS (partially, by the Nokia Fund).

### References

- [1] Alekseev A., Rossi C. A., Torossian C., Willwacher T. (2016) Logarithms and deformation quantization, *Invent. Math.* 206:1, 1–28. (Preprint arXiv:1401.3200 [q-alg]); Rossi C. A., Willwacher T. (2014) P. Etingof’s conjecture about Drinfeld associators, Preprint arXiv:1404.2047 [q-alg]
- [2] Arnal D., Manchon D., Masmoudi M. (2002) Choix des signes pour la formalité de M. Kontsevich. *Pacific J. Math.* 203:1, 23–66. (Preprint arXiv:q-alg/0003003)
- [3] Banks P., Panzer E., Pym B. (2018) Multiple zeta values in deformation quantization, 71 p. (software available), Preprint arXiv:1812.11649 [q-alg]
- [4] Bouisaghouane A., Buring R., Kiselev A. (2017) The Kontsevich tetrahedral flow revisited, *J. Geom. Phys.* 119, 272–285. (Preprint arXiv:1608.01710 [q-alg])
- [5] Buring R., Kiselev A. V. (2017) On the Kontsevich  $\star$ -product associativity mechanism, *PEPAN Letters 14:2 Supersymmetry and Quantum Symmetries’2015*, 403–407. (Preprint arXiv:1602.09036 [q-alg])
- [6] Buring R., Kiselev A. V. (2019) The orientation morphism: from graph cocycles to deformations of Poisson structures, *J. Phys.: Conf. Ser.* 1194 Proc. 32nd Int. colloquium on Group-theoretical methods in Physics: Group32 (9–13 July 2018, CVUT Prague, Czech Republic), Paper 012017, 10 p. (Preprint arXiv:1811.07878 [math.CO])
- [7] Buring R., Kiselev A. V. (2019) The expansion  $\star \bmod \bar{o}(\hbar^4)$  and computer-assisted proof schemes in the Kontsevich deformation quantization, *Experimental Math.*, 67 p. (revised). (Preprint IHÉS/M/17/05, arXiv:1702.00681 [math.CO])
- [8] Buring R., Kiselev A. V., Rutten N. J. (2018) Poisson brackets symmetry from the pentagon-wheel cocycle in the graph complex, *Physics of Particles and Nuclei 49:5 Supersymmetry and Quantum Symmetries’2017*, 924–928. (Preprint arXiv:1712.05259 [math-ph])
- [9] Cattaneo A. (2005) Formality and star products. (Lect. notes D. Indelicato) *Poisson geometry, deformation quantisation and group representations*. London Math. Soc., Lect. Note Ser. 323, 79–144 (Cambridge Univ. Press, Cambridge).
- [10] Cattaneo A. S., Felder G. (2000) A path integral approach to the Kontsevich quantization formula, *Comm. Math. Phys.* 212:3, 591–611. (Preprint arXiv:q-alg/9902090)
- [11] Cattaneo A., Keller B., Torossian C., Bruguières A. (2005) Déformation, quantification, théorie de Lie. *Panoramas et Synthèses 20*, Soc. Math. de France, Paris.
- [12] Felder G., Willwacher T. (2010) On the (ir)rationality of Kontsevich weights, *Int. Math. Res. Not. IMRN* 2010:4, 701–716. (Preprint arXiv:0808.2762 [q-alg])
- [13] Kontsevich M. (1994) Feynman diagrams and low-dimensional topology, *First Europ. Congr. of Math.* 2 (Paris, 1992), *Progr. Math.* 120, Birkhäuser, Basel, 97–121; Kontsevich M. (1995) Homological algebra of mirror symmetry, *Proc. Intern. Congr. Math.* 1 (Zürich, 1994), Birkhäuser, Basel, 120–139.
- [14] Kontsevich M. (1997) Formality conjecture. *Deformation theory and symplectic geometry* (Ascona 1996, D. Sternheimer, J. Rawnsley and S. Gutt, eds), *Math. Phys. Stud.* 20, Kluwer Acad. Publ., Dordrecht, 139–156.
- [15] Kontsevich M. (2003) Deformation quantization of Poisson manifolds, *Lett. Math. Phys.* 66:3, 157–216. (Preprint arXiv:q-alg/9709040)
- [16] Lada T., Stasheff J. (1993) Introduction to sh Lie algebras for physicists, *Internat. J. Theoret. Phys.* 32:7, 1087–1103. (Preprint arXiv:hep-th/9209099)

- [17] Schlessinger M., Stasheff J. (1985) The Lie algebra structure of tangent cohomology and deformation theory, *J. Pure Appl. Alg.* 38, 313–322.
- [18] Willwacher T., Calaque D. (2012) Formality of cyclic cochains, *Adv. Math.* 231:2, 624–650. (Preprint arXiv:0806.4095 [q-alg])

# Chapter 13

## The heptagon-wheel cocycle in the Kontsevich graph complex

This chapter is based on the peer-reviewed journal publication *R. Buring, A. V. Kiselev, and N. J. Ruten, J. Nonlin. Math. Phys.*, **24**: Suppl. 1 ‘Local & Nonlocal Symmetries in Mathematical Physics’, 157–173, 2017. (Preprint [arXiv:1710.00658](https://arxiv.org/abs/1710.00658) [math.CO] – 17 p.)

*Commentary.* In reference to Part I of the dissertation, the material of this chapter is used in Chapters 4 and 5. The SageMath code (in Appendix B within this chapter) for the graph insertion, bracket of graphs, and vertex-expanding differential served as the beginning of the `gcaops` software. The encodings of undirected graph cocycles  $\gamma_3, \gamma_5, \gamma_7$ , and  $[\gamma_3, \gamma_5]$  are contained in Appendix E of the dissertation.

# THE HEPTAGON-WHEEL COCYCLE IN THE KONTSEVICH GRAPH COMPLEX

RICARDO BURING<sup>(a)</sup>, ARTHEMY KISELEV<sup>(b,c)</sup>, AND NINA RUTTEN<sup>(b)</sup>

*Special Issue JNMP 2017 “Local & nonlocal symmetries in Mathematical Physics”*

ABSTRACT. The real vector space of non-oriented graphs is known to carry a differential graded Lie algebra structure. Cocycles in the Kontsevich graph complex, expressed using formal sums of graphs on  $n$  vertices and  $2n - 2$  edges, induce – under the orientation mapping – infinitesimal symmetries of classical Poisson structures on arbitrary finite-dimensional affine real manifolds. Willwacher has stated the existence of a nontrivial cocycle that contains the  $(2\ell + 1)$ -wheel graph with a nonzero coefficient at every  $\ell \in \mathbb{N}$ . We present detailed calculations of the differential of graphs; for the tetrahedron and pentagon-wheel cocycles, consisting at  $\ell = 1$  and  $\ell = 2$  of one and two graphs respectively, the cocycle condition  $d(\gamma) = 0$  is verified by hand. For the next, heptagon-wheel cocycle (known to exist at  $\ell = 3$ ), we provide an explicit representative: it consists of 46 graphs on 8 vertices and 14 edges.

**Introduction.** The structure of differential graded Lie algebra on the space of non-oriented graphs, as well as the cohomology groups of the graph complex, were introduced by Kontsevich in the context of mirror symmetry [10, 11]. It can be shown that by orienting a graph cocycle on  $n$  vertices and  $2n - 2$  edges (and by adding to every graph in that cocycle two new edges going to two sink vertices) in all such ways that each of the  $n$  old vertices is a tail of exactly two arrows, and by placing a copy of a given Poisson bracket  $\mathcal{P}$  in every such vertex, one obtains an infinitesimal symmetry of the space of Poisson structures. This construction is universal with respect to all finite-dimensional affine real manifolds (see [12] and [2]).<sup>1</sup> Until recently two such differential-polynomial symmetry flows were known (of nonlinearity degrees 4 and 6 respectively). Namely, the tetrahedral graph flow  $\dot{\mathcal{P}} = \mathcal{Q}_{1;\frac{6}{2}}(\mathcal{P})$  was proposed in the seminal paper [12] (see also [2, 3]). Consisting of 91 oriented bi-vector graphs on  $5 + 1 = 6$  vertices, the Kontsevich–Willwacher pentagon-wheel flow will presently be described in [7].

---

*Date:* 24 November 2017.

*2010 Mathematics Subject Classification.* 13D10, 32G81, 53D17, 81S10, also 53D55, 58J10, 90C35.

*Key words and phrases.* Non-oriented graph complex, differential, cocycle, symmetry, Poisson geometry.

<sup>(a)</sup>*Address:* Institut für Mathematik, Johannes Gutenberg–Universität, Staudingerweg 9, D-55128 Mainz, Germany.

<sup>(b)</sup>*Address:* Johann Bernoulli Institute for Mathematics and Computer Science, University of Groningen, P.O. Box 407, 9700 AK Groningen, The Netherlands. <sup>(c)</sup>*E-mail:* A.V.Kiselev@rug.nl.

<sup>1</sup>The dilation  $\dot{\mathcal{P}} = \mathcal{P}$ , also universal with respect to all Poisson manifolds, is obtained by orienting the graph  $\bullet$  on one vertex and no edges, yet that graph is not a cocycle,  $d(\bullet) = -\bullet\bullet \neq 0$ . The single-edge graph  $\bullet\bullet \in \ker d$  on two vertices is a cocycle but its bi-grading differs from  $(n, 2n - 2)$ . However, by satisfying the zero-curvature equation  $d(\bullet\bullet) + \frac{1}{2}[\bullet\bullet, \bullet\bullet] = 0$  the graph  $\bullet\bullet$  is a Maurer–Cartan element in the graph complex.

The cohomology of the graph complex in degree 0 is known to be isomorphic to the Grothendieck–Teichmüller Lie algebra  $\mathbf{grt}$  (see [9] and [16]); under the isomorphism, the  $\mathbf{grt}$  generators correspond to nontrivial cocycles. Using this correspondence, Willwacher gave in [16, Proposition 9.1] the existence proof for an infinite sequence of the Deligne–Drinfel’d nontrivial cocycles on  $n$  vertices and  $2n - 2$  edges. (Formulas which describe these cocycles in terms of the  $\mathbf{grt}$  Lie algebra generators are given in the preprint [15].) To be specific, at each  $\ell \in \mathbb{N}$  every cocycle from that sequence contains the  $(2\ell + 1)$ -wheel with nonzero coefficient (e.g., the tetrahedron alone making the cocycle  $\gamma_3$  at  $\ell = 1$ ), and possibly other graphs on  $2\ell + 2$  vertices and  $4\ell + 2$  edges. For instance, at  $\ell = 2$  the pentagon-wheel cocycle  $\gamma_5$  consists of two graphs, see Fig. 1 on p. 327 below.

In this paper we describe the next one, the heptagon-wheel cocycle  $\gamma_7$  from that sequence of solutions to the equation

$$d\left(\sum_{\{\text{graphs}\}} (\text{coefficient} \in \mathbb{R}) \cdot (\text{graph with an ordering of its edge set})\right) = 0.$$

Our representative of the cocycle  $\gamma_7$  consists of 46 connected graphs on 8 vertices and 14 edges. (This number of nonzero coefficients can be increased by adding a coboundary.) This solution has been obtained straightforwardly, that is, by solving the graph equation  $d(\gamma_7) = 0$  directly. One could try reconstructing the cocycle  $\gamma_7$  from a set of the  $\mathbf{grt}$  Lie algebra generators, which are known in low degrees. Still an explicit verification that  $\gamma_7 \in \ker d$  would be appropriate for that way of reasoning.

In this paper we also confirm that the three cocycles known so far – namely the tetrahedron and pentagon- and heptagon-wheel solutions – span the space of nontrivial cohomology classes which are built of connected graphs on  $n \leq 8$  vertices and  $2n - 2$  edges. At  $n = 9$ , there is a unique nontrivial cohomology class with graphs on nine vertices and sixteen edges: namely, the Lie bracket  $[\gamma_3, \gamma_5]$  of the previously found cocycles. (Brown showed in [4] that the elements  $\sigma_{2\ell+1}$  in the Lie algebra  $\mathbf{grt}$  which – under the Willwacher isomorphism – correspond to the wheel cocycles  $\gamma_{2\ell+1}$  generate a free Lie algebra; hence it was expected that the cocycle  $[\gamma_3, \gamma_5]$  is non-trivial.) To verify that the list of currently known d-cocycles is exhaustive – under all the assumptions which were made about the graphs at our disposal – at every  $n \leq 9$  we count the dimension of the space of cocycles minus the dimension of the space of respective coboundaries.<sup>2</sup> Our findings fully match the dimensions from [14, Table 1].

This text is structured as follows. Necessary definitions and some notation from the graph complex theory are recalled in §1. These notions are illustrated in §2 where a step-by-step calculation of the (vanishing) differentials  $d(\gamma_3)$  and  $d(\gamma_5)$  is explained. Our main result is Theorem 7 with the heptagon-wheel solution of the equation  $d(\gamma_7) = 0$ . Also in §3, in Proposition 8 we verify the count of number of cocycles modulo coboundaries which are formed by all connected graphs on  $n$  vertices and  $2n - 2$  edges (here  $4 \leq n \leq 9$ ). The graphs which constitute  $\gamma_7$  are drawn on pp. 334–340 in Appendix A. The code in SAGE programming language, allowing one to calculate the differential for a given graph  $\gamma$  and ordering  $E(\gamma)$  on the set of its edges, is contained in

---

<sup>2</sup>The proof scheme is computer-assisted (cf. [2, 6]); it can be applied to the study of other cocycles: either on higher number of vertices or built at arbitrary  $n \geq 2$  from not necessarily connected graphs.

Appendix B; the same code can be run to calculate the dimension of graph cohomology groups.

The main purpose of this paper is to provide a pedagogical introduction into the subject.<sup>3</sup> Besides, the formulas of the three cocycle representatives will be helpful in the future search of an easy recipe to calculate all the wheel cocycles  $\gamma_{2\ell+1}$ . (No general recipe is known yet, except for a longer reconstruction of those cohomology group elements from the generators of Lie algebra **grt**.) Thirdly, our present knowledge of both the cocycles  $\gamma_i$  and the respective flows  $\dot{\mathcal{P}} = \mathcal{Q}_i(\mathcal{P})$  on the spaces of Poisson structures will be important for testing and verifying explicit formulas of the orientation mapping  $\text{Or}$  such that  $\mathcal{Q}_i = \text{Or}(\gamma_i)$ .

### 1. THE NON-ORIENTED GRAPH COMPLEX

We work with the real vector space generated by finite non-oriented graphs<sup>4</sup> without multiple edges nor tadpoles and endowed with a wedge ordering of edges: by definition, an edge swap  $e_i \wedge e_j = -e_j \wedge e_i$  implies the change of sign in front of the graph at hand. Topologically equal graphs are equal as vector space elements if their edge orderings  $\mathbf{E}$  differ by an even permutation; otherwise, the graphs are opposite to each other (i.e. they differ by the factor  $-1$ ).

**Definition 1.** A graph which equals minus itself – under a symmetry that induces a parity-odd permutation of edges – is called a *zero graph*. In particular (view  $\bullet\text{---}\bullet\text{---}\bullet$ ), every graph possessing a symmetry which swaps an odd number of edge pairs is a zero graph.

**Notation.** For a given labelling of vertices in a graph, we denote by  $ij$  (equivalently, by  $ji$ ) the edge connecting the vertices  $i$  and  $j$ . For instance, both 12 and 21 is the notation for the edge between the vertices 1 and 2. (No multiple edges are allowed, hence 12 is *the* edge. Indeed, by Definition 1 all graphs with multiple edges would be zero graphs.) We also denote by  $N(v)$  the valency of a vertex  $v$ .

**Example 1.** The 4-wheel  $12 \wedge 13 \wedge 14 \wedge 15 \wedge 23 \wedge 25 \wedge 34 \wedge 45 = I \wedge \dots \wedge VIII$  or likewise, the  $2\ell$ -wheel at any  $\ell > 1$  is a zero graph; here, the reflection symmetry is  $I \rightleftharpoons III, V \rightleftharpoons VII, \text{ and } VI \rightleftharpoons VIII$ .

Note that every term in a sum of non-oriented graphs  $\gamma$  with real coefficients is fully encoded by an ordering  $\mathbf{E}$  on the set of adjacency relations for its vertices  $v$  (if  $N(v) > 0$ ). From now on, we assume  $N(v) \geq 3$  unless stated otherwise explicitly.

**Example 2.** The tetrahedron (or 3-wheel) is the full graph on four vertices and six edges (enumerated in the ascending order:  $12 = I, \dots, 34 = VI$ ),

$$\gamma_3 = 12 \wedge 13 \wedge 14 \wedge 23 \wedge 24 \wedge 34 = I \wedge \dots \wedge VI = \img alt="A tetrahedron graph with vertices labeled 1, 2, 3, and 4. Vertex 1 is at the bottom left, 2 is at the top, 3 is at the bottom right, and 4 is at the center. Edges connect 1-2, 1-3, 1-4, 2-3, 2-4, and 3-4." data-bbox="718 767 789 808"/>$$

This graph is nonzero. (The axis vertex is labelled 4 in this figure.)

<sup>3</sup>The first example of practical calculations of the graph cohomology – with respect to the edge contracting differential – is found in [1]; a wide range of vertex-edge bi-degrees is considered there.

<sup>4</sup>The vector space of graphs under study is infinite dimensional; however, it is endowed with the bi-grading ( $\#$ vertices,  $\#$ edges) so that all the homogeneous components are finite dimensional.



**Example 3.** The linear combination  $\gamma_5$  of two 6-vertex 10-edge graphs, namely, of the pentagon wheel and triangular prism with one extra diagonal (here,  $12 = I$  and so on),

$$\begin{aligned} \gamma_5 = & 12 \wedge 23 \wedge 34 \wedge 45 \wedge 51 \wedge 16 \wedge 26 \wedge 36 \wedge 46 \wedge 56 \\ & + \frac{5}{2} \cdot 12 \wedge 23 \wedge 34 \wedge 41 \wedge 45 \wedge 15 \wedge 56 \wedge 36 \wedge 26 \wedge 13 \end{aligned}$$

is drawn in Fig. 1 on p. 327 below (cf. [1]).

Let  $\gamma_1$  and  $\gamma_2$  be connected non-oriented graphs. The definition of insertion  $\gamma_1 \circ_i \gamma_2$  of the entire graph  $\gamma_1$  into vertices of  $\gamma_2$  and the construction of Lie bracket  $[\cdot, \cdot]$  of graphs and differential  $d$  in the non-oriented graph complex, referring to a sign convention, are as follows (cf. [12] and [8, 14, 16]); these definitions apply to sums of graphs by linearity.

**Definition 2.** The insertion  $\gamma_1 \circ_i \gamma_2$  of an  $n_1$ -vertex graph  $\gamma_1$  with ordered set of edges  $E(\gamma_1)$  into a graph  $\gamma_2$  with  $\#E(\gamma_2)$  edges on  $n_2$  vertices is a sum of graphs on  $n_1 + n_2 - 1$  vertices and  $\#E(\gamma_1) + \#E(\gamma_2)$  edges. Topologically, the sum  $\gamma_1 \circ_i \gamma_2 = \sum(\gamma_1 \rightarrow v \text{ in } \gamma_2)$  consists of all the graphs in which a vertex  $v$  from  $\gamma_2$  is replaced by the entire graph  $\gamma_1$  and the edges touching  $v$  in  $\gamma_2$  are re-attached to the vertices of  $\gamma_1$  in all possible ways.<sup>5</sup> By convention, in every new term the edge ordering is  $E(\gamma_1) \wedge E(\gamma_2)$ .

To simplify sums of graphs, first eliminate the zero graphs. Now suppose that in a sum, two non-oriented graphs, say  $\alpha$  and  $\beta$ , are isomorphic (topologically, i.e. regardless of the respective vertex labellings and edge orderings  $E(\alpha)$  and  $E(\beta)$ ). By using that isomorphism, which establishes a 1–1 correspondence between the edges, extract the sign from the equation  $E(\alpha) = \pm E(\beta)$ . If “+”, then  $\alpha = \beta$ ; else  $\alpha = -\beta$ . Collecting similar terms is now elementary.

**Lemma 1.** The bi-linear graded skew-symmetric operation,

$$[\gamma_1, \gamma_2] = \gamma_1 \circ_i \gamma_2 - (-)^{\#E(\gamma_1) \cdot \#E(\gamma_2)} \gamma_2 \circ_i \gamma_1,$$

is a Lie bracket on the vector space  $\mathfrak{G}$  of non-oriented graphs.<sup>6</sup>

**Lemma 2.** The operator  $d(\text{graph}) = [\bullet\!\!\!\bullet, \text{graph}]$  is a differential:  $d^2 = 0$ .

In effect, the mapping  $d$  blows up every vertex  $v$  in its argument in such a way that whenever the number of adjacent vertices  $N(v) \geq 2$  is sufficient, each end of the inserted edge  $\bullet\!\!\!\bullet$  is connected with the rest of the graph by at least one edge.

**Theorem 3** ([12]). *The real vector space  $\mathfrak{G}$  of non-oriented graphs is a differential graded Lie algebra (dgLa) with Lie bracket  $[\cdot, \cdot]$  and differential  $d = [\bullet\!\!\!\bullet, \cdot]$ . The differential  $d$  is a graded derivation of the bracket  $[\cdot, \cdot]$  (due to the Jacobi identity for this Lie algebra structure).*

<sup>5</sup>Let the enumeration of vertices in every such term in the sum start running over the enumerated vertices in  $\gamma_2$  until  $v$  is reached. Now the enumeration counts the vertices in the graph  $\gamma_1$  and then it resumes with the remaining vertices (if any) that go after  $v$  in  $\gamma_2$ .

<sup>6</sup>The postulated precedence or antecedence of the wedge product of edges from  $\gamma_1$  with respect to the edges from  $\gamma_2$  in every graph within  $\gamma_1 \circ_i \gamma_2$  produce the operations  $\circ_i$  which coincide with or, respectively, differ from Definition 2 by the sign factor  $(-)^{\#E(\gamma_1) \cdot \#E(\gamma_2)}$ . The same applies to the Lie bracket of graphs  $[\gamma_1, \gamma_2]$  if the operation  $\gamma_1 \circ_i \gamma_2$  is the insertion of  $\gamma_2$  into  $\gamma_1$  (as in [14]). Anyway, the notion of d-cocycles which we presently recall is well defined and insensitive to such sign ambiguity.

The graphs  $\gamma_3$  and  $\gamma_5$  from Examples 2 and 3 are d-cocycles (this will be shown in §2). Therefore, their commutator  $[\gamma_3, \gamma_5]$  is also in  $\ker d$ . Neither  $\gamma_3$  nor  $\gamma_5$  is exact, hence marking a nontrivial cohomology class in the non-oriented graph complex.

**Theorem 4** ([8, Th. 5.5]). *At every  $\ell \in \mathbb{N}$  in the connected graph complex there is a nontrivial d-cocycle on  $2\ell + 1$  vertices and  $4\ell + 2$  edges. Such cocycle contains the  $(2\ell + 1)$ -wheel in which, by definition, the axis vertex is connected with every other vertex by a spoke so that each of those  $2\ell$  vertices is adjacent to the axis and two neighbours; the cocycle marked by the  $(2\ell + 1)$ -wheel graph can contain other  $(2\ell + 1, 4\ell + 2)$ -graphs.*

**Example 4.** For  $\ell = 3$  the heptagon wheel cocycle  $\gamma_7$ , which we present in this paper, consists of the heptagon-wheel graph on  $(2 \cdot 3 + 1) + 1 = 8$  vertices and  $2(2 \cdot 3 + 1) = 14$  edges and forty-five other graphs with equally many vertices and edges (hence of the same number of generators of their homotopy groups, or basic loops:  $7 = 14 - (8 - 1)$ ), and with real coefficients. All these weighted graphs are drawn in Appendix A (see pp. 334–340). The chosen –lexicographic– ordering of edges in each term is read from the encoding of every such graph (see also Table 1 on p. 331; each entry of that table is a listing  $I \prec \dots \prec XIV$  of the ordered edge set, followed by the coefficient of that graph). A verification of the cocycle condition  $d(\gamma_7) = 0$  for this solution is computer-assisted; it has been performed by using the code (in SAGE programming language) which is contained in Appendix B.

2. CALCULATING THE DIFFERENTIAL OF GRAPHS

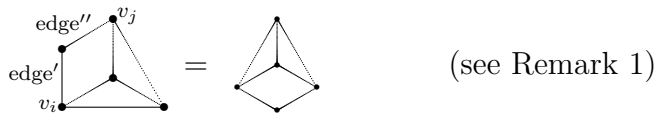
**Example 5** ( $d\gamma_3 = 0$ ). The tetrahedron  $\gamma_3$  is the full graph on  $n = 4$  vertices; we are free to choose any ordering of the six edges in it, so let it be lexicographic:

$$E(\gamma_3) = 12 \wedge 13 \wedge 14 \wedge 23 \wedge 24 \wedge 34 = I \wedge II \wedge III \wedge IV \wedge V \wedge VI.$$

The differential of this graph is equal to

$$d(\gamma_3) = [\bullet\bullet, \gamma_3] = \bullet\bullet \circ_i \gamma_3 - (-)^{\#E(\bullet\bullet) \cdot \#E(\gamma_3)} \gamma_3 \circ_i \bullet\bullet = \bullet\bullet \circ_i \gamma_3 - \gamma_3 \circ_i \bullet\bullet,$$

since  $\#E(\gamma_3) = 6$ . Note that every vertex of valency one appears twice in  $d(\gamma_3)$ : namely in the minuend (where the edge ordering is  $E \wedge I \wedge \dots \wedge VI$  by definition of  $\circ_i$ ) and subtrahend (where the edge ordering is  $I \wedge \dots \wedge VI \wedge E$ ). Because these edge orderings differ by a parity-even permutation, such graphs in  $\bullet\bullet \circ_i \gamma_3$  and  $\gamma_3 \circ_i \bullet\bullet$  carry the same sign. Hence they cancel in the difference  $\bullet\bullet \circ_i \gamma_3 - \gamma_3 \circ_i \bullet\bullet$ , and no longer shall we pay any attention to the leaves, absent in the differential of any graph. It is readily seen that the twenty-four graphs ( $24 = 4$  vertices  $\cdot \binom{3}{1} \cdot 2$  ends of  $\bullet\bullet$ ) we are left with in  $d(\gamma_3)$  are of the shape drawn here. A vertex is blown up to the new edge  $E = \bullet\bullet$



whose ends are both attached to the rest of the graph along the old edges. This shape can be obtained in two ways: by blowing up  $v_i$ , so that edge' is the newly inserted edge, or by blowing up  $v_j$ , so that edge'' is the newly inserted edge. By Lemma 5 below we conclude that  $d(\gamma_3) = 0$ .

*Remark 1.* Incidentally, every graph which was obtained in  $d(\gamma_3)$  itself is a zero graph. Indeed, it is symmetric with respect to a flip over the vertical line and this symmetry swaps three edge pairs (see Definition 1).

**Lemma 5** (handshake). In the differential of any graph  $\gamma$  such that the valency of all vertices in  $\gamma$  is strictly greater than two, the graphs in which one end of the newly inserted edge  $\bullet\bullet$  has valency two, all cancel.

*Proof.* Let  $v$  be such a vertex in  $d(\gamma)$ , i.e. the vertex  $v$  is an end of the inserted edge  $\bullet\bullet$  and it has valency 2. Locally (near  $v$ ), we have either  $a \bullet \xrightarrow{E'} \bullet_v \xrightarrow{\text{Old}'} \bullet_b$  or  $a \bullet \xrightarrow{\text{Old}''} \bullet_v \xrightarrow{E''} \bullet_b$ . In the two respective graphs in  $d(\gamma)$  the rest, consisting only of old edges and vertices of valency  $\geq 3$  from  $\gamma$ , is the same. Yet the two graphs are topologically equal; furthermore, they have the same ordering of edges except for  $E' = \text{Old}''$  and  $\text{Old}' = E''$ . Recall that by construction, the edge ordering of the first graph is  $E' \wedge \cdots \wedge \text{Old}' \wedge \cdots$ , whereas for the second graph it is  $E'' \wedge \cdots \wedge \text{Old}'' \wedge \cdots$ ; the new edge always goes first. So effectively, two edges are swapped. Therefore,

$$E'' \wedge \cdots \wedge \text{Old}'' \wedge \cdots = \text{Old}' \wedge \cdots \wedge E' \wedge \cdots = -E' \wedge \cdots \wedge \text{Old}' \wedge \cdots .$$

Hence in every such pair in  $d(\gamma)$ , the graphs occur with opposite signs. Moreover, the initial hypothesis  $N(a) \geq 3$  about the valency of all vertices  $a$  in the graph  $\gamma$  guarantees that the cancelling pairs of graphs in  $d(\gamma)$  do not intersect,<sup>7</sup> and thus all cancel.  $\square$

**Corollary 6** (to Lemma 5). In the differential of any graph with vertices of valency  $> 2$ , the blow up of a vertex of valency 3 produces only the handshakes, that is the graphs which cancel out by Lemma 5 (cf. footnote 9 on p. 332 below).

**Example 6** ( $d\gamma_5 = 0$ ). The pentagon-wheel cocycle is the sum of two graphs with real coefficients which is drawn in Fig. 1. The edges in every term are ordered by

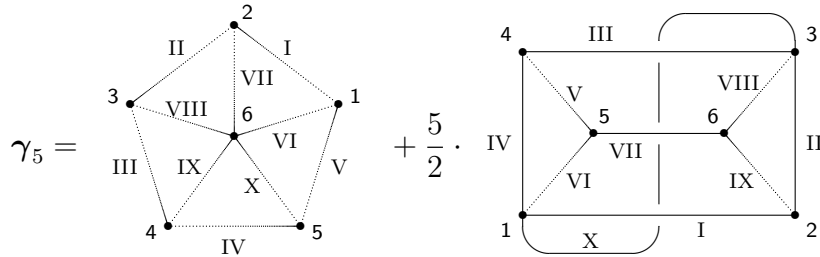


FIGURE 1. The Kontsevich–Willwacher pentagon-wheel cocycle  $\gamma_5$ .

$I \wedge \cdots \wedge X$ . The differential of a sum of graphs is the sum of their differentials; this is why we calculate them separately and then collect similar terms. By the above, neither contains any leaves; likewise by the handshake Lemma 5, all the graphs – in which a new vertex (of valency 2) appears as midpoint of the already existing edge – cancel. By Corollary 6 it remains for us to consider the blow-ups of only the vertices of valency  $\geq 4$  (cf. [12]). Such are the axis vertex of the pentagon wheel and vertices

<sup>7</sup>This is why the assumption  $N(v) \geq 3$  is important. Indeed, the disjoint-pair cancellation mechanism does work only for chains with even numbers of valency-two vertices  $v$  in  $\gamma$ . Here is an example (of one such vertex  $v$  between  $a$  and  $b$ ) when it actually does not: in the differential of a graph that contains  $a \bullet \xrightarrow{I} \bullet_v \xrightarrow{II} \bullet_b$ , we locally obtain  $a \bullet \xrightarrow{E} \bullet_{a'} \xrightarrow{II} \bullet_v \bullet_b + a \bullet \xrightarrow{I} \bullet_v \bullet_{v'} \xrightarrow{II} \bullet_b + a \bullet \xrightarrow{I} \bullet_v \bullet_b \xrightarrow{E} \bullet_{b'}$ , so that the middle term can be cancelled against either the first or the last one but not with both of them simultaneously.

labelled 1 and 3 in the other graph (the prism). By blowing up the pentagon wheel axis we shall obtain the (nonzero) ‘human’ and the (zero) ‘monkey’ graphs, presented in what follows. Likewise from the prism graph in  $\gamma_5$  one obtains the ‘human’, the ‘monkey’, and the (zero) ‘stone’. Let us now discuss this in full detail.

From the pentagon wheel we obtain  $2 \cdot 5$  Da Vinci’s ‘human’ graphs, two of which are portrayed in Fig. 2. (The factor 2 occurs from the two distinct ways to attach three versus two old edges in the wheel to the loose ends of the inserted edge  $\bullet\text{---}\bullet$ .) We claim

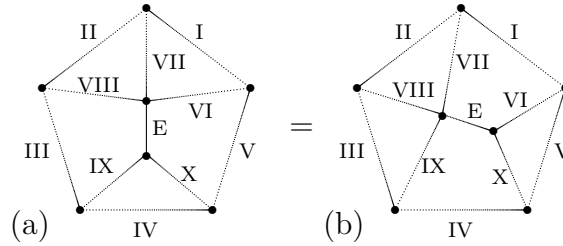


FIGURE 2. Two of the fourteen Da Vinci’s ‘human’ graphs occurring with weights in  $d\gamma_5$ .

that all the five ‘human’ graphs (i.e. standing with their feet on the edges  $I, \dots, V$  in the pentagon wheel) carry the same sign, providing the overall coefficient  $+10 = 2 \cdot (+5)$  of such graph in the differential of the wheel. The graph (b) is topologically equal to the graph (a); indeed, the matching of their edges is  $I^{(b)} = V^{(a)}, II^{(b)} = I^{(a)}, III^{(b)} = II^{(a)}, IV^{(b)} = III^{(a)}, V^{(b)} = IV^{(a)}, VI^{(b)} = X^{(a)}, VII^{(b)} = VI^{(a)}, VIII^{(b)} = VII^{(a)}, IX^{(b)} = VIII^{(a)}$ , and  $X^{(b)} = IX^{(a)}$ ; also  $E^{(b)} = E^{(a)}$ . Hence the postulated ordering of edges in (b) is

$$E^{(b)} \wedge I^{(b)} \wedge \dots \wedge X^{(b)} = E^{(a)} \wedge V^{(a)} \wedge I^{(a)} \wedge II^{(a)} \wedge III^{(a)} \wedge IV^{(a)} \wedge X^{(a)} \wedge VI^{(a)} \wedge VII^{(a)} \wedge VIII^{(a)} \wedge IX^{(a)} = +E^{(a)} \wedge I^{(a)} \wedge \dots \wedge X^{(a)}, \quad (1)$$

which equals the edge ordering of the graph (a). For the other three graphs of this shape the equalities of wedge products are similar: a parity-even permutation of edges works out the mapping of graphs, e.g., to the graph (a) which we take as the reference.

From the pentagon wheel we also obtain  $2 \cdot 5$  ‘monkey’ graphs, a specimen of which is shown in Fig. 3 below. Note that the ‘monkey’ graph is mirror-symmetric, see the

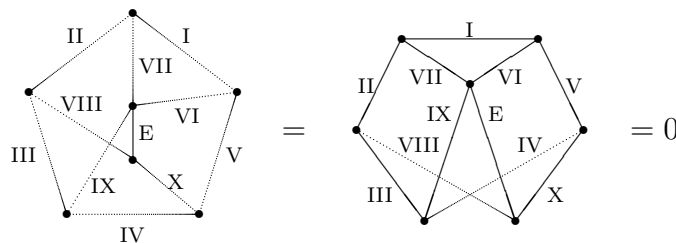


FIGURE 3. The ‘monkey’ graph: animal touches earth with its palm; this is an example of zero graph.

redrawing. This symmetry induces a permutation of edges which swaps 5 pairs, so (since 5 is odd) the ‘monkey’ graph is equal to zero.

Now consider the graphs obtained by blowing up vertices 1 and 3 in the prism graph. How are the four old neighbors distributed over the ends of the inserted edge? Whenever those four old neighbours are distributed in proportion  $4 = 3 + 1$  (i.e. with valencies 4 and 2 for the two ends of the inserted edge), there is no contribution from the resulting graphs to  $d(\text{prism})$  by the handshake Lemma 5. So the graphs which could contribute are only those with the  $4 = 2 + 2$  distribution (i.e. with valency 3 for either of the ends of the inserted edge). For one fixed neighbour of one of the new edge's ends there are three ways to choose the second neighbour of that vertex. This is how the 'human', 'monkey', and 'stone' graphs are presently obtained.

Let us blow up vertex 1 in the prism in these three different ways. First we make the end (now marked 1) of the inserted edge adjacent to 2 and 3, and the other end (marked 1') to vertices 4 and 5; the resulting graph is the 'human' graph shown in Fig. 4. From the prism graph we obtain  $2 \cdot 2 = 4$  such 'human' graphs. One of the factors 2 is

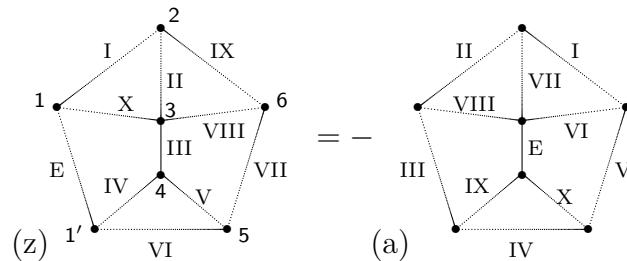
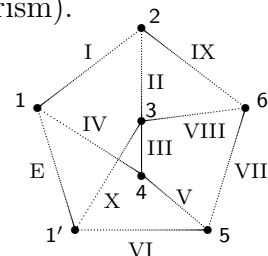


FIGURE 4. One of the 'human' graphs obtained by blowing up – according to a scenario discussed in the text – a vertex of valency four in the prism graph from  $\gamma_5$ .

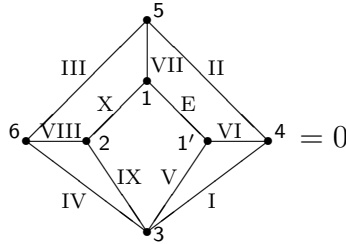
obtained like before, namely by attaching a given set of old edges to one or the other end of the inserted edge  $\bullet\bullet$ , see p. 328; the other factor 2 comes by the rotational symmetry of the prism graph. Indeed, the prism with one diagonal is symmetric under the rotation by angle  $\pi$  that transposes the vertices  $1 \rightleftharpoons 3$ ,  $2 \rightleftharpoons 4$ , and  $5 \rightleftharpoons 6$ . This is why the same 'human' graph is obtained when the vertex 3 is blown up according to a similar scenario. We claim that the permutation of edges that relates the two graphs is parity-even (similar to (1)), so they do not cancel but add up. Summarizing, the overall coefficient of the 'human' graph –produced in  $d(\text{prism})$  for the edge ordering  $E \wedge I \wedge \dots \wedge X$  shown in Fig. 4– equals  $2 \cdot 2 = +4$ .

The count of an overall contribution  $10 + \frac{5}{2} \cdot (+4) \cdot (-1 \text{ from edge ordering}) = 0$  to the differential  $d(\gamma_5)$  of the cocycle  $\gamma_5$  will be performed using Eq. (2); right now let us inspect the vanishing of contributions from the other two types of graphs which are obtained by the two possible edge distribution scenarios (with respect to the ends of the new edge  $\bullet\bullet$  that replaces the blown-up vertex 1 or 3 in the prism).

The 'monkey' graph is obtained by blowing up the vertex 1 (or 3) in the prism and then attaching the new edge's end, still marked 1, to the vertices 2 and 4. The other end, now marked 1', of the new edge becomes adjacent to the vertices 3 and 5. We keep in mind that every 'monkey' graph itself is equal to zero, hence no contribution to  $d(\text{prism})$  occurs.



So far, the new vertex 1 has always been a fixed neighbour of vertex 2, and it was made adjacent to 3 in the ‘human’ and to 4 in the ‘monkey’ graphs, respectively. The overall set of neighbours of the new edge 1–1′, apart from the fixed vertex 2, consists of vertices 3, 4 and 5. So the third scenario to consider is the ‘stone’ graph in which the new vertex 1 is adjacent to 1′, 2, and 5, whereas the new vertex 1′ neighbours 1, 3, and 4. This graph is mirror-symmetric under the transposition of vertices 1′ ⇌ 2



and 4 ⇌ 6, which induces the swaps in five edge pairs, namely, II ⇌ III, E ⇌ X, VI ⇌ VIII, V ⇌ IX, and I ⇌ IV. Arguing as before, we deduce that every such ‘stone’ graph (obtained by a blow up of either 1 or 3 in the prism) is zero.

Our final task in the calculation of  $d(\gamma_5)$  is collecting the coefficients of the ‘human’ graphs from  $d(5\text{-wheel})$  and  $d(\text{prism})$ , coming not only with coefficients 10 and 4 respectively, but also with the respective edge orderings. To discriminate edges between the two pictures, that is originating from the pentagon wheel and the prism, let us use the superscripts  $(a)$  and  $(z)$ , see Fig. 4. The edge matching is  $E^{(z)} = III^{(a)}$ ,  $I^{(z)} = II^{(a)}$ ,  $II^{(z)} = VII^{(a)}$ ,  $III^{(z)} = E^{(a)}$ ,  $IV^{(z)} = IX^{(a)}$ ,  $V^{(z)} = X^{(a)}$ ,  $VI^{(z)} = IV^{(a)}$ ,  $VII^{(z)} = V^{(a)}$ ,  $VIII^{(z)} = VI^{(a)}$ ,  $IX^{(z)} = I^{(a)}$ , and  $X^{(z)} = VIII^{(a)}$ . Consequently, for the edge orderings we have

$$\begin{aligned}
 E^{(z)} \wedge I^{(z)} \wedge \dots \wedge X^{(z)} &= \\
 III^{(a)} \wedge II^{(a)} \wedge VII^{(a)} \wedge E^{(a)} \wedge IX^{(a)} \wedge X^{(a)} \wedge IV^{(a)} \wedge V^{(a)} \wedge VI^{(a)} \wedge I^{(a)} \wedge VIII^{(a)} \\
 &= (-)^{23} E^{(a)} \wedge I^{(a)} \wedge \dots \wedge X^{(a)}. \quad (2)
 \end{aligned}$$

This argument shows that the graph differential of the linear combination  $(+1) \cdot \text{pentagon-wheel} + \frac{5}{2} \cdot \text{prism}$ , with either graph’s edge ordering specified as in Example 3, vanishes. In other words,  $\gamma_5$  is a d-cocycle.

### 3. A REPRESENTATIVE OF THE HEPTAGON-WHEEL COCYCLE $\gamma_7$

It is already known that the heptagon-wheel cocycle  $\gamma_7$ , the existence of which was stated in Theorem 4, is unique modulo d-trivial terms in the respective cohomology group of connected graphs on 8 vertices and 14 edges (hence with 7 basic loops), cf. [14].

**Theorem 7.** *The encoding of every term in a representative of the cocycle  $\gamma_7$  is given in Table 1, the format of lines in which is the lexicographic-ordered list of fourteen edges  $I \wedge \dots \wedge XIV$  followed by the nonzero real coefficient. The forty-six graphs that form this representative of the d-cohomology class  $\gamma_7$  are shown on pages 334–340.*

*Proof scheme.* This reasoning is computer-assisted. First, all connected graphs on 8 vertices and 14 edges, and without multiple edges were generated. (There are 1579 such graphs; note that arbitrary valency  $N(v) \geq 1$  of vertices was allowed.) The coefficient

TABLE 1. The heptagon-wheel graph cocycle  $\gamma_7$ .

Graph encoding	Coeff.	Graph encoding	Coeff.
16 17 18 23 25 28 34 38 46 48 57 58 68 78	1	12 13 18 25 26 37 38 45 46 47 56 57 68 78	-7
12 14 18 23 27 35 37 46 48 57 58 67 68 78	-21/8	12 14 16 23 25 36 37 45 48 57 58 67 68 78	77/8
13 14 18 23 25 28 37 46 48 56 57 67 68 78	-77/4	13 16 17 24 25 26 35 37 45 48 58 67 68 78	-7
12 13 15 24 27 35 36 46 48 57 58 67 68 78	-35/8	14 15 17 23 26 28 37 38 46 48 56 57 68 78	49/4
12 13 18 24 26 37 38 46 47 56 57 58 68 78	49/8	12 16 18 27 28 34 36 38 46 47 56 57 58 78	-147/8
14 17 18 23 25 26 35 37 46 48 56 58 67 78	77/8	12 15 16 27 28 35 36 38 45 46 47 57 68 78	-21/8
12 13 18 26 27 35 38 45 46 47 56 57 68 78	-105/8	12 14 18 23 27 35 36 45 46 57 58 67 68 78	-35/8
12 14 18 23 27 36 38 46 48 56 57 58 67 78	7/8	14 15 16 23 26 28 37 38 46 48 57 58 67 78	-49/4
12 14 15 23 27 35 36 46 48 57 58 67 68 78	35/8	12 15 18 23 28 34 37 46 48 56 57 67 68 78	105/8
12 13 14 27 28 36 38 46 47 56 57 58 68 78	-49/8	12 14 17 23 26 37 38 46 48 56 57 58 68 78	-49/8
12 13 18 25 27 34 36 47 48 56 58 67 68 78	35/4	12 16 18 25 27 35 36 37 45 46 48 57 68 78	49/16
12 13 14 25 26 36 38 45 47 57 58 67 68 78	-119/16	12 13 18 25 27 35 36 46 47 48 56 57 68 78	7
12 13 15 24 28 36 38 47 48 56 57 67 68 78	49/8	12 14 18 25 28 34 36 38 47 57 58 67 68 78	-7
12 13 14 23 28 37 46 48 56 57 58 67 68 78	77/4	12 16 18 25 27 35 36 37 45 46 48 58 67 78	-77/16
12 15 17 25 26 35 36 38 45 47 48 67 68 78	-49/8	12 14 18 23 27 35 38 46 47 57 58 67 68 78	77/4
13 15 18 24 26 28 37 38 46 47 56 57 68 78	-49/4	12 14 15 23 27 36 38 46 48 57 58 67 68 78	35/2
13 14 18 25 26 28 36 38 47 48 56 57 67 78	-49/4	12 13 18 25 27 34 36 46 48 57 58 67 68 78	-105/8
12 14 18 23 28 35 37 46 48 56 57 67 68 78	-7	12 15 16 25 27 35 36 38 46 47 48 57 68 78	-7
12 14 18 23 28 36 38 46 47 56 57 58 67 78	-7	12 13 16 25 28 34 37 47 48 57 58 67 68 78	-147/16
12 15 16 25 27 35 36 38 46 47 48 58 67 78	49/8	12 13 17 25 26 35 37 45 46 48 58 67 68 78	-77/4
12 14 18 23 28 36 37 46 47 56 57 58 68 78	49/8	12 14 17 23 27 35 38 46 48 57 58 67 68 78	-49/8
12 13 15 26 27 35 36 45 47 48 58 67 68 78	-7	12 13 15 26 28 35 37 45 46 47 58 67 68 78	-7/4
12 13 18 24 28 35 38 46 47 57 58 67 68 78	7	12 14 18 23 26 36 38 47 48 56 57 58 67 78	-7

of the heptagon wheel was set equal to +1, all other coefficients still to be determined. After calculating the differential of the sum of all these weighted graphs (we used a program in SAGE, see Appendix B), zero graphs were eliminated and the remaining terms were collected (in the same way as is explained in §2). In the resulting sum of weighted graphs on 9 vertices and 15 edges, we equated each coefficient to zero. We solved this linear algebraic system w.r.t. the coefficients of graphs in  $\gamma_7$ . There are  $N_{\text{im}}(7) = 35$  free parameters in the general solution; such parameters count the coboundaries which cannot modify the cohomology class marked by any particular representative (see Table 2 on p. 332 below). Therefore the solution  $\gamma_7$  is unique modulo d-exact terms. All those free parameters are now set to zero and the resulting nonzero values of the graph coefficients are listed in Table 1.  $\square$

**Proposition 8** (see [14, Table 1]). The space of nontrivial d-cocycles which are built of connected graphs on  $n$  vertices and  $2n - 2$  edges at  $1 \leq n \leq 9$  is spanned by the tetrahedron  $\gamma_3$ , pentagon-wheel cocycle  $\gamma_5$  that consists of two graphs (see Example 3), heptagon-wheel cocycle  $\gamma_7$  from Theorem 7, and the Lie bracket  $[\gamma_3, \gamma_5]$ . At the same time, for either  $n = 5$  or  $n = 7$ , the respective graph cohomology groups are trivial.<sup>8</sup>

*Verification.* The dimension  $N_{\text{ker}}$  of the space of cocycles built of connected graphs  $\gamma$  on  $n$  vertices and  $2n - 2$  edges is equal to the number of free parameters in the general solution to the linear system  $d(\text{sum of such graphs } \gamma \text{ with undetermined coefficients}) = 0$ . At the same time, to determine the dimension  $N_{\text{im}}$  of the subspace of coboundaries  $\gamma = d(\delta)$ , i.e. of those cocycles which are the differentials of connected graphs on  $n - 1$  vertices and  $2n - 3$  edges, we first count the number of  $N_\delta$  of nonzero

<sup>8</sup>None of the results in Theorem 7 and Proposition 8 involves floating point operations in the way how it is obtained; hence even if computer-assisted, both the claims are exact.

connected graphs  $\delta$  in that vertex-edge bi-grading. Then we subtract from  $N_\delta$  the number  $N_0$  of free parameters in the general solution to the linear algebraic system  $d(\text{sums of such graphs } \delta \text{ with undetermined coefficients}) = 0$ . This subtrahend counts the number of relations between exact terms  $\gamma = d(\delta)$ ; for  $n < 9$  it is zero. The dimension of cohomology group  $H^*(n)$  in bi-grading  $(n, 2n - 2)$  is then  $N_{\ker} - N_{\text{im}} = N_{\ker} - (N_\delta - N_0)$ .

Our present count of the overall number of connected graphs (and of the zero graphs among them) and the dimensions  $N_{\ker}, N_\delta, N_0$  and  $N_{\text{im}}$  of the respective vector spaces are summarized in Tables 2 and 3. □

TABLE 2. Dimensions of connected graph spaces and cohomology groups.

$n$	$\#E$	$\#(\text{graphs})$	$\#(= 0)$	$\#(\neq 0), N_\delta$	$N_{\ker}, N_0$	$N_{\text{im}}$	$\dim H^*(n)$
4	6	1	0	1	1		1
	3 5	0	–	–	–	–	
5	8	2	2	0	–	–	0
	4 7	0	–	–	–	–	
6	10	14	8	6	1	–	1
	5 9	1	1	–	0	–	
7	12	126	78	48	1	–	0
	6 11	9	8	–	1	0 1	
8	14	1579	605	974	36	–	1
	7 13	95	60	–	35	0 35	
9	16	26631	7557	19074	883	–	1
	8 15	1515	602	–	913	31 882	

*Remark 2.* This reasoning covers all the connected graphs with specified number of vertices and edges, meaning that the valency  $N(v)$  of every graph vertex  $v$  can be any positive number (if  $n > 1$ ). By Lemma 5 on p. 327 it is seen that for the subspaces  $V_{>2}$  of connected graphs restricted by  $N(v) > 2$  for all  $v$ , the inclusion  $d(V_{>2}) \subseteq V_{>2}$  holds. Therefore, the dimensions of cohomology groups for graphs *with* such restriction on valency cannot exceed the dimension of respective cohomology groups for all the graphs under study (i.e.  $N(v) > 0$ ).<sup>9</sup> This means that trivial cohomology groups remain trivial under the extra assumption  $N(v) > 2$  on valency; yet we already know the generators  $\gamma_3, \gamma_5, \gamma_7$ , and  $[\gamma_3, \gamma_5]$  of all the nontrivial cohomology groups at  $n \leq 9$ . This is confirmed in Table 3.

We finally note that the numbers of nonzero graphs with a specified number of vertices and edges (and  $N(v) > 2$ ), which we list in Table 3, all coincide with the respective entries in Table II in the paper [17].

*Remark 3.* We expect that there are many d-cocycles on  $n$  vertices and  $2n - 2$  edges other than the ones containing the  $(2\ell + 1)$ -wheel graphs (which Theorem 4 provides) or their iterated commutators. Namely, some terms in a weighted sum  $\gamma \in \ker d$  can

<sup>9</sup>Indeed, we recall that these cohomology dimensions – in the count with versus without restriction  $N(v) > 2$  of the valency – are the same (e.g., see [16, Proposition 3.4] with a sketch of the proof).



TABLE 3. Dimensions of connected graph spaces with  $N(v) > 2$  and dimensions of cohomology groups in bi-degree  $(n, 2n - 2)$ .

$n$	$\#E$	$\#(\text{graphs})$	$\#(= 0)$	$\#(\neq 0), N_\delta$	$N_{\ker}, N_0$	$N_{\text{im}}$	$\dim H^*(n)$
4	3	6	1	0	1	1	1
		5	0	–	–	–	–
5	4	8	1	1	0	–	0
		7	0	–	–	–	–
6	5	10	4	2	2	1	1
		9	1	1	–	0	–
7	6	12	18	12	6	1	0
		11	5	4	–	1	0
8	7	14	136	61	75	11	1
		13	30	20	–	10	0
9	8	16	1377	498	879	164	1
		15	309	130	–	179	16

be disjoint graphs; moreover, the vertex-edge bi-grading of a connected component of a given term can be other than  $(m, 2m - 2)$  for  $m \in \mathbb{N}$ . Indeed, for any tuple of d-cocycles  $\gamma_i$  on  $n_i$  vertices and  $E_i$  edges satisfying  $\sum_i n_i = n$  and  $\sum_i E_i = 2n - 2$ , one has that  $\gamma := \bigsqcup_i \gamma_i \in \ker d$ . The graphs  $\gamma_i$  can be restricted by a requirement that each of them belongs to the domain of the orientation mapping  $\text{Or}$ , so that  $\text{Or}(\gamma)$  is a Kontsevich bi-vector graph (see [12] and [2, 7]). In this way new classes of generators of infinitesimal symmetries  $\hat{\mathcal{P}} = \text{Or}(\gamma)(\mathcal{P})$  are obtained for Poisson structures  $\mathcal{P}$ .

**Acknowledgements.** The authors are grateful to M. Kontsevich and T. Willwacher for helpful discussion; the authors thank the referees for criticism and advice. This research was supported in part by JBI RUG project 106552 (Groningen, The Netherlands). A part of this research was done while R. Buring and A. Kiselev were visiting at the IHÉS (Bures-sur-Yvette, France) and A. Kiselev was visiting at the MPIM (Bonn) and Johannes Gutenberg–Universität in Mainz, Germany.

## REFERENCES

- [1] Bar-Natan D., McKay B. D. (2001) Graph cohomology — An overview and some computations, 13 p. (*unpublished*), <http://www.math.toronto.edu/~drorbn/papers/GCOC/GCOC.ps>
- [2] Bouisaghouane A., Buring R., Kiselev A. (2017) The Kontsevich tetrahedral flow revisited, *J. Geom. Phys.* **119**, 272–285. (*Preprint arXiv:1608.01710* [q-alg])
- [3] Bouisaghouane A., Kiselev A. V. (2017) Do the Kontsevich tetrahedral flows preserve or destroy the space of Poisson bi-vectors? *J. Phys.: Conf. Ser.* **804** Proc. XXIV Int. conf. ‘Integrable Systems and Quantum Symmetries’ (14–18 June 2016, ČVUT Prague, Czech Republic), Paper 012008, 10 p. (*Preprint arXiv:1609.06677* [q-alg])
- [4] Brown F. (2012) Mixed Tate motives over  $\mathbb{Z}$ , *Ann. Math. (2)* **175:2**, 949–976.
- [5] Buring R., Kiselev A. V. (2017) On the Kontsevich  $\star$ -product associativity mechanism, *PEPAN Letters* **14:2**, 403–407. (*Preprint arXiv:1602.09036* [q-alg])
- [6] Buring R., Kiselev A. V. (2017) The expansion  $\star \bmod \bar{o}(\hbar^4)$  and computer-assisted proof schemes in the Kontsevich deformation quantization, *Preprint IHÉS/M/17/05*, [arXiv:1702.00681](https://arxiv.org/abs/1702.00681) [math.CO], 67 p.

[7] *Buring R., Kiselev A. V., Rutten N. J.* (2017) The Kontsevich–Willwacher pentagon-wheel symmetry of Poisson structures, SDSP IV (12–16 June 2017, ČVUT Děčín, Czech Republic).

[8] *Dolgushev V. A., Rogers C. L., Willwacher T. H.* (2015) Kontsevich’s graph complex, GRT, and the deformation complex of the sheaf of polyvector fields, *Ann. Math.* **182**:3, 855–943. (*Preprint arXiv:1211.4230* [math.KT])

[9] *Drinfel’d V. G.* (1990) On quasitriangular quasi-Hopf algebras and on a group that is closely connected with  $\text{Gal}(\overline{\mathbb{Q}}/\mathbb{Q})$ , *Algebra i Analiz* **2**:4, 149–181 (in Russian); Eng. transl. in: *Leningrad Math. J.* **2**:4, 829–860 (1990).

[10] *Kontsevich M.* (1994) Feynman diagrams and low-dimensional topology, First Europ. Congr. of Math. **2** (Paris, 1992), Progr. Math. **120**, Birkhäuser, Basel, 97–121.

[11] *Kontsevich M.* (1995) Homological algebra of mirror symmetry, Proc. Intern. Congr. Math. **1** (Zürich, 1994), Birkhäuser, Basel, 120–139.

[12] *Kontsevich M.* (1997) Formality conjecture. Deformation theory and symplectic geometry (Ascona 1996, D. Sternheimer, J. Rawnsley and S. Gutt, eds), Math. Phys. Stud. **20**, Kluwer Acad. Publ., Dordrecht, 139–156.

[13] *Kontsevich M.* (2017) Derived Grothendieck–Teichmüller group and graph complexes [after T. Willwacher], *Séminaire Bourbaki* (69ème année, Janvier 2017), no. 1126, 26 p.

[14] *Khoroshkin A., Willwacher T., Živković M.* (2017) Differentials on graph complexes, *Adv. Math.* **307**, 1184–1214. (*Preprint arXiv:1411.2369* [q-alg])

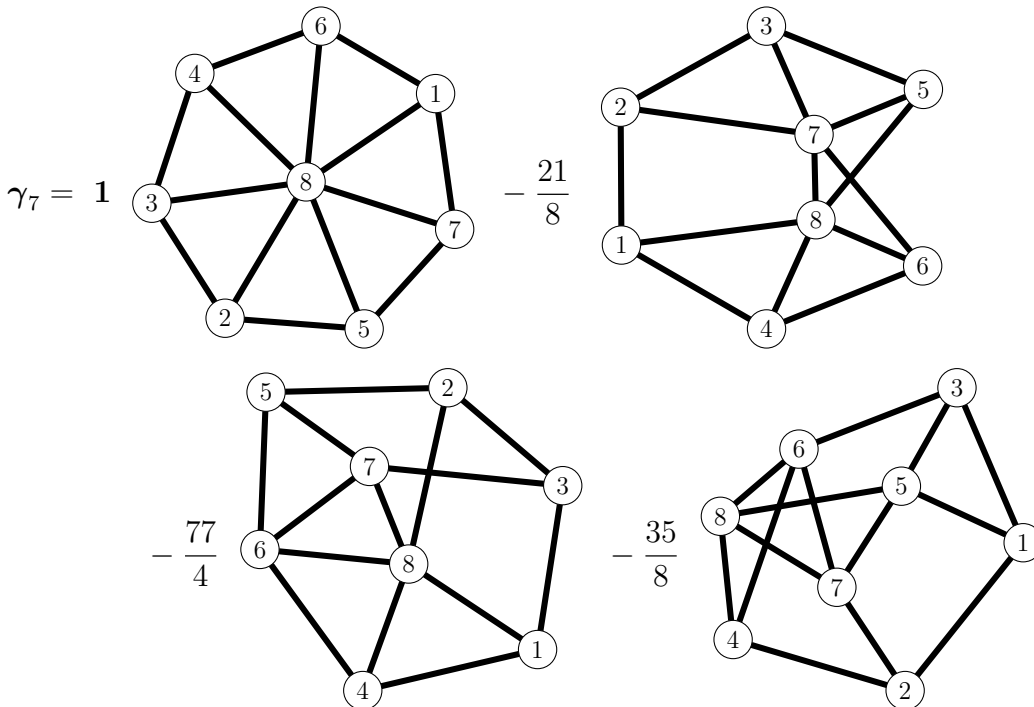
[15] *Rossi C. A., Willwacher T.* (2014) P. Etingof’s conjecture about Drinfeld associators, *Preprint arXiv:1404.2047* [q-alg], 47 p.

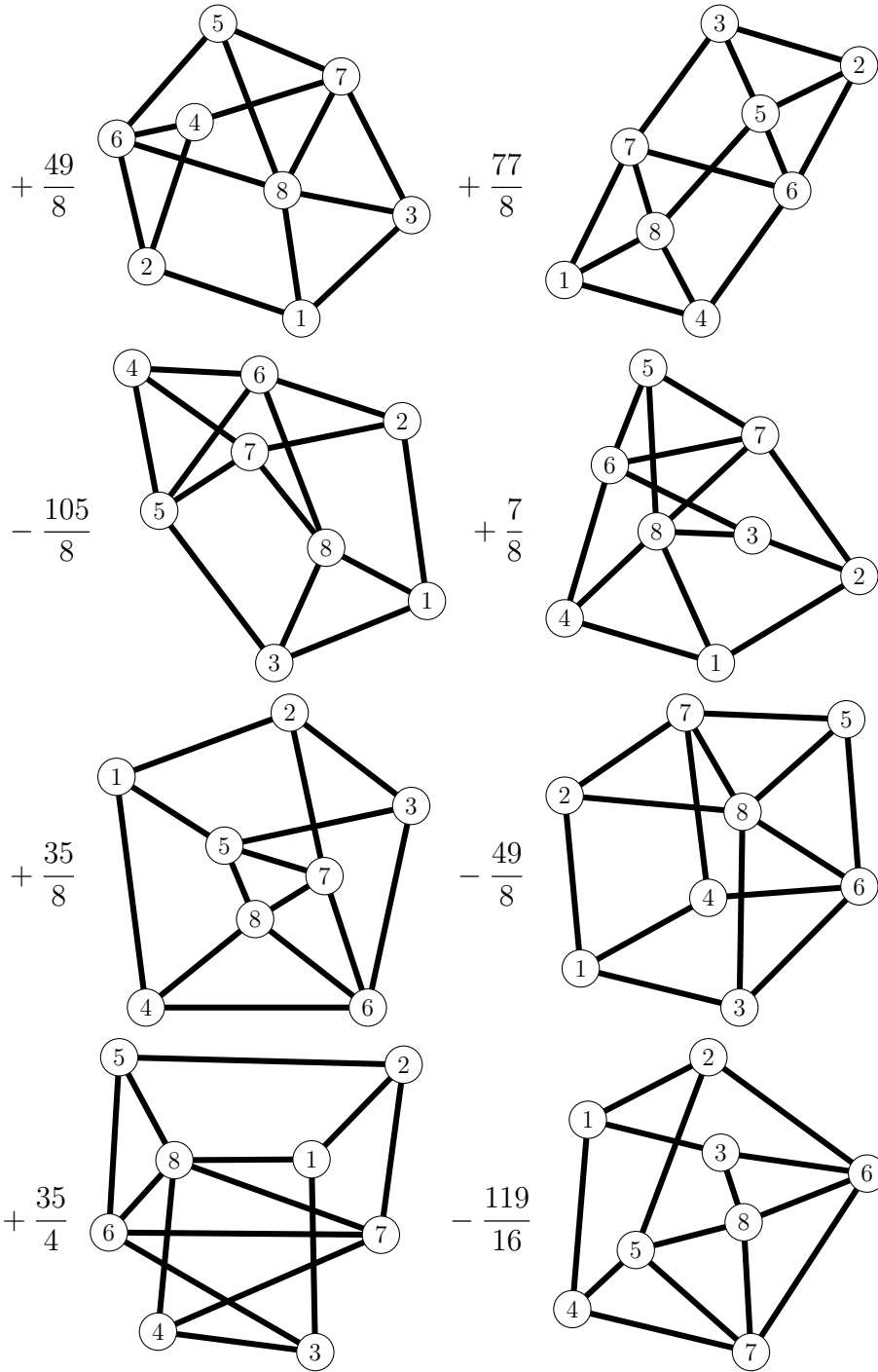
[16] *Willwacher T.* (2015) M. Kontsevich’s graph complex and the Grothendieck–Teichmüller Lie algebra, *Invent. Math.* **200**:3, 671–760. (*Preprint arXiv:1009.1654* [q-alg])

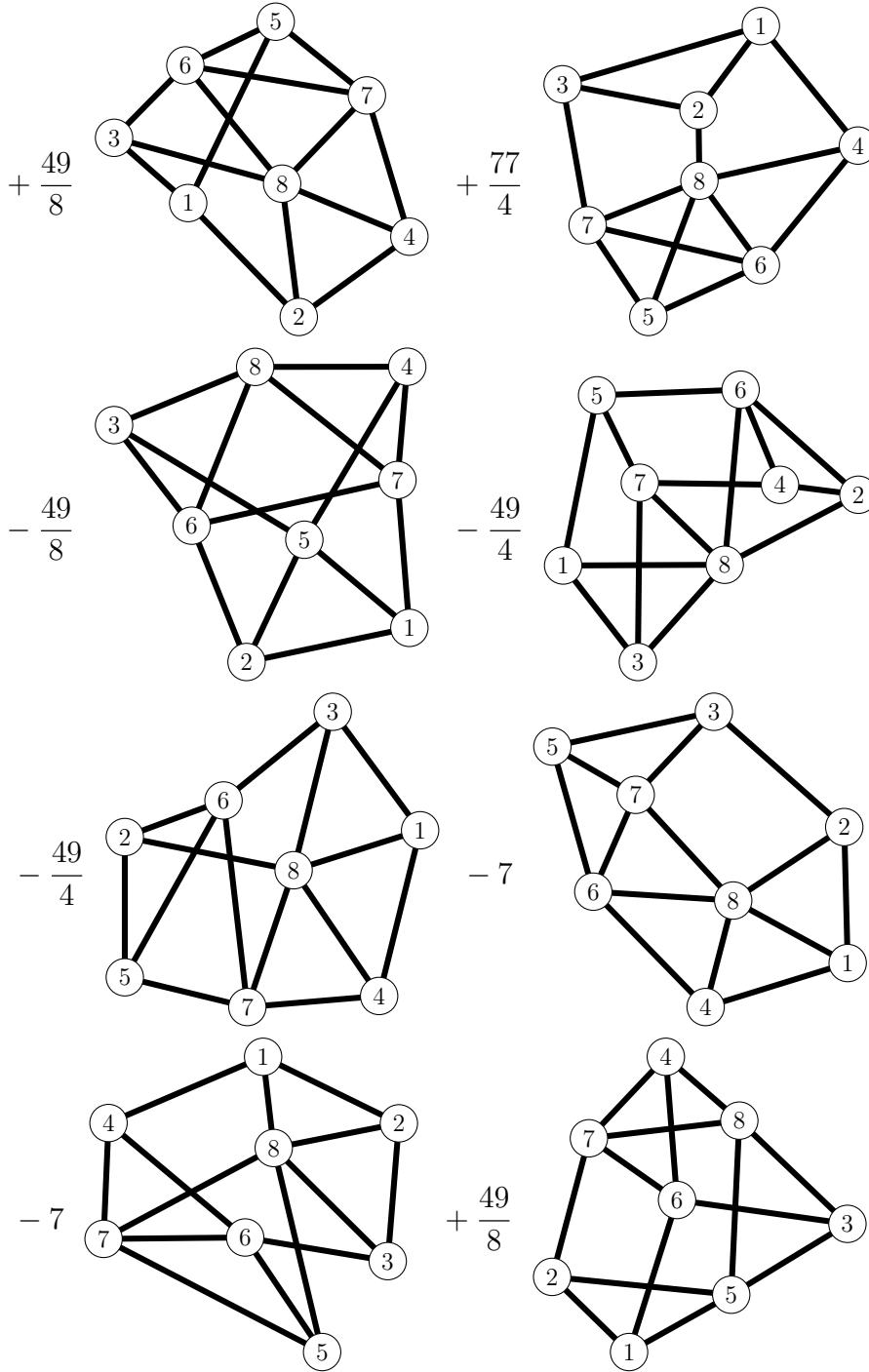
[17] *Willwacher T., Živković M.* (2015) Multiple edges in M. Kontsevich’s graph complexes and computations of the dimensions and Euler characteristics, *Adv. Math.* **272**, 553–578. (*Preprint arXiv:1401.4974* [q-alg])

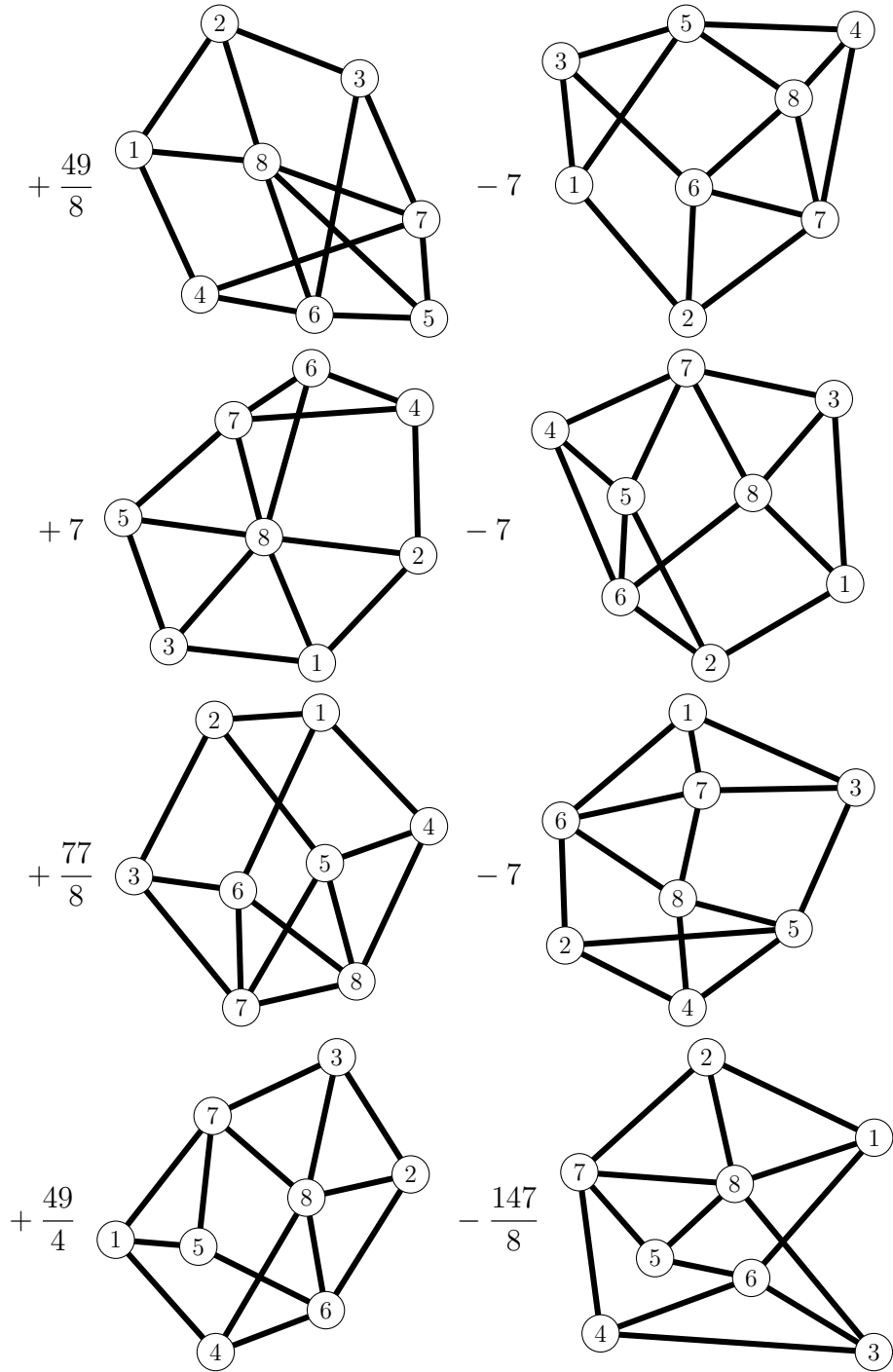
APPENDIX A. THE HEPTAGON-WHEEL COCYCLE  $\gamma_7$

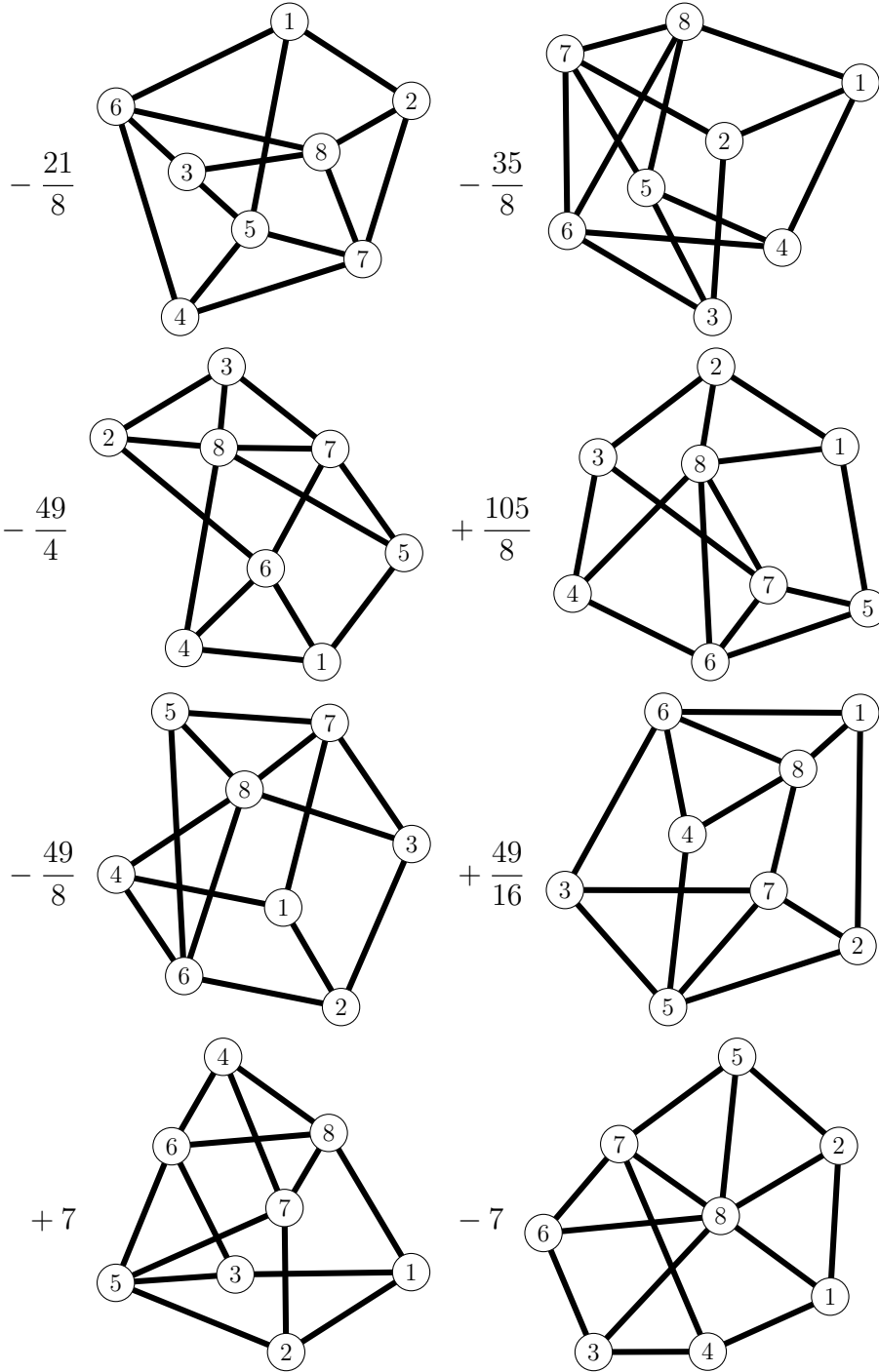
In each term, the ordering of edges is lexicographic (cf. Table 1).

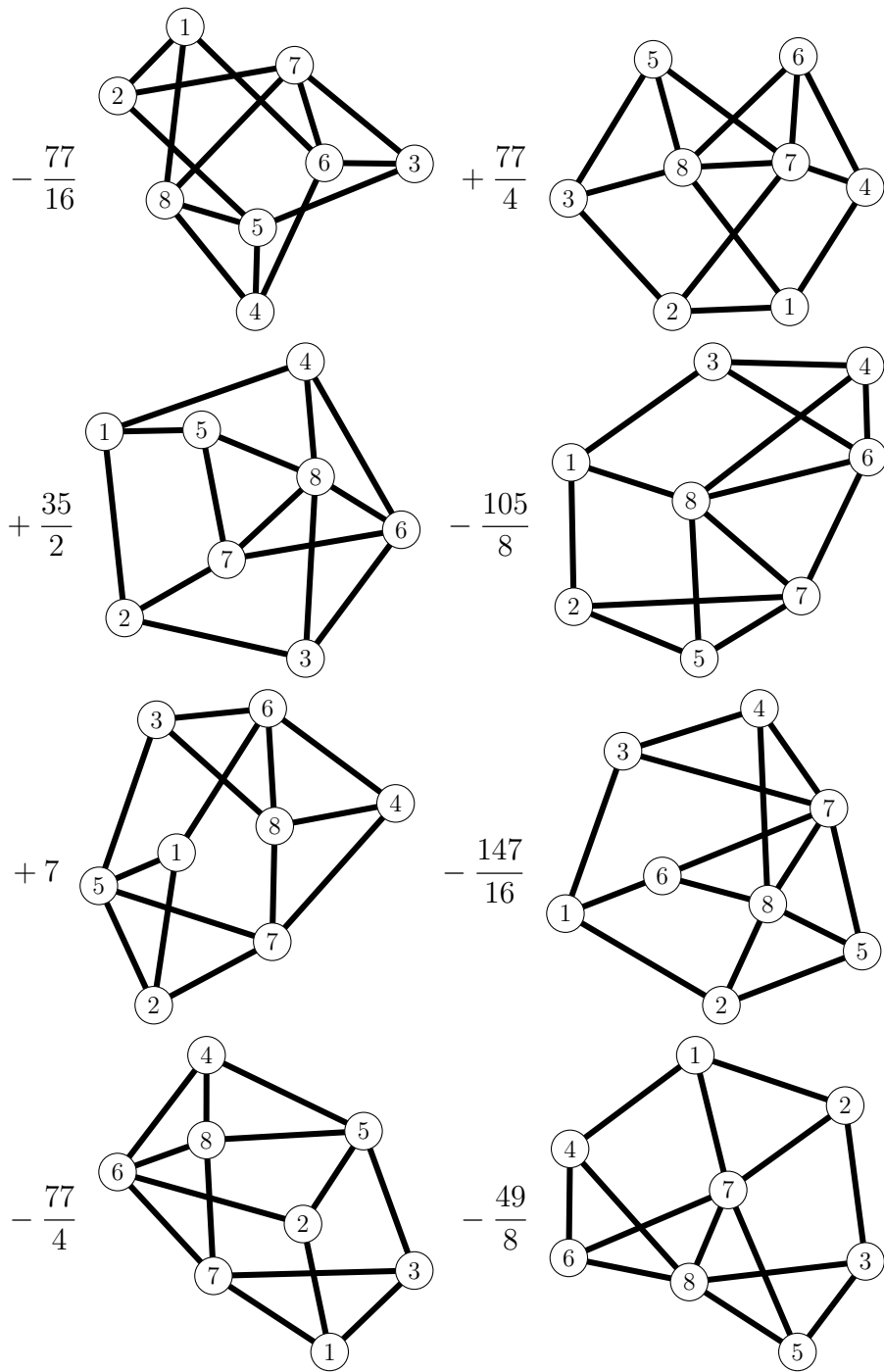


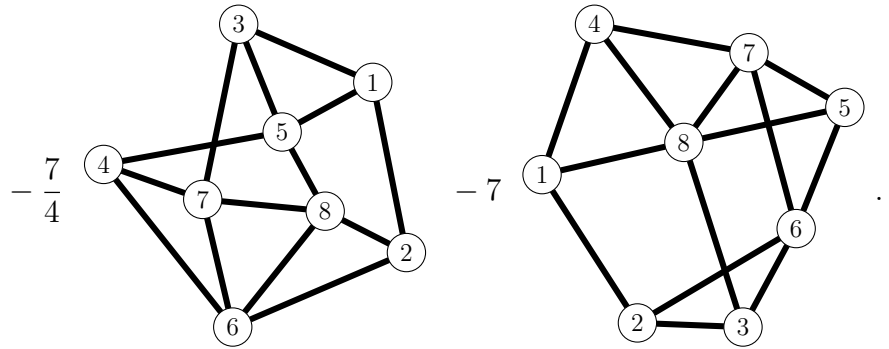












The sum of graphs  $\gamma_7$  is a d-cocycle *because* when the differential  $d(\gamma_7)$  is constructed, the images of many terms from  $\gamma_7$  overlap in  $d(\gamma_7)$  (by graphs on 9 vertices and 15 edges). Finding out what the resulting adjacency table is for the forty-six graphs in  $\gamma_7$  and –more generally– exploring whether such ‘meta-graphs’, the vertices of which themselves are graphs that constitute d-cocycles modulo coboundaries, are in any sense special, is an intriguing open problem. (We claim that for  $\gamma_7$ , its meta-graph is connected.)



APPENDIX B. SAGE CODE FOR THE GRAPH DIFFERENTIAL

The following script, written in SAGE version 7.2, can calculate the differential of an arbitrary sum of non-oriented graphs with a specified ordering on the set of edges for every term, and reduce sums of graphs modulo vertex and edge labelling.<sup>10</sup> As an illustration, it is shown how this can be used to find cocycles in the graph complex.

```
import itertools

def insert(user, victim, position):
    result = []
    victim = victim.relabel({k : k + position - 1 for k in victim.vertices()}),
                        inplace=False)
    victim = victim.copy(immutable=False)
    for edge in victim.edges():
        victim.set_edge_label(edge[0], edge[1], edge[2] + len(user.edges()))
    user = user.relabel({k : k if k <= position else k + len(victim) - 1 for k in user.vertices()}),
                    inplace=False)
    for attachment in itertools.product(victim, repeat=len(user.edges_incident(position))):
        new_graph = user.union(victim)
        edges_in = user.edges_incident(position)
        new_graph.delete_edges(edges_in)
        new_edges = [(k if a == position else a, k if b == position else b, c)
                     for ((a,b,c), k) in zip(edges_in, attachment)]
        new_graph.add_edges(new_edges)
        result.append((1, new_graph))
    return result

def graph_bracket(graph1, graph2):
    result = []
    for v in graph2:
        result.extend(insert(graph2, graph1, v))
    sign_factor = 1 if len(graph1.edges()) % 2 == 1 and len(graph2.edges()) % 2 == 1 else -1
    for v in graph1:
        result.extend([(sign_factor*c, g) for (c,g) in insert(graph1, graph2, v)])
    return result

def graph_differential(graph):
    edge = Graph([(1,2,1)])
    return graph_bracket(edge, graph)

def differential(graph_sum):
    result = []
    for (c,g) in graph_sum:
        result.extend([(c*d,h) for (d,h) in graph_differential(g)])
    return result

def is_zero(graph):
    for sigma in graph.automorphism_group():
        edge_permutation = Permutation([graph.edge_label(sigma(i), sigma(j))
                                         for (i,j,l) in sorted(graph.edges(), key=lambda (a,b,c): c)])
        if edge_permutation.sign() == -1:
            return True
    return False

def reduce(graph_sum):
    graph_table = {}
    for (c,g) in graph_sum:
        if is_zero(g): continue
```

---

<sup>10</sup>Another software package for numeric computation of the graph complex cohomology groups in various degrees and loop orders is available from <https://github.com/wilthoma/GHoL>.

```

# canonically label vertices:
g_canon, relabeling = g.canonical_label(certify=True)
# shift labeling up by one:
g_canon.relabel({k : k + 1 for k in g_canon.vertices()})

# canonically label edges (keeping track of the edge permutation):
count = 1
edges_seen = set([])
edge_relabeling = {}
for v in g_canon:
    edges_in = sorted(g_canon.edges_incident(v), key = lambda (a,b,c): a if b == v else b)
    for e in edges_in:
        if frozenset([e[0], e[1]]) in edges_seen: continue
        edge_relabeling[count] = e[2]
        g_canon.set_edge_label(e[0], e[1], count)
        edges_seen.add(frozenset([e[0], e[1]]))
        count += 1

permutation = Permutation([edge_relabeling[i] for i in range(1, len(g.edges()+1))]
g_canon = g_canon.copy(immutable=True)
if g_canon in graph_table:
    graph_table[g_canon] += permutation.sign()*c
else:
    graph_table[g_canon] = permutation.sign()*c
return [(graph_table[g], g) for g in graph_table if not graph_table[g] == 0]

# Examples of graphs:

def wheel(n):
    return Graph([(k, 1, k-1) for k in range(2, n+2)] + [(k, k+1 if k <= n else 2, n+k-1)
        for k in range(2, n+2)])

tetrahedron = wheel(3)
fivewheel = wheel(5)

print "The differential of the tetrahedron is", reduce(graph_differential(tetrahedron))

# Finding all cocycles on 6 vertices and 10 edges:

n = 6
graph_list = list(filter(lambda G: G.is_connected() and len(G.edges()) == 2*n - 2, graphs(n)))
# shift labeling up by one
for g in graph_list:
    g.relabel({k : k+1 for k in g.vertices()})
    for (k, (i,j,_)) in enumerate(g.edges()):
        g.set_edge_label(i, j, k+1)
# build an ansatz for a cocycle, with undetermined coefficients
nonzeros = filter(lambda g: not is_zero(g), graph_list)
coeffs = [var('c%d' % k) for k in range(0, len(nonzeros))]
cocycle = zip(coeffs, nonzeros)
# calculate its differential and reduce it
d_cocycle = []
for cocycle_term in cocycle:
    d_cocycle.extend(reduce(differential([cocycle_term])))
d_cocycle = reduce(d_cocycle)
# set the coefficients of the graphs in the reduced sum to zero, and solve
linsys = []
for (c,g) in d_cocycle:
    linsys.append(c==0)
print solve(linsys, coeffs)

```

We finally recall that, to the best of our knowledge, the routines by McKay [1] for graph automorphism computation are now used in SAGE (hence by the above program).



## Chapter 14

# Infinitesimal deformations of Poisson bi-vectors using the Kontsevich graph calculus

This chapter is based on the peer-reviewed conference proceedings *R. Buring, A. V. Kiselev, and N. J. Rutten, J. Phys.: Conf. Ser., 965: Proc. XXV Int.conf. ‘Integrable Systems & Quantum Symmetries’ (6–10 June 2017, CVUT Prague, Czech Republic), Paper 012010, 2018. (Preprint arXiv:1710.02405 [math.CO] – 12 p.)*

*Commentary.* In reference to Part I of the dissertation, the material of this chapter is used in §3.5 and Chapter 5. A way to encode Leibniz graphs was originally developed in this chapter; we now refer to the Corrigendum in §3.3 which explains why the old method could produce repetitions of Leibniz graphs.

# Infinitesimal deformations of Poisson bi-vectors using the Kontsevich graph calculus

Ricardo Buring<sup>1</sup>, Arthemy V Kiselev<sup>2</sup> and Nina Rutten<sup>2</sup>

<sup>1</sup> Institut für Mathematik, Johannes Gutenberg–Universität, Staudingerweg 9, D-55128 Mainz, Germany

<sup>2</sup> Johann Bernoulli Institute for Mathematics and Computer Science, University of Groningen, P.O. Box 407, 9700 AK Groningen, The Netherlands

E-mail: [rburing@uni-mainz.de](mailto:rburing@uni-mainz.de), [A.V.Kiselev@rug.nl](mailto:A.V.Kiselev@rug.nl)

**Abstract.** Let  $\mathcal{P}$  be a Poisson structure on a finite-dimensional affine real manifold. Can  $\mathcal{P}$  be deformed in such a way that it stays Poisson? The language of Kontsevich graphs provides a universal approach – with respect to all affine Poisson manifolds – to finding a class of solutions to this deformation problem. For that reasoning, several types of graphs are needed. In this paper we outline the algorithms to generate those graphs. The graphs that encode deformations are classified by the number of internal vertices  $k$ ; for  $k \leq 4$  we present all solutions of the deformation problem. For  $k \geq 5$ , first reproducing the pentagon-wheel picture suggested at  $k = 6$  by Kontsevich and Willwacher, we construct the heptagon-wheel cocycle that yields a new unique solution without 2-loops and tadpoles at  $k = 8$ .

**Introduction.** This paper contains a set of algorithms to generate the Kontsevich graphs that encode polydifferential operators – in particular multi-vectors – on Poisson manifolds. We report a result of implementing such algorithms in the problem of finding symmetries of Poisson structures. Namely, continuing the line of reasoning from [1, 2], we find all the solutions of this deformation problem that are expressed by the Kontsevich graphs with at most four internal vertices. Next, we present one six-vertex solution (based on the previous work by Kontsevich [10] and Willwacher [13]). Finally, we find a heptagon-wheel eight-vertex graph which, after the orientation of its edges, gives a new universal Kontsevich flow. We refer to [8, 9] for motivations, to [2, 4] for an exposition of basic theory, and to [6] and [5] for more details about the pentagon-wheel (5+1)-vertex and heptagon-wheel (7+1)-vertex solutions respectively. Let us remark that all the algorithms outlined here can be used without modification in the course of constructing all  $k$ -vertex Kontsevich graph solutions with higher  $k \geq 5$  in the deformation problem under study.

**Basic concept.** We work with real vector spaces generated by finite graphs of the following two types: (1)  $k$ -vertex non-oriented graphs, without multiple edges nor tadpoles, endowed with a wedge ordering of edges, e.g.,  $E = e_1 \wedge \cdots \wedge e_{2k-2}$ ; (2) oriented graphs on  $k$  internal vertices and  $n$  sinks such that every internal vertex is a tail of two edges with a given ordering Left  $\prec$  Right. Every connected component of a non-oriented graph  $\gamma$  is fully encoded by an ordering  $E$

on the set of adjacency relations for its vertices.<sup>1</sup> Every such oriented graph is given by the list of ordered pairs of directed edges. An edge swap  $e_i \wedge e_j = -e_j \wedge e_i$  and the reversal Left  $\leftrightarrow$  Right of those edges' order in the tail vertex implies the change of sign in front of the graph at hand.<sup>2</sup>

**Example 1.** The sum  $\gamma_5$  of two 6-vertex 10-edge graphs,

$$\begin{aligned} \gamma_5 = & 12^{(I)} \wedge 23^{(II)} \wedge 34^{(III)} \wedge 45^{(IV)} \wedge 51^{(V)} \wedge 16^{(VI)} \wedge 26^{(VII)} \wedge 36^{(VIII)} \wedge 46^{(IX)} \wedge 56^{(X)} \\ & + \frac{5}{2} \cdot 12^{(I)} \wedge 23^{(II)} \wedge 34^{(III)} \wedge 41^{(IV)} \wedge 45^{(V)} \wedge 15^{(VI)} \wedge 56^{(VII)} \wedge 36^{(VIII)} \wedge 26^{(IX)} \wedge 13^{(X)}, \end{aligned}$$

is drawn in Theorem 7 on p. 355 below.

**Example 2.** The sum  $\mathcal{Q}_{1;\frac{6}{2}}$  of three oriented 8-edge graphs on  $k = 4$  internal vertices and  $n = 2$  sinks (enumerated using 0 and 1, see the notation on p. 349),

$$\mathcal{Q}_{1;\frac{6}{2}} = 2\ 4\ 1\ 0\ 1\ 2\ 4\ 2\ 5\ 2\ 3 - 3(2\ 4\ 1\ 0\ 3\ 1\ 4\ 2\ 5\ 2\ 3 + 2\ 4\ 1\ 0\ 3\ 4\ 5\ 1\ 2\ 2\ 4)$$

is obtained from the non-oriented tetrahedron graph  $\gamma_3 = 12^{(I)} \wedge 13^{(II)} \wedge 14^{(III)} \wedge 23^{(IV)} \wedge 24^{(V)} \wedge 34^{(VI)}$  on four vertices and six edges by taking all the admissible edge orientations (see Theorem 4 and Remark 1).

**I.1.** Let  $\gamma_1$  and  $\gamma_2$  be connected non-oriented graphs. The definition of insertion  $\gamma_1 \circ_i \gamma_2$  of the entire graph  $\gamma_1$  into vertices of  $\gamma_2$  and the construction of Lie bracket  $[\cdot, \cdot]$  of graphs and differential  $d$  in the non-oriented graph complex, referring to a sign convention, are as follows (cf. [8] and [7, 11, 12]); these definitions apply to sums of graphs by linearity.

**Definition 1.** The insertion  $\gamma_1 \circ_i \gamma_2$  of a  $k_1$ -vertex graph  $\gamma_1$  with ordered set of edges  $E(\gamma_1)$  into a graph  $\gamma_2$  with  $\#E(\gamma_2)$  edges on  $k_2$  vertices is a sum of graphs on  $k_1 + k_2 - 1$  vertices and  $\#E(\gamma_1) + \#E(\gamma_2)$  edges. Topologically, the sum  $\gamma_1 \circ_i \gamma_2 = \sum(\gamma_1 \rightarrow v \text{ in } \gamma_2)$  consists of all the graphs in which a vertex  $v$  from  $\gamma_2$  is replaced by the entire graph  $\gamma_1$  and the edges touching  $v$  in  $\gamma_2$  are re-attached to the vertices of  $\gamma_1$  in all possible ways.<sup>3</sup> By convention, in every new term the edge ordering is  $E(\gamma_1) \wedge E(\gamma_2)$ .

To simplify sums of graphs, first eliminate the zero graphs. Now suppose that in a sum, two non-oriented graphs, say  $\alpha$  and  $\beta$ , are isomorphic (topologically, i.e. regardless of the respective vertex labellings and edge orderings  $E(\alpha)$  and  $E(\beta)$ ). By using that isomorphism, which establishes a 1-1 correspondence between the edges, extract the sign from the equation  $E(\alpha) = \pm E(\beta)$ . If “+”, then  $\alpha = \beta$ ; else  $\alpha = -\beta$ . Collecting similar terms is now elementary.

**Lemma 1.** The bi-linear graded skew-symmetric operation,

$$[\gamma_1, \gamma_2] = \gamma_1 \circ_i \gamma_2 - (-)^{\#E(\gamma_1) \cdot \#E(\gamma_2)} \gamma_2 \circ_i \gamma_1,$$

<sup>1</sup> The edges are antipermutable so that a graph which equals minus itself –under a symmetry that induces a parity-odd permutation of edges– is proclaimed to be equal to zero. In particular (view  $\bullet\text{---}\bullet\text{---}\bullet$ ), every graph possessing a symmetry which swaps an odd number of edge pairs is a zero graph. For example, the 4-wheel  $12 \wedge 13 \wedge 14 \wedge 15 \wedge 23 \wedge 25 \wedge 34 \wedge 45 = I \wedge \dots \wedge VIII$  or the  $2\ell$ -wheel at any  $\ell > 1$  is such; here, the reflection symmetry is  $I \rightleftharpoons III$ ,  $V \rightleftharpoons VII$ , and  $VI \rightleftharpoons VIII$ .

<sup>2</sup> An oriented graph equals minus itself, hence it is a zero graph if there is a permutation of labels for its internal vertices such that the adjacency tables for the two vertex labellings coincide but the two realisations of the same graph differ by the ordering of outgoing edges at an odd number of internal vertices (see Example 3 below).

<sup>3</sup> Let the enumeration of vertices in every such term in the sum start running over the enumerated vertices in  $\gamma_2$  until  $v$  is reached. Now the enumeration counts the vertices in the graph  $\gamma_1$  and then it resumes with the remaining vertices (if any) that go after  $v$  in  $\gamma_2$ .

is a Lie bracket on the vector space  $\mathfrak{G}$  of non-oriented graphs.<sup>4</sup>

**Lemma 2.** The operator  $d(\text{graph}) = [\bullet\!\!\!\bullet, \text{graph}]$  is a differential:  $d^2 = 0$ .

In effect, the mapping  $d$  blows up every vertex  $v$  in its argument in such a way that whenever the number of adjacent vertices  $N(v) \geq 2$  is sufficient, each end of the inserted edge  $\bullet\!\!\!\bullet$  is connected with the rest of the graph by at least one edge.

Summarising, the real vector space  $\mathfrak{G}$  of non-oriented graphs is a differential graded Lie algebra (dgLa) with Lie bracket  $[\cdot, \cdot]$  and differential  $d = [\bullet\!\!\!\bullet, \cdot]$ . The graphs  $\gamma_5$  and  $\gamma_3$  from Examples 1 and 2 are  $d$ -cocycles. Neither is exact, hence marking a nontrivial cohomology class in the non-oriented graph complex.

**Theorem 3** ([7, Th. 5.5]). *At every  $\ell \in \mathbb{N}$  in the connected graph complex there is a  $d$ -cocycle on  $2\ell + 1$  vertices and  $4\ell + 2$  edges. Such cocycle contains the  $(2\ell + 1)$ -wheel in which, by definition, the axis vertex is connected with every other vertex by a spoke so that each of those  $2\ell$  vertices is adjacent to the axis and two neighbours; the cocycle marked by the  $(2\ell + 1)$ -wheel graph can contain other  $(2\ell + 1, 4\ell + 2)$ -graphs (see Example 1 and [5]).*

**I.2.** The oriented graphs under study are built over  $n$  sinks from  $k$  wedges  $\overleftarrow{i_\alpha} \bullet \overrightarrow{j_\alpha}$  (here  $\overleftarrow{i_\alpha} \overleftarrow{j_\alpha}$ ) so that every edge is decorated with its own summation index which runs from 1 to the dimension of a given affine Poisson manifold  $(\mathcal{N}, \mathcal{P})$ . Each edge  $\overrightarrow{i}$  encodes the derivation  $\partial/\partial x^i$  of the arrowhead object with respect to a local coordinate  $x^i$  on  $\mathcal{N}$ . By placing an  $\alpha$ th copy  $P^{i_\alpha j_\alpha}(\mathbf{x})$  of the Poisson bi-vector  $\mathcal{P}$  in the wedge top ( $1 \leq \alpha \leq k$ ), by taking the product of contents of the  $n + k$  vertices (and evaluating all objects at a point  $\mathbf{x} \in \mathcal{N}$ ), and summing over all indices, we realise a polydifferential operator in  $n$  arguments; the operator coefficients are differential-polynomial in  $\mathcal{P}$ . Totally skew-symmetric operators of differential order one in each argument are well-defined  $n$ -vectors on the affine manifolds  $\mathcal{N}$ .

The space of multi-vectors  $G$  encoded by oriented graphs is equipped with a graded Lie algebra structure, namely the Schouten bracket  $[[\cdot, \cdot]]$ . Its realisation in terms of oriented graphs is shown in [2, Remark 4]. Recall that by definition the bi-vectors  $\mathcal{P}$  at hand are Poisson by satisfying the Jacobi identity  $[[\mathcal{P}, \mathcal{P}]] = 0$ . The Poisson differential  $\partial_{\mathcal{P}} = [[\mathcal{P}, \cdot]]$  now endows the space of multi-vectors on  $\mathcal{N}$  with the differential graded Lie algebra (dgLa) structure. The cohomology groups produced by the two dgLa structures introduced so far are correlated by the edge orientation mapping  $\text{Or}$ .

**Theorem 4** ([8] and [12, App. K]). *Let  $\gamma \in \ker d$  be a cocycle on  $k$  vertices and  $2k - 2$  edges in the non-oriented graph complex. Denote by  $\{\Gamma\} \subset G$  the subspace spanned by all those bi-vector graphs  $\Gamma$  which are obtained from (each connected component in)  $\gamma$  by adding to it two edges to the new sink vertices and then by taking the sum of graphs with all the admissible orientations of the old  $2k - 2$  edges (so that a set of Kontsevich graphs built of  $k$  wedges is produced). Then in that subspace  $\{\Gamma\}$  there is a sum of graphs that encodes a nonzero Poisson cocycle  $Q(\mathcal{P}) \in \ker \partial_{\mathcal{P}}$ .*

Consequently, to find some cocycle  $Q(\mathcal{P})$  in the Poisson complex on any affine Poisson manifold it suffices to find a cocycle in the non-oriented graph complex and then consider the sum of graphs which are produced by the orientation mapping  $\text{Or}$ . On the other hand, to list all the  $\partial_{\mathcal{P}}$ -cocycles  $Q(\mathcal{P})$  encoded by the bi-vector graphs made of  $k$  wedges  $\overleftarrow{\bullet} \bullet \overrightarrow{\bullet}$ , one must generate all the relevant oriented graphs and solve the equation  $\partial_{\mathcal{P}}(Q) \doteq 0$  via  $[[\mathcal{P}, \mathcal{P}]] = 0$ , that is, solve

<sup>4</sup> The postulated precedence or antecedence of the wedge product of edges from  $\gamma_1$  with respect to the edges from  $\gamma_2$  in every graph within  $\gamma_1 \circ_i \gamma_2$  produce the operations  $\circ_i$  which coincide with  $\text{or}$ , respectively, differ from Definition 1 by the sign factor  $(-)^{\#E(\gamma_1) \cdot \#E(\gamma_2)}$ . The same applies to the Lie bracket of graphs  $[\gamma_1, \gamma_2]$  if the operation  $\gamma_1 \circ_i \gamma_2$  is the insertion of  $\gamma_2$  into  $\gamma_1$  (as in [11]). Anyway, the notion of  $d$ -cocycles which we presently recall is well defined and insensitive to such sign ambiguity.



graphically the factorisation problem  $[[\mathcal{P}, Q(\mathcal{P})]] = \diamond(\mathcal{P}, [[\mathcal{P}, \mathcal{P}]])$  in which the cocycle condition in the left-hand side holds by virtue of the Jacobi identity in the right. Such construction of some and classification (at a fixed  $k > 0$ ) of all universal infinitesimal symmetries of Poisson brackets are the problems which we explore in this paper.

*Remark 1.* To the best of our knowledge [10], in a bi-vector graph  $Q(\mathcal{P}) = \text{Or}\vec{\gamma}$ , at every internal vertex which is the tail of two oriented edges towards other internal vertices, the edge ordering  $\text{Left} \prec \text{Right}$  is inherited from a chosen wedge product  $E(\gamma)$  of edges in the non-oriented graph  $\gamma$ . How are the new edges towards the sinks ordered, either between themselves at a vertex or with respect to two other oriented edges, coming from  $\gamma$  and issued from different vertices in  $Q(\mathcal{P})$ ? Our findings in [6] will help us to verify the order preservation claim and assess answers to this question.

### 1. The Kontsevich graph calculus

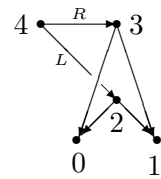
**Definition 2.** Let us consider a class of oriented graphs on  $n+k$  vertices labelled  $0, \dots, n+k-1$  such that the consecutively ordered vertices  $0, \dots, n-1$  are sinks, and each of the internal vertices  $n, \dots, n+k-1$  is a source for two edges. For every internal vertex, the two outgoing edges are ordered using  $L \prec R$ : the preceding edge is labelled  $L$  (Left) and the other is  $R$  (Right). An oriented graph on  $n$  sinks and  $k$  internal vertices is a *Kontsevich graph* of type  $(n, k)$ .

For the purpose of defining a graph normal form, we now consider a Kontsevich graph  $\Gamma$  together with a *sign*  $s \in \{0, \pm 1\}$ , denoted by concatenation of the symbols:  $s\Gamma$ .

**Notation** (Encoding of the Kontsevich graphs). The format to store a signed graph  $s\Gamma$  for a Kontsevich graph  $\Gamma$  is the integer number  $n > 0$ , the integer  $k \geq 0$ , the sign  $s$ , followed by the (possibly empty, when  $k = 0$ ) list of  $k$  ordered pairs of targets for edges issued from the internal vertices  $n, \dots, n+k-1$ , respectively. The full format is then  $(n, k, s; \text{list of ordered pairs})$ .

**Definition 3** (Normal form of a Kontsevich graph). The list of targets in the encoding of a graph  $\Gamma$  can be considered as a  $2k$ -digit integer written in base- $(n+k)$  notation. By running over the entire group  $S_k \times (\mathbb{Z}_2)^k$ , and by this over all the different re-labellings of  $\Gamma$ , we obtain many different integers written in base- $(n+k)$ . The *absolute value*  $|\Gamma|$  of  $\Gamma$  is the re-labelling of  $\Gamma$  such that its list of targets is *minimal* as a nonnegative base- $(n+k)$  integer. For a signed graph  $s\Gamma$ , the *normal form* is the signed graph  $t|\Gamma|$  which represents the same polydifferential operator as  $s\Gamma$ . Here we let  $t = 0$  if the graph is zero (see Example 3 below).

**Example 3** (Zero Kontsevich graph). Consider the graph with the encoding  $2\ 3\ 1\ 0\ 1\ 0\ 1\ 2\ 3$ . The swap of vertices  $2 \rightleftharpoons 3$  is a symmetry of this graph, yet it also swaps the ordered edges  $(4 \rightarrow 2) \prec (4 \rightarrow 3)$ , producing a minus sign. Equal to minus itself, this Kontsevich graph is zero.



**Notation.** Every Kontsevich graph  $\Gamma$  on  $n$  sinks (or every sum  $\Gamma$  of such graphs) yields the sum  $\text{Alt } \Gamma$  of Kontsevich graphs which is totally skew-symmetric with respect to the  $n$  sinks content  $s_1, \dots, s_n$ . Indeed, let

$$(\text{Alt } \Gamma)(s_1, \dots, s_n) = \sum_{\sigma \in \mathcal{S}_n} (-)^{\sigma} \Gamma(s_{\sigma(1)}, \dots, s_{\sigma(n)}). \quad (1)$$

Due to skew-symmetrisation, the sum of graphs  $\text{Alt } \Gamma$  can contain zero graphs or repetitions.

**Example 4** (The Jacobiator). The left-hand side of the Jacobi identity is a skew sum of Kontsevich graphs (e.g. it is obtained by skew-symmetrizing the first term)

$$\begin{array}{c} \boxed{\bullet \bullet} \\ \downarrow \quad \downarrow \quad \downarrow \\ 1 \quad 2 \quad 3 \end{array} := \begin{array}{c} \bullet \\ \swarrow \quad \searrow \\ i \quad j \quad k \\ \downarrow \quad \downarrow \quad \downarrow \\ 1 \quad 2 \quad 3 \end{array} - \begin{array}{c} \bullet \\ \swarrow \quad \searrow \\ i \quad j \quad k \\ \downarrow \quad \downarrow \quad \downarrow \\ 1 \quad 2 \quad 3 \end{array} - \begin{array}{c} \bullet \\ \swarrow \quad \searrow \\ i \quad j \quad k \\ \downarrow \quad \downarrow \quad \downarrow \\ 1 \quad 2 \quad 3 \end{array}. \quad (2)$$

The default ordering of edges is the one which we see.

**Definition 4** (Leibniz graph). A *Leibniz graph* is a graph whose vertices are either sinks, or the sources for two arrows, or the Jacobiator (which is a source for three arrows). There must be at least one Jacobiator vertex. The three arrows originating from a Jacobiator vertex must land on three distinct vertices. Each edge falling on a Jacobiator works by the Leibniz rule on the two internal vertices in it.

**Example 5.** The Jacobiator itself is a Leibniz graph (on one tri-valent internal vertex).

**Definition 5** (Normal form of a Leibniz graph with one Jacobiator). Let  $\Gamma$  be a Leibniz graph with one Jacobiator vertex  $\text{Jac}$ . From (2) we see that expansion of  $\text{Jac}$  into a sum of three Kontsevich graphs means adding one new edge  $w \rightarrow v$  (namely joining the internal vertices  $w$  and  $v$  within the Jacobiator). Now, from  $\Gamma$  construct three Kontsevich graphs by expanding  $\text{Jac}$  using (2) and letting the edges which fall on  $\text{Jac}$  in  $\Gamma$  be directed only to  $v$  in every new graph. Next, for each Kontsevich graph find the relabelling  $\tau$  which brings it to its normal form and re-express the edge  $w \rightarrow v$  using  $\tau$ . Finally, out of the three normal forms of three graphs pick the minimal one. By definition, the *normal form* of the Leibniz graph  $\Gamma$  is the pair: normal form of Kontsevich graph, that edge  $\tau(w) \rightarrow \tau(v)$ .

We say that a sum of Leibniz graphs is a *skew Leibniz graph*  $\text{Alt } \Gamma$  if it is produced from a given Leibniz graph  $\Gamma$  by alternation using formula (1).

**Definition 6** (Normal form of a skew Leibniz graph with one Jacobiator). Likewise, the normal form of a skew Leibniz graph  $\text{Alt } \Gamma$  is the minimum of the normal forms of Leibniz graphs (specifically, of the graph but not edge encodings) which are obtained from  $\Gamma$  by running over the group of permutations of the sinks content.

**Lemma 5** ([3]). In order to show that a sum  $S$  of weighted skew-symmetric Kontsevich graphs vanishes for all Poisson structures  $\mathcal{P}$ , it suffices to express  $S$  as a sum of skew Leibniz graphs:  $S = \diamond(\mathcal{P}, \text{Jac}(\mathcal{P}))$ .

### 1.1. Formulation of the problem

Let  $\mathcal{P} \mapsto \mathcal{P} + \varepsilon \mathcal{Q}(\mathcal{P}) + \bar{o}(\varepsilon)$  be a deformation of bi-vectors that preserves their property to be Poisson at least infinitesimally on all affine manifolds:  $\llbracket \mathcal{P} + \varepsilon \mathcal{Q} + \bar{o}(\varepsilon), \mathcal{P} + \varepsilon \mathcal{Q} + \bar{o}(\varepsilon) \rrbracket = \bar{o}(\varepsilon)$ . Expanding and equating the first order terms, we obtain the equation  $\llbracket \mathcal{P}, \mathcal{Q}(\mathcal{P}) \rrbracket \doteq 0$  via  $\llbracket \mathcal{P}, \mathcal{P} \rrbracket = 0$ . The language of Kontsevich graphs allows one to convert this infinite analytic problem within a given set-up  $(\mathcal{N}^n, \mathcal{P})$  in dimension  $n$  into a set of finite combinatorial problems whose solutions are universal for all Poisson geometries in all dimensions  $n < \infty$ .

Our first task in this paper is to find the space of flows  $\dot{\mathcal{P}} = \mathcal{Q}(\mathcal{P})$  which are encoded by the Kontsevich graphs on a fixed number of internal vertices  $k$ , for  $1 \leq k \leq 4$ . Specifically, we solve the graph equation

$$\llbracket \mathcal{P}, \mathcal{Q}(\mathcal{P}) \rrbracket = \diamond(\mathcal{P}, \text{Jac}(\mathcal{P})) \tag{3}$$

for the Kontsevich bi-vector graphs  $\mathcal{Q}(\mathcal{P})$  and Leibniz graphs  $\diamond$ . We then factor out the Poisson-trivial and improper solutions, that is, we quotient out all bi-vector graphs that can be written in the form  $\mathcal{Q}(\mathcal{P}) = \llbracket \mathcal{P}, X \rrbracket + \nabla(\mathcal{P}, \text{Jac}(\mathcal{P}))$ , where  $X$  is a Kontsevich one-vector graph and  $\nabla$  is a Leibniz bi-vector graph. (The bi-vectors  $\llbracket \mathcal{P}, X \rrbracket$  make  $\llbracket \mathcal{P}, \mathcal{Q}(\mathcal{P}) \rrbracket$  vanish since  $\llbracket \mathcal{P}, \cdot \rrbracket$  is a differential. The improper graphs  $\nabla(\mathcal{P}, \text{Jac}(\mathcal{P}))$  vanish identically at all Poisson bi-vectors  $\mathcal{P}$  on every affine manifold.

Before solving factorisation problem (3) with respect to the operator  $\diamond$ , we must generate – e.g., iteratively as described below – an ansatz for expansion of the right-hand side using skew Leibniz graphs with undetermined coefficients.

### 1.2. How to generate Leibniz graphs iteratively

The first step is to construct a layer of skew Leibniz graphs, that is, all skew Leibniz graphs which produce at least one graph in the input (in the course of expansion of skew Leibniz graphs using formula (1) and then in the course of expansion of every Leibniz graph at hand to a sum of Kontsevich graphs). For a given Kontsevich graph in the input  $S$ , one such Leibniz graph can be constructed by contracting an edge between two internal vertices so that the new vertex with three outgoing edges becomes the Jacobiator vertex. Note that these Leibniz graphs, which are designed to reproduce  $S$ , may also produce extra Kontsevich graphs that are not given in the input. Clearly, if the set of Kontsevich graphs in  $S$  coincides with the set of such graphs obtained by expansion of all the Leibniz graphs in the ansatz at hand, then we are done: the extra graphs, not present in  $S$ , are known to all cancel. Yet it could very well be that it is not possible to express  $S$  using only the Leibniz graphs from the set accumulated so far. Then we proceed by constructing the next layer of skew Leibniz graphs that reproduce at least one of the extra Kontsevich graphs (which were not present in  $S$  but which are produced by the graphs in the previously constructed layer(s) of Leibniz graphs). In this way we proceed iteratively until no new Leibniz graphs are found; of course, the overall number of skew Leibniz graphs on a fixed number of internal vertices and sinks is bounded from above so that the algorithm always terminates. Note that the Leibniz graphs obtained in this way are the only ones that can in principle be involved in the vanishing mechanism for  $S$ .

**Notation.** Let  $v$  be a graph vertex. Denote by  $N(v)$  the set of neighbours of  $v$ , by  $H(v)$  the (possibly empty) set of arrowheads of oriented edges issued from the vertex  $v$ , and by  $T(v)$  the (possibly empty) set of tails for oriented edges pointing at  $v$ . For example,  $\#N(\bullet) = 2$ ,  $\#H(\bullet) = 2$ , and  $T(\bullet) = \emptyset$  for the top  $\bullet$  of the wedge graph  $\leftarrow \bullet \rightarrow$ .

**Algorithm** Consider a skew-symmetric sum  $S_0$  of oriented Kontsevich graphs with real coefficients. Let  $S_{\text{total}} := S_0$  and create an empty table  $L$ . We now describe the  $i$ th iteration of the algorithm ( $i \geq 1$ ).

**Loop**  $\circ$  Run over all Kontsevich graphs  $\Gamma$  in  $S_{i-1}$ : for each internal vertex  $v$  in a graph  $\Gamma$ , run over all vertices  $w \in T(v)$  in the set of tails of oriented edges pointing at  $v$  such that  $v \notin T(w)$  and  $H(v) \cap H(w) = \emptyset$  for the sets of targets of oriented edges issued from  $v$  and  $w$ . Replace the edge  $w \rightarrow v$  connecting  $w$  to  $v$  by Jacobiator (2), that is, by a single vertex Jac with three outgoing edges and such that  $T(\text{Jac}) = (T(v) \setminus w) \cup T(w)$  and  $H(\text{Jac}) = H(v) \cup (H(w) \setminus v) =: \{a, b, c\}$ .

Because we shall always expand the skew Leibniz graphs in what follows, we do not actually contract the edge  $w \rightarrow v$  (to obtain a Leibniz graph explicitly) in this algorithm but instead we continue working with the original Kontsevich graphs containing the distinct vertices  $v$  and  $w$ .

For every edge that points at  $w$ , redirect it to  $v$ . Sum over the three cyclic permutations that provide three possible ways to attach the three outgoing edges for  $v$  and  $w$  (excluding  $w \rightarrow v$ ) – now seen as the outgoing edges of the Jacobiator – to the target vertices  $a, b$ , and  $c$  depending on  $w$  and  $v$ . Skew-symmetrise<sup>5</sup> each of these three graphs with respect to the sinks by applying formula (1).

For every marked edge  $w \rightarrow v$  indicating the internal edge in the Jacobiator vertex in a graph, replace each sum of the Kontsevich graphs which is skew with respect to the sink content by using the normal form of the respective skew Leibniz graph, see Definition 6. If this skew Leibniz graph is not contained in  $L$ , apply the Leibniz rule(s) for all the derivations acting on the Jacobiator vertex Jac. Otherwise speaking, sum over all possible ways to attach the incoming edges of the target  $v$  in the marked edge  $w \rightarrow v$  to its source  $w$  and target. To each Kontsevich

<sup>5</sup> This algorithm can be modified so that it works for an input which is not skew, namely, by replacing skew Leibniz graphs by ordinary Leibniz graphs (that is, by skipping the skew-symmetrisation). For example, this strategy has been used in [3, 4] to show that the Kontsevich star product  $\star \bmod \bar{o}(\hbar^4)$  is associative: although the associator  $(f \star g) \star h - f \star (g \star h) = \diamond(\mathcal{P}, \text{Jac}(\mathcal{P}))$  is not skew, it does vanish for every Poisson structure  $\mathcal{P}$ .

graph resulting from a skew Leibniz graph at hand assign the same undetermined coefficient, and add all these weighted Kontsevich graphs to the sum  $S_i$ . Further, add a row to the table  $L$ , that new row containing the normal form of this skew Leibniz graph (with its coefficient that has been made common to the Kontsevich graphs).

By now, the new sum of Kontsevich graphs  $S_i$  is fully composed. Having thus finished the current iteration over all graphs  $\Gamma$  in the set  $S_{i-1}$ , redefine the algebraic sum of weighted graphs  $S_{\text{total}}$  by subtracting from it the newly formed sum  $S_i$ . Collect similar terms in  $S_{\text{total}}$  and reduce this sum of Kontsevich graphs modulo their skew-symmetry under swaps  $L \rightleftharpoons R$  of the edge ordering in every internal vertex, so that all zero graphs (see Example 3) also get eliminated. ○ end loop

Increment  $i$  by 1 and repeat the iteration until the set of weighted (and skew) Leibniz graphs  $L$  stabilizes. Finally, solve – with respect to the coefficients of skew Leibniz graphs – the linear algebraic system obtained from the graph equation  $S_{\text{total}} = 0$  for the sum of Kontsevich graphs which has been produced from its initial value  $S_0$  by running the iterations of the above algorithm.

**Example 6.** For the skew sum of Kontsevich graphs in the right-hand side of (2), the algorithm would produce just one skew Leibniz graph: namely, the Jacobiator itself.

**Example 7 (The 3-wheel).** For the Kontsevich tetrahedral flow  $\dot{\mathcal{P}} = \mathcal{Q}_{1.6/2}(\mathcal{P})$  on the spaces of Poisson bi-vectors  $\mathcal{P}$ , see [8, 9] and [1, 2], building a sufficient set of skew Leibniz graphs in the r.-h. s. of factorisation problem (3) requires two iterations of the above algorithm: 11 Leibniz graphs are produced at the first step and 50 more are added by the second step, making 61 in total. One of the two known solutions of this factorisation problem [2] then consists of 8 skew Leibniz graphs (expanding to 27 Leibniz graphs). In turn, as soon as all the Leibniz rules acting on the Jacobiators are processed and every Jacobiator vertex is expanded via (2), the right-hand side  $\diamond(\mathcal{P}, \text{Jac}(\mathcal{P}))$  equals the sum of 39 Kontsevich graphs which are assembled into the 9 totally skew-symmetric terms in the left-hand side  $\llbracket \mathcal{P}, \mathcal{Q}_{1.6/2} \rrbracket$ .

**Example 8 (The 5-wheel).** Consider the factorisation problem  $\llbracket \mathcal{P}, \text{Or}(\gamma_5) \rrbracket = \diamond(\mathcal{P}, \text{Jac}(\mathcal{P}))$  for the pentagon-wheel deformation  $\dot{\mathcal{P}} = \text{Or}(\gamma_5)(\mathcal{P})$  of Poisson bi-vectors  $\mathcal{P}$ , see [10, 13] and [6]. The ninety skew Kontsevich graphs encoding the bi-vector  $\text{Or}(\gamma_5)$  are obtained by taking all the admissible orientations of two  $(5+1)$ -vertex graphs  $\gamma_5$ , one of which is the pentagon wheel with five spokes, the other graph complementing the former to a cocycle in the non-oriented graph complex. By running the iterations of the above algorithm for self-expanding construction of the Leibniz tri-vector graphs in this factorisation problem, we achieve stabilisation of the number of such graphs after the seventh iteration, see Table 1 below.

**Table 1.** The number of skew Leibniz graphs produced iteratively for  $\llbracket \mathcal{P}, \text{Or}(\gamma_5) \rrbracket$ .

No. iteration $i$	1	2	3	4	5	6	7	8
# of graphs	1518	14846	41031	54188	56318	56503	56509	56509
of them new	all	+13328	+26185	+13157	+2130	+185	+6	none

## 2. Generating the Kontsevich multi-vector graphs

Let us return to problem (3): it is the ansatz for bi-vector Kontsevich graphs  $\mathcal{Q}(\mathcal{P})$  with  $k$  internal vertices, as well as the Kontsevich 1-vectors  $\mathcal{X}$  with  $k-1$  internal vertices (to detect trivial terms  $\mathcal{Q}(\mathcal{P}) = \llbracket \mathcal{P}, \mathcal{X} \rrbracket$ ) which must be generated at a given  $k$ . (At  $1 \leq k \leq 4$ , one can still

expand with respect to *all* the Leibniz graphs in the r.-h.s. of (3), not employing the iterative algorithm from §1.2. So, a generator of the Kontsevich (and Leibniz) tri-vectors will also be described presently.)

The Kontsevich graphs corresponding to  $n$ -vectors are those graphs with  $n$  sinks (each containing the respective argument of  $n$ -vector) in which exactly one arrow comes into each sink, so that the order of the differential operator encoded by an  $n$ -vector graph equals one w.r.t. each argument, and which are totally skew-symmetric in their  $n$  arguments. Let us explain how one can economically obtain the set of one-vectors and skew-symmetric bi- and tri-vectors with  $k$  internal vertices in three steps (including graphs with eyes  $\bullet \rightleftharpoons \bullet$  but excluding graphs with tadpoles). This approach can easily be extended to the construction of  $n$ -vectors with any  $n \geq 1$ .

### 2.1. *One-vectors*

Each one-vector under study is encoded by a Kontsevich graph with one sink. Since the sink has one incoming arrow, there is an internal vertex as the tail of this incoming arrow. The target of another edge issued from this internal vertex can be any internal vertex other than itself.

*Step 1.* Generate all Kontsevich graphs on  $k - 1$  internal vertices and one sink (i.e. graphs including those with eyes yet excluding those with tadpoles, and not necessarily of differential order one with respect to the sink content).

*Step 2.* For every such graph with  $k - 1$  internal vertices, add the new sink and make it a target of the old sink, which itself becomes the  $k$ th internal vertex. Now run over the  $k - 1$  internal vertices excluding the old sink and – via the Leibniz rule – make every such internal vertex the second target of the old sink.

### 2.2. *Bi-vectors*

There are two cases in the construction of bi-vectors encoded by the Kontsevich graphs. At all  $k \geq 1$  the first variant is referred to those graphs with an internal vertex that has both sinks as targets.

*Variant 1: Step 1.* Generate all  $k$ -vertex graphs on  $k - 1$  internal vertices and one sink.

*Variant 1: Step 2.* For every such graph, add two new sinks and proclaim them as targets of the old sink.

Note that the obtained graphs are skew-symmetric.

The second variant produces those graphs which contain two internal vertices such that one has the first sink as target and the other has the second sink as target. The second target of either such internal vertex can be any internal vertex other than itself. Note that for  $k = 1$  only the first variant applies.

*Variant 2: Step 1.* Generate all  $k$ -vertex Kontsevich graphs on  $k - 2$  internal vertices and two sinks. These sinks now become the  $(k - 1)$ th and  $k$ th internal vertices.

*Variant 2: Step 2.* For every such graph, add two new sinks, make the first new sink a target of the first old sink and make the second new sink a target of the second old sink. Now run over the  $k - 1$  internal vertices excluding the first old sink, each time proclaiming an internal vertex the second target of the first old sink. Simultaneously, run over the  $k - 1$  internal vertices excluding the second old sink and likewise, declare an internal vertex to be the second target of the second old sink.

*Variant 2: Step 3.* Skew-symmetrise each graph with respect to the content of two sinks using (1).

### 2.3. *Tri-vectors*

For  $k \geq 3$ , there exist two variants of tri-vectors. The first variant at all  $k \geq 2$  yields those Kontsevich graphs with two internal vertices such that one has two of the three sinks as its targets while another internal vertex has the third sink as one of its targets. The second target

of this last vertex can be any internal vertex other than itself. The second variant contains those graphs with three internal vertices such that the first one has the first sink as a target, the second one has the second sink as a target, and the third one has the third sink as a target. For each of these three internal vertices with a sink as target, the second target can be any internal vertex other than itself.

*Variant 1: Step 1.* Generate all  $k$ -vertex Kontsevich graphs on  $k - 2$  internal vertices and two sinks.

*Variant 1: Step 2.* For every such graph, add three new sinks, make the first two new sinks the targets of the first old sink and make the third new sink a target of the second old sink. Now run over the  $k - 1$  internal vertices excluding the second old sink and every time declare an internal vertex the second target of the second old sink.

*Variant 1: Step 3.* Skew-symmetrise all graphs at hand by applying formula (1) to each of them.

Note that for  $k = 1$  there are no tri-vectors encoded by Kontsevich graphs and also note that for  $k = 2$  only the first variant applies.

*Variant 2: Step 1.* Generate all Kontsevich graphs on  $k - 3$  internal vertices and three sinks.

*Variant 2: Step 2.* For every such graph, add three new sinks, make the first new sink a target of the first old sink, make the second new sink a target of the second old sink and make the third new sink a target of the third old sink. Now run over the  $k - 1$  internal vertices excluding the first old sink and declare every such internal vertex the second target of the first old sink. Independently, run over the  $k - 1$  internal vertices excluding the second old sink and declare each internal vertex to be the second target of the second old sink. Likewise, run over the  $k - 1$  internal vertices excluding the third old sink and declare each internal vertex to be the second target of the third old sink.

*Variant 2: Step 3* Skew-symmetrise all the graphs at hand using (1).

#### 2.4. *Non-iterative generator of the Leibniz $n$ -vector graphs*

The following algorithm generates all Leibniz graphs with a prescribed number of internal vertices and sinks. Note that not only multi-vectors, but also all graphs of arbitrary differential order with respect to the sinks can be generated this way.

*Step 1:* Generate all Kontsevich graphs of prescribed type on  $k - 1$  internal vertices and  $n$  sinks, e.g., all  $n$ -vectors.

*Step 2:* Run through the set of these Kontsevich graphs and in each of them, run through the set of its internal vertices  $v$ . For every vertex  $v$  do the following: re-enumerate the internal vertices so that this vertex is enumerated by  $k - 1$ . This vertex already targets two vertices,  $i$  and  $j$ , where  $i < j < k - 1$ . Proclaim the last,  $(k - 1)$ th vertex to be the placeholder of the Jacobiator (see (2)), so we must still add the third arrow. Let a new index  $\ell$  run up to  $i - 1$  starting at  $n$  if only the  $n$ -vectors are produced.<sup>6</sup> For every admissible value of  $\ell$ , generate a new graph where the  $\ell$ th vertex is proclaimed the third target of the Jacobiator vertex  $k - 1$ . (Restricting  $\ell$  by  $< i$ , we reduce the number of possible repetitions in the set of Leibniz graphs. Indeed, for every triple  $\ell < i < j$ , the same Leibniz graph in which the Jacobiator stands on those three vertices would be produced from the three Kontsevich graphs: namely, those in which the  $(k - 1)$ th vertex targets at the  $\ell$ th and  $i$ th, at the  $\ell$ th and  $j$ th, and at the  $i$ th and  $j$ th vertices. In these three cases it is the  $j$ th,  $i$ th, and  $\ell$ th vertex, respectively, which would be appointed by the algorithm as the Jacobiator's third target.)

We use this algorithm to generate the Leibniz tri- and bi-vector graphs: to establish Theorem 6, we list all possible terms in the right-hand side of factorisation problem (3) at  $k \leq 4$  and then we filter out the improper bi-vectors in the found solutions  $Q(\mathcal{P})$ .

<sup>6</sup> If we want to generate not only  $n$ -vectors but all graphs of arbitrary differential orders, then we let  $\ell$  start at 0 (so that the sinks are included).

*Remark 2.* There are at least 265,495 Leibniz graphs on 3 sinks and 6 internal vertices of which one is the Jacobiator vertex. When compared with Table 1 on p. 352, this estimate suggests why at large  $k \gtrsim 5$ , the breadth-first-search iterative algorithm from §1.2 generates a smaller number of the Leibniz tri-vector graphs, namely, only the ones which can in principle be involved in the factorisation under study.

### 3. Main result

**Theorem 6** ( $k \leq 4$ ). *The few-vertex solutions of problem (3) are these (note that disconnected Kontsevich graphs in  $\mathcal{Q}(\mathcal{P})$  are allowed!):*

- $k = 1$ : *The dilation  $\dot{\mathcal{P}} = \mathcal{P}$  is a unique, nontrivial and proper solution.*
- $k = 2$ : *No solutions exist (in particular, neither trivial nor improper).*
- $k = 3$ : *There are no solutions: neither Poisson-trivial nor Leibniz bi-vectors.*
- $k = 4$ : *A unique nontrivial and proper solution is the Kontsevich tetrahedral flow  $\mathcal{Q}_{1, \frac{6}{2}}(\mathcal{P})$  from Example 2 (see [8, 9] and [1, 2]). There is a one-dimensional space of Poisson trivial (still proper) solutions  $[[\mathcal{P}, X]]$ ; the Kontsevich 1-vector  $X$  on three internal vertices is drawn in [2, App. F]. Intersecting with the former by  $\{0\}$ , there is a three-dimensional space of improper (still Poisson-nontrivial) solutions of the form  $\nabla(\mathcal{P}, \text{Jac}(\mathcal{P}))$ .*

*None of the solutions  $\mathcal{Q}(\mathcal{P})$  known so far contains any 2-cycles (or “eyes”  $\bullet \rightleftharpoons \bullet$ ).*<sup>7</sup>

We now report a classification of Poisson bi-vector symmetries  $\dot{\mathcal{P}} = \mathcal{Q}(\mathcal{P})$  which are given by those Kontsevich graphs  $\mathcal{Q} = \text{Or}(\gamma)$  on  $k$  internal vertices that can be obtained at  $5 \leq k \leq 9$  by orienting  $k$ -vertex connected graphs  $\gamma$  without double edges. By construction, this extra assumption keeps only those Kontsevich graphs which may not contain eyes.

We first find such graphs  $\gamma$  that satisfy  $d(\gamma) = 0$ , then we exclude the coboundaries  $\gamma = d(\gamma')$  for some graphs  $\gamma'$  on  $k - 1$  vertices and  $2k - 3$  edges.

**Theorem 7** ( $5 \leq k \leq 8$ ). *Consider the vector space of non-oriented connected graphs on  $k$  vertices and  $2k - 2$  edges, without tadpoles and without multiple edges. All nontrivial d-cocycles for  $5 \leq k \leq 8$  are exhausted by the following ones:*

- $k = 5, 7$ : *No solutions.*
- $k = 6$ : *A unique solution<sup>8</sup> is given by the Kontsevich–Willwacher pentagon-wheel cocycle (see Example 1). The established factorisation  $[[\mathcal{P}, \text{Or}(\gamma_5)]] = \diamond(\mathcal{P}, \text{Jac}(\mathcal{P}))$  will be addressed in a separate paper (see [6]).*
- $k = 8$ : *The only solution  $\gamma_7$  consists of the heptagon-wheel and 45 other graphs (see Table 2, in which the coefficient of heptagon graph is **1** in bold, and [5]).*

$$\gamma_5 = \text{Diagram 1} + \frac{5}{2} \text{Diagram 2}$$

*Remark 3.* The wheel graphs are built of triangles. The differential  $d$  cannot produce any triangle since multiple edges are not allowed. Therefore, all wheel cocycles are nontrivial. Note also that every wheel graph with  $2\ell$  spokes is invariant under a mirror reflection with respect to a diagonal consisting of two edges attached to the centre. Hence there exists an edge permutation that swaps  $2\ell - 1$  pairs of edges. By footnote 1 such graph equals zero.

### Appendix A. How the orientation mapping $\text{Or}$ is calculated

The algorithm lists all ways in which a given non-oriented graph can be oriented in such a way that it becomes a Kontsevich graph on two sinks. It consists of two steps:

- (i) choosing the source(s) of the two arrows pointing at the first and second sink, respectively;

<sup>7</sup> Finding solutions  $\mathcal{Q}(\mathcal{P})$  with tadpoles or extra sinks – with fixed arguments – is a separate problem.

<sup>8</sup> There are only 12 admissible graphs to build cocycles from; of these 12, as many as 6 are zero graphs. This count shows to what extent the number of graphs decreases if one restricts to only the flows  $\mathcal{Q} = \text{Or}(\gamma)$  obtained from cocycles  $\gamma \in \ker(d)$  in the non-oriented graph complex.

**Table 2.** The heptagon-wheel graph cocycle  $\gamma_7$ .

Graph encoding	Coeff.	Graph encoding	Coeff.
16 17 18 23 25 28 34 38 46 48 57 58 68 78	1	12 13 18 25 26 37 38 45 46 47 56 57 68 78	-7
12 14 18 23 27 35 37 46 48 57 58 67 68 78	-21/8	12 14 16 23 25 36 37 45 48 57 58 67 68 78	77/8
13 14 18 23 25 28 37 46 48 56 57 67 68 78	-77/4	13 16 17 24 25 26 35 37 45 48 58 67 68 78	-7
12 13 15 24 27 35 36 46 48 57 58 67 68 78	-35/8	14 15 17 23 26 28 37 38 46 48 56 57 68 78	49/4
12 13 18 24 26 37 38 46 47 56 57 58 68 78	49/8	12 16 18 27 28 34 36 38 46 47 56 57 58 78	-147/8
14 17 18 23 25 26 35 37 46 48 56 58 67 78	77/8	12 15 16 27 28 35 36 38 45 46 47 57 68 78	-21/8
12 13 18 26 27 35 38 45 46 47 56 57 68 78	-105/8	12 14 18 23 27 35 36 45 46 57 58 67 68 78	-35/8
12 14 18 23 27 36 38 46 48 56 57 58 67 78	7/8	14 15 16 23 26 28 37 38 46 48 57 58 67 78	-49/4
12 14 15 23 27 35 36 46 48 57 58 67 68 78	35/8	12 15 18 23 28 34 37 46 48 56 57 67 68 78	105/8
12 13 14 27 28 36 38 46 47 56 57 58 68 78	-49/8	12 14 17 23 26 37 38 46 48 56 57 58 68 78	-49/8
12 13 18 25 27 34 36 47 48 56 58 67 68 78	35/4	12 16 18 25 27 35 36 37 45 46 48 57 68 78	49/16
12 13 14 25 26 36 38 45 47 57 58 67 68 78	-119/16	12 13 18 25 27 35 36 46 47 48 56 57 68 78	7
12 13 15 24 28 36 38 47 48 56 57 67 68 78	49/8	12 14 18 25 28 34 36 38 47 57 58 67 68 78	-7
12 13 14 23 28 37 46 48 56 57 58 67 68 78	77/4	12 16 18 25 27 35 36 37 45 46 48 58 67 78	-77/16
12 15 17 25 26 35 36 38 45 47 48 67 68 78	-49/8	12 14 18 23 27 35 38 46 47 57 58 67 68 78	77/4
13 15 18 24 26 28 37 38 46 47 56 57 68 78	-49/4	12 14 15 23 27 36 38 46 48 57 58 67 68 78	35/2
13 14 18 25 26 28 36 38 47 48 56 57 67 78	-49/4	12 13 18 25 27 34 36 46 48 57 58 67 68 78	-105/8
12 14 18 23 28 35 37 46 48 56 57 67 68 78	-7	12 15 16 25 27 35 36 38 46 47 48 57 68 78	-7
12 14 18 23 28 36 38 46 47 56 57 58 67 78	-7	12 13 16 25 28 34 37 47 48 57 58 67 68 78	-147/16
12 15 16 25 27 35 36 38 46 47 48 58 67 78	49/8	12 13 17 25 26 35 37 45 46 48 58 67 68 78	-77/4
12 14 18 23 28 36 37 46 47 56 57 58 68 78	49/8	12 14 17 23 27 35 38 46 48 57 58 67 68 78	-49/8
12 13 15 26 27 35 36 45 47 48 58 67 68 78	-7	12 13 15 26 28 35 37 45 46 47 58 67 68 78	-7/4
12 13 18 24 28 35 38 46 47 57 58 67 68 78	7	12 14 18 23 26 36 38 47 48 56 57 58 67 78	-7

(ii) orienting the edges between the internal vertices in all admissible ways, so that only Kontsevich graphs are obtained.

*Step 1.* Enumerate the  $k$  vertices of a given non-oriented, connected graph using  $2, \dots, k + 1$ . They become the internal vertices of the oriented graph. Now add the two sinks to the non-oriented graph, the sinks enumerated using 0 and 1. Let  $a$  and  $b$  be a non-strictly ordered ( $a \leq b$ ) pair of internal vertices in the graph. Extend the graph by oriented edges  $a \rightarrow 0$  and  $b \rightarrow 1$  from vertices  $a$  and  $b$  to the sinks 0 and 1, respectively.

*Remark 4.* The choice of such a base pair, that is, the vertex or vertices from which two arrows are issued to the sinks, is an external input in the orientation procedure. Let us agree that if, at any step of the algorithm, a contradiction is achieved so that a graph at hand cannot be of Kontsevich type, the oriented graph draft is discarded; one proceeds with the next options in that loop, or if the former loop is finished, with the next level-up loops, or – having returned to the choice of base vertices – with the next base. In other words, we do not exclude in principle a possibility to have no admissible orientations for a particular choice of the base for a given non-oriented graph.

**Notation.** Let  $v$  be an internal vertex. Recalling from p. 351 the notation for the set  $N(v)$  of neighbours of  $v$ , the (initially empty) set  $H(v)$  of arrowheads of oriented edges issued from the vertex  $v$ , and the (initially empty) set  $T(v)$  of tails for oriented edges pointing at  $v$ , we now put by definition  $F(v) := N(v) \setminus (H(v) \cup T(v))$ . In other words,  $F(v)$  is the subset of neighbours connected with  $v$  by a non-oriented edge.

*Step 2.1. Inambiguous orientation of (some) edges.* Here we use that every internal vertex of a Kontsevich graph should be the tail of exactly two outgoing arrows. We run over the set of all internal vertices  $v$ . For every vertex such that the number of elements  $\#H(v) = 2$ ,



proclaim  $T(v) := N(v) \setminus H(v)$ , whence  $F(v) = \emptyset$ . If for a vertex  $v$  we have that  $\#H(v) = 1$  and  $\#F(v) = 1$ , then include  $F(v) \hookrightarrow H(v)$ , that is, convert a unique non-oriented edge touching  $v$  into an outgoing edge issued from this vertex. If  $\#H(v) = 0$  and  $\#F(v) = 2$ , also include  $F(v) \hookrightarrow H(v)$ , effectively making both non-oriented edges outgoing from  $v$ .

Repeat the three parts of Step 2.1 while any of the sets  $F(v)$ ,  $T(v)$ , or  $S(v)$  is modified for at least one internal vertex  $v$  unless a contradiction is revealed. Summarising, Step 2.1 amounts to finding the edge orientations which are implied by the choice of the base pair  $a, b$  and by all the orientations of edges fixed earlier.

*Step 2.2. Fixing the orientation of (some) remaining edges.* Choose an internal vertex  $v$  such that  $H(v) < 2$  and such that  $H(v) \neq \emptyset$  or  $T(v) \neq \emptyset$ , that is, choose a vertex that is not yet equipped with two outgoing edges and that is attached to an oriented edge. If  $\#H(v) = 1$ , then run over the non-empty set  $F(v)$ : for every vertex  $w$  in  $F(v)$ , include  $\{w\} \hookrightarrow H(v)$  and start over at Step 2.1. Otherwise, i.e. if  $H(v) = \emptyset$ , run over all ordered pairs  $(u, v)$  of vertices in the set  $F(v)$ : for every such pair, make  $H(v) := \{u, w\}$  and start over at Step 2.1.

By realising Steps 1 and 2 we accumulate the sum of fully oriented Kontsevich graphs.

## Acknowledgments

A. V. Kiselev thanks the Organising committee of the international conference ISQS'25 on integrable systems and quantum symmetries (6–10 June 2017 in ČVUT Prague, Czech Republic) for a warm atmosphere during the meeting. The authors are grateful to M. Kontsevich and T. Willwacher for helpful discussion. We also thank Center for Information Technology of the University of Groningen for providing access to Peregrine high performance computing cluster. This research was supported in part by JBI RUG project 106552 (Groningen, The Netherlands). A part of this research was done while R. Buring and A. V. Kiselev were visiting at the IHÉS (Bures-sur-Yvette, France) and A. V. Kiselev was visiting at the MPIM (Bonn, Germany).

## References

- [1] Bouisaghouane A., Kiselev A. V. (2017) Do the Kontsevich tetrahedral flows preserve or destroy the space of Poisson bi-vectors? *J. Phys.: Conf. Ser.* **804** Proc. XXIV Int. conf. ‘Integrable Systems and Quantum Symmetries’ (14–18 June 2016, ČVUT Prague, Czech Republic), Paper 012008, 10 p. (Preprint [arXiv:1609.06677](#) [q-alg])
- [2] Bouisaghouane A., Buring R., Kiselev A. (2017) The Kontsevich tetrahedral flow revisited, *J. Geom. Phys.* **119**, 272–285. (Preprint [arXiv:1608.01710](#) [q-alg])
- [3] Buring R., Kiselev A. V. (2017) On the Kontsevich  $\star$ -product associativity mechanism, *PEPAN Letters* **14**:2, 403–407. (Preprint [arXiv:1602.09036](#) [q-alg])
- [4] Buring R., Kiselev A. V. (2017) The expansion  $\star \bmod \bar{o}(\hbar^4)$  and computer-assisted proof schemes in the Kontsevich deformation quantization, *Preprint* [arXiv:1702.00681](#) [math.CO]
- [5] Buring R., Kiselev A. V., Rutten N. J. (2017) The heptagon-wheel cocycle in the Kontsevich graph complex, *J. Nonlin. Math. Phys.* **24** Suppl. 1 ‘Local & Nonlocal Symmetries in Mathematical Physics’, 157–173. (Preprint [arXiv:1710.00658](#) [math.CO])
- [6] Buring R., Kiselev A. V., Rutten N. J. (2017) Poisson brackets symmetry from the pentagon-wheel cocycle in the graph complex, *Preprint* [arXiv:1712.05259](#) [math-ph]
- [7] Dolgushev V. A., Rogers C. L., Willwacher T. H. (2015) Kontsevich’s graph complex, GRT, and the deformation complex of the sheaf of polyvector fields, *Ann. Math.* **182**:3, 855–943. (Preprint [arXiv:1211.4230](#) [math.KT])
- [8] Kontsevich M. (1997) Formality conjecture. Deformation theory and symplectic geometry (Ascona 1996, D. Sternheimer, J. Rawnsley and S. Gutt, eds), *Math. Phys. Stud.* **20**, Kluwer Acad. Publ., Dordrecht, 139–156.
- [9] Kontsevich M. (2017) Derived Grothendieck–Teichmüller group and graph complexes [after T. Willwacher], *Séminaire Bourbaki* (69ème année, Janvier 2017), no. 1126, 26 p.
- [10] Kontsevich M. (2017) Private communication.
- [11] Khoroshkin A., Willwacher T., Živković M. (2017) Differentials on graph complexes, *Adv. Math.* **307**, 1184–1214. (Preprint [arXiv:1411.2369](#) [q-alg])

- [12] *Willwacher T.* (2015) M. Kontsevich's graph complex and the Grothendieck–Teichmüller Lie algebra, *Invent. Math.* **200**:3, 671–760. (*Preprint arXiv:1009.1654* [q-alg])
- [13] *Willwacher T.* (2017) Private communication.

# Chapter 15

## The Kontsevich tetrahedral flow revisited

This chapter is based on the peer-reviewed journal publication *A. Bouisaghouane, R. Buring, and A. V. Kiselev, J. Geom. Phys.*, **119**, 272–285, 2017. (Preprint [arXiv:1608.01710](https://arxiv.org/abs/1608.01710) [q-alg] – 29 p.)

*Commentary.* In reference to Part I of the dissertation, the material of this chapter is used in Chapter 2, §5.1, §6.1, Chapter 7, Chapter 8, and Chapter 9. With this chapter we amend the formula for the tetrahedral flow (originally proposed by Kontsevich in Ascona '96). The Poisson cocycle factorization problem which we study here is a particular example of the general construction, which we address in Chapter 17. In the meantime, we discover multiple solutions to that factorization problem (see §3.5.2 of Part I).

# THE KONTSEVICH TETRAHEDRAL FLOW REVISITED

A. BOUISAGHOUANE, R. BURING, AND A. KISELEV<sup>\*,§</sup>

ABSTRACT. We prove that the Kontsevich tetrahedral flow  $\dot{\mathcal{P}} = \mathcal{Q}_{a:b}(\mathcal{P})$ , the right-hand side of which is a linear combination of two differential monomials of degree four in a bi-vector  $\mathcal{P}$  on an affine real Poisson manifold  $N^n$ , does infinitesimally preserve the space of Poisson bi-vectors on  $N^n$  if and only if the two monomials in  $\mathcal{Q}_{a:b}(\mathcal{P})$  are balanced by the ratio  $a : b = 1 : 6$ . The proof is explicit; it is written in the language of Kontsevich graphs.

**Introduction.** The main question which we address in this paper is how Poisson structures can be deformed in such a way that they stay Poisson. We reveal one such method that works for all Poisson structures on affine real manifolds; the construction of that flow on the space of bi-vectors was proposed in [11]: the formula is derived from two differently oriented tetrahedral graphs on four vertices. The flow is a linear combination of two terms, each quartic-nonlinear in the Poisson structure. By using several examples of Poisson brackets with high polynomial degree coefficients, the first and last authors demonstrated in [1] that the ratio  $1 : 6$  is the only possible balance at which the tetrahedral flow can preserve the property of the Cauchy datum to be Poisson. But does the Kontsevich tetrahedral flow  $\dot{\mathcal{P}} = \mathcal{Q}_{1:6}(\mathcal{P})$  with ratio  $1 : 6$  actually preserve the space of *all* Poisson bi-vectors?

We prove the infinitesimal version of this claim: namely, we show that  $[[\mathcal{P}, \mathcal{Q}_{1:6}(\mathcal{P})]] = 0$  for every bi-vector  $\mathcal{P}$  satisfying the master-equation  $[[\mathcal{P}, \mathcal{P}]] = 0$  for Poisson structures. The proof is graphical: to prove that equation (2) holds, we find an operator  $\diamond$ , encoded by using the Kontsevich graphs, that solves equation (10). We also show that there is no universal mechanism (that would involve the language of Kontsevich graphs) for the tetrahedral flow to be trivial in the respective Poisson cohomology.

The text is structured as follows. In section 1 we recall how oriented graphs can be used to encode differential operators acting on the space of multivectors. In particular, differential polynomials in a given Poisson structure are obtained as soon as a copy of that Poisson bi-vector is placed in every internal vertex of a graph. Specifically, the right-hand side  $\mathcal{Q}_{a:b} = a \cdot \Gamma_1 + b \cdot \Gamma_2$  of the Kontsevich tetrahedral flow  $\dot{\mathcal{P}} = \mathcal{Q}_{a:b}(\mathcal{P})$  on the space of bi-vectors on an affine Poisson manifold  $(N^n, \mathcal{P})$  is a linear combination

---

*Date:* 24 November 2016, revised 12 March 2017.

*2010 Mathematics Subject Classification.* 53D55, 58E30, 81S10; secondary 53D17, 58Z05, 70S20.

*Key words and phrases.* Poisson bracket, affine manifold, graph complex, tetrahedral flow, Poisson cohomology.

*Address:* Johann Bernoulli Institute for Mathematics and Computer Science, University of Groningen, P.O. Box 407, 9700 AK Groningen, The Netherlands. \**E-mail:* A.V.Kiselev@rug.nl

\* Institut des Hautes Études Scientifiques, 35 route de Chartres, Bures-sur-Yvette, F-91440 France.

§ *Present address:* Max Planck Institute for Mathematics, Vivatsgasse 7, D-53111 Bonn, Germany.

of two differential monomials,  $\Gamma_1(\mathcal{P})$  and  $\Gamma_2(\mathcal{P})$ , of degree four in the bi-vector  $\mathcal{P}$  that evolves.

We determine at which balance  $a : b$  the Kontsevich tetrahedral flow  $\dot{\mathcal{P}} = \mathcal{Q}_{a:b}(\mathcal{P})$  infinitesimally preserves the space of Poisson bi-vectors, that is, the bi-vector  $\mathcal{P} + \varepsilon \mathcal{Q}_{a:b}(\mathcal{P}) + \bar{o}(\varepsilon)$  satisfies the equation

$$\llbracket \mathcal{P} + \varepsilon \mathcal{Q}_{a:b}(\mathcal{P}) + \bar{o}(\varepsilon), \mathcal{P} + \varepsilon \mathcal{Q}_{a:b}(\mathcal{P}) + \bar{o}(\varepsilon) \rrbracket \doteq \bar{o}(\varepsilon) \quad \text{via } \llbracket \mathcal{P}, \mathcal{P} \rrbracket = 0; \quad (1)$$

here we denote by  $\llbracket \cdot, \cdot \rrbracket$  the Schouten bracket (see formula (5) on page 364). Expanding, we obtain the cocycle condition,

$$\llbracket \mathcal{P}, \mathcal{Q}_{a:b}(\mathcal{P}) \rrbracket \doteq 0 \quad \text{via } \llbracket \mathcal{P}, \mathcal{P} \rrbracket = 0, \quad (2)$$

with respect to the Poisson differential  $\partial_{\mathcal{P}} = \llbracket \mathcal{P}, \cdot \rrbracket$ . Viewed as an equation with respect to the ratio  $a : b$ , condition (2) is the main object of our study.

Recent counterexamples [1] show that the bi-vector  $\mathcal{P} + \varepsilon \mathcal{Q}_{a:b}(\mathcal{P}) + \bar{o}(\varepsilon)$  can stay Poisson *only if* the balance  $a : b$  in  $\mathcal{Q}_{a:b}$  is equal to  $1 : 6$ . We now prove the infinitesimal part of sufficiency: the deformation  $\mathcal{P} + \varepsilon \mathcal{Q}_{1:6}(\mathcal{P}) + \bar{o}(\varepsilon)$  is always infinitesimally Poisson, whence the balance  $a : b = 1 : 6$  in the Kontsevich tetrahedral flow is universal for all Poisson bi-vectors  $\mathcal{P}$  on all affine manifolds  $N^n$ . The proof is explicit: in section 2 we reveal the mechanism of factorization – via the Jacobi identity – in (2) at  $a : b = 1 : 6$ . On the left-hand side of factorization problem (2) we expand the Poisson differential of the Kontsevich tetrahedral flow at the balance  $1 : 6$  into the sum of 39 graphs (see Figure 3 on page 367 and Table 2 in Appendix A). On the other side of that factorization, we take the sum that runs with undetermined coefficients over all those fragments of differential consequences of the Jacobi identity  $\llbracket \mathcal{P}, \mathcal{P} \rrbracket = 0$  which are known to vanish independently. We then find a linear polydifferential operator  $\diamond(\mathcal{P}, \cdot)$  that acts on the filtered components of the Jacobiator  $\text{Jac}(\mathcal{P}) := \llbracket \mathcal{P}, \mathcal{P} \rrbracket$  for the bi-vector  $\mathcal{P}$ ; the operator  $\diamond$  provides the factorization  $\llbracket \mathcal{P}, \mathcal{Q}_{1:6}(\mathcal{P}) \rrbracket(f, g, h) = \diamond(\mathcal{P}, \text{Jac}(\mathcal{P})(\cdot, \cdot, \cdot))(f, g, h)$  of the  $\partial_{\mathcal{P}}$ -cocycle condition, see (2), through the Jacobi identity  $\text{Jac}(\mathcal{P}) = 0$ . To describe the differential operators that produce such consequences of the Jacobi identity, we use the pictorial language of graphs: every internal vertex contains a copy of the bi-vector  $\mathcal{P}$  and the operators are reduced by using its skew-symmetry. There remain 7,025 graphs, the coefficients of which are linear in the unknowns. We now solve the arising inhomogeneous linear algebraic system. Its solution yields a polydifferential operator  $\diamond$ , encoded using Leibniz graphs (see p. 371), that provides the sought-for factorization  $\llbracket \mathcal{P}, \mathcal{Q}_{1:6} \rrbracket = \diamond(\mathcal{P}, \text{Jac}(\mathcal{P}))$ . It is readily seen from formula (11) that this operator  $\diamond$  is completely determined by only 8 nonzero coefficients (out of 1132 total).<sup>1</sup> Therefore, although finding an operator  $\diamond$  was hard, verifying that it does solve the factorization problem has become almost immediate, as we show in the proof of Theorem 3. We thereby establish the main result (namely, Corollary 4 on page 369). The paper concludes with the formulation of an open problem about the integration of tetrahedral flow in (1) to higher order expansions in  $\varepsilon$ , see (13) on p. 373.

In Appendix B we outline a different method to tackle the factorization problem, namely, by making the Jacobi identity visible in (2) by perturbing the original structure  $\mathcal{P} \mapsto \tilde{\mathcal{P}}$  in such a way that  $\tilde{\mathcal{P}}$  is not Poisson and  $\mathcal{Q}_{1:6}(\tilde{\mathcal{P}}) \neq 0$ . Hence  $\tilde{\mathcal{P}}$  contributes

<sup>1</sup>The maximally detailed description of that solution  $\diamond$  is contained in Appendix A.

to the right-hand side of (2) such that the respectively perturbed bi-vector  $Q_{1:6}(\tilde{\mathcal{P}})$  stops being compatible with the perturbed Poisson structure  $\tilde{\mathcal{P}}$ . The first-order balance of both sides of perturbed equation (2) then suggests the coefficients of those differential consequences of the Jacobiator which are actually involved in the factorization mechanism. The coefficients of operators realized by graphs which were found by following this scheme are reproduced in the full run-through that gave us the solution  $\diamond$  in section 2.

1. THE MAIN PROBLEM: FROM GRAPHS TO MULTIVECTORS

**1.1. The language of graphs.** Let us formalise a way to encode polydifferential operators – in particular multivectors – using oriented graphs [9, 10]. In an affine real manifold  $N^n$  (here  $2 \leq n < \infty$ ), take a chart  $U_\alpha \hookrightarrow \mathbb{R}^n$  and denote the Cartesian coordinates by  $\mathbf{x} = (x^1, \dots, x^n)$ . We now consider only the oriented graphs whose vertices are either tails for an ordered pair of arrows, each decorated with its own index, or sinks (with no issued edges) like the vertices  $1, 2$  in  $(1) \xleftarrow{i} \bullet \xrightarrow{j} (2)$ . The arrowtail vertices are called *internal*. Every internal vertex  $\bullet$  carries a copy of a given Poisson bi-vector  $\mathcal{P} = \mathcal{P}^{ij}(\mathbf{x}) \partial_i \wedge \partial_j$  with its own pair of indices. For each internal vertex  $\bullet$ , the pair of out-going edges is ordered  $L \prec R$ . The ordering  $L \prec R$  of decorated out-going edges coincides with the ordering “first  $\prec$  second” of the indexes in the coefficients of  $\mathcal{P}$ . Namely, the left edge (L) carries the first index and the other edge (R) carries the second index. By definition, the decorated edge  $\bullet \xrightarrow{i} \bullet$  denotes at once the derivation  $\partial/\partial x^i \equiv \partial_i$  (that acts on the content of the arrowhead vertex) and the summation  $\sum_{i=1}^n$  (over the index  $i$  in the object which is contained within the arrowtail vertex). As it has been explained in [7, 12], the operator which every graph encodes is equal to the sum (running over all the indexes) of products (running over all the vertices) of those vertices content (differentiated by the in-coming arrows, if any). Moreover, we let the sinks be ordered (like  $1 \prec 2$  above), so that every such graph defines a polydifferential operator: its arguments are thrown into the respective sinks.

**Example 1.** The *wedge graph*  $(1) \xleftarrow[L]{i} \mathcal{P}^{ij}(\mathbf{x}) \xrightarrow[R]{j} (2)$  encodes the bi-differential operator  $\sum_{i,j=1}^n (1) \overleftarrow{\partial}_i \cdot \mathcal{P}^{ij}(\mathbf{x}) \cdot \overrightarrow{\partial}_j (2)$ . Such graph specifies a Poisson bracket (on every chart  $U_\alpha \subseteq N^n$ ) if it satisfies the Jacobi identity, see (4) below.

*Remark 1.* In principle, we allow the presence of both the tadpoles and cycles over two vertices (or “eyes”), see Fig. 1. However, in hindsight there will be neither tadpoles nor eyes in the solution which we shall have found in section 2 below.



FIGURE 1. A tadpole and an “eye”.

*Remark 2.* Under the above assumptions, there exist inhabited graphs that encode zero differential operators. Namely, consider the graph with a double edge:

$$\begin{array}{c}
 \begin{array}{c} i \\ \curvearrowright \\ \mathbf{a} \end{array} \begin{array}{c} L \\ \curvearrowleft \\ R \\ \end{array} \begin{array}{c} \\ \\ z \end{array} = \begin{array}{c} j \\ \curvearrowright \\ \mathbf{a} \end{array} \begin{array}{c} L \\ \curvearrowleft \\ R \\ \end{array} \begin{array}{c} \\ \\ z \end{array} = -\begin{array}{c} j \\ \curvearrowright \\ \mathbf{a} \end{array} \begin{array}{c} R \\ \curvearrowleft \\ L \\ \end{array} \begin{array}{c} \\ \\ z \end{array} \stackrel{\text{flip}}{=} -\begin{array}{c} i \\ \curvearrowright \\ \mathbf{a} \end{array} \begin{array}{c} L \\ \curvearrowleft \\ R \\ \end{array} \begin{array}{c} \\ \\ z \end{array}
 \end{array}$$

By first relabelling the summation indices and then swapping  $L \rightleftharpoons R$  (and redrawing) we evaluate the operator acting at  $z$  to  $\sum_{i,j=1}^n a^{ij} \partial_i \partial_j(z) = -\sum_{i,j=1}^n a^{ij} \partial_i \partial_j(z)$ ; whence the operator is zero. In the same way, any graph containing a double edge encodes a zero operator. Graphs can also encode zero differential operators in a more subtle way. For example consider the wedge on two wedges:

$$\begin{array}{c}
 \begin{array}{c} 3 \\ \curvearrowright \\ \mathbf{a} \end{array} \begin{array}{c} R \\ \curvearrowleft \\ L \\ \end{array} \begin{array}{c} \\ \\ 2 \end{array} \\
 \begin{array}{c} \\ \\ \mathbf{a} \end{array} \begin{array}{c} \\ \\ 1 \end{array} \\
 \begin{array}{c} f \\ \curvearrowright \\ \mathbf{a} \end{array} \begin{array}{c} \\ \\ g \end{array}
 \end{array} = 0. \tag{3}$$

Swapping the labels  $1 \rightleftharpoons 2$  of the lower wedges does not change the operator. On the other hand, doing the same in a different way, namely, by swapping ‘left’ and ‘right’ in the top wedge introduces a minus sign. Hence the graph encodes a differential operator equal to minus itself, i.e. zero. Proving that a graph which contains the left-hand side of (3) as a subgraph equals zero is an elementary exercise (cf. Example 3 on p. 376).

Besides the trivial vanishing mechanism in Remark 2, there is the Jacobi identity together with its differential consequences, which will play a key role in what follows. For any three arguments  $1, 2, 3 \in C^\infty(N^n)$ , the Jacobi identity  $\text{Jac}_{\mathcal{P}}(1, 2, 3) = 0$  is realized<sup>2</sup> by the graph

$$\begin{array}{c} \bullet \\ \bullet \end{array} \begin{array}{c} \bullet \\ \bullet \end{array} := \begin{array}{c} i \\ \curvearrowright \\ \mathbf{a} \end{array} \begin{array}{c} j \\ \curvearrowright \\ \mathbf{a} \end{array} \begin{array}{c} k \\ \curvearrowright \\ \mathbf{a} \end{array} - \begin{array}{c} i \\ \curvearrowright \\ \mathbf{a} \end{array} \begin{array}{c} L \\ \curvearrowright \\ R \\ \end{array} \begin{array}{c} j \\ \curvearrowright \\ \mathbf{a} \end{array} \begin{array}{c} k \\ \curvearrowright \\ \mathbf{a} \end{array} - \begin{array}{c} i \\ \curvearrowright \\ \mathbf{a} \end{array} \begin{array}{c} j \\ \curvearrowright \\ \mathbf{a} \end{array} \begin{array}{c} k \\ \curvearrowright \\ \mathbf{a} \end{array} = 0. \tag{4}$$

In our notation this identity’s left-hand side encodes a sum over all  $(i, j, k)$ ; instead restricting to fixed  $(i, j, k)$  corresponds to taking a coefficient of the differential operator (cf. Lemma 1 below), which yields the respective component  $\text{Jac}_{\mathcal{P}}^{ijk}$  of the Jacobiator  $\text{Jac}(\mathcal{P})$ . Clearly, the Jacobiator is totally skew-symmetric with respect to its arguments  $1, 2, 3$ .

In fact, the Jacobiator  $\text{Jac}(\mathcal{P})$  is the *Schouten bracket* of a given Poisson bi-vector  $\mathcal{P}$  with itself:  $\text{Jac}(\mathcal{P}) = \llbracket \mathcal{P}, \mathcal{P} \rrbracket$  (depending on conventions, times a constant which is here omitted, cf. [8]). The bracket  $\llbracket \cdot, \cdot \rrbracket$  is a unique extension of the commutator  $[\cdot, \cdot]$  on the space of vector fields  $\mathcal{X}^1(N^n)$  to the space  $\mathcal{X}^*(N^n)$  of multivector fields. Let us recall its inductive definition in the finite-dimensional set-up.

**Definition 1.** The Schouten bracket  $\llbracket \cdot, \cdot \rrbracket: \mathcal{X}^*(N^n) \times \mathcal{X}^*(N^n) \rightarrow \mathcal{X}^*(N^n)$  coincides with the commutator  $[\cdot, \cdot]$  when evaluated on 1-vectors; when evaluated at a  $p$ -vector  $\mathcal{X}$ ,

<sup>2</sup>The notation  $\text{Jac}_{\mathcal{P}}(1, 2, 3)$  is synonymic to  $\text{Jac}(\mathcal{P})(1 \otimes 2 \otimes 3)$ .

$q$ -vector  $\mathcal{Y}$ , and  $r$ -vector  $\mathcal{Z}$  for  $p, q, r \geq 1$ , the Schouten bracket is shifted-graded skew-symmetric,  $[[\mathcal{X}, \mathcal{Y}]] = -(-1)^{(p-1)(q-1)}[[\mathcal{Y}, \mathcal{X}]]$ , and it works over each argument via the graded Leibniz rule:  $[[\mathcal{X}, \mathcal{Y} \wedge \mathcal{Z}]] = [[\mathcal{X}, \mathcal{Y}]] \wedge \mathcal{Z} + (-1)^{(p-1)q} \mathcal{Y} \wedge [[\mathcal{X}, \mathcal{Z}]]$ . The bracket is then extended by linearity from homogeneous components to the entire space of multivector fields on  $N^n$ .

*Remark 3.* The construction of Schouten bracket also reads as follows. Denote by  $\xi_i$  the parity-odd canonical conjugate of the variable  $x^i$  for every  $i = 1, \dots, n$ . For instance, every bi-vector is realised in terms of local coordinates  $x^i$  and  $\xi_i$  on  $\Pi T^*N^n$  by using  $\mathcal{P} = \frac{1}{2} \langle \xi_i \mathcal{P}^{ij}(\mathbf{x}) \xi_j \rangle$ . The Schouten bracket  $[[\cdot, \cdot]]$  is the parity-odd Poisson bracket which is locally determined on  $\Pi T^*N^n$  by the canonical symplectic structure  $d\mathbf{x} \wedge d\xi$ . Our working formula is<sup>3</sup>

$$[[\mathcal{P}, \mathcal{Q}]] = (\mathcal{P}) \overleftarrow{\partial} \cdot \overrightarrow{\partial} \frac{(\mathcal{Q})}{\partial x^i} - (\mathcal{Q}) \overleftarrow{\partial} \cdot \overrightarrow{\partial} \frac{(\mathcal{P})}{\partial \xi_i}. \tag{5}$$

It is now readily seen that the Schouten bracket of homogeneous arguments satisfies its own, shifted-graded Jacobi identity,

$$[[\mathcal{X}, [[\mathcal{Y}, \cdot]]](\mathcal{Z}) - (-1)^{(|\mathcal{X}|-1) \cdot (|\mathcal{Y}|-1)} [[\mathcal{Y}, [[\mathcal{X}, \cdot]]](\mathcal{Z}) = [[[[\mathcal{X}, \mathcal{Y}], \cdot]](\mathcal{Z}).$$

Hence for a bi-vector  $\mathcal{P}$  such that  $[[\mathcal{P}, \mathcal{P}]] = 0$ , the map  $\partial_{\mathcal{P}} = [[\mathcal{P}, \cdot]]: \mathcal{X}^\ell(N^n) \rightarrow \mathcal{X}^{\ell+1}(N^n)$  is a differential.

*Remark 4.* The graphical calculation of the Schouten bracket  $[[\cdot, \cdot]]$  of two arguments amounts to the action – via the Leibniz rule – of every out-going edge in an argument on all the internal vertices in the other argument. For the Schouten bracket of a  $k$ -vector with an  $\ell$ -vector, the rule of signs is this. For the sake of definition, enumerate the sinks in the first and second arguments by using  $0, \dots, k - 1$  and  $0, \dots, \ell - 1$ , respectively. Then the arrow into the  $j$ th sink in the second argument acts on the internal vertices of the first argument, acquiring the sign factor  $(-)^j$ ; here  $0 \leq j < \ell$ . On the other hand, the arrow to the  $i$ th sink in the first argument acts on the second argument’s internal vertices with the sign factor  $-(-)^{(k-1)-i}$  for  $0 \leq i \leq k - 1$ .

The rule of signs, as it has been phrased above, is valid — provided that, for a  $k$ -vector  $\mathcal{X}$  and  $\ell$ -vector  $\mathcal{Y}$ , the numbers  $0, \dots$  of the  $k$  (or  $k - 1$ ) sinks originating in the  $(k + \ell - 1)$ -vector  $[[\mathcal{X}, \mathcal{Y}]]$  from the first argument  $\mathcal{X}$  precede the numbers of  $\ell - 1$  (resp.,  $\ell$ ) sinks originating from  $\mathcal{Y}$  in the overall enumeration of those  $k + \ell - 1$  sinks.<sup>4</sup> For example, it is this ordering of sinks using  $1 \prec 2$  which is shown in (6),

$$[[\begin{array}{c} \bullet \\ \swarrow \quad \searrow \\ \underset{1}{\bullet} \quad \underset{2}{\bullet} \end{array}, \begin{array}{c} \bullet \\ \downarrow \\ \underset{1}{\bullet} \end{array}]] = + \begin{array}{c} \bullet \\ \downarrow \\ \begin{array}{c} \swarrow \quad \searrow \\ \underset{1}{\bullet} \quad \underset{2}{\bullet} \end{array} \end{array} - \left( \begin{array}{c} \bullet \\ \swarrow \quad \searrow \\ \underset{1}{\bullet} \quad \underset{2}{\bullet} \end{array} - \begin{array}{c} \bullet \\ \downarrow \\ \begin{array}{c} \swarrow \quad \searrow \\ \underset{2}{\bullet} \quad \underset{1}{\bullet} \end{array} \end{array} \right); \tag{6}$$

here  $k = 2, \ell = 1$  and the enumeration of arguments begins at 1.

<sup>3</sup>In the set-up of infinite jet spaces  $J^\infty(\pi)$  (see [13] and [5, 6, 7]) the four partial derivatives in formula (5) for  $[[\cdot, \cdot]]$  become the variational derivatives with respect to the same variables, which now parametrise the fibres in the Whitney sum  $\pi \times_{M^m} \Pi \widehat{\pi}$  of (super-)bundles over the  $m$ -dimensional base  $M^m$ .

<sup>4</sup>Such is the default convention which formula (5) suggests for the product of parity-odd variables  $\xi_{i_\alpha}$ , where  $1 \leq \alpha \leq k + \ell - 1$ .



Still let us note that in its realization via Kontsevich graphs, the calculation of the Schouten bracket  $[[\cdot, \cdot]]$  effectively amounts to a consecutive plugging of one of its arguments into each of the other argument's sinks (see (6) again). Therefore, it would be more natural to start enumerating the sinks of the graph that acts (on the new content in one of its sinks, possibly the first), but when that new argument is reached, to interrupt and now enumerate the argument's own sinks, then continuing the enumeration of sinks (if there still remain any to be counted) in the graph that acts. This change of enumeration strategy comes at a price of having extra sign factors in front of the graphs. Namely, the arrow into the  $j$ th sink in the second argument acquires the extra sign factor  $(-)^{j \cdot k}$ . Similarly, the arrow to the  $i$ th sink in the first argument of  $[[\cdot, \cdot]]$  must now be multiplied by  $(-)^{\ell \cdot (k-1-i)}$ ; we recall that  $0 \leq i < k$  and  $0 \leq j < \ell$ . We note that for  $k$  and  $\ell$  even (e.g.,  $k = 2$  and  $\ell = 2$  in formula (9)) no extra sign factors appear at all from the re-ordering at a price of  $(-)^{j \cdot k}$  and  $(-)^{\ell \cdot (k-1-i)}$ . For example, such is the final ordering of the  $3 = 2 + 1 = 1 + 2$  sinks which is shown in Fig. 3 on p. 367.

Summarizing, to be Poisson a bi-vector  $\mathcal{P}$  must satisfy the *master-equation*,

$$[[\mathcal{P}, \mathcal{P}]] = 0, \tag{7}$$

of which formula (4) is the component expansion with respect to the indices  $(i, j, k)$  in the tri-vector  $[[\mathcal{P}, \mathcal{P}]](\mathbf{x}, \boldsymbol{\xi})$ .

**Definition 2.** Let  $\mathcal{P}$  be a Poisson bi-vector on the manifold  $N^n$  at hand and consider its deformation  $\mathcal{P} \mapsto \mathcal{P} + \varepsilon \mathcal{Q}(\mathcal{P}) + \bar{o}(\varepsilon)$ . We say that after such deformation the bi-vector stays *infinitesimally Poisson* if

$$[[\mathcal{P} + \varepsilon \mathcal{Q}(\mathcal{P}) + \bar{o}(\varepsilon), \mathcal{P} + \varepsilon \mathcal{Q}(\mathcal{P}) + \bar{o}(\varepsilon)]] = \bar{o}(\varepsilon), \tag{1'}$$

that is, the master-equation is still satisfied up to  $\bar{o}(\varepsilon)$  for a given solution  $\mathcal{P}$  of (7).

*Remark 5.* Nowhere above should one expect that the leading deformation term  $\mathcal{Q}$  in  $\mathcal{P} + \varepsilon \mathcal{Q} + \bar{o}(\varepsilon)$  itself would be a Poisson bi-vector. This may happen for  $\mathcal{Q}$  only incidentally.

Expanding the left-hand side of equation (1) and using the shifted-graded skew-symmetry of the Schouten bracket  $[[\cdot, \cdot]]$ , we extract the deformation equation

$$[[\mathcal{P}, \mathcal{Q}]] \doteq 0 \quad \text{via } [[\mathcal{P}, \mathcal{P}]] = 0. \tag{2}$$

Let us consider a class of its solutions  $\mathcal{Q} = \mathcal{Q}(\mathcal{P})$  which are universal with respect to all finite-dimensional affine Poisson manifolds  $(N^n, \mathcal{P})$ .

**1.2. The Kontsevich tetrahedral flow.** In the paper [11], Kontsevich proposed a particular construction of infinitesimal deformations  $\mathcal{P} \mapsto \mathcal{P} + \varepsilon \mathcal{Q}(\mathcal{P}) + \bar{o}(\varepsilon)$  for Poisson structures on affine real manifolds. One such flow  $\dot{\mathcal{P}} = \mathcal{Q}(\mathcal{P})$  on the space of Poisson bi-vectors  $\mathcal{P}$  is associated with the complete graph on four vertices, that is, the tetrahedron. Up to symmetry, there are two essentially different ways, resulting in  $\Gamma_1$  and  $\Gamma'_2$ , to orient its edges, provided that every vertex is a source for two arrows and, as an elementary count suggests, there are two arrows leaving the tetrahedron and acting on the arguments of the bi-differential operator which the tetrahedral graph encodes. The two oriented tetrahedral graphs are shown in Fig. 2. Unlike the operator encoded by  $\Gamma_1$ , that of  $\Gamma'_2$  is generally speaking not skew-symmetric with respect to its arguments. By

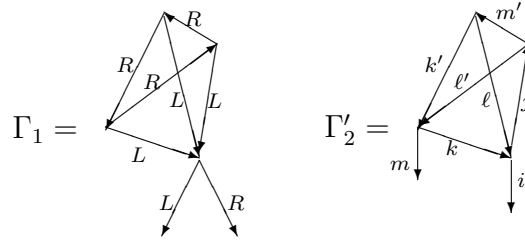


FIGURE 2. The Kontsevich tetrahedral graphs encode two bi-linear bi-differential operators on the product  $C^\infty(N^n) \times C^\infty(N^n)$ .

definition, put  $\Gamma_2 := \frac{1}{2}(\Gamma'_2(1, 2) - \Gamma'_2(2, 1))$  to extract the antisymmetric part, that is, the bi-vector encoded by  $\Gamma'_2$ . Explicitly, the quartic-nonlinear differential polynomials  $\Gamma_1(\mathcal{P})$  and  $\Gamma_2(\mathcal{P})$ , depending on a Poisson bi-vector  $\mathcal{P}$ , are given by the formulae

$$\Gamma_1(\mathcal{P}) = \sum_{i,j=1}^n \left( \sum_{k,\ell,m,k',\ell',m'=1}^n \frac{\partial^3 \mathcal{P}^{ij}}{\partial x^k \partial x^\ell \partial x^m} \frac{\partial \mathcal{P}^{kk'}}{\partial x^{\ell'}} \frac{\partial \mathcal{P}^{\ell\ell'}}{\partial x^{m'}} \frac{\partial \mathcal{P}^{mm'}}{\partial x^{k'}} \right) \frac{\partial}{\partial x^i} \wedge \frac{\partial}{\partial x^j} \quad (8a)$$

and

$$\Gamma_2(\mathcal{P}) = \sum_{i,m=1}^n \left( \sum_{j,k,\ell,k',\ell',m'=1}^n \frac{\partial^2 \mathcal{P}^{ij}}{\partial x^k \partial x^\ell} \frac{\partial^2 \mathcal{P}^{km}}{\partial x^{k'} \partial x^{\ell'}} \frac{\partial \mathcal{P}^{k'\ell}}{\partial x^{m'}} \frac{\partial \mathcal{P}^{m'\ell'}}{\partial x^j} \right) \frac{\partial}{\partial x^i} \wedge \frac{\partial}{\partial x^m}, \quad (8b)$$

respectively. To construct a class of flows on the space of bi-vectors, Kontsevich suggested to consider linear combinations, balanced by using the ratio  $a : b$ , of the bi-vectors  $\Gamma_1$  and  $\Gamma_2$ . We recall from section 1.1 that every internal vertex of each graph is inhabited by a copy of a given Poisson bi-vector  $\mathcal{P}$ , so that the linear combination of two graphs encodes the bi-vector  $\mathcal{Q}_{a,b}(\mathcal{P}) = a \cdot \Gamma_1(\mathcal{P}) + b \cdot \Gamma_2(\mathcal{P})$ , quartic in  $\mathcal{P}$  and balanced using  $a : b$ . We now inspect at which ratio  $a : b$  the bi-vector  $\mathcal{P} + \varepsilon \mathcal{Q}_{a,b}(\mathcal{P}) + \bar{o}(\varepsilon)$  stays infinitesimally Poisson, that is,

$$[\mathcal{P} + \varepsilon \mathcal{Q}_{a,b}(\mathcal{P}) + \bar{o}(\varepsilon), \mathcal{P} + \varepsilon \mathcal{Q}_{a,b}(\mathcal{P}) + \bar{o}(\varepsilon)] = \bar{o}(\varepsilon). \quad (1)$$

The left-hand side of the deformation equation,

$$[\mathcal{P}, \mathcal{Q}_{a,b}(\mathcal{P})] \doteq 0 \quad \text{via } [\mathcal{P}, \mathcal{P}] = 0, \quad (2)$$

can be seen in terms of graphs:

$$[\mathcal{P}, a \cdot \Gamma_1 + b \cdot \Gamma_2] = \left[ \begin{array}{c} \text{graph with two vertices } 1, 2 \\ \text{graph } \Gamma_1 \\ \text{graph } \Gamma_2 \end{array} \right]. \quad (9)$$

Let  $a : b = 1 : 6$  (specifically,  $a = \frac{1}{4}$  and  $b = \frac{3}{2}$ ). Then the left-hand side of (2) takes the shape depicted in Fig. 3. After the expansion of Leibniz rules and skew-symmetrization, the sum in Fig. 3 simplifies to 39 graphs; they are listed in Table 2 on p. 374 below. Collecting, we conclude that the left-hand side of (2) is the sum of 9 manifestly skew-symmetric expressions, see Fig. 4 (and Table 3 in Appendix A). For example, when outlining a proof of our main theorem (see p. 371), we shall explain how the coefficient  $-\frac{1}{2}$  of the first and second graphs in Fig. 4 is accumulated from the terms in the right-hand side of (10). Simultaneously, we shall track how the coefficients

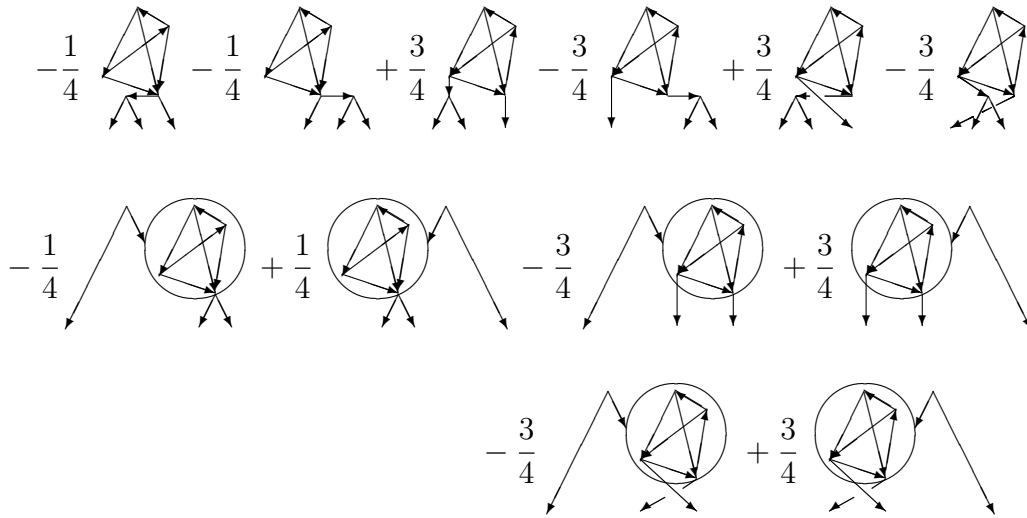


FIGURE 3. Incoming arrows act on the content of boxes via the Leibniz rule; to obtain the tri-vector, the entire picture must be skew-symmetrized over the content of three sinks using  $\sum_{\sigma \in S_3} (-)^\sigma$ . Expanding and skew-symmetrizing, one obtains 39 graphs in the left-hand side of (2).

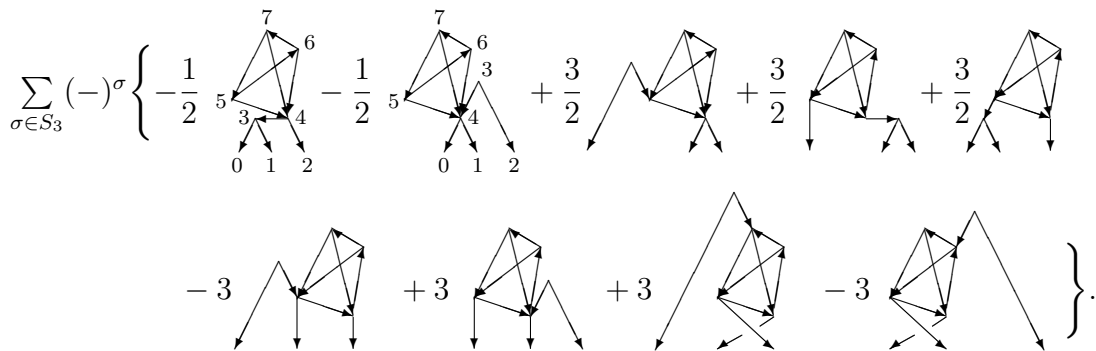


FIGURE 4. This sum of graphs is the skew-symmetrized content of Fig. 3. In what follows, we realize these 9 terms in the left-hand side of (2) by using an operator  $\diamond$  acting, in the right-hand side of (10) below, on the Jacobiator (4).

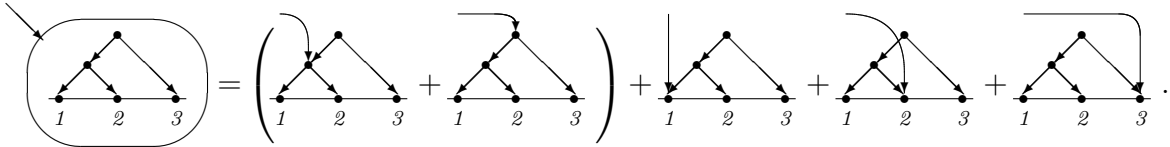
cancel out for the two other graphs which are produced by expanding the same Leibniz rules (that gave the above two graphs).

**1.3. Main result.** The reason why we are particularly concerned with the ratio  $a : b = 1 : 6$  is that this condition is *necessary* for equation (2) to hold. This has been proved in [1] by producing examples of Poisson bi-vector  $\mathcal{P}$  such that  $[[\mathcal{P}, Q_{a,b}(\mathcal{P})]] = 0$  only when  $a : b = 1 : 6$ . Let us now inspect whether this condition is also sufficient. The task is to factorize the content of Fig. 4 through the Jacobi identity in (4).

We first examine the mechanism for the tri-vector  $[[\mathcal{P}, \mathcal{Q}_{1:6}(\mathcal{P})]]$  in (2) to vanish by virtue of the Jacobi identity  $\text{Jac}(\mathcal{P}) = 0$  for a given Poisson bi-vector  $\mathcal{P}$  on an affine manifold  $N^n$  of any dimension. We claim that the Jacobiator  $\text{Jac}_{\mathcal{P}}(\cdot, \cdot, \cdot)$  is not necessarily (indeed, far not always!) evaluated at the three arguments  $f, g, h$  of the tri-vector  $[[\mathcal{P}, \mathcal{Q}_{1:6}(\mathcal{P})]]$ . A sample graph that can actually appear in such factorizing operators  $\diamond$  is drawn in Fig. 5 below.

**Lemma 1** ([2]). A tri-differential operator  $C = \sum_{|I|,|J|,|K|\geq 0} c^{IJK} \partial_I \otimes \partial_J \otimes \partial_K$  with coefficients  $c^{IJK} \in C^\infty(N^n)$  vanishes identically iff all its homogeneous components  $C_{ijk} = \sum_{|I|=i,|J|=j,|K|=k} c^{IJK} \partial_I \otimes \partial_J \otimes \partial_K$  vanish for all differential orders  $(i, j, k)$  of the respective multi-indices  $(I, J, K)$ ; here  $\partial_L = \partial_1^{\alpha_1} \circ \dots \circ \partial_n^{\alpha_n}$  for a multi-index  $L = (\alpha_1, \dots, \alpha_n)$ .

In practice, Lemma 1 states that for every arrow falling on the Jacobiator  $\text{Jac}_{\mathcal{P}}(1, 2, 3)$  – for which, in turn, a triple of arguments  $1, 2, 3$  is specified – the expansion of the Leibniz rule yields four fragments which vanish separately: e.g., we have that



Namely, there is the fragment such that the derivation acts on the content  $\mathcal{P}$  of the Jacobiator’s two internal vertices, and there are three fragments such that the arrow falls on the first, second, or third argument of the Jacobiator. Now it is readily seen that the action of a derivative  $\partial_i$  on an argument of the Jacobiator amounts to an appropriate redefinition of that argument:  $\partial_i(\text{Jac}_{\mathcal{P}}(1, 2, 3)) =$

$$\underbrace{(\partial_i \text{Jac}_{\mathcal{P}})(1, 2, 3)}_{=0} + \underbrace{\text{Jac}_{\mathcal{P}}(\partial_i(1), 2, 3)}_{=0} + \underbrace{\text{Jac}_{\mathcal{P}}(1, \partial_i(2), 3)}_{=0} + \underbrace{\text{Jac}_{\mathcal{P}}(1, 2, \partial_i(3))}_{=0} = 0.$$

Let us introduce a name for the (class of) graphs which make the first term – out of four – in the expansion of Leibniz rule in the above formula.

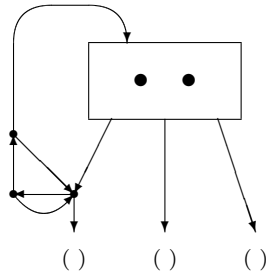
**Definition 3.** A *Leibniz graph* is a graph whose vertices are either sinks, or the sources for two arrows, or the Jacobiator (which is a source for three arrows). There must be at least one Jacobiator vertex. The three arrows originating from a Jacobiator vertex must land on three distinct vertices. Each edge falling on a Jacobiator works by the Leibniz rule on the two internal vertices in it.

An example of a Leibniz graph is given in Fig. 5. Every Leibniz graph can be expanded to a sum of Kontsevich graphs, by expanding both the Leibniz rule(s) and all copies of the Jacobiator; e.g. see (12). In this way Leibniz graphs also encode (poly)differential operators, depending on the bi-vector  $\mathcal{P}$  and the tri-vector  $\text{Jac}(\mathcal{P})$ .

**Proposition 2.** For every Poisson bi-vector  $\mathcal{P}$  the value – at the Jacobiator  $\text{Jac}(\mathcal{P})$  – of every (poly)differential operator encoded by the Leibniz graph(s) is zero.

**Theorem 3.** *There exists a polydifferential operator*

$$\diamond \in \text{PolyDiff} \left( \Gamma(\bigwedge^2 TN^n) \times \Gamma(\bigwedge^3 TN^n) \rightarrow \Gamma(\bigwedge^3 TN^n) \right)$$



- There is a cycle,
- there is a loop,
- there are no tadpoles in this graph,
- an arrow falls back on  $\text{Jac}(\mathcal{P})$ ,
- and  $\text{Jac}(\mathcal{P})$  does not stand on all of the three sinks.

FIGURE 5. This is an example of Leibniz graph of which the factorizing operators can consist.

which solves the factorization problem

$$[[\mathcal{P}, \mathcal{Q}_{1.6}(\mathcal{P})]](f, g, h) = \diamond (\mathcal{P}, \text{Jac}_{\mathcal{P}}(\cdot, \cdot, \cdot))(f, g, h). \tag{10}$$

The polydifferential operator  $\diamond$  is realised using Leibniz graphs in formula (11), see p. 371 below.

**Corollary 4** (Main result). Whenever a bi-vector  $\mathcal{P}$  on an affine real manifold  $N^n$  is Poisson, the deformation  $\mathcal{P} + \varepsilon \mathcal{Q}_{1.6}(\mathcal{P}) + \bar{o}(\varepsilon)$  using the Kontsevich tetrahedral flow is infinitesimally Poisson.

*Remark 6.* It is readily seen that the Kontsevich tetrahedral flow  $\dot{\mathcal{P}} = \mathcal{Q}_{1.6}(\mathcal{P})$  is well defined on the space of Poisson bi-vectors on a given affine manifold  $N^n$ . Indeed, it does not depend on a choice of coordinates up to their arbitrary affine reparametrisations. In other words, the velocity  $\dot{\mathcal{P}}|_{\mathbf{u} \in N^n}$  does not depend on the choice of a chart  $\mathcal{U} \ni \mathbf{u}$  from an atlas in which only *affine* changes of variables are allowed. (Let us remember that affine manifolds can of course be topologically nontrivial.)

Suppose however that a given affine structure on the manifold  $N^n$  is extended to a larger atlas on it; for the sake of definition let that atlas be a smooth one. Assume that the smooth structure is now reduced –by discarding a number of charts– to another affine structure on the same manifold. The tetrahedral flow  $\dot{\mathcal{P}} = \mathcal{Q}_{1.6}(\mathcal{P})$  which one initially had can be contrasted with the tetrahedral flow  $\dot{\tilde{\mathcal{P}}} = \mathcal{Q}_{1.6}(\tilde{\mathcal{P}})$  which one finally obtains for the Poisson bi-vector  $\tilde{\mathcal{P}}|_{\tilde{\mathbf{u}}(\mathbf{u})} = \mathcal{P}|_{\mathbf{u}}$  in the course of a nonlinear change of coordinates on  $N^n$ . Indeed, the respective velocities  $\dot{\mathcal{P}}$  and  $\dot{\tilde{\mathcal{P}}}$  can be different whenever they are expressed by using essentially different parametrisations of a neighbourhood of a point  $\mathbf{u}$  in  $N^n$ . For example, the tetrahedral flow vanishes identically when expressed in the Darboux canonical variables on a chart in a symplectic manifold. But after a nonlinear transformation, the right-hand side  $\mathcal{Q}_{1.6}(\tilde{\mathcal{P}})$  can become nonzero at the same points of that Darboux chart.

This shows that an affine structure on the manifold  $N^n$  is a necessary part of the input data for construction of the Kontsevich tetrahedral flows  $\dot{\mathcal{P}} = \mathcal{Q}_{1.6}(\mathcal{P})$ .

## 2. SOLUTION OF THE FACTORIZATION PROBLEM

Expanding the Leibniz rules in  $[[\mathcal{P}, \mathcal{Q}_{1.6}(\mathcal{P})]]$ , we obtain the sum of 39 graphs with 5 internal vertices and 3 sinks (so that from Figure 3 we produce Table 2, see page 374

below). By construction, the Schouten bracket  $[[\mathcal{P}, \mathcal{Q}_{1:6}(\mathcal{P})]] \in \Gamma(\wedge^3 TN^n)$  is a tri-vector on the underlying manifold  $N^n$ , that is, it is a totally antisymmetric tri-linear polyderivation  $C^\infty(N^n) \times C^\infty(N^n) \times C^\infty(N^n) \rightarrow C^\infty(N^n)$ . At the same time, we seek to recognize the tri-vector  $[[\mathcal{P}, \mathcal{Q}_{1:6}(\mathcal{P})]]$  as the result of application of a (poly)differential operator  $\diamond$  (see (10) in Theorem 3) to the Jacobiator  $\text{Jac}(\mathcal{P})$  (see (4) on p. 363).

We now explain how the operator  $\diamond$  is found.<sup>5</sup> The ansatz for  $\diamond$  is the sum – with undetermined coefficients – of all (separately vanishing) Leibniz graphs containing one Jacobiator and three wedges, and having differential order  $(1, 1, 1)$  with respect to the sinks (see Fig. 6). We thus have 28,202 unknowns introduced (counted with possible repetitions of graphs which they refer to). Expanding all the Leibniz rules and Jacobiators, we obtain a sum of Kontsevich graphs with 5 internal vertices on 3 sinks.

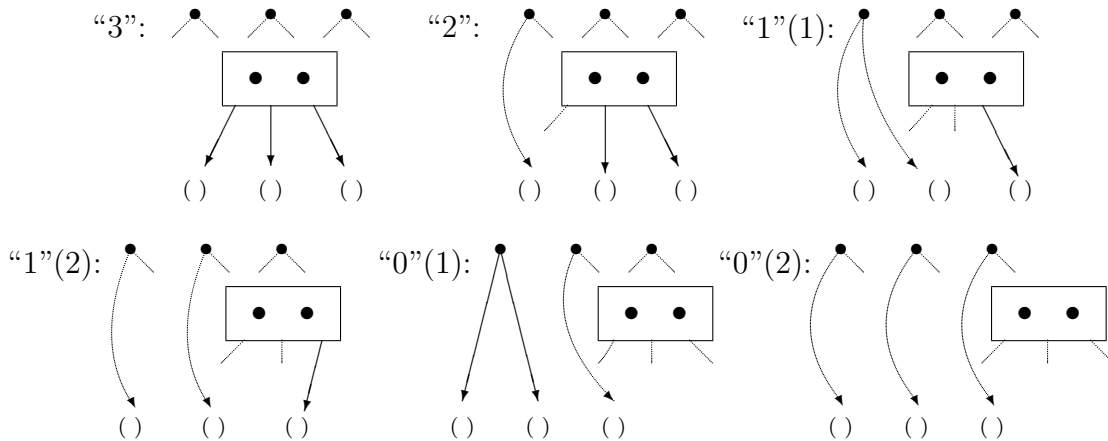


FIGURE 6. This is the list of all different types of Leibniz graphs which are linear in the Jacobiator and which have differential order  $(1, 1, 1)$  with respect to the sinks. The list is ordered by the number of ground vertices on which the Jacobiator stands.

As soon as we take into account the order  $L \prec R$  and the antisymmetry of graphs under the reversion of that ordering at an internal vertex, the graphs that encode zero differential operators are eliminated.<sup>6</sup> There remain 7,025 admissible graphs with 5 internal vertices on 3 sinks; the coefficient of every such graph is a linear combination of the undetermined coefficients of the Leibniz graphs. In conclusion, we view (10) as the system of 7,025 linear inhomogeneous equations for the coefficients of Leibniz graphs in the operator  $\diamond$ . Solving this linear system is a way towards a proof of our main result (which is expressed in Corollary 4). The process of finding a solution  $\diamond$  itself does not constitute that proof. Therefore, the justification of the claim in Theorem 3 will be performed separately. In the meantime, using software tools, we solve the linear algebraic system at hand. The duplications of graph labellings are conveniently

<sup>5</sup>Another method for solving the factorization problem is outlined in Appendix B.

<sup>6</sup>The relevant algebra of sums of graphs modulo skew-symmetry and the Jacobi identity has been realized in software by the second author. An implementation of those tools in the problem of high-order expansion of the Kontsevich  $\star$ -product is explained in a separate paper, see [3].

eliminated by our request for the program to find a solution with a minimal number of nonzero components. Totally antisymmetric in tri-vector's arguments, the solution consists of 27 Leibniz graphs, which are assimilated into the sum of 8 manifestly skew-symmetric terms as follows:

$$\begin{aligned}
 \diamond &= \text{Graph 1} + 3 \sum_{\tau \in S_2} (-)^\tau \text{Graph 2} + 3 \sum_{\circlearrowleft} \text{Graph 3} \\
 &+ 3 \sum_{\circlearrowleft} \left\{ \text{Graph 4} + \text{Graph 5} + \text{Graph 6} \right\} \\
 &+ 3 \sum_{\sigma \in S_3} (-)^\sigma \left\{ \text{Graph 7} + \text{Graph 8} \right\}.
 \end{aligned} \tag{11}$$

To display the  $L \prec R$  ordering at every internal vertex and to make possible the arithmetic and algebra of graphs, we use the notation which is explained in Appendix A.

*Remark 7.* We remember that the set  $\{1, 2, 3\}$  of three arguments of the Jacobiator need not coincide with the set  $\{f, g, h\}$  of the arguments of the tri-vector  $\diamond(\mathcal{P}, \text{Jac}(\mathcal{P}))$ . Of course, the two sets can intersect; this provides a natural filtration for the components of solution (11). Namely, the number of elements in the intersection runs from three for the first term to zero in the second or third graph.

In fact, Remark 7 reveals a highly nontrivial role of the operator  $\diamond$  in (10). Some of the three internal vertices of its graphs can be arguments of  $\text{Jac}(\mathcal{P})$  whereas some of the other such vertices (if any) can be tails for the arrows falling on  $\text{Jac}(\mathcal{P})$ . In retrospect, the two subsets of such vertices of  $\diamond$  do not intersect; every vertex in the intersection, if it were nonempty, would produce a two-cycle, but there are no “eyes” in (11).

*Proof of Theorem 3.* So far, we have constructed operator (11); it involves a reasonably small number of Leibniz graphs so that the factorization in (10) can be verified by a straightforward calculation. The sums in (11) contain 27 Leibniz graphs. Now expand all the Leibniz rules; this yields the sum of 201 Kontsevich graphs with 3 sinks

and 5 internal vertices: together with their coefficients, they are listed in Table 5 in Appendix A, see page 374. We claim that by collecting similar terms, one obtains the 39 graphs from the left-hand side of (10), see Fig. 4 and the encoding of those graphs in Table 3 on page 375. Because we are free to enumerate the five internal vertices in every graph in a way we like, and because the ordering of every pair of outgoing edges is also under our control, at once do we bring all the graphs to their normal form.<sup>7</sup>

It is readily seen that there are many repetitions in Table 5. We must inspect what vanishes and what stays. Let us do a sample reasoning first. Namely, let us inspect the contribution to the left-hand side of (10) from the first term of (11). We have that

$$\begin{array}{c} \text{Graph} \end{array} = \sum_{\sigma} \left\{ 5 \begin{array}{c} \text{Graph 1} \\ \text{Graph 2} \end{array} + 3 \begin{array}{c} \text{Graph 3} \\ \text{Graph 4} \end{array} \right\}. \tag{12}$$

The right-hand side of (12) expands into the sum of 12 different graphs. They are marked in the first twenty-four lines of Table 5 by  $\diamond_i, \heartsuit_i, \clubsuit_i$  and  $\spadesuit_i$  for  $1 \leq i \leq 3$ , respectively; by definition, a suit with different values of its subscript  $i$  denotes the  $i$ th cyclic permutation of the ground vertices for the same graph.<sup>8</sup> For example, the symbols  $\diamond_1, \diamond_2, \diamond_3$  mark the three cyclic permutations of arguments in the first term in the right-hand side of (12). The sum of the first two terms in the right-hand side of (12) – marked by  $\diamond_i$  and  $\heartsuit_i$ , respectively – equals the sum of the first two terms in Fig. 4.<sup>9</sup> At the same time, the sum of the last two terms – whose encodings with coefficients  $\pm 1$  are marked by  $\clubsuit_i$  and  $\spadesuit_i$ , respectively – cancels against the contributions from the fourth and sixth terms in solution (11) – with coefficients  $\pm 3$ , also marked by  $\clubsuit_i$  and  $\spadesuit_i$  in the rest of Table 5. In Table 1 we calculate the coefficient of each graph marked by the respective indexed symbol.

Now, in the same way all other similar terms are collected. There remain only 39 terms with nonzero coefficients. One verifies that those 39 terms are none other than the entries of Table 2, that is, realizations of the 39 graphs in the left-hand side of (10). This shows that equation (10) holds for the operator  $\diamond$  contained in (11).  $\square$

<sup>7</sup>The normal form of a graph is obtained by running over the group  $S_5 \times (\mathbb{Z}_2)^5$  of all the relabellings of internal vertices and swaps  $L \rightleftharpoons R$  of orderings at each vertex. (We recall that every swap negates the coefficient of a graph; the permutations from  $S_5$  are responsible for encoding a given topological profile in seemingly “different” ways.) By definition, the *normal form* of a graph is the minimal sequence of five ordered pairs of target vertices viewed as 10-digit base-(3 + 5) numbers. (By convention, the three ordered sinks are enumerated 0, 1, 2 and the internal vertices are the octonary digits 3, . . . , 7.)

<sup>8</sup>By taking a graph, placing it consecutively over three cyclic permutations of its sinks’ content, and bringing the three graph encodings to their normal form, see above, one can obtain an extra sign factor in front of some of these graphs. This is due to a convention about “minimal” graph encoding, not signalling any mismatch in the arithmetic. For example, after the normalization such is the case with the columns in Table 1: each column refers to a cyclic permutation of three arguments and the coefficients in every line would coincide if one encoded the graphs for the last column not using the respective minimal 10-digit octonary numbers. To make all the three coefficients in each line coinciding, it is enough to swap  $L \rightleftharpoons R$  in one internal vertex in every graph from the third column.

<sup>9</sup>We inspect further that no other graphs in Table 5 make any contribution to the coefficients of these two graphs.



TABLE 1. The coefficients of graphs marked by the four suits.

$\diamond_1 :$	$-1$	$\diamond_2 :$	$-1$	$\diamond_3 :$	$+1$
$\heartsuit_1 :$	$-1$	$\heartsuit_2 :$	$-1$	$\heartsuit_3 :$	$+1$
$\clubsuit_1 :$	$-1 - 1 - 1 + 3 = 0$	$\clubsuit_2 :$	$-1 - 1 - 1 + 3 = 0$	$\clubsuit_3 :$	$+1 + 1 + 1 - 3 = 0$
$\spadesuit_1 :$	$-1 - 1 - 1 + 3 = 0$	$\spadesuit_2 :$	$-1 - 1 - 1 + 3 = 0$	$\spadesuit_3 :$	$+1 + 1 + 1 - 3 = 0$

*Remark 8.* Operator (11) is not a unique solution of factorization problem (10). We claim that apart from this sum of 27 Leibniz graphs, there is another solution which consists of 102 Leibniz graphs; it is also linear with respect to the Jacobiator (that is, its realization in the form  $\diamond(\mathcal{P}, \text{Jac}(\mathcal{P}), \text{Jac}(\mathcal{P}))$  is not possible).

DISCUSSION

**Non-triviality.** A flow specified on the space of Poisson bi-vectors by using the Kontsevich graphs can be Poisson cohomology trivial modulo a sum of Leibniz graphs that would vanish identically at any Poisson structure. However, this is not the case of the Kontsevich tetrahedral flow  $\hat{\mathcal{P}} = \mathcal{Q}_{1:6}(\mathcal{P})$ .

**Proposition 5.** There is no 1-vector field  $\mathcal{X}$  encoded over  $N^n$  by the Kontsevich graphs and there is no operator  $\nabla$  encoded using the Leibniz graphs such that

$$\mathcal{Q}_{1:6}(\mathcal{P}) = \llbracket \mathcal{P}, \mathcal{X} \rrbracket + \nabla(\mathcal{P}, \text{Jac}(\mathcal{P})).$$

The claim is established by a run-through over all Kontsevich graphs with three internal vertices and one sink (making an ansatz for  $\mathcal{X}$ ) and all Leibniz graphs (in the operator  $\nabla$ ) with two copies of  $\mathcal{P}$  and one Jacobiator in the internal vertices; all such graphs of both types are taken with undetermined coefficients. The resulting inhomogeneous linear algebraic system has no solution.

**Integrability.** By using the technique of Kontsevich graphs one can proceed with a higher order expansion of the tetrahedral deformation,

$$\mathcal{P} \mapsto \mathcal{P} + \varepsilon \mathcal{Q}_{1:6}(\mathcal{P}) + \varepsilon \mathcal{R}(\mathcal{P}) + \dots + \bar{o}(\varepsilon^d), \quad d \geq 2,$$

for Poisson structures  $\mathcal{P}$ . Assuming that the master-equation holds up to  $\bar{o}(\varepsilon^d)$ ,

$$\begin{aligned} \llbracket \mathcal{P} + \varepsilon \mathcal{Q}_{1:6}(\mathcal{P}) + \varepsilon \mathcal{R}(\mathcal{P}) + \dots + \bar{o}(\varepsilon^d), \mathcal{P} + \varepsilon \mathcal{Q}_{1:6}(\mathcal{P}) + \varepsilon \mathcal{R}(\mathcal{P}) + \dots + \bar{o}(\varepsilon^d) \rrbracket &\doteq \bar{o}(\varepsilon^d) \\ &\text{via } \llbracket \mathcal{P}, \mathcal{P} \rrbracket = 0, \end{aligned} \quad (13)$$

we obtain a chain of linear equations for the higher order expansion terms, namely,

$$2\llbracket \mathcal{P}, \mathcal{R}(\mathcal{P}) \rrbracket + \llbracket \mathcal{Q}_{1:6}(\mathcal{P}), \mathcal{Q}_{1:6}(\mathcal{P}) \rrbracket \doteq 0 \quad \text{via } \llbracket \mathcal{P}, \mathcal{P} \rrbracket = 0, \text{ etc.} \quad (14)$$

A solution consisting of  $\mathcal{R}(\mathcal{P})$  and consecutive terms at higher powers of the deformation parameter<sup>10</sup> can be sought using the same factorization techniques and computer-assisted proof schemes [3] which have been implemented in this paper — whenever such solution actually exists. It is clear that there can be Poisson cohomological obstructions

<sup>10</sup>In every graph at  $\varepsilon^k$  the number of internal vertices is  $3k + 1$ .

to resolvability of cocycle conditions (14). Hence the integrability issue for the Kontsevich tetrahedral flow may be Poisson model-dependent, unlike the universal nature of such deformation’s infinitesimal part.

APPENDIX A. ENCODING OF THE SOLUTION

Let  $\Gamma$  be a labelled Kontsevich graph with  $n$  internal and  $m$  external vertices. We assume the ground vertices of  $\Gamma$  are labelled  $[0, \dots, m - 1]$  and the internal vertices are labelled  $[m, \dots, m + n - 1]$ . We define the *encoding* of  $\Gamma$  to be the *prefix*  $(n, m)$ , followed by a list of *targets*. The list of targets consists of ordered pairs where the  $k$ th pair ( $k \geq 0$ ) contains the two targets of the internal vertex number  $m + k$ .

The expansion of the Schouten bracket  $[[\mathcal{P}, \mathcal{Q}_{a,b}]]$  for the ratio  $a : b = 1 : 6$  depicted in Figure 3 simplifies to a sum of 39 graphs with coefficients  $\pm \frac{1}{4}, \pm \frac{3}{4}$ . The encodings of these graphs, followed by their respective coefficients, are listed in Table 2. The graphs

TABLE 2. Machine-readable encoding of Fig. 3 on p. 367.

1.1	3	5	4	2	0	1	4	6	4	7	4	5	1/4	7.1	3	5	6	2	7	0	1	4	4	5	5	6	3/4
1.2	3	5	4	0	1	2	4	6	4	7	4	5	1/4	7.2	3	5	6	0	7	1	2	4	4	5	5	6	3/4
1.3	3	5	4	1	2	0	4	6	4	7	4	5	1/4	7.3	3	5	6	1	7	2	0	4	4	5	5	6	3/4
2.1	3	5	7	0	3	5	3	6	3	4	1	2	1/4	8.1	3	5	7	2	7	0	1	4	4	5	5	6	3/4
2.2	3	5	7	1	3	5	3	6	3	4	2	0	1/4	8.2	3	5	7	0	7	1	2	4	4	5	5	6	3/4
2.3	3	5	7	2	3	5	3	6	3	4	0	1	1/4	8.3	3	5	7	1	7	2	0	4	4	5	5	6	3/4
3.1	3	5	5	2	0	1	4	6	4	7	4	5	3/4	9.1	3	5	4	2	7	1	0	4	4	5	5	6	-3/4
3.2	3	5	5	0	1	2	4	6	4	7	4	5	3/4	9.2	3	5	4	0	7	2	1	4	4	5	5	6	-3/4
3.3	3	5	5	1	2	0	4	6	4	7	4	5	3/4	9.3	3	5	4	1	7	0	2	4	4	5	5	6	-3/4
4.1	3	5	6	7	0	3	3	4	4	5	1	2	3/4	10.1	3	5	5	2	7	1	0	4	4	5	5	6	-3/4
4.2	3	5	6	7	1	3	3	4	4	5	2	0	3/4	10.2	3	5	5	0	7	2	1	4	4	5	5	6	-3/4
4.3	3	5	6	7	2	3	3	4	4	5	0	1	3/4	10.3	3	5	5	1	7	0	2	4	4	5	5	6	-3/4
5.1	3	5	4	2	7	0	1	4	4	5	5	6	3/4	11.1	3	5	6	2	7	1	0	4	4	5	5	6	-3/4
5.2	3	5	4	0	7	1	2	4	4	5	5	6	3/4	11.2	3	5	6	0	7	2	1	4	4	5	5	6	-3/4
5.3	3	5	4	1	7	2	0	4	4	5	5	6	3/4	11.3	3	5	6	1	7	0	2	4	4	5	5	6	-3/4
6.1	3	5	5	2	7	0	1	4	4	5	5	6	3/4	12.1	3	5	7	2	7	1	0	4	4	5	5	6	-3/4
6.2	3	5	5	0	7	1	2	4	4	5	5	6	3/4	12.2	3	5	7	0	7	2	1	4	4	5	5	6	-3/4
6.3	3	5	5	1	7	2	0	4	4	5	5	6	3/4	12.3	3	5	7	1	7	0	2	4	4	5	5	6	-3/4
														13.1	3	5	6	0	7	3	3	4	4	5	1	2	-3/4
														13.2	3	5	6	1	7	3	3	4	4	5	2	0	-3/4
														13.3	3	5	6	2	7	3	3	4	4	5	0	1	-3/4

are collected into groups of three, consisting of the skew-symmetrization – by a sum

over cyclic permutations – of a single graph. Within the encodings in the groups of three, the lists of targets only differ by a cyclic permutation of the target vertices 0, 1, 2.

TABLE 3. Machine-readable encoding of Fig. 4 on p. 367.

3	5	0	1	2	3	4	6	4	7	4	5	-1/2
3	5	0	4	1	2	4	6	4	7	4	5	-1/2
3	5	0	4	5	6	1	2	5	7	4	5	3/2
3	5	0	1	2	5	6	7	3	4	4	6	3/2
3	5	0	4	5	6	1	2	3	7	3	4	3/2
3	5	0	4	5	6	1	6	2	7	4	5	-3
3	5	0	4	5	6	1	7	5	7	2	4	3
3	5	0	4	1	5	2	6	4	7	4	5	3
3	5	0	4	2	5	6	7	1	4	4	6	-3

Consisting of 8 skew-symmetric terms, the solution (see (11) on p. 371) is encoded in Table 4: the sought-for values of coefficients are written after the encoding of the respective 27 Leibniz graphs. Here the sums over permutations of the ground vertices

TABLE 4. Machine-readable encoding of solution (11) on p. 371.

1.1 3 5 4 6 5 6 3 6 0 1 6 2 -1	6.1 3 5 1 2 3 5 3 6 0 3 6 4 3
2.1 3 5 0 4 1 5 2 3 3 4 6 5 -3	6.2 3 5 0 2 3 5 3 6 1 3 6 4 -3
2.2 3 5 0 4 2 5 1 3 3 4 6 5 3	6.3 3 5 4 6 0 1 3 4 2 4 6 5 -3
3.1 3 5 0 4 1 2 3 4 3 4 6 5 -3	7.1 3 5 1 5 3 5 2 6 0 3 6 4 -3
3.2 3 5 0 1 2 3 3 4 3 4 6 5 -3	7.2 3 5 1 5 3 5 0 6 2 3 6 4 3
3.3 3 5 0 2 1 3 3 4 3 4 6 5 3	7.3 3 5 0 5 3 5 2 6 1 3 6 4 3
4.1 3 5 4 5 1 6 4 6 0 2 6 3 -3	7.4 3 5 2 5 3 5 1 6 0 3 6 4 3
4.2 3 5 4 5 0 6 4 6 1 2 6 3 3	7.5 3 5 2 5 3 5 0 6 1 3 6 4 -3
4.3 3 5 5 6 3 5 2 6 0 1 6 4 -3	7.6 3 5 0 5 3 5 1 6 2 3 6 4 -3
5.1 3 5 1 4 5 6 3 6 0 2 6 3 3	8.1 3 5 1 4 2 5 3 6 0 3 6 4 -3
5.2 3 5 0 4 5 6 3 6 1 2 6 3 -3	8.2 3 5 1 5 2 3 4 6 0 3 6 4 -3
5.3 3 5 5 6 2 3 4 6 0 1 6 4 -3	8.3 3 5 0 4 2 5 3 6 1 3 6 4 3
	8.4 3 5 0 5 2 3 4 6 1 3 6 4 3
	8.5 3 5 4 6 0 5 1 3 2 4 6 5 -3
	8.6 3 5 4 6 1 5 0 3 2 4 6 5 3

are expanded (thus making the 27 Leibniz graphs out of the 8 skew-symmetric groups). In every entry of Table 4, the sum of three graphs in Jacobiator (4) is represented by its first term. For all the in-coming arrows, the vertex 6 is the placeholder for the Jacobiator (again, see (4) on p. 363); in earnest, the Jacobiator contains the internal vertices 6 and 7. This convention is helpful: for every set of derivations acting on the Jacobiator with internal vertices 6 and 7, only the first term is listed, namely the one where each edge lands on 6.

**Example 2.** The first entry of Table 4 encodes a graph containing a three-cycle over internal vertices 3, 4, 5. Issued from each of these three, the other edge lands on the vertex 6: the placeholder for the Jacobiator. This entry is the first term in (11) on p. 371.

**Example 3.** The entry 3.1 is one of three terms produced by the third graph in solution (11); the Jacobiator in this entry is expanded using formula (4), resulting in three terms (by definition). It is easy to see that the first term contains picture (3) from Remark 2 as a subgraph. Hence the polydifferential operator encoded by this graph vanishes due to skew-symmetry. However, the other two terms produced in the entry 3.1 by formula (4) do not vanish by skew-symmetry. Likewise, there is one term vanishing by the same mechanism in the entry 3.2 and in 3.3.

The proof of Theorem 3 amounts to expanding the Leibniz rules on Jacobiators in Table 4 according to the rules above (resulting in Table 5 on p. 377, where the prefix “3 5” of each graph has been omitted for brevity), simplifying by collecting terms, and seeing that one obtains Table 3.

APPENDIX B. PERTURBATION METHOD

In section 2 above, the run-through method gave all the terms at once in the operator  $\diamond$  that establishes the factorization  $[[\mathcal{P}, \mathcal{Q}_{1:6}]] = \diamond(\mathcal{P}, \text{Jac}(\mathcal{P}))$ . At the same time, there is another method to find  $\diamond$ ; the operator  $\diamond$  is then constructed gradually, term after term in (11), by starting with a zero initial approximation for  $\diamond$ . This is the perturbation scheme which we now outline. (In fact, the perturbation method was tried first, revealing the typical graph patterns and their topological complexity.)

The difficulty is that because the condition  $[[\mathcal{P}, \mathcal{Q}_{1:6}]] = 0$  and the Jacobi identity  $[[\mathcal{P}, \mathcal{P}]] = 0$  are valid, it is impossible to factorize one through the other; both are invisible. So, we first make both expressions visible by perturbing the Poisson bi-vector  $\mathcal{P} \mapsto \mathcal{P}_\epsilon = \mathcal{P} + \epsilon\Delta$  in such a way that the tri-vector  $[[\mathcal{P}_\epsilon, \mathcal{Q}_{1:6}(\mathcal{P}_\epsilon)]]$  and the Jacobiator  $[[\mathcal{P}_\epsilon, \mathcal{P}_\epsilon]]$  stop vanishing identically:

$$[[\mathcal{P}_\epsilon, \mathcal{Q}_{1:6}(\mathcal{P}_\epsilon)]] \neq 0 \quad \text{and} \quad [[\mathcal{P}_\epsilon, \mathcal{P}_\epsilon]] \neq 0.$$

To begin with, put  $\diamond := 0$ . Now consider a class of Poisson brackets on  $\mathbb{R}^3$  (cf. [4]) by using the pre-factor  $f(x, y, z)$  and arbitrary function  $g(x, y, z)$  in the formula

$$\{u, v\}_{\mathcal{P}} = f \cdot \det \left( \frac{\partial(g, u, v)}{\partial(x, y, z)} \right);$$

it is helpful to start with some very degenerate dependencies of  $f$  and  $g$  of their arguments (see [1] and [14]). The next step is to perturb the coefficients of the Poisson bracket  $\{\cdot, \cdot\}_{\mathcal{P}}$  at hand; in a similar way, one starts with degenerate dependency of the perturbation  $\Delta$ . The idea is to take perturbations which destroy the validity of Jacobi identity for  $\mathcal{P}_\epsilon$  in the linear approximation in the deformation parameter  $\epsilon$ . It is readily seen that the expansion of (10) in  $\epsilon$  yields the equality

$$[[\mathcal{P}_\epsilon, \mathcal{Q}_{1:6}]](\epsilon) = (\diamond + \bar{o}(1)) ([[ \mathcal{P}_\epsilon, \mathcal{P}_\epsilon ]]) = 2\epsilon \cdot (\diamond + \bar{o}(1)) ([[ \mathcal{P}, \Delta ]]) + (\diamond + \bar{o}(1)) ([[ \mathcal{P}, \mathcal{P} ]]) + \bar{o}(\epsilon).$$

Knowing the left-hand side at first order in  $\epsilon$  and taking into account that  $[[\mathcal{P}, \mathcal{P}]] \equiv 0$  for the Poisson bi-vector  $\mathcal{P}$  which we perturb by  $\Delta$ , we reconstruct the operator  $\diamond$  that

TABLE 5. Expansion of Leibniz rules on Jacobiators in Table 4.

◇ <sub>1</sub>	0 1 2 3 3 6 3 7 3 5	-1		0 4 2 5 6 7 1 3 3 6	-3		0 4 2 5 3 6 3 4 1 6	-3
♣ <sub>1</sub>	0 1 2 3 3 6 3 7 4 5	-1		0 4 5 6 1 3 3 5 2 3	3		0 4 2 5 3 6 1 7 3 4	-3
♣ <sub>1</sub>	0 1 2 3 3 6 3 7 4 5	-1		0 4 5 6 1 3 5 7 2 3	-3		0 4 2 5 3 6 1 4 3 6	3
♠ <sub>1</sub>	0 1 2 3 3 6 4 7 4 5	-1		0 4 5 6 1 7 3 5 2 3	3		0 4 2 5 3 6 3 7 1 4	3
♣ <sub>1</sub>	0 1 2 3 3 6 3 7 4 5	-1		0 4 5 6 1 7 5 7 2 3	-3		0 4 5 6 1 3 2 3 5 6	-3
♠ <sub>1</sub>	0 1 2 3 3 6 4 7 4 5	-1		0 4 1 2 3 4 3 7 4 5	-3		0 4 5 6 2 3 5 7 1 3	-3
♠ <sub>1</sub>	0 1 2 3 3 6 4 7 4 5	-1	♣ <sub>2</sub>	0 4 1 2 3 6 4 7 4 5	3		0 4 5 6 2 3 3 5 1 6	-3
♥ <sub>1</sub>	0 1 2 3 4 6 4 7 4 5	-1		0 4 5 6 1 2 5 7 4 5	3		0 4 5 6 1 7 2 3 3 6	3
◇ <sub>2</sub>	0 4 1 2 4 6 4 7 4 5	-1		0 4 5 6 1 2 5 7 3 5	3		0 4 5 6 1 6 2 3 3 5	-3
♣ <sub>2</sub>	0 4 1 2 3 6 4 7 4 5	-1		0 4 5 6 1 2 3 5 3 5	3		0 4 5 6 2 3 3 7 1 5	3
♣ <sub>2</sub>	0 4 1 2 3 6 4 7 4 5	-1		0 4 5 6 1 2 5 7 3 5	-3		0 4 2 5 1 3 3 4 5 6	3
♠ <sub>2</sub>	0 4 1 2 3 6 3 7 4 5	-1		0 4 1 5 2 6 3 4 3 5	3		0 4 2 5 6 7 1 3 3 4	3
♣ <sub>2</sub>	0 4 1 2 3 6 4 7 4 5	-1		0 4 1 5 2 6 4 7 3 5	-3		0 4 2 5 3 6 3 4 1 5	-3
♣ <sub>2</sub>	0 4 1 2 3 6 3 7 4 5	-1		0 4 5 6 1 6 2 7 4 5	-3		0 4 2 5 1 6 3 7 3 4	-3
♠ <sub>2</sub>	0 4 1 2 3 6 3 7 4 5	-1		0 4 5 6 1 6 2 7 3 5	-3		0 4 2 5 1 6 3 4 3 5	3
♥ <sub>2</sub>	0 4 1 2 3 6 3 7 3 5	-1		0 4 2 5 3 6 1 4 3 6	-3		0 4 2 5 3 6 1 7 3 4	3
◇ <sub>3</sub>	0 2 1 3 3 6 3 7 3 5	1		0 4 2 5 6 7 1 4 3 6	3		0 4 1 5 2 3 3 5 4 6	3
♣ <sub>3</sub>	0 2 1 3 3 6 3 7 4 5	1		0 4 1 5 3 6 2 4 3 6	3		0 4 5 6 1 7 3 7 2 3	3
♣ <sub>3</sub>	0 2 1 3 3 6 3 7 4 5	1		0 4 1 5 6 7 2 4 3 6	-3		0 4 5 6 2 3 3 5 1 4	-3
♠ <sub>3</sub>	0 2 1 3 3 6 4 7 4 5	1		0 4 5 6 1 7 2 5 4 6	-3		0 4 1 5 6 7 2 3 3 6	-3
♣ <sub>3</sub>	0 2 1 3 3 6 3 7 4 5	1		0 4 5 6 1 7 2 5 3 6	-3		0 4 1 5 3 6 2 3 4 6	-3
♠ <sub>3</sub>	0 2 1 3 3 6 4 7 4 5	1		0 4 2 5 1 6 3 4 3 5	-3		0 4 5 6 1 7 2 3 3 6	-3
♠ <sub>3</sub>	0 2 1 3 3 6 4 7 4 5	1		0 4 2 5 1 6 4 7 3 5	3		0 4 1 5 2 3 3 4 5 6	-3
♥ <sub>3</sub>	0 2 1 3 4 6 4 7 4 5	1		0 4 1 5 2 3 3 7 4 5	-3		0 4 1 5 6 7 2 3 3 4	-3
	0 1 2 5 3 6 3 4 3 4	-3		0 4 1 5 2 6 3 7 4 5	-3		0 4 1 5 3 6 3 4 2 5	3
	0 1 2 5 3 6 4 7 3 4	3		0 4 5 6 1 7 5 7 2 4	3		0 4 1 5 2 6 3 7 3 4	3
	0 1 2 5 6 7 3 4 3 4	-3		0 4 5 6 1 7 5 7 2 3	3		0 4 1 5 2 6 3 4 3 5	-3
	0 1 2 5 6 7 3 4 4 6	3		0 4 5 6 1 6 2 3 3 5	3		0 4 1 5 3 6 2 7 3 4	-3
	0 4 1 5 2 6 4 7 4 5	3		0 4 5 6 1 6 2 7 3 5	3		0 4 2 5 1 3 3 5 4 6	-3
	0 4 1 5 2 6 4 7 3 5	3		0 4 2 5 1 3 3 7 4 5	3		0 4 5 6 2 7 3 7 1 3	-3
	0 4 1 5 2 6 3 7 4 5	3		0 4 2 5 1 6 3 7 4 5	3		0 4 5 6 1 3 3 5 2 4	3
	0 4 1 5 2 6 3 7 3 5	3		0 4 5 6 2 7 5 7 1 4	-3		0 4 2 5 6 7 1 3 3 6	3
	0 4 2 5 3 6 3 4 1 3	-3		0 4 5 6 2 7 5 7 1 3	-3		0 4 2 5 3 6 1 3 4 6	3
	0 4 2 5 3 6 4 7 1 3	3		0 4 5 6 1 3 2 5 3 6	3		0 4 5 6 1 3 2 7 3 5	-3
	0 4 2 5 6 7 1 3 3 4	-3		0 4 5 6 1 7 2 5 3 6	3		0 1 2 3 3 4 3 5 4 6	3
	0 4 2 5 6 7 1 3 4 6	3		0 4 5 6 1 2 3 5 3 5	-3	♣ <sub>1</sub>	0 1 2 3 3 6 3 7 4 5	3
	0 1 2 3 3 4 3 7 4 5	-3		0 4 5 6 1 2 3 7 3 5	-3		0 1 2 5 3 6 3 7 3 5	-3
	0 1 2 5 3 6 4 7 3 4	-3		0 4 5 6 3 7 3 7 1 2	-3		0 1 2 5 3 6 3 7 3 4	-3
♠ <sub>1</sub>	0 1 2 3 3 6 4 7 4 5	3		0 4 5 6 3 6 3 7 1 2	3		0 1 2 5 3 6 3 4 3 4	3
	0 1 2 5 3 6 4 7 4 5	3		0 4 5 6 2 3 3 5 1 5	-3		0 1 2 5 3 6 3 7 3 4	3
	0 4 1 5 6 7 2 4 4 6	3		0 4 5 6 2 3 3 7 1 5	-3		0 4 1 5 3 6 2 3 4 6	3
	0 4 1 5 6 7 2 3 4 6	3		0 4 5 6 1 7 3 7 2 3	-3		0 4 1 5 3 6 4 7 2 3	3
	0 4 1 5 6 7 2 4 3 6	3		0 4 5 6 1 7 3 5 2 3	-3		0 4 1 5 3 6 3 4 2 6	3
	0 4 1 5 6 7 2 3 3 6	3		0 4 5 6 1 3 3 5 2 5	3		0 4 1 5 3 6 2 7 3 4	3
	0 4 5 6 2 3 3 5 1 3	-3		0 4 5 6 1 3 3 7 2 5	3		0 4 1 5 3 6 2 4 3 6	-3
	0 4 5 6 2 7 3 5 1 3	-3		0 4 5 6 2 7 3 7 1 3	3		0 4 1 5 3 6 3 7 2 4	-3
	0 4 5 6 2 3 5 7 1 3	3		0 4 5 6 2 7 3 5 1 3	3		0 4 5 6 1 3 2 5 3 6	-3
	0 4 5 6 2 7 5 7 1 3	3		0 4 1 2 3 4 3 5 4 6	3		0 4 5 6 1 3 3 7 2 5	-3
	0 2 1 5 3 6 3 4 3 4	3	♠ <sub>2</sub>	0 4 1 2 3 6 3 7 4 5	3		0 4 5 6 1 3 3 5 2 6	3
	0 2 1 5 3 6 4 7 3 4	-3		0 4 5 6 1 2 3 7 3 5	3		0 4 5 6 1 3 2 7 3 5	3
	0 2 1 5 6 7 3 4 3 4	3		0 4 5 6 1 2 3 7 3 4	3		0 4 5 6 1 3 2 3 5 6	3
	0 2 1 5 6 7 3 4 4 6	-3		0 4 2 5 3 6 3 4 1 4	-3		0 4 5 6 1 3 5 7 2 3	3
	0 4 2 5 1 6 4 7 4 5	-3		0 4 2 5 3 6 3 7 1 4	-3		0 1 2 3 3 4 3 4 5 6	-3
	0 4 2 5 1 6 4 7 3 5	-3		0 4 1 5 2 6 3 7 3 5	-3		0 1 2 3 3 4 3 7 4 5	3
	0 4 2 5 1 6 3 7 4 5	-3		0 4 1 5 2 6 3 7 3 4	-3		0 1 2 3 3 4 3 5 4 6	-3
	0 4 2 5 1 6 3 7 3 5	-3		0 4 1 5 3 6 3 4 2 4	3		0 2 1 3 3 4 3 4 5 6	3
	0 4 1 5 3 6 3 4 2 3	3		0 4 1 5 3 6 3 7 2 4	3		0 2 1 3 3 4 3 7 4 5	-3
	0 4 1 5 3 6 4 7 2 3	-3		0 4 2 5 1 6 3 7 3 5	3		0 2 1 3 3 4 3 5 4 6	3
	0 4 1 5 6 7 2 3 3 4	3		0 4 2 5 1 6 3 7 3 4	3		0 4 1 2 3 4 3 4 5 6	-3
	0 4 1 5 6 7 2 3 4 6	-3		0 2 1 3 3 4 3 5 4 6	-3		0 4 1 2 3 4 3 7 4 5	3
	0 2 1 3 3 4 3 7 4 5	3	♣ <sub>3</sub>	0 2 1 3 3 6 3 7 4 5	-3		0 4 1 2 3 4 3 5 4 6	-3
♠ <sub>3</sub>	0 2 1 3 3 6 4 7 4 5	-3		0 2 1 5 3 6 3 7 3 5	3		0 4 1 5 2 3 3 4 5 6	3
	0 2 1 5 3 6 4 7 3 4	3		0 2 1 5 3 6 3 7 3 4	3		0 4 1 5 2 3 3 7 4 5	3
	0 2 1 5 3 6 4 7 4 5	-3		0 2 1 5 3 6 3 4 3 4	-3		0 4 1 5 2 3 3 5 4 6	-3
	0 4 2 5 6 7 1 4 4 6	-3		0 2 1 5 3 6 3 7 3 4	-3		0 4 2 5 1 3 3 4 5 6	-3
	0 4 2 5 6 7 1 4 3 6	-3		0 4 2 5 3 6 1 3 4 6	-3		0 4 2 5 1 3 3 7 4 5	-3
	0 4 2 5 6 7 1 3 4 6	-3		0 4 2 5 3 6 4 7 1 3	-3		0 4 2 5 1 3 3 5 4 6	3

now acts on the known tri-vector  $2\llbracket\mathcal{P}, \Delta\rrbracket$ . In this sense, the Jacobiator  $\llbracket\mathcal{P}, \mathcal{P}\rrbracket$  shows up through the term  $\llbracket\mathcal{P}, \Delta\rrbracket$ .

For each pair  $(\mathcal{P}, \Delta)$ , the above balance at  $\epsilon^1$  contains sums over indexes that mark the derivatives falling on the Jacobiator. By taking those formulae, we guess the candidates for graphs that form the next, yet unknown, part of the operator  $\diamond$ . Specifically, we inspect which differential operator(s), acting on the Jacobi identity, become visible and we list the graphs that provide such differential operators via the Leibniz rule(s). For a while we keep every such candidate with an undetermined coefficient. By repeating the iteration, now for a different Poisson bi-vector  $\mathcal{P}$  or its new, less degenerate perturbation  $\Delta$ , we obtain linear constraints for the already introduced undetermined coefficients. Simultaneously, we continue listing the new candidates and introducing new coefficients for them.

*Remark 9.* By translating formulae into graphs, we convert the dimension-dependent expressions into the dimension-independent operators which are encoded by the graphs. An obvious drawback of the method which is outlined here is that, presumably, some parts of the operator  $\diamond$  could always stay invisible for all Poisson structures over  $\mathbb{R}^3$  if they show up only in the higher dimensions. Secondly, the number of variants to consider and in practice, the number of irrelevant terms, each having its own undetermined coefficient, grows exponentially at the initial stage of the reasoning.

By following the loops of iterations of this algorithm, we managed to find two non-zero coefficients and five zero coefficients in solution (11). Namely, we identified the coefficient  $\pm 1$  for the tripod, which is the first term in (11), and we also recognized the coefficient  $\pm 3$  of the sum of ‘elephant’ graphs, which is the second to last term in (11).

*Remark 10.* Because of the known skew-symmetry of the tri-vector  $\llbracket\mathcal{P}, \mathcal{Q}_{1:6}\rrbracket$  with respect to its arguments  $f, g, h$ , finding one term in a sum within formula (11) for  $\diamond$  means that the entire such sum is reconstructed. Indeed, one then takes the sum over a subgroup of  $S_3$  acting on  $f, g, h$ , depending on the actual skew-symmetry of the term which has been found.

For instance, the first term in (11), itself making a sum running over  $\{\text{id}\} \prec S_3$ , is obviously totally antisymmetric with respect to its arguments. The other graph which we found by using the perturbation method (see the last graph in the second line of formula (11) on p. 371) is skew-symmetric with respect to its second and third arguments but it is not yet totally skew-symmetric with respect to the full set of its arguments. This shows that it suffices to take the sum over the group  $\circlearrowleft = A_3 \prec S_3$  of cyclic permutations of  $f, g, h$ , thus reconstructing the sixth term in solution (11).

*Acknowledgements.* A. K. thanks M. Kontsevich for posing the problem; the authors are grateful to P. Vanhaecke and A. G. Sergeev for stimulating discussions. The authors are profoundly grateful to the referee for constructive criticism and advice.

This research was supported in part by JBI RUG project 106552 (Groningen) and by the IHÉS and MPIM (Bonn), to which A. K. is grateful for warm hospitality. A. B. and R. B. thank the organizers of the 8<sup>th</sup> international workshop GADEIS VIII on Group Analysis of Differential Equations and Integrable Systems (12–16 June 2016, Larnaca, Cyprus) for partial financial support and warm hospitality. A. B. and R. B. are also grateful to the Graduate School of Science (Faculty of Mathematics and Natural

Sciences, University of Groningen) for financial support. We thank the Center for Information Technology of the University of Groningen for providing access to the Peregrine high performance computing cluster.

## REFERENCES

- [1] *Bouisaghouane A., Kiselev A. V.* (2017) Do the Kontsevich tetrahedral flows preserve or destroy the space of Poisson bi-vectors? *J. Phys.: Conf. Ser.* Proc. XXIV Int. conf. ‘Integrable Systems and Quantum Symmetries’ (14–18 June 2016, CVUT Prague, Czech Republic), to appear, 10 p. (*Preprint arXiv:1609.06677 [q-alg]*)
- [2] *Buring R., Kiselev A. V.* (2017) On the Kontsevich  $\star$ -product associativity mechanism, *PEPAN Letters* **14**:2, 403–407. (*Preprint arXiv:1602.09036 [q-alg]*)
- [3] *Buring R., Kiselev A. V.* (2017) Software modules and computer-assisted proof schemes in the Kontsevich deformation quantization, *Preprint arXiv:1702.00681 [math.CO]*, 44 + xvi p.
- [4] *Grabowski J., Marmo G., Perelomov A. M.* (1993) Poisson structures: towards a classification, *Mod. Phys. Lett.* **A8**:18, 1719–1733.
- [5] *Kiselev A. V.* (2013) The geometry of variations in Batalin–Vilkovisky formalism, *J. Phys.: Conf. Ser.* **474**, Paper 012024, 1–51. (*Preprint 1312.1262 [math-ph]*)
- [6] *Kiselev A. V.* (2016) The calculus of multivectors on noncommutative jet spaces. *Preprint arXiv:1210.0726 (v4) [math.DG]*, 53 p.
- [7] *Kiselev A. V.* (2015) Deformation approach to quantisation of field models, *Preprint IHÉS/M/15/13 (Bures-sur-Yvette, France)*, 37 p.
- [8] *Kiselev A. V., Ringers S.* (2013) A comparison of definitions for the Schouten bracket on jet spaces, *Proc. 6th Int. workshop ‘Group analysis of differential equations and integrable systems’* (18–20 June 2012, Protaras, Cyprus), 127–141. (*Preprint arXiv:1208.6196 [math.DG]*)
- [9] *Kontsevich M.* (1994) Feynman diagrams and low-dimensional topology. First Europ. Congr. of Math. **2** (Paris, 1992), *Progr. Math.* **120**, Birkhäuser, Basel, 97–121.
- [10] *Kontsevich M.* (1995) Homological algebra of mirror symmetry. Proc. Intern. Congr. Math. **1** (Zürich, 1994), Birkhäuser, Basel, 120–139.
- [11] *Kontsevich M.* (1997) Formality conjecture. Deformation theory and symplectic geometry (Ascona 1996, D. Sternheimer, J. Rawnsley and S. Gutt, eds), *Math. Phys. Stud.* **20**, Kluwer Acad. Publ., Dordrecht, 139–156.
- [12] *Kontsevich M.* (2003) Deformation quantization of Poisson manifolds, *Lett. Math. Phys.* **66**:3, 157–216. (*Preprint q-alg/9709040*)
- [13] *Olver P. J.* (1993) Applications of Lie groups to differential equations, *Grad. Texts in Math.* **107** (2nd ed.), Springer–Verlag, NY.
- [14] *Vanhaecke P.* (1996) Integrable systems in the realm of algebraic geometry, *Lect. Notes Math.* **1638**, Springer–Verlag, Berlin.

## EXTRA REFERENCES

- [15] *Bouisaghouane A.* (2017) The Kontsevich tetrahedral flow in 2D: a toy model, *Preprint arXiv:1702.06044* [math.DG], 6 p.
- [16] *Dubrovin B.* (2005) Bihamiltonian structures of PDE's and Frobenius manifolds, Summer School ICTP, <http://indico.ictp.it/event/a04198/session/47/contribution/26/material/0/0.pdf>.
- [17] *Kiselev A. V.* (2012) The twelve lectures in the (non)commutative geometry of differential equations, *Preprint IHÉS/M/12/13* (Bures-sur-Yvette, France), 140 pp.
- [18] *Kontsevich M.* (1993) Formal (non)commutative symplectic geometry, The Gel'fand Mathematical Seminars, 1990-1992 (L. Corwin, I. Gelfand, and J. Lepowsky, eds), Birkhäuser, Boston MA, 173–187.
- [19] *Laurent–Gengoux C., Picherau A., Vanhaecke P.* (2013) Poisson structures. *Grundlehren der mathematischen Wissenschaften* **347**, Springer–Verlag, Berlin.
- [20] *Manin Yu. I.* (1999) Frobenius manifolds, quantum cohomology, and moduli spaces. *AMS Colloquium publications* **47**, Providence RI.
- [21] *Merkulov S. A.* (2010) Exotic automorphisms of the Schouten algebra of polyvector fields. *Preprint arXiv:0809.2385* (v6) [q-alg], 37 p.
- [22] *Olver P. J., Sokolov V. V.* (1998) Integrable evolution equations on associative algebras, *Comm. Math. Phys.* **193**:2, 245–268;  
*Olver P. J., Sokolov V. V.* (1998) Non-abelian integrable systems of the derivative nonlinear Schrödinger type, *Inverse Prob.* **14**:6, L5–L8.



APPENDIX C. THE CONDITION  $a : b = 1 : 6$  IS NECESSARY (AND MAYBE SUFFICIENT ?)

**Proposition 6** ([1]). The tetrahedral flow  $\dot{P} = \mathcal{Q}_{a:b}(\mathcal{P})$  preserves the property of  $\mathcal{P} + \varepsilon \mathcal{Q}_{a:b}(\mathcal{P}) + \bar{o}(\varepsilon)$  to be (at least infinitesimally) Poisson for all Poisson bi-vectors  $\mathcal{P}$  on all affine real manifolds  $N^n$  *only if* the ratio is  $a : b = 1 : 6$ .

Our proof amounts to producing at least one counterexample when any ratio other than  $1 : 6$  violates equation (2) for a given Poisson bi-vector  $\mathcal{P}$ .

*Proof.* Let  $x, y, z$  be the Cartesian coordinates on  $\mathbb{R}^3$ . Consider the Poisson bracket  $\{u, v\}_{\mathcal{P}} = x \cdot \det(\partial(xyz + y, u, v) / \partial(x, y, z))$  given by the Jacobian, so that the coefficient matrix is

$$\mathcal{P}^{ij} = \begin{pmatrix} 0 & x^2y & -x(xz+1) \\ -x^2y & 0 & xyz \\ -x(xz+1) & -xyz & 0 \end{pmatrix}.$$

The coefficient matrices of both bi-vectors are

$$\Gamma_1(\mathcal{P}) = 6 \cdot \begin{pmatrix} 0 & -x^5y & -x^4(xz+1) \\ x^5y & 0 & -x^3y \\ x^4(xz+1) & x^3y & 0 \end{pmatrix}, \quad \Gamma_2(\mathcal{P}) = \begin{pmatrix} 0 & x^5y & x^4(xz+2) \\ -x^5y & 0 & -2x^3y \\ -x^4(xz+2) & 2x^3y & 0 \end{pmatrix}.$$

It is readily seen that no non-trivial linear combination  $a \cdot \Gamma_1(\mathcal{P}) + b \cdot \Gamma_2(\mathcal{P})$  of the two flows vanishes everywhere on  $\mathbb{R}^3 \ni (x, y, z)$  for this example. Acting on the bi-vectors  $\Gamma_1$  and  $\Gamma_2$  by the Poisson differential  $[[\mathcal{P}, \cdot]]$ , we obtain two tri-vectors which are completely determined by one component each. Namely, we have that

$$[[\mathcal{P}, \Gamma_1(\mathcal{P})]]^{123} = 36x^6yz + 48x^5y, \quad [[\mathcal{P}, \Gamma_2(\mathcal{P})]]^{123} = -6x^6yz - 8x^5y.$$

Clearly, the balance  $a : b = 1 : 6$  is the only ratio at which the non-trivial linear combination  $\mathcal{Q}_{a:b}(\mathcal{P}) = a \cdot \Gamma_1(\mathcal{P}) + b \cdot \Gamma_2(\mathcal{P})$  solves the equation  $[[\mathcal{P}, \mathcal{Q}_{a:b}(\mathcal{P})]] \equiv 0$ .  $\square$

In fact, more is known — this time, about the sufficiency of the condition  $a : b = 1 : 6$ . First, let us recall from [4] that on  $\mathbb{R}^3$  with coordinates  $x, y$ , and  $z$  there is a class of Poisson brackets that admit first integrals at least locally:<sup>11</sup>

$$\{u, v\}_{\mathcal{P}} = f \cdot \det \left( \frac{\partial(g, u, v)}{\partial(x, y, z)} \right) \quad \text{for } u, v \in C^\infty(\mathbb{R}^3), \tag{15}$$

where the free parameter  $g$  is a function and the parameter  $f$  is a density so that

$$f(x, y, z) \cdot \det \left( \frac{\partial(g, u, v)}{\partial(x, y, z)} \right) dx dy dz = f(x, y, z) \Big|_{\substack{x=x'(x',y',z') \\ y=y'(x',y',z') \\ z=z'(x',y',z')}} \cdot \det \left( \frac{\partial(g, u, v)}{\partial(x', y', z')} \right) dx' dy' dz'.$$

In any given coordinate system the parameter  $f$  can be chosen freely; then it is recalculated as shown above.

<sup>11</sup>The referee points out that not all the Poisson brackets are given by the Jacobian determinants. Indeed, the function  $g$  in (15) is always a Casimir of such bracket, but there are real Poisson structures on  $\mathbb{R}^3$  which do not have (smooth) Casimirs near all of its points: some point(s) can be singular so that in no neighbourhood of it would a Casimir exist. In fact, no exhaustive description is known for Poisson brackets on  $\mathbb{R}^3$ .

**Proposition 7** ( $\mathbb{R}^3, \{\cdot, \cdot\}_{\mathcal{P}}$ ). The tetrahedral flow  $\dot{\mathcal{P}} = \mathcal{Q}_{1:6}(\mathcal{P})$  does preserve the property of  $\mathcal{P} + \varepsilon \mathcal{Q}_{a:b}(\mathcal{P}) + \bar{o}(\varepsilon)$  to be infinitesimally Poisson for all Poisson structures (15) on  $\mathbb{R}^3$ .

We used Proposition 7 as an heuristic motivation to our main Theorem 3 in which the claim from Proposition 7 is extended to *all* Poisson structures on all finite-dimensional affine real manifolds. Therefore, in hindsight, Proposition 7 above has been proven rigorously as soon as Theorem 3 was established.

To verify the claim in Proposition 7 by direct calculation, it would take years for man still only a few seconds for a computer.<sup>12</sup> A computer-assisted proof of Proposition 7 is realized through running the script in Maple (see below). (All computations are done with the coefficient matrices of bi-vectors at hand. The bi-vectors are computed by using working formulas (8a) and (8b).) For the balanced flow we have:

```
FlowQ := proc (P, y, a, b)
description "Eval flow Q_a:b of q-dim bi-vector P.";
local i, j, q, A, F, G, B, T, C;
q := op(P)[1];
F := proc (i, j, k, l, m, n, p, r) options operator, arrow;
a*(diff(P[i, j], y[k], y[l], y[m]))*(diff(P[k, n], y[p]))
*(diff(P[l, p], y[r]))*(diff(P[m, r], y[n])) end proc;
G := proc (i, j, k, l, m, n, p, r) options operator, arrow;
b*(diff(P[i, j], y[k], y[l]))*(diff(P[k, m], y[n], y[p]))
*(diff(P[n, l], y[r]))*(diff(P[r, p], y[j])) end proc;
B := Array(1 .. q, 1 .. q);
T := combinat:-cartprod([seq([seq(1 .. q)], i = 1 .. 8)]);
while not T[finished] do
C := op(T[nextvalue]());
B[C[1], C[2]] := B[C[1], C[2]]+F(C);
B[C[1], C[5]] := B[C[1], C[5]]+G(C);
end do;
A := Array(1 .. q, 1 .. q);
for i from 1 to q do
for j from 1 to q do
A[i, j] := simplify((1/2)*B[i, j]-(1/2)*B[j, i]);
end do;
end do;
Matrix(A);
end proc;
```

To implement the Schouten bracket of two bi-vectors  $A$  and  $B$ , we use a component expansion (cf. [16]):

$$\llbracket A, B \rrbracket^{ijk} = \sum_{s=1}^n A^{sk} B_s^{ij} + B^{sk} A_s^{ij} + A^{sj} B_s^{ki} + B^{sj} A_s^{ki} + A^{si} B_s^{jk} + B^{si} A_s^{jk},$$

where superscripts and subscripts denote the bi-vector components and partial derivatives with respect to the coordinates  $y^s$ , respectively.

<sup>12</sup>Running the script below took us approximately 5 seconds.

```

SchoutenBracket := proc (A, B, y)
description "Evaluate the Schouten-bracket of A and B.";
local T, t, F, n, res, cnt;
n := op(A)[1];
F := proc (i, j, k) options operator, arrow;
A[s, k]*(diff(B[i, j], y[s]))+B[s, k]*(diff(A[i, j], y[s]))+
A[s, j]*(diff(B[k, i], y[s]))+B[s, j]*(diff(A[k, i], y[s]))+
A[s, i]*(diff(B[j, k], y[s]))+B[s, i]*(diff(A[j, k], y[s])) end proc;
T := combinat:-choose(n, 3);
for t in T do
print([[t[1], t[2], t[3]],simplify(add(F(t[1], t[2], t[3]), s = 1 .. n))]);
end do;
end proc:

```

Finally, the following script provides a computer-assisted proof of Proposition 7.

```

# All 3-dimensional Poisson bi-vectors are of the following form.
> P:=<<0, -f(x,y,z)*(diff(g(x,y,z),z)), f(x,y,z)*(diff(g(x,y,z),y))>|
    <f(x,y,z)*(diff(g(x,y,z),z)), 0, -f(x,y,z)*(diff(g(x,y,z),x))>|
    <-f(x,y,z)*(diff(g(x,y,z),y)), f(x,y,z)*(diff(g(x,y,z),x)), 0>>:
# We evaluate the balanced flow  $Q_{\{1:6\}}$  on the above bi-vector.
> Q:=FlowQ(P, {x,y,z}, 1, 6)
    [Length of output exceeds limit of 1000000]
# If so, let us inspect whether the flow  $Q_{\{1:6\}}$  vanishes.
> LinearAlgebra:-Equal(Q, Matrix(1..3, 1..3, 0))
    false
# Still, let us act on this  $Q_{\{1:6\}}$  by the Poisson differential.
> SchoutenBracket(P, Q, {x,y,z})
    [[1,2,3], 0]

```

This reasoning hints us that the condition  $a : b = 1 : 6$  could be sufficient for equation (2) to hold for all Poisson structures on all finite dimensional affine real manifolds. A rigorous proof of the respective claim in Theorem 3 is provided in section 2.

#### APPENDIX D. THE COUNT OF LEIBNIZ GRAPHS IN FIG. 6

We count all possible differential consequences of the Jacobi identity, that is, we consider the differential operators acting on the Jacobiator. We do this by constructing all possible graphs that encode trivector-valued differential consequences (see Lemma 1 on p. 368). The graphs that encode such differential consequences have 3 ground vertices. The Schouten bracket  $[[\mathcal{P}, \mathcal{Q}_{1:6}(\mathcal{P})]]$  consists of graphs with 5 internal vertices. Since two of these internal vertices are accounted for by the Jacobi identity, there remain 3 spare internal vertices.

First, let the Jacobiator stand, with all its three edges, on the 3 ground vertices. The only freedom that remains is how the 3 free internal vertices act on each other and on the Jacobiator. With its first edge, every free internal vertex can act on itself, on its 2 neighbouring free vertices, or on the Jacobiator; there are 4 possible targets. No second edge can meet the first edge at the same target (as this would yield no contribution due

to the anti-symmetry, which is explained in Remark 2). Hence there are only 3 possible targets for this second edge. Finally, again due to anti-symmetry, every possibility is constructed exactly twice this way. Swapping the targets of the first and second edge only contributes to the sign of the graph. The total number of this type of differential consequence is therefore  $\left(\frac{4\cdot 3}{2}\right)^3 = 216$  graphs. This type of graph is drawn first from the top-left in Figure 6.

Now let the Jacobiator stand on only 2 of the ground vertices. The remaining edge of the Jacobiator has only 3 possible targets, as the third edge cannot fall back onto the Jacobiator itself. One of the free internal vertices acts with an edge on the remaining ground vertex. The other edge has 4 candidates as its target, namely the vertex itself, the neighbouring 2 free internal vertices, and the Jacobiator. The 2 internal vertices not falling on a ground vertex have each  $\frac{4\cdot 3}{2}$  possible targets. The total number of graphs is therefore equal to  $3 \cdot 4 \cdot \left(\frac{4\cdot 3}{2}\right)^2 = 432$ . This type of graph is the second from the top-left in Figure 6.

Next, let the Jacobiator stand on only 1 ground vertex. We distinguish between two cases: namely, the case where 1 free internal vertex stands on both the remaining ground vertices and the case where two different internal vertices act by one edge each on the remaining two ground vertices. These are the third and fourth graphs from the top-left in Figure 6, respectively.

- In the first case, the remaining 2 internal vertices each have  $\frac{4\cdot 3}{2}$  possible targets. The Jacobiator must act with its two remaining free edges on two different targets out of the 3 available, yielding 3 possibilities. The number of graphs in the first case is  $3 \cdot \left(\frac{4\cdot 3}{2}\right)^2 = 108$ .
- For the second case, two internal vertices can each act on themselves, on the neighbouring 2 internal vertices, or on the Jacobiator. With two of its edges, the Jacobiator can act in 3 different ways on the 3 internal vertices. The third internal vertex has  $\frac{4\cdot 3}{2}$  possible targets. This brings the total number of graphs for the second case to  $4 \cdot 4 \cdot \frac{4\cdot 3}{2} \cdot 3 = 288$ .

The last case to consider is where the Jacobiator does not act on any of the ground vertices. Again, since the outgoing edges of the Jacobiator must have different targets, it is clear that the Jacobiator acts in a unique way on all 3 internal vertices. We now distinguish two cases: namely, the case where 1 free internal vertex stands on 2 ground vertices, 1 free internal vertex acts on 1 ground vertex, and 1 free internal vertex falling on no ground vertex, and the second case where each internal vertex acts with one edge on one ground vertex. These two cases are represented by the last 2 graphs in Figure 6, respectively.

- In the first case, there is a free internal vertex with one free edge, which has 4 possible targets. The remaining free internal vertex with two free edges has  $\frac{4\cdot 3}{2}$  possible targets. The total number of graphs for this case is  $4 \cdot \frac{4\cdot 3}{2} = 24$ .
- In the second case, each internal vertex can act on itself, on its 2 neighbouring internal vertices, and on the Jacobiator. This results in a total of  $4^3 = 64$  graphs.

Summarizing, the total number of all trivector-valued Leibniz graphs, linear in the Jacobiator and containing five internal vertices, is 1132.

APPENDIX E. PROPERTIES OF THE FOUND SOLUTION

*Remark 11.* Let us recall that equation (2) yields the linear system of 7,025 inhomogeneous equations for the coefficients of 1132 patterns from Fig. 6. This shows that the algebraic system at hand is extremely overdetermined. Moreover, out of those 1132 admissible totally antisymmetric graphs, solution (11) involves only 8 of them. In this sense, the factorising operator  $\diamond$  in (2) is special; for it expands via (11) over a very low dimensional affine subspace in the affine space of unknowns in that inhomogeneous linear algebraic system.

**Property 1.** The relevant Leibniz graphs, with respect to which the solution  $\diamond(\mathcal{P}, \cdot)$  expands, do not contain tadpoles nor two-cycles (or “eyes”, see Fig. 1 on p. 362).

- None of the arrows that act back on the Jacobiator is issued from any of its arguments.
- In all the graphs the source vertices (if any), on which no arrows fall after all the Leibniz rules are expanded, belong to the Jacobiator (cf. (4) on p. 363).

**Property 2.** The found solution  $\diamond$  does contain the graphs in which two or three arrows fall on the Jacobiator.<sup>13</sup>

It has been explained in [5, 7] that the existence of two or more such arrows falling on the equation  $[[\mathcal{P}, \mathcal{P}]] = 0$  is an obstruction to an extension of the main claim,

$$[[\mathcal{P}, \mathcal{Q}_{1:6}(\mathcal{P})]] \doteq 0 \quad \text{via } [[\mathcal{P}, \mathcal{P}]] = 0, \tag{2}$$

to the infinite-dimensional geometry of jet spaces  $J^\infty(\pi)$  for affine bundles over a manifold  $M^m$  or jet spaces  $J^\infty(M^m \rightarrow N^n)$  of maps from  $M^m$ , and of variational Poisson brackets  $\{, \}_{\mathcal{P}}$  for functionals on such jet spaces (see [13, 17] and [6, 7]). Namely, it can then be that

$$[[\mathcal{P}, \mathcal{Q}_{1:6}(\mathcal{P})]] \not\cong 0 \quad \text{although } [[\mathcal{P}, \mathcal{P}]] \cong 0. \tag{16}$$

We denote here by  $[[, ]]$  the variational Schouten bracket; the variational bi-vector  $\mathcal{Q}_{1:6}$  is constructed from the variational Poisson bi-vector  $\mathcal{P}$  by using techniques from the geometry of iterated variations of functionals (see [5, 6, 7]). An explicit counterexample of (16) is known from [1] for the variational Poisson structure of the Harry Dym partial differential equation.

The reason why the obstruction arises is that in the variational setting, the second and higher order variations of a trivial integral functional  $\text{Jac}(\mathcal{P}) \cong 0$  in the horizontal cohomology can still be nonzero (although its first variation would of course vanish).<sup>14</sup>

*Remark 12.* The eight graphs in (11) represent a *linear* differential operator with respect to the Jacobiator  $\text{Jac}(\mathcal{P})$ . However, a quadratic nonlinearity with respect to the two-vertex argument  $\text{Jac}(\mathcal{P})$  could be hidden in the five-vertex graphs in formula (11), so that it would in fact encode a bi-differential operator  $\diamond(\mathcal{P}, \cdot, \cdot)$ . If this be the

<sup>13</sup>For instance, the first term in  $\diamond$  is the tripod standing on  $\text{Jac}(\mathcal{P})$ .

<sup>14</sup>The same effect has been foreseen for a variational lift of deformation quantisation [12]: it has been argued in [7] why the associativity of noncommutative star-product  $\star = \times + \hbar\{ \cdot, \cdot \}_{\mathcal{P}} + \bar{o}(\hbar)$  can leak and it has been shown in [2] that if it actually does at  $O(\hbar^k)$ , the order  $k$  at which this leak of associativity can occur is high:  $k \geq 4$ .

case, expansion of one or the other copy of the Jacobiator using (4) in such a polydifferential operator  $\diamond(\mathcal{P}, \cdot, \cdot)$  would produce two seemingly distinct linear differential operators  $\diamond(\mathcal{P}, \cdot)$ .

The scenarios to build the bi-linear, bi-differential terms in the operator  $\diamond$  are drawn in Fig. 7 below. We consider – in fact, without any loss of generality – only those eight

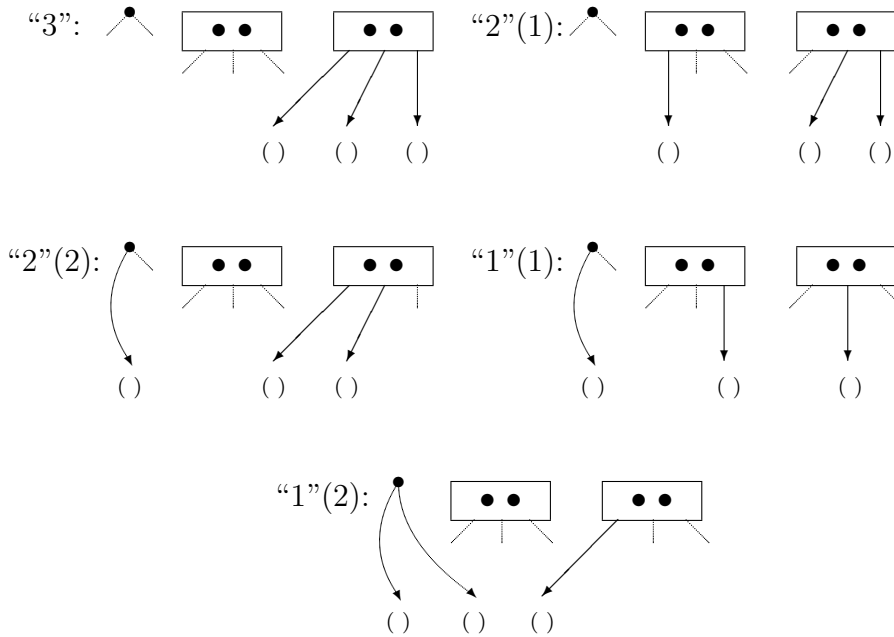


FIGURE 7. The Leibniz graphs by using which a quadratic – with respect to the Jacobiator – part  $\diamond(\mathcal{P}, \cdot, \cdot)$  of the factorizing operator could be sought for in (10); such quadratic part (if any) itself is necessarily totally skew-symmetric with respect to the three sinks.

Leibniz graphs in which

- the three arguments of each copy of Jacobiator (4) are different; in particular,
- neither of the Jacobiators acts on the other copy by two or three arrows (so that only none or one such arrow is possible).

We recall that known solution (11) is the sum of 39 graphs from which a linear dependence on the Jacobiator  $\text{Jac}(\mathcal{P})$  is retrieved by using the 27 Leibniz graphs (see Table 4 on p. 375). Let us inspect whether any solution of equation (10) can be nonlinear in  $\text{Jac}(\mathcal{P})$ ; in particular, let us check whether there is (or is not) a bi-linear dependence in  $\text{Jac}(\mathcal{P})$  hidden in (11).

**Proposition 8.** There is no quadratic part in all the solutions of equation (10).

This claim is supported by a computer-assisted run-through over all Leibniz graphs with linear and with quadratic dependence on the Jacobiator, combined with a requirement that at least one coefficient of those quadratic (in  $\text{Jac}(\mathcal{P})$ ) Leibniz graphs be nonzero. There is no solution.

APPENDIX F. OPEN PROBLEMS

**F.1.** For the factorisation  $[[\mathcal{P}, \mathcal{Q}_{1.6}(\mathcal{P})]] = \diamond(\mathcal{P}, \text{Jac}(\mathcal{P}))$  to guarantee that the equality  $\partial_{\mathcal{P}}(\mathcal{Q}_{1.6}(\mathcal{P})) = 0$  holds if  $\text{Jac}(\mathcal{P}) = 0$ , its mechanism is nontrivial. Relying on Lemma 1 (see [2]), it tells us how the differential consequences of Jacobi identity are split into separately vanishing expressions. This mechanism works not only in the construction of flows that satisfy (2) but also in the associativity,

$$\text{Assoc}_{\mathcal{P}}(f, g, h) := (f \star g) \star h - f \star (g \star h) \doteq 0 \quad \text{via } [[\mathcal{P}, \mathcal{P}]] = 0,$$

of the non-commutative unital star-product  $\star = \times + \hbar\{\cdot, \cdot\}_{\mathcal{P}} + o(\hbar)$ . The formula for  $\star$ -products was given in [12], establishing the deformation quantisation  $\times \mapsto \star$  of the usual product  $\times$  in the algebra  $C^\infty(N^n) \ni f, g, h$  on a finite-dimensional affine Poisson manifold  $(N^n, \mathcal{P})$ , see also [2, 7]. In fact, the construction of graph complex and the pictorial language of graphs [11, 12] that encode polydifferential operators is common to all these deformation procedures (cf. [3], also [21]).

**Open problem 1.** Consider the Kontsevich star-product  $\star = \times + \hbar\{\cdot, \cdot\}_{\mathcal{P}} + o(\hbar)$  in the algebra  $C^\infty(N^n)[[\hbar]]$  on a finite-dimensional affine Poisson manifold  $(N^n, \mathcal{P})$ . Given by the tetrahedra  $\Gamma_1$  and  $\Gamma'_2$  (see Fig. 2 on p. 366), the infinitesimal deformation  $\mathcal{P} \mapsto \mathcal{P} + \varepsilon\mathcal{Q}_{1.6}(\mathcal{P}) + o(\varepsilon)$  induces the infinitesimal deformation  $\star \mapsto \star + \hbar\varepsilon[[[\mathcal{Q}_{1.6}(\mathcal{P}), \cdot]], \cdot] + o(\varepsilon)$  of the star-product. What are the properties of this infinitesimally deformed  $\star(\varepsilon)$ -product? In particular, is the condition that  $\mathcal{Q}_{1.6}(\mathcal{P})$  be  $\partial_{\mathcal{P}}$ -trivial necessary for the  $\star(\varepsilon)$ -product to be gauge-equivalent to the unperturbed  $\star$ -product at  $\varepsilon = 0$ ?

We recall that the theory of (infinitesimal) deformations of associative algebra structures is very well studied in the broadest context (e.g., of the Yang–Baxter equation, Witten–Dijkgraaf–Verlinde–Verlinde (WDVV) equation, Frobenius manifolds and F-structures, etc.), see [16, 20]. We expect that in that theory’s part which is specific to the deformation of associative structures on finite-dimensional affine Poisson manifolds  $N^n$ , there must be a dictionary between the construction of Kontsevich flows for spaces of Poisson bi-vectors and other instruments to deform the associative product in the algebra  $C^\infty(N^n)$ .

**F.2.** The Kontsevich tetrahedral flow  $\dot{\mathcal{P}} = \mathcal{Q}_{1.6}(\mathcal{P})$  is a universal procedure to deform a given Poisson bi-vector  $\mathcal{P}$  on any finite-dimensional affine real manifold  $N^n$  (i. e. not necessarily topologically trivial). For consistency, let us recall that generally speaking, not every infinitesimal deformation  $\mathcal{P} \mapsto \mathcal{P} + \varepsilon\mathcal{Q} + \bar{o}(\varepsilon)$  of a Poisson bi-vector  $\mathcal{P}$  can be completed to a Poisson deformation  $\mathcal{P} \mapsto \mathcal{P} + \mathcal{Q}(\varepsilon)$  at all orders in  $\varepsilon$ . The obstructions are contained in the third  $\partial_{\mathcal{P}}$ -cohomology group  $H^3_{\partial_{\mathcal{P}}} = \{T \in \Gamma(\wedge^3 TN) \mid \partial_{\mathcal{P}}(T) = 0\} / \{T = \partial_{\mathcal{P}}(R), R \in \Gamma(\wedge^2 TN)\}$ . Indeed, cast the master-equation  $[[\mathcal{P} + \mathcal{Q}(\varepsilon), \mathcal{P} + \mathcal{Q}(\varepsilon)]] = 0$  for the Poisson deformation to the coboundary statement  $[[\mathcal{Q}(\varepsilon), \mathcal{Q}(\varepsilon)]] = \partial_{\mathcal{P}}(-\mathcal{P} - 2\mathcal{Q}(\varepsilon))$ , whence  $\partial_{\mathcal{P}}([[ \mathcal{Q}(\varepsilon), \mathcal{Q}(\varepsilon) ]]) \equiv 0$  by  $\partial_{\mathcal{P}}^2 = 0$ . Therefore, the vanishing of the third  $\partial_{\mathcal{P}}$ -cohomology group guarantees the existence of a power series solution  $\mathcal{Q}(\varepsilon)$  to the cocycle-coboundary equation  $[[\mathcal{Q}(\varepsilon), \mathcal{Q}(\varepsilon)]] = -2\partial_{\mathcal{P}}(\mathcal{Q}(\varepsilon))$ : known to be a cocycle, the left-hand side has been proven to be a coboundary as well. (In other words, an infinitesimal deformation  $\mathcal{P} \mapsto \mathcal{P} + \varepsilon\mathcal{Q}_{1.6}(\mathcal{P}) + o(\varepsilon)$  can be completed to the construction of Poisson bi-vector  $\mathcal{P}(\varepsilon)$  such that  $\mathcal{P}(\varepsilon = 0) = \mathcal{P}$  and

$\frac{d}{d\varepsilon}\Big|_{\varepsilon=0} \mathcal{P}(\varepsilon) = \mathcal{Q}_{1:6}(\mathcal{P})$  if the third Poisson cohomology group  $H^3_{\mathcal{P}}(N^n)$  with respect to the Poisson differential  $\partial_{\mathcal{P}} = \llbracket \mathcal{P}, \cdot \rrbracket$  vanishes for the manifold  $N^n$ .)

In the symplectic case, i. e. for  $n$  even and bracket  $\{\cdot, \cdot\}_{\mathcal{P}}$  nondegenerate, the Poisson complex is known to be isomorphic to the de Rham complex for  $N^n$  (see [19]). We are not yet aware of any way to constrain the Poisson cohomology groups  $H^k_{\mathcal{P}}(N^n)$  for *degenerate* Poisson brackets  $\{\cdot, \cdot\}_{\mathcal{P}}$  on real manifolds  $N^n$  of not necessarily even dimension  $n < \infty$ . (E.g., the algorithm for construction of cubic Poisson brackets on the basis of a class of  $R$ -matrices, which is explained in [19], yields a rank-six bracket on  $N^9 \subset \mathbb{R}^9$ .)

**F.3.** The second Poisson cohomology group  $H^2_{\mathcal{P}}(N^n)$  of the manifold  $N^n$ , if nonzero, provides room for the  $\partial_{\mathcal{P}}$ -nontrivial deformations of  $\mathcal{P}$  using  $\mathcal{Q}_{1:6}(\mathcal{P})$  such that  $\mathcal{Q}_{1:6}(\mathcal{P}) \neq \llbracket \mathcal{P}, \mathcal{X} \rrbracket$  for all globally defined 1-vectors  $\mathcal{X}$  on  $N^n$ . In particular, this implies that there are no  $\partial_{\mathcal{P}}$ -nontrivial tetrahedral graph flows on even-dimensional star-shaped domains equipped with nondegenerate Poisson brackets.

A possibility for the right-hand side  $\mathcal{Q}_{1:6}(\mathcal{P})$  of the tetrahedral flow to be  $\partial_{\mathcal{P}}$ -trivial is thus a global, topological effect; it cannot always be seen within a single chart in  $N^n$ . Moreover, it is not universal with respect to the calculus of graphs.

*Remark 13.* Kontsevich notes [11] that if  $n = 2$  so that every bi-vector  $\mathcal{P}$  on  $N^2$  is Poisson and every flow  $\dot{\mathcal{P}} = \mathcal{Q}_{a:b}(\mathcal{P})$  preserves this property, the tetrahedron  $\Gamma_1$  (or, equivalently, the velocity  $\mathcal{Q}_{1:0}(\mathcal{P})$ ) is always  $\partial_{\mathcal{P}}$ -exact. The required 1-vector field  $\mathcal{X}(\mathcal{P})$  in the coboundary statement  $\mathcal{Q}_{1:0}(\mathcal{P}) = \llbracket \mathcal{P}, \mathcal{X} \rrbracket$  can be expressed in terms of the bi-vector  $\mathcal{P}$ , e.g., by the Leibniz-rule graph  $\mathcal{X} = \textcircled{\text{graph}}$ . (This is a particular, not general solution.) We recall that after the dimension  $n$  is fixed (here  $n = 2$ ), a given differential polynomial in  $\mathcal{P}$  can be encoded by the Kontsevich graphs in non-unique way (cf. [15] for details).

**Open problem 2.** The formalism developed in [11] suggests that there are, most likely, infinitely many Kontsevich graph flows on the spaces of Poisson bi-vectors on finite-dimensional affine Poisson manifolds. Forming an example  $\mathcal{Q}_{1:6}(\mathcal{P})$  of such a cocycle in the graph complex, the tetrahedra  $\Gamma_1$  and  $\Gamma'_2$  in Fig. 2 are built over four internal vertices. What is or are the next – with respect to the ordering of natural numbers – Poisson cohomology-nontrivial Kontsevich graph cocycle(s) built over five or more internal vertices?

**F.4.** The tetrahedral flow  $\dot{\mathcal{P}} = \mathcal{Q}_{1:6}(\mathcal{P})$  preserves the space  $\{\mathcal{P} \in \Gamma(\wedge^2 TN^n) \mid \llbracket \mathcal{P}, \mathcal{P} \rrbracket = 0\}$  of Poisson bi-vectors; this is guaranteed by Theorem 3 that asserts  $\partial_{\mathcal{P}}(\mathcal{Q}_{1:6}) \doteq 0$  within the (graded-)commutative geometry of finite-dimensional affine real manifolds  $N^n$ .

**Open problem 3.** Does the proven property,

$$\llbracket \mathcal{P}, \mathcal{Q}_{1:6}(\mathcal{P}) \rrbracket \doteq 0 \quad \text{via} \quad \llbracket \mathcal{P}, \mathcal{P} \rrbracket = 0, \tag{2}$$

generalize to the formal noncommutative symplectic supergeometry [18], to the calculus of multivectors performed by using their necklace brackets (see [6] and references therein), and to Poisson structures on the commutative non-associative unital algebras of cyclic words (e. g., see [22])?



## Chapter 16

# Poisson brackets symmetry from the pentagon-wheel cocycle in the graph complex

This chapter is based on the peer-reviewed journal publication *R. Buring, A. V. Kiselev, and N. J. Rutten, Physics of Particles and Nuclei*, **49**(5): Supersymmetry and Quantum Symmetries 2017, 924–928, 2018. (Preprint [arXiv:1712.05259](https://arxiv.org/abs/1712.05259) [math-ph] – 4 p.)

*Commentary.* In reference to Part I of the dissertation, the material of this chapter is used in §5.2, Chapter 6 (§6.2), and §7.4. In the Appendix within this chapter we give the analytic formula of the pentagon-wheel flow on the spaces of Poisson structures. Interestingly, an alternative solution  $\diamond_2$  of the Poisson cocycle factorization problem (via 8691 Leibniz graphs) was found long before the canonical Kontsevich solution  $\diamond_1$  (consisting of only 3876 Leibniz graphs).

# Poisson brackets symmetry from the pentagon-wheel cocycle in the graph complex

R. Buring<sup>\*,‡</sup> A. V. Kiselev<sup>†,§</sup> N. J. Ruiten<sup>†</sup>

E-mail: <sup>‡</sup>rburing@uni-mainz.de, <sup>§</sup>A.V.Kiselev@rug.nl

## Abstract

Kontsevich designed a scheme to generate infinitesimal symmetries  $\dot{\mathcal{P}} = \mathcal{Q}(\mathcal{P})$  of Poisson brackets  $\mathcal{P}$  on all affine manifolds  $M^r$ ; every such deformation is encoded by oriented graphs on  $n + 2$  vertices and  $2n$  edges. In particular, these symmetries can be obtained by orienting sums of non-oriented graphs  $\gamma$  on  $n$  vertices and  $2n - 2$  edges. The bi-vector flow  $\dot{\mathcal{P}} = \text{Or}\bar{\gamma}(\mathcal{P})$  preserves the space of Poisson structures if  $\gamma$  is a cocycle with respect to the vertex-expanding differential in the graph complex.

A class of such cocycles  $\gamma_{2\ell+1}$  is known to exist: marked by  $\ell \in \mathbb{N}$ , each of them contains a  $(2\ell + 1)$ -gon wheel with a nonzero coefficient. At  $\ell = 1$  the tetrahedron  $\gamma_3$  itself is a cocycle; at  $\ell = 2$  the Kontsevich–Willwacher pentagon-wheel cocycle  $\gamma_5$  consists of two graphs. We reconstruct the symmetry  $\mathcal{Q}_5(\mathcal{P}) = \text{Or}\bar{\gamma}_5(\mathcal{P})$  and verify that  $\mathcal{Q}_5$  is a Poisson cocycle indeed:  $[[\mathcal{P}, \mathcal{Q}_5(\mathcal{P})]] \doteq 0$  via  $[[\mathcal{P}, \mathcal{P}]] = 0$ .

Generic classical Poisson brackets  $\mathcal{P}$  can be deformed along no less than countably many directions (in the spaces of bi-vectors) such that they stay Poisson at least infinitesimally and the change of brackets is not necessarily induced by a diffeomorphism along integral curves of a vector field on the Poisson manifold at hand.<sup>1</sup> The use of graphs converts this infinite analytic problem into a set of finite combinatorial problems of finding cocycles  $\gamma \in \ker d$  in the graph complex and orienting them:  $\mathcal{Q}(\mathcal{P}) = \text{Or}\bar{\gamma}(\mathcal{P})$ , see the diagram.

$$\left| \begin{array}{l} \text{cocycles} \in \ker d: \text{ sums of} \\ n\text{-vertex } (2n-2)\text{-edge non-} \\ \text{oriented graphs with} \\ \text{E}(\gamma) = \bigwedge_i e_i \text{ and coeff} \in \mathbb{R} \end{array} \right| \xrightarrow[\text{make skew}]{\text{Or}\bar{\gamma}} \left| \begin{array}{l} \text{sums of Kontsevich graphs } \mathcal{Q} \text{ on} \\ 2 \text{ sinks, } n \text{ internal vertices, and} \\ 2n \text{ edges in } n \times (\overset{\leftarrow}{L} \bullet \overset{\rightarrow}{R}) \text{ with} \\ \text{Left} \prec \text{Right} \end{array} \right| \xrightarrow[\bullet]{\text{put } \mathcal{P} \text{ into}} \left| \begin{array}{l} \text{bi-vector fields} \\ \mathcal{Q}(\mathcal{P}) = \text{Or}\bar{\gamma}(\mathcal{P}): \\ \text{Poisson 2-cocycles} \\ \in \ker \partial_{\mathcal{P}} = [[\mathcal{P}, \cdot]] \end{array} \right|$$

**1. Graph complex theory.** There are several ways to introduce a differential on the space of non-oriented graphs (see [7, 8]). We consider the real vector space of finite non-oriented graphs such that each of them is equipped with a wedge product of edges, i.e. we suppose that for every graph its edges  $e_i$  are enumerated  $I, II, \dots$  and proclaimed parity-odd, so that  $\text{E}(\gamma) := \bigwedge_i e_i$  and  $(\gamma, I \wedge II \wedge III \wedge \dots) = -(\gamma, II \wedge I \wedge III \wedge \dots)$ , etc.

<sup>\*</sup>Mathematical Institute, Johannes Gutenberg University of Mainz, Staudingerweg 9, D-55128 Germany.

<sup>†</sup>Johann Bernoulli Institute for Mathematics & Computer Science, University of Groningen, P.O. Box 407, 9700 AK Groningen, The Netherlands. Partially supported by JBI RUG project 103511 (Groningen). A part of this research was done while R.B. and A.V.K. were visiting at the IHÉS (Bures-sur-Yvette, France) and A.V.K. was visiting at the University of Mainz.

<sup>1</sup>The dilation  $\dot{\mathcal{P}} = \mathcal{P}$  is an example of symmetry for Jacobi identity; we study nonlinear flows  $\dot{\mathcal{P}} = \mathcal{Q}(\mathcal{P})$  which are universal w.r.t. all affine manifolds and should persist under the quantization  $\frac{\hbar}{i}\{\cdot, \cdot\}_{\mathcal{P}} \mapsto [\cdot, \cdot]$ .

Suppose also that all vertices are at least tri-valent (cf. [4, 9]). On this subspace (which we study here), the differential amounts to a blow-up – via the Leibniz rule – of vertices in a graph  $\gamma$ ; every vertex  $v$  at hand is replaced by the new edge  $E$  such that every edge which was incident to  $v$  in  $\gamma$  is now re-directed to one of the two ends of  $E$ . The choice where to direct a given edge does not depend on a similar choice for other such edges, but overall, the valency of either end of  $E$  must be at least two.<sup>2</sup> By construction, the new edge  $E$  is placed firstmost in the wedge product of edges in every graph  $g$  in  $d(\gamma)$ : whenever  $E(\gamma) = I \wedge II \wedge \dots$ , let  $E(g) = E \wedge I \wedge II \wedge \dots$ . Now one has  $d^2 = 0$ .

*Example 1.* Let  $w_4 := \square$ , and let the edge ordering in these graphs be lexicographic:

$$\delta_6 := d \left( \begin{array}{c} \text{1} \quad \text{4} \\ \text{3} \quad \text{6} \\ \text{5} \quad \text{2} \end{array} \right) = 2 \begin{array}{c} \text{5} \quad \text{3} \\ \text{4} \quad \text{6} \\ \text{7} \quad \text{1} \end{array} + 4 \begin{array}{c} \text{5} \quad \text{1} \\ \text{3} \quad \text{6} \\ \text{7} \quad \text{4} \end{array} + 4 \begin{array}{c} \text{7} \quad \text{3} \\ \text{4} \quad \text{6} \\ \text{2} \quad \text{1} \end{array} - 4 \begin{array}{c} \text{2} \quad \text{1} \\ \text{3} \quad \text{6} \\ \text{7} \quad \text{5} \end{array}$$

A flip over a diagonal in  $w_4$  swaps three pairs of edges; 3 is odd, so by this symmetry,  $E(w_4) = -E(w_4)$ , i.e.  $w_4$  is a *zero graph*.<sup>3</sup> By this,  $d(w_4) = 0$ . Because  $d^2 = 0$ , one has  $d(\delta_6) = 0$  for the coboundary  $\delta_6 \in \text{im } d$ . Put  $\gamma_3 := \square$ ; another example of *nontrivial* cocycle,  $\gamma_5 \notin \text{im } d$ , also on  $n$  vertices and  $2n - 2$  edges, is given on p. 392.

The notion of *oriented* Kontsevich graphs from [7] was recalled in [1, 2, 5]. Every such graph is built over  $m$  ordered sinks from  $n$  wedges  $\overset{L}{\leftarrow} \bullet \overset{R}{\rightarrow}$ : each top  $\bullet$  of the wedge is the source of exactly two arrows (which are ordered by Left  $\prec$  Right). Let  $(M^r, \mathcal{P})$  be a real affine Poisson manifold of dimension  $r$ ; let  $x^1, \dots, x^r$  be local coordinates. By decorating each edge with its own summation index that runs from 1 to  $r$ , by identifying every such edge  $\overset{i}{\rightarrow}$  with  $\partial/\partial x^i$  acting on the content of arrowhead vertex, and by placing a copy of the Poisson bivector  $\mathcal{P} = (\mathcal{P}^{ij})$  at the top  $\bullet$  of each wedge  $\overset{i}{\leftarrow} \bullet \overset{j}{\rightarrow}$ , we associate a polydifferential operator (e.g., an  $m$ -vector) with every such graph. The arguments of the operator are contained in the  $m$  respective sinks. The resulting polydifferential operators are differential-polynomial in the coefficients  $\mathcal{P}^{ij}$  of a given Poisson structure  $\mathcal{P}$ . It is known that for  $\mathcal{P}$  Poisson (hence  $[[\mathcal{P}, \mathcal{P}]] = 0$  under the Schouten bracket), its adjoint action  $\partial_{\mathcal{P}} := [[\mathcal{P}, \cdot]]$  is a differential on the space of multi-vectors. One can try finding Poisson cohomology cocycles  $\mathcal{Q} \in \ker \partial_{\mathcal{P}}$  by assuming they are realized using the Kontsevich oriented graphs.

Now let us note that certain sums  $\mathcal{Q}$  of oriented graphs built on two sinks from  $n$  wedges can be obtained by taking all admissible ways to orient graphs  $\gamma$  on  $n$  vertices and  $2n - 2$  edges (clearly, two sinks and two edges into them are added). Moreover, suppose that  $\gamma \in \ker d$  in vertex-edge bi-grading  $(n, 2n - 2)$  is such that this sum of graphs can be oriented to yield a sum of Kontsevich graphs on two sinks,  $n$  internal vertices and  $2n$  edges. Then, in fact, these oriented graphs, taken with suitable coefficients  $\in \mathbb{R}$ , do assemble to a Poisson cocycle  $\mathcal{Q}(\mathcal{P}) \in \ker \partial_{\mathcal{P}}$ . Let this orientation mapping be denoted by  $\text{Or}$  (cf. [7] and [1, 5]).<sup>4</sup>

**2. The pentagon-wheel cocycle.** The mechanism of factorization  $[[\mathcal{P}, \mathcal{Q}(\mathcal{P})]] \doteq 0$  via  $[[\mathcal{P}, \mathcal{P}]] = 0$  for the cocycle condition  $\mathcal{Q}(\mathcal{P}) \in \ker \partial_{\mathcal{P}}$  is known from [2], where it is used in a similar problem of the  $\star$ -product associativity (cf. [3]). In [1] this mechanism is applied to the Kontsevich tetrahedral flow  $\mathcal{Q}_3(\mathcal{P}) = \text{Or}(\gamma_3)(\mathcal{P})$ . Would the mapping  $\text{Or}$  be known, the verification  $\text{Or}(\gamma) \in \ker \partial_{\mathcal{P}}$  is still compulsory (e.g., by using a factorization via the Jacobi identity for  $\mathcal{P}$ ). But for us now, the factorization  $[[\mathcal{P}, \mathcal{Q}_5(\mathcal{P})]] = \diamond(\mathcal{P}, [[\mathcal{P}, \mathcal{P}]])$  is the way to

<sup>2</sup>In earnest, graphs with valency 1 of an end of  $E$  cancel out in the action of this differential  $d$ , cf. [4, 8].

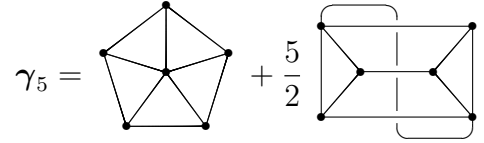
<sup>3</sup>One proves that  $d(\text{zero graph}) = \text{sum of zero graphs and graphs with zero coefficients}$ .

<sup>4</sup>The present paper is aimed to help us reveal the general formula of the morphism  $\text{Or}$  which connects the two graph complexes.

find the right formula of the flow  $\dot{\mathcal{P}} = \mathcal{Q}_5(\mathcal{P})$  that should correspond to the Kontsevich–Willwacher pentagon-wheel cocycle  $\gamma_5$  under the orientation mapping,  $\mathcal{Q}_5 = \text{Or}(\gamma_5)$ , giving one solution  $\mathcal{Q}_5$  yet not necessarily unique operator  $\diamond$ .

*Example 2.* There are only two essentially different admissible ways to orient (and skew-symmetrize with respect to sinks) the tetrahedron  $\gamma_3 \in \ker d$ . Each of the three oriented graphs in the flow  $\mathcal{Q}_3$  is encoded by the list of targets for the ordered pair of edges issued from the  $i$ th vertex ( $m = 2 \leq i \leq 5 = m + n - 1$ ), and a coefficient  $\in \mathbb{Z}$ . Specifically, we have that  $\mathcal{Q}_3 = 1 \cdot (0, 1; 2, 4; 2, 5; 2, 3) - 3 \cdot (0, 3; 1, 4; 2, 5; 2, 3 + 0, 3; 4, 5; 1, 2; 2, 4)$ ; the analytic formula of the respective bi-differential operators acting on the sinks content  $f, g$  is  $\mathcal{Q}_3(f, g) = \partial_{kmp} \mathcal{P}^{ij} \partial_q \mathcal{P}^{kl} \partial_\ell \mathcal{P}^{mn} \partial_n \mathcal{P}^{pq} \cdot \partial_i f \partial_j g - 3 \partial_{mp} \mathcal{P}^{ij} \partial_{jq} \mathcal{P}^{kl} \partial_\ell \mathcal{P}^{mn} \partial_n \mathcal{P}^{pq} \cdot \partial_i f \partial_k g - 3 \partial_{np} \mathcal{P}^{ij} \partial_j \mathcal{P}^{kl} \partial_{kq} \mathcal{P}^{mn} \partial_\ell \mathcal{P}^{pq} \cdot \partial_i f \partial_m g$ . A factorization of  $[[\mathcal{P}, \mathcal{Q}_3(\mathcal{P})]]$  via 8 tri-vector graphs containing  $[[\mathcal{P}, \mathcal{P}]]$  is explained in [1], based on [2].

Now consider the pentagon-wheel cocycle  $\gamma_5 \in \ker d$ , see [4]. By orienting both graphs in  $\gamma_5$  (i.e. by shifting the vertex labelling by  $+1 = m - 1$ , adding two edges



to the sinks 0, 1, and keeping only those oriented graphs out of  $1024 = 2^{\#\text{edges}}$  which are built from  $\leftarrow \bullet \rightarrow$ ) and skew-symmetrizing with respect to  $0 \rightleftharpoons 1$ , we obtain 91 parameters for Kontsevich graphs on 2 sinks, 6 internal vertices, and 12 (= 6 pairs) of edges. We take the sum  $\mathcal{Q}$  of these 91 bi-vector graphs (or skew differences of Kontsevich graphs) with their undetermined coefficients, and for the set of tri-vector graphs occurring in  $[[\mathcal{P}, \mathcal{Q}]]$ , we generate all the possibly needed tri-vector “Leibniz” graphs with  $[[\mathcal{P}, \mathcal{P}]]$  inside.<sup>5</sup> This yields 41031 such Leibniz graphs, which, with undetermined coefficients, provide the ansatz for the r.-h.s. of the factorization problem  $[[\mathcal{P}, \mathcal{Q}(\mathcal{P})]] = \diamond(\mathcal{P}, [[\mathcal{P}, \mathcal{P}]])$ . This gives us an inhomogeneous system of 463,344 linear algebraic equations for both the coefficients in  $\mathcal{Q}$  and  $\diamond$ . In its l.-h.s., we fix the coefficient of one bi-vector graph<sup>6</sup> by setting it to  $+2$ .

**Claim.** For  $\gamma_5$ , the factorization problem  $[[\mathcal{P}, \mathcal{Q}(\mathcal{P})]] = \diamond(\mathcal{P}, [[\mathcal{P}, \mathcal{P}]])$  has a solution  $(\mathcal{Q}_5, \diamond_5)$ ; the sum  $\mathcal{Q}_5$  of 167 Kontsevich graphs (on  $m = 2$  sinks 0, 1 and  $n = 6$  internal vertices 2, ..., 7) with integer coefficients is given in the table below.<sup>7</sup>

0 1 2 4 2 5 3 6 4 7 2 4	10	0 3 4 5 1 2 2 6 2 7 2 4	-10	0 3 1 4 5 6 2 3 2 7 4 5	5	0 3 1 4 2 5 2 6 3 7 2 5	5
0 1 2 4 2 5 2 6 4 7 3 4	-10	0 3 1 4 5 6 2 3 3 7 2 3	-10	0 3 4 5 2 6 4 7 1 2 3 6	5	0 3 4 5 1 2 4 6 4 7 2 5	-5
0 3 1 4 2 5 6 7 2 4 3 4	10	0 3 4 5 2 6 2 7 1 2 2 6	10	0 3 1 4 5 6 2 3 5 7 2 4	-5	0 3 1 4 2 5 2 6 2 7 3 5	5
0 3 4 5 1 2 6 7 2 3 3 4	-10	0 1 2 4 2 5 2 6 2 7 2 3	2	0 3 4 5 1 2 6 7 2 4 4 6	-5	0 3 1 4 5 6 2 6 3 7 2 3	-5
0 3 1 4 2 5 2 6 4 7 3 4	10	0 1 2 4 2 5 2 6 3 7 3 4	-5	0 3 1 4 2 5 6 7 2 3 2 6	-5	0 3 4 5 2 6 4 7 2 7 1 2	-5
0 3 4 5 1 2 4 6 3 7 2 3	-10	0 1 2 4 2 5 3 6 3 7 2 4	5	0 3 1 4 5 6 2 3 5 7 2 3	-5	0 3 1 4 5 6 2 3 2 7 3 4	-5
0 3 1 4 2 5 3 6 4 7 2 4	-10	0 1 2 4 2 5 2 6 3 7 4 5	-5	0 3 4 5 2 6 4 7 1 2 2 6	5	0 3 4 5 2 6 6 7 1 2 2 3	-5
0 3 4 5 1 2 2 6 3 7 3 4	-10	0 1 2 4 2 5 2 6 4 7 3 5	-5	0 3 1 4 2 5 6 7 2 3 3 4	5	0 3 1 4 5 6 2 3 3 7 2 4	-5
0 3 1 4 5 6 2 3 5 7 2 5	-10	0 3 1 4 5 6 2 7 5 7 2 3	5	0 3 4 5 1 2 6 7 2 3 2 4	-5	0 3 4 5 2 6 2 7 1 2 3 6	5
0 3 4 5 2 6 4 7 1 2 4 6	10	0 3 4 5 5 6 6 7 2 7 1 2	5	0 3 1 4 2 5 3 6 4 7 2 3	-5	0 3 1 4 2 5 3 6 2 7 3 5	-5
0 3 4 5 1 6 2 4 5 7 2 5	10	0 3 1 4 2 5 6 7 2 4 3 6	5	0 3 4 5 1 2 2 6 3 7 2 4	-5	0 3 4 5 1 2 2 6 4 7 2 5	5
0 3 4 5 2 6 4 6 1 7 2 4	-10	0 3 4 5 1 2 6 7 2 7 3 4	-5	0 3 1 4 2 5 6 7 2 3 3 6	-5	0 3 1 4 2 5 3 6 3 7 2 5	-5
0 3 4 5 2 6 4 7 2 7 1 4	-10	0 3 1 4 2 5 2 6 3 7 4 5	5	0 3 4 5 1 2 6 7 2 4 2 6	-5	0 3 4 5 1 2 2 6 2 7 4 5	5
0 3 4 5 1 6 2 4 3 7 2 3	10	0 3 4 5 1 2 4 6 2 7 3 5	-5	0 3 4 5 1 2 6 7 2 4 3 4	-5	0 3 4 5 5 6 6 7 1 2 2 6	-5
0 3 4 5 2 6 6 7 1 3 2 3	-10	0 3 1 4 2 5 2 6 4 7 3 5	5	0 3 1 4 2 5 6 7 2 3 2 4	5	0 3 1 4 5 6 2 6 2 7 2 3	5
0 3 4 5 2 6 2 7 1 3 3 6	10	0 3 4 5 1 2 4 6 3 7 2 5	-5	0 3 4 5 1 2 4 6 3 7 2 4	-5	0 1 2 4 2 5 2 6 2 7 3 4	-5
0 3 4 5 1 6 4 7 2 3 2 3	-10	0 3 4 5 1 2 6 7 2 3 4 6	5	0 3 1 4 2 5 2 6 4 7 2 3	-5	0 1 2 4 2 5 2 6 3 7 2 5	-5
0 3 4 5 1 5 2 6 2 7 4 5	10	0 3 1 4 2 5 6 7 2 7 3 4	5	0 1 2 4 2 5 6 7 2 7 3 4	-5	0 1 2 4 2 5 2 6 2 7 3 5	-5
0 3 4 5 1 6 2 7 2 3 3 4	10	0 3 4 5 1 2 2 6 4 7 3 5	5	0 1 2 4 2 5 3 6 2 7 4 5	5	0 3 4 5 2 6 6 7 1 2 4 6	5
0 3 4 5 1 5 2 6 4 7 2 5	10	0 3 1 4 2 5 3 6 2 7 4 5	-5	0 1 2 4 2 5 3 6 4 7 2 5	5	0 3 1 4 5 6 2 3 2 7 2 5	-5
0 3 4 5 1 2 4 6 4 7 2 4	-10	0 3 4 5 1 2 2 6 3 7 4 5	5	0 1 2 4 2 5 3 6 2 7 3 5	5	0 3 4 5 1 2 4 6 4 7 2 3	-5
0 3 1 4 2 5 2 6 2 7 2 3	-10	0 3 1 4 2 5 3 6 4 7 2 5	-5	0 1 2 4 2 5 3 6 3 7 2 5	5	0 3 1 4 2 5 2 6 2 7 3 4	5
0 3 1 4 2 5 3 6 3 7 2 3	-10	0 3 4 5 2 6 6 7 1 2 3 4	5	0 3 4 5 1 2 4 6 2 7 4 5	-5		(see next page)

<sup>5</sup>The algorithm from [5, §1.2] produces 41031 Leibniz graphs in  $\nu = 3$  iterations and 56509 at  $\nu \geq 7$ .

<sup>6</sup>This is done because it is anticipated that, counting the number of ways to obtain a given bi-vector while orienting the nonzero cocycle  $\gamma_5$ , none of the coefficients in a solution  $\mathcal{Q}_5$  vanishes.

<sup>7</sup>The analytic formula of degree-six nonlinear differential polynomial  $\mathcal{Q}_5(\mathcal{P})$  is given in App. A. The encoding of 8691 Leibniz tri-vector graphs containing the Jacobiator  $[[\mathcal{P}, \mathcal{P}]]$  for the Poisson structure  $\mathcal{P}$  that occur in the r.-h.s.  $\diamond(\mathcal{P}, [[\mathcal{P}, \mathcal{P}]])$  is available at <https://rburing.nl/Q5d5.txt>. The machine format to encode such graphs (with one tri-valent vertex for the Jacobiator) is explained in [5] (see also [1, 3]).

0 3 4 5 1 2 2 6 4 7 3 4	-5	0 3 4 5 1 5 6 7 2 4 2 6	5	0 3 4 5 1 6 2 7 2 3 4 6	-5	0 3 4 5 1 6 2 4 2 7 2 5	5
0 3 1 4 2 5 3 6 2 7 2 4	-5	0 3 4 5 2 6 2 7 1 5 3 6	5	0 3 4 5 1 5 2 6 4 7 2 3	5	0 3 4 5 1 6 4 6 2 7 2 4	5
0 3 1 4 5 6 2 3 3 7 2 5	-5	0 3 4 5 1 6 2 6 3 7 2 4	5	0 3 4 5 1 5 2 6 2 7 3 4	-5	0 3 4 5 1 6 2 4 2 7 2 3	5
0 3 4 5 2 6 2 7 1 2 4 6	5	0 3 4 5 2 6 2 6 1 7 3 4	-5	0 3 4 5 1 6 4 7 2 3 2 6	-5	0 3 4 5 2 6 4 7 5 7 1 2	5
0 3 1 4 5 6 2 7 3 7 2 3	-5	0 3 4 5 2 6 4 7 1 5 2 6	-5	0 3 4 5 1 6 2 4 2 7 4 5	-5	0 3 1 4 5 6 2 6 3 7 2 5	5
0 3 4 5 2 6 6 7 2 7 1 2	-5	0 3 4 5 1 6 2 7 2 5 3 4	5	0 3 4 5 1 6 2 7 2 7 2 4	-5	0 3 4 5 2 5 6 7 1 2 4 6	-5
0 3 1 4 2 5 3 6 3 7 2 4	-5	0 3 4 5 1 6 4 7 2 5 2 6	5	0 3 4 5 1 6 2 4 5 7 2 4	5	0 3 1 4 5 6 2 7 3 5 2 6	5
0 3 4 5 1 2 2 6 2 7 3 4	-5	0 3 4 5 1 6 4 7 2 7 2 3	-5	0 3 4 5 2 6 2 6 1 7 2 4	-5	0 3 4 5 2 5 6 7 1 2 3 6	-5
0 3 1 4 2 5 2 6 3 7 3 4	5	0 3 4 5 1 6 4 6 2 7 2 5	5	0 3 4 5 1 5 2 6 4 7 2 4	5	0 3 1 4 5 6 2 7 3 5 2 4	5
0 3 4 5 1 2 4 6 2 7 2 3	-5	0 3 4 5 1 6 2 7 3 5 2 4	-5	0 3 4 5 1 6 2 7 2 3 2 4	5	0 3 4 5 2 6 6 7 3 7 1 2	5
0 3 4 5 1 6 2 7 5 7 2 4	-5	0 3 4 5 2 5 6 7 1 4 2 6	-5	0 3 4 5 1 6 2 4 2 7 3 4	5	0 3 1 4 5 6 2 7 3 7 2 4	5
0 3 4 5 2 6 4 6 1 7 2 5	-5	0 3 4 5 2 6 4 7 2 7 1 3	-5	0 3 4 5 1 6 2 6 2 7 2 4	-5	0 3 4 5 5 6 6 7 1 2 2 3	5
0 3 4 5 1 6 2 7 2 5 4 6	5	0 3 4 5 2 5 6 7 1 3 2 6	-5	0 3 4 5 1 6 2 4 3 7 2 4	5	0 3 1 4 5 6 2 6 2 7 3 4	5
0 3 4 5 1 6 4 7 2 5 2 3	-5	0 3 4 5 2 6 6 7 1 7 2 4	5	0 3 4 5 2 6 2 7 1 5 2 6	5	0 3 4 5 1 2 2 6 4 7 2 4	-5
0 3 4 5 1 6 2 6 2 7 4 5	5	0 3 4 5 1 6 2 4 5 7 2 3	5	0 3 4 5 2 6 6 7 1 3 2 6	-5	0 3 1 4 2 5 3 6 2 7 2 3	-5
0 3 4 5 1 6 2 7 2 7 3 4	5	0 3 4 5 2 6 6 7 2 7 1 4	-5	0 3 4 5 2 6 2 7 1 3 2 6	5	0 3 4 5 2 6 6 7 1 2 2 6	5
0 3 4 5 2 6 6 7 1 7 2 3	-5	0 3 4 5 1 6 2 4 3 7 2 5	5	0 3 4 5 1 6 4 7 2 3 2 4	-5	0 3 1 4 5 6 2 3 2 7 2 3	-5
0 3 4 5 1 5 6 7 2 3 2 4	5	0 3 4 5 2 6 2 7 1 3 4 6	5	0 3 4 5 1 5 2 6 2 7 2 4	-5	0 3 4 5 1 2 4 6 2 7 2 4	-5
0 3 4 5 2 6 4 6 1 7 2 3	-5	0 3 4 5 2 6 6 7 1 3 2 4	-5	0 3 4 5 1 6 4 7 2 7 2 4	5	0 3 1 4 2 5 2 6 3 7 2 3	-5

*Remark.* To establish the formula for the morphism  $\text{Or}$  that would be universal with respect to all cocycles  $\gamma \in \ker d$ , we are accumulating a sufficient number of pairs (d-cocycle  $\gamma$ ,  $\partial_{\mathcal{P}}$ -cocycle  $\mathcal{Q}$ ), in which  $\mathcal{Q}$  is built exactly from graphs that one obtains from orienting the graphs in  $\gamma$ . Let us remember that not only nontrivial cocycles (e.g.,  $\gamma_3$ ,  $\gamma_5$ , or  $\gamma_7$  from [4], cf. [6, 9]) but also d-trivial, like  $\delta_6$  on p. 391, or even the ‘zero’ non-oriented graphs are suited for this purpose: e.g., a unique  $\text{Or}(w_4)(\mathcal{P}) \equiv 0$  constrains  $\text{Or}$ . In every such case, the respective  $\partial_{\mathcal{P}}$ -cocycle is obtained<sup>a</sup> by solving the factorization problem  $[[\mathcal{P}, \mathcal{Q}(\mathcal{P})]] \doteq 0$  via  $[[\mathcal{P}, \mathcal{P}]] = 0$ . The formula of the orientation morphism  $\text{Or}$  will be the object of another paper.

**Acknowledgements.** The authors thank M. Kontsevich and T. Willwacher for recalling the existence of the orientation morphism  $\text{Or}$ . A.V.K. thanks the organizers of international workshop SQS’17 (July 31 – August 5, 2017 at JINR Dubna, Russia) for discussions.<sup>b</sup>

<sup>a</sup>The actually found  $\partial_{\mathcal{P}}$ -cocycle  $\mathcal{Q}$  might differ from the value  $\text{Or}(\gamma)$  by  $\partial_{\mathcal{P}}$ -trivial or improper terms, i.e.  $\mathcal{Q} = \text{Or}(\gamma) + \partial_{\mathcal{P}}(\mathcal{X}) + \nabla(\mathcal{P}, [[\mathcal{P}, \mathcal{P}]])$  for some vector field  $\mathcal{X}$  realized by Kontsevich graphs and for some ‘Leibniz’ bi-vector graphs  $\nabla$  vanishing identically at every Poisson structure  $\mathcal{P}$ .

<sup>b</sup>As soon as the expression of 167 Kontsevich graph coefficients in  $\mathcal{Q}_5$  via the 91 integer parameters was obtained, the linear system in factorization  $[[\mathcal{P}, \mathcal{Q}_5(\mathcal{P})]] = \diamond(\mathcal{P}, [[\mathcal{P}, \mathcal{P}]])$  for the pentagon-wheel flow  $\dot{\mathcal{P}} = \mathcal{Q}_5(\mathcal{P})$  was solved independently by A. Steel (Sydney) using the Markowitz pivoting run in MAGMA. The flow components  $\mathcal{Q}_5$  of all the known solutions  $(\mathcal{Q}_5, \diamond_5)$  match identically. (For the flow  $\dot{\mathcal{P}} = \mathcal{Q}_5(\mathcal{P}) = \text{Or}(\gamma_5)(\mathcal{P})$ , uniqueness is not claimed for the operator  $\diamond$  in the r.-h.s. of the factorization.)

## References

- [1] Bouisaghoulane A., Buring R., Kiselev A. The Kontsevich tetrahedral flow revisited // J. Geom. Phys. 2017. V. 119. P. 272–285.
- [2] Buring R., Kiselev A.V. On the Kontsevich  $\star$ -product associativity mechanism // PEPAN Letters. 2017. V. 14(2). P. 403–407.
- [3] Buring R., Kiselev A.V. Software modules and computer-assisted proof schemes in the Kontsevich deformation quantization. Preprint IHÉS/M/17/05. 2017.
- [4] Buring R., Kiselev A.V., Rutten N.J. The heptagon-wheel cocycle in the Kontsevich graph complex // J. Nonlin. Math. Phys. 2017. V. 24, Suppl. 1. P. 157–173.
- [5] Buring R., Kiselev A.V., Rutten N.J. Infinitesimal deformations of Poisson bi-vectors using the Kontsevich graph calculus // Proc. of Conf. ISQS’25. Prague, June 6–10, 2017. Preprint arXiv:1710.02405
- [6] Dolgushev V. A., Rogers C. L., Willwacher T. H. Kontsevich’s graph complex, GRT, and the deformation complex of the sheaf of polyvector fields // Annals of Math. 2015. V. 182(3). P. 855–943; Willwacher T., Živković M. Multiple edges in M. Kontsevich’s graph complexes and computations of the dimensions and Euler characteristics // Advances of Math. 2015. V. 272. P. 553–578.
- [7] Kontsevich M. Formality conjecture // Proc. of Conf. ‘‘Deformation theory and symplectic geometry’’. Ascona, June 17–21, 1996. Dordrecht: Kluwer Acad. Publ., 1997. P. 139–156; Kontsevich M. Derived Grothendieck–Teichmüller group and graph complexes [after T. Willwacher] // Séminaire Bourbaki (69ème année, Janvier 2017). 2017. No. 1126. P. 1–26.
- [8] Khoroshkin A., Willwacher T., Živković M. Differentials on graph complexes // Advances of Math. 2017. V. 307. P. 1184–1214.

- [9] Willwacher T. M. Kontsevich's graph complex and the Grothendieck–Teichmüller Lie algebra // Invent. Math. 2015. V. 200(3). P. 671–760.















# Chapter 17

## The orientation morphism: from graph cocycles to deformations of Poisson structures

This chapter is based on the peer-reviewed conference proceedings *R. Buring and A. V. Kiselev, J. Phys.: Conf. Ser.* **1194**, Paper 012017, 2019. (Preprint [arXiv:1811.07878](https://arxiv.org/abs/1811.07878) [math.CO] – 10 p.) That paper follows the talk given by the dissertant at the *32nd International colloquium on Group-theoretical methods in Physics: Group32* (9–13 July 2018, CVUT Prague, Czech Republic).

*Commentary.* In reference to Part **I** of the dissertation, the material of this chapter is used in Chapter **4**, Chapter **5**, and Chapter **6** (the Nijenhuis–Richardson bracket shows up in §**6.3**). The explanations in this chapter build on a paper by Jost (2013), which itself follows an outline in a paper by Willwacher (2010–15), which in turn comments on the seminal paper by Kontsevich (1996).

# THE ORIENTATION MORPHISM: FROM GRAPH COCYCLES TO DEFORMATIONS OF POISSON STRUCTURES

R. BURING<sup>‡</sup> AND A. V. KISELEV<sup>§</sup>

ABSTRACT. We recall the construction of the Kontsevich graph orientation morphism  $\gamma \mapsto \text{Or}(\gamma)$  which maps cocycles  $\gamma$  in the non-oriented graph complex to infinitesimal symmetries  $\dot{\mathcal{P}} = \text{Or}(\gamma)(\mathcal{P})$  of Poisson bi-vectors on affine manifolds. We reveal in particular why there always exists a factorization of the Poisson cocycle condition  $[[\mathcal{P}, \text{Or}(\gamma)(\mathcal{P})]] \doteq 0$  through the differential consequences of the Jacobi identity  $[[\mathcal{P}, \mathcal{P}]] = 0$  for Poisson bi-vectors  $\mathcal{P}$ . To illustrate the reasoning, we use the Kontsevich tetrahedral flow  $\dot{\mathcal{P}} = \text{Or}(\gamma_3)(\mathcal{P})$ , as well as the flow produced from the Kontsevich–Willwacher pentagon-wheel cocycle  $\gamma_5$  and the new flow obtained from the heptagon-wheel cocycle  $\gamma_7$  in the unoriented graph complex.

**Introduction.** On an affine manifold  $M^r$ , the Poisson bi-vector fields are those satisfying the Jacobi identity  $[[\mathcal{P}, \mathcal{P}]] = 0$ , where  $[[\cdot, \cdot]]$  is the Schouten bracket ([12], see also Example 1 below). A deformation  $\mathcal{P} \mapsto \mathcal{P} + \varepsilon \mathcal{Q} + \bar{o}(\varepsilon)$  of a Poisson bi-vector  $\mathcal{P}$  preserves the Jacobi identity infinitesimally if  $[[\mathcal{P}, \mathcal{Q}]] = 0$ . If, by assumption, the deformation term  $\mathcal{Q}$  (itself not necessarily Poisson) depends on the bi-vector  $\mathcal{P}$ , then the equation  $[[\mathcal{P}, \mathcal{Q}(\mathcal{P})]] \doteq 0$  must be satisfied by force of  $[[\mathcal{P}, \mathcal{P}]] = 0$ . In [10] Kontsevich designed a way to produce infinitesimal deformations  $\dot{\mathcal{P}} = \mathcal{Q}(\mathcal{P})$  which are *universal* with respect to all Poisson structures on all affine manifolds: for a given bi-vector  $\mathcal{P}$ , the coefficients of bi-vector  $\mathcal{Q}(\mathcal{P})$  are differential polynomial in the coefficients of  $\mathcal{P}$ .

The original construction from [10] goes in three steps, as follows. First, recall that the vector space  $(\text{Gra}_{\# \text{Vert}=:n \geq 1}^{\wedge_i \text{edge}_i})_{S_n}$  of unoriented finite graphs with unlabelled vertices and wedge ordering on the set of edges carries the structure of a complex with respect to the vertex-expanding differential  $d$ . In fact, this space is a differential graded Lie algebra such that the differential  $d$  is the Lie bracket with a single edge,  $d = [\bullet\bullet, \cdot]$ . Let  $\gamma = \sum_i c^i \gamma_i$  be a sum of graphs with  $n$  vertices and  $2n - 2$  edges, satisfying  $d(\gamma) = 0$ . Then let us sum – with signs, which will be discussed in §1.2 below – over all possible ways to orient the graphs  $\gamma_i$  in the cocycle  $\gamma$  such that each vertex is the arrowtail for two outgoing edges; create two extra edges going to two new vertices, the sinks. Secondly, skew-symmetrize (w.r.t. the sinks) the resulting sum of Kontsevich oriented graphs. Finally, insert a Poisson bi-vector  $\mathcal{P}$  into each vertex of every  $\gamma_i$  in the sum of Kontsevich graphs at hand. Now, every oriented graph built of the decorated wedges  $\leftarrow \begin{array}{c} i \\ \bullet \\ \text{Left} \end{array} \bullet \begin{array}{c} j \\ \bullet \\ \text{Right} \end{array} \rightarrow$  determines a differential-polynomial expression in the coefficients  $\mathcal{P}^{ij}(x^1,$

---

*Date:* 2 December 2018.

*2010 Mathematics Subject Classification.* 05C22, 68R10, 16E45, 53D17, 81R60.

<sup>‡</sup>*Address:* Institut für Mathematik, Johannes Gutenberg–Universität, Staudingerweg 9, D-55128 Mainz, Germany. E-mail: [rburing@uni-mainz.de](mailto:rburing@uni-mainz.de).

<sup>§</sup>*Address:* Bernoulli Institute for Mathematics, Computer Science & Artificial Intelligence, University of Groningen, P.O. Box 407, 9700 AK Groningen, The Netherlands. E-mail: [A.V.Kiselev@rug.nl](mailto:A.V.Kiselev@rug.nl).

$\dots, x^r)$  of a bivector  $\mathcal{P}$  whenever the arrows  $\xrightarrow{a}$  denote derivatives  $\partial/\partial x^a$  in a local coordinate chart, each vertex  $\bullet$  at the top of a wedge contains a copy of  $\mathcal{P}$ , and one takes the product of vertex contents and sums up over all the indexes. The right-hand side of the symmetry flow  $\dot{\mathcal{P}} = \mathcal{Q}(\mathcal{P})$  is obtained!

We give an explicit, relatively elementary proof that this recipe does the job, i.e. why the Poisson cocycle condition  $[[\mathcal{P}, \mathcal{Q}(\mathcal{P})]] \doteq 0$  is satisfied for every Poisson structure  $\mathcal{P}$ , and for every  $\mathcal{Q} = \text{Or}(\gamma)$  obtained from a graph cocycle  $\gamma \in \ker d$  in this way. The reasoning is based on that given by Jost [9], which in turn follows an outline by Willwacher [15], itself referring to the seminal paper [10] by Kontsevich.

At the same time, the present text concludes a series of papers [1, 2, 5] with an empiric search for the factorizations  $[[\mathcal{P}, \mathcal{Q}(\mathcal{P})]] = \diamond(\mathcal{P}, [[\mathcal{P}, \mathcal{P}]])$  using the Jacobiator  $[[\mathcal{P}, \mathcal{P}]]$ , as well as containing an independent verification of the numerous rules of signs for many graded objects under study — the ultimate aim being to understand the morphism  $\text{Or}$ .

Section 1.2 establishes the formula<sup>1</sup> of Poisson cocycle factorization through the Jacobiator  $[[\mathcal{P}, \mathcal{P}]]$ :

$$2 \cdot [[\mathcal{P}, \text{Or}(\gamma)(\mathcal{P}, \dots, \mathcal{P})]] = \text{Or}(\gamma)([[\mathcal{P}, \mathcal{P}], \mathcal{P}, \dots, \mathcal{P}) + \dots + \text{Or}(\gamma)(\mathcal{P}, \dots, \mathcal{P}, [[\mathcal{P}, \mathcal{P}]], \mathcal{P}, \dots, \mathcal{P}) + \dots + \text{Or}(\gamma)(\mathcal{P}, \dots, \mathcal{P}, [[\mathcal{P}, \mathcal{P}]]), \quad (1)$$

where the r.-h.s. consists of oriented graphs with one copy of the tri-vector  $[[\mathcal{P}, \mathcal{P}]]$  inserted consecutively into a vertex of the graph(s)  $\gamma$ .

We illustrate the work of orientation morphism  $\text{Or}$  which maps  $\ker d \ni \gamma \mapsto \mathcal{Q}(\mathcal{P}) \in \ker [[\mathcal{P}, \cdot]]$  by using four examples, which include in particular the first elements  $\gamma_3, \gamma_5, \gamma_7 \in \ker d$  of nontrivial graph cocycles found by Willwacher in [15]: the Kontsevich tetrahedral flow  $\dot{\mathcal{P}} = \text{Or}(\gamma_3)(\mathcal{P})$  (see [10] and [1, 2]), the Kontsevich–Willwacher pentagon wheel cocycle  $\gamma_5$  and the respective flow  $\dot{\mathcal{P}} = \text{Or}(\gamma_5)(\mathcal{P})$  (here, see [6] and [15]), and similarly, the heptagon-wheel cocycle  $\gamma_7$  and its flow. In each case, the reasoning reveals a factorization  $[[\mathcal{P}, \text{Or}(\gamma)(\mathcal{P})]] = \diamond(\mathcal{P}, [[\mathcal{P}, \mathcal{P}]])$  through the Jacobi identity  $[[\mathcal{P}, \mathcal{P}]] = 0$ . For the tetrahedral flow  $\dot{\mathcal{P}} = \text{Or}(\gamma_3)(\mathcal{P})$  we thus recover the factorization of  $[[\mathcal{P}, \dot{\mathcal{P}}]]$  – in terms of the “Leibniz” graphs with the tri-vector  $[[\mathcal{P}, \mathcal{P}]]$  inside – which had been obtained in [2] by a brute force calculation. Let it be noted that such factorizations,  $[[\mathcal{P}, \dot{\mathcal{P}}]] = \diamond(\mathcal{P}, [[\mathcal{P}, \mathcal{P}]])$ , are known to be non-unique for a given flow  $\dot{\mathcal{P}}$ ; the scheme which we presently consider provides one such operator  $\diamond$  (out of many, possibly).

Trivial graph cocycles, i.e. d-coboundaries  $\gamma = d(\beta)$  also serve as an illustration. Under the orientation mapping  $\text{Or}$  their “potentials”  $\beta$  (sums of graphs with  $n - 1$  vertices and  $2n - 3$  edges) are transformed into the vector fields  $\mathcal{X}$ , also codified by the Kontsevich oriented graphs, which trivialize the respective flows  $\dot{\mathcal{P}} = \text{Or}(\gamma)(\mathcal{P})$  in the space of bi-vectors: namely,  $\text{Or}(d(\beta))(\mathcal{P}) = [[\mathcal{P}, \text{Or}(\beta)(\mathcal{P})]]$  so that the resulting

---

<sup>1</sup>The existence of this formula with *some* vanishing right-hand side is implied in [10, 15, 8] where it is stated that there is an action of the graph complex on Poisson structures (or Maurer–Cartan elements of  $T_{\text{poly}}(M)$ ). The precise right-hand side is all but written in [9]; still to the best of our knowledge, the exact formula is presented here and on p. 408 below for the first time. — The same applies to Jacobi identity (2) for the Lie bracket of graphs (cf. [14]).

flow  $\dot{\mathcal{P}} = \mathcal{Q}(\mathcal{P}) = \llbracket \mathcal{P}, \mathcal{X}(\mathcal{P}) \rrbracket$  is trivial in Poisson cohomology. We offer an example on p. 409: here,  $\mathcal{X}(\mathcal{P}) = 2\text{Or}(\beta_6)(\mathcal{P})$ .

This paper continues in §1.3 with some statistics about the number of graphs (i) in the “known” cocycles  $\gamma = \sum c^i \gamma_i \in \ker d$ , (ii) in the respective flows  $\mathcal{Q} = \text{Or}(\gamma)$  which consist of the oriented Kontsevich graphs, (iii) in the factorizing operators  $\diamond$  (provided by the proof) which are encoded by the Leibniz graphs (see [3, 2]), and (iv) in the cocycle equations  $\llbracket \mathcal{P}, \text{Or}(\gamma)(\mathcal{P}) \rrbracket \doteq 0$ . We see that for thousands and millions of oriented graphs in the left- and right-hand sides of (1) the coefficients match perfectly.

### 1. THE PARALLEL WORLDS OF GRAPHS AND ENDOMORPHISMS

The universal deformations  $\dot{\mathcal{P}} = \mathcal{Q}(\mathcal{P})$  which we consider will be given by certain endomorphisms evaluated at copies of a given Poisson structure  $\mathcal{P}$ . In particular, the resulting expressions will be differential polynomials in the coefficients of  $\mathcal{P}$ . Moreover, such expressions will be built using graphs, so that properties of objects in the graph complex are translated into properties of the objects realized by the graphs in the Poisson complex. To this end, let us recall and compare the notions of operads of non-oriented graphs and of endomorphisms of multi-vector fields on affine manifolds. This material is standard; we follow [10, 9, 15, 13].

1.1. **Endomorphisms**  $\text{End}(T_{\text{poly}}(M)[1])$  (e.g., the Schouten bracket  $\llbracket \cdot, \cdot \rrbracket$ ). Denote the shifted-graded vector space of all multi-vector fields on the manifold  $M^r$  by<sup>2</sup>

$$T_{\text{poly}}(M)[1] = \bigoplus_{\bar{\ell} \geq -1} T_{\text{poly}}^{\bar{\ell}}(M) \quad \text{where} \quad \ell = \bar{\ell} + 1.$$

The grading in  $T_{\text{poly}}(M)[1] = T_{\text{poly}}^{\downarrow[1]}(M)$  is shifted *down* so that, by definition, a bi-vector  $\mathcal{P}$  has degree  $|\mathcal{P}| = 2$  but  $\bar{\mathcal{P}} = 1$ , etc. We let the multi-vectors be encoded in a standard way using a local coordinate chart  $x^1, \dots, x^r$  on  $M^r$  and the respective parity-odd variables  $\xi_1, \dots, \xi_r$  along the reverse-parity fibres of  $\Pi T^* M^r$  over that chart. For example, a bi-vector is written in coordinates as  $\mathcal{P} = \sum_{1 \leq i < j \leq r} P^{ij}(\mathbf{x}) \xi_i \xi_j$ .<sup>3</sup>

An endomorphism of  $T_{\text{poly}}(M)[1]$  of arity  $k$  and degree  $\bar{d}$  is a  $k$ -linear (over the field  $\mathbb{R}$ ) map  $\theta: T_{\text{poly}}(M)[1] \otimes \dots \otimes T_{\text{poly}}(M)[1] \rightarrow T_{\text{poly}}(M)[1]$ , not necessarily (graded-)skew in its  $k$  arguments, and such that for grading-homogeneous arguments we have that

$$\theta: T_{\text{poly}}^{\bar{d}_1}(M) \otimes \dots \otimes T_{\text{poly}}^{\bar{d}_k}(M) \rightarrow T_{\text{poly}}^{\bar{d}_1 + \dots + \bar{d}_k + \bar{d}}(M),$$

i.e.  $\theta$  restricts to a map of degree  $\bar{d}$ .

*Example 1.* The Schouten bracket  $\llbracket \cdot, \cdot \rrbracket: T_{\text{poly}}^{\bar{d}_1}(M) \otimes T_{\text{poly}}^{\bar{d}_2}(M) \rightarrow T_{\text{poly}}^{\bar{d}_1 + \bar{d}_2}(M)$  has arity 2 and shifted degree  $\overline{\text{deg}}(\llbracket \cdot, \cdot \rrbracket) = 0$  (note  $|\llbracket \cdot, \cdot \rrbracket| = -1$ ). It is expressed in coordinates by the formula

$$\llbracket \mathcal{P}, \mathcal{Q} \rrbracket = \sum_{\ell=1}^r (\mathcal{P}) \frac{\bar{\partial}}{\partial \xi_\ell} \cdot \frac{\bar{\partial}}{\partial x^\ell} (\mathcal{Q}) - (\mathcal{P}) \frac{\bar{\partial}}{\partial x^\ell} \cdot \frac{\bar{\partial}}{\partial \xi_\ell} (\mathcal{Q}).$$

<sup>2</sup>This notation for the space of multi-vectors should not be confused with a similar notation for the space of vector fields with polynomial coefficients on an affine manifold  $M^r$ . Nor should it be read as the space of multi-vectors on a super-manifold.

<sup>3</sup>Our notation is such that the wedge product of multi-vectors does not include any constant factor.



**Notation.** The bi-graded vector space of endomorphisms under study is denoted by

$$\text{End}(T_{\text{poly}}(M)[1]) = \bigoplus_{\substack{\vec{d} \in \mathbb{Z} \\ k \geq 1}} \text{End}^{k, \vec{d}}(T_{\text{poly}}(M)[1]).$$

This space has the structure of an operad (with an action by the permutation group  $S_k$  on the part of arity  $k$ ): indeed, endomorphisms can be inserted one into another.

Let  $\theta_a$  and  $\theta_b$  be two endomorphisms of respective arities  $k_a$  and  $k_b$ . The insertion of  $\theta_a$  into the  $i^{\text{th}}$  argument of  $\theta_b$  is denoted by  $\theta_a \vec{\circ}_i \theta_b$ . For instance,  $(\theta_a \vec{\circ}_1 \theta_b)(p_1, \dots, p_{k_a+k_b-1}) = \theta_b(\theta_a(p_1, \dots, p_{k_a}), p_{k_a+1}, \dots, p_{k_a+k_b-1})$ . Likewise, the notation  $\theta_a \overleftarrow{\circ}_i \theta_b$  means the insertion of the succeeding object  $\theta_b$  into the preceding  $\theta_a$ , whence  $(\theta_a \overleftarrow{\circ}_1 \theta_b)(\mathbf{p}) = \theta_a(\theta_b(p_1, \dots, p_{k_b}), p_{k_b+1}, \dots, p_{k_a+k_b-1})$ . Without an arrow pointing left, this notation  $\vec{\circ}_i$  is used in other papers; it is also natural because the graded objects  $\theta_a$  and  $\theta_b$  are not swapped.

**Definition 1.** The *insertion*  $\vec{\circ}$  of an endomorphism  $\theta_a$  into an endomorphism  $\theta_b$  of arity  $k_b$  is the sum of insertions:  $\theta_a \vec{\circ} \theta_b = \sum_{i=1}^{k_b} \theta_a \vec{\circ}_i \theta_b$ . The graded commutator of endomorphisms of degrees  $d_a$  and  $d_b$  is  $[\theta_a, \theta_b] = \theta_a \vec{\circ} \theta_b - (-)^{|\theta_a| \cdot |\theta_b|} \theta_b \vec{\circ} \theta_a$ . An endomorphism  $\theta$  of arity  $k$  is *skew* with respect to permutations of its graded arguments if it acquires the Koszul sign,  $\theta(p_1, \dots, p_k) = \epsilon_{\mathbf{p}}(\sigma) \theta(p_{\sigma(1)}, \dots, p_{\sigma(k)})$  under  $\sigma \in S_k$ . Here  $\epsilon_{\mathbf{p}}((1\ 2)) = (-)^{(1\ 2)} (-)^{\bar{p}_1 \cdot \bar{p}_2}$  and similarly for all other transpositions which generate the permutation group  $S_k$ . Suppose that both of the endomorphisms  $\theta_a$  and  $\theta_b$  from the above are graded skew-symmetric. The *Nijenhuis–Richardson bracket*  $[\theta_a, \theta_b]_{\text{NR}}$  of those skew endomorphisms (of degrees  $d_a$  and  $d_b$  respectively) is the skew-symmetrization of  $[\theta_a, \theta_b]$  with respect to the permutations, graded by the Koszul signs.

*Example 2.* The shifted-graded skew-symmetric Schouten bracket

$$\pi_S(p_1, p_2) := (-)^{|p_1|-1} [[p_1, p_2]] \in \text{End}^2(T_{\text{poly}}(M)[1])$$

of multivectors  $a, b, c$  of respective homogeneities satisfies the shifted-graded Jacobi identity

$$[[a, [b, c]]] - (-)^{\bar{a}\bar{b}} [[b, [a, c]]] = [[[a, b], c]] = 0,$$

or equivalently,

$$[[a, [b, c]]] + (-)^{\bar{a}\bar{b}+\bar{a}\bar{c}} [[b, [c, a]]] + (-)^{\bar{c}\bar{a}+\bar{c}\bar{b}} [[c, [a, b]]] = 0.$$

Taken four times,  $[\pi_S, \pi_S]_{\text{NR}}$  evaluated (with Koszul signs shifted by  $\text{deg}[[\cdot, \cdot]] = -1$ ) at  $a, b, c$  yields the l.h.s. of the Jacobi identity for  $[[\cdot, \cdot]]$ . This shows that  $[\pi_S, \pi_S]_{\text{NR}} = 0$ .

**Proposition 1.** *The Nijenhuis–Richardson bracket (of homogeneous arguments of respective degrees) itself satisfies the graded Jacobi identity*

$$[a, [b, c]_{\text{NR}}]_{\text{NR}} - (-)^{|a| \cdot |b|} [b, [a, c]_{\text{NR}}]_{\text{NR}} = [[a, b]_{\text{NR}}, c]_{\text{NR}}, \tag{2}$$

or equivalently,

$$[a, [b, c]_{\text{NR}}]_{\text{NR}} + (-)^{|a| \cdot |b| + |a| \cdot |c|} [b, [c, a]_{\text{NR}}]_{\text{NR}} + (-)^{|c| \cdot |a| + |c| \cdot |b|} [c, [a, b]_{\text{NR}}]_{\text{NR}} = 0.$$

**Corollary 1.** *The map  $\partial := [\pi_S, \cdot]_{\text{NR}}$  is a differential on the space of skew endomorphisms.*

**1.2. Graphs vs endomorphisms.** Having studied the natural differential graded Lie algebra (dgLa) structure on the space of graded skew-symmetric endomorphisms  $\text{End}_{\text{skew}}^{*,*}(T_{\text{poly}}(M)[1])$ , we observe that its construction goes in parallel with the dgLa structure on the vector space  $\bigoplus_k (\text{Gra}_{\# \text{Vert}=:k \geq 1}^{\wedge_i \text{edge}_i})_{S_k}$  of finite non-oriented graphs with wedge ordering of edges (and without leaves). Referring to [8, 9, 10, 15] (and references therein), as well as to [6, 7, 14] with explicit examples of calculations in the graph complex, we summarize the set of analogous objects and structures in Table 1 below.

TABLE 1. From graphs to endomorphisms: the respective objects or structures.

World of graphs	World of endomorphisms
Graphs $(\gamma, E(\gamma))$	Endomorphisms
Insertion $\vec{\sigma}_i$ of graph into $i^{\text{th}}$ vertex	Insertion of endomorphism into $i^{\text{th}}$ argument
Insertion $\vec{\sigma}$ of graph into graph	Insertion $\vec{\sigma}$
Bracket $[a, b] = a \vec{\sigma} b - (-)^{ E(a)  \cdot  E(b) } b \vec{\sigma} a$	Bracket $[a, b] = a \vec{\sigma} b - (-)^{ a  \cdot  b } b \vec{\sigma} a$
Lie bracket $([a, b], E([a, b]) := E(a) \wedge E(b))$	Nijenhuis-Richardson bracket $[a, b]_{\text{NR}}$ on the space of skew endomorphisms
The stick $\bullet\!\!\!\bullet$	The Schouten bracket $\pi_S = \pm \llbracket \cdot, \cdot \rrbracket$
Master equation $[\bullet\!\!\!\bullet, \bullet\!\!\!\bullet] = 0$	Master equation $[\pi_S, \pi_S]_{\text{NR}} = 0$
Graded Jacobi identity for $[\cdot, \cdot]$	Graded Jacobi identity for $[\cdot, \cdot]_{\text{NR}}$
Differential $d = [\bullet\!\!\!\bullet, \cdot]$	Differential $\partial = [\pi_S, \cdot]_{\text{NR}}$

The orientation morphism  $\text{Or}$ , which we presently discuss, provides a transition “ $\implies$ ” from graphs to endomorphisms. Our goal is to have a Lie algebra morphism

$$\left\{ \bigoplus_k (\text{Gra}_{\# \text{Vert}=:k \geq 1}^{\wedge_i \text{edge}_i})_{S_k}, \quad d = [\bullet\!\!\!\bullet, \cdot] \right\} \xrightarrow{\text{Or}} \left\{ \text{End}_{\text{skew}}^{*,*}(T_{\text{poly}}(M)[1]), \quad \partial = [\pi_S, \cdot]_{\text{NR}} \right\},$$

hence a dgLa morphism because the differentials  $d = [\bullet\!\!\!\bullet, \cdot]$  and  $\partial = [\pi_S, \cdot]_{\text{NR}}$  are the adjoint actions of the Maurer–Cartan elements.

In the meantime, we claim without proof that the edge  $\bullet\!\!\!\bullet$  is taken to the Schouten bracket  $\pi_S = \pm \llbracket \cdot, \cdot \rrbracket$  by  $\text{Or}$ : namely,  $\bullet\!\!\!\bullet \mapsto \pi_S = \pm \llbracket \cdot, \cdot \rrbracket$  (see (3) below). So, having a Lie algebra morphism implies that  $\text{Or}([\bullet\!\!\!\bullet, \gamma]) = [\pi_S, \text{Or}(\gamma)]_{\text{NR}}$  for a graph  $\gamma$  with edge ordering  $E(\gamma)$ , i.e. the following diagram is commutative:

$$\begin{array}{ccc} (\gamma, E(\gamma)) & \xrightarrow{\text{Or}} & \text{Or}(\gamma) \\ \downarrow d & & \downarrow \partial \\ [\bullet\!\!\!\bullet, \gamma] & \xrightarrow{\text{Or}} & [\pi_S, \text{Or}(\gamma)]_{\text{NR}}. \end{array}$$

When this diagram is reached, it will be seen – by evaluating the endomorphisms at copies of  $\mathcal{P}$  – why the mapping of d-cocycles in the graph complex to Poisson cocycles  $\in \ker \llbracket \mathcal{P}, \cdot \rrbracket$  is well defined. This will solve the problem of producing universal infinitesimal symmetries  $\dot{\mathcal{P}} = \text{Or}(\gamma)(\mathcal{P})$  of Poisson brackets  $\mathcal{P}$  from d-cocycles  $\gamma \in \ker d$ .

Let  $\gamma$  be an unoriented graph on  $k$  vertices and let  $p_1, \dots, p_k \in T_{\text{poly}}(M)$  be a  $k$ -tuple of multivectors. Not yet at the level of Lie algebras but at the level of two operads

with the respective graph- and endomorphism insertions  $\vec{\sigma}$ , let the linear mapping  $\vec{\sigma}$  be given by the formula [10]

$$\vec{\sigma}(\gamma)(p_1, \dots, p_k)(\mathbf{x}, \boldsymbol{\xi}) := \text{mult}_k \left( \prod_{(i,j) \in E(\gamma)} \vec{\Delta}_{ij}(p_1 \otimes \dots \otimes p_k) \right) (\mathbf{x}, \boldsymbol{\xi}),$$

where for each edge  $(i, j) = e_{ij}$  in the graph  $\gamma$ , the operator  $\Delta_{ij}: e_{ij} \mapsto (i \xrightarrow{\ell} j) + (i \xleftarrow{\ell} j)$ ,

$$\vec{\Delta}_{ij} = \sum_{\ell=1}^r \left( \frac{\vec{\partial}}{\partial x_{(j)}^\ell} \frac{\vec{\partial}}{\partial \xi_{(i)}^\ell} + \frac{\vec{\partial}}{\partial \xi_{(j)}^\ell} \frac{\vec{\partial}}{\partial x_{(i)}^\ell} \right),$$

acts on the  $i^{\text{th}}$  and  $j^{\text{th}}$  factors in the ordered tensor product of arguments  $p_1, \dots, p_k$ . By construction, the right-to-left ordering of the operators  $\vec{\Delta}_{ij}$  is inherited from the wedge ordering of edges  $E(\gamma)$  in the graph  $\gamma$ : the operator corresponding to the firstmost edge acts first.<sup>4</sup> The operator  $\text{mult}_k$ , acting at the end of the day, is the ordered multiplication of the resulting terms in  $\prod_{(i,j)} \vec{\Delta}_{ij}(p_1 \otimes \dots \otimes p_k)$ .

It can be seen ([9, 15]) that the graph insertions  $\vec{\sigma}_i$  are mapped by  $\vec{\sigma}$  to the insertions  $\vec{\sigma}_i$  of endomorphisms:  $\vec{\sigma}(\gamma_1 \vec{\sigma}_i \gamma_2) = \vec{\sigma}(\gamma_1) \vec{\sigma}_i \vec{\sigma}(\gamma_2)$ . Consequently, the sum of insertions  $\vec{\sigma}$  goes – under  $\vec{\sigma}$  – to the sum of insertions  $\vec{\sigma}$ . The mapping  $\vec{\sigma}$  induces the linear mapping  $\text{Or}\vec{\sigma}$  taking graphs to the space of graded-skew endomorphisms  $\text{End}_{\text{skew}}(T_{\text{poly}}(M)[1])$ . We reach the important equality:

$$\text{Or}\vec{\sigma}(\bullet\bullet) = \pi_S, \quad \text{i.e.} \quad \text{Or}\vec{\sigma}(\bullet\bullet)(p_1, p_2) = (-)^{\bar{p}_1} \llbracket p_1, p_2 \rrbracket \quad \text{for } p_1, p_2 \in T_{\text{poly}}(M)[1]. \quad (3)$$

Recall also that both the domain and image of  $\text{Or}\vec{\sigma}$ , i.e. graphs with wedge ordering of edges and their skew-symmetrized images in the space  $\text{End}_{\text{skew}}(T_{\text{poly}}(M)[1])$  carry the respective Lie algebra structures. The conclusion is this:

**Proposition 2.** *The mapping  $\text{Or}\vec{\sigma}: \bigoplus_k (\text{Gra}_{\# \text{Vert}=k \geq 1}^{\wedge_i \text{edge}_i})_{S_k} \rightarrow \text{End}_{\text{skew}}^{*,*}(T_{\text{poly}}(M)[1])$  is a Lie algebra morphism:  $\text{Or}\vec{\sigma}([\gamma, \beta]) = [\text{Or}\vec{\sigma}(\gamma), \text{Or}\vec{\sigma}(\beta)]_{\text{NR}}$ .*

**Corollary 2.**  $\text{Or}\vec{\sigma}(d(\gamma)) = \text{Or}\vec{\sigma}([\bullet\bullet, \gamma]) = [\text{Or}\vec{\sigma}(\bullet\bullet), \text{Or}\vec{\sigma}(\gamma)]_{\text{NR}} = [\pi_S, \text{Or}\vec{\sigma}(\gamma)]_{\text{NR}}$ .

Let there be  $k$  vertices and  $2k - 2$  edges in  $\gamma$ , whence  $k + 1$  vertices in  $d(\gamma)$ . Evaluating both sides of the endomorphism equality  $\text{Or}\vec{\sigma}(d(\gamma)) = [\pi_S, \text{Or}\vec{\sigma}(\gamma)]_{\text{NR}}$  at a tuple of Poisson bi-vectors  $\mathcal{P}$ , we have that  $\text{Or}\vec{\sigma}([\bullet\bullet, \gamma])(\mathcal{P} \otimes \dots \otimes \mathcal{P}) =$

$$\begin{aligned} &= (\pi_S \vec{\sigma} \text{Or}\vec{\sigma}(\gamma))(\mathcal{P} \otimes \dots \otimes \mathcal{P}) - (-)^{(|\pi_S|=-1) \cdot (|\text{Or}\vec{\sigma}(\gamma)|=-|E(\gamma)|)} (\text{Or}\vec{\sigma}(\gamma) \vec{\sigma} \pi_S)(\mathcal{P} \otimes \dots \otimes \mathcal{P}) \\ &= \text{Or}\vec{\sigma}(\gamma)(\pi_S(\mathcal{P}, \mathcal{P}), \underline{\mathcal{P}}, \dots, \underline{\mathcal{P}}_{k-1}) + \dots + \text{Or}\vec{\sigma}(\gamma)(\underline{\mathcal{P}}, \dots, \underline{\mathcal{P}}_{k-1}, \pi_S(\mathcal{P}, \mathcal{P})) - \\ &\quad - \pi_S(\text{Or}\vec{\sigma}(\gamma)(\underline{\mathcal{P}}, \dots, \underline{\mathcal{P}}_k), \mathcal{P}) - \pi_S(\mathcal{P}, \text{Or}\vec{\sigma}(\gamma)(\underline{\mathcal{P}}, \dots, \underline{\mathcal{P}}_k)). \end{aligned} \quad (4)$$

**Theorem 1.** *Whenever  $\mathcal{P}$  is a Poisson bi-vector so that  $\pi_S(\mathcal{P}, \mathcal{P}) = 0 = \llbracket \mathcal{P}, \mathcal{P} \rrbracket$ , and whenever  $\gamma \in \ker d$  is a cocycle on  $k$  vertices and  $2k - 2$  edges (so that  $[\bullet\bullet, \gamma] = 0$ ), then  $\text{Or}\vec{\sigma}(\gamma)(\underline{\mathcal{P}} \otimes \dots \otimes \underline{\mathcal{P}}_k)$  is a Poisson cocycle (so that  $\llbracket \mathcal{P}, \text{Or}\vec{\sigma}(\gamma)(\mathcal{P}, \dots, \mathcal{P}) \rrbracket \doteq 0$  modulo the Jacobi identity  $\llbracket \mathcal{P}, \mathcal{P} \rrbracket = 0$  for the Poisson structure).*

<sup>4</sup>By construction, own grading of the endomorphism  $\vec{\sigma}(\gamma)$  equals minus the number of edges in  $\gamma$  (because each edge differentiates one  $\xi_\ell$ ):  $|\vec{\sigma}(\gamma)| = -|E(\gamma)|$ , cf. Table 1.

*Proof.* This is immediate from (4): its l.h.s. vanishes by  $\gamma \in \ker d$ ; in its right-hand side, the Jacobiator  $\pi_S(\mathcal{P}, \mathcal{P})$  is an argument of endomorphisms which are *linear*, hence all the  $k$  terms in the minuend vanish. The subtrahend remains, it yields 2 times the cocycle condition  $\text{Or}(\gamma)(\mathcal{P}, \dots, \mathcal{P}) \in \ker \llbracket \mathcal{P}, \cdot \rrbracket$ .  $\square$

**Corollary 3** (A realization of  $\diamond$  by Leibniz graphs). *The operator  $\diamond$  in the factorization problem*

$$\partial_{\mathcal{P}}(\text{Or}(\gamma)(\mathcal{P}, \dots, \mathcal{P})) = \diamond(\mathcal{P}, \llbracket \mathcal{P}, \mathcal{P} \rrbracket), \quad \gamma \in \ker d,$$

*is the sum of Leibniz graphs obtained from  $\gamma$  by inserting the Jacobiator  $\llbracket \mathcal{P}, \mathcal{P} \rrbracket$  into one of its vertices (by the Leibniz rule) and skew-symmetrizing w.r.t. the sinks.*

*Constructive proof.* Indeed, as (4) yields (with  $\pi_S(\mathcal{P}, \mathcal{P}) = (-)^{2-1}\llbracket \mathcal{P}, \mathcal{P} \rrbracket$ ) equality (1),

$$\llbracket \mathcal{P}, \text{Or}(\gamma)(\mathcal{P}, \dots, \mathcal{P}) \rrbracket = \frac{1}{2} \{ \text{Or}(\gamma)(\llbracket \mathcal{P}, \mathcal{P} \rrbracket, \mathcal{P}, \dots, \mathcal{P}) + \dots + \text{Or}(\gamma)(\mathcal{P}, \dots, \mathcal{P}, \llbracket \mathcal{P}, \mathcal{P} \rrbracket) \},$$

the left-hand side of the cocycle condition factors, in particular, through the explicitly given set of Leibniz graphs with  $\llbracket \mathcal{P}, \mathcal{P} \rrbracket$  in one vertex in the right-hand side.  $\square$

**Corollary 4.** *Suppose that  $\delta = d(\gamma)$  is a trivial d-cocycle in the graph complex: let there be  $k$  vertices and  $2k - 1$  edges in  $\gamma$ . Then, reading (4) again, we have that for  $\mathcal{P}$  Poisson,*

$$\begin{aligned} \text{Or}(\delta)(\mathcal{P}, \dots, \mathcal{P}) &= 0 + \dots + 0 - \underbrace{\pi_S(\text{Or}(\gamma)(\mathcal{P}, \dots, \mathcal{P}), \mathcal{P})}_{1\text{-vector}} - \underbrace{\pi_S(\mathcal{P}, \text{Or}(\gamma)(\mathcal{P}, \dots, \mathcal{P}))}_{1\text{-vector}} = \\ &= -(-)^{1-1} \llbracket \text{Or}(\gamma)(\mathcal{P}, \dots, \mathcal{P}), \mathcal{P} \rrbracket - (-)^{2-1} \llbracket \mathcal{P}, \text{Or}(\gamma)(\mathcal{P}, \dots, \mathcal{P}) \rrbracket = 2 \llbracket \mathcal{P}, \text{Or}(\gamma)(\mathcal{P}, \dots, \mathcal{P}) \rrbracket. \end{aligned} \tag{5}$$

*Equality (5) provides the composition of the 1-vector field  $\mathcal{X}(\mathcal{P}) := 2 \text{Or}(\gamma)(\mathcal{P}, \dots, \mathcal{P})$  trivializing  $\text{Or}(\delta)(\mathcal{P}, \dots, \mathcal{P}) = \llbracket \mathcal{P}, \mathcal{X}(\mathcal{P}) \rrbracket$  in the Poisson cohomology.*

*Remark 1.* From the above proof we also recognize the composition of Leibniz graphs (i.e. improper terms which vanish by the Jacobi identity  $\llbracket \mathcal{P}, \mathcal{P} \rrbracket = 0$ ) in the factorization problem


$$\text{Or}(\delta)(\mathcal{P}, \dots, \mathcal{P}) - \llbracket \mathcal{P}, \mathcal{X}(\mathcal{P}) \rrbracket \doteq \nabla(\mathcal{P}, \llbracket \mathcal{P}, \mathcal{P} \rrbracket).$$

Namely, it is the terms  $\text{Or}(\gamma)(\pi_S(\mathcal{P}, \mathcal{P}), \mathcal{P}, \dots, \mathcal{P}) + \dots + \text{Or}(\gamma)(\mathcal{P}, \dots, \mathcal{P}, \pi_S(\mathcal{P}, \mathcal{P}))$  from (4).

**1.3. The morphism  $\text{Or}$  at work: examples.** The following collection of examples illustrates (i) the construction of infinitesimal symmetries  $\dot{\mathcal{P}} = \mathcal{Q}(\mathcal{P})$  for Poisson structures  $\mathcal{P}$  by orienting cocycles  $\gamma \in \ker d$ , so that  $\mathcal{Q} = \text{Or}(\gamma)$ , and (ii) the construction of trivializing vector fields  $\mathcal{X} = 2\text{Or}(\gamma)$  in  $\mathcal{Q} = \text{Or}(d(\gamma))$ . At the same time, we detect (iii) the non-uniqueness of factorizations  $\llbracket \mathcal{P}, \mathcal{Q}(\mathcal{P}) \rrbracket = \diamond(\mathcal{P}, \llbracket \mathcal{P}, \mathcal{P} \rrbracket)$  for such cocycles and flows.

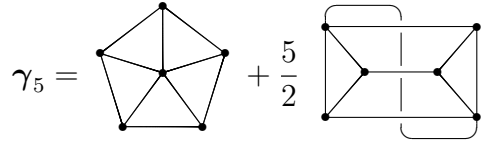
We remember that the (iterated commutators of the) infinite sequence of d-cocycles  $\gamma_{2\ell+1}$ , marked by  $(2\ell + 1)$ -gon wheel graphs (see [15]), is a regular source of universal symmetries for Poisson structures. Moreover, no flows  $\dot{\mathcal{P}} = \mathcal{Q}(\mathcal{P})$  other than these ones,  $\mathcal{Q}(\mathcal{P}) = \text{Or}(\gamma)(\mathcal{P})$ , are currently known (under the assumption that the cocycles  $\gamma$  be sums of connected graphs).

Let us remark finally that it is also an **open problem** whether these flows,  $\mathcal{Q}(\mathcal{P}) = \text{Or}(\gamma)(\mathcal{P})$ , can be Poisson cohomology nontrivial, that is  $\mathcal{Q} \neq \llbracket \mathcal{P}, \mathcal{X} \rrbracket$  for some Poisson structure  $\mathcal{P}$  and a globally defined vector field  $\mathcal{X}$  on an affine manifold  $M$ .

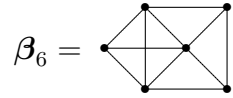
*Example 3* (The tetrahedron  $\gamma_3$ ). For the tetrahedron  $\gamma_3 \in \ker d$ , i.e. the full graph  on 4 vertices and 6 edges (see [10]), both the Kontsevich flow  $\dot{\mathcal{P}} = \mathcal{Q}_{1:6/2}(\mathcal{P})$  and the factorizing operator  $\diamond$  in the problem  $\llbracket \mathcal{P}, \mathcal{Q}_{1:6/2}(\mathcal{P}) \rrbracket = \diamond(\mathcal{P}, \llbracket \mathcal{P}, \mathcal{P} \rrbracket)$  are presented in [2] (cf. [11]). The operator  $\diamond$  is of the form given by Corollary 3.

*Example 4* (The pentagon-wheel cocycle  $\gamma_5 \in \ker d$ ).

For the pentagon-wheel cocycle, the set of oriented Kontsevich graphs that encode the flow  $\dot{\mathcal{P}} = \text{Or}(\gamma_5)(\mathcal{P})$  is listed in [5]. The resulting differential polynomial expression of this infinitesimal symmetry is available in Appendix A below. But the factorizing operator for  $\text{Or}(\gamma_5)(\mathcal{P})$  reported in [5], i.e. expressing  $\llbracket \mathcal{P}, \text{Or}(\gamma_5)(\mathcal{P}) \rrbracket$  as a sum of Leibniz graphs, is *different* from the operator  $\diamond$  which Corollary 3 provides for the cocycle  $\gamma_5$ . This demonstrates that such operators can be non-unique (as one obtains it in this particular example).<sup>5</sup>



*Example 5* (Coboundary  $\delta_6 = d(\beta_6)$ ). Take the only nonzero (with respect to the wedge ordering of edges) connected graph  $\beta_6$  on six vertices and 11 edges, and put  $\delta_6 = d(\beta_6) \in \ker d$  (indeed,  $d^2 = 0$ ). In view of Corollary 4 and Remark 1, we verify the decomposition,



$$\text{Or}(d(\beta_6))(\mathcal{P}) = \llbracket \mathcal{P}, \mathcal{X}(\mathcal{P}) \rrbracket + \nabla(\mathcal{P}, \llbracket \mathcal{P}, \mathcal{P} \rrbracket),$$

into the Poisson cohomology trivial and improper terms. Indeed, the vector field  $\mathcal{X}$  stems from  $\text{Or}(\beta_6)(\mathcal{P}, \dots, \mathcal{P})$  and the improper part comes from the terms like  $\text{Or}(\beta_6)(\llbracket \mathcal{P}, \mathcal{P} \rrbracket, \dots, \mathcal{P})$ . Interestingly, all the graphs from the  $\partial_{\mathcal{P}}$ -exact term  $\llbracket \mathcal{P}, \mathcal{X} \rrbracket$  also appear in the improper terms, and in fact they cancel. (There are 598 graphs in the former and 2098 in the latter;  $2098 - 598 = 1500$ , cf. Table 2 below.)

*Example 6* (The heptagon-wheel cocycle  $\gamma_7 \in \ker d$ ). The d-cocycle starting with the heptagon-wheel graph is presented in [6]. The flow  $\dot{\mathcal{P}} = \text{Or}(\gamma_7)(\mathcal{P})$  is realized by 37,185 Kontsevich graphs on 2 sinks; they are listed in a standard format (see [2, Implementation 1]) at <http://rburing.nl/gamma7.zip>. The factorizing operator  $\diamond$  is provided by Corollary 3 so that the validity of cocycle equation  $\llbracket \mathcal{P}, \text{Or}(\gamma_7)(\mathcal{P}) \rrbracket = \diamond(\mathcal{P}, \llbracket \mathcal{P}, \mathcal{P} \rrbracket)$  is verified experimentally. (It would be unfeasible to solve this equation w.r.t. the unknown coefficients of the Leibniz graphs in the right-hand side, pretending that a solution is not known from §1.2. Note however that no uniqueness is claimed for this  $\diamond$ .)

<sup>5</sup>We say that two Leibniz graphs (i.e. graphs with a tri-vector  $\llbracket \mathcal{P}, \mathcal{P} \rrbracket$  in a vertex) are *adjacent vertices* in the Leibniz meta-graph if the expansions of these Leibniz graphs have at least one Kontsevich oriented graph in common. (In the meta-graphs, multiple edges are allowed.) The known existence of several factorizations,  $\llbracket \mathcal{P}, \mathcal{Q} \rrbracket = \diamond_1(\mathcal{P}, \llbracket \mathcal{P}, \mathcal{P} \rrbracket) = \diamond_2(\mathcal{P}, \llbracket \mathcal{P}, \mathcal{P} \rrbracket)$ , into Leibniz graphs reveals the identities  $(\diamond_1 - \diamond_2)(\mathcal{P}, \llbracket \mathcal{P}, \mathcal{P} \rrbracket) \equiv 0$  for  $\diamond_1 \neq \diamond_2$ , that is, a nontrivial topology of the meta-graph. Its study is an **open problem**.

TABLE 2. The number of graphs in the problem  $\llbracket \mathcal{P}, \text{Or}\vec{\gamma}(\mathcal{P}) \rrbracket = \diamond(\mathcal{P}, \llbracket \mathcal{P}, \mathcal{P} \rrbracket)$ .

Cocycle:	$\gamma_3$	$\gamma_5$	$\delta_6 = d(\beta_6)$	$\gamma_7$
#vertices:	4	6	7	8
#edges:	6	10	12	14
#graphs:	1	2	4	46
#or.graphs in $\mathcal{Q}(\mathcal{P}) = \text{Or}\vec{\gamma}(\mathcal{P}, \dots, \mathcal{P})$ :	3	167	1,500	37,185
#or.graphs in $\llbracket \mathcal{P}, \mathcal{Q}(\mathcal{P}) \rrbracket$ :	39	3,495	35,949	1,003,611
#skew Leibniz graphs in $\diamond(\mathcal{P}, \llbracket \mathcal{P}, \mathcal{P} \rrbracket)$ :	8	843	9,556	293,654

**Implementation.** All calculations above were performed by using the software packages `graph_complex-cpp` and `kontsevich_graph_series-cpp`, which are released under the MIT free software license and available from <https://github.com/rburing>. Specifically, the programs `expanding_differential` and `kernel` have been used to find non-oriented graph cocycles  $\gamma$ , `orient` yields the sums of Kontsevich oriented graphs  $\text{Or}\vec{\gamma}(\mathcal{P}, \dots, \mathcal{P})$  and sums of Leibniz graphs  $\text{Or}\vec{\llbracket \mathcal{P}, \mathcal{P} \rrbracket}, \dots, \mathcal{P}$ , and `schouten_bracket` implements the Schouten bracket. The program `leibniz_expand` expands sums of Leibniz graphs into Kontsevich graphs, and `reduce_mod_skew` reduces sums of Kontsevich oriented graphs modulo skew-symmetry,  $L \prec R = -R \prec L$ , of the Left  $\prec$  Right mark-up of outgoing edges.

*Acknowledgements.* The research of RB was supported in part by IM JGU Mainz; AVK was partially supported by JBI RUG project 106552. The authors are grateful to the organizers of the 32nd International Colloquium on Group Theoretical Methods in Physics (GROUP32, 9 – 13 July 2018, CVUT Prague, Czech Republic) for partial financial support (for RB) and warm hospitality. The authors thank M. Kontsevich, A. Sharapov, and T. Willwacher for helpful discussions. A part of this research was done while RB was visiting at RUG and AVK was visiting at JGU Mainz (supported by IM via project 5020).







$$\begin{aligned}
& +5\partial_t\partial_p\mathcal{P}^{ij}\partial_r\partial_j\mathcal{P}^{kl}\partial_\ell\mathcal{P}^{mn}\partial_s\partial_m\mathcal{P}^{pq}\partial_v\partial_n\mathcal{P}^{rs}\partial_q\mathcal{P}^{tv}\partial_i f\partial_{kg} - 5\partial_s\partial_m\mathcal{P}^{ij}\partial_t\partial_j\mathcal{P}^{kl}\partial_k\mathcal{P}^{mn}\partial_n\partial_\ell\mathcal{P}^{pq}\partial_v\partial_p\mathcal{P}^{rs}\partial_q\mathcal{P}^{tv}\partial_i f\partial_{rg} \\
& +5\partial_t\partial_p\mathcal{P}^{ij}\partial_r\partial_j\mathcal{P}^{kl}\partial_v\partial_\ell\mathcal{P}^{mn}\partial_s\partial_m\mathcal{P}^{pq}\partial_n\mathcal{P}^{rs}\partial_q\mathcal{P}^{tv}\partial_i f\partial_{kg} + 5\partial_v\partial_m\mathcal{P}^{ij}\partial_r\partial_j\mathcal{P}^{kl}\partial_k\mathcal{P}^{mn}\partial_\ell\mathcal{P}^{pq}\partial_p\partial_n\mathcal{P}^{rs}\partial_s\partial_q\mathcal{P}^{tv}\partial_i f\partial_{tg} \\
& +5\partial_t\partial_p\mathcal{P}^{ij}\partial_r\partial_j\mathcal{P}^{kl}\partial_v\partial_\ell\mathcal{P}^{mn}\partial_m\mathcal{P}^{pq}\partial_n\mathcal{P}^{rs}\partial_s\partial_q\mathcal{P}^{tv}\partial_i f\partial_{kg} + 5\partial_t\partial_s\mathcal{P}^{ij}\partial_v\partial_j\mathcal{P}^{kl}\partial_k\mathcal{P}^{mn}\partial_m\partial_\ell\mathcal{P}^{pq}\partial_p\partial_n\mathcal{P}^{rs}\partial_q\mathcal{P}^{tv}\partial_i f\partial_{rg} \\
& +5\partial_r\partial_p\mathcal{P}^{ij}\partial_t\partial_j\mathcal{P}^{kl}\partial_v\partial_\ell\mathcal{P}^{mn}\partial_m\mathcal{P}^{pq}\partial_q\partial_n\mathcal{P}^{rs}\partial_s\mathcal{P}^{tv}\partial_i f\partial_{kg} - 5\partial_t\partial_p\partial_n\mathcal{P}^{ij}\partial_j\mathcal{P}^{kl}\partial_v\partial_r\partial_k\mathcal{P}^{mn}\partial_\ell\mathcal{P}^{pq}\partial_q\mathcal{P}^{rs}\partial_s\mathcal{P}^{tv}\partial_i f\partial_{mg} \\
& -5\partial_t\partial_r\partial_m\mathcal{P}^{ij}\partial_v\partial_p\partial_j\mathcal{P}^{kl}\partial_\ell\mathcal{P}^{mn}\partial_n\mathcal{P}^{pq}\partial_q\mathcal{P}^{rs}\partial_s\mathcal{P}^{tv}\partial_i f\partial_{kg} + 5\partial_t\partial_s\partial_m\mathcal{P}^{ij}\partial_j\mathcal{P}^{kl}\partial_k\mathcal{P}^{mn}\partial_\ell\mathcal{P}^{pq}\partial_v\partial_p\partial_n\mathcal{P}^{rs}\partial_q\mathcal{P}^{tv}\partial_i f\partial_{rg} \\
& -5\partial_t\partial_r\partial_p\mathcal{P}^{ij}\partial_v\partial_q\partial_j\mathcal{P}^{kl}\partial_\ell\mathcal{P}^{mn}\partial_m\mathcal{P}^{pq}\partial_n\mathcal{P}^{rs}\partial_s\mathcal{P}^{tv}\partial_i f\partial_{kg} - 5\partial_t\partial_r\partial_n\mathcal{P}^{ij}\partial_j\mathcal{P}^{kl}\partial_v\partial_p\partial_k\mathcal{P}^{mn}\partial_\ell\mathcal{P}^{pq}\partial_q\mathcal{P}^{rs}\partial_s\mathcal{P}^{tv}\partial_i f\partial_{mg} \\
& \quad -5\partial_t\partial_p\partial_m\mathcal{P}^{ij}\partial_v\partial_r\partial_j\mathcal{P}^{kl}\partial_\ell\mathcal{P}^{mn}\partial_n\mathcal{P}^{pq}\partial_q\mathcal{P}^{rs}\partial_s\mathcal{P}^{tv}\partial_i f\partial_{kg}.
\end{aligned}$$

## REFERENCES

- [1] Bouisaghouane A., Kiselev A. V. (2017) Do the Kontsevich tetrahedral flows preserve or destroy the space of Poisson bi-vectors? *J. Phys.: Conf. Ser.* **804** Proc. XXIV Int. conf. ‘Integrable Systems and Quantum Symmetries’ (14–18 June 2016, ČVUT Prague, Czech Republic), Paper 012008, 10 p. (Preprint arXiv:1609.06677 [q-alg])
- [2] Bouisaghouane A., Buring R., Kiselev A. (2017) The Kontsevich tetrahedral flow revisited, *J. Geom. Phys.* **119**, 272–285. (Preprint arXiv:1608.01710 [q-alg])
- [3] Buring R., Kiselev A. V. (2017) On the Kontsevich  $\star$ -product associativity mechanism, *PEPAN Letters* **14**:2, 403–407. (Preprint arXiv:1602.09036 [q-alg])
- [4] Buring R., Kiselev A. V. (2017) The expansion  $\star \bmod \bar{o}(\hbar^4)$  and computer-assisted proof schemes in the Kontsevich deformation quantization, Preprint arXiv:1702.00681 [math.CO]
- [5] Buring R., Kiselev A. V., Rutten N. J. (2018) Poisson brackets symmetry from the pentagon-wheel cocycle in the graph complex, *PEPAN Letters* **49**:5, 924–928. (Preprint arXiv:1712.05259 [math-ph])
- [6] Buring R., Kiselev A. V., Rutten N. J. (2017) The heptagon-wheel cocycle in the Kontsevich graph complex, *J. Nonlin. Math. Phys.* **24**, Suppl. 1, 157–173. (Preprint arXiv:1710.00658 [math.CO])
- [7] Buring R., Kiselev A. V., Rutten N. J. (2017) Infinitesimal deformations of Poisson bi-vectors using the Kontsevich graph calculus, *J. Phys.: Conf. Ser.* **965**, Paper 012010, 12 pp. (Preprint arXiv:1710.02405 [math.CO])
- [8] Dolgushev V. A., Rogers C. L., Willwacher T. H. (2015) Kontsevich’s graph complex, GRT, and the deformation complex of the sheaf of polyvector fields, *Ann. Math.* **182**:3, 855–943. (Preprint arXiv:1211.4230 [math.KT])
- [9] Jost C. (2013) Globalizing  $L$ -infinity automorphisms of the Schouten algebra of polyvector fields, *Differ. Geom. Appl.* **31**:2, 239–247. (Preprint arXiv:1201.1392 [math.QA])
- [10] Kontsevich M. (1997) Formality conjecture. Deformation theory and symplectic geometry (Ascona 1996, D. Sternheimer, J. Rawnsley and S. Gutt, eds), *Math. Phys. Stud.* **20**, Kluwer Acad. Publ., Dordrecht, 139–156.
- [11] Kontsevich M. (2017) Derived Grothendieck–Teichmüller group and graph complexes [after T. Willwacher], *Séminaire Bourbaki* (69ème année, Janvier 2017), no. 1126, 26 p.
- [12] Laurent-Gengoux C., Pichereau M., Vanhaecke P. (2013) Poisson Structures. Grundlehren der mathematischen Wissenschaften **347**, Springer-Verlag Berlin Heidelberg.
- [13] Merkulov S., Willwacher T. (2014) Grothendieck–Teichmüller and Batalin–Vilkovisky, *Lett. Math. Phys.* **104**:5, 625–634. (Preprint arXiv:1012.2467 [math.QA])
- [14] Rutten N. J., Kiselev A. V. (2018) The defining properties of the Kontsevich unoriented graph complex, Preprint arXiv:1811.10638 [math.CO], 10 p.
- [15] Willwacher T. (2015) M. Kontsevich’s graph complex and the Grothendieck–Teichmüller Lie algebra, *Invent. Math.* **200**:3, 671–760. (Preprint arXiv:1009.1654 [q-alg])



# Chapter 18

## The Kontsevich graph orientation morphism revisited

This chapter is based on the peer-reviewed journal publication *A.V. Kiselev and R. Bur-  
ing, Banach Center Publ. 123 Homotopy algebras, deformation theory & quantization,*  
123–139, 2021. (Preprint [arXiv:1904.13293](https://arxiv.org/abs/1904.13293) [math.CO] – 18 p.)

*Commentary.* In reference to Part I of the dissertation, the material of this chapter is  
used in Chapter 5 (particularly, the `attach_to_ground` method). All the examples of  
universal flows encoded by graphs are borrowed in this chapter from Chapters 15 and 16.

# THE KONTSEVICH GRAPH ORIENTATION MORPHISM REVISITED

ARTHEMY V. KISELEV

*Bernoulli Institute for Mathematics, Computer Science and Artificial Intelligence,  
University of Groningen, P.O.Box 407, 9700 AK Groningen, The Netherlands.  
E-mail: A.V.Kiselev@rug.nl (corresponding author)*

RICARDO BURING

*Institut für Mathematik, Johannes Gutenberg–Universität,  
Staudingerweg 9, D-55128 Mainz, Germany.  
E-mail: rburing@uni-mainz.de*

**Abstract.** The orientation morphism  $\text{Or}(\cdot)(\mathcal{P}): \gamma \mapsto \dot{\mathcal{P}}$  associates differential-polynomial flows  $\dot{\mathcal{P}} = \mathcal{Q}(\mathcal{P})$  on spaces of bi-vectors  $\mathcal{P}$  on finite-dimensional affine manifolds  $N^d$  with (sums of) finite unoriented graphs  $\gamma$  with ordered sets of edges and without multiple edges and one-cycles. It is known that d-cocycles  $\gamma \in \ker d$  with respect to the vertex-expanding differential  $d = [\bullet \dashrightarrow \bullet, \cdot]$  are mapped by  $\text{Or}$  to Poisson cocycles  $\mathcal{Q}(\mathcal{P}) \in \ker [[\mathcal{P}, \cdot]]$ , that is, to infinitesimal symmetries of Poisson bi-vectors  $\mathcal{P}$ . The formula of orientation morphism  $\text{Or}$  was expressed in terms of the edge orderings as well as parity-odd and parity-even derivations on the odd cotangent bundle  $\Pi T^* N^d$  over any  $d$ -dimensional affine real Poisson manifold  $N^d$ . We express this formula in terms of (un)oriented graphs themselves, *i.e.* without explicit reference to supermathematics on  $\Pi T^* N^d$ .

**Introduction.** A differential graded Lie algebra structure on the vector space of un-oriented graphs with ordered sets of parity-odd edges was introduced by Kontsevich in [14, 15]. The vertex-expanding differential  $d = [\bullet \dashrightarrow \bullet, \cdot]$  is the adjoint action by the edge  $\bullet \dashrightarrow \bullet$ , which itself satisfies the master equation  $[\bullet \dashrightarrow \bullet, \bullet \dashrightarrow \bullet] = (\text{const} \neq 0) \cdot \mathbf{0} \in \text{Graph complex}$ . Examples of d-cocycles are discussed in [16, 22] or [9]. Properties of the un-oriented graph complex and its relation to the Grothendieck–Teichmüller group were explored by Willwacher (see [21]).

---

2010 *Mathematics Subject Classification*: Primary 05C22, 16E45, 53D17; Secondary 68R10, 81R60.

*Key words and phrases*: Graph complex, vertex-expanding differential, orientation morphism, Poisson bracket, deformation

The paper is in final form and no version of it will be published elsewhere.

The language of graphs allows encoding (poly)differential operators on affine manifolds (let us consider  $\mathbb{R}^d$  for simplicity), as well as encoding graded-symmetric endomorphisms  $\pi \in \text{End}(T_{\text{poly}}(\mathbb{R}^d))$  on the spaces of totally skew-symmetric multivector fields on  $\mathbb{R}^d$ . For example, a Poisson bivector  $\mathcal{P}$  on  $\mathbb{R}^d$  is represented by the wedge  $\wedge$  with Left  $<$  Right edge ordering, whereas the Schouten bracket is an endomorphism:

$$\begin{aligned} \pi_S(F, G) &= \text{Or}\left(\begin{array}{c} 1 \\ \bullet \text{---} \bullet \\ 2 \end{array}\right)(F \otimes G) = \frac{1}{2!} \left( \left. \frac{\vec{\partial}}{\partial \xi} \right|_1 \otimes \left. \frac{\vec{\partial}}{\partial \mathbf{x}} \right|_2 + \left. \frac{\vec{\partial}}{\partial \xi} \right|_2 \otimes \left. \frac{\vec{\partial}}{\partial \mathbf{x}} \right|_1 \right) (F|_1 \cdot G|_2 + F|_2 \cdot G|_1) = \\ &= \frac{\vec{\partial}}{\partial \xi}(F) \cdot \frac{\vec{\partial}}{\partial \mathbf{x}}(G) + (-)^{|F|} \frac{\vec{\partial}}{\partial \mathbf{x}}(F) \cdot \frac{\vec{\partial}}{\partial \xi}(G) = (-)^{|F|-1} \left\{ (F) \frac{\vec{\partial}}{\partial \xi} \cdot \frac{\vec{\partial}}{\partial \mathbf{x}}(G) - (F) \frac{\vec{\partial}}{\partial \mathbf{x}} \cdot \frac{\vec{\partial}}{\partial \xi}(G) \right\} = \\ &= (-)^{|F|-1} \llbracket F, G \rrbracket = (-)^{|F|-1} \{ F \longrightarrow G \quad - \quad F \longleftarrow G \}, \end{aligned}$$

for any homogeneous multivectors  $F$  and  $G$ . It is readily seen from the definition that the endomorphism  $\pi_S$  is graded-symmetric,

$$\pi_S(F, G) = (-)^{|F| \cdot |G|} \pi_S(G, F),$$

and the usual variant of the Schouten bracket is shifted-graded skew-symmetric,

$$\llbracket F, G \rrbracket = -(-)^{(|F|-1) \cdot (|G|-1)} \llbracket G, F \rrbracket.$$

This language of oriented graphs is used in Kontsevich’s solution of the problem of deformation quantisation on finite-dimensional Poisson manifolds (see [16, 18] and [4, 6, 7]). It provides also the construction of universal – with respect to all affine Poisson manifolds – infinitesimal symmetries  $\dot{\mathcal{P}} = \mathcal{Q}(\mathcal{P})$  of classical Poisson structures (see [16, 19] and [2]).

The orientation morphism  $\text{Or}$  associates Poisson cocycles  $\mathcal{Q}(\mathcal{P}) \in \ker \llbracket \mathcal{P}, \cdot \rrbracket$  with  $[\bullet \text{---} \bullet, \cdot]$ -cocycles in the unoriented graph complex. The algebraic formula of  $\text{Or}$  was given in [16]; its constructions was discussed in more detail by Jost in [12], by Willwacher in [21], and by the authors in a recent paper [5] (cf. [8]). This morphism is a tool which produces symmetries of Poisson brackets. If such a symmetry is not Poisson-exact in the second Poisson cohomology, then it yields deformations of classical Liouville-integrable systems, preserving the property that the dynamics is Poisson. If the symmetry of a given bracket is Poisson-exact, then its flow produces a family of diffeomorphisms – typically, nonlinear – of the affine Poisson manifold under study. A construction of universal Poisson 1-cocycles  $\vec{\mathcal{X}} = \text{Or}(\gamma)(\vec{V}, \mathcal{P}^{\otimes n-1})$  from graph cocycles  $\gamma = \sum_a c_a \cdot \gamma_a$  with  $n$  vertices in each term  $\gamma_a$  is introduced in [8] for homogeneous Poisson bi-vectors  $\mathcal{P} = L_{\vec{V}}(\mathcal{P}) = \llbracket \vec{V}, \mathcal{P} \rrbracket$ . (The domain of definition of  $\text{Or}$  is of course larger than the homogeneous component of unoriented graphs on  $n$  vertices and  $2n - 2$  edges,  $n \in \mathbb{N}$ .) So far, one example of its work, namely  $\text{Or}: \gamma_3 \in \ker d \mapsto \mathcal{Q}_{1;\frac{6}{2}}(\mathcal{P}) \in \ker \partial_{\mathcal{P}}$ , was known from the seminal paper [16] in which the tetrahedral flow was introduced (cf. [19] and [2]). The pentagon-wheel flow  $\text{Or}(\gamma_5)(\mathcal{P})$  has been obtained in [10] by solving the factorisation problem  $\llbracket \mathcal{P}, \mathcal{Q}_5(\mathcal{P}) \rrbracket = \diamond(\mathcal{P}, \llbracket \mathcal{P}, \mathcal{P} \rrbracket)$  with respect to the Leibniz graphs in  $\diamond$  and the Kontsevich bivector graphs in  $\mathcal{Q}_5 = \text{Or}(\gamma_5)$  on 6 internal vertices. The heptagon-wheel flow  $\mathcal{Q}_7 = \text{Or}(\gamma_7)$  on 8 internal vertices is described in [5].

The algebraic formula of the orientation morphism amounts to a simple but extensive calculation using  $\mathbb{Z}/2\mathbb{Z}$ -gradings, *i.e.* supermathematics. The morphism  $\text{Or}$  determines both the orgraph multiplicities and the signs of all the Kontsevich graphs  $\Gamma$  (which, we

recall, are equipped with an ordering  $\text{Left} < \text{Right}$  at all their internal vertices  $v$ , such that  $\Gamma(L < R|_v) = -\Gamma(R < L|_v)$ . Nevertheless, the algebraic formula is external with respect to the (un)oriented graph complexes. We pose the question whether doing this  $\mathbb{Z}_2$ -graded computation – involving thousands and millions of graphs – is the only way to build each element of a known infinite sequence  $\dot{\mathcal{P}} = \mathcal{Q}_{2\ell+1}(\mathcal{P})$  of flows.<sup>1</sup>

The aim of this note is to express the orientation morphism in geometric terms, that is, by using combinatorial data which are intrinsic with respect to the (un)oriented graphs  $\gamma_a$  equipped with edge orderings  $I \prec II \prec \dots$  in  $\gamma_a$  and wedge orderings  $L_i < R_i$  at vertices  $v_i$  in  $\text{Or}(\gamma_a)$ , respectively. We let a cocycle  $\gamma = \sum_a c_a \cdot \gamma_a \in \ker d$  on  $n$  vertices and with a given ordering  $I \prec II \prec III \prec \dots$  of  $2n - 2$  edges in each graph  $\gamma_a$  be the initial datum. The problem is to reveal the few rules of matching for the Kontsevich oriented graphs: whenever a sign is fixed in front of just one Kontsevich graph which is obtained by orienting a given graph  $\gamma_a$  in a cocycle  $\gamma = c_1\gamma_1 + \dots + c_m\gamma_m \in \ker d$ , the signs in front of *all* admissible orientations of *all* the graphs in their linear combination  $\gamma$  are then determined without calculation of  $\text{Or}$  for each of them.

This paper contains three sections. For consistency, in section 1 we recall the algebraic formula of the orientation morphism  $\text{Or}$  for the Kontsevich graph complex; the approach is operadic (cf. [17]). In section 2, we establish the main rule for the count of multiplicities and signs of orgraphs. In the final section, we expand on the rule of signs purely in terms of oriented Kontsevich graphs; to this end, we analyse their admissible types and combinatorial properties of transitions between them.

**1. The algebraic formula of graph orientation morphism  $\text{Or}$ .** Denote by  $\text{Gra}$  the real vector space of (formal sums<sup>2</sup> of) unoriented, – by default, connected – finite graphs  $\gamma = \sum_a c_a \cdot \gamma_a$  equipped with wedge ordering of edges  $\mathbf{E}(\gamma_a) = I \wedge II \wedge \dots$  in each graph  $\gamma_a$ ; by definition, put  $(\gamma, I \wedge II \wedge \dots) = -(\gamma, II \wedge I \wedge \dots)$ , etc. The vertices of a graph  $\gamma_a$  are not ordered; one is free to choose any labelling of these  $\#\text{Vert}(\gamma_a)$  vertices by using  $1, \dots, n = \#\text{Vert}(\gamma_a)$  because the assignment  $\wp_i \mapsto v_{\sigma(i)}$  of an  $n$ -tuple  $\wp_1, \dots, \wp_n$  of multivectors, which form the ordered set of graded arguments for an endomorphism defined in formula (1) below, will be provided by the entire permutation group  $S_n \ni \sigma$ : the  $i$ th multivector is placed into the vertex  $v_{\sigma(i)}$ .

A *zero unoriented graph*  $\mathbf{0}$  is a graph which is equal to minus itself under an automorphism (that induces a permutation of edges whose parity we thus inspect).

**EXAMPLE 1.** The two-edge connected graph  $\bullet \text{---} \bullet \text{---} \bullet$  is a zero graph because the flip, a symmetry of this graph, swaps the edges which anticommute,  $I \wedge II = -II \wedge I$ , whence the graph at hand equals minus its image under its own automorphism, i.e. minus itself.

<sup>1</sup>Willwacher showed in [21] that at every  $\ell \in \mathbb{N}$ , the  $(2\ell + 1)$ -wheel graph marks a nontrivial  $d$ -cocycle in the unoriented graph complex, and those are expected to yield – for generic Poisson structures  $\mathcal{P}$  – nontrivial terms in the respective Poisson cohomology under the orientation morphism. It remains an open problem to find a graph cocycle  $\gamma$  and Poisson bracket  $\mathcal{P}$  on an affine manifold  $N^d$  such that the respective Poisson cocycle  $\text{Or}(\gamma)(\mathcal{P})$  is nontrivial. Iterated commutators of the cocycles  $\gamma_{2\ell+1}$  yield Poisson cocycles as well.

<sup>2</sup>For the sake of definition, we assume by default that all terms in each sum are homogeneous w.r.t. the various gradings.

The operadic insertion  $\circ_i: (\gamma_1, \mathbf{E}(\gamma_1)) \otimes (\gamma_2, \mathbf{E}(\gamma_2)) \mapsto (\gamma_1 \circ_i \gamma_2, \mathbf{E}(\gamma_1) \wedge \mathbf{E}(\gamma_2))$  is the Leibniz-rule sum of insertions of  $\gamma_1$  into vertices of  $\gamma_2$  such that every edge which was incident to a vertex  $v$  in  $\gamma_2$  is redirected to vertices in  $\gamma_1$  in all possible ways according to another Leibniz rule. The super Lie bracket  $[\gamma_1, \gamma_2] = \gamma_1 \circ_i \gamma_2 - (-)^{\#\mathbf{E}(\gamma_1) \cdot \#\mathbf{E}(\gamma_2)} \gamma_2 \circ_i \gamma_1$  on Gra yields the vertex-expanding differential  $d = [\bullet \dashrightarrow \bullet, \cdot]$ .

Denote by  $\text{End}(T_{\text{poly}}(\mathbb{R}^d))$  the vector space of graded-symmetric endomorphisms on the space of multivector fields on the space  $\mathbb{R}^d$  (which we view here as an affine manifold). In a full analogy with graphs (see [5]), one inserts a given endomorphism into an argument slot of another endomorphism, proceeding over the slots consecutively and taking the sum. The Richardson–Nijenhuis bracket  $[\cdot, \cdot]_{\text{RN}}$  is the graded symmetrisation (with respect to its multivector arguments) of the difference of insertions:  $[\pi, \rho] = \text{Sym}(\pi \circ \rho - (-)^{|\pi| \cdot |\rho|} \rho \circ \pi)$ . The Schouten bracket  $\pi_S(R, S) = (-)^{|R|-1} \llbracket R, S \rrbracket$  of multivectors  $R, S$  is a graded symmetric endomorphism which satisfies the (shifted-) graded Jacobi identity  $[\pi_S, \pi_S]_{\text{RN}} = 0$ ; hence by the graded Jacobi identity for the Richardson–Nijenhuis bracket (see [20]), the operation  $[\pi_S, \cdot]_{\text{RN}}$  is a differential.

For a given graph  $\gamma$  on  $n$  vertices and for an ordered  $n$ -tuple  $\wp$  of multivectors  $\wp_1, \dots, \wp_n$ , the morphism  $\text{Or}\vec{\gamma}: \text{Gra} \rightarrow \text{End}(T_{\text{poly}}(\mathbb{R}^d))$  is encoded by the formula

$$\text{Or}\vec{\gamma}(\wp_1 \otimes \dots \otimes \wp_n) = \frac{1}{n!} \sum_{\sigma \in S_n} \prod_{e_{ij} \in \mathbf{E}(\gamma)} \vec{\Delta}_{ij} (\wp_1|_{v_{\sigma(1)}} \dots \wp_n|_{v_{\sigma(n)}}), \tag{1}$$

where, in the ordered product of edges, the first edge  $I \in \mathbf{E}(\gamma)$  acts first and the last edge acts last. For every edge  $e_{ij}$  connecting the vertices  $v_i$  and  $v_j$  in the graph  $\gamma$ , the edge operator,

$$\vec{\Delta}_{ij} = \sum_{\alpha=1}^{\dim \mathbb{R}^d} \{i \xrightarrow{\alpha} j + i \xleftarrow{\alpha} j\} = \sum_{\alpha=1}^{\dim \mathbb{R}^d} \left( \frac{\vec{\partial}}{\partial \xi_\alpha^{(i)}} \otimes \frac{\vec{\partial}}{\partial x_{(j)}^\alpha} + \frac{\vec{\partial}}{\partial x_{(i)}^\alpha} \otimes \frac{\vec{\partial}}{\partial \xi_\alpha^{(j)}} \right)$$

acts from left to right by the (graded-) derivations along the ordered sequence of multivectors  $\wp_1 \otimes \dots \otimes \wp_n$ , now placed by a permutation  $\sigma \in S_n$  into the respective vertices  $v_{\sigma(1)}, \dots, v_{\sigma(n)}$ . In this way, the derivations in each edge operator  $\vec{\Delta}_{ij}$  reach the content  $\wp_{\sigma^{-1}(i)}$  and  $\wp_{\sigma^{-1}(j)}$  of the vertices  $v_i$  and  $v_j$ . Summarizing, the graph with an edge ordering  $(\gamma, \mathbf{E}(\gamma))$  is the set of topological data which determine all the portraits of couplings within the  $n$ -tuple of multivectors: the above formula is referred to a  $d$ -tuple  $\{x^\alpha\}$  of affine coordinates on  $\mathbb{R}^d$  and the canonical conjugate  $d$ -tuple  $\{\xi_\alpha\}$  of fibre coordinates on the parity-odd cotangent bundle  $\Pi T^* \mathbb{R}^d$ , so that the summation  $\sum_{\alpha_{ij}=1}^{\dim \mathbb{R}^d}$  gives the coupling along the edge  $e_{ij}$ . (Clearly, one proceeds by linearity over sums of graphs  $\gamma$  and over sums of multivectors  $\wp_i$  at every  $i$  running from 1 to  $n$ .)

LEMMA 1. *Suppose that at most one – in the rest of this paper, none usually, still exactly one only in Corollary 3 below and likewise in [8] – multivector  $\wp_1, \dots, \wp_{\#\text{Vert}(Z \in \text{Gra})}$  is odd-graded, while all the rest are even-grading (so that all are permutable without signs appearing). Then, for the sets of endomorphisms’ arguments restricted in this way, the*

morphism  $\text{Or}$  in (1) is well defined on the space  $\text{Gra}$ , that is, its action gives

$$\begin{aligned} \text{Or}(\text{zero graph } \mathcal{Z}) &= \\ &= (0 \in \mathbb{R}) \cdot (\text{nonzero oriented graphs}) + (\text{coeffs} \in \mathbb{R}) \cdot (\text{zero Kontsevich orgraphs}). \end{aligned}$$

*Proof.* By definition, a zero unoriented graph  $\mathcal{Z}$  has a symmetry  $\sigma \in \text{Aut}(\mathcal{Z})$  which acts by a parity-odd permutation  $\sigma_E$  on the wedge-ordered set  $\text{E}(\mathcal{Z})$  of its edges and by a permutation  $\sigma_V$  on the set  $\text{Vert}(\mathcal{Z})$  of its vertices. The permutation of edges yields the same parity-odd permutation  $\sigma_E$  of the edge operators  $\Delta_{ij}$  in their ordered product  $\vec{\Delta} = \prod_{e_{ij} \in \text{E}(\mathcal{Z})} \vec{\Delta}_{ij}$ . The permutation  $\sigma_V$  of the content of vertices does not yield any signs because by our assumption, all the multivectors are pairwise permutable. Put  $n = \#\text{Vert}(\mathcal{Z})$ . Acting by the symmetry  $\sigma$  of unoriented graph  $\mathcal{Z}$  on the anticommuting edge operators and on the multivectors  $\wp_1, \dots, \wp_n$ , which are assigned consecutively by all the  $n!$  permutations  $\tau \in S_n$  to the vertices of  $\mathcal{Z}$  (and the ordered product of which is the argument of  $\vec{\Delta}$ ), we obtain – for the isomorphic unoriented graph  $\sigma(\mathcal{Z})$  – the expression

$$\begin{aligned} \sum_{\tau \in S_n} \prod_{e_{ij} \in \text{E}(\mathcal{Z})} \sigma_E(\vec{\Delta}_{ij}) \left( \prod_{\text{Vert}(\mathcal{Z})} \tau(\sigma_V(\wp)) \right) &= \\ = \sum_{\tau \in S_n} \prod_{e_{ij} \in \text{E}(\mathcal{Z})} \vec{\Delta}_{\sigma_E(e_{ij})} \left( \prod_{\text{Vert}(\mathcal{Z})} \tau(\sigma_V(\wp)) \right) &= \sum_{\tau \in S_n} (-\vec{\Delta}) \left( + \prod_{\text{Vert}(\mathcal{Z})} \tau(\wp) \right). \end{aligned}$$

This confirms that  $\text{Or}(\mathcal{Z})(\wp_1, \dots, \wp_n)$  is a differential polynomial (w.r.t. the components of multivectors  $\wp_i$ ) which equals minus itself, hence it is identically zero. But this defining property of the differential-polynomial expression means that the Kontsevich orgraph that encodes the endomorphism  $\text{Or}(\mathcal{Z})$  is a zero orgraph. ■

**PROPOSITION 2.** *Under the assumption – similar to the above in Lemma 1 – that all the multivector arguments  $\wp_i$  of the endomorphisms are in fact copies of a given Poisson bi-vector  $\mathcal{P}$ , the morphism  $\text{Or}$  “respects the operadic insertions” by sending the super Lie bracket of graphs to the Richardson–Nijenhuis bracket of endomorphisms:*

$$\text{Or}([\gamma_1, \gamma_2]) = [\text{Or}(\gamma_1), \text{Or}(\gamma_2)]_{RN},$$

where the right-hand side is graded symmetrised by construction.

In particular,  $\text{Or}(\bullet\text{---}\bullet) = \pi_S$ , i.e. the orientation morphism sends the edge to the Schouten bracket, whence

$$\text{Or}(d(\gamma)) = [\pi_S, \text{Or}(\gamma)]_{RN}. \tag{2}$$

This is standard (see Appendix A below and also [12, 16, 17] or [5]).

**COROLLARY 3.** *Evaluating the endomorphisms in both sides of Lie superalgebra morphism (2) at  $n + 1$  copies of a given Poisson bivector  $\mathcal{P}$  satisfying  $[[\mathcal{P}, \mathcal{P}]] = 0$  on  $\mathbb{R}^d$ ,*

$$\text{Or}([\bullet\text{---}\bullet, \gamma])(\mathcal{P}, \dots, \mathcal{P}) = 2[[\mathcal{P}, \text{Or}(\gamma)(\mathcal{P}, \dots, \mathcal{P})]] - \sum_i \text{Or}(\gamma)(\mathcal{P}, \dots, \mathcal{P}, \underbrace{[[\mathcal{P}, \mathcal{P}]]}_{i\text{th slot}}, \mathcal{P}, \dots, \mathcal{P}),$$

one obtains – for a cocycle  $\gamma$  on  $n$  vertices and  $2n - 2$  edges – an explicit solution of the factorisation problem,

$$[[\mathcal{P}, \text{Or}(\gamma)(\mathcal{P})]] = \diamond(\mathcal{P}, [[\mathcal{P}, \mathcal{P}]]), \tag{3}$$



for the Poisson cocycles  $\mathcal{Q}(\mathcal{P}) = \text{Or}(\gamma)(\mathcal{P})$  corresponding to  $[\bullet\!\!\!\rightarrow\!\!\!\bullet, \cdot]$ -cocycles in the unoriented graph complex.

Otherwise speaking, the orientation morphism  $\text{Or}$  sends  $d$ -cocycle graphs to  $\partial_{\mathcal{P}}$ -cocycles (i.e. Poisson cocycles). Moreover,  $d$ -exact unoriented graphs  $\gamma = d(\zeta)$  yield  $\partial_{\mathcal{P}}$ -coboundaries  $\mathcal{Q}(\mathcal{P}) = \llbracket \mathcal{P}, \mathcal{X}(\mathcal{P}) \rrbracket$  with one-vectors  $\mathcal{X}(\mathcal{P}) = 2 \cdot \text{Or}(\zeta)(\mathcal{P})$  (modulo the improper terms which, by definition, vanish on the entire  $\mathbb{R}^d$  whenever a bivector  $\mathcal{P}$  is Poisson).

EXAMPLE 2. The factorisation operators  $\diamond$  in (3) for the tetrahedral flow  $\dot{\mathcal{P}} = \mathcal{Q}_{1:\frac{6}{2}}(\mathcal{P}) = \text{Or}(\gamma_3)(\mathcal{P})$ , pentagon-wheel flow  $\text{Or}(\gamma_5)(\mathcal{P})$ , and for the heptagon-wheel flow  $\text{Or}(\gamma_7)(\mathcal{P})$  have been presented in [2], [10], and [5], respectively.<sup>3</sup>

PROPOSITION 4 (proved in App. A). For a given Poisson bi-vector  $\mathcal{P}$ , the graph orientation mapping,

$$\text{Or}(\cdot)(\mathcal{P}) : \ker d|_{(n, 2n-2)} \ni \gamma \mapsto \mathcal{Q}(\mathcal{P}) \in \ker \partial_{\mathcal{P}},$$

is a Lie superalgebra morphism that takes the bracket of two cocycles in bi-grading  $(n, 2n-2)$  to the commutator  $[\frac{d}{d\varepsilon_1}, \frac{d}{d\varepsilon_2}](\mathcal{P})$  of two symmetries  $\frac{d}{d\varepsilon_i}(\mathcal{P}) = \mathcal{Q}_i(\mathcal{P})$ .<sup>4</sup>

**2. Kontsevich orgraphs: The count of multiplicities and signs.** In this section we derive the rule for calculation of orgraph multiplicities and matching the signs of oriented Kontsevich graphs. This defining property completely determines the evaluation  $\text{Or}(\gamma)(\mathcal{P})$  of the orientation morphism for cocycles  $\gamma \in \ker d$  at (tuples of) Poisson bivectors  $\mathcal{P}$ .

CONVENTION. In the sequel, for a given graph  $\gamma_a$  in a cocycle  $\sum_a c_a \cdot \gamma_a = \gamma \in \ker d$ , sums are taken over the big set of  $2^{\#\text{E}(\gamma_a)}$  of ways to orient edges of  $\gamma_a$  by using the operator  $\vec{\Delta}$ .

Not every such way to orient edges would yield a Kontsevich orgraph built of  $n$  wedges (hence, having  $n$  internal vertices,  $2n$  arrow edges formed by  $2n-2$  old edges of  $\gamma_a$  and 2 new edges  $S_0, S_1$  to the new sink vertices  $0, 1$  with their content  $f, g$ ). But the Kontsevich orgraphs are selected automatically whenever a copy of bi-vector  $\mathcal{P} = \frac{1}{2}P^{ij}(\mathbf{x}) \xi_i \xi_j$  is placed in each internal vertex of every orgraph, so that every vertex of  $\gamma_a$  is the arrowtail for exactly two oriented edges.

CONVENTION. Every Kontsevich orgraph on  $n$  internal vertices and  $2n$  edges is encoded by the ordered (using an arbitrary fixed labelling of internal vertices) list of ordered (w.r.t. Left < Right at a vertex) pairs of target vertices for the respective outgoing edges.

DEFINITION 1. A Kontsevich orgraph (built of wedges) is a  $\Lambda$ -shaped orgraph if one vertex is the arrowtail of both edges directed to the sinks. Otherwise, a Kontsevich orgraph – in

<sup>3</sup>Let us emphasize that such operators are in general not unique. For instance, there are known solutions  $\diamond$  besides the subtrahend in the right-hand side of (2). E.g., the operator  $\diamond$  in (3) for the orgraph sum  $\text{Or}(\gamma_5)(\mathcal{P})$  is different from the factorisation found in [10]. Likewise, two linearly independent solutions of (3) for the tetrahedral flow  $\text{Or}(\gamma_3)(\mathcal{P})$  are built in [2].

<sup>4</sup>By Brown [3], the commutator does in general – for a generic Poisson structure  $\mathcal{P}$  – not vanish for Willwacher’s odd-sided wheel cocycles.

which the edges to sinks are issued from different vertices – is called a  $\Pi$ -shaped orgraph (see Fig. 1).

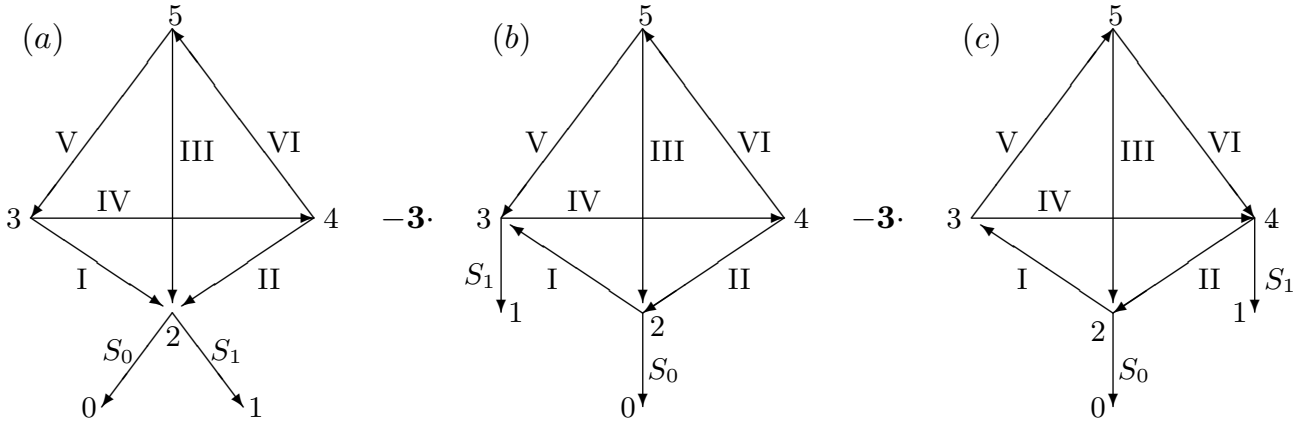


Fig. 1. The right-hand side  $\mathcal{Q}_{1;\frac{6}{2}}$  of the tetrahedral flow is encoded by the  $\Lambda$ -shaped and two  $\Pi$ -shaped orgraphs; the edge ordering in the wedges is (4), cf. Example 4 below.

EXAMPLE 3. In the tetrahedral flow

$$\mathcal{Q}_{1;\frac{6}{2}} = 1 \cdot (0, 1; 2, 4; 2, 5; 2, 3) - 3 \cdot (0, 3; 1, 4; 2, 5; 2, 3 + 0, 3; 4, 5; 1, 2; 2, 4), \quad (4)$$

the  $\Lambda$ -shaped orgraph and the  $\Pi$ -shaped orgraph, skew-symmetrised over its sinks, occur in proportion  $8 : 24 = 1 : 3$  (see [2] and [19]).<sup>5</sup>

Remark 1. Each encoding of a way to orient a graph  $\gamma_a$  to a Kontsevich orgraph on  $2n$  edges yields a permutation of the ordered set  $\mathbf{E}(\gamma_a) \wedge S_0 \wedge S_1 = +S_0 \wedge S_1 \wedge \mathbf{E}(\gamma_a)$  of these edges. Namely, the permutation is encoded by the ordered (using any given labelling of internal vertices) list of ordered (w.r.t. Left  $<$  Right at a vertex) pairs of outgoing edges.

CONVENTION. We let the default, occurring with ‘+’ sign, ordering of oriented edges in a Kontsevich orgraph under study be  $S_0 \prec S_1 \prec \mathbf{E}(\gamma_a) = S_0 \prec S_1 \prec I \prec II \prec \dots$ .

EXAMPLE 4. The tetrahedral graph flow is

$$\begin{aligned} \mathcal{Q}_{1;\frac{6}{2}} = & (+1) \cdot \begin{pmatrix} 0 & 1 & 2 & 4 & 2 & 5 & 2 & 3 \\ S_0 & S_1 & I & IV & II & VI & III & V \end{pmatrix} \\ & - 3 \cdot \left[ \begin{pmatrix} 0 & 3 & 1 & 4 & 2 & 5 & 2 & 3 \\ S_0 & I & S_1 & IV & II & VI & III & V \end{pmatrix} + \begin{pmatrix} 0 & 3 & 4 & 5 & 1 & 2 & 2 & 4 \\ S_0 & I & IV & V & S_1 & II & III & VI \end{pmatrix} \right]. \end{aligned}$$

The parity sign of edge permutation in the  $\Lambda$ -shaped graph is  $(-)^4 = (+)$ , here we count the transpositions w.r.t.  $S_0 \prec S_1 \prec I \prec \dots \prec VI$ ; the permutations of edges in the second and third,  $\Pi$ -shaped graphs are parity-odd:  $(-)^5 = (-) = (-)^7$ , respectively.

Using the set of all the  $2^{\#\mathbf{E}(\gamma_a)}$  orientations of edges in a single graph  $(\gamma_a, \mathbf{E}(\gamma_a))$  from a cocycle  $\gamma$ , select all topologically isomorphic Kontsevich orgraphs.

THEOREM 5. Every orgraph  $\Gamma$ , isomorphic to a Kontsevich orgraph  $\Gamma_0$ , acquires under (1) the sign

$$\text{sign}(\Gamma) = (-)^{\sigma_E} \cdot \text{sign}(\Gamma_0), \quad (5)$$

<sup>5</sup>We verify in App. B that the third graph is equal to minus the second term with edges  $S_0 \rightleftharpoons S_1$  interchanged.

where the permutation  $\sigma_E: \text{Edge}(\Gamma) \simeq \text{Edge}(\Gamma_0)$  of oriented edges is induced by the orgraph isomorphism  $\sigma: \Gamma \simeq \Gamma_0$ .

This is the rule for the count of orgraph multiplicities and signs. For instance, the multiplicity +8 of the first Kontsevich orgraph in the tetrahedral flow (cf. [16, 19] and [2]) is obtained in Example 5 at the end of this section. Examples illustrating how rule (5) works in the various counts of signs will be given in section 3 (see p. 426 below, as well as Example 4 earlier in this section).<sup>6</sup>

*Proof.* Represent the  $\alpha$ th copy of Poisson bi-vector  $\mathcal{P}_{(\alpha)}$  using  $\xi_{p(\alpha)}\xi_{q(\alpha)} \cdot \frac{1}{2}P_{(\alpha)}^{p(\alpha)q(\alpha)}(\mathbf{x})$ , here  $1 \leq \alpha \leq n$ , and collect the set of parity-even pairs  $(\xi\xi)_\alpha$  to the left of the product of bivector coefficients. The edge orienting operator  $\vec{\Delta}$  acts from the left on the string  $(\xi\xi)_1 \cdots (\xi\xi)_n$ ; the topology of unoriented graph  $\gamma_a$  and the choice of  $\vec{\partial}/\partial x_{(i)}^\nu \otimes \vec{\partial}/\partial \xi_{(j)}^{(i)}$  versus  $\vec{\partial}/\partial x_{(j)}^\nu \otimes \vec{\partial}/\partial \xi_{(i)}^{(j)}$  at each pair  $(ij)$  specify the  $n$ th degree differential monomial in coefficients  $P_{(\alpha)}^{p(\alpha)q(\alpha)}$ . (Note that the index  $\nu$  for every edge  $e_{ij}$  is a summation index.)

Under the orgraph isomorphism  $\sigma$ , which unshuffles the vertex pairs  $(\xi\xi)_\alpha$  by an even-parity permutation of the letters  $\xi$ , the parity-odd edge operators  $\Delta_{ij}$  and the two new edges  $S_0, S_1$  are permuted by  $\sigma_E$ . This permutation of edges corresponds to a permutation of the parity-odd letters  $\xi$  that indicate the tails of those edges. Therefore, the two Kontsevich orgraph encodings differ by the sign factor  $(+) \cdot (-)^{\sigma_E} = (-)^{\sigma_E}$ . ■

**COROLLARY 6.** *A skew-symmetry w.r.t. the sinks is guaranteed for bi-vector orgraphs. Indeed, for orgraphs with swapped new edges  $S_0$  and  $S_1$  issued to the sinks  $f, g$  from some vertex (or two distinct vertices) of  $\gamma_a$ , the sign of transposition  $S_0 \rightleftharpoons S_1$  balances the two types of orgraphs: with  $S_0$  to  $f$  and  $S_1$  to  $g$  against orgraphs with the edge  $S_1$  now issued to  $g$  from the old source of  $S_0$ , and the new edge  $S_0$  issued to  $f$  from the old source of the old edge  $S_1$ .*

**LEMMA 7.** *The sign of edge permutation in (5) trivializes the contribution from any zero unoriented graph  $\mathcal{Z}$  to the sum over all possible ways to orient its edges.*

*Proof.* Run over the entire sum of  $2^{\#\text{E}(\mathcal{Z})}$  ways to orient edges in a zero graph  $\mathcal{Z}$ , and select all admissible not identically zero differential monomials similar to a given one, i.e. pick all Kontsevich orgraphs  $\Gamma$  isomorphic to a given one (denote it by  $\Gamma_0$ ). Each isomorphism  $\Gamma \simeq \Gamma_0$  represents an element of the automorphism group  $\text{Aut}(\mathcal{Z})$ . By the definition of zero unoriented graph, the subgroup  $H \leq \mathbb{S}_{2n-2}$  of unoriented edge permutations – under the action of the group  $\text{Aut}(\mathcal{Z})$  – contains at least one parity-odd element  $\sigma_E$ . Therefore, the numbers of parity-even and parity-odd elements in  $H$  coincide. This implies that these  $\#\text{Aut}(\mathcal{Z})$  elements cancel in disjoint pairs, so that the contribution of each topological profile in  $\text{Or}(\mathcal{Z})(\mathcal{P}, \dots, \mathcal{P})$  comes with zero coefficient. ■

*Remark 2.* The wedge orderings Left < Right of outgoing edges at internal vertices of the Kontsevich orgraphs under study play no role in the count of multiplicities and signs!

---

<sup>6</sup>The secret confined in Theorem 5 is that the calculation of edge permutation parities is in fact redundant for finding the signs in front of the Kontsevich orgraphs; this will be seen in the next section.

EXAMPLE 5. Indeed, were the Left  $<$  Right oriented edge orderings contributing with a ‘ $-$ ’ sign factor per vertex whenever Right  $\prec$  Left in the edge ordering  $S_0 \prec S_1 \prec I \prec II \prec \dots$ , the  $\Lambda$ -shaped graph of differential orders  $(3, 1, 1, 1)$  in the tetrahedral flow  $\text{Or}(\gamma_3)(\mathcal{P})$  would accumulate the wrong coefficient  $4 - 4 = 0$  instead of the true value  $4 + 4 = 8$  in the balance  $8 : 24$  for the linear combination  $\mathcal{Q}_{1:\frac{9}{2}}(\mathcal{P})$  of the Kontsevich graphs (see Fig. 2 and Table 1). For instance,  $I^a \simeq V^h$  and  $IV^a \simeq I^h$  so that in the

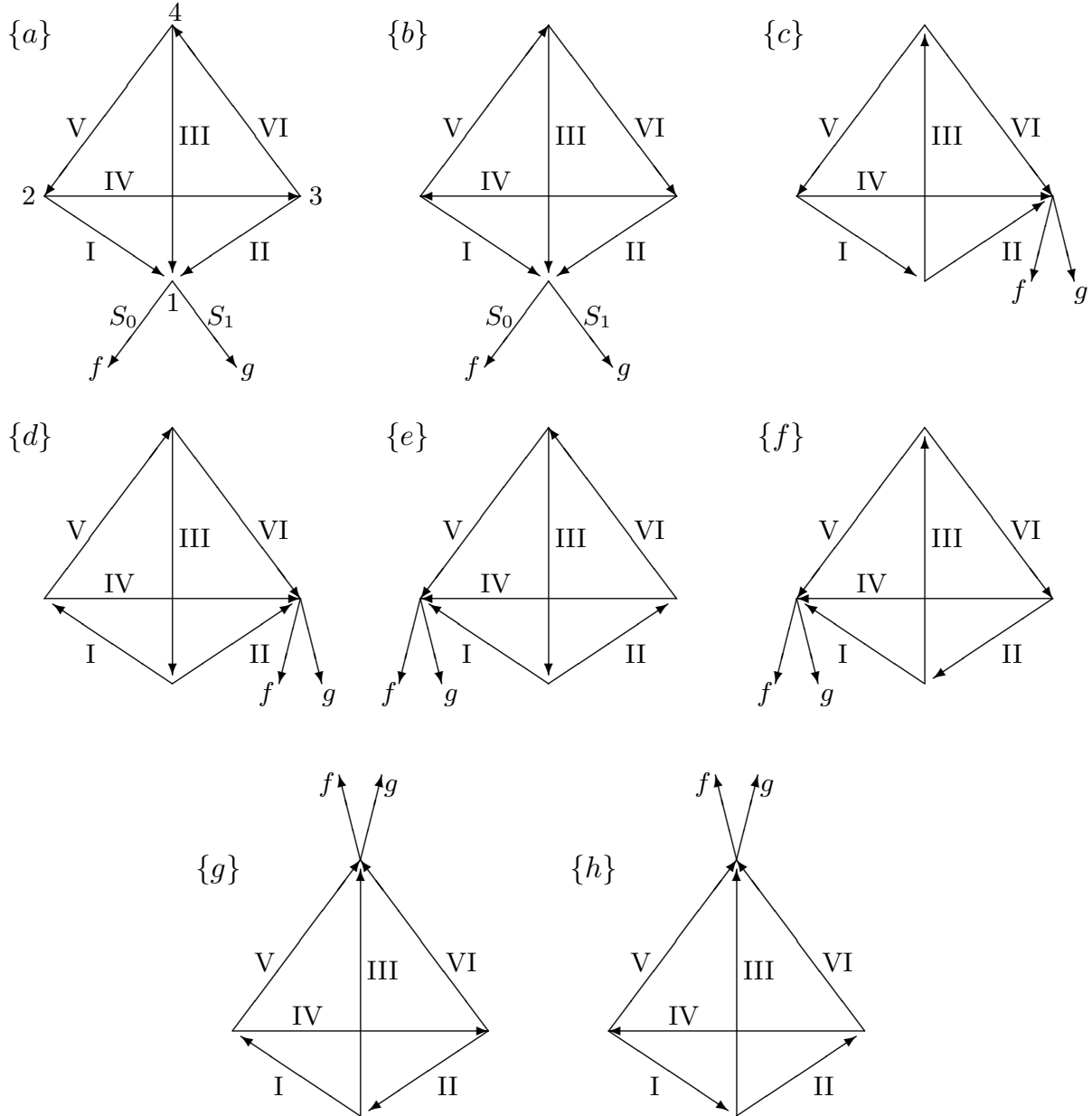


Fig. 2. The eight  $\Lambda$ -shaped ways to orient the tetrahedron:  $\{a\} \div \{h\}$  yield  $+8$ .

orgraph  $\{a\}$ ,  $I^a < IV^a$ , but under the orgraph isomorphism  $\{a\} \simeq \{h\}$ , one obtains the inversion  $V^h < I^h$  at a vertex.<sup>7</sup>

<sup>7</sup>The full list of inversions is this:  $\{c\} V > VI, I > IV$ ;  $\{d\} I > II, III > VI$ ;  $\{e\} III > V$ ;  $\{f\} II > IV$ ;  $\{g\} IV > V, II > VI, I > III$ ;  $\{h\} I > V, II > III, IV > VI$ .

Table 1. Permutations of six edges  $I, \dots, VI$  in the  $\Lambda$ -shaped orientations of the tetrahedron.

							$(-)^{\sigma_E}$	$\#(L > R)$	$(-)^{\#(L > R)}$
$\{a\}$	I	II	III	IV	V	VI	+	0	+
$\{b\}$	I	III	II	V	IV	VI	+	0	+
$\{c\}$	II	VI	IV	III	I	V	+	2	+
$\{d\}$	II	IV	VI	I	III	V	+	2	+
$\{e\}$	IV	I	V	II	VI	III	+	1	-
$\{f\}$	I	V	IV	III	II	VI	+	1	-
$\{g\}$	V	VI	III	IV	I	II	+	3	-
$\{h\}$	V	III	VI	I	IV	II	+	3	-

**3. The rule of signs in terms of Kontsevich orgraphs.** In this section we derive two rules which allow the matching of signs in front of Kontsevich’s orgraphs by simply looking at these graphs (or their encodings), so that no calculation of parities for permutations of *all* the edges is needed. The first rule is specific to  $\Pi$ -shaped orgraphs. The second rule describes the sign factors which are gained in the course of transitions  $\Pi \rightleftharpoons \Pi$ ,  $\Lambda \rightleftharpoons \Pi$ , and  $\Lambda \rightleftharpoons \Lambda$  between the orgraphs of respective shapes, as long as they are taken from the set of all admissible ways to orient a given graph  $\gamma_a$  in a cocycle  $\gamma = \sum_a c_a \cdot \gamma_a$ . The work of both rules is illustrated using the tetrahedral and pentagon-wheel cocycle flows  $\text{Or}(\gamma_3)(\mathcal{P})$  and  $\text{Or}(\gamma_5)(\mathcal{P})$  from [2] and [10], respectively.

**RULE 1.** A  $\Pi$ -shaped orgraph with ordered edge pairs  $(S_0, A)(S_1, B) \dots$  issued from two distinct vertices acquires under (1) the extra sign factor  $(-)$ , compared with a graph with the ordered edge pairs  $(S_0, B)(S_1, A) \dots$ , if  $A \prec B$  in the edge ordering  $E(\gamma_a)$  of a graph to orient.

*Proof.* Indeed,  $S_0 \wedge S_1 \wedge A \wedge B = -S_0 \wedge A \wedge S_1 \wedge B = (-)^2 S_0 \wedge B \wedge S_1 \wedge A = +S_1 \wedge A \wedge S_0 \wedge B$ . ■

**EXAMPLE 6.** Both  $\Pi$ -shaped graphs in the tetrahedral flow (see its encoding in Example 4 in section 2) do acquire a sign factor by Rule 1.

**DEFINITION 2.** The *body* of a Kontsevich orgraph which is obtained by orienting  $\gamma_a$  in a cocycle  $\gamma$  is the set of oriented edges inherited from  $\gamma_a$ , i.e. excluding the new edges  $S_i$  to the sinks.

**RULE 2.** Let  $\Gamma_1$  and  $\Gamma_2$  be two topologically nonisomorphic orgraphs which are obtained by orienting the same graph  $\gamma_a$  in a cocycle  $\gamma$ .<sup>8</sup>

$\Pi \rightleftharpoons \Pi$  If both the orgraphs are  $\Pi$ -shaped, then the sign in front of (the multiplicity of) the orgraph  $\Gamma_2$  is determined from such sign given by (1) for  $\Gamma_1$  by now using the formula

$$\text{sign}(\Gamma_2) = (-)^{\#\{\text{reverses of arrows in the body as } \Gamma_1 \rightarrow \Gamma_2\}} \cdot \text{sign}(\Gamma_1). \tag{6}$$

<sup>8</sup>For instance, such obviously are all the terms in the Kontsevich flow  $\text{Or}(\gamma_3)(\mathcal{P})$  where the tetrahedron  $\gamma_3 \in \ker d$  is oriented, or the orgraphs which one obtains by orienting the pentagon wheel and the prism graph in the Kontsevich–Willwacher cocycle  $\gamma_5$ , cf. [9, 10].

$\Lambda \rightleftharpoons \Pi$  Transitions  $\Lambda \rightleftharpoons \Pi$  yield the product of sign factors  $(-)$   $\times$  formula (6), i.e. the extra  $(-)$  is universal, distinguishing between the shapes.

$\Lambda \rightleftharpoons \Lambda$  Same-shape transitions  $\Lambda \rightleftharpoons \Lambda$  acquire only the sign factor (6).

In other words, the transition  $\Lambda \rightleftharpoons \Pi$  signals the sign factor  $(-)$ , and the number of body arrow reversals contributes in all cases.

*Proof.* **Case  $\Pi \rightleftharpoons \Pi$ .** For the sake of clarity, assume at once that the edge operators  $\vec{\Delta}_{ij}$  corresponding to the edges whose orientation is *not* reversed have already acted on the argument of two operators  $\vec{\Delta}$  corresponding to the two graphs,  $\Gamma_1$  and  $\Gamma_2$ . There remain  $\kappa$  edge operators acting on the product of  $\kappa + 2$  comultiples  $\xi \cdots \xi$  times an even factor formed by the coefficients  $P_{(\alpha)}^{pq}(\mathbf{x})$  of bi-vector copies.

Consider the rightmost operator  $\vec{\Delta}_{ij}$  from what remains; it is the sum  $\vec{\partial}/\partial\xi_{\text{tail}}^{\text{old}} \otimes \vec{\partial}/\partial x_{\text{head}}^{\text{old}}$  and  $\vec{\partial}/\partial\xi_{\text{head}}^{\text{old}} \otimes \vec{\partial}/\partial x_{\text{tail}}^{\text{old}} = \vec{\partial}/\partial\xi_{\text{tail}}^{\text{new}} \otimes \vec{\partial}/\partial x_{\text{head}}^{\text{new}}$ . Because the derivatives  $\vec{\partial}/\partial\mathbf{x}$  have even parity, we focus on the choice of superderivation to orient the edge (resp., fix and then reverse its orientation). In the ordered string  $\xi \cdots \xi$ , let us bring next to each other the symbols  $\xi_i$  and  $\xi_j$  from the copies  $\mathcal{P}_{(i)}$  and  $\mathcal{P}_{(j)}$  contained in the  $i$ th and  $j$ th vertices. It is obvious that the action by  $\vec{\partial}/\partial\xi$  on one such comultiple instead of the other creates the sign factor  $(-)$ . Doing this  $\kappa$  times counts the number of arrow reversions in the body of  $\Pi$ -shaped orgraph, whence  $(-)^{\kappa}$ .

**Case  $\Lambda \rightleftharpoons \Pi$ .** To avoid an agglomeration of symbols, we omit the letters  $\xi$  and display their subscripts, thus indicating either which body edge it is (say  $A$  or  $B$ ,  $A \prec B$ ) or where it goes to ( $S_0 := F$  to the argument  $f$  in the sink  $0$  and  $S_1 := G$  to the argument  $g$  in the sink  $1$ ). Remember that the edge letters  $A$ ,  $B$ ,  $F$ , and  $G$  are parity-odd by construction.

Without loss of generality, let us assume that in the string of  $2n$  comultiples the four rightmost are,

$$\begin{aligned} \text{for the } \Lambda\text{-shaped orgraph:} & \quad A B F G, \\ \text{for the } \Pi\text{-shaped orgraph:} & \quad A F B G. \end{aligned}$$

We see that  $(AB)(FG) = -(AF)(BG)$ , whence we obtain the sought-for universal sign factor  $(-)$  for any transitions between the different shapes  $\Lambda \rightleftharpoons \Pi$  (see Examples 7 and 8 in what follows). Now, the count of body edge reversals goes exactly as before.

**Case  $\Lambda \rightleftharpoons \Lambda$ .** There remains almost nothing to prove: in the above notation, we have that  $ABFG = FGAB$ , hence no extra sign factor is produced when the wedge of two edges directed to sinks is transported from one internal vertex to another.<sup>9</sup> ■

---

<sup>9</sup>This will presently be illustrated in Example 9 by using topologically nonisomorphic  $\Lambda$ -shaped orgraphs in the set of admissible orientations of the pentagon wheel in the Kontsevich–Willwacher cocycle  $\gamma_5$ .

EXAMPLE 7. Consider the r.-h.s.  $\mathcal{Q}_{1:\frac{6}{2}} = \text{Or}(\gamma_3)(\mathcal{P})$  of the Kontsevich tetrahedral flow,

$$\mathcal{Q}_{1:\frac{6}{2}} = (+1) \cdot \underbrace{\left( \begin{matrix} 0 & 1 & 2 & 4 & 2 & 5 & 2 & 3 \\ S_0 & S_1 & I & IV & II & VI & III & V \end{matrix} \right)}_{\Lambda\text{-shaped}} - 3 \cdot \left[ \underbrace{\left( \begin{matrix} 0 & 3 & 1 & 4 & 2 & 5 & 2 & 3 \\ S_0 & I & S_1 & IV & II & VI & III & V \end{matrix} \right)}_{\text{minuend}} + \underbrace{\left( \begin{matrix} 0 & 3 & 4 & 5 & 1 & 2 & 2 & 4 \\ S_0 & I & IV & V & S_1 & II & III & VI \end{matrix} \right)}_{\text{subtrahend}} \right].$$

Using Rules 1 and 2, let us show why the sign which relates the  $\Lambda$ -shaped orgraph to the skew-symmetrisation of  $\Pi$ -shaped orgraph is equal to  $(-)$ ; the count of multiplicities,  $8 : 24 = 1 : 3$ , is standard.<sup>10</sup>

- In the minuend, which is a  $\Pi$ -shaped orgraph, Rule 1 contributes  $-$  for the edge pairs  $(S_0 I) (S_1 IV) \cdots -$  with the first factor  $(-)$ .
- In the course of transition  $\Lambda \rightleftharpoons \Pi$  to the minuend, one arrow in the body of orgraph is reversed (namely, it is the edge  $I$  bridging the edges  $S_0$  and  $S_1$  issued from the vertices 2 and 3), whence another minus sign,  $(-) = (-)^1$ .
- The transition  $\Lambda \rightleftharpoons \Pi$  itself contributes with a universal sign  $(-)$ , see Rule 2 again.

In total, we accumulate the sign factor  $(-) \cdot (-) \cdot (-) = (-)$ , which indeed is the sign that relates the skew-symmetric orgraphs in the flow  $\dot{\mathcal{P}} = \text{Or}(\gamma_3)(\mathcal{P})$ .

EXAMPLE 8. Consider two  $\Lambda$ -shaped terms and a  $\Pi$ -shaped term – in Fig. 3 – from the

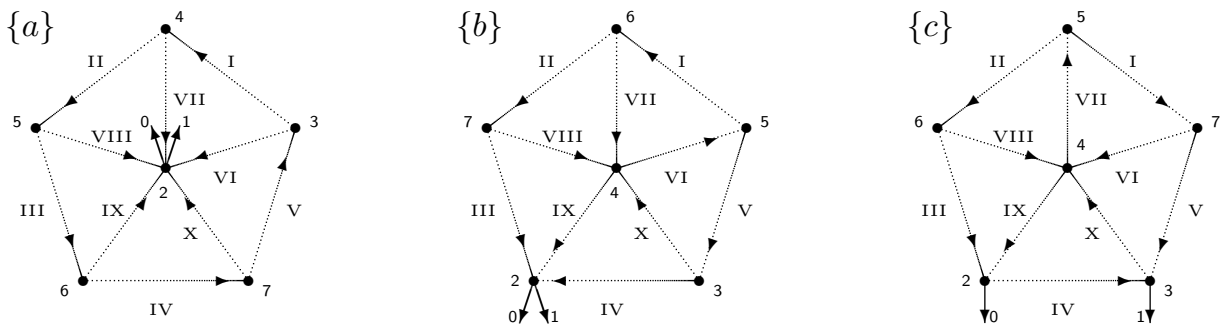


Fig. 3. Several  $\Lambda$ -shaped and  $\Pi$ -shaped terms from the result  $\text{Or}(\gamma_5)$  of orienting to Kontsevich orgraphs the pentagon wheel graph in the cocycle  $\gamma_5$ .

right-hand side  $\mathcal{Q}_5 = \text{Or}(\gamma_5)(\mathcal{P})$  of the flow determined by the Kontsevich–Willwacher

<sup>10</sup>The admissible  $\Lambda$ -shaped orientations of the tetrahedron are obtained by attaching the wedge  $S_0 S_1$  to one of the four vertices and orienting the opposite face using one of two admissible ways, so that  $4 \cdot 2 = 8$ . The  $\Pi$ -shaped Kontsevich graphs are obtained by selecting an edge from six of them, directing it in one of the two ways, and orienting the opposite edge also in one of two ways, whence  $6 \cdot 2 \cdot 2 = 24$ .

pentagon-wheel cocycle  $\gamma_5 \in \ker d$  (see [5, 10] and [9]):

$$\begin{aligned} \mathcal{Q}_5 = & (+2) \cdot \left( \begin{smallmatrix} 0 & 1 & 2 & 4 & 2 & 5 & 2 & 6 & 2 & 7 & 2 & 3 \\ S_0 & S_1 & VI & I & VII & II & VIII & III & IX & IV & X & V \end{smallmatrix} \right) + \\ & + 10 \cdot \left( \begin{smallmatrix} 0 & 1 & 2 & 4 & 2 & 5 & 3 & 6 & 4 & 7 & 2 & 4 \\ S_0 & S_1 & IV & X & IX & VI & V & I & VII & II & III & VIII \end{smallmatrix} \right) + \\ & + 10 \cdot \left( \begin{smallmatrix} 0 & 3 & 1 & 4 & 2 & 5 & 6 & 7 & 2 & 4 & 3 & 4 \\ S_0 & IV & S_1 & X & IX & VII & II & I & III & VIII & V & VI \end{smallmatrix} \right) + \dots \end{aligned}$$

The first and second graphs, which we denote by  $\{a\}$  and  $\{b\}$ , are  $\Lambda$ -shaped whereas the third graph  $\{c\}$  is  $\Pi$ -shaped; there are 167 terms in  $\mathcal{Q}_5$ , of which some are grouped in pairs so that there are 91 bi-vector terms in total: of them, 15 orgraphs are  $\Lambda$ -shaped and the rest,  $\Pi$ -shaped, undergo the skew-symmetrisation.

The transition  $\{a\} \mapsto \{c\}$  employs the following sign matching factors:<sup>11</sup>

- Rule 1 for  $\{c\}$  having  $(S_0 IV)(S_1 X) \dots$  contributes with  $(-)$ .
- The number of arrow reversals in the body of orgraph in the course of transition  $\{a\} \mapsto \{c\}$  equals 4 (specifically, these are edges  $I, V, VII$ , and  $IX$ ), whence  $(-)^4 = (+)$  by Rule 2.
- The transition  $\Lambda \rightleftharpoons \Pi$  between different shapes yields the universal sign factor  $(-)$ .

In total, we have that for  $\{a\} \mapsto \{c\}$ , the overall sign is  $(-) \cdot (+) \cdot (-) = (+)$ .

Counting the parity of three permutations of the edges  $S_0 \prec S_1 \prec I \prec \dots \prec X$  in the graphs  $\{a\}, \{b\}, \{c\}$  is left as an exercise,<sup>12</sup> cf. (5).

EXAMPLE 9. The same-shape  $\Lambda \rightleftharpoons \Lambda$ -transition  $\{a\} \rightleftharpoons \{b\}$  in the pentagon-wheel flow  $\mathcal{Q}_5 = \text{Or}(\gamma_5)(\mathcal{P})$ , see previous example, amounts to the reversal of four arrows in the body of orgraph (specifically, the edges  $IV^{\{b\}} = 2\text{-}3, V^{\{b\}} = 3\text{-}5, VI^{\{b\}} = 4\text{-}5$ , and  $IX^{\{b\}} = 2\text{-}4$ ). Rule 2 tells us at once that the orgraph multiplicities, 2 for  $\{a\}$  and 10 for  $\{b\}$ , are taken with equal signs (here,  $+2 : +10$ ).

Rules 1 and 2 completely determine the signs of all Kontsevich orgraphs (counted with their multiplicities) as long as they are obtained by orienting a given graph  $\gamma_a$  in a cocycle  $\gamma = \sum_a c_a \cdot \gamma_a$ .

Finally, let  $\gamma_a$  and  $\gamma_b$  be topologically nonisomorphic unoriented graphs in a cocycle  $\gamma \in \ker d$  such that the differentials  $d(\gamma_a)$  and  $d(\gamma_b)$  have a least one nonzero unoriented graph in common.

RULE 3. The matching of signs for –clearly, topologically nonisomorphic– Kontsevich orgraphs which appear under (1) in the course of orienting different terms,  $\gamma_a$  and  $\gamma_b$ , in a cocycle  $\gamma = \sum_s c_s \cdot \gamma_s \in \ker d$  is provided by the cocycle itself, that is, by the coefficients  $c_s$  and respective edge orderings  $E(\gamma_a)$  and  $E(\gamma_b)$ .

The signs in front of encodings of all the Kontsevich orgraphs are thus determined for the linear combination  $\text{Or}(\gamma)(\mathcal{P})$ . Now, in each orgraph (and – independently from other orgraphs), one can swap, at a price of the minus sign factor, the Left and Right

<sup>11</sup>This example of transition between orgraphs,  $\{a\} \rightleftharpoons \{c\}$  as well as  $\{b\} \rightleftharpoons \{c\}$ , is instructive also in that the number of inversions, i.e. outgoing edge pairs Left  $<$  Right such that Left  $>$  Right, does change parity in the course of  $\{a\}, \{b\} \rightleftharpoons \{c\}$  (specifically, from 5 and 3 to 2) but does *not* contribute to the signs in front of the orgraph multiplicities.

<sup>12</sup>The respective numbers of elementary transpositions are 10, 24, and 26.



outgoing edges issued from any vertex. For example, this is done during the *normalisation* of encodings, when an orgraph, given in terms of  $n$  pairs of  $n + 2$  target vertices, is realised by using a *minimal* base- $(n + 2)$  positive number.<sup>13</sup>

DEFINITION 3. An *inversion* is a situation where, at a vertex of a Kontsevich orgraph, Left  $\succ$  Right in the overall edge ordering  $S_0 \prec S_1 \prec I \prec II \prec \dots$ .

RULE 4. For any Kontsevich orgraph  $\Gamma$  obtained from  $\Gamma_0$  by relabelling vertices and possibly, for some of the internal vertices, swapping the consecutive order of two edges issued from any such vertex, we have

$$\text{sign}(\Gamma) = \frac{(-)^{\#\text{inversions}(\Gamma)}}{(-)^{\#\text{inversions}(\Gamma_0)}} \cdot \text{sign}(\Gamma_0).$$

Indeed, permutations of vertices induce parity-even permutations in the ordered string of edges, whereas each elementary transposition – within a pair of edges referred to a specific vertex – is parity-odd.

---

*Remark 3.* Apart from the  $\partial_{\mathcal{P}}$ -nontrivial linear scaling  $\dot{\mathcal{P}} = \mathcal{P}$ , the only  $\partial_{\mathcal{P}}$ -(non)trivial, nonlinear and proper ( $\neq 0$ ) flows  $\dot{\mathcal{P}} = \mathcal{Q}(\mathcal{P})$  on spaces of Poisson structures which are known so far (cf. [13]) are only those  $\mathcal{Q} = \text{Or}\vec{\gamma}$  which are obtained by orienting d-cocycles, that is, graphs  $\gamma \in \ker d$  without multiple edges. In consequence, none of the known orgraphs  $\mathcal{Q}(\mathcal{P})$  contains any two-cycles  $\bullet \rightleftarrows \bullet$ . All the more surprising it is that orgraphs which do contain such two-cycles are *dominant* at the order  $\hbar^4$  in the expansion of Kontsevich  $\star$ -product (presumably, so they are at higher orders of the parameter  $\hbar$ ), see [6] and [11, 18].

It would also be interesting to apply the technique of infinitesimal deformations,  $\dot{\mathcal{P}} = \text{Or}\vec{\gamma}(\mathcal{P})$ , of Poisson structures  $\mathcal{P}$  by using graph complex cocycles  $\gamma$  and the orientation morphism  $\text{Or}\vec{\gamma}$ , and the technique of formal deformations  $\mathcal{P} \mapsto \mathcal{P}[\hbar]$  of Poisson structures by using the noncommutative  $\star$ -product (see [16, 18] and [11]) to deformations and *deformation* quantisation of minimal surfaces which are specified by the Schild action functional [1].

**A. The proof of Proposition 4.** The Lie bracket of unoriented graphs  $(\gamma_1, E(\gamma_1))$  and  $(\gamma_2, E(\gamma_2))$  on  $n_i$  vertices and  $2n_i - 2$  edges in each term is, effectively,

$$(\gamma_1 \circ \gamma_2 - \gamma_1 \bar{\circ} \gamma_2, E(\gamma_1) \wedge E(\gamma_2)).$$

The commutator  $[\vec{d}/d\varepsilon_1, \vec{d}/d\varepsilon_2](\mathcal{P})$  of the flows  $\frac{d}{d\varepsilon_i}(\mathcal{P}) = \mathcal{Q}_i(\mathcal{P}) = \text{Or}\vec{\gamma}_i(\mathcal{P}^{\otimes n_i})$  amounts to the consecutive insertions of the bi-vector  $\mathcal{Q}_i(\mathcal{P})$  instead of a copy of the bi-vector  $\mathcal{P}$  in one vertex of the orgraph  $\mathcal{Q}_{2-i}$ . The claim is that

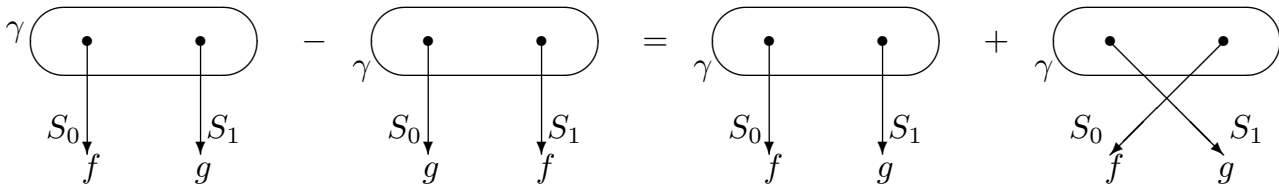
$$[\text{Or}\vec{\gamma}_1(\mathcal{P}^{\otimes n_1}), \text{Or}\vec{\gamma}_2(\mathcal{P}^{\otimes n_2})]_{\text{flows}} = \text{Or}\vec{([\gamma_1, \gamma_2]_{\text{graphs}})}(\mathcal{P}^{\otimes n_1+n_2-1}). \tag{7}$$

---

<sup>13</sup>The normalisation of orgraph encodings, which can be performed independently for different graphs, can actually make it harder to count the number of arrow reverses in the course of transitions which are controlled by Rule 2.

Let us show that the minuend in the left-hand side is equal to the minuend in the right-hand side, and the same for the subtrahends.

**The left-hand side.** The two orgraphs  $\gamma_i$  are oriented independently from each other. The resulting orgraphs are built of wedges (such that the body edges get oriented in all admissible ways). Every such Kontsevich orgraph either is automatically skew-symmetric w.r.t. the content  $f, g$  of sinks (*i.e.* with respect to the ordered pair of arguments of this bi-vector) or it is skew-symmetrised by the mechanism which already worked in Corollary 6. Namely, the *difference* of orgraphs with the identical labelling of ordered edges,  $S_0 \prec S_1 \prec E(\gamma)$  or  $E(\gamma) \prec S_0 \prec S_1$ , but with the content of the two sinks



swapped is equal to the *sum* of orgraphs with the identical labelling of the body edges but with the tails of the arrows  $S_0$  (heading to  $f$ ) and  $S_1$  (heading to  $g$ ) swapped.

**The right-hand side: minuend.** Summing over vertices and attachments, we replace a vertex  $v_0$  in the “victim” graph  $\gamma_2$  by the graph  $\gamma_1$  on  $n_1$  vertices and  $2n_1 - 2$  edges. In the graph  $\gamma_2$ , the edges which were incident to the blown-up vertex  $v_0$  are now attached – in all possible ways – to some vertices of the inserted graph  $\gamma_1$ . Note that if two such edges,  $vu$  and  $v'u$  now connect two distinct vertices,  $v$  and  $v'$ , of the victim graph  $\gamma_2$  with the same vertex  $u$  of the graph  $\gamma_1$ , then one of the two edges precedes the other with respect to the old edge ordering in the victim graph. Likewise, if two such edges,  $vv_0$  and  $v'v_0$  (for which the ordering was defined), now connect by  $vu$  and  $v'u'$  two distinct vertices,  $v$  and  $v'$ , in the victim graph  $\gamma_2$  with two distinct vertices,  $u$  and  $u'$  in the graph  $\gamma_1$ , then the insertion  $\gamma_1 \vec{\sigma} \gamma_2$  contains another graph in which the only difference from the above is that the two edges  $vv_0$  and  $v'v_0$  become  $vu'$  and  $v'u$  (but all the other edges  $v_0w$  in the body of the victim graph  $\gamma_2$  are attached to vertices of the graph  $\gamma_1$  in the same way as they are in the former case).

In every term of the graph  $\gamma_1 \vec{\sigma} \gamma_2$ , consider the subgraph  $\gamma_1$ ; it remains intact in the course of insertion  $\vec{\sigma}$ . When the big graph  $\gamma_1 \vec{\sigma} \gamma_2$  is oriented by  $\text{Or}(\cdot)(\mathcal{P})$ , so is the subgraph  $\gamma_1$ . There were  $2n_1 - 2$  edges in the (body of the) graph  $\gamma_1$ ; none of these edges, still between two vertices of the (sub)graph  $\gamma_1$ , can be oriented using any wedge issued from a vertex of the outer graph  $\gamma_2$ . This implies that exactly  $2n_1 - 2$  arrows belonging to the  $n_1$  bi-vector wedges are spent on orienting the body of the subgraph  $\gamma_1$  in the big graph  $\gamma_1 \vec{\sigma} \gamma_2$ . Only two arrows leave the subgraph  $\gamma_1$ : they head either to one or two sinks of the orgraph  $\text{Or}(\gamma_1 \vec{\sigma} \gamma_2)$  or to a vertex<sup>14</sup> or two vertices in the rest of the victim graph  $\gamma_2$ , *i.e.* excluding the blown-up vertex  $v_0$ . All the other edges which were of

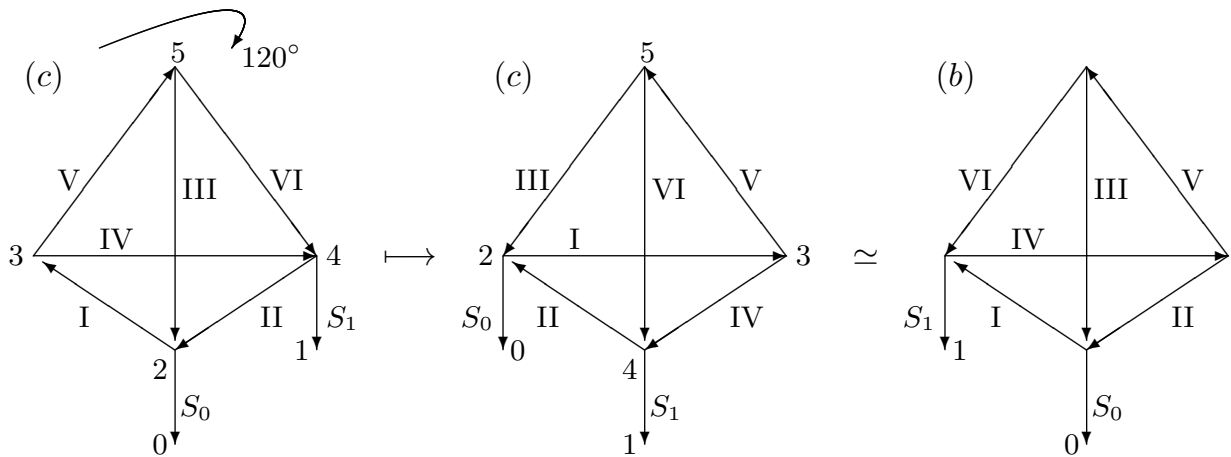
<sup>14</sup>It cannot be that two arrows,  $\vec{uv}$  and  $\vec{u'v}$ , from the subgraph  $\gamma_1$  head towards the same vertex  $v$  in the victim graph  $\gamma_2$  because, with regards to the old topology of  $\gamma_2$  in which a vertex  $v_0$  will be replaced by the graph  $\gamma_1$ , this would mean a double edge  $v_0v$ , hence  $\gamma_2$  was a zero graph.

the form  $v_0v$  in the graph  $\gamma_2$  now become arrows  $\overrightarrow{vu}$  heading towards vertices  $u$  of the subgraph  $\gamma_1$  in the big graph  $\gamma_1 \vec{\sigma} \gamma_2$ .

To establish the equality of the minuend in the left-hand side of (7) to the minuend in the right-hand side of that formula, it remains to recall that by construction, all body edges of the graph  $\gamma_2$  antecede those of  $\gamma_1$  (and *vice versa*: body edges of the graph  $\gamma_1$  precede those of the graph  $\gamma_2$ ), so that now, the ordering  $S_0^{(1)} \prec S_1^{(1)}$  of the arrows which are issued to the arguments of the bi-vector  $\mathcal{Q}_1(\mathcal{P}) = \text{Or}(\gamma_1)(\mathcal{P})$  is always dictated by the ordering  $E(\gamma_2) \wedge S_0^{(2)} \wedge S_1^{(2)}$  of two edges from the (or)graph  $\text{Or}(\gamma_2)$ .

The subtrahends in which the graph  $\gamma_2$  is inserted into some vertex of the graph  $\gamma_1$  are processed in an analogous way. The proof is complete.

**B. The tetrahedron: its  $\Pi$ -shaped orientation skew-symmetrized.** The edge orderings  $E(c) = (S_0 \wedge S_1 \wedge I \wedge \dots \wedge VI)^{(c)}$  and  $E(b) = (S_0 \wedge S_1 \wedge I \wedge \dots \wedge VI)^{(b)}$  are related



by the equalities  $S_0^{(c)} = S_1^{(b)}$ ,  $S_1^{(c)} = S_0^{(b)}$ ,  $I^{(c)} = IV^{(b)}$ ,  $II^{(c)} = I^{(b)}$ ,  $III^{(c)} = V^{(b)}$ ,  $IV^{(c)} = II^{(b)}$ ,  $V^{(c)} = VI^{(b)}$ , and  $VI^{(c)} = III^{(b)}$ , whence one easily verifies that

$$E(c) = -(-)^6 E(b),$$

the leading minus coming from the relabelling  $S_0 \rightleftharpoons S_1$  and the rest from the permutation of body edges. We conclude that the arithmetic sum of two Kontsevich orgraphs (b–c) in Fig. 1 on p. 422 is the skew-symmetrisation of the  $\Pi$ -shaped orientation of the tetrahedron  $\gamma_3$  by using arrow wedges.

**Acknowledgements.** The authors thank the Organisers of international workshop ‘Homotopy algebras, deformation theory and quantization’ (16–22 September 2018 in Bedlewo, Poland) for helpful discussions and warm atmosphere during the meeting. The authors are grateful to the anonymous referee for remarks and suggestions, and to G. Felder, S. Gutt, and M. Kontsevich for helpful discussion. A part of this research was done while RB was visiting at RUG and AVK was visiting at the IHÉS in Bures-sur-Yvette, France and at the JGU Mainz (supported by IM JGU via project 5020 and JBI RUG project 106552). The research of AVK was supported by the IHÉS (partially, by the Nokia Fund).

## References

- [1] J. Arnlind, J. Hoppe, M. Kontsevich, *Quantum minimal surfaces*, arXiv:1903.10792 [math-ph]
- [2] A. Bouisaghouane, R. Buring, A. Kiselev, *The Kontsevich tetrahedral flow revisited*, J. Geom. Phys. 119 (2017), 272–285. (Preprint arXiv:1608.01710 [q-alg])
- [3] F. Brown, *Mixed Tate motives over  $\mathbb{Z}$* , Annals of Math. 175 (2012), 949–976.
- [4] R. Buring, A. V. Kiselev, *On the Kontsevich  $\star$ -product associativity mechanism*, PEPAN Letters 14:2 *Supersymmetry and Quantum Symmetries'2015* (2017), 403–407. (Preprint arXiv:1602.09036 [q-alg])
- [5] R. Buring, A. V. Kiselev, *The orientation morphism: from graph cocycles to deformations of Poisson structures*, in: J. Phys.: Conf. Ser. 1194 (2019), Proc. 32nd Int. colloquium on Group-theoretical methods in Physics: GROUP32 (9–13 July 2018, CVUT Prague, Czech Republic), Paper 012017, 1–10. (Preprint arXiv:1811.07878 [math.CO])
- [6] R. Buring, A. V. Kiselev, *The expansion  $\star \bmod \bar{o}(\hbar^4)$  and computer-assisted proof schemes in the Kontsevich deformation quantization*, Experimental Math. (2019), 54 p., in press, doi:10.1080/10586458.2019.1680463. (Preprint IHÉS/M/17/05, arXiv:1702.00681 [math.CO])
- [7] R. Buring, A. V. Kiselev, *Formality morphism as the mechanism of  $\star$ -product associativity: how it works*, Collection of works Inst. Math., Kyiv 16:1 (2019), *Symmetry & Integrability of Equations of Mathematical Physics*, 22–43. (Preprint arXiv:1907.00639 [q-alg])
- [8] R. Buring, A. V. Kiselev, *Universal cocycles and the graph complex action on homogeneous Poisson brackets by diffeomorphisms*, PEPAN Letters (in press) *Supersymmetry and Quantum Symmetries'2019* (2020), 8 pages. (Preprint IHÉS/M/19/20 (2019), arXiv:1912.12664 [math.SG])
- [9] R. Buring, A. V. Kiselev, N. J. Rutten, *The heptagon-wheel cocycle in the Kontsevich graph complex*, J. Nonlin. Math. Phys. 24 (2017), Suppl. 1 ‘Local & Nonlocal Symmetries in Mathematical Physics’, 157–173. (Preprint arXiv:1710.00658 [math.CO])
- [10] R. Buring, A. V. Kiselev, N. J. Rutten, *Poisson brackets symmetry from the pentagon-wheel cocycle in the graph complex*, Physics of Particles and Nuclei 49:5 *Supersymmetry and Quantum Symmetries'2017* (2018), 924–928. (Preprint arXiv:1712.05259 [math-ph])
- [11] A. S. Cattaneo, G. Felder, *A path integral approach to the Kontsevich quantization formula*, Comm. Math. Phys. 212:3 (2000), 591–611. (Preprint arXiv:q-alg/9902090)
- [12] C. Jost, *Globalizing  $L_\infty$ -automorphisms of the Schouten algebra of polyvector fields*, Differential Geom. Appl. 31:2 (2013), 239–247. (Preprint arXiv:1201.1392 [q-alg])
- [13] A. V. Kiselev, *Open problems in the Kontsevich graph construction of Poisson bracket symmetries*, in: J. Phys.: Conf. Ser. 1416 (2019), Proc. XXVI Int. conf. ‘Integrable Systems & Quantum Symmetries’ (8–12 July 2019, CVUT Prague, Czech Republic), Paper 012018, 1–8. (Preprint arXiv:1910.05844 [math-ph])
- [14] M. Kontsevich, *Feynman diagrams and low-dimensional topology*, in: *First Europ. Congr. of Math.* 2 (Paris, 1992), Progr. Math. 120, Birkhäuser, Basel, 1994, 97–121.
- [15] M. Kontsevich, *Homological algebra of mirror symmetry*, in: *Proc. Intern. Congr. Math.* 1 (Zürich, 1994), Birkhäuser, Basel, 1995, 120–139.
- [16] M. Kontsevich, *Formality conjecture*, in: *Deformation theory and symplectic geometry* (Ascona 1996), D. Sternheimer, J. Rawnsley and S. Gutt (eds.), Math. Phys. Stud. 20, Kluwer Acad. Publ., Dordrecht, 1997, 139–156.
- [17] M. Kontsevich, *Operads and motives in deformation quantization*, Lett. Math. Phys. 48:1

- (1999), *Moshé Flato (1937–1998)*, 35–72. (Preprint arXiv:math.QA/9904055)
- [18] M. Kontsevich, *Deformation quantization of Poisson manifolds*, Lett. Math. Phys. 66:3 (2003), 157–216. (Preprint arXiv:q-alg/9709040)
- [19] M. Kontsevich, *Derived Grothendieck–Teichmüller group and graph complexes [after T. Willwacher]*, in: Séminaire Bourbaki (69ème année, 2016–2017), Janvier 2017, No. 1126 (2017), 183–212.
- [20] N. J. Rutten, A. V. Kiselev, *The defining properties of the Kontsevich unoriented graph complex*, in: J. Phys.: Conf. Ser. 1194 (2019), Proc. 32nd Int. colloquium on Group-theoretical methods in Physics: GROUP32 (9–13 July 2018, CVUT Prague, Czech Republic), Paper 012095, 1–10. (Preprint arXiv:1811.10638 [math.CO])
- [21] T. Willwacher, *M. Kontsevich’s graph complex and the Grothendieck–Teichmüller Lie algebra*, Invent. Math. 200:3 (2015), 671–760. (Preprint arXiv:1009.1654 [q-alg])
- [22] T. Willwacher, M. Živković, *Multiple edges in M. Kontsevich’s graph complexes and computations of the dimensions and Euler characteristics*, Adv. Math. 272 (2015), 553–578. (Preprint arXiv:1401.4974 [q-alg])



# Chapter 19

## Universal cocycles and the graph complex action on homogeneous Poisson brackets by diffeomorphisms

This chapter is based on the peer-reviewed journal publication *R. Buring and A.V. Kiselev, Physics of Particles and Nuclei Letters* **17**:5 Supersymmetry and Quantum Symmetries 2019, 707–713, 2020. (Preprint [arXiv:1912.12664](https://arxiv.org/abs/1912.12664) [math.SG] – 8 p.) This result was presented to M. Kontsevich at the IHÉS in December 2019.

*Commentary.* In reference to Part I of the dissertation, the material of this chapter is used in Chapter 2 (especially §2.6), Chapter 7 (§7.1), and Chapter 8. The claim in the end of this chapter (see Proposition 5 below) about the Poisson non-triviality—in the class of differential polynomials—for the restriction of tetrahedral flow to the class of rescaled Nambu–Poisson structures in 3D was false, as seen from the next chapter (and from §7.1.5 in Part I). Besides, let us give a counterexample (in 2D) when a Poisson structure (with non-polynomial coefficients) is homogeneous with respect to a vector field (with polynomial coefficients) but its tetrahedral flow is *not* homogeneous in that way.

**Counterexample.** On  $\mathbb{R}^2$  with coordinates  $x, y$ , let  $P = x^2 y^2 \exp(1/x) \partial_x \wedge \partial_y$  and  $V = -x^2 \partial_x - y^2 \partial_y$ . Then  $\llbracket V, P \rrbracket = P$  but  $\llbracket V, Q_{\text{tetra}}(P) \rrbracket$  is *not* proportional to  $Q_{\text{tetra}}(P)$ . Indeed, we have

$$Q_{\text{tetra}}(P) = -32(6x - 1)x^2 y^5 \exp(4/x) \partial_x \wedge \partial_y$$

and

$$\llbracket V, Q_{\text{tetra}}(P) \rrbracket = 32(6x^2 + 18xy - 24x - 3y + 4)x^2 y^5 \exp(4/x) \partial_x \wedge \partial_y,$$

where  $6x^2 + 18xy - 24x - 3y + 4$  is not divisible by  $6x - 1$ .

# Universal cocycles and the graph complex action on homogeneous Poisson brackets by diffeomorphisms

R. Buring<sup>\*,¶</sup> A. V. Kiselev<sup>†‡§,ℒ</sup>

E-mail: ¶rburing@uni-mainz.de, ℒA.V.Kiselev@rug.nl

## Abstract

The graph complex acts on the spaces of Poisson bi-vectors  $\mathcal{P}$  by infinitesimal symmetries. We prove that whenever a Poisson structure is homogeneous, i.e.  $\mathcal{P} = L_{\vec{V}}(\mathcal{P})$  w.r.t. the Lie derivative along some vector field  $\vec{V}$ , but not quadratic (the coefficients of  $\mathcal{P}$  are not degree-two homogeneous polynomials), and whenever its velocity bi-vector  $\dot{\mathcal{P}} = \mathcal{Q}(\mathcal{P})$ , also homogeneous w.r.t.  $\vec{V}$  by  $L_{\vec{V}}(\mathcal{Q}) = n\mathcal{Q}$  whenever  $\mathcal{Q}(\mathcal{P}) = \text{Or}(\gamma)(\mathcal{P}^{\otimes n})$  is obtained using the orientation morphism  $\text{Or}$  from a graph cocycle  $\gamma$  on  $n$  vertices and  $2n - 2$  edges, then the 1-vector  $\vec{\mathcal{X}} = \text{Or}(\gamma)(\vec{V} \otimes \mathcal{P}^{\otimes n-1})$  is a Poisson cocycle. Its construction is uniform for all Poisson bi-vectors  $\mathcal{P}$  satisfying the above assumptions, on all finite-dimensional affine manifolds  $M$ . Still, if the bi-vector  $\mathcal{Q} \neq 0$  is exact in the respective Poisson cohomology, so there exists a vector field  $\vec{y}$  such that  $\mathcal{Q}(\mathcal{P}) = \llbracket \vec{y}, \mathcal{P} \rrbracket$ , then the universal cocycle  $\vec{\mathcal{X}}$  does not belong to the coset of  $\vec{y}$  mod  $\ker \llbracket \mathcal{P}, \cdot \rrbracket$ . We illustrate the construction using two examples of cubic-coefficient Poisson brackets associated with the  $R$ -matrices for the Lie algebra  $\mathfrak{gl}(2)$ .

**Introduction.** Bi-vector cocycles  $\mathcal{Q}(\mathcal{P}) = \text{Or}(\gamma)(\mathcal{P}^{\otimes n}) \in \ker \llbracket \mathcal{P}, \cdot \rrbracket$  are obtained by Kontsevich's graph orientation morphism  $\text{Or}$  from graph cocycles  $\gamma$  on  $n$  vertices and  $2n - 2$  edges in a way which is uniform for all finite-dimensional affine Poisson manifolds  $(M^r, \mathcal{P})$ . The (non)triviality of cocycles  $\mathcal{Q}(\mathcal{P})$  in the second Poisson cohomology w.r.t. the differential  $\partial_{\mathcal{P}} = \llbracket \mathcal{P}, \cdot \rrbracket$  remains an open problem, twenty-five years after the discovery of the graph complex and orientation morphism (see [11]). In all the Poisson geometries probed so far, the known infinitesimal symmetries  $\dot{\mathcal{P}} = \mathcal{Q}(\mathcal{P})$  of the Jacobi identity  $\frac{1}{2} \llbracket \mathcal{P}, \mathcal{P} \rrbracket = 0$  are  $\partial_{\mathcal{P}}$ -exact: there always exists a vector field  $\vec{y}$  such that  $\mathcal{Q}(\mathcal{P}) = \llbracket \vec{y}, \mathcal{P} \rrbracket$ . The evolution  $\mathcal{P}(\varepsilon = 0) \mapsto \mathcal{P}(\varepsilon > 0)$  of the tensor  $\mathcal{P}$  then amounts to its reparametrisations under the diffeomorphisms of Poisson manifold which are induced by the shifts along the integral trajectories of the vector field  $\vec{y}$ . This is why, instead of producing new Poisson brackets from a given one, the Kontsevich graph flows on the spaces of Poisson bi-vectors induce (non)linear

\*Mathematical Institute, Johannes Gutenberg University of Mainz, Staudingerweg 9, D-55128 Germany. A part of this research was done while R.B. was visiting at the IHÉS, supported by MI JGU project 5020.

†Bernoulli Institute for Mathematics, Computer Science & Artificial Intelligence, University of Groningen, P.O. Box 407, 9700 AK Groningen, The Netherlands.

‡Current address: Institut des Hautes Études Scientifiques (IHÉS), Le Bois-Marie, 35 route de Chartres, Bures-sur-Yvette, 91440 France.

§Supported by BI RUG project 135110 (Groningen) and the IHÉS (in part, by the Nokia Fund).



diffeomorphisms of the base manifold  $M$ , although no more than its affine structure was the initial assumption and no possibility of smooth coordinate reparametrizations was presumed.

For a much used class of (scaling-)homogeneous Poisson bi-vectors  $\mathcal{P} = L_{\vec{V}}(\mathcal{P})$ , we obtain an explicit formula,  $\vec{\mathcal{X}} = \text{Or}(\gamma)(\vec{V} \otimes \mathcal{P}^{\otimes n-1})$ , of a 1-vector cocycle  $\vec{\mathcal{X}}(\gamma, \vec{V}, \mathcal{P}) \in \ker[[\mathcal{P}, \cdot]]$  which is built from the graph cocycles  $\gamma$  uniformly for all homogeneous Poisson bi-vectors  $\mathcal{P}$  on affine manifolds  $M^{r < \infty}$ . The cocycle  $\vec{\mathcal{X}}$  is however not necessarily a 1-vector representative of the coset  $\vec{\mathcal{Y}} \bmod \{\vec{\mathcal{Z}} \in \ker[[\mathcal{P}, \cdot]]\}$  which would trivialise the value  $\mathcal{Q}(\mathcal{P}) = [[\vec{\mathcal{Y}}, \mathcal{P}]]$  of Kontsevich's symmetries at homogeneous Poisson structures. Indeed, the Poisson cocycle  $\mathcal{Q}(\mathcal{P})$  can be, we show, a nonzero bi-vector on  $M^r$ , whereas the bi-vector  $[[\vec{\mathcal{X}}, \mathcal{P}]]$  is identically zero on  $M^r$  by construction. We contrast the formulas of universal cocycles  $\vec{\mathcal{X}}(\gamma, \vec{V}, \mathcal{P})$  and trivialising vector fields  $\vec{\mathcal{Y}}$  for nonzero symmetries  $\dot{\mathcal{P}} = \text{Or}(\gamma)(\mathcal{P})$  by two examples, namely, using cubic-coefficient Poisson brackets associated with the  $R$ -matrices for  $\mathfrak{gl}(2)$ .

This paper is organized as follows. In §1 we recall elements of Poisson cohomology theory in the context of Kontsevich's universal deformations of bi-vectors by using the unoriented graph cocycles. In §2 we phrase the notion of structures which are homogeneous w.r.t. a 1-vector field, and we prove the main theorem. Finally, we illustrate the result (cf. [10]).

**1. Poisson cohomology and the graph complex.** A Poisson bracket  $\{\cdot, \cdot\}_{\mathcal{P}}$  on a real manifold  $M$  is a bi-linear skew-symmetric bi-derivation which takes  $C^\infty(M) \times C^\infty(M) \rightarrow C^\infty(M)$  and satisfies the Jacobi identity  $\frac{1}{2} \sum_{\sigma \in S_3} \{\{\sigma(f), \sigma(g)\}_{\mathcal{P}}, \sigma(h)\}_{\mathcal{P}} = 0$  for any  $f, g, h \in C^\infty(M)$ . The fact that both the arguments  $f, g$  and their bracket  $\{f, g\}_{\mathcal{P}}$  are scalars dictates the tensor transformation law of the components  $\mathcal{P}^{ij}$  of a bi-vector  $\mathcal{P} = \sum_{i,j} \mathcal{P}^{ij}(\mathbf{x}) \partial_i \otimes \partial_j = \frac{1}{2} \sum_{i,j} \mathcal{P}^{ij}(\mathbf{x})(\partial_i \otimes \partial_j - \partial_j \otimes \partial_i) = \frac{1}{2} \sum_{i,j} \mathcal{P}^{ij} \partial_i \wedge \partial_j$  whenever the structure is referred to a system of coordinates  $\mathbf{x} = (x^1, \dots, x^r)$  and  $\partial_i = \partial/\partial x^i$  is a shorthand notation.

The calculus on the space of multivectors  $\Gamma(\wedge^\bullet TM) \cong C^\infty(\Pi T^*M)$  is simplified if one uses the parity-odd coordinates  $\xi_i$  along the directions  $dx^i$  in the fibres of the cotangent bundle  $T^*M$  over points  $\mathbf{a} \in M$  (which are parametrized by  $x^i$ ). The symbol  $\xi_i$  thus corresponds to  $\partial/\partial x^i$  dual to  $dx^i$ , and bi-vectors are  $\mathcal{P} = \frac{1}{2} \sum_{i,j} \mathcal{P}^{ij} \xi_i \xi_j$ , so that  $\{f, g\}_{\mathcal{P}}(\mathbf{a}) = (f) \vec{\partial}/\partial x^\mu \cdot \vec{\partial}/\partial \xi_\nu (\mathcal{P}) \vec{\partial}/\partial \xi_\nu \cdot \vec{\partial}/\partial x^\nu (g)$ ; here, both the coefficients  $\mathcal{P}^{ij}$  and derivatives  $\partial/\partial x^k$  are evaluated at the point  $\mathbf{a} \in M$  as in the left-hand side.<sup>1</sup>

The space of multivectors is endowed with the parity-odd Poisson bracket  $[[\cdot, \cdot]]$  (the Schouten bracket, or antibracket) of own degree  $-1$ . For arbitrary multivectors  $\mathcal{P}, \mathcal{Q}$ , the formula is  $[[\mathcal{P}, \mathcal{Q}]] = (\mathcal{P}) \vec{\partial}/\partial \xi_i \cdot \vec{\partial}/\partial x^i (\mathcal{Q}) - (\mathcal{Q}) \vec{\partial}/\partial x^i \cdot \vec{\partial}/\partial \xi_i (\mathcal{P})$ ; in particular,  $[[\vec{\mathcal{X}}, \vec{\mathcal{Y}}]] = [[\vec{\mathcal{X}}, \vec{\mathcal{Y}}]]$  is the usual commutator of vector fields  $\vec{\mathcal{X}}, \vec{\mathcal{Y}}$  on  $M$ . The Schouten bracket  $[[\cdot, \cdot]]$  is shifted-graded skew-symmetric:  $[[\mathcal{Q}, \mathcal{P}]] = -(-)^{(|\mathcal{P}|-1) \cdot (|\mathcal{Q}|-1)} [[\mathcal{P}, \mathcal{Q}]]$  for  $\mathcal{P}$  and  $\mathcal{Q}$  grading-homogeneous. This is why, unlike the tautology  $[[\vec{\mathcal{X}}, \vec{\mathcal{X}}]] \equiv 0$ , the equation  $[[\mathcal{P}, \mathcal{P}]] = 0$  is a nontrivial restriction for bi-vectors  $\mathcal{P}$ , containing the tri-vector in the l.-h.s. of the Jacobi identity  $\frac{1}{2} [[\mathcal{P}, \mathcal{P}]](f, g, h) = 0$  for the bracket  $\{f, g\}_{\mathcal{P}} = [[f, \mathcal{P}], g]$ . The Schouten bracket itself satisfies the graded Jacobi identity  $[[\mathcal{P}, [[\mathcal{Q}, \mathcal{R}]]]] - (-)^{(|\mathcal{P}|-1) \cdot (|\mathcal{Q}|-1)} [[\mathcal{Q}, [[\mathcal{P}, \mathcal{R}]]]] = [[[\mathcal{P}, \mathcal{Q}], \mathcal{R}]]$  with  $\mathcal{P}$  and  $\mathcal{Q}$  grading-homogeneous. This identity implies that for Poisson bi-vectors  $\mathcal{P}$ , their adjoint action by  $\partial_{\mathcal{P}} = [[\mathcal{P}, \cdot]]$  is a differential of degree  $+1$  on the space of multivectors on  $M$ . The Poisson differential  $\partial_{\mathcal{P}}$  gives rise to the Poisson cohomology  $H_{\mathcal{P}}^i(M)$  of the manifold  $M$  (see [13]).<sup>2</sup>

<sup>1</sup>The dot  $\cdot$  denotes the coupling of iterated variations of the objects  $f, \mathcal{P}$ , and  $g$  with respect to the canonically conjugate variables  $x^i$  and  $\xi_i$ , see [9] and references therein.

<sup>2</sup>The group  $H_{\mathcal{P}}^0(M)$  spans the Casimirs, i.e. the functions which Poisson-commute with any  $f \in C^\infty(M)$ ; the group  $H_{\mathcal{P}}^1(M)$  consists of vector fields which preserve the Poisson structure but do not amount to the Hamiltonian vector fields  $\vec{\mathcal{X}}_h = [[\mathcal{P}, h]]$ ; the second group  $H_{\mathcal{P}}^2(M) \ni \mathcal{Q}$  contains infinitesimal symmetries  $\mathcal{P} \mapsto \mathcal{P} + \varepsilon \mathcal{Q} + \vec{\partial}(\varepsilon)$  of Poisson bi-vectors, whereas the next group  $H_{\mathcal{P}}^3(M)$  stores the obstructions to formal

If a bi-vector  $\mathcal{Q} = \llbracket \vec{\mathcal{X}}, \mathcal{P} \rrbracket$  is a trivial Poisson cocycle, then it certainly is an infinitesimal symmetry of the Jacobi identity  $\frac{1}{2} \llbracket \mathcal{P}, \mathcal{P} \rrbracket = 0$ . But the infinitesimal change  $\llbracket \vec{\mathcal{X}}, \mathcal{P} \rrbracket$  of the tensor  $\mathcal{P}$  then amounts to its reparametrisation under the infinitesimal change of coordinates  $\mathbf{x}'(\mathbf{x}) \rightleftharpoons \mathbf{x}(\mathbf{x}')$  along the integral trajectories of the vector field  $\vec{\mathcal{X}}$  on the manifold  $M$ . The following fact is true for all multivectors (regardless of the concept of Poisson cohomology).

**Proposition 1.** *Let  $\mathbf{a} \in M$  be a point of an  $r$ -dimensional manifold and  $\vec{\mathcal{X}} \in \Gamma(TM)$  be a vector field on it. For every  $\varepsilon \in \mathcal{I} \subseteq \mathbb{R}$  such that there is the integral trajectory bringing  $\mathbf{b}(-\varepsilon) := \exp(-\varepsilon \vec{\mathcal{X}})(\mathbf{a})$  to  $\mathbf{a}$  by the  $(+\varepsilon)$ -shift, and for any choice of the  $r$ -tuple  $\mathbf{x} = (x^1, \dots, x^r)$  of local coordinates in a chart  $U_\alpha$  around  $\mathbf{a} \in M$  (and for  $|\varepsilon|$  small enough for the points  $\mathbf{b}(-\varepsilon)$  to not yet run out of the chart  $U_\alpha$ ), introduce a new parametrization<sup>3</sup> for the point  $\mathbf{a}$  by using the new  $r$ -tuple  $\mathbf{x}'$ . By definition, put  $\mathbf{x}'(\mathbf{a}) := \mathbf{x}(\mathbf{b}(-\varepsilon))$ . Let  $\Omega$  be any multi-vector field near  $\mathbf{a}$  on  $M$ . Under the reparametrization  $\mathbf{x}'(\mathbf{x})$ , the speed at which the components of  $\Omega$  at the point  $\mathbf{a}$  change in  $\varepsilon$ , as  $\varepsilon \rightarrow 0$ , equals  $\left. \frac{d}{d\varepsilon} \right|_{\varepsilon=0} \Omega(\mathbf{a}) = \llbracket \vec{\mathcal{X}}, \Omega \rrbracket(\mathbf{a})$ . In particular, a 1-vector field  $\vec{\mathcal{Y}}$  near  $\mathbf{a}$  would change at  $\mathbf{a}$  as fast as its commutator with the vector field  $\vec{\mathcal{X}}$ :  $\left. \frac{d}{d\varepsilon} \right|_{\varepsilon=0} \vec{\mathcal{Y}}(\mathbf{a}) = [\vec{\mathcal{X}}, \vec{\mathcal{Y}}](\mathbf{a})$ .*

The geography of the set of Poisson structures near a given bracket  $\{\cdot, \cdot\}_{\mathcal{P}}$  on a given manifold  $M^r$  is, generally speaking, unknown. All the more it was a priori unclear whether Poisson bi-vectors  $\mathcal{P}$ , irrespective of the dimension  $r \geq 3$ , topology of  $M^r$ , etc., can be infinitesimally shifted by Poisson 2-cocycles  $\mathcal{Q}(\mathcal{P})$ , the construction of which would be universal for all  $\mathcal{P}$ . The discovery of the graph complex in 1993–94 allowed Kontsevich to state (in [11]) the affirmative answer to the above question. Namely, the graph orientation morphism  $\text{Or}(\cdot)(\mathcal{P}) : \ker d \ni \gamma \mapsto \mathcal{Q}(\mathcal{P}) \in \ker \partial_{\mathcal{P}}$  takes graph cocycles on  $n$  vertices and  $2n - 2$  edges in each term (e.g., the tetrahedron, cf. [1, 3, 5, 6]) to Poisson cocycles whenever the bi-vector  $\mathcal{P}$  itself is Poisson. Willwacher [15] revealed that the generators of Drinfeld’s Grothendieck–Teichmüller Lie algebra  $\mathfrak{grt}$  are source of at least countably many such cocycles in the vertex-edge bi-grading  $(n, 2n - 2)$ ; these cocycles are marked by the  $(2\ell + 1)$ -wheel graphs (e.g., see [6, 7]). Brown proved in [2] that, under the Willwacher isomorphism  $\mathfrak{grt} \cong H^0(\text{GRA})$  these graph cocycles with wheels generate a free Lie subalgebra in  $\mathfrak{grt}$ , which means effectively that the iterated commutators of already known cocycles – under the bracket in the differential graded Lie algebra GRA of graphs – would never vanish. The commutator of two cocycles is a cocycle by the Jacobi identity. All of them again being of the bi-grading  $(n, 2n - 2)$ , these graph cocycles determine countably many infinitesimal symmetries of a given Poisson bi-vector  $\mathcal{P}$ ; the construction is uniform for all the geometries  $(M^r, \mathcal{P})$ .

**Lemma 2.** *For a given Poisson bi-vector  $\mathcal{P}$ , the graph orientation mapping  $\text{Or}(\cdot)(\mathcal{P}) : \ker d \ni \gamma \mapsto \mathcal{Q}(\mathcal{P}) \in \ker \partial_{\mathcal{P}}$  is a Lie algebra morphism that takes the bracket of two cocycles in bi-grading  $(n, 2n - 2)$  to the commutator  $[\frac{d}{d\varepsilon_1}, \frac{d}{d\varepsilon_2}](\mathcal{P})$  of two symmetries  $\frac{d}{d\varepsilon_i}(\mathcal{P}) = \mathcal{Q}_i(\mathcal{P})$ .<sup>4</sup>*

By construction, the components of universal symmetry bi-vectors  $\mathcal{Q}(\mathcal{P})$  are differential polynomials w.r.t. the components  $\mathcal{P}^{ij}$  of the Poisson bi-vector  $\mathcal{P}$  that evolves. It can of course be that a graph flow  $\dot{\mathcal{P}} = \text{Or}(\gamma)(\mathcal{P})$  vanishes identically over the manifold  $M^r$

---

integration  $\mathcal{P} \mapsto \mathcal{P}(\varepsilon) = \mathcal{P} + \sum_{k \geq 1} \varepsilon^k \mathcal{Q}_{(k)}$  of infinitesimal symmetries  $\mathcal{Q} = \mathcal{Q}_{(1)}$  to Poisson bi-vector formal power series satisfying  $\llbracket \mathcal{P}(\varepsilon), \mathcal{P}(\varepsilon) \rrbracket = 0$ .

<sup>3</sup>Actually, this is a way to construct new coordinates for *all* points of  $M$  near  $\mathbf{a}$  in  $U_\alpha$ , i.e. not only those which lie on a piece of the integral trajectory of  $\vec{\mathcal{X}}$  passing through  $\mathbf{a}$ .

<sup>4</sup>By Brown [2], the commutator does in general not vanish for Willwacher’s odd-sided wheel cocycles.

whenever  $\mathcal{Q}$  is evaluated at a particular class of Poisson structures  $\mathcal{P}$ .<sup>5</sup> Nevertheless, there is no mechanism which would force a given Kontsevich's graph flow to vanish at all Poisson structures on all manifolds of all dimensions.<sup>6</sup> Independently, it remains an open problem (cf. [10]) whether there is a Poisson manifold  $(M^r, \mathcal{P})$  and a graph cocycle  $\gamma$  such that the Poisson cohomology class of  $\mathcal{Q}(\mathcal{P}) := \text{Or}(\gamma)(\mathcal{P})$  would be *nontrivial* in  $H_{\mathcal{P}}^2(M)$ . In other words, for all the shifts  $\mathcal{Q} = \text{Or}(\gamma)$  and all Poisson bi-vectors tried so far, the Poisson coboundary equation  $\mathcal{Q}(\mathcal{P}) = \llbracket \vec{\mathcal{X}}, \mathcal{P} \rrbracket$  did have vector field solutions  $\vec{\mathcal{X}}$  on the manifolds  $M$ .

*Remark 1.* Obtained from the graphs  $\gamma \in \ker d$ , the symmetries  $\mathcal{Q}(\mathcal{P}) = \text{Or}(\gamma)(\mathcal{P}) \in \ker \llbracket \mathcal{P}, \cdot \rrbracket$  are independent of a choice of local coordinates  $x^i$  (hence  $\xi_i$ ) on a chart if, the Kontsevich construction requires, the manifold  $M^r$  is endowed with an *affine* structure: all the coordinate transformations amount to  $\mathbf{x}' = A\mathbf{x} + \vec{\mathbf{b}}$  with a constant (over the intersection of charts) Jacobian matrix  $A$ . The parity-odd fibre variables are transformed using the inverse Jacobian matrix,  $\xi_i = A_i^j \xi'_j$ , making sense of the couplings  $\vec{\partial}/\partial \xi_i \cdot \vec{\partial}/\partial x^i$  which decorate the oriented edges of Kontsevich's graphs after the morphism  $\text{Or}$  works (see [3, 11]). The problem of Poisson cohomology class (non)triviality for the Kontsevich infinitesimal symmetries  $\dot{\mathcal{P}} = \mathcal{Q}(\mathcal{P}) \in \ker \llbracket \mathcal{P}, \cdot \rrbracket$  thus acquires two diametrically opposite interpretations:

**1** (as in [11]). The Poisson manifold  $M^{r < \infty}$  is equipped with *both* the smooth and affine structures.<sup>7</sup> By definition, two Poisson bi-vectors are equivalent,  $\mathcal{P}_1 \sim \mathcal{P}_2$ , if they are related by a diffeomorphism of the manifold  $M$ : using its smooth structure, the diffeomorphism identifies points in two copies of  $M$ , then relating the Poisson tensors by local coordinate reparametrizations near the respective points. The affine structure on  $M$  is now used to run the Kontsevich flows in two initial value problems  $\dot{\mathcal{P}}_i(\varepsilon) = \mathcal{Q}(\mathcal{P}_i(\varepsilon))$ ,  $\mathcal{P}_i(\varepsilon = 0) = \mathcal{P}_i$ . The Poisson triviality  $\mathcal{Q}(\mathcal{P}(\varepsilon)) = \llbracket \vec{\mathcal{X}}(\varepsilon), \mathcal{P}(\varepsilon) \rrbracket$  would relate either of bi-vectors  $\mathcal{P}_i(\varepsilon)$  back to the Cauchy datum  $\mathcal{P}_i$  by diffeomorphisms (as long as  $|\varepsilon|$  is small enough). Consequently, the Poisson bi-vectors  $\mathcal{P}_1(\varepsilon) \sim \mathcal{P}_2(\varepsilon)$  do not run out of the old equivalence class. In conclusion, the goal is to produce essentially new Poisson brackets by using a nontrivial cocycle  $\mathcal{Q}$ , two given structures on the manifold  $M^r$ , and its diffeomorphism. No examples of nontrivial action, so that  $\mathcal{P}_2(\varepsilon) \not\sim \mathcal{P}_i \not\sim \mathcal{P}_1(\varepsilon)$  at  $\varepsilon > 0$ , have ever been produced since 1996 (see [7, 11]).

**2** (as in [10]). The Poisson manifold  $M^{r < \infty}$  is equipped only with an affine structure. The countably many **grt**-related graph cocycles on  $n$  vertices and  $2n - 2$  edges in every term (the tetrahedron, the pentagon-wheel cocycle, etc., see [6, 15]) generate a noncommutative Lie algebra of infinitesimal symmetries  $\mathcal{Q}(\mathcal{P}) = \text{Or}(\gamma)(\mathcal{P})$  for a given Poisson structure  $\mathcal{P}$ . Consider the extreme case when *all* the cocycles  $\mathcal{Q}(\mathcal{P}) \in \ker \llbracket \mathcal{P}, \cdot \rrbracket$  are exact in the

<sup>5</sup>**Example.** So it is for the Kontsevich tetrahedral flow ([11] and [1]) evaluated at the Kirillov–Kostant linear Poisson brackets on the duals  $\mathfrak{g}^*$  of Lie algebras because in every term within the cocycle  $\mathcal{Q}(\mathcal{P})$  under study, at least one copy of  $\mathcal{P}$  is differentiated at least twice with respect to the global coordinates on  $\mathfrak{g}^*$ .

<sup>6</sup>**Example.** The Poisson bi-vectors  $\mathcal{P} = da_1 \wedge \dots \wedge da_m / \text{dvol}(\mathbb{R}^{m+2})$  of Nambu type with arbitrary Casimirs  $a_1, \dots, a_m \in C^\infty(\mathbb{R}^{m+2})$  and an arbitrary density in the volume element can have polynomial components  $\mathcal{P}^{ij} \in \mathbb{R}[x^1, \dots, x^{m+2}]$  of degrees as high as need be w.r.t. the global Cartesian coordinates  $x^\alpha$  on the vector space  $\mathbb{R}^{m+2}$ . The universal symmetries  $\dot{\mathcal{P}} = \text{Or}(\gamma)(\mathcal{P})$  obtained from Kontsevich's graph cocycles deform the symplectic foliation (which is given in  $\mathbb{R}^{m+2}$  by the intersections of the level sets for the Casimirs  $a_1, \dots, a_m$ ) in a regular way on an open dense subset of  $\mathbb{R}^{m+2}$ , so that the symmetries  $\dot{\mathcal{P}} = \mathcal{Q}(\mathcal{P})$  preserve this Nambu class of Poisson brackets: the flows force the evolution of the Casimirs and the volume density. Its integrability is an open problem; by Lemma 2 and [2], the evolutions induced by different graph cocycles do not commute.

<sup>7</sup>On the circle  $\mathbb{S}^1$ , the affine coordinate ‘angle’ is obvious whereas the smooth structure is used in the realm of Poincaré topology. A smooth atlas is always available for the spheres  $\mathbb{S}^r$ , but not for all  $r \in \mathbb{N}$  would the  $r$ -dimensional sphere admit an affine structure.

cohomology group  $H_{\mathcal{P}}^2(M)$  w.r.t. the Poisson differential  $\partial_{\mathcal{P}}$ . This assumption gives rise to the countable set of vector fields  $\vec{Y}(\gamma, \mathcal{P})$  on  $M$  such that  $\mathcal{Q}(\mathcal{P}) = \llbracket \vec{Y}, \mathcal{P} \rrbracket$ . (Some of these vector fields can be identically zero over  $M$ .) But if at least one such vector field is not constant w.r.t. the affine structure on  $M$ , then the shifts along its integral trajectories are nonlinear diffeomorphisms of  $M$ . The evolution of bi-vector  $\mathcal{P}$  is  $\dot{\mathcal{P}} = \mathcal{Q}(\mathcal{P}) = \llbracket \vec{Y}, \mathcal{P} \rrbracket$  or similarly,  $\dot{\Omega} = \llbracket \vec{Y}, \Omega \rrbracket$  for any multi-vector  $\Omega$  on  $M$  (see Proposition 1); this evolution is now seen as multivectors' response to the diffeomorphism whose construction refers only to the simple, affine local portrait of  $M$ . Summarizing, the store of flows  $\text{Or}(\gamma)(\mathcal{P})$  from the **grt**-related graph cocycles  $\gamma$  could be enough to approximate arbitrary smooth vector fields on  $M^r$ , that is, imitate its smooth structure. Whether this theoretical possibility is actually realised in relevant Poisson models is an open problem.

The Kontsevich symmetry construction is, therefore, either a generator of new Poisson brackets or the mechanism that provides diffeomorphisms of the underlying manifold.

**2. Homogeneous Poisson structures.** By definition, a bi-vector  $\mathcal{P}$  on a manifold  $M$  is called *homogeneous* (of scale  $\lambda$ ) with respect to a vector field  $\vec{V}$  on  $M$  if  $\llbracket \vec{V}, \mathcal{P} \rrbracket = \lambda \cdot \mathcal{P}$ .

**Example 1.** Let  $M = \mathbb{R}^r$  be a vector space (only linear reparametrizations  $\mathbf{x}' = A\mathbf{x}$  are allowed, so that the polynomial degrees of monomials in the ring  $\mathbb{R}[x^1, \dots, x^r]$  is well defined). Introduce the Euler vector field  $\vec{\mathcal{E}} = \sum_{i=1}^r x^i \partial/\partial x^i$ , and let all the components  $\mathcal{P}^{ij}$  of a bi-vector  $\mathcal{P}$  be homogeneous polynomials of degree  $d$  in the variables  $x^i$ . Then we have that  $\llbracket \vec{\mathcal{E}}, \mathcal{P} \rrbracket = (d-2) \cdot \mathcal{P}$ , which means that  $\mathcal{P}$  is homogeneous of scale  $d-2$  w.r.t. the Euler vector field  $\vec{\mathcal{E}}$ . In particular, if  $d \neq 2$  (i.e. if the coefficients of bi-vector  $\mathcal{P}$  are not quadratic), then we set  $\vec{V} = (d-2)^{-1} \cdot \vec{\mathcal{E}}$  and from the equality  $\mathcal{P} = \llbracket \vec{V}, \mathcal{P} \rrbracket$  we obtain that the same bi-vector  $\mathcal{P}$  has homogeneity scale  $\lambda = 1$  w.r.t. the multiple  $\vec{V}$  of the Euler vector field  $\vec{\mathcal{E}}$  on  $\mathbb{R}^r$ .

**Example 2.** Under the same assumptions, suppose further that  $\gamma = \sum_a c_a \gamma_a$  is a graph cocycle with  $n$  vertices and  $2n-2$  edges in every term  $\gamma_a$  (e.g., take the tetrahedron). Orient the ordered (by First  $\prec \dots \prec$  Last) edges in every  $\gamma_a$  using the edge decoration operators  $\vec{\Delta}_{ij} = \sum_{\mu=1}^r (\vec{\partial}/\partial \xi_{\mu}^{(i)} \otimes \vec{\partial}/\partial x_{(j)}^{\mu} + \vec{\partial}/\partial \xi_{\mu}^{(j)} \otimes \vec{\partial}/\partial x_{(i)}^{\mu})$ . By placing a copy of bi-vector  $\mathcal{P} = \frac{1}{2} \mathcal{P}^{kl}(\mathbf{x}) \xi_k \xi_l$  in each vertex  $v^{(i)}$  of  $\gamma_a$  and taking the sum (over the graph index  $a$ ) of products of the content of vertices in  $\gamma_a$  after all the edge operators  $\vec{\Delta}_{ij}$  work, we obtain<sup>8</sup> the bi-vector  $\mathcal{Q}(\mathcal{P}) := \text{Or}(\gamma)(\mathcal{P})$ . Then the coefficients of the bi-vector  $\mathcal{Q}(\mathcal{P})$  are homogeneous polynomials of degree  $n \cdot d - (2n-2)$  with respect to  $x^1, \dots, x^r$ , so that  $\llbracket \vec{\mathcal{E}}, \mathcal{Q}(\mathcal{P}) \rrbracket = n(d-2) \mathcal{Q}(\mathcal{P})$ . In particular, if  $d \neq 2$ , then  $\llbracket \vec{V}, \mathcal{Q}(\mathcal{P}) \rrbracket = n \cdot \mathcal{Q}(\mathcal{P})$ , whereas quadratic-coefficient bi-vectors  $\mathcal{P}$  (with  $d = 2$ ) are deformed within their subspace by the quadratic bi-vectors  $\mathcal{Q}(\mathcal{P})$  which are obtained from the Kontsevich graph cocycles.

**Lemma 3.** *If a Poisson bi-vector  $\mathcal{P} = \llbracket \vec{V}, \mathcal{Q}(\mathcal{P}) \rrbracket$  is homogeneous and  $\mathcal{Q}(\mathcal{P}) = \text{Or}(\gamma)(\mathcal{P}^{\otimes n})$  is built from a graph cocycle  $\gamma$  on  $n$  vertices, now containing a copy of  $\mathcal{P}$  in each vertex, then the bi-vector  $\mathcal{Q}(\mathcal{P})$  is also homogeneous:  $\llbracket \vec{V}, \mathcal{Q}(\mathcal{P}) \rrbracket = n \cdot \mathcal{Q}(\mathcal{P})$ , so that its scale is  $n$ .*<sup>9</sup>

*Remark 2* ([14, Rem. 4.9]). Consider a Nambu-type Poisson bi-vector  $\mathcal{P} = da/dxdydz$  on  $\mathbb{R}^3$  with Cartesian coordinates  $x, y, z$ ; here  $a \in \mathbb{R}[x, y, z]$  is a weight-homogeneous polynomial

<sup>8</sup>We refer to the original paper [11] and to [3] for illustrations and discussion how the graph orientation morphism works in practice.

<sup>9</sup>The proof amounts to the Leibniz rule: let us inspect how fast the bi-vector  $\mathcal{Q}(\mathcal{P})$ , which by construction is a homogeneous differential polynomial of degree  $n$  in  $\mathcal{P}$ , evolves along the vector field  $\vec{V}$ .

with an isolated singularity at the origin<sup>10</sup>, so that  $(w_{(x)} \cdot x\partial/\partial x + w_{(y)} \cdot y\partial/\partial y + w_{(z)} \cdot z\partial/\partial z)(a) = w_{(a)} \cdot a$ . Then a vector field  $\vec{V}$  with polynomial components satisfying the first-order PDE  $\mathcal{P} = \llbracket \vec{V}, \mathcal{P} \rrbracket$  exists if and only if<sup>11</sup> the weight degree  $w_{(a)}$  of the polynomial  $a$  is not equal to the sum  $w_{(x)} + w_{(y)} + w_{(z)}$  of weight degrees for the variables  $x, y, z$ .<sup>12</sup>

Summarizing, the homogeneity assumption about bi-vectors  $\mathcal{P}$  is restrictive; it is not always satisfied in Poisson models.

**Theorem 4.** *Let  $(M, \mathcal{P})$  be an affine finite-dimensional real Poisson manifold with  $\mathcal{P} = \llbracket \vec{V}, \mathcal{P} \rrbracket$  homogeneous. Let  $\gamma = \sum_a c_a \cdot \gamma_a$  be a graph cocycle consisting of unoriented graphs  $\gamma_a$  over  $n$  vertices and  $2n - 2$  edges (with a fixed ordering of edges in each  $\gamma_a$ ). Then the 1-vector  $\vec{X}(\gamma, \vec{V}, \mathcal{P}) = \text{Or}(\gamma)(\vec{V} \otimes \mathcal{P}^{\otimes n-1})$ , which is obtained by representing each edge  $i-j$  with the operator  $\vec{\Delta}_{ij}$  and by (graded-)symmetrizing over all the ways  $\sigma \in \mathbb{S}_n$  to send the  $n$ -tuple  $\vec{V} \otimes \mathcal{P}^{\otimes n-1}$  into the  $n$  vertices in each  $\gamma_a$ , is a Poisson cocycle:  $\vec{X} \in \ker \llbracket \mathcal{P}, \cdot \rrbracket$ .<sup>13</sup>*

*The vector field  $\vec{X}$  is defined up to adding arbitrary Poisson 1-cocycles  $\vec{Z} \in \ker \llbracket \mathcal{P}, \cdot \rrbracket$ .*

*Proof.* The expansion  $0 = \text{Or}(d\gamma)(\vec{V} \otimes \mathcal{P}^{\otimes n})$  for  $\gamma \in \ker d$  goes along the lines of [11] and [3, 7, 8], but the  $(n+1)$ -tuple of multivectors now contains one 1-vector and only  $n$  copies of the Poisson bi-vector  $\mathcal{P}$ . By assumption,  $d\gamma = \mathbf{0} \in \text{GRA}$ ; recall that  $\text{Or}(\mathbf{0})(\text{any multivectors}) = 0 \in \Gamma(\wedge^\bullet TM)$ . This zero l.h.s. equates  $0 = (\pi_S \vec{\sigma} \text{Or}(\gamma) - (-)^{(-1) \cdot (-N)} \text{Or}(\gamma) \vec{\sigma} \pi_S)(\vec{V} \otimes \mathcal{P}^{\otimes n})$ .<sup>14</sup>

The appointment of graded (multi)vectors into the vertices of  $d\gamma$  (hence, into the argument slots of the endomorphism  $\text{Or}(d\gamma)$ ) is achieved by the graded symmetrization using  $((n+1)!)^{-1} \text{Or}(d\gamma)(\pm \sigma(\vec{V} \otimes \mathcal{P}^{\otimes n}))$ . Fortunately, the field  $\vec{V}$  is the only parity-odd object, so its transpositions with the parity-even bi-vectors  $\mathcal{P}$  produce no sign factor: these  $\pm$  are all  $+$ . Likewise, the  $n!$  permutations of  $n$  indistinguishable copies of  $\mathcal{P}$  leave only  $n+1$  from  $(n+1)!$  in the denominator; to get rid of it, let us multiply by  $n+1$  both sides of the equality  $0 = \text{Or}(d\gamma)(\vec{V} \otimes \mathcal{P}^{\otimes n})$ . The symmetrization thus amounts, by the linearity of  $\text{Or}(\gamma)$ , to its evaluation at the sum of arguments,  $\vec{V} \cdot \mathcal{P}^n + \mathcal{P} \cdot \vec{V} \cdot \mathcal{P}^{n-1} + \dots + \mathcal{P}^n \cdot \vec{V}$ , in which the ordering of (multi)vectors now matches an arbitrary fixed enumeration of the vertices.

The rest of the proof is standard.<sup>15</sup> There remains  $0 = \text{Or}(\gamma)(\pi_S(\vec{V}, \mathcal{P}) \cdot \mathcal{P}^{n-1}) + \mathcal{P} \cdot$

<sup>10</sup>The Milnor number is the dimension  $\dim_{\mathbb{R}} \mathbb{R}[x, y, z]/(\partial a/\partial x, \partial a/\partial y, \partial a/\partial z)$  – here,  $< \infty$  by assumption.

<sup>11</sup>This means that not all Nambu-type Poisson bi-vectors  $\mathcal{P} = da/dxdydz$  are homogeneous w.r.t. a vector field  $\vec{V}$  with polynomial components; the PDE  $\mathcal{P} = \llbracket \vec{V}, \mathcal{P} \rrbracket$  with polynomial coefficients and unknown  $\vec{V}$  can in principle admit non-polynomial solutions.

<sup>12</sup>**Example.** If the weights of  $(x, y, z)$  are  $(1, 1, 1)$  and  $a = \frac{1}{3}(x^3 + y^3 + z^3)$  is cubic-homogeneous, then the components of Poisson bi-vector  $\mathcal{P}$  are quadratic and (by the above and also by [12, Exerc. 4.5.7c]) not of the form  $\mathcal{P} = \llbracket \vec{V}, \mathcal{P} \rrbracket$  for any polynomial-coefficient vector field  $\vec{V}$ . The non-existence of a solution  $\vec{V}$  with smooth non-polynomial coefficients is a separate problem.

<sup>13</sup>**Open problem** (for  $\mathcal{P}$  homogeneous and Poisson). Is the universal 1-vector field  $\vec{X}(\gamma, \vec{V}, \mathcal{P}) \in \ker \partial_{\mathcal{P}}$  Hamiltonian, i.e.  $\vec{X} = \llbracket \mathcal{P}, h \rrbracket$  for  $h \in C^\infty(M)$  or at least,  $\vec{X} = \mathcal{P} \lrcorner \eta$  for a maybe not exact 1-form  $\eta$  on  $M$ ?

<sup>14</sup>Here,  $\pi_S$  is the graded-symmetric Schouten bracket (so  $\pi_S(F, G) = (-)^{|F|-1} \llbracket F, G \rrbracket$ ), the graph insertion  $\vec{\sigma}$  into vertices is now the endomorphism insertion into argument slots,  $|\pi_S| = -1$ , and  $N = 2n - 2$  is the even number of edges in  $\gamma$ , hence minus the even number of  $\partial/\partial \xi_\mu$  in the edge operators  $\vec{\Delta}_{ij}$  making  $\text{Or}(\gamma)$ .

<sup>15</sup>We have  $0 = \text{Or}(\gamma)(\pi_S(\vec{V}, \mathcal{P}), \mathcal{P}^{n-1}) + \text{Or}(\gamma)(\pi_S(\mathcal{P}, \vec{V}), \mathcal{P}^{n-1}) + \text{Or}(\gamma)(\pi_S(\mathcal{P}, \mathcal{P}), \vec{V}, \mathcal{P}^{n-2}) + \dots + \text{Or}(\gamma)(\pi_S(\mathcal{P}, \mathcal{P}), \mathcal{P}^{n-2}, \vec{V}) + \text{Or}(\gamma)(\vec{V}, \pi_S(\mathcal{P}, \mathcal{P}), \mathcal{P}^{n-2}) + \text{Or}(\gamma)(\mathcal{P}, \pi_S(\vec{V}, \mathcal{P}), \mathcal{P}^{n-2}) + \text{Or}(\gamma)(\mathcal{P}, \pi_S(\mathcal{P}, \vec{V}), \mathcal{P}^{n-2}) + \text{Or}(\gamma)(\mathcal{P}, \pi_S(\mathcal{P}, \mathcal{P}), \vec{V}, \mathcal{P}^{n-3}) + \dots + \text{Or}(\gamma)(\mathcal{P}, \pi_S(\mathcal{P}, \mathcal{P}), \mathcal{P}^{n-3}, \vec{V}) + \dots$  (the Schouten bracket  $\pi_S$  passes along the slots towards the end)  $+ \text{Or}(\gamma)(\vec{V}, \mathcal{P}^{n-2}, \pi_S(\mathcal{P}, \mathcal{P})) + \dots + \text{Or}(\gamma)(\mathcal{P}^{n-2}, \vec{V}, \pi_S(\mathcal{P}, \mathcal{P})) + \text{Or}(\gamma)(\mathcal{P}^{n-1}, \pi_S(\vec{V}, \mathcal{P})) + \text{Or}(\gamma)(\mathcal{P}^{n-1}, \pi_S(\mathcal{P}, \vec{V})) - (-)^N \cdot [\pi_S(\text{Or}(\gamma)(\vec{V} \cdot \mathcal{P}^{n-1} + \mathcal{P} \cdot \vec{V} \cdot \mathcal{P}^{n-2} + \dots + \mathcal{P}^{n-1} \cdot \vec{V}), \mathcal{P}) + \pi_S(\text{Or}(\gamma)(\mathcal{P}^n), \vec{V}) + \pi_S(\vec{V}, \text{Or}(\gamma)(\mathcal{P}^n)) + \pi_S(\mathcal{P}, \text{Or}(\gamma)(\vec{V} \cdot \mathcal{P}^{n-1} + \mathcal{P} \cdot \vec{V} \cdot \mathcal{P}^{n-2} + \dots + \mathcal{P}^{n-1} \cdot \vec{V}))]$ . For  $\mathcal{P}$  Poisson,  $\pi_S(\mathcal{P}, \mathcal{P}) = 0$ , so we exclude all such terms ([4]). The remaining graded-symmetric Schouten brackets  $\pi_S$  contain a bi-vector as one of the arguments, hence those can be swapped at no sign factor; all doubles, so let us divide by 2.

$\pi_S(\vec{V}, \mathcal{P}) \cdot \mathcal{P}^{n-2} + \dots + \mathcal{P}^{n-1} \cdot \pi_S(\vec{V}, \mathcal{P})) - (-)^N [\pi_S(\text{Or}(\gamma)(\vec{V} \cdot \mathcal{P}^{n-1} + \mathcal{P} \cdot \vec{V} \cdot \mathcal{P}^{n-2} + \dots + \mathcal{P}^{n-1} \cdot \vec{V}), \mathcal{P}) + \pi_S(\text{Or}(\gamma)(\mathcal{P}^n), \vec{V})]$ . By the homogeneity assumption,  $\pi_S(\vec{V}, \mathcal{P}) = (-)^{1-1} \llbracket \vec{V}, \mathcal{P} \rrbracket = \mathcal{P}$ , and by construction,  $\text{Or}(\gamma)(\mathcal{P}^n) = \mathcal{Q}(\mathcal{P})$ , whence the minuend equals  $n \cdot \mathcal{Q}(\mathcal{P})$ . By Lemma 3, the graph flow is also homogeneous:  $\llbracket \vec{V}, \mathcal{Q}(\mathcal{P}) \rrbracket = \lambda \cdot \mathcal{Q}(\mathcal{P})$  with the vertex count  $\lambda = n$ . We obtain the equality

$$\begin{aligned} (-)^{2n-2} \cdot \llbracket \text{Or}(\gamma)(\vec{V} \cdot \mathcal{P}^{n-1} + \mathcal{P} \cdot \vec{V} \cdot \mathcal{P}^{n-2} + \dots + \mathcal{P}^{n-1} \cdot \vec{V}), \mathcal{P} \rrbracket &= \\ &= n \cdot \mathcal{Q}(\mathcal{P}) - (-)^{2n-2} \lambda \cdot \mathcal{Q}(\mathcal{P}) = (n - (-)^{\text{even}} n) \cdot \mathcal{Q}(\mathcal{P}) \equiv 0. \end{aligned}$$

We conclude that the 1-vector  $\vec{X} := \text{Or}(\gamma)(\vec{V} \otimes \mathcal{P}^{\otimes n-1})$  lies in  $\ker \llbracket \mathcal{P}, \cdot \rrbracket$ .<sup>16</sup> □

**Example 3.** Take the Lie algebra  $\mathfrak{gl}_2(\mathbb{R})$  with its four-dimensional vector space structure; denote by  $x, y, z, v$  the Cartesian coordinates. Consider the  $R$ -matrix  $\begin{pmatrix} x & y \\ z & v \end{pmatrix} \mapsto \begin{pmatrix} 0 & y \\ -z & 0 \end{pmatrix}$  known from [12]; the standard construction then yields the Poisson bi-vector in the algebra of coordinate functions,  $\mathcal{P} = (x^2y + y^2z) \partial_x \wedge \partial_y + (x^2z + yz^2) \partial_x \wedge \partial_z + (2xyz + 2yzv) \partial_x \wedge \partial_v + (y^2z + yv^2) \partial_y \wedge \partial_v + (yz^2 + zv^2) \partial_z \wedge \partial_v$ . This bracket has cubic-nonlinear homogeneous polynomial coefficients, hence  $d = 3$ . The vector field  $\vec{V} = (d - 2)^{-1} \cdot \vec{E}$  is the (multiple of the) Euler vector field on  $\mathbb{R}^4$ . As the graph cocycle  $\gamma$ , we take the tetrahedron (see [1, 11]); then the symmetry flow is  $\dot{\mathcal{P}} = \mathcal{Q}(\mathcal{P}) = (-48x^5y - 288x^3y^2z - 240xy^3z^2 + 192y^3z^2v - 384xy^2zv^2 - 192y^2zv^3) \partial_x \wedge \partial_y + (-48x^5z - 288x^3yz^2 - 240xy^2z^3 + 192y^2z^3v - 384xy^2zv^2 - 192yz^2v^3) \partial_x \wedge \partial_z + (-336x^4yz - 480x^2y^2z^2 - 576x^3yzv + 480y^2z^2v^2 + 576xyzv^3 + 336yzv^4) \partial_x \wedge \partial_v + (192x^3y^2z - 192xy^3z^2 + 288y^2zv^3 + 48yv^5 + 48(8x^2y^2z + 5y^3z^2)v) \partial_y \wedge \partial_v + (192x^3yz^2 - 192xy^2z^3 + 288yz^2v^3 + 48zv^5 + 48(8x^2yz^2 + 5y^2z^3)v) \partial_z \wedge \partial_v$ . We detect that this bi-vector is a coboundary,  $\mathcal{Q}(\mathcal{P}) = \llbracket \vec{Y}, \mathcal{P} \rrbracket$  with the vector  $\vec{Y} = (-24x^4 + 120y^2z^2 - 96yzv^2) \partial_x + (96x^3y - 96yv^3) \partial_y + (96x^3z - 96zv^3) \partial_z + (96x^2yz - 120y^2z^2 + 24v^4) \partial_v \pmod{\ker \llbracket \mathcal{P}, \cdot \rrbracket}$ . The vector field  $\vec{Y} \notin \ker \partial_{\mathcal{P}}$  cannot be Poisson-exact (clearly,  $\mathcal{Q}(\mathcal{P}) \neq 0$ ), hence  $\vec{Y}$  does not mark the Poisson cocycle of zero 1-vector.<sup>17</sup> But the universal vector field  $\vec{X}(\gamma, \vec{V}, \mathcal{P}) \in \ker \partial_{\mathcal{P}}$  is identically zero on  $\mathbb{R}^4$ . Indeed, the Euler field  $\vec{E} = \vec{V}$  is linear, yet it is readily seen from the figures in [1] that in every orgraph from the 1-vector  $\text{Or}(\gamma)(\vec{V} \otimes \mathcal{P}^{\otimes n-1})$ , the vertex with  $\vec{V}$  is differentiated at least twice (and at most thrice), so  $\vec{X} \equiv 0$ .

**Proposition 5.** *The flow  $\dot{\mathcal{P}} = \text{Or}(\text{tetrahedron } \gamma_3)(\mathcal{P})$  preserves the Nambu class of Poisson brackets,  $\{f, g\}_{\mathcal{P}} = \varrho(x, y, z) \cdot \det(\partial(a, f, g)/\partial(x, y, z))$  with arbitrary  $\varrho$  and global Casimir  $a$  on  $\mathbb{R}^3$ : the flow forces the nonlinear evolution  $\dot{a}, \dot{\varrho}$  with differential-polynomial r.h.s.*

• *This flow  $\dot{\mathcal{P}} = \mathcal{Q}(\mathcal{P})$  is not Poisson-exact in terms of any vector field  $\vec{Y}$  with differential-polynomial coefficients (cubic in both  $a$  and  $\varrho$ , of total differential order eight).*

<sup>16</sup>**Exercise.** Extend the proof to the case  $n = 1$ ,  $\gamma = \bullet$ ,  $d\gamma = -\bullet-\bullet$  (so that the l.h.s. was nonzero).

<sup>17</sup>Likewise, by using another  $R$ -matrix for  $\mathfrak{gl}_2(\mathbb{R})$ , namely  $\begin{pmatrix} x & y \\ z & v \end{pmatrix} \mapsto \begin{pmatrix} x & y \\ -z & v \end{pmatrix}$  also from [12], we obtain the Poisson bi-vector  $\mathcal{P} = 2x^2y \partial_x \wedge \partial_y + 2yz^2 \partial_x \wedge \partial_z + (2xyz + 2yzv) \partial_x \wedge \partial_v + (-2xyz + 2yzv) \partial_y \wedge \partial_z + 2yv^2 \partial_y \wedge \partial_v + 2yz^2 \partial_z \wedge \partial_v$  on  $\mathbb{R}^4$  with Cartesian coordinates  $x, y, z, v$ . The tetrahedral flow then equals  $\dot{\mathcal{P}} = \mathcal{Q}(\mathcal{P}) = (-384x^5y - 384x^3y^2z - 1536xy^2zv^2 + 384(x^2y^2z - 4y^3z^2)v) \partial_x \wedge \partial_y + (-384x^3yz^2 - 2688xy^2z^3 + 1152xyz^2v^2 + 384yz^2v^3 - 384(3x^2yz^2 - 7y^2z^3)v) \partial_x \wedge \partial_z + (-384x^4yz - 2688x^2y^2z^2 - 1536x^3yzv + 2688y^2z^2v^2 + 1536xyzv^3 + 384yzv^4) \partial_x \wedge \partial_v + (384x^4yz + 384x^2y^2z^2 + 1536y^3z^3 - 384xyzv^3 + 384yzv^4 + 384(x^2yz + y^2z^2)v^2 - 384(x^3yz - 2xy^2z^2)v) \partial_y \wedge \partial_z + (1536xy^3z^2 + 1536x^2y^2zv - 384xy^2zv^2 + 384y^2zv^3 + 384yv^5) \partial_y \wedge \partial_v + (-384x^3yz^2 - 2688xy^2z^3 + 1152xyz^2v^2 + 384yz^2v^3 - 384(3x^2yz^2 - 7y^2z^3)v) \partial_z \wedge \partial_v$ . It is Poisson-trivial:  $\mathcal{Q}(\mathcal{P}) = \llbracket \vec{Y}, \mathcal{P} \rrbracket$  with a representative  $\vec{Y} = (-96x^4 + 576y^2z^2 - 384yzv^2) \partial_x + (-192xy^2z + 192y^2zv - 384yv^3) \partial_y + (-96x^3z - 96xzv^2 + 96zv^3 + 96(x^2z - 4yz^2)v) \partial_z + (-576y^2z^2 - 384xyzv + 96v^4) \partial_v$ . These explicit examples of Poisson-exact bi-vector flows  $\dot{\mathcal{P}} = \mathcal{Q}(\mathcal{P}) = \llbracket \vec{Y}, \mathcal{P} \rrbracket$  will be useful in the future study of the mechanism  $\vec{Y} = \vec{Y}(\gamma, \vec{V}, \mathcal{P})$  of their observed  $\partial_{\mathcal{P}}$ -triviality.

The cocycle equation at hand,  $\mathcal{E}(\gamma_3, a, \varrho) = \{\dot{\mathcal{P}} = \llbracket \vec{\mathcal{Y}}, \mathcal{P} \rrbracket\}$ , is a first-order PDE with differential-polynomial coefficients (their skew-symmetry under permutations of  $x, y, z$  is inherited from the property of the Jacobian determinant and from the transformation law for the density  $\varrho$  in  $\mathcal{P}$ ). Whether this equation  $\mathcal{E}$  does not admit any non-polynomial solutions  $\vec{\mathcal{Y}}(a_{|\sigma|\leq 3}, \varrho_{|\tau|\leq 2})$  is an open problem.

**Acknowledgements.** A.V.K. thanks the organizers of international workshop SQS'19 (August 26–31, 2019 in Yerevan, Armenia) for a warm atmosphere during the event. A part of this research was done at the IHÉS (Bures-sur-Yvette, France); the authors are grateful to the RATP and stif for setting up stimulating working conditions. The authors thank G. H. E. Duchamp and M. Kontsevich for helpful discussions.

## References

- [1] Bouisaghouane A., Buring R., Kiselev A. The Kontsevich tetrahedral flow revisited // J. Geom. Phys. 2017. V. 119. P. 272–285.
- [2] Brown F. Mixed Tate motives over  $\mathbb{Z}$  // Annals of Math. 2012. V. 175. P. 949–976.
- [3] Buring R., Kiselev A. V. The orientation morphism: from graph cocycles to deformations of Poisson structures // J. Phys.: Conf. Ser. 2019. V. 1194. Paper 012017 P. 1–10; Kiselev A. V., Buring R. The Kontsevich graph orientation morphism revisited. 2019. Preprint arXiv:1904.13293 [math.CO]. P. 1–11.
- [4] Buring R., Kiselev A. V. On the Kontsevich  $\star$ -product associativity mechanism // PEPAN Letters. 2017. V. 14(2). P. 403–407; Buring R., Kiselev A. V. The expansion  $\star \bmod \bar{o}(\hbar^4)$  and computer-assisted proof schemes in the Kontsevich deformation quantization // Experimental Math. doi:10.1080/10586458.2019.1680463 (Preprint arXiv:1702.00681 [math.CO])
- [5] Buring R., Kiselev A. V., Rutten N. J. Poisson brackets symmetry from the pentagon-wheel cocycle in the graph complex // PEPAN Letters 2018. V. 49(5). P. 924–928.
- [6] Buring R., Kiselev A. V., Rutten N. J. The heptagon-wheel cocycle in the Kontsevich graph complex // J. Nonlin. Math. Phys. 2017. V. 24, Suppl. 1. P. 157–173.
- [7] Dolgushev V. A., Rogers C. L., Willwacher T. H. Kontsevich’s graph complex, GRT, and the deformation complex of the sheaf of polyvector fields // Annals of Math. 2015. V. 182(3). P. 855–943.
- [8] Jost C. Globalizing  $L$ -infinity automorphisms of the Schouten algebra of polyvector fields // Differ. Geom. Appl. 2013. V. 31(2). P. 239–247.
- [9] Kiselev A. V. The calculus of multivectors on noncommutative jet spaces // J. Geom. Phys. 2018. V. 130. P. 130–167.
- [10] Kiselev A. V. Open problems in the Kontsevich graph construction of Poisson bracket symmetries // J. Phys.: Conf. Ser. 2019. V. 1416. Paper 012018 P. 1–8.
- [11] Kontsevich M. Formality conjecture // Proc. of Conf. “Deformation theory and symplectic geometry”. Ascona, June 17–21, 1996. Dordrecht: Kluwer Acad. Publ., 1997. P. 139–156; Kontsevich M. Derived Grothendieck–Teichmüller group and graph complexes [after T. Willwacher] // Séminaire Bourbaki (69ème année, 2016–2017). Janvier 2017. No. 1126. P. 183–212.
- [12] Laurent–Gengoux C., Pichereau A., Vanhaecke P. Poisson structures. Grundlehren der mathematischen Wissenschaften 347. Springer-Verlag Berlin Heidelberg, 2013.
- [13] Lichnerowicz A. Les variétés de Poisson et leurs algèbres de Lie associées // J. Differential Geom. 1977. V. 12(2). P. 253–300.
- [14] Pichereau A. Poisson (co)homology and isolated singularities // J. Algebra 2006. V. 299(2). P. 747–777.
- [15] Willwacher T. M. Kontsevich’s graph complex and the Grothendieck–Teichmüller Lie algebra // Invent. Math. 2015. V. 200(3). P. 671–760.





## Chapter 20

# The hidden symmetry of Kontsevich's graph flows on the spaces of Nambu-determinant Poisson brackets

This chapter is based on the preprint *R. Buring, D. Lipper and A. V. Kiselev. arXiv:2112.03897 [math.SG] – 23+iv p. (submitted).*

*Commentary.* In reference to Part I of the dissertation, the material of this chapter is used in Chapter 7. The final chapter concludes with a list of open problems.

# THE HIDDEN SYMMETRY OF KONTSEVICH'S GRAPH FLOWS ON THE SPACES OF NAMBU-DETERMINANT POISSON BRACKETS

R. BURING<sup>\*,a</sup>, D. LIPPER<sup>b</sup>, AND A. V. KISELEV<sup>\*,§,b</sup>

*This text involves graph theory, Poisson geometry, and combinatorics;  
it concludes with 7 research problems about Kontsevich's universal symmetries  
of the spaces of Nambu-determinant Poisson brackets on  $\mathbb{R}^3$  and  $\mathbb{R}^4$ .*

ABSTRACT. Kontsevich's graph flows are – universally for all finite-dimensional affine Poisson manifolds – infinitesimal symmetries of the spaces of Poisson brackets. We show that the previously known tetrahedral flow and the recently obtained pentagon-wheel flow preserve the class of Nambu-determinant Poisson bi-vectors  $P = d\mathbf{a}/d\text{vol}(\mathbf{x}) = \varrho(\mathbf{x}) \cdot d\mathbf{a}/d\mathbf{x}$  on  $\mathbb{R}^d \ni \mathbf{x}$  for  $d = 3$  and  $4$ , including the general case  $\varrho \neq 1$ . We establish that the Poisson bracket evolution  $\dot{P} = Q_\gamma(P \otimes \#^{\text{Vert}(\gamma)})$  is trivial in the respective Poisson cohomology,  $Q_\gamma = \llbracket P, \vec{X}([\varrho], [\mathbf{a}]] \rrbracket$ , for the Nambu-determinant bi-vectors  $P(\varrho, [\mathbf{a}])$ . For the global Casimirs  $\mathbf{a} = (a_1, \dots, a_{d-2})$  and inverse density  $\varrho$  on  $\mathbb{R}^d$ , we analyse the combinatorics of their evolution induced by the Kontsevich graph flows, namely  $\dot{\varrho} = \dot{\varrho}([\varrho], [\mathbf{a}])$  and  $\dot{\mathbf{a}} = \dot{\mathbf{a}}([\varrho], [\mathbf{a}])$  with differential-polynomial right-hand sides. Besides the anticipated collapse of these formulas by using the Civita symbols (three for the tetrahedron  $\gamma_3$  and five for the pentagon-wheel graph cocycle  $\gamma_5$ ), as dictated by the behaviour  $\varrho(\mathbf{x}') = \varrho(\mathbf{x}) \cdot \det\|\partial\mathbf{x}'/\partial\mathbf{x}\|$  of the inverse density  $\varrho$  under reparametrizations  $\mathbf{x} \rightleftharpoons \mathbf{x}'$ , we discover another, so far hidden discrete symmetry in the construction of these evolution equations.

**Introduction.** Kontsevich's infinitesimal symmetries  $P \mapsto P + \varepsilon Q(P) + \bar{o}(\varepsilon)$  of the spaces of Poisson structures are universal for all finite-dimensional affine Poisson manifolds  $(M_{\text{aff}}^d, P)$ , preserving the property of the Cauchy datum  $P(\varepsilon = 0)$  to remain Poisson modulo  $\bar{o}(\varepsilon)$  at  $\varepsilon > 0$ . The goal of this paper is to explore the combinatorics that arises for the restriction of these symmetry flows  $\dot{P} = Q(P)$  to the class of generalized Nambu-determinant Poisson brackets,

$$\{f, g\} = \varrho(\mathbf{x}) \cdot \det\|\partial(a_1, \dots, a_{d-2}, f, g)/\partial(x^1, \dots, x^d)\|,$$

---

*Date:* 7 December 2021.

*2010 Mathematics Subject Classification.* 05C22, 68R10, also 53D17.

*Key words and phrases.* Poisson geometry, Nambu-determinant Poisson bracket, Poisson cohomology, symmetry, Kontsevich's graph complex.

\* A part of this research was done while RB and AVK were visiting at the IHÉS in December 2019.

§ Corresponding author. *E-mail:* A.V.Kiselev@rug.nl.

<sup>a</sup>*Address:* Institut für Mathematik, Johannes Gutenberg-Universität, Staudingerweg 9, D-55128 Mainz, Germany.

<sup>b</sup>*Address:* Bernoulli Institute for Mathematics, Computer Science and Artificial Intelligence, University of Groningen, P.O. Box 407, 9700 AK Groningen, The Netherlands.

with  $d - 2$  global Casimirs  $\mathbf{a} = (a_1, \dots, a_{d-2})$  and inverse density  $\varrho$  on  $\mathbb{R}^d$ . Differential-polynomial in the bi-vector components,

$$P^{ij} = \{x^i, x^j\} = \varrho(\mathbf{x}) \cdot \sum_{i_1, \dots, i_{d-2}} \varepsilon^{i_1 \dots i_{d-2} ij} \cdot \frac{\partial a_1}{\partial x^{i_1}} \dots \frac{\partial a_{d-2}}{\partial x^{i_{d-2}}},$$

the right-hand sides  $Q([P])$  of the flows are encoded by using the graph cocycles in the Kontsevich undirected graph complex. The tetrahedral cocycle flow  $\dot{P} = \text{Or}(\gamma_3)(P^{\otimes 4})$  is the first example from the pioneering paper [14] (cf. [19] and [2]); graph cocycles beyond the tetrahedron  $\gamma_3$  are discussed in [7] (see references therein); the next, higher nonlinearity degree flows are constructed for the pentagon-wheel cocycle  $\gamma_5$  in [9] and for the heptagon-wheel cocycle  $\gamma_7$  in [8]. We now study the restriction of this universal construction to a particular class of Poisson brackets, so that their analytic properties repercuss in the combinatorics of algebraic structures and in the Poisson-cohomological (non)triviality of the infinitesimal deformations  $P \mapsto P + \varepsilon Q(P) + \bar{o}(\varepsilon)$  with the markers  $Q \in \ker[[P, \cdot]]$  of second Poisson cohomology classes  $[Q] \in H_P^2(M_{\text{aff}}^d = \mathbb{R}^d)$ .

Linear in the functional parameters  $\varrho$  and  $\mathbf{a}$ , the Nambu-determinant Poisson bi-vectors  $P = \varrho(\mathbf{x}) \cdot d\mathbf{a}/d\mathbf{x}$  are both special and generic in many situations within Poisson geometry. Among the most well known examples of Poisson structures from this class we recall, for instance,

- the Euler top bracket  $\{x^i, x^j\} = \varepsilon^{ijk} \cdot x^k$  on  $\mathbb{E}^3 \simeq so(3)^*$ , that is  $\{x, y\} = z$  and so on w.r.t. the signed permutations  $\sigma \in \mathbb{S}_3$ . This bracket is Nambu-class with  $\varrho \equiv 1$  and the global polynomial Casimir  $a(x, y, z) = \frac{1}{2}(x^2 + y^2 + z^2)$ .
- the log-symplectic bracket  $\{x, y\} = xy$  (and so on, cyclically), given on  $\mathbb{R}^3$  with  $\varrho \equiv 1$  by the Casimir  $a = xyz$ . This bracket is important in deformation quantization (on  $\mathbb{R}^2 \subset \mathbb{R}^3$ ) since it is explicitly known that  $x \star y = \exp(\hbar) \cdot y \star x$  for the associative noncommutative star-product with this Poisson bracket,  $\{x, y\} = xy$ , in the leading deformation term (see [15, 16, 18] and [1]).

The generalized ( $\varrho \neq 1$ ) Nambu-determinant Poisson brackets  $P = \varrho(\mathbf{x}) \cdot d\mathbf{a}/d\mathbf{x}$  are “generic” in the sense that as soon as a differential-polynomial identity which holds by force of the Jacobi identity  $\mathcal{E} = \{\frac{1}{2}[[P, P]] = 0\}$  and its differential consequences (as well as by force of other constraints: e.g., the cyclic weight relations and multiplicativity of the Kontsevich graph weights in the  $\star$ -products, cf. [17]), for instance,

$$(f \star g) \star h - f \star (g \star h) \doteq 0 \quad \text{on } \mathcal{E}^\infty,$$

is achieved by splitting and solving an overdetermined system of homogeneous differential-polynomial equations in  $\varrho$  and  $\mathbf{a}$ , then that identity in practice holds at once for *generic* Poisson bi-vectors  $P$ , now by force of a nontrivial, explicitly constructed operator  $\diamond$  in the identity’s right-hand side (see [4, Theor. 9 and 12]): respectively,

$$(f \star g) \star h - f \star (g \star h) = \diamond(P, [[P, P]]).$$

On the other hand, for any choice of  $\varrho \neq 1$ , the Nambu-determinant Poisson brackets  $P = \varrho(\mathbf{x}) \cdot d\mathbf{a}/d\mathbf{x}$  are “special” in the sense that they admit the maximal set of  $d - 2$  scalar Casimirs  $\mathbf{a} = (a_1, \dots, a_{d-2})$ . The space  $\mathbb{R}^d$  is foliated by the intersections of the level sets  $\{a_i = \text{const}\}$  into symplectic leaves, which are generally two-dimensional: e.g., consider the concentric spheres  $\{\mathbf{x} \mid x^2 + y^2 + z^2 = \text{const} > 0\}$  for the Euler top.

But not every Poisson bracket on  $\mathbb{R}^3$  admits a global polynomial Casimir  $a \neq \text{const}$  if the coefficients  $P^{ij}$  of the bi-vector  $P$  are polynomial. (Whereas for the Nambu class this is achieved tautologically by taking  $\varrho, a_i \in \mathbb{R}[x^1, \dots, x^d]$  in any fixed system of affine coordinates on  $\mathbb{R}^d$  in any dimension  $d \geq 3$ .)

**Counterexample 1** ([20]). On  $\mathbb{R}^d$  with Cartesian coordinates  $\mathbf{x} = (x^1, \dots, x^d)$ , consider the Euler vector field  $\vec{E} = \sum_i x^i \cdot \partial/\partial x^i$  and, for any  $k \geq 2$ , take another homogeneous vector field  $\vec{V} = \sum_j (x^j)^k \cdot \partial/\partial x^j$ . By definition, put  $P := \vec{V} \wedge \vec{E}$ . Then the bi-vector  $P$  is Poisson — yet it does not admit any non-constant global polynomial Casimir  $a$  on  $\mathbb{R}^d$ . (A proof is recalled in Appendix A, see p. 465 below.)

In the same context of competing “generic vs special”, Kontsevich’s graph flows provide (markers of the) second Poisson cohomology classes  $Q([P]) \in \ker[[P, \cdot]]$  in an extremely broad setting: indeed, universally for all finite-dimensional affine Poisson manifolds  $(M_{\text{aff}}^d, P)$ . This automatically poses the problem of (non)triviality for these Poisson cohomology classes  $[Q] \in H_P^2(M_{\text{aff}}^d)$ . We recall from [2, 5] that for nontrivial graph cocycles  $\gamma$  in the Kontsevich undirected graph complex, there does not exist any mechanism that would trivialize the flows  $\dot{P} = \text{Or}(\gamma)(P^{\otimes \#\text{Vert}(\gamma)})$  at the level of Kontsevich’s graphs, that is, by using a would-be universal trivializing vector field  $\vec{X}$  again determined within the graph language, and hence by a formula that would work uniformly in *all* dimensions. For instance, such is manifestly the case for the tetrahedron  $\gamma_3$ , for the pentagon-wheel cocycle  $\gamma_5$ , etc., provided the dimension  $d$  of Poisson manifold is greater than two. In other words, the coboundary equation,

$$\text{Or}(\gamma)(P^{\otimes \#\text{Vert}(\gamma)}) - [[P, \vec{X}(\gamma)]] = \diamond(P, [[P, P]]),$$

has no solution  $(\gamma', \diamond)$  for the main sequence of nontrivial graph cocycles  $\gamma_3, \gamma_5, \gamma_7, \dots$  and their iterated commutators (if  $d \geq 3$  is not fixed *a priori*; if  $d = 2$ , the graph  $\gamma'$  trivializing the tetrahedral  $\gamma_3$ -flow is found in [2]). The present work serves to continue — from [2, 5, 6] — the line of study on the Poisson (non)triviality of Kontsevich’s graph flows.

The fact we discover is that, for rich classes of Poisson structures, the Kontsevich graph flows *are* Poisson-trivial, so that the resulting shifts  $Q([P]) = \text{Or}(\gamma)(P^{\otimes n})$  of Poisson bi-vectors  $P$  are induced by highly nonlinear, non-affine reparametrizations of the base coordinates — along the integral trajectories of the trivializing vector fields  $\vec{X}$  — on the *affine* Poisson manifolds  $M_{\text{aff}}^d$ . Such is the case for the Nambu-determinant class of brackets  $P(\varrho, [a])$  on  $\mathbb{R}^3$  and the graph flows preserving it. We establish the fact of trivialization and we collapse the formula of the vector field  $\vec{X}([\varrho], [a])$  by using the features of the Nambu–Poisson geometry under study. (All these analytic and combinatorial results are verified by direct calculation.) It remains to express the vector field  $\vec{X}$  through deeper invariants, that is, explain the work of the trivialization and collapse mechanisms at the level of graphs and symplectic foliation.

*Remark 1.* For a chosen volume element  $\text{dvol}(\mathbf{x}) = \text{d}\mathbf{x}/\varrho(\mathbf{x})$  with smooth  $\varrho$ , needed for construction of the Nambu-determinant bi-vectors  $P = \text{d}\mathbf{a}/\text{dvol}(\mathbf{x})$ , the zero locus of the inverse density  $\varrho$  provides a tiling of the affine space  $\mathbb{R}^d$ . Inside each cell bounded by the walls  $\{\mathbf{x} \mid \varrho(\mathbf{x}) = 0\}$ , that is on every maximal subset where the restriction of  $\varrho$  is nowhere vanishing, the inverse density can be brought to a constant  $\varrho'(\mathbf{x}') \equiv \pm 1$

by a (non)linear, pointwise-dependent rescaling of local coordinates. The restriction of the graph flows to the subclass of ‘genuine’ Nambu-determinant brackets  $P = d\mathbf{a}/d\mathbf{x}$  can either degenerate (e.g., for the tetrahedral flow over  $\mathbb{R}^3$ ) or stay nonzero (e.g., for the tetrahedral flow over  $\mathbb{R}^4$ ), see below. In all these cases, the trivializing vector fields  $\vec{X}$  behave in a usual way, as tensors do, in the course of such transformations to the normal coordinates; note that the vector fields  $\vec{X}$  can acquire arbitrary Poisson-exact components  $\llbracket P, H \rrbracket$ . Yet the construction of the normal coordinates satisfying  $\varrho'(\mathbf{x}') = \pm 1$  is *a priori* not correlated at all with the affine structure — which the graph flows refer to.

---

This paper is structured as follows. In §1 we recall some facts about the Nambu-determinant Poisson brackets  $P(\varrho, [\mathbf{a}])$  on  $\mathbb{R}^d$  and about the Kontsevich graph flows over affine Poisson manifolds  $(M_{\text{aff}}^d, P)$ . Next, in §2 we detect that the Nambu class of Poisson brackets on  $\mathbb{R}^3$  and  $\mathbb{R}^4$  is preserved by the graph flows for the tetrahedral cocycle  $\gamma_3$  and by the pentagon-wheel cocycle  $\gamma_5$  over  $\mathbb{R}^3$ . The structure of induced evolution  $\dot{\varrho}([\varrho], [\mathbf{a}]), \dot{a}([\varrho], [\mathbf{a}])$  is then put, in §3, in correspondence with the original graph cocycle, and the formulas of induced velocities are collapsed by using the Civita symbols (one per graph vertex minus one overall: e.g., three symbols for the tetrahedron); the affine structure of  $\mathbb{R}^d$  is crucial at that point. In §4 we analyze the algebra and combinatorics of the marker-monomials under the sums with multiple Civita symbols. Here we discover an extra symmetry of the Kontsevich graph flows’ restriction to the spaces of Nambu-determinant Poisson structures. Finally, we establish in §5 that the tetrahedral flow over  $\mathbb{R}^3$  is Poisson-cohomology trivial, and we collapse the formula of the trivializing vector field  $\vec{X}$  by using the same mechanism of Civita symbols as before. The paper concludes with a list of open problems about the graph flows and combinatorics of their restrictions to the Nambu class of brackets.

## 1. PRELIMINARIES

**1.1. The generalized Nambu-determinant Poisson brackets.** In the context of quark dynamics and  $n$ -ary interactions, Nambu introduced ([22], cf. [10, 11]) a class of Poisson brackets with global Casimirs  $\mathbf{a} = (a_1, \dots, a_{d-2})$  on  $\mathbb{R}^d \ni \mathbf{x}$ : the Poisson bi-vectors are  $P = d\mathbf{a}/d\text{vol}(\mathbf{x}) = \varrho(\mathbf{x}) \cdot da_1 \wedge \dots \wedge da_{d-2}/dx^1 \wedge \dots \wedge dx^d$  with a not necessarily constant inverse of the volume density,  $\varrho(\mathbf{x})$ . The coordinate expressions are, for example,

$$\{f, g\} = \varrho(\mathbf{x}) \cdot \left| \frac{\partial(a, f, g)}{\partial(x, y, z)} \right| = \varrho(x, y, z) \cdot \begin{vmatrix} a_x & f_x & g_x \\ a_y & f_y & g_y \\ a_z & f_z & g_z \end{vmatrix}$$

on  $\mathbb{R}^3 \ni \mathbf{x} = (x, y, z)$ , and likewise,

$$\{f, g\} = \varrho(x, y, z, w) \cdot \left| \frac{\partial(a_1, a_2, f, g)}{\partial(x, y, z, w)} \right|$$

on  $\mathbb{R}^4$  with global (e.g., Cartesian) coordinates  $\mathbf{x} = (x, y, z, w)$ . It is obvious that the given functions  $a_i$  which show up in the construction of the bi-vector  $P$  Poisson-commute with any argument  $f \in C^\infty(\mathbb{R}^d)$ . The scalars  $a_i(\mathbf{x}) = a_i(\mathbf{x}'(\mathbf{x}))$  do not change

under the coordinate transformations  $\mathbf{x}(\mathbf{x}') \rightleftharpoons \mathbf{x}'(\mathbf{x})$ . Given two scalars  $f, g \in C^\infty(\mathbb{R}^d)$ , their Poisson bracket is also a scalar. To counterbalance the behaviour of the Jacobian determinant in the course of coordinate transformations,

$$\left| \frac{\partial(a, f, g)}{\partial(x, y, z)} \right| = \left| \frac{\partial(a, f, g)}{\partial(x', y', z')} \right| \cdot \left| \frac{\partial(x', y', z')}{\partial(x, y, z)} \right|,$$

the object  $\varrho(\mathbf{x}) \rightleftharpoons \varrho'(\mathbf{x}')$  behaves accordingly,

$$\varrho(x, y, z) \cdot \left| \frac{\partial(x', y', z')}{\partial(x, y, z)} \right| = \varrho'(x', y', z'),$$

with an elementary general fact that  $d\mathbf{x}/\varrho(\mathbf{x}) = d\mathbf{x}'/\varrho'(\mathbf{x}')$  and

$$\varrho(\mathbf{x}) \cdot |\partial(\mathbf{x}')/\partial\mathbf{x}| = \varrho'(\mathbf{x}') \tag{1}$$

for all dimensions  $d \geq 3$ . So, let us keep in mind that the inverse density  $\varrho(\mathbf{x}) = \varrho'(\mathbf{x}') \cdot |\partial\mathbf{x}/\partial\mathbf{x}'|$  in the volume element  $d\text{vol}(\mathbf{x}) = d\mathbf{x}/\varrho(\mathbf{x})$  is nontrivially reparametrized under the changes  $\mathbf{x}(\mathbf{x}') \rightleftharpoons \mathbf{x}'(\mathbf{x})$ , whereas the scalars  $a_i$  are not transformed. Let us remember also that so far, the coordinate changes could be arbitrarily nonlinear, that is, not necessarily linear or affine on  $\mathbb{R}^d$ .

**1.2. Kontsevich’s graph flows.** In the seminal paper [14] (see also [19] and [2, 9, 5, 12, 13] for illustrations and discussion), Kontsevich designed a method to construct infinitesimal symmetries  $P = Q([P])$  of the spaces of Poisson structures on affine finite-dimensional Poisson manifolds  $(M_{\text{aff}}^d, P)$ . The construction is universal for all such geometries (with  $\mathbf{x}' = A\mathbf{x} + \mathbf{b}$  as the only admissible coordinate reparametrizations). The right-hand side  $Q$  of the evolution  $\dot{P} = Q([P])$ , differential-polynomial in the components of the bi-vector  $P$ , is described by using linear combinations (with real coefficients) of directed graphs; these graphs are built of wedges  $\leftarrow \bullet \rightarrow$  with prescribed ordering Left  $\prec$  Right of the outgoing arrows in every internal vertex. Each edge is decorated with its own summation index which runs from 1 to the dimension  $d = \dim M^d$ ; each decorated edge  $\xrightarrow{i}$  corresponds to the derivative  $\partial/\partial x^i$  w.r.t. a local coordinate in an affine chart of  $M^d$ ; each internal vertex of the directed graph is inhabited by a copy of the Poisson bi-vector  $P = (P^{ij}(\mathbf{x}))$ ; each graph determines a differential-polynomial expression (w.r.t. the structure  $P$  and the content of sink vertices) in a natural way: take the product of the (differentiated) contents of the vertices and sum over all the indices. Two factors, namely (i) the contraction of lower indices – from  $\partial/\partial x^i$  and  $\partial/\partial x^j$  on the respective Left and Right outgoing edges – with the first and second indices  $i, j$  in the skew-symmetric bi-vector components  $P^{ij}(\mathbf{x})$  in the arrowtail vertex, and (ii) the independence of the Jacobians  $A$  (in the affine changes  $\mathbf{x}' = A\mathbf{x} + \mathbf{b}$ ) from a point of two charts’ overlap, make the Kontsevich construction well defined for an arbitrary choice of local affine coordinates on  $(M_{\text{aff}}^d, P)$ .

The graph cocycles  $\gamma$  on  $n$  vertices and  $2n - 2$  edges in the Kontsevich undirected graph complex (see [14] as well as [5, 13] and references therein), when directed (inheriting the edge ordering from  $\gamma$ ) and evaluated at  $n$  copies of a given Poisson bi-vector, yield a natural class of Kontsevich’s graph flows  $\dot{P} = \text{O}\check{r}(\gamma)(P^{\otimes n})$  on the spaces of Poisson structures. Willwacher’s construction of suitable graph cocycles  $\gamma$  from the Grothendieck–Teichmüller Lie algebra  $\mathfrak{grt}$  gives us the main sequence to work with:

Kontsevich's tetrahedron  $\gamma_3$  (which is the wheel graph with three spikes), the Kontsevich–Willwacher pentagon-wheel cocycle  $\gamma_5$  (see [8, 7] and [9]), the heptagon-wheel cocycle  $\gamma_7$  (see [7] and [5]), etc., and their iterated commutators (always on  $n$  vertices and  $2n - 2$  edges, for instance with 9 vertices and 16 edges in  $[\gamma_3, \gamma_5]$ ). The construction of Lie brackets on the vector space of graphs with wedge ordering of edges is explained in [14] and [23, 7].

**Example 2** ([14, 19] and [2, 13]). The tetrahedron  $\gamma_3$ , when oriented by the morphism  $\text{Or}$  to the balanced (by  $8 : 24 = 1 : 3$ ) sum  $\Gamma_1(1, 2) + 3(\Gamma'_2(1, 2) - \Gamma''_2(1, 2))$  of two skew-symmetrized bi-vector graphs built of wedges (see Fig. 1) now encodes the

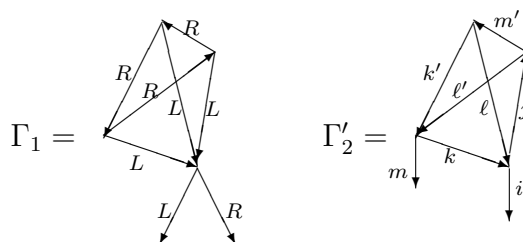


FIGURE 1. The components of Kontsevich's tetrahedral flow  $\dot{P}(1, 2) = \Gamma_1(1, 2) + 3(\Gamma'_2(1, 2) - \Gamma''_2(1, 2))$  on the on the space of Poisson bi-vectors  $P$  on  $\mathbb{R}^d$  in any dimension  $d \geq 3$ .

differential-polynomial velocity of Poisson bi-vectors:

$$Q_{\text{tetra}}(P) = 1 \cdot \left( \frac{\partial^3 P^{ij}}{\partial x^k \partial x^\ell \partial x^m} \frac{\partial P^{kk'}}{\partial x^{\ell'}} \frac{\partial P^{\ell\ell'}}{\partial x^{m'}} \frac{\partial P^{mm'}}{\partial x^{k'}} \right) \frac{\partial}{\partial x^i} \wedge \frac{\partial}{\partial x^j} + 3 \cdot \left( \frac{\partial^2 P^{ij}}{\partial x^k \partial x^\ell} \frac{\partial^2 P^{km}}{\partial x^{k'} \partial x^{\ell'}} \frac{\partial P^{k'\ell}}{\partial x^{m'}} \frac{\partial P^{m'\ell'}}{\partial x^j} \right) \frac{\partial}{\partial x^i} \wedge \frac{\partial}{\partial x^m}.$$

Indeed, we place copies of a given bi-vector  $P$  into the internal vertices, match their first and second indices with the summation indices that decorate the arrows (note that the ordering, available in the digraph encoding in *loc. cit.*, is not everywhere displayed in Fig. 1, but it is easily retrieved from the differential-polynomial formula), and for all values of all the indices, we take the sum of products of all the differentiated contents of the vertices. It is clear that for an arbitrary affine Poisson manifold, the flow  $\dot{P} = \text{Or}(\gamma_3)(P^{\otimes 4})$  is coordinate-free.

Other examples of nonlinear proper ( $\neq 0$  if  $P$  is Poisson) Kontsevich's graph flows are constructed in [9] for the pentagon-wheel cocycle  $\gamma_5$  and in [5] for the heptagon-wheel cocycle  $\gamma_7$  (see also [13]).

## 2. STRUCTURAL STABILITY OF THE NAMBU-DETERMINANT BRACKETS UNDER KONTSEVICH'S FLOWS

The first main question is which we explore in this note is how, in precisely which way the class  $P = \mathbf{d}\mathbf{a}/\text{dvol}(\mathbf{x})$  of generalized ( $\varrho \neq 1$ ) Nambu-determinant bi-vectors  $P(\varrho, \mathbf{a})$  on  $\mathbb{R}^d$  is stable under Kontsevich's universal deformations  $\dot{P} = Q_\gamma([P])$  given by the graph cocycles  $\gamma$ . The other question for us to explore is the combinatorial

mechanism of this stability. So, let us first inspect how the infinitesimal symmetries  $\dot{P} = \text{Or}(\gamma)(P^{\otimes \#V(\gamma)})$  of the – actually, unknown – space of *all* Poisson brackets  $P$  on  $\mathbb{R}^d$  (where  $\mathbb{R}^d$  is viewed as an affine manifold) restrict to the subspace of Nambu-determinant Poisson brackets.

Because the Nambu-determinant bi-vectors  $P(\varrho, [\mathbf{a}]) = \varrho(\mathbf{x}) \cdot d\mathbf{a}/d\mathbf{x}$  are *linear* in both the inverse density  $\varrho$  and Casimirs  $\mathbf{a} = (a_1, \dots, a_{d-2})$ , the class  $\{P(\varrho, [\mathbf{a}])\}$  is stable if, by definition, there exist the velocities  $\dot{\varrho}$  and  $\dot{\mathbf{a}}$  (depending on the point  $(\varrho, \mathbf{a})$  in the functional parameter space) such that

$$\dot{P}(\varrho, [\mathbf{a}]) = P(\dot{\varrho}, [\mathbf{a}]) + \sum_{i=1}^{d-2} P(\varrho, [a_1, \dots, [\dot{a}_i], \dots, [a_{d-2}]]). \tag{2}$$

In particular, the stability of the class is achieved if the evolution  $\dot{\varrho}$  and  $\dot{\mathbf{a}}$  is differential-polynomial (of finite degrees and differential orders) in the parameters that evolve,

$$\dot{\varrho} = \dot{\varrho}([\varrho], [\mathbf{a}]), \quad \dot{\mathbf{a}} = \dot{\mathbf{a}}([\varrho], [\mathbf{a}]). \tag{3}$$

The construction of the Kontsevich flow  $\dot{P} = \text{Or}(\gamma)(P^{\otimes n})$  from a graph cocycle  $\gamma$  on  $n$  vertices and the count of homogeneities always allow us to estimate both the order and polynomial degrees of such (non)linear PDE evolution — provided it exists.

**Example 3** ( $\gamma_3$ -flow over  $\mathbb{R}^3$ ). First, if  $\varrho \equiv 1$  and the Poisson bi-vector is  $P = da(x, y, z)/dx dy dz$ , then the Kontsevich tetrahedral flow  $\dot{P} = \text{Or}(\gamma_3)(P^{\otimes 4}[a])$  vanishes identically. In retrospect, this is true because every term in  $\dot{\mathbf{a}}$  contains a derivative of  $\varrho$ , and all the more each term in  $\dot{\varrho}$  does so, whence the Cauchy datum  $\varrho = \text{const}$  makes the flow well defined but identically zero.

Let the inverse density  $\varrho(x, y, z)$  be not necessarily constant over  $\mathbb{R}^3$ . A simple *a priori* estimate of homogeneities suggests that the terms in the differential-polynomial right-hand side of  $\dot{\varrho}$  and  $\dot{\mathbf{a}}$  are constrained by the ansatz

$$\begin{aligned} \dot{\mathbf{a}} &\sim a^4 \varrho^3, \text{ with 9 derivatives in each monomial,} \\ &\quad \text{at most 3rd order derivatives of } a \text{ and of } \varrho; \\ \dot{\varrho} &\sim a^3 \varrho^4, \text{ with 9 derivatives in each monomial,} \\ &\quad \text{at most 3rd order derivatives of } a \text{ and of } \varrho. \end{aligned} \tag{4}$$

By using the method of undetermined coefficients, implementing the problem in software for differential calculus on jet spaces (e.g., `Jets` by M. Marvan [21] or `gcaops` by R. Buring [3]), we obtain the nontrivial solution (see also Example 6 on p. 454 below). The differential polynomial  $\dot{\mathbf{a}}([\varrho], [a])$  consists of 228 monomials with nonzero coefficients, and  $\dot{\varrho}([\varrho], [a])$  is 426 monomial long. It is seen that the actual dependence of  $\dot{\mathbf{a}}$  and  $\dot{\varrho}$  on the jet variables  $a_\sigma$  and  $\varrho_\tau$  is such that the lengths of multi-indices  $\sigma$  and  $\tau$  are bounded by  $1 \leq |\sigma| \leq 3$  and  $0 \leq |\tau| \leq 1$  for  $\dot{\mathbf{a}}$  and by  $1 \leq |\sigma| \leq 2$  and  $0 \leq |\tau| \leq 3$  for  $\dot{\varrho}$ . Apparent is also that in each monomial, there are exactly three derivatives w.r.t.  $x$ , exactly three w.r.t.  $y$ , and exactly three w.r.t.  $z$ . Here is a sample how these formulas read:

$$\begin{aligned} \dot{\mathbf{a}} &= -12\varrho^2 a_x \varrho_y a_{xy} a_{zz} a_{xyz} + 12\varrho^2 a_x \varrho_y a_{xy} a_{xz} a_{yzz} + 12\varrho^2 a_x \varrho_y a_{xy} a_{xzz} a_{yz} + \dots, \\ \dot{\varrho} &= -12\varrho \varrho_x \varrho_y a_x a_z \varrho_{xy} a_{zz} - 12\varrho \varrho_x \varrho_y a_x a_z \varrho_{xz} a_{yy} + 24\varrho \varrho_x \varrho_y a_x a_z \varrho_{xy} a_{yz} + \dots; \end{aligned}$$



both formulas are given in full in Appendix B. In what follows, we shall explain these empiric facts; by understanding the combinatorics in these formulas, we collapse them to tiny Eqs. (7) on p. 457.

**Example 4** ( $\gamma_3$ -flow over  $\mathbb{R}^4$ ). In contrast with 3D, the tetrahedral flow  $\dot{P} = \text{Or}(\gamma_3)(P^{\otimes 4})$  is nonzero for the “authentic” Nambu-determinant Poisson bi-vector  $P = da_1 \wedge da_2/dxdydzdw$  with pre-factor  $\varrho \equiv 1$  on  $\mathbb{R}^4 \ni \mathbf{x} = (x, y, z, w)$ . With this Cauchy datum  $\varrho \equiv 1$  implying  $\dot{\varrho} \equiv 0$ , we obtain that the differential polynomial velocities  $\dot{a}_1([a_1], [a_2])$  and  $\dot{a}_2([a_1], [a_2])$  each contain 9024 monomials (of two unequal differential profiles, 4512 and 4512 each, see Example 9).

Now the full case on  $\mathbb{R}^4$ : take the generalized Nambu--Poisson bi-vector  $P(\varrho, [a_1], [a_2]) = \varrho(\mathbf{x}) \cdot da_1 \wedge da_2/dxdydzdw$ . The tetrahedral flow  $\dot{P} = \text{Or}(\gamma_3)(P^{\otimes 4})$  does preserve this class of Poisson brackets: there exist differential-polynomial velocities of the inverse density  $\varrho(\mathbf{x})$  and of the two Casimirs  $a_1, a_2$  such that

$$\begin{aligned} \dot{\varrho} &= \dot{\varrho}([\varrho], [a_1], [a_2]) \text{ with } 90,024 \text{ terms,} \\ \dot{a}_1, \dot{a}_2 &([\varrho], [a_1], [a_2]) \text{ with } 33,084 \text{ terms each.} \end{aligned}$$

The combinatorial structure of these right-hand sides in the general case ( $\varrho \neq 1$ ) can be analysed by the technique which we develop in what follows: each of the three expressions is collapsed by using the marker-monomials for the triple summation with the Civita symbols on  $\mathbb{R}^4$ . For instance, the differential monomials in either  $\dot{a}_1$  or  $\dot{a}_2$  are partitioned according to the homogeneity profiles of derivatives of  $\varrho, a_1$ , and  $a_2$  with respect to the four coordinates on  $\mathbb{R}^4$  (see Table 2 on p. 462 below). And all the 33,048 terms in  $\dot{a}_1$  and  $\dot{a}_2$  are expressed by formulas (9) on p. 458.

**Example 5** ( $\gamma_5$ -flow over  $\mathbb{R}^3$ ). The pentagon-wheel flow  $P = \text{Or}(\gamma_5)(P^{\otimes 6})$  on the space of all Poisson structures on  $\mathbb{R}^3$  restricts to the Nambu-determinant class of brackets  $\{P(\varrho, [a])\}$ . In the differential-polynomial formulas of evolution  $\dot{\varrho}([\varrho], [a])$  and  $\dot{a}([\varrho], [a])$ , the right-hand side of  $\dot{a}$  contains 79,212 monomials, and there are as many as 146,340 in  $\dot{\varrho}$  (before either formula is collapsed by using five Civita symbols). In the meantime, one can estimate the homogeneity degrees and orders, that is the polynomial degrees of each term in  $\dot{a}$  and  $\dot{\varrho}$  with respect to the jet variables  $a_\sigma$  and  $\varrho_\tau$ , as well as the bounds on the possible (but not necessarily attained) lengths of the multi-indices  $\sigma$  and  $\tau$  counting the derivatives. We note that in every monomial, there are 5 subscripts  $x$  (for derivatives, which is the usual notation), 5 subscripts  $y$ , and 5 subscripts  $z$ ; both  $\dot{a}$  and  $\dot{\varrho}$  are manifestly skew-symmetric w.r.t. permutations of the base variables  $x, y, z$  (meaning that the right-hand sides contain at least one Civita symbol  $\varepsilon^{i_1 i_2 i_3}$ ).

### 3. THE STRUCTURE OF INDUCED EVOLUTION $\dot{\varrho}, \dot{\mathbf{a}}$

**3.1. Encoding  $\dot{\mathbf{a}}, \dot{\varrho}$  by the Kontsevich graphs.** The Kontsevich flow  $\dot{P} = \text{Or}(\gamma)(P^{\otimes n})$  on the class of generalized ( $\varrho \neq 1$ ) Nambu-determinant Poisson brackets  $P(\varrho, [\mathbf{a}])$  preserves their structure. Let us interpret this fact back in the language of Kontsevich's directed graphs.

**Proposition 1** ([20]). The evolution  $\dot{a}_i$  of each Casimir in the Jacobian determinant within the Nambu–Poisson bracket,

$$\{f, h\}_{P(\varrho, [\mathbf{a}])} = \varrho(\mathbf{x}) \cdot \det \left\| \partial(a_1, \dots, a_{d-2}, f, g) / \partial(x^1, \dots, x^d) \right\|,$$

is equal to the value of the graph orientation morphism  $\text{Or}$  at the  $n$ -tuple  $P^{\otimes n-1} \otimes a_i$  (here  $n = \#\text{Vert}(\gamma)$ ):

$$\dot{a}_i = \text{Or}(\gamma)(P^{\otimes n-1}, a_i), \quad (5)$$

where the right-hand side represents the sum of  $n$ -linear polydifferential operators which are encoded by the directed graph cocycle  $\text{Or}(\gamma)$  and which are evaluated at  $a_i$  placed consecutively in one of the vertices and the other vertices filled in by copies of the bi-vector  $P(\varrho, [\mathbf{a}])$ .

*Commentary.* Indeed, the Kontsevich graph flows  $\dot{P} = \text{Or}(\gamma)(P^{\otimes n})$  are such that no arrows fall on the checked factor  $\check{\varrho}$  in the Leibniz formula for  $\dot{P}$ ,

$$\dot{P}([\varrho], [\mathbf{a}]) = P(\check{\varrho}, [\mathbf{a}]) + \sum_{i=1}^{d-2} P(\check{\varrho}, [a_1], \dots, [\dot{a}_i], \dots, [a_{d-2}]). \quad (2)$$

More specifically, to let exist the restriction of Kontsevich’s graph flow to the class of Nambu-determinant Poisson brackets  $P(\varrho, [\mathbf{a}]) = \varrho(x) \cdot d\mathbf{a}/d\mathbf{x}$ , the directed graph formula, working over the content of each internal vertex by the Leibniz rule for each incoming arrow, automatically singles out the terms in which (i) the pre-factor  $\varrho$  remains intact and (ii) the in-coming derivatives are not spread over several Casimirs in the Jacobian inside that vertex. Nontrivial in this claim is that precisely all –without exception– terms of such structure do form the well defined tuple of velocities  $\dot{\mathbf{a}}$ .  $\square$

**Corollary 2.** As soon as the evolution  $\dot{\mathbf{a}}$  of the Casimirs is obtained according to formula (5), from the structure of  $P = \varrho(x) \cdot d\mathbf{a}/d\mathbf{x}$  of the Nambu bracket and from the Leibniz rule in Eq. (2) we deduce the speed of evolution for the inverse density  $\dot{\varrho}$ . Namely, we have that

$$\dot{\varrho} \cdot \left| \frac{\partial(a_1, \dots, a_{d-2}, f, g)}{\partial(x^1, \dots, x^d)} \right| = \left( \dot{P}([\varrho], [\mathbf{a}]) - \sum_{i=1}^{d-2} P(\varrho, [a_1], \dots, [\dot{a}_i], \dots, [a_{d-2}]) \right) (f, g), \quad (6)$$

where  $f, g \in C^\infty(\mathbb{R}^d)$ , the right-hand side with the known flow  $\dot{P} = \text{Or}(\gamma)(P^{\otimes n})$  is the value of the linear combination of bi-vectors at  $f \otimes g$ , and  $\dot{\varrho}$  is extracted from the left-hand side by division.

*Commentary.* Indeed, by the above, the right-hand side is a whole multiple of the Jacobian, which itself is equal to  $P(\varrho \equiv 1, [\mathbf{a}])(f, g)$ .  $\square$

**Example 6.** The above proposition and corollary, resulting in the explicit differential-polynomial expressions for the velocities  $\dot{\varrho}$  and  $\dot{\mathbf{a}}$  that induce a given graph flow  $\dot{P} = \text{Or}(\gamma)(P^{\otimes n})$  for the Nambu structures  $P = \varrho(\mathbf{x}) \cdot d\mathbf{a}/d\mathbf{x}$  on  $\mathbb{R}^d$ , are illustrated in [3, Ch. 6] by using the `gcaops` software for differential calculus on jet spaces. So far, the graph formulas are explicitly verified for

- the tetrahedral flow ( $\gamma = \gamma_3$ ) on  $\mathbb{R}^3$ ;
- the tetrahedral flow ( $\gamma = \gamma_3$ ) on  $\mathbb{R}^4 \ni \mathbf{x}$  with  $\varrho \equiv 1$  (special case) and generic  $\varrho(\mathbf{x})$  which implies  $\dot{\varrho} \neq 0$ ;

- the pentagon-wheel flow ( $\gamma = \gamma_5$ ) on  $\mathbb{R}^3$  with generic  $\varrho$ .

For the tetrahedral  $\gamma_3$ -flow on  $\mathbb{R}^3$ , the findings from Example 3 are reproduced identically. A naive attempt to use the method of undetermined coefficients would be practically unfeasible in the other three cases, yet Eqs. (5) and (6) serve the correct formulas of  $\dot{\mathbf{a}}$  and  $\dot{\varrho}$  without any need to solve a linear algebraic system.

**3.2. Civita symbols in  $\dot{\varrho}$ ,  $\dot{\mathbf{a}}$ : collapsing the formulas.** Our present task is to analyze the combinatorial structure of the differential-polynomial expressions for  $\dot{\mathbf{a}}$  and  $\dot{\varrho}$  in formulas (5) and (6), respectively, and collapse them as much as possible by using this new knowledge.

3.2.1. *The determinant provides one Civita symbol.* One simple fact is immediate from the presence of Jacobian determinant in the Nambu brackets.

**Proposition 3.** The differential polynomials  $\dot{\mathbf{a}}([\varrho], [\mathbf{a}])$  and  $\dot{\varrho}([\varrho], [\mathbf{a}])$  are shifted-graded skew-symmetric w.r.t. permutations of the base variables  $x^1, \dots, x^d$  (i.e. the coordinates on the Poisson manifold  $\mathbb{R}^d$ ): for a graph cocycle  $\gamma$  on  $n$  vertices in each term, the velocities  $\dot{\varrho}$  and  $\dot{a}_i$  are skew-symmetric in  $x^1, \dots, x^d$  if  $n$  is even (e.g., as for  $\gamma_3, \gamma_5, \gamma_7, \dots, \gamma_{2\ell+1}, \dots$ ) and symmetric in  $x^1, \dots, x^d$  if  $n$  is odd (e.g., such is the case for the cocycle  $[\gamma_3, \gamma_5]$  on 9 vertices and 16 edges).

*Proof.* Every Poisson bracket of two scalar functions itself is a scalar. For the Nambu bracket in particular,

$$\{f, g\}_{P(\varrho, [\mathbf{a}])} = \varrho(\mathbf{x}) \cdot \det \left\| \partial(a_1, \dots, a_{d-2}, f, g) / \partial(x^1, \dots, x^d) \right\|,$$

this invariance is provided by the response (1) of the inverse density  $\varrho$  to a permutation  $\sigma$  of rows in the Jacobian determinant:  $\varrho(\mathbf{x}) = (-)^\sigma \cdot \varrho'(\mathbf{x}'(\mathbf{x}))$  if  $\mathbf{x} = \sigma(\mathbf{x}')$ . A simple count shows that for a graph cocycle  $\gamma$  on  $n$  vertices (and  $2n - 2$  edges), we have that

$$\begin{aligned} \dot{\varrho} &\sim \varrho^n \cdot a_1^{n-1} \cdot \dots \cdot a_{d-1}^{n-1} && \text{with } (n - 1) \times d \text{ base variables } x^1, \dots, x^d; \\ \dot{a}_i &\sim \varrho^{n-1} \cdot a_i^n \cdot \prod_{j \neq i} a_j^{n-1} && \text{with } (n - 1) \times d \text{ base variables } x^1, \dots, x^d. \end{aligned}$$

For the velocities  $\dot{a}_i$  to be scalars and for the objects  $\dot{\varrho}$  to behave according to the same law,  $\dot{\varrho}|_{\mathbf{x}} = (-)^\sigma \dot{\varrho}'|_{\mathbf{x}'(\mathbf{x})}$ , as the inverse density  $\varrho$  satisfies, both the right-hand sides have the parity  $((-)^{n-1})^\sigma$  whenever the base variables are permuted:  $\mathbf{x} = \sigma(\mathbf{x}')$ .  $\square$

**Example 7** ( $\gamma_3$ -flow over  $\mathbb{R}^3$ ). Indeed, for the tetrahedral  $\gamma_3$ -flow on the space of Nambu–Poisson structures  $P(\varrho, [a])$  on  $\mathbb{R}^3$ , with 228 terms in  $\dot{\mathbf{a}}$  and 426 terms in  $\dot{\varrho}$ , we verify that

$$\begin{aligned} \dot{\mathbf{a}}([\varrho], [a])(x, y, z) &= \sum_{\sigma \in \mathbb{S}_3} (-)^\sigma \sigma(x, y, z) \text{ acts on (sum of 38 terms),} \\ \dot{\varrho}([\varrho], [a])(x, y, z) &= \sum_{\sigma \in \mathbb{S}_3} (-)^\sigma \sigma(x, y, z) \text{ acts on (sum of 71 terms).} \end{aligned}$$

The differential monomials in the right-hand sides are obtained by the greedy algorithm: for a monomial that still remains in the expression to be represented as an alternating sum, take its skew-symmetrization w.r.t.  $\sigma \in \mathbb{S}_3$  acting on  $x, y, z$ , subtract it from the expression, collect similar terms and reduce, then proceed recursively until the list of monomials, initially met in the velocity, is empty.

We shall presently recognize such one-time skew-symmetrizations (when  $n$  is even) within  $\dot{\rho}$  and  $\dot{a}_i$  as a consequence of a much stronger claim about the independent action of  $n - 1$  copies of the permutation group  $\mathbb{S}_d$  on the  $d$ -tuples  $\{x^1, \dots, x^d\}_k$  for  $1 \leq k \leq n - 1$  in the right-hand sides  $\dot{\rho}$  and  $\dot{\mathbf{a}}$ . For instance, in the above example (here  $n = 4$  and  $d = 3$ ) the sign factor  $(-)^{\sigma}$  is produced by the restriction on the diagonal,

$$\sum_{\sigma \in \mathbb{S}_3} (-)^{\sigma} \sigma \left( \bigotimes_{k=1}^{n-1} \{x, y, z\}_k \right) = \sum_{\sigma_1, \dots, \sigma_{n-1} \in \mathbb{S}_3} (-)^{\sigma_1} \dots (-)^{\sigma_{n-1}} \bigotimes_{k=1}^{n-1} \sigma_k(\{x, y, z\}_k) \Big|_{\sigma_k = \sigma},$$

in the set of  $n - 1 = 3$  permutations  $\sigma_k \in \mathbb{S}_d$  acting on the  $n - 1$  non-intersecting  $d$ -tuples  $\{x^1, \dots, x^d\}_k$  that partition the set of  $(n - 1) \times d$  derivatives occurring in the right-hand sides of  $\dot{\rho}$  and  $\dot{a}$ .

3.2.2. *How  $\rho^k$  yields  $k$  Civita symbols, or: Jacobians generalized.* Let us recall three facts from analysis:

- the Casimirs  $\mathbf{a} = (a_1, \dots, a_{d-2})$  of the Nambu–Poisson brackets are scalars;
- the inverse density  $\rho$  obeys the transformation law  $\rho(\mathbf{x}) = \rho'(\mathbf{x}') \cdot \det \|\partial \mathbf{x} / \partial \mathbf{x}'\| \Big|_{\mathbf{x}'(\mathbf{x})}$  under a change  $\mathbf{x}(\mathbf{x}') \rightleftharpoons \mathbf{x}'(\mathbf{x})$ ;
- the objects' velocities inherit the behaviour of those objects under the coordinate transformations.

Consider a Kontsevich flow  $\dot{P} = \text{Or}(\gamma)(P^{\otimes n})$  associated with a graph cocycle  $\gamma$  on  $n$  vertices. These three facts, put together, reveal that the reparametrization of derivatives of  $\mathbf{a}$  and  $\rho$  in the differential monomials within  $\dot{\rho}([\rho], [\mathbf{a}])$  and  $\dot{a}([\rho], [\mathbf{a}])$  match the nontrivial reparametrization of  $n - 1$  copies of  $\rho$  therein.

More specifically, the  $(n - 1) \times d$  derivations  $\partial_{x^1}^{\otimes n-1} \otimes \dots \otimes \partial_{x^d}^{\otimes n-1}$  arrange into  $n - 1$  totally skew-symmetric  $d$ -tuples  $\varepsilon^{i_1 \dots i_d} \partial_{x^1} \otimes \dots \otimes \partial_{x^d}, \dots, \varepsilon^{i_1^{n-1} \dots i_d^{n-1}} \partial_{x^1} \otimes \dots \otimes \partial_{x^d}$ , where  $\varepsilon^{\vec{i}^\alpha}$  is the Civita symbol on  $\mathbb{R}^d$ . The derivatives from each  $d$ -tuple act on different comultiples of a *marker-monomial* (which stands under the sum over the  $n - 1$  tuples  $\vec{i}^1, \dots, \vec{i}^{n-1}$  with  $d$  indices in each tuple and which thus marks, generally speaking, many monomials in the polynomial expressions  $\dot{\rho}$  and  $\dot{a}_\ell$  when the sums over  $\vec{i}^\alpha$  are expanded). In effect, each of the  $d$ -tuples  $\partial_{x^1} \wedge \dots \wedge \partial_{x^d}$  provides its own Jacobian determinant  $\det \|\partial \mathbf{x}' / \partial \mathbf{x}\|$  when the coordinates are reparametrized on the affine base manifold  $\mathbb{R}^d$ . These  $n - 1$  Jacobians  $|\partial \mathbf{x}' / \partial \mathbf{x}|$  cancel against the  $n - 1$  Jacobians  $|\partial \mathbf{x} / \partial \mathbf{x}'|$  from the reparametrizations of the inverse density  $\rho$  in either  $\dot{\rho}$  or  $\dot{a}_\rho$ .

**Theorem 4.** *For a graph cocycle  $\gamma$  on  $n$  vertices, the Kontsevich flow  $\dot{P} = \text{Or}(\gamma)(P^{\otimes n})$  restricts to the Nambu class  $P = \rho(\mathbf{x}) \cdot d\mathbf{a} / d\mathbf{x}$  of Poisson brackets on the affine space  $\mathbb{R}^d$  in such a way that*

$$\begin{aligned} \dot{a}_\ell &= \sum_{\sigma_1, \dots, \sigma_{n-1} \in \mathbb{S}_d} \left( \prod_{i=1}^{n-1} (-)^{\sigma_i} \sigma_i((x^1, \dots, x^d)_i) \right) (\text{marker-monomials} \sim \rho^{n-1} a_\ell^n \cdot \prod_{k \neq \ell}' a_k^{n-1}), \\ \dot{\rho} &= \sum_{\sigma_1, \dots, \sigma_{n-1} \in \mathbb{S}_d} \left( \prod_{i=1}^{n-1} (-)^{\sigma_i} \sigma_i((x^1, \dots, x^d)_i) \right) (\text{marker-monomials} \sim \rho^n a_1^{n-1} \dots a_{d-2}^{n-1}). \end{aligned}$$

The permutations  $\sigma_i \in \mathbb{S}_d$  act on the partitioned set of subscripts (for derivatives)  $\bigsqcup_{i=1}^{n-1} ((x^1, \dots, x^d)_i)$  in each marker-monomial. Equivalently, we have that

$$\begin{aligned} \dot{a}_\ell &= \sum_{\vec{i}^1, \dots, \vec{i}^{n-1}} \left( \bigotimes_{\alpha=1}^{n-1} \varepsilon^{\vec{i}^\alpha} \cdot \partial_{\vec{i}^\alpha} \right) (\text{comultiples in marker-monomials} \sim \varrho^{n-1} a_\ell^n \cdot \prod_{k \neq \ell}' a_k^{n-1}), \\ \dot{\varrho} &= \sum_{\vec{i}^1, \dots, \vec{i}^{n-1}} \left( \bigotimes_{\alpha=1}^{n-1} \varepsilon^{\vec{i}^\alpha} \cdot \partial_{\vec{i}^\alpha} \right) (\text{comultiples in marker-monomials} \sim \varrho^n a_1^{n-1} \cdot \dots \cdot a_{d-2}^{n-1}), \end{aligned}$$

with  $d$ -component multi-indices  $\vec{i}^\alpha = (i_1^\alpha, \dots, i_d^\alpha)$  in the Civita symbols  $\varepsilon^{\vec{i}^\alpha}$  on  $\mathbb{R}^d$ .

*Commentary.* The right-hand side of the velocity  $\dot{a}^\ell$  or  $\dot{\varrho}$  is a very interesting object from analytic and combinatorial viewpoint: with all the first- and higher-order derivatives in the velocity, it looks like the Jacobian determinant w.r.t. *each* tuple  $(x^1, \dots, x^d) \Leftrightarrow (\partial_{x^1}, \dots, \partial_{x^d})$ , although the derivatives from different tuples can act on the same comultiple. It remains therefore to control the non-tensorial behaviour of the *higher-order* derivatives (which do actually occur in the expressions under study, as seen from examples). Fortunately, this is where our initial assumption works: Kontsevich's graph flow is defined over an *affine* manifold so that the second- and higher-order derivatives of the coordinate changes vanish identically for  $\mathbf{x} = A\mathbf{x}' + \mathbf{b}$  (with a constant Jacobian matrix  $A$ ). Thus, higher derivatives of  $a_\ell$  and  $\varrho$  are transformed by using only the first derivatives of the coordinate changes, whence the assertion.  $\square$

**Example 8** ( $\gamma_3$ -flow over  $\mathbb{R}^3$ ). For the tetrahedral flow  $\dot{P} = \text{Or}(\gamma_3)(P^{\otimes 4})$  over  $\mathbb{R}^3$  we recall from Eq. (4) in Example 3 that

$$\begin{aligned} \dot{a} &\sim \varrho^3 a^4 \text{ with } xxxxyyzzz \text{ in each monomial,} \\ \dot{\varrho} &\sim \varrho^4 a^3 \text{ with } xxxxyyzzz \text{ in each monomial.} \end{aligned}$$

Now Theorem 4 works: the  $(4 - 1) \times 3$  base variables are partitioned in three triples  $(x, y, z)$  in each term, with a skew-symmetrization over each triple. Indeed, by a brute force calculation we verify that for the  $\gamma_3$ -flow over  $\mathbb{R}^3$ ,

$$\begin{aligned} \dot{a} &= \sum_{\sigma, \tau, \zeta \in \mathbb{S}_3} (-)^\sigma (-)^\tau (-)^\zeta (2a_{u_1} a_{u_2} a_{u_3} \varrho_{w_1} \varrho_{w_2} \varrho_{w_3} a_{v_1} a_{v_2} a_{v_3} - 6\varrho a_{u_1} a_{v_2} a_{u_2} a_{u_3} \varrho_{w_1} \varrho_{w_3} a_{v_1} a_{v_3} w_2 \\ &\quad - 6\varrho^2 a_{u_1} a_{u_2} a_{u_3} a_{v_1} a_{v_2} \varrho_{w_3} a_{v_3} w_1 w_2), \\ \dot{\varrho} &= \sum_{\sigma, \tau, \zeta \in \mathbb{S}_3} (-)^\sigma (-)^\tau (-)^\zeta (-2a_{u_1} a_{u_2} a_{u_3} \varrho_{v_1} \varrho_{v_2} \varrho_{v_3} \varrho_{w_1} \varrho_{w_2} w_3 + 6a_{u_1} a_{v_2} a_{u_2} a_{u_3} \varrho_{v_1} \varrho_{v_3} \varrho_{w_2} \varrho_{w_1} w_3 \\ &\quad - 12\varrho a_{u_1} a_{u_2} a_{u_3} a_{v_1} a_{v_2} \varrho_{v_3} \varrho_{w_1} \varrho_{w_2} w_3 - 6\varrho a_{u_1} a_{v_2} a_{u_2} a_{u_3} \varrho_{v_1} \varrho_{v_3} \varrho_{w_1} w_2 w_3 \\ &\quad + 6\varrho^2 a_{u_1} a_{u_2} a_{u_3} a_{v_1} a_{v_2} \varrho_{v_3} \varrho_{w_1} w_2 w_3), \end{aligned} \tag{7}$$

where each summation runs over three permutations  $\sigma, \tau, \zeta \in \mathbb{S}_3$  giving three triples  $(u_1, v_1, w_1) = (\sigma(x), \sigma(y), \sigma(z))$ , also  $(u_2, v_2, w_2) = (\tau(x), \tau(y), \tau(z))$ , and  $(u_3, v_3, w_3) = (\zeta(x), \zeta(y), \zeta(z))$ .

We conclude that the 228 monomials in  $\dot{a}$  and 426 monomials in  $\dot{\varrho}$  which we started with are completely determined by only three marker-monomials for  $\dot{a}$  and five marker-monomials for  $\dot{\varrho}$  by using three Civita symbols in either formula.<sup>1</sup>

<sup>1</sup> Not only this: the three and five respective marker-monomials in both the velocities  $\dot{a}$  and  $\dot{\varrho}$  and the 1,504 differential monomials in each component of the bi-vector flow  $\dot{P} = \text{Or}(\gamma_3)(P^{\otimes 4})$  for  $P(\varrho, [a])$

The natural question is *how* the nine symbols  $xxxyyyzzz$  in each term were distributed among the disjoint triples  $xyz, xyz, xyz$  (to be permuted by  $\sigma, \tau$  and  $\zeta$  respectively); we shall analyze this in the next section.

**Example 9** ( $\gamma_3$ -flow of  $P(\varrho \equiv 1, [a_1], [a_2])$  on  $\mathbb{R}^4$ ). Consider the “authentic” Nambu-determinant bracket  $P(\varrho \equiv 1, [a_1], [a_2])$  and induce the  $\gamma_3$ -flow of the Casimirs  $a_1$  and  $a_2$ , see Example 4. Owing to Theorem 4 we collapse the 9,024 terms in either  $\dot{a}_1$  and  $\dot{a}_2$  to the thrice alternating formulas, namely

$$\dot{a}_1 = \sum_{\sigma, \tau, \zeta \in \mathbb{S}_4} (-)^{\sigma} (-)^{\tau} (-)^{\zeta} \left( 3a_{1;s_1 u_2 u_3} a_{1;t_1 t_2} a_{2;s_2} a_{2;s_3 v_1} a_{2;t_3 u_1} a_{1;v_2} a_{1;v_3} + 6a_{1;s_1 u_2} a_{1;t_1} a_{1;t_2 v_3} a_{1;u_3 v_1 v_2} a_{2;t_3 u_1} a_{2;s_2} a_{2;s_3} \right), \tag{8a}$$

$$\dot{a}_2 = \sum_{\sigma, \tau, \zeta \in \mathbb{S}_4} (-)^{\sigma} (-)^{\tau} (-)^{\zeta} \left( 3a_{1;s_1} a_{2;t_1 u_2} a_{2;u_1 u_3 v_2} a_{1;s_2 t_3} a_{1;s_3 t_2} a_{2;v_1} a_{2;v_3} - 3a_{1;s_1 t_2} a_{2;u_1} a_{2;u_2 u_3 v_1} a_{1;t_1} a_{1;t_3} a_{2;s_2 v_3} a_{2;s_3 v_2} \right), \tag{8b}$$

where each summation runs over three permutations  $\sigma, \tau, \zeta \in \mathbb{S}_4$  giving three 4-tuples  $(s_1, t_1, u_1, v_1) = (\sigma(x), \sigma(y), \sigma(z), \sigma(w))$ , also  $(s_2, t_2, u_2, v_2) = (\tau(x), \tau(y), \tau(z), \tau(w))$ , and  $(s_3, t_3, u_3, v_3) = (\zeta(x), \zeta(y), \zeta(z), \zeta(w))$ .

Again, our task is to explain *how* these formulas are obtained, i.e. how one can guess the right partitionings of  $xxxyyyzzzwww$  in each monomial into three 4-tuples  $(x, y, z, w)$ .

*Remark 2.* The partitioning  $xxxyyyzzzwww = xyzw \sqcup xyzw \sqcup xyzw$  within the second marker-monomial in the polynomial under the sum for the velocity  $\dot{a}_1$  in (8a) is different from the analogous partitioning in the second marker-monomial (with coefficient  $-3$ ) in the mirror-symmetric formula (8b) of the velocity  $\dot{a}_2$ . The structural inequivalence of the two partitionings does occur modulo the relabelling  $a_1 \rightleftharpoons a_2$  and modulo arbitrary reshuffles of the three 4-tuples  $\{s, t, u, v\}_k$  indexed by  $k \in \{1, 2, 3\}$  and arbitrary permutations of  $s, t, u, v$  in any of the 4-tuples. Indeed, the last marker-monomial in (8a) for  $\dot{a}_1$  contains the product of second derivatives  $a_{1;s_1 u_2} \cdot a_{1;t_2 v_3}$  in which all the three 4-tuples are mixed, whereas one of the 4-tuples is *not* present at all in the product of second derivatives  $a_{2;s_2 v_3} \cdot a_{2;s_3 v_2}$  in the last marker-monomial in (8b) for  $\dot{a}_2$ .

Yet both the marker-monomials yield the sums (over  $\sigma, \tau, \zeta \in \mathbb{S}_4$ ) which are mirror-reflections of each other under the swap  $a_1 \rightleftharpoons a_2$ . This is an example of marker-monomials’ hidden symmetry which we discuss in the next section.

**Example 10** ( $\gamma_3$ -flow over  $\mathbb{R}^4$  with  $\varrho \neq 1$ ). The 33,084 terms in  $\dot{a}_1$  or in its mirror-reflection  $\dot{a}_2$  are captured –for the tetrahedral  $\gamma_3$ -flow on the space of generalized Nambu–Poisson brackets  $P(\varrho, [a_1], [a_2])$  on  $\mathbb{R}^4 \ni \mathbf{x} = (x, y, z, w)$ – by three Civita symbols (or equivalently, by three permutations) using the formulas

---

on  $\mathbb{R}^3$  are completely determined by the eleven marker-monomials in the trivializing vector field  $\vec{X}$ , which we obtain for the  $\gamma_3$ -flow over  $\mathbb{R}^3$  in section 5.1.

$$\begin{aligned} \dot{a}_1 = & \sum_{\sigma_1, \sigma_2, \sigma_3 \in \mathbb{S}_4} (-)^{\sigma_1} (-)^{\sigma_2} (-)^{\sigma_3} \cdot (3a_{1;s_1 u_2 u_3} a_{1;t_1 t_2} a_{2;s_2} a_{2;s_3} v_1 a_{2;t_3} u_1 a_{1;v_2} a_{1;v_3} \varrho^3 \\ & + 6a_{1;s_1 u_2} a_{1;t_1} a_{1;t_2} v_3 a_{1;u_3} v_1 v_2 a_{2;t_3} u_1 a_{2;s_2} a_{2;s_3} \varrho^3 + 3a_{1;v_2} a_{1;t_1} u_2 v_3 a_{2;v_1} \varrho_{s_1} a_{1;u_1} a_{1;u_3} a_{2;s_2} t_3 a_{2;s_3} t_2 \varrho^2 \\ & - 6a_{1;s_1} v_3 a_{2;s_2} t_1 \varrho_{v_1} a_{1;s_3} u_1 v_2 a_{1;u_2} a_{1;u_3} a_{2;t_2} a_{2;t_3} \varrho^2 - 6a_{1;s_1} v_2 v_3 a_{1;t_1} a_{1;u_2} a_{1;u_3} v_1 a_{2;s_2} a_{2;s_3} u_1 a_{2;t_3} \varrho t_2 \varrho^2 \\ & + 6a_{1;s_1} a_{1;s_2} v_3 a_{1;s_3} u_1 a_{1;t_1} t_2 t_3 a_{2;v_1} \varrho_{v_2} a_{2;u_2} a_{2;u_3} \varrho^2 - 6a_{1;s_1} a_{1;t_1} u_2 u_3 a_{2;u_1} v_2 a_{1;t_2} a_{1;t_3} a_{2;s_2} a_{2;s_3} \varrho_{v_1} \varrho_{v_3} \varrho \\ & + 6a_{1;v_2} a_{1;s_1} a_{1;s_2} s_3 t_1 a_{1;t_2} t_3 \varrho_{v_1} \varrho_{v_3} a_{2;u_1} a_{2;u_2} a_{2;u_3} \varrho - 2a_{1;s_1} a_{1;t_2} a_{1;t_3} u_1 u_2 a_{1;u_3} a_{2;t_1} a_{2;s_2} a_{2;s_3} \varrho_{v_1} \varrho_{v_2} \varrho_{v_3}), \quad (9a) \end{aligned}$$

$$\begin{aligned} \dot{a}_2 = & \sum_{\sigma_1, \sigma_2, \sigma_3 \in \mathbb{S}_4} (-)^{\sigma_1} (-)^{\sigma_2} (-)^{\sigma_3} \cdot (3a_{1;s_1} a_{2;t_1} u_2 a_{2;u_1} u_3 v_2 a_{1;s_2} t_3 a_{1;s_3} t_2 a_{2;v_1} a_{2;v_3} \varrho^3 \\ & - 3a_{1;s_1} t_2 a_{2;u_1} a_{2;u_2} u_3 v_1 a_{1;t_1} a_{1;t_3} a_{2;s_2} v_3 a_{2;s_3} v_2 \varrho^3 - 6a_{1;u_1} a_{2;t_1} t_2 v_3 \varrho_{t_3} a_{1;u_2} v_1 a_{1;u_3} v_2 a_{2;s_1} a_{2;s_2} a_{2;s_3} \varrho^2 \\ & + 6a_{1;s_1} a_{1;t_1} t_3 a_{1;u_2} a_{2;v_2} a_{2;s_2} v_1 v_3 a_{2;s_3} t_2 a_{2;u_1} \varrho_{u_3} \varrho^2 + 6a_{1;t_1} u_2 a_{2;u_1} v_2 \varrho_{u_3} a_{2;s_1} s_2 s_3 a_{1;v_1} a_{1;v_3} a_{2;t_2} a_{2;t_3} \varrho^2 \\ & + 3a_{2;v_1} a_{2;s_1} s_2 s_3 \varrho_{t_1} a_{2;t_2} v_3 a_{2;t_3} v_2 a_{1;u_1} a_{1;u_2} a_{1;u_3} \varrho^2 - 6a_{1;t_1} u_2 a_{2;u_1} u_3 v_2 a_{1;v_1} a_{1;v_3} \varrho_{t_2} \varrho_{t_3} a_{2;s_1} a_{2;s_2} a_{2;s_3} \varrho \\ & - 6a_{2;t_1} u_2 v_3 a_{2;u_1} u_3 a_{2;v_1} a_{2;v_2} \varrho_{t_2} \varrho_{t_3} a_{1;s_1} a_{1;s_2} a_{1;s_3} \varrho + 2a_{2;s_1} a_{2;s_2} s_3 t_1 \varrho_{v_1} a_{2;v_2} a_{2;v_3} \varrho_{t_2} \varrho_{t_3} a_{1;u_1} a_{1;u_2} a_{1;u_3}), \quad (9b) \end{aligned}$$

here  $\{s_i, t_i, u_i, v_i\} = \sigma_i(x, y, z, w)$  for  $\sigma_i \in \mathbb{S}_4$ . Finding a compact expression of  $\dot{\varrho} \neq 0$  with 90,024 differential monomials in it, now using three Civita symbols, is a computationally much larger task than collapsing the velocities of the Casimirs.

#### 4. MARKER-MONOMIALS AND THEIR HIDDEN SYMMETRY

**Definition 1.** A *marker-monomial* in the fibre variables  $\varrho, a_1, \dots, a_{d-2}$  over the base variables  $x^1, \dots, x^d$  is a differential monomial in the jet variables  $\varrho_\kappa, a_{1;\lambda_1}, \dots, a_{d-2;\lambda_{d-2}}$  (here the multi-indices  $\kappa, \lambda_i$  for the derivatives satisfy  $0 \leq |\kappa|, |\lambda_i| < \infty$ ) such that  $|\kappa| + \sum_{i=1}^{d-2} |\lambda_i| = \mu \cdot d$  with  $\mu \in \mathbb{N}_{\geq 1}$ , such that  $\kappa \cup \bigcup_{i=1}^{d-2} \lambda_i = \bigcup_{k=1}^{\mu} \{x^1, \dots, x^d\}_k$ , and such that all the base variables  $x^\ell$  in the multi-indices (denoting the respective derivatives) are partitioned into  $\mu$  disjoint  $d$ -tuples  $x^1, \dots, x^d$ . Every such tuple then corresponds to its own alternating sum  $\sum_{\sigma \in \mathbb{S}_d} (-)^\sigma \sigma(x^1, \dots, x^d)$  acting on the marker-monomial, or equivalently, corresponds to the Civita summation  $\sum_{\vec{v}} \varepsilon^{\vec{v}}$  chosen such that the marker-monomial “as is” occurs with the plus sign when  $\vec{v} = (1, 2, \dots, d)$ , so the base variables  $x^1, \dots, x^d$  in the  $d$ -tuple are then represented by the variables  $x^{i_1}, \dots, x^{i_d}$  in the subscripts, respectively.

**Definition 2.** A marker-monomial is called *zero* if the alternating sum over all permutations of all the  $d$ -tuples  $(x^1, \dots, x^d)$  in it is identically equal to zero.

**Example 11.** Let  $x, y$  be the base variables and  $\varrho$  be the fibre variable. Consider the marker-monomial  $M_1 = \varrho_{x_1} \varrho_{y_1} \varrho_{x_2 y_2}$  with the two-tuples  $\{x_1 y_1\} \sqcup \{x_2 y_2\}$ . Taking the alternating sum,

$$\sum_{\sigma \in \mathbb{S}_2} \sum_{\tau \in \mathbb{S}_2} (-)^\sigma (-)^\tau \varrho_{\sigma(x)} \varrho_{\sigma(y)} \cdot \varrho_{\tau(x)\tau(y)} = (\varrho_x \varrho_y - \varrho_y \varrho_x) \cdot (\varrho_{xy} - \varrho_{yx}) \equiv 0,$$

we establish that the marker-monomial  $M_1$  is a zero marker.

• But let us instead take the marker-monomial  $M_2 = \varrho_{x_1} \varrho_{y_2} \varrho_{x_2 y_1}$  with a different partitioning of the letters  $xyxy$  as they are seen in  $M_1$ . We now have that

$$\varrho_x \varrho_y \varrho_{xy} - \varrho_y \varrho_y \varrho_{xx} - \varrho_x \varrho_x \varrho_{yy} + \varrho_y \varrho_x \varrho_{yx} \neq 0. \quad (10)$$

In other words, the new marker-monomial  $M_2$  is not zero any longer, even though the profile  $|\sigma_1| = 1 = |\sigma_2|$ ,  $|\sigma_3| = 2$  of the comultiples in the product  $\varrho_{\sigma_1} \cdot \varrho_{\sigma_2} \cdot \varrho_{\sigma_3}$  is the same as in  $M_1$ .

**Definition 3.** The *differential profile* of orders of the derivatives in a marker-monomial  $M = \varrho_{\kappa_1} \varrho_{\kappa_2} \dots a_{1;\lambda_1} \dots a_{d-2;\mu_1} \dots$  is the set of pairs  $\{\varrho|\kappa_1|, \varrho|\kappa_2|, \dots, a_1|\lambda_1|, \dots, a_{d-2}|\mu_1|, \dots\} \stackrel{\text{def}}{=} \{\varrho|\kappa_1||\kappa_2| \dots, a_1|\lambda_1| \dots, \dots, a_{d-2}|\mu_1| \dots\}$ : each (instance of a) fibre variable is followed by the nonnegative order(s) of its derivative(s).<sup>2</sup>

**Example 12.** Both marker-monomials in Example 11 have the same differential profile  $\varrho_1 \varrho_1 \varrho_2$  (equivalently,  $\varrho_1 1 2$ ), yet  $M_1$  is zero whereas  $M_2$  is not zero as a marker.

The differential profile of a marker-monomial is thus a coarse invariant (w.r.t. permutations of all the base variable in it, or w.r.t. a permutation of the base variables within one of the  $d$ -tuples  $x^1, \dots, x^d$ ). It is clear also that marker-monomials of unequal differential profiles cannot be obtained one from another by permuting the comultiples or by permuting the base variables (what the alternating sum does by definition). This implies that to represent a differential polynomial by an alternating sum over the permutations which act on the base variable in the marker-monomials, the sums of terms of unequal differential profiles can be processed independently one from another.

*Remark 3.* Representations of a differential polynomial by using marker-monomials are not unique. Indeed, the marker can be picked for any value of the permutation(s). For instance, we have that

$$\begin{aligned} \sum_{\sigma, \tau \in \mathbb{S}_2} (-)^{\sigma} (-)^{\tau} \varrho_{\sigma(x)} \varrho_{\tau(x)} \varrho_{\sigma(y)\tau(y)} &= \sum_{\sigma, \tau \in \mathbb{S}_2} (-)^{\sigma} (-)^{\tau} \varrho_{\sigma(y)} \varrho_{\tau(y)} \varrho_{\sigma(x)\tau(x)} = \\ &= - \sum_{\sigma, \tau \in \mathbb{S}_2} (-)^{\sigma} (-)^{\tau} \varrho_{\sigma(x)} \varrho_{\tau(y)} \varrho_{\sigma(y)\tau(x)} = - \sum_{\sigma, \tau \in \mathbb{S}_2} (-)^{\sigma} (-)^{\tau} \varrho_{\sigma(y)} \varrho_{\tau(x)} \varrho_{\sigma(x)\tau(y)}. \end{aligned}$$

Indeed, each of the four choices of the monomial marks the same expression,  $\varrho_x^2 \varrho_y - 2\varrho_x \varrho_y \varrho_{xy} + \varrho_y^2 \varrho_{xx} \neq 0$ .

At the same time, for two nonzero marker-monomials of equal differential profiles it can be that their alternating sums are neither equal nor proportional to each other but intersect, that is, the two resulting differential polynomials have common term(s).

**Counterexample 13.** The monomial  $a_x \varrho_x a_{yy} \varrho_{xy}$  is a term in the alternating sums for the markers

$$M_3 = a_{\sigma(x)} \varrho_{\tau(x)} a_{\sigma(y)\zeta(y)} \varrho_{\zeta(x)\tau(y)} \quad \text{and} \quad M_4 = a_{\sigma(x)} \varrho_{\tau(x)} a_{\tau(y)\zeta(y)} \varrho_{\zeta(x)\sigma(y)},$$

indeed showing up when  $\sigma = \tau = \zeta = \text{id}$ , but the two fully alternating sums are not equal,

$$\sum_{\sigma, \tau, \zeta \in \mathbb{S}_2} (-)^{\sigma} (-)^{\tau} (-)^{\zeta} \sigma \otimes \tau \otimes \zeta(M_3) \neq \sum_{\sigma, \tau, \zeta \in \mathbb{S}_2} (-)^{\sigma} (-)^{\tau} (-)^{\zeta} \sigma \otimes \tau \otimes \zeta(M_4),$$

which can be seen by straightforward expansion. The two differential polynomials are not even multiples of one another.

<sup>2</sup>The first variant of notation is inevitable if some of the orders is at least 10; in this note, the other variant of notation is enough (see Tables 1–2 in the next section).



This implies that to represent a given sum, the marker-monomial can be unique (up to a given permutation of the base variables in a  $d$ -tuple) but the choice of the base variables' partitioning (into the disjoint  $d$ -tuples) can be not unique, and only the right choice does the job. This ambiguity yields a nontrivial problem of finding the “true” partitioning of the  $\mu \cdot d$  derivatives into  $\mu$  tuples  $\{x^1, \dots, x^d\}$  in each term of the right-hand sides  $\dot{\rho}([\rho], [\mathbf{a}]), \dot{a}_\ell([\rho], [\mathbf{a}])$  for a given Kontsevich's graph flow on the space of Nambu–Poisson brackets  $P(\rho, [\mathbf{a}])$ .

We discover that this anticipated ambiguity is heavily suppressed by an extra, so far hidden symmetry of these graph flows on this particular class of Poisson brackets on  $\mathbb{R}^d$ .

**Proposition 5** ( $\dot{a}, \dot{\rho}$  for  $\gamma_3$ -flow over  $\mathbb{R}^3$ ). In the evolution  $\dot{\rho}, \dot{a}$  which is induced by the tetrahedral flow on the class of generalized ( $\rho \neq 1$ ) Nambu–Poisson brackets  $P(\rho, [a])$  on  $\mathbb{R}^3$ , the count of differential monomials of unequal differential profiles is presented in Table 1.<sup>3</sup>

TABLE 1. The number of monomials and their differential profiles in  $\dot{a}$  and  $\dot{\rho}$  for the tetrahedral  $\gamma_3$ -flow over  $\mathbb{R}^3$ .

In $\dot{a}$	In $\dot{\rho}$
54: $a1113\rho111$	54: $a111\rho1113$
	102: $a112\rho1112$
102: $a1123\rho011$	102: $a112\rho0113$
	96: $a122\rho0112$
72: $a1223\rho001$	72: $a122\rho0013$

- For each of the three differential profiles of monomials in  $\dot{a}$  and five in  $\dot{\rho}$ , we discover that for *any* choice of nonzero marker-monomial with that profile, its total skew-symmetrization (using three permutations, each acting on its own tuple  $xyz$ ), taken with a suitable nonzero coefficient, exactly equals the entire sum of all the terms with that differential profile. In other words, for each of the 3+5 differential profiles of monomials in  $\dot{a}$  and  $\dot{\rho}$  respectively, the total skew-symmetrizations of all nonzero markers of a fixed profile are multiples of each other.

This reveals a previously hidden, extra symmetry of the objects in Kontsevich's flow under study.

The case of Nambu–Poisson structures (with  $\rho \neq 1$ ) on  $\mathbb{R}^4$ , when the tetrahedral  $\gamma_3$ -flow induces the evolution  $\dot{a}_1, \dot{a}_2$  and  $\dot{\rho} \neq 0$ , is even more interesting: we observe the exact same extra symmetry for all but one differential profiles, and one profile exceptionally requires the use of two marker-monomials.

<sup>3</sup> From Proposition 1 we recall that the velocity  $\dot{a}$  is encoded using the Kontsevich graphs by formula (5). Because the entire flow  $\dot{P} = \text{Or}(\gamma_3)(P^{\otimes 4})$  is specified by the directed graph cocycle  $\text{Or}(\gamma_3)$ , the velocity  $\dot{\rho}$  is deduced from Eq. (6). One can inspect in full detail how the arrows, targeted on a copy of  $a$  in the construction of  $\dot{a}$ , spread over copies of  $\rho$  and  $a$  to form  $\dot{\rho}$  in Eq. (2). This is why there is much similarity in the differential profiles of terms in the two velocities (as seen from Table 1).

**Proposition 6** ( $\dot{a}_1, \dot{a}_2$  for  $\gamma_3$ -flow with  $\varrho \neq 1$  over  $\mathbb{R}^4$ ). The count of differential monomials of unequal profiles in the velocities  $\dot{a}_1$  and  $\dot{a}_2$  (see Example 9) is summarized in Table 2. (The symmetry in how the Casimirs  $a_1$  and  $a_2$  appear in the Nambu-determinant Poisson bracket is naturally reflected in their evolution under the tetrahedral  $\gamma_3$ -flow).

TABLE 2. The count of monomials w.r.t. their differential profiles in  $\dot{a}_1$  and  $\dot{a}_2$  for the tetrahedral  $\gamma_3$ -flow on the space of generalized ( $\varrho \neq 1$ ) Nambu–Poisson brackets on  $\mathbb{R}^4$ .

In $\dot{a}_1$	In $\dot{a}_2$
4512: $a_1 1123 a_2 122 \varrho 000$	4512: $a_1 122 a_2 1123 \varrho 000$
4512: $a_1 1223 a_2 112 \varrho 000$	4512: $a_1 112 a_2 1223 \varrho 000$
3168: $a_1 1113 a_2 122 \varrho 001$	3168: $a_1 122 a_2 1113 \varrho 001$
7872: $a_1 1123 a_2 112 \varrho 001$	7872: $a_1 112 a_2 1123 \varrho 001$
3168: $a_1 1223 a_2 111 \varrho 001$	3168: $a_1 111 a_2 1223 \varrho 001$
3984: $a_1 1113 a_2 112 \varrho 011$	3984: $a_1 112 a_2 1113 \varrho 011$
3984: $a_1 1123 a_2 111 \varrho 011$	3984: $a_1 111 a_2 1123 \varrho 011$
1848: $a_1 1113 a_2 111 \varrho 111$	1848: $a_1 111 a_2 1113 \varrho 111$

- The homogeneous differential polynomial components of all profiles except the 7872 terms with  $a_1 1123 a_2 112 \varrho 001$  and the 7872 terms with  $a_1 112 a_2 1123 \varrho 001$  enjoy the same extra symmetry as at  $d = 3$ : just one, arbitrarily chosen nonzero marker-monomial suffices to express the entire sum. In particular, this is always so in the restricted case  $\varrho \equiv 1$  when  $\dot{\varrho} \equiv 0$  and the nontrivial velocities  $\dot{a}_1, \dot{a}_2$  realize the entire evolution of the class  $\{P(\varrho \equiv 1, [a_1], [a_2])\}$ .

In either of the two exceptional cases (one in  $\dot{a}_1$  and the other in  $\dot{a}_2$ , with necessarily  $\varrho \neq 1$ ), when two marker-monomials are needed, the first choice is still arbitrary but the next choice is constrained by the former.<sup>4</sup>

The marker-monomial expression of  $\dot{\varrho}$  in the generic case  $\varrho \neq 1$  on  $\mathbb{R}^4$  carrying the tetrahedral  $\gamma_3$ -flow – and a simultaneous study of the presence or absence of the new extra symmetry in it – is a computationally challenging problem; the same applies to the pentagon-wheel  $\gamma_5$ -flow on  $\mathbb{R}^3$  (to collapse the known evolution  $\dot{\varrho}, \dot{a} \neq 0$  by using five Civita symbols and to check the extra symmetry in the course of building the hypotheses about the  $\mu \cdot d = 5 \cdot 3$  base variables’ partitioning into  $\mu \cdot \{xyz\}$ ).

### 5. VECTOR FIELDS WHICH TRIVIALIZE THE FLOWS OF NAMBU BRACKETS

Finally, we examine the Poisson triviality of the *restriction* of Kontsevich’s graph flow  $\dot{P} = \text{Or}(\gamma)(P^{\otimes n})$  to the space of Nambu–Poisson structures  $P(\varrho, [\mathbf{a}])$  on  $\mathbb{R}^d$ . (There is no known mechanism for Kontsevich’s graph flows to be trivial in the second Poisson cohomology of  $P$  for nontrivial graph cocycles  $\gamma$  and generic Poisson structures.)

<sup>4</sup> This looks similar to the construction of a basis in  $\mathbb{E}^2$  by using a root system with the Coxeter graph  $\bullet \rightarrow \bullet$ : selecting the first vector is free but as one proceeds, the remaining direction is constrained.

**5.1. The trivializing vector field  $\vec{X}(\gamma_3, \varrho, a)$  and Civita symbols in it.** Let us make a few estimates of differential polynomial degrees and orders. For every graph cocycle  $\gamma = \sum_{\ell} c^{\ell} \cdot \gamma_{\ell}$  with graphs  $\gamma_{\ell}$  on  $n$  vertices and  $2n - 2$  edges, the restriction of Kontsevich's flow  $\dot{P} = \text{Or}(\gamma)(P^{\otimes n})$  to the space of generalized ( $\varrho \neq 1$ ) Nambu-determinant Poisson bi-vectors  $P(\varrho, [\mathbf{a}])$  with  $d - 2$  global Casimirs  $\mathbf{a} = (a_1, \dots, a_{d-2})$  on  $\mathbb{R}^d$  contains, in each term of the differential-polynomial coefficient of the bi-vector  $P(\varrho, [\mathbf{a}])$ ,  $n \cdot (d - 2) + 2n - 2 = nd - 2$  derivatives spread over  $\varrho^n \cdot a_1^n \dots a_{d-2}^n$ . (Hence there are  $(nd - 2) - (d - 2) = (n - 1)d$  derivatives spread over  $\varrho^{n-1} \cdot a_k^n \cdot \prod'_{j \neq k} a_j^{n-1}$  in  $\dot{a}_k$  and over  $\varrho^n \cdot a_1^{n-1} \dots a_{d-2}^{n-1}$  in  $\dot{\varrho}$ .) The trivializing vector field  $\vec{X} = \sum_{i=1}^d X^i([\varrho], [\mathbf{a}]) \partial / \partial x^i$  with differential-polynomial coefficients satisfying the coboundary equation  $\text{Or}(\gamma)(P^{\otimes n}) = \llbracket P, \vec{X} \rrbracket$  for  $P(\varrho, [\mathbf{a}])$  would therefore have  $(nd - 2) - (d - 2) - 1 = (n - 1)d - 1$  derivatives spread over  $\varrho^{n-1} \cdot a_1^{n-1} \dots a_{d-2}^{n-1}$  in every term of each coefficient  $X^i$ . The Civita mechanism of base coordinates' partitioning now applies to the trivializing vector field. For the object  $\vec{X}$  to be a vector field under coordinate reparametrizations  $\mathbf{x}(\mathbf{x}') \rightleftharpoons \mathbf{x}'(\mathbf{x})$ , the behaviour of  $n - 1$  comultiples  $\varrho$  dictates that there are  $n - 1$  Civita symbols in each  $X^i$ :

$$\vec{X} = \sum_{\vec{i}^1, \dots, \vec{i}^{n-1}} \varepsilon^{\vec{i}^1} \dots \varepsilon^{\vec{i}^{n-1}} \cdot X_{\vec{i}^1, \dots, \vec{i}^{n-2}; i_1^{n-1} \dots i_{d-1}^{n-1}} \cdot \partial / \partial x^{i_d^{n-1}}. \tag{11}$$

In other words, the vector field coefficients  $X^i$  collapse by using all the indices of  $n - 2$  Civita symbols  $\varepsilon^{\vec{i}^\alpha}$  on  $\mathbb{R}^d$  and by using all but one last index of the  $(n - 1)$ th Civita symbol.

It is readily seen that if the trivializing vector field exists, the velocities of the scalar Casimirs are  $\dot{a}_k = -\vec{X}(a_k)$ ; the proof is standard. Nontrivial here is that *zero* marker-monomials can be produced in the velocity  $\dot{a}_k = -(\sum_{i=1}^d X^i([\varrho], [\mathbf{a}]) \partial / \partial x^i)(a_k)$  from nonzero marker-monomials in the right-hand side of (11). This prompts that the velocity  $\dot{a}_k$ , which was obtained directly from the graph cocycle  $\gamma$  by using formula (5), can involve fewer marker-monomials than there are terms to express the coefficient  $X^i$  by (11). We observe this effect already in the simplest case, namely for the Kontsevich tetrahedral flow (so  $n = 4$ ) and the generalized ( $\varrho \neq 1$ ) Nambu-determinant Poisson structures  $P(\varrho, [a])$  on  $\mathbb{R}^3$  (so  $d = 3$ ).

**Theorem 7.** *The Kontsevich tetrahedral flow  $\dot{P} = \text{Or}(\gamma_3)(P^{\otimes 4})$  for the Nambu-Poisson brackets  $P(\varrho, [a])$  on  $\mathbb{R}^3$  is Poisson-cohomology trivial.*

- *The equivalence class  $\vec{X} \text{ mod } \llbracket P, H \rrbracket$  of trivializing vector fields  $\vec{X}$  satisfying the coboundary condition  $\text{Or}(\gamma_3)(P^{\otimes 4}) = \llbracket P, \vec{X} \rrbracket$  is represented by the following vector field with differential-polynomial coefficients  $X^i([\varrho], [a])$ :*

$$\vec{X} = \sum_{\vec{i}, \vec{j}, \vec{k}} \varepsilon^{\vec{i}} \varepsilon^{\vec{j}} \varepsilon^{\vec{k}} \cdot X_{\vec{i} \vec{j} \vec{k}},$$

where

$$\begin{aligned}
 X_{\vec{i}\vec{j}\vec{k}} = & + 12 \underline{\underline{\rho \rho_{x^{k_2}} \rho_{x^{i_1 x^{j_1}} a_{x^{k_3}} a_{x^{i_2 x^{j_2}} a_{x^{i_3 x^{j_3}}}}}} \cdot \partial / \partial x^{k_1} + 48 \underline{\underline{\rho \rho_{x^{j_3}} \rho_{x^{i_1 x^{j_1}} a_{x^{k_3}} a_{x^{i_2 x^{j_2}} a_{x^{i_3 x^{k_1}}}}}} \cdot \partial / \partial x^{k_2} \\
 & + 8 \underline{\underline{\rho_{x^{j_2}} \rho_{x^{i_1 x^{k_1}} \rho_{x^{i_2 x^{k_2}} a_{x^{i_3}} a_{x^{j_3}} a_{x^{k_3}}}}}} \cdot \partial / \partial x^{j_1} - 40 \underline{\underline{\rho_{x^{i_3}} \rho_{x^{j_2}} \rho_{x^{i_1 x^{k_1}} a_{x^{j_3}} a_{x^{k_3}} a_{x^{i_2 x^{k_2}}}}}} \cdot \partial / \partial x^{j_1} \\
 & + 8 \underline{\underline{\rho_{x^{i_3}} \rho_{x^{j_2}} \rho_{x^{k_3}} a_{x^{j_3}} a_{x^{i_1 x^{k_1}} a_{x^{i_2 x^{k_2}}}}}} \cdot \partial / \partial x^{j_1} + 24 \underline{\underline{\rho_{x^{j_2}} \rho_{x^{k_3}} \rho_{x^{i_1 x^{k_1}} a_{x^{i_3}} a_{x^{j_3}} a_{x^{j_1 x^{k_2}}}}}} \cdot \partial / \partial x^{i_2} \\
 & - 12 \underline{\underline{\rho^2 \rho_{x^{k_2}} a_{x^{i_1 x^{j_1}} a_{x^{i_2 x^{j_2}} a_{x^{i_3 x^{j_3}} x^{k_3}}}}}} \cdot \partial / \partial x^{k_1} + 24 \underline{\underline{\rho \rho_{x^{j_2}} \rho_{x^{k_1}} a_{x^{k_2}} a_{x^{i_1 x^{j_1}} a_{x^{i_3 x^{j_3}} x^{k_3}}}}}} \cdot \partial / \partial x^{i_2} \\
 & - 36 \underline{\underline{\rho \rho_{x^{i_2}} \rho_{x^{j_2}} a_{x^{k_2}} a_{x^{i_1 x^{j_1}} a_{x^{i_3 x^{j_3}} x^{k_3}}}}}} \cdot \partial / \partial x^{k_1} + 8 \underline{\underline{\rho_{x^{i_2}} \rho_{x^{j_1}} \rho_{x^{k_1}} a_{x^{j_2}} a_{x^{k_2}} a_{x^{i_3 x^{j_3}} x^{k_3}}}}}} \cdot \partial / \partial x^{i_1} \\
 & - 8 \underline{\underline{\rho_{x^{j_1}} \rho_{x^{k_1}} \rho_{x^{i_3 x^{j_3}} x^{k_3}} a_{x^{i_2}} a_{x^{j_2}} a_{x^{k_2}}}}}} \cdot \partial / \partial x^{i_1}.
 \end{aligned}$$

There are eleven terms in the marker-polynomial for  $X_{\vec{i}\vec{j}\vec{k}}$  but only the three underlined terms survive when the vector field  $\vec{X}$  acts on the Casimir  $a$ ; the rest contributes to the velocity  $\dot{a}$  with zero markers. The evolution  $\dot{a}$  in Eq. (5) is thus reproduced: we verify that  $\dot{a} = -\vec{X}(a)$ .

**5.2. Open problems about the graph flows and their trivializing vector fields  $\vec{X}([\rho], [a])$ .** The study of Kontsevich flows – for the tetrahedral and pentagon-wheel graph cocycles (or higher vertex number cocycles  $\gamma_7, [\gamma_3, \gamma_5], \gamma_9$ , etc.) – restricted to the spaces of generalized ( $\rho \neq 1$ ) Nambu-determinant Poisson brackets  $P([\rho], [\mathbf{a}])$  on  $\mathbb{R}^3$  and  $\mathbb{R}^4$  (or higher-dimensional affine spaces  $\mathbb{R}^d$ ) is, first of all, a source of combinatorial and algorithmic problems about finding the explicit shape of the objects. In particular, such is the task to collapse formulae, originally derived within the graph language, by using the Civita symbols. The other set of problems concerns the geometric nature and properties of the objects; such are the construction of the trivializing vector fields and explanation of the deeper symmetry in the choice of marker-monomials under the sums with Civita symbols. Let us summarize these problems in the order how they naturally emerge.

**Open problem 1** ( $\dot{\rho}$  in  $\gamma_3$ -flow over  $\mathbb{R}^4$ ). Represent the known velocity  $\dot{\rho}([\rho], [\mathbf{a}])$  for the tetrahedral  $\gamma_3$ -flow over  $\mathbb{R}^4$  by using three Civita symbols. Does the choice of marker-monomials enjoy the extra symmetry which is revealed in Proposition 5 for the  $\gamma_3$ -flow over  $\mathbb{R}^3$  and in Proposition 6 for  $\dot{\mathbf{a}}$  over  $\mathbb{R}^4$ ?

**Open problem 2** ( $\dot{\rho}, \dot{a}$  in  $\gamma_5$ -flow over  $\mathbb{R}^3$ ). Represent the known velocities  $\dot{\rho}([\rho], [a])$  and  $\dot{a}([\rho], [a])$  for the pentagon-wheel  $\gamma_5$ -flow over  $\mathbb{R}^3$  by using five Civita symbols. Does the extra symmetry persist for the marker-monomials in either velocity?

**Open problem 3** ( $\vec{X}$  for  $\gamma_3$ -flow on  $\mathbb{R}^4$  with  $\rho \equiv 1$ ). Inspect whether the restriction of the tetrahedral  $\gamma_3$ -flow to the space of Nambu-determinant Poisson structures  $P([\mathbf{a}])$  on  $\mathbb{R}^4$  with  $\rho \equiv 1$  is trivial in the second Poisson cohomology. — Let us presume that there exists a trivializing vector field  $\vec{X}([\mathbf{a}])$  with differential-polynomial coefficients. If it actually does, represent the coefficients – of possibly another vector field from the coset  $\vec{X} \bmod \llbracket P, \cdot \rrbracket$  – by using three Civita symbols on  $\mathbb{R}^4$ . Do the marker-monomials in  $\vec{Y}([\mathbf{a}])$  enjoy the extra symmetry?

**Open problem 4** ( $\vec{X}$  for  $\gamma_3$ -flow on  $\mathbb{R}^4$  with  $\rho \neq 1$ ). Extend and solve Problem 3 in the general case  $\rho \neq 1$  on  $\mathbb{R}^4$ , now for the trivializing vector field  $\vec{X}([\rho], [\mathbf{a}])$ .

**Open problem 5** ( $\vec{X}$  for  $\gamma_5$ -flow on  $\mathbb{R}^3$ ). Solve the trivialization problem – fully analogous to the above Problems 3–4 – for the pentagon-wheel  $\gamma_5$ -flow over  $\mathbb{R}^3$ . If it exists,

the trivializing vector field  $\vec{Y}([\varrho], [a])$  from the coset  $\vec{X} \bmod \llbracket P, \cdot \rrbracket$  (defined modulo Hamiltonian vector fields) will again be realizable by using five Civita symbols on  $\mathbb{R}^3$ .

**Open problem 6.** Can the trivializing vector fields  $\vec{X}([\varrho], [\mathbf{a}])$  be constructed – for nontrivial graph cocycles  $\gamma$  – and induce the graph flows  $\dot{P} = \text{Or}(\gamma)(P^{\otimes n})$  on the spaces of Nambu-determinant Poisson brackets  $P(\varrho, [\mathbf{a}])$  directly from the graph cocycles  $\gamma$  on  $n$  vertices and from the properties of the particular Poisson geometry of the Nambu brackets with global Casimirs? In other words, what are the marker-monomials for  $\vec{Y} \in \vec{X} \bmod \llbracket P, \cdot \rrbracket$  as differential-geometric objects? (Note that the knowledge of the vector field  $\vec{X}([\varrho], [\mathbf{a}])$  as the parent object for the Lie derivative  $L_{\vec{X}}$  and for  $\dot{P} = \llbracket P, \vec{X} \rrbracket$  is enough to calculate  $\dot{\mathbf{a}}$  and  $\dot{\varrho}$ .)

**Open problem 7.** Is there a relation between the (pseudo)group of diffeomorphisms generated by the highly nonlinear vector fields  $\vec{X}([\varrho], [\mathbf{a}])$  from the graph cocycle flows and, on the other hand, the (local) diffeomorphisms  $\mathbf{x} \rightleftharpoons \mathbf{x}'$  that map (by blowing up the local coordinates) the cells bounded by  $\varrho(\mathbf{x}) = 0$  in  $\mathbb{R}^d$  to the domains on which  $\varrho'(\mathbf{x}') \equiv \pm 1$ ?

In conclusion, we note that whenever they are Poisson-cohomology trivial (as we observe so far in all the cases), the nontrivial graph cocycle flows on the spaces of generalized Nambu-determinant Poisson brackets  $P(\rho, [\mathbf{a}])$  not only preserve the symplectic foliation (dictated by the Casimirs  $\mathbf{a}$ ) by merely reparametrizing the coordinate description of points still not anyhow displacing the symplectic leaves, but also preserve the tiling of the affine space  $\mathbb{R}^d$  with respect to the zero locus of the inverse density  $\varrho$  in  $P(\varrho, [\mathbf{a}])$ . Both the foliation and tiling are thus rigid under the graph cocycle flows.

APPENDIX A. A CLASS OF (NON)POLYNOMIAL POISSON BRACKETS ON  $\mathbb{R}^d$  WITHOUT GLOBAL POLYNOMIAL CASIMIR

First, let us recall a particular construction of homogeneous polynomial-coefficient Poisson brackets on  $\mathbb{R}^d$  with Cartesian coordinates  $x^1, \dots, x^d$ .

Denote by  $\vec{E}$  the Euler vector field,  $\vec{E} = \sum_i x^i \cdot \partial/\partial x^i$ , and consider another nonzero vector field  $\vec{V} = \sum_j V^j(x^1, \dots, x^d) \cdot \partial/\partial x^j$  with homogeneous polynomial coefficients  $V^j$  of total degree  $k \gg 1$  (conveniently starting at  $k = 2$ ). This homogeneity assumption implies that  $\vec{E}(\vec{V}) = k \cdot \vec{V}$  and  $\vec{V}(\vec{E}) = 1 \cdot \vec{V}$ , whence  $[\vec{E}, \vec{V}] = (k - 1) \cdot \vec{V}$ .

By definition, put  $P := \vec{V} \wedge \vec{E}$ ; this is a bi-vector with homogeneous-polynomial coefficients (of degree  $k + 1$ ).

**Lemma 8.** All such bi-vectors  $P = \vec{V} \wedge \vec{E}$  on  $\mathbb{R}^d$  are Poisson.

*Proof.* Let us calculate the Schouten bracket  $\llbracket P, P \rrbracket = \llbracket \vec{V} \wedge \vec{E}, \vec{V} \wedge \vec{E} \rrbracket$  by using its inductive definition for decomposable multi-vectors and thus, reducing it to the calculation of commutators for 1-vector fields:

$$\begin{aligned} \llbracket \vec{V} \wedge \vec{E}, \vec{V} \wedge \vec{E} \rrbracket &= \vec{V} \wedge [\vec{E}, \vec{V}] \wedge \vec{E} - \vec{V} \wedge [\vec{E}, \vec{E}] \wedge \vec{V} - \vec{E} \wedge [\vec{V}, \vec{V}] \wedge \vec{E} + \vec{E} \wedge [\vec{V}, \vec{E}] \wedge \vec{V} \\ &= 2\vec{V} \wedge [\vec{E}, \vec{V}] \wedge \vec{E} = 2(k - 1) \cdot \vec{V} \wedge \vec{V} \wedge \vec{E} \equiv 0. \end{aligned}$$

This proves that the Jacobi identity  $\frac{1}{2}\llbracket P, P \rrbracket = 0$  holds, so  $P$  is Poisson. □

*Remark 4.* The above construction of Poisson bi-vectors  $P = \vec{V} \wedge \vec{E}$  naturally extends to homogeneous vector fields  $\vec{V}$  (possibly not on the entire  $\mathbb{R}^d$ ) with not necessarily polynomial coefficients but still satisfying the condition  $\vec{E}(\vec{V}) = \lambda \cdot \vec{V}$  with  $\lambda \neq 0, 1$ .

We now claim that not all of these Poisson bi-vectors  $P = \vec{V} \wedge \vec{E}$  on  $\mathbb{R}^d$  are Nambu-determinant type. Specifically, let us produce a family of such Poisson bi-vectors  $P = \vec{V} \wedge \vec{E}$  (with polynomial coefficients) which do not admit any global non-constant polynomial Casimirs — and this is in contrast with the Nambu class  $P = \varrho(\mathbf{x}) \cdot d\mathbf{a}/d\mathbf{x}$  for polynomial parameters  $\mathbf{a} = (a_1, \dots, a_{d-2})$ .

Indeed, suppose that there is a polynomial Casimir  $a$  for  $P = \vec{V} \wedge \vec{E}$  as above.<sup>5</sup> By the definition of Casimir, we have that

$$[[P, a]] = [[\vec{V} \wedge \vec{E}, a]] = \vec{V} \cdot \vec{E}(a) - \vec{E} \cdot \vec{V}(a) = 0,$$

whence we obtain the system of PDE: for each  $i$  running from 1 to  $d$ , the Casimir  $a$  satisfies the equation

$$V^i \cdot \sum_j x^j \cdot \partial a / \partial x^j = x^i \cdot \sum_j V^j \cdot \partial a / \partial x^j.$$

An infinite family of counterexamples is now produced by taking the vector fields  $\vec{V}$  with coefficients  $V^i := (x^i)^k$  for  $k \geq 2$ . Indeed, we obtain that

$$(x^i)^{k-1} \cdot \sum_\ell x^\ell \cdot \partial a / \partial x^\ell = \sum_j (x^j)^k \cdot \partial a / \partial x^j,$$

and the Casimir  $a$  is by assumption polynomial in all  $x^{j'}$  for  $j' \neq i$  in particular. With respect to every  $x^{j'}$  at  $j' \neq i$  for a fixed  $i$ ,  $1 \leq i \leq d$ , the degree of the left-hand side, viewed as a polynomial in  $x^{j'}$ , is strictly not equal to that degree of the right-hand side (as  $k > 1$ ) unless  $\partial a / \partial x^{j'} \equiv 0$  for all  $j' \neq i$ . Cycling over all the equations indexed by  $i$  in the system, we conclude that every polynomial Casimir  $a$  for the Poisson bi-vector  $P = \vec{V} \wedge \vec{E}$  with  $V^i = (x^i)^k$  is a constant over  $\mathbb{R}^d$ . (For the Nambu-determinant brackets  $d\mathbf{a}/d\text{vol}(\mathbf{x})$  this means that the bi-vector vanishes identically on  $\mathbb{R}^d$ .)  $\square$

*Acknowledgements.* The first and last authors thank M. Kontsevich for helpful advice and discussions. The research of R.B. is supported by project 5020 at the Institute of Mathematics, Johannes Gutenberg–Universität Mainz and by CRC-326 grant GAUS “Geometry and Arithmetic of Uniformized Structures”. The travel of A.K. was partially supported by project 135110 at the Bernoulli Institute, University of Groningen. R. Buring and A. Kiselev are grateful to the IHÉS for hospitality and financial support.

#### REFERENCES

- [1] Banks P., Panzer E., Pym B. (2020) Multiple zeta values in deformation quantization, *Invent. Math.* **222**:1, 79–159. (*Preprint arXiv:1812.11649 [math.QA]*)
- [2] Bouisaghouane A., Buring R., Kiselev A. V. (2017) The Kontsevich tetrahedral flow revisited, *J. Geom. Phys.* **119**, 272–285. (*Preprint arXiv:1608.01710 [q-alg]*).

<sup>5</sup> From the homogeneity of objects it is readily seen that if  $a$  is polynomial, then it is homogeneous of some degree  $D$  (for the homogeneous brackets from the Nambu class,  $D$  equals  $1 + \deg(P^{ij})$ :  $\vec{E}(a) = D \cdot a$ ).

- [3] *Buring R.* (2022) The action of Kontsevich's graph complex on Poisson structures and star products: an implementation. PhD dissertation, Johannes Gutenberg-Universität Mainz, in preparation.
- [4] *Buring R., Kiselev A. V.* (2019) The expansion  $\star \bmod \bar{o}(\hbar^4)$  and computer-assisted proof schemes in the Kontsevich deformation quantization, *Experimental Math.*, 54 p. (doi:10.1080/10586458.2019.1680463, in press). (Preprint arXiv:1702.00681 [math.CO])
- [5] *Buring R., Kiselev A. V.* (2019) The orientation morphism: from graph cocycles to deformations of Poisson structures, *J. Phys.: Conf. Ser.* **1194** Proc. 32nd Int. colloquium on Group-theoretical methods in Physics: GROUP32 (9–13 July 2018, CVUT Prague, Czech Republic), Paper 012017, 10 p. (Preprint arXiv:1811.07878 [math.CO])
- [6] *Buring R., Kiselev A. V.* (2020) Universal cocycles and the graph complex action on homogeneous Poisson brackets by diffeomorphisms, *Physics of Particles and Nuclei Letters* **17**:5 Supersymmetry and Quantum Symmetries 2019, 707–713. (Preprint arXiv:1912.12664 [math.SG])
- [7] *Buring R., Kiselev A. V., Rutten N. J.* (2017) The heptagon-wheel cocycle in the Kontsevich graph complex, *J. Nonlin. Math. Phys.* **24**:Suppl. 1 'Local & Nonlocal Symmetries in Mathematical Physics', 157–173. (Preprint arXiv:1710.00658 [math.CO])
- [8] *Buring R., Kiselev A. V., Rutten N. J.* (2018) Infinitesimal deformations of Poisson bi-vectors using the Kontsevich graph calculus, *J. Phys.: Conf. Ser.* **965** Proc. XXV Int. conf. 'Integrable Systems & Quantum Symmetries' (6–10 June 2017, CVUT Prague, Czech Republic), Paper 012010, 1–12. (Preprint arXiv:1710.02405 [math.CO])
- [9] *Buring R., Kiselev A. V., Rutten N. J.* (2018) Poisson brackets symmetry from the pentagon-wheel cocycle in the graph complex, *Physics of Particles and Nuclei* **49**:5 Supersymmetry and Quantum Symmetries 2017, 924–928. (Preprint arXiv:1712.05259 [math-ph])
- [10] *Donin J.* (1998) On the quantization of quadratic Poisson brackets on a polynomial algebra of four variables, in: Lie groups and Lie algebras, *Math. Appl.* **433**, Kluwer Acad. Publ., Dordrecht, 17–25.
- [11] *Grabowski J., Marmo G., Perelomov A. M.* (1993) Poisson structures: towards a classification, *Modern Phys. Lett.* **A8**:18, 1719–1733.
- [12] *Kiselev A. V.* (2019) Open problems in the Kontsevich graph construction of Poisson bracket symmetries, *J. Phys.: Conf. Ser.* **1416** Proc. XXVI Int. conf. 'Integrable Systems & Quantum Symmetries' (8–12 July 2019, CVUT Prague, Czech Republic), Paper 012018, 8 p. (Preprint arXiv:1910.05844 [math-ph])
- [13] *Kiselev A. V., Buring R.* (2021) The Kontsevich graph orientation morphism revisited. *Banach Center Publ.* **123** Homotopy algebras, deformation theory & quantization, 123–139. (Preprint arXiv:1904.13293 [math.CO])
- [14] *Kontsevich M.* (1997) Formality conjecture. Deformation theory and symplectic geometry (Ascona 1996, D. Sternheimer, J. Rawnsley and S. Gutt, eds), *Math. Phys. Stud.* **20**, Kluwer Acad. Publ., Dordrecht, 139–156.

- [15] *Kontsevich M.* (1999) Operads and motives in deformation quantization. Moshé Flato (1937–1998). *Lett. Math. Phys.* **48**:1, 35–72. (*Preprint arXiv:q-alg/9904055*)
- [16] *Kontsevich M.* (2001) Deformation quantization of algebraic varieties. EuroConference Moshé Flato 2000, Part III (Dijon). *Lett. Math. Phys.* **56**:3, 271–294. (*Preprint arXiv:math.AG/0106006*)
- [17] *Kontsevich M.* (2003) Deformation quantization of Poisson manifolds, *Lett. Math. Phys.* **66**:3, 157–216. (*Preprint arXiv:q-alg/9709040*)
- [18] *Kontsevich M.* (2008) Hodge structures in non-commutative geometry (Notes by E. Lupercio), in: XI Solomon Lefschetz Memorial Lecture series. Contemp. Math. **462** Non-commutative geometry in mathematics and physics, AMS, Providence RI, 1–21. (*Preprint arXiv:math.AG/0801.4760*)
- [19] *Kontsevich M.* (2019) Derived Grothendieck–Teichmüller group and graph complexes [after T. Willwacher]. *Séminaire Bourbaki*, Vol. 2016/2017. Exposés 1120–1135. Astérisque **407**, No. 1126, 183–211.
- [20] *Kontsevich M.* (2019) Private communication, 13 December 2019, IHÉS.
- [21] *Marvan M.* (2009) Sufficient set of integrability conditions of an orthonomic system, *Foundat. Comput. Math.* **9**, 651–674.
- [22] *Nambu Y.* (1973) Generalized Hamiltonian dynamics, *Phys. Rev.* **D7**, 2405–2412.
- [23] *Willwacher T.* (2015) M. Kontsevich’s graph complex and the Grothendieck–Teichmüller Lie algebra, *Invent. Math.* **200**:3, 671–760. (*Preprint arXiv:1009.1654 [q-alg]*)











# Curriculum Vitae

Ricardo Buring was born in Groningen, the Netherlands on the 30th of May in 1992. He received his B.Sc. diploma in Mathematics from the University of Groningen in 2013, and his M.Sc. diploma in Mathematics from the same university in 2017. In 2017 he moved to Johannes Gutenberg-University of Mainz in Germany for his Ph.D. studies in the group led by Prof. dr. D. van Straten, supervised by dr. hab. A.V. Kiselev (Groningen) and Prof. dr. D. van Straten (Mainz).

During his master's and doctoral studies Ricardo Buring participated in eight international conferences, gave talks at five colloquia internationally, and spoke six times at research seminars (in Mainz and abroad). He spent a week at the IHÉS in Bures-sur-Yvette, France, in April 2017 and another week in December 2019, discussing his work with M. Kontsevich. Ricardo Buring was selected to participate in the 7th Heidelberg Laureate Forum (2019), and further selected as one of 10 participants to be interviewed.<sup>1</sup>

Ricardo Buring is a coauthor of ten publications in peer-reviewed journals and one preprint, jointly with A.V. Kiselev and other collaborators (A. Bouisaghoulane, N.J. Rutten, and D. Lipper). He is the author of the MIT-licensed `kontsevich_graph_series-cpp` and `gcaops` software packages, which have been used to obtain results contained in the aforementioned publications.

By using this software, in the Autumn semester 2020/21 he contributed 15 tutorials and two PC demo sessions to a master's course “Deformation Quantization, Graph Complex, and Number Theory”, read by A.V. Kiselev, in the Dutch national MASTERMATH programme. During the four year Ph.D. term in Mainz, Ricardo Buring taught tutorials in undergraduate courses Computeralgebra, Discrete Mathematics for Computer Scientists, and “Geometry, Algebra, and Number Theory”. Ricardo Buring co-supervised two bachelor projects (N.J. Rutten in 2017/18, D. Lipper in 2020/21), and helped—e.g. by using software—with master theses of A. Bouisaghoulane (2016/17) and Willem de Kok (2020/21), and B.Sc. work of S. Kerkhove (2020).

Currently, Ricardo Buring is a research associate at the Institute of Mathematics, Johannes Gutenberg-University Mainz.

---

<sup>1</sup><https://scilogs.spektrum.de/hlf/10-out-of-200-from-diagrams-to-formulas-via-computers-ricardo-buring-loves-teaching-math/>



# Acknowledgements

First I express my thanks to Duco van Straten, who made this Ph.D. thesis possible. Equally, I thank my supervisor Arthemy Kiselev, for his continued guidance and his seemingly inexhaustible stream of good ideas, practical advice, and relevant stories.

I am grateful to all the members of the examination committee, namely [REDACTED], [REDACTED], [REDACTED], [REDACTED], [REDACTED], Duco van Straten and Arthemy Kiselev, for their attention, time, criticism, and feedback. Thanks to [REDACTED] for writing the minutes at my doctoral examination. Special thanks to [REDACTED] for asking particularly good questions during my defense.

Thanks to [REDACTED] for creating this entire theory, for helpful discussions, for suggesting areas of research, and for foretelling answers before we knew how to tackle the problem. [REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

Many thanks to my colleagues and friends in Mainz for the great times we spent together; thank you in particular Tom, Matthias, Raymond, Laura and Markus, Philipp, Nutsa, and Lina. Not least, I am deeply grateful to my friends and my family for their patience and kindness.

Mainz, July 2022



# Zusammenfassung

Poisson-Klammern treten auf, wenn das punktweise Produkt von Skalarfunktionen auf einer affinen Mannigfaltigkeit so deformiert wird, dass es assoziativ bleibt. Kontsevich bewies das Gegenteil: eine universelle Formel ordnet jeder Poisson-Klammer eine solche assoziative Deformation zu. Ebenso können Poisson-Klammern durch universelle Formeln deformiert werden. In beiden Konstruktionen werden die universellen Formeln mit Hilfe von Graphen gebildet.

Um die Tausende von Graphen zu handhaben, entwickeln und präsentieren wir das Softwarepaket `gcaops` (*Graph Complex Action on Poisson Structures*) für SageMath. Mit diesem Paket, • entwickeln wir Kontsevich's  $\star$ -Produkt bis auf  $\bar{o}(\hbar^4)$ ; • wir setzen  $\star \bmod \bar{o}(\hbar^6)$  aus externen Daten von Banks–Panzer–Pym zusammen, und wir erhalten das Sternprodukt  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$  für affine Poisson-Klammern; • wir verifizieren dass die von Banks–Panzer–Pym gefundene Graphgewichte viele bekannte Gleichungen erfüllen; • wir illustrieren den expliziten Beweis der Assoziativität für das vollständige Sternprodukt modulo  $\bar{o}(\hbar^6)$  und für das affine Sternprodukt modulo  $\bar{o}(\hbar^7)$ ; • wir finden neue explizite Formeln für Graphencozyklen und universelle Poisson-Cozyklen, und • wir beweisen die Faktorisierung der Poisson-Cozyklus-Bedingung über die Jacobi-Identität in jedem Fall.



# Samenvatting

Poisson-haakjes duiken op wanneer het puntsgewijze product van scalaire functies op een affiene variëteit zo gedeformeerd wordt dat het associatief blijft. Kontsevich bewees het omgekeerde: een universele formule wijst aan elk Poisson-haakje zo'n associatieve deformatie toe. Op een soortgelijke manier kunnen Poisson-haakjes zelf door universele formules gedeformeerd worden. In beide constructies worden de universele formules opgesteld met behulp van grafen.

Om de duizenden grafen te handhaven, ontwikkelen en presenteren we het softwarepakket `gcaops` (*Graph Complex Action on Poisson Structures*) voor SageMath. Met behulp van dit pakket, • ontwikkelen we Kontsevich's  $\star$ -product tot op  $\bar{o}(\hbar^4)$ ; • we bouwen  $\star \bmod \bar{o}(\hbar^6)$  met externe data van Banks–Panzer–Pym, en we verkrijgen het sterproduct  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$  voor affiene Poisson-haakjes; • we bevestigen dat de gewichten van grafen gevonden door Banks–Panzer–Pym aan vele bekende vergelijkingen voldoen; • we illustreren het expliciete bewijs van de associativiteit voor het volledige sterproduct modulo  $\bar{o}(\hbar^6)$  en voor het affiene sterproduct modulo  $\bar{o}(\hbar^7)$ ; • we vinden nieuwe expliciete formules voor graaf-cocykels en universele Poisson-cocykels, en • we bewijzen de factorisatie van de Poisson-cocykel conditie via de Jacobi-identiteit in elk afzonderlijk geval.



# Summary for Laymen

This is a dissertation in fundamental mathematics research. The area of mathematics under study is Kontsevich’s deformation quantization of Poisson structures and the related universal flows on spaces of Poisson structures. These topics are distinguished by their involvement of graphs (i.e. objects consisting of vertices and edges, visualized as dots and lines respectively) in a place of mathematics where they were previously not expected: graphs are used to construct formulas. This domain of mathematics has received a lot of attention in the form of abstract theory building, but examples have so far remained scarce. Part of the reason for the lack of examples is that producing them requires computational power and programming effort. We believe that finally having a way to easily generate those examples would benefit not only those who first come in contact with the material, but also the experts. Experimentally observed properties can—and actually do—lead to the recognition of patterns and forming new conjectures; old conjectures can be checked in particular cases. In this dissertation we develop the **gcaops** software (*Graph Complex Action on Poisson Structures*), package for **SageMath**, that constructs the needed examples and verifies some conjectures.

Let us discuss the mathematical component of this result, now produced by using the new software. The Kontsevich  $\star$ -product is a formula that looks as follows:

$$\begin{aligned}
 f \star g = & f \cdot g + \hbar(P^{ij} \cdot \partial_i f \cdot \partial_j g) + \hbar^2\left(\frac{1}{2}P^{ij} \cdot P^{kl} \cdot \partial_k \partial_i f \cdot \partial_l \partial_j g + \frac{1}{3}\partial_l P^{ij} \cdot P^{kl} \cdot \partial_k \partial_i \cdot f \partial_j g \right. \\
 & - \frac{1}{3}\partial_l P^{ij} \cdot P^{kl} \cdot \partial_i f \partial_k \cdot \partial_j g - \frac{1}{6}\partial_l P^{ij} \cdot \partial_j P^{kl} \cdot \partial_i f \cdot \partial_k g) + \hbar^3\left(\frac{1}{6}P^{ij} \cdot P^{kl} \cdot P^{mn} \cdot \partial_m \partial_k \partial_i f \cdot \partial_n \partial_l \partial_j g \right. \\
 & - \frac{1}{6}\partial_m \partial_l P^{ij} \cdot \partial_n \partial_j P^{kl} \cdot P^{mn} \cdot \partial_i f \cdot \partial_k g - \frac{1}{6}P^{ij} \cdot \partial_n P^{kl} \cdot \partial_l P^{mn} \cdot \partial_k \partial_i f \cdot \partial_m \partial_j g \\
 & - \frac{1}{6}\partial_m \partial_l P^{ij} \cdot \partial_n P^{kl} \cdot P^{mn} \cdot \partial_k \partial_i f \cdot \partial_j g - \frac{1}{6}\partial_m \partial_l P^{ij} \cdot \partial_n P^{kl} \cdot P^{mn} \cdot \partial_i f \cdot \partial_k \partial_j g \\
 & + \frac{1}{6}\partial_n \partial_l P^{ij} \cdot P^{kl} \cdot P^{mn} \cdot \partial_m \partial_k \partial_i f \cdot \partial_j g + \frac{1}{6}\partial_n \partial_l P^{ij} \cdot P^{kl} \cdot P^{mn} \cdot \partial_i f \cdot \partial_m \partial_k \partial_j g \\
 & + \frac{1}{3}\partial_n P^{ij} \cdot P^{kl} \cdot P^{mn} \cdot \partial_m \partial_k \partial_i f \cdot \partial_l \partial_j g - \frac{1}{3}\partial_n P^{ij} \cdot P^{kl} \cdot P^{mn} \cdot \partial_k \partial_i f \cdot \partial_m \partial_l \partial_j g \\
 & - \frac{1}{6}\partial_l P^{ij} \cdot \partial_n \partial_j P^{kl} \cdot P^{mn} \cdot \partial_m \partial_i f \cdot \partial_k g + \frac{1}{6}\partial_n \partial_l P^{ij} \cdot \partial_j P^{kl} \cdot P^{mn} \cdot \partial_i f \cdot \partial_m \partial_k g \\
 & \left. - \frac{1}{6}\partial_n P^{ij} \cdot P^{kl} \partial_l \cdot P^{mn} \cdot \partial_k \partial_i f \cdot \partial_m \partial_j g - \frac{1}{6}\partial_l P^{ij} \cdot \partial_n P^{kl} \cdot P^{mn} \cdot \partial_k \partial_i f \cdot \partial_m \partial_j g) + \bar{o}(\hbar^3).
 \end{aligned}$$

Here the inputs  $f, g$  are scalar functions on  $\mathbb{R}^n$ , the  $P^{ij}$  are the function coefficients of a Poisson structure and the sum over powers of the variable  $\hbar$  extends infinitely (i.e. it is a power series). The  $\star$ -product deforms the ordinary (pointwise) product of functions  $(f \cdot g)(x) = f(x) \cdot g(x)$  in such a way that the associativity  $(f \star g) \star h = f \star (g \star h)$  is preserved. If the  $\star$ -product is known up to a certain order, then its associativity is guaranteed up to the same order. We find Kontsevich’s star-product up to the order 4, by using the method of undetermined coefficients (for the coefficients of graphs) and finding many (new) relations between these weights. Here we also use the Shoikhet–Felder–Willwacher cyclic weight relations. We verify the associativity of Kontsevich’s  $\star$ -product up to the order 4, by expanding the associator  $(f \star g) \star h - f \star (g \star h) \bmod \bar{o}(\hbar^4)$  in terms of graphs

and collecting those sums of graphs which are known to be zero by the Jacobi identity for the Poisson bracket  $P$ . In this context we also verify that the graph weights found by Banks–Panzer–Pym (2018) satisfy many relations.

Secondly we investigated the Kontsevich universal graph flows on the spaces of Poisson structures. A Poisson structure  $P$  on  $\mathbb{R}^n$  can be considered as an  $n \times n$  skew-symmetric matrix of functions satisfying a system of partial differential equations  $\llbracket P, P \rrbracket = 0$ . This is the master equation, given by the Schouten bracket of bi-vectors. To deform a Poisson structure means to add a power series in  $\varepsilon$  with a leading deformation term  $Q$ , that is  $P \mapsto P + \varepsilon Q + \bar{o}(\varepsilon)$ ; the linearity and graded symmetry of the Schouten bracket yields  $\llbracket P + \varepsilon Q + \bar{o}(\varepsilon), P + \varepsilon Q + \bar{o}(\varepsilon) \rrbracket = \llbracket P, P \rrbracket + 2\varepsilon \llbracket P, Q \rrbracket + \bar{o}(\varepsilon)$ , hence the condition on the leading deformation term  $Q$  is  $\llbracket P, Q \rrbracket = 0$ , which means for  $Q$  to be a 2-cocycle in the Poisson complex of  $P$ . Kontsevich (1996) wrote universal formulas mapping Poisson structures to their deformations:  $P \mapsto Q(P)$  where  $\llbracket P, Q(P) \rrbracket = 0$ . To illustrate them, we produce explicit formulas and weighted graph encodings of new examples of flows. We examine the Poisson-(non)triviality of the tetrahedral flow for particular Poisson structures  $P$ , i.e. we inspect whether  $Q(P)$  can be expressed as  $\llbracket P, X \rrbracket$  for a vector field  $X$  (in coordinates, a column vector with functions as coefficients). For this we use different classes of Poisson structures of different origin (e.g. quadratic and cubic brackets from Li–Parmentier, as well as brackets with differential polynomial coefficients from Nambu). In all cases considered, the flow was Poisson-trivial. On the other hand, there is no known mechanism for it to be universally trivial. We investigate in detail what happens in dimension two ( $n = 2$ ). There all the flows considered are trivial, with their trivializing vector field realized in terms of Kontsevich’s directed graphs. To deform  $\star \bmod \bar{o}(\hbar^4)$  exactly at  $\hbar^4$  by using the tetrahedral flow, we found no mechanism for it to be universally trivial with respect to gauge transformations of star products. The produced formulas for Poisson 2-cocycles may be of interest to the larger Poisson geometry community.

Thirdly we calculated graph cohomology in several vertex-edge bi-gradings, and we found new explicit examples of graph cocycles. In particular, we expressed the heptagon-wheel cocycle and we calculated the bracket  $[\gamma_3, \gamma_5]$  of two previously known cocycles. This result can be used to produce graph flows for Poisson structures. Independently, this result is useful for the study of the graded Lie algebra of graphs. Indeed, it is easier to study with examples than without. Also, we recall that Willwacher established a bridge between the Grothendieck–Teichmüller Lie algebra **grt** and the Kontsevich graph complex **GC**, but only *one* example of this transition (for the tetrahedron) is available from all the previous work. This dissertation provides handy examples of what one has on the graph side of the bridge.

We hope that in the future the experimental facts and the revealed properties will be explained theoretically. Likewise, this research allows us to pose new problems.

The results in this dissertation were obtained using diverse methods. First it was necessary to learn the theory and to implement it accurately. When an algorithm was designed and the computer program ready, it was run. The output and results are reported in this dissertation: e.g. tables with the count of graphs, allowing one to evaluate the size of the problem. We used the method of undetermined coefficients whenever appropriate. That is, in the beginning we preview the use of all (in hindsight, too many) potentially needed structures, form equations, and solve them, so that new solutions collapse to low dimensional subspaces. (Examples of solutions which are small are graph cocycles and factorizations via Leibniz graphs.) Next, we systematically try the theory on examples and detect patterns; this is specific in particular to the graph flows. Naturally,

we practice strict rigorous mathematical proof of identities and (graph) equalities, e.g. for the Nambu–Poisson brackets.

To understand the mechanism at work in the algebraic theory of star-products described using graphs and in the geometric and algebraic theory of flows, again described by using graphs, I had to learn many adjacent domains of mathematics, such as *(i)* superalgebra, superanalysis, and Lie superalgebras, *(ii)* differential graded Lie algebras and their  $L_\infty$ -morphisms, *(iii)* operads and endomorphisms of the space of multivector fields, and *(iv)* elements of Poisson geometry and Poisson cohomology. It is wonderful that all these different domains are brought together in the study of Kontsevich’s deformation quantization.

This dissertation allowed me to better learn and practice scientific communication: I went to conferences, listened to talks, chatted informally with experts, and so on. Particularly memorable to me are the conference GADEIS VIII in Larnaca, the Groenewold symposium in Groningen, the Oxford seminar, as well as the workshop in Banff, where people in the audience also had something to say. There could be expected and there indeed was much feedback. Besides, I enjoyed teaching, for I like answering questions. Let us keep in mind that that the Mastermath course tutorials contributed to Part I (Computer Demonstrations), amounting to 40% of this dissertation.

All the relevant feedback was incorporated in this dissertation, for which I am grateful to everyone in reference. This dissertation is now based on 10 peer-reviewed publications and one preprint, and on many conference talks. The new **gcaops** software is available from <https://github.com/rburing/gcaops> and external data files (which can be appreciated separately) are stored at <https://rburing.nl/gcaops>.





# Appendix A

## Introduction to SageMath

This supplementary chapter is based on a document originally written and developed by the dissertant for the first two exercise classes in Computeralgebra at JGU Mainz (SoSe 2018, 2019, 2020, 2021).

### 1. What to do with this document

This document consists of two parts:

- An introduction to SageMath,
- Miscellaneous topics (that you can consider if you are interested).

This document was written as a SageMath Jupyter notebook, containing both text (also  $\LaTeX$ ) and executable code cells. (See below.)

If you have opened this document as a notebook (the `.ipynb` file, available separately from the `.pdf` file), then you can **run** the cells by giving them focus (e.g. clicking on them) and pressing **Shift+Enter**.

(Note: Double-clicking on a text cell like this one opens its editor. Press Shift+Enter to save it and exit that editor.)

If you are viewing this document as a PDF then you should type the commands into a SageMath session (beware that copying symbols and whitespace from a PDF can be problematic) and run them there (or open the notebook instead).

To open this notebook

- using a local installation:
  - on Linux, run `sage -n jupyter`; on Windows, press Start and then Sage Jupyter notebook,
  - open your web browser and navigate to `http://localhost:8888` (if it was not done automatically),
  - press ‘Upload’, select the `.ipynb` file, and press Upload again,
  - click the notebook to open it.

- on [CoCalc](#):
  - create a project,
  - press ‘New’
  - click to upload the `.ipynb` file,
  - open it.

Have a look through this document, run the cells which have been prepared, and write your own code in the empty cells for the exercises.

## 2. SageMath

**SageMath** is a free open-source mathematics software system. It builds on top of existing open-source packages, such as [Singular](#), [PARI/GP](#), [GAP](#), and many more. In SageMath we can access their combined power through a common language, based on [Python 3](#) (since SageMath 9.0).

The SageMath language is just Python 3 with a *preprocessing* step to allow some “more mathematical” syntax:

```
[1]: 1/3
```

```
[1]: 1/3
```

```
[2]: 2^3
```

```
[2]: 8
```

The SageMath inputs and outputs above will come as no surprise to a pure mathematician.

- In SageMath, when you enter `1/3`, two objects (`1` and `3`) of type `Integer` are created, and the operation `/` of division of an `Integer` by an `Integer` is defined to be the appropriate rational (of type `Rational`), hence the result is `1/3`.

This is in contrast to Python 3:

- In Python 3, when you enter `1/3`, two objects (`1` and `3`) of type `int` are created, and the operation `/` of division of an `int` by an `int` is defined to be the floating point approximation (of type `float`), hence the result is `0.3333333333333333`.

The way SageMath works is that the parser replaces literal numbers such as `1` in your code (which would normally become an `int` object) by `Integer(1)`, so that an object of type `Integer` is created instead, on which operations are defined in the way mathematicians expect.

In the SageMath language, many pre-defined functions, symbols, and constants are already *imported* for your convenience, so you can access them immediately. We will see examples of this below. While knowledge about Python would be advantageous, it is not required for this guide.

## 3. Introduction to SageMath

By working through the examples and small exercises in this section, you will gradually learn the basics of SageMath. Ultimately you will be able to *write your own (simple) functions*. That is a very practical skill, as you will see. Good luck!

### 3.1. Calculations with integers

SageMath can be used as a calculator. Try to understand the results of the following calculations, which involve only integers (whole numbers). If something is not clear, see the next section.

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

#### 3.1.1. Exercise

Calculate  $3385 \bmod 2048$ .

[ ]:

#### 3.1.2. Exercise

Find the smallest prime with 20 digits.

[ ]:

## 3.2. Help with functions

By running the cells above you were calling *functions* such as `gcd`, by writing the name of the function, followed by opening and closing parentheses, with *arguments* (inputs to the function, separated by commas) in between the parentheses: e.g. `gcd(101,25)` calls `gcd` with arguments `101` and `25`.

SageMath contains a built-in help system for functions. To find out what a function does, enter its name followed by a question mark:

[ ]:

You will get a short description, a list of INPUT that the function accepts, what OUTPUT the function returns, and usually also some EXAMPLES. When you are finished reading,

you can close the documentation (sub)window in the notebook interface by pressing the `x` button on the right, and in the command line interface by pressing the `q` button.

How do you find out which functions exist in the first place? The [SageMath reference manual](#) contains documentation for (almost) all of Sage's features, and you can search through it. Also, the next section of this intro is a mainly a collection of functions that will be useful later in the course.

You can also start typing the name of a function, and press the TAB key for auto-completion:

```
[ ]: next_p
```

### 3.3. Variables

Defining and (re)assigning variables is done with the `=` sign.

```
[ ]: a = 3
```

```
[ ]: b = 4
```

Variables can be defined in terms of (the values of) other variables:

```
[ ]: c = sqrt(a^2 + b^2)
```

To retrieve the value of a variable, enter its name:

```
[ ]: c
```

Variables can be used in expressions:

```
[ ]: a^2 + b^2
```

Note that the following is valid code:

```
[ ]: x = 1
```

```
[ ]: x = x + 1
```

```
[ ]: x
```

In an assignment such as `x = x + 1`, the right-hand side is evaluated first, and then assigned to the variable in the left-hand side. Effectively, this statement increments the value of `x` by one. A shorthand notation that achieves the same effect is `x += 1`.

### 3.4. Methods

Each type of object (such as `Integer`) also has a bunch of *methods* associated to it, which can be called on all objects of that type.

```
[ ]: 5.factorial()
```

```
[ ]: 8.nth_root(3)
```

In general: enter (the name of) an object, followed by a dot (`.`), followed by the name of the method, followed by opening and closing parentheses (with the *arguments* of the method, if any, in between the parentheses).

**Warning:** To call a method, the parentheses (as above) are always required. It's possible to refer to a method by name (omitting the parentheses):

```
[ ]: 5.factorial
```

The output of the cell above just tells you that the method exists; it does not call the method, i.e. it does not tell you what the value of 5 factorial is.

### 3.5. Help with methods

For technical reasons, there is a (minor) limitation on the help system for getting help with methods.

It requires you to first define a variable of the desired type, and then you can get help with the methods which are available for that variable.

```
[ ]: z = 101
```

```
[ ]: z.factorial?
```

In the documentation of a method, the object on which it is called (`z` here) is referred to as `self`.

Given a variable, it is possible to list all its publicly available methods by typing its name, followed by a dot (`.`), and pressing the TAB key.

```
[ ]: z.
```

You can scroll through the list by using the arrow keys.

#### 3.5.1. Exercise

Find a method to obtain the list of decimal digits of an integer. (We will learn more about lists later.)

```
[ ]:
```

#### 3.5.2. Exercise

Find a method to obtain the list of binary digits of an integer. Hint: Check the documentation of the method you found before.

```
[ ]:
```

### 3.6. Functions versus methods

For convenience there exist some functions that (internally) call the respective methods:

```
[ ]: factorial(5)
```

This is just an alias for `5.factorial()`. You might prefer to write `factorial(5)`, e.g. because it looks better or because it is one keystroke shorter.

Similarly:

```
[ ]: is_prime(25)
```

**Warning:** It is not always the case that for every method there is a corresponding function that calls the method. Use the help system and TAB completion (as explained above) to discover functions and methods.

### 3.7. Logical expressions

Truth values in SageMath (as in Python) are called `True` and `False`; aliases are `true` and `false` respectively.

For equality testing in SageMath you use the double equals sign.

```
[ ]: 1 + 1 == 3
```

```
[ ]: 3^2 + 4^2 == 5^2
```

Inequalities are written in the usual way:

```
[ ]: 1 < 3
```

```
[ ]: 2 >= 3
```

For non-equality testing you use the `!=` operator:

```
[ ]: 1 != 2
```

More logical expressions:

```
[ ]: 5 % 2 == 0
```

```
[ ]: 5 % 2 == 1
```

```
[ ]: gcd(3, 12) == 1
```

```
[ ]: 25.is_prime()
```

```
[ ]: 9.is_square()
```

Several truth values can be combined using logical AND and logical OR.

```
[ ]: 8 % 2 == 0 and 8.is_square()
```

```
[ ]: 8 % 2 == 0 or 8.is_square()
```

#### 3.7.1. Exercise

Translate the (true) expression “3 is an odd prime number” into a line of code that evaluates to `True`.

[ ]:

### 3.8. Order of execution and multi-line programs

Clearly the order of execution is important:

[ ]: `x = 3`[ ]: `x = 4`[ ]: `x.is_prime()`

Here, the result of `x.is_prime()` depends on how `x` is defined, i.e. which definition of `x` was executed last.

So far we have only executed (cells containing) a single line of code at a time. We can use more than one line in a cell:

[ ]: `x = 3`  
`x.is_prime()`

The code is executed one line at a time, from top to bottom.

In a program with more than one line, the value of the last line (if it is not `None`) is displayed in the output (in the notebook interface).

### 3.9. Strings and printing

Another type of variable is `str`, for a string of characters, written between single or double quotation marks:

[ ]: `'Hello, World'`

The `print` function is used to output things to the screen:

[ ]: `print('Hello, World')`

Note the difference in the output of the above two code cells. In the first, the string value is displayed as the output value. In the second, the string is displayed but there is no output value (which would be marked by `Out[n]:` in the notebook interface). This is because `print` does the work of displaying the value, and then returns the value `None`, which is not displayed.

Here is another example, where there is output from `print`, and the value `True` is displayed because it is the value of the last line:

[ ]: `x = 3`  
`print('Is 3 a prime?')`  
`x.is_prime()`

Strings can also be stored in variables, and then used:

[ ]: `message = 'Hi'`  
`print(message)`

Passing several values to `print` outputs them in succession, with a space in between:

```
[ ]: print('Hi', 'there')
```

Variables of other types can also be printed:

```
[ ]: print(a,b,c)
```

```
[ ]: print('a =', a, 'and b =', b, 'and c =', c)
```

```
[ ]: print('Does a^2 + b^2 = c^2 hold?', a^2 + b^2 == c^2)
```

```
[ ]: print('Is c prime?', c.is_prime())
```

**Big Tip:** When using `print` to display the values of multiple variables, always make it clear which value belongs to which variable (e.g. as above).

It's also possible to define strings spanning multiple lines, by using triple quotes as delimiters:

```
[ ]: print(""" We're no strangers to love
You know the rules and so do I""")
```

```
[ ]: lyrics = """      Never gonna give you up
      Never gonna let you down
      Never gonna run around and desert you"""
print(lyrics)
```

You can build strings from variables using the `format` method of `str`:

```
[ ]: name = 'Ricardo'
age = 2022 - 1992
sentence = 'My name is {} and I am {} years old.'.format(name, age)
print(sentence)
```

Addition of strings is concatenation, and in this way we can also build strings dynamically:

```
[ ]: sentence = 'My name is ' + name + ' and I am ' + str(age) + ' years old.'
print(sentence)
```

Note that you cannot add `str` and `Integer` objects directly: you must convert the `Integer` to a `str` explicitly, using `str(...)`.

### 3.9.1. Exercise

Define an integer variable `k`, print it, and print some questions with answers about it: is it even, is it prime, is it a square?

```
[ ]:
```

## 3.10. Escape sequences

The backslash character `\` has a special meaning inside a string. It is used for *escape sequences* such as `\n` which is replaced by a newline, `\t` which is replaced by a tab, `\'` which is replaced by a literal quotation mark, and `\\` which is replaced by a literal backslash.



```
[ ]: print('Hello,\nWorld')
```

```
[ ]: print('I = R \ \ Q')
```

To write a *raw* string without escape sequences, write *r* before the opening quote sign.

```
[ ]: print(r'I = R \ Q')
```

### 3.11. Comments

Text written after a pound/hash/octothorpe sign *#* is ignored by the Python/SageMath interpreter (except when *#* is used inside a string).

```
[ ]: 1 + 2 + 3 + 4 + 5 # the 5th triangular number
```

Such text is called a *comment*. Comments are often used to explain *why* code is written the way it is. Comments that explain *what* code does are mostly useless:

```
[ ]: n = 3 # define n to be 3
     n += 1 # add 1 to n
     print(n) # print n to the screen
```

Those comments are unlikely to be helpful. Another use for comments is to “disable” a piece of code, without removing it entirely:

```
[ ]: n = 3
     #n += 1
     print(n)
```

This is called “commenting out” a piece of code.

To comment out multiple lines without writing a lot of *#* signs, we can surround the lines by triple quotes:

```
[ ]: x = 1
     y = 2
     z = 3
     """x += 1
     y += 1
     z += 1"""
     print(x, y, z)
```

This makes use of the fact that triple quoted strings on their own are valid statements that don’t do anything.

### 3.12. Conditionals

For  $n \in \mathbb{N}$ , the integer  $C_n$  is defined by  $n/2$  if  $n$  is even and by  $3n + 1$  if  $n$  is odd. In code, such a conditional can be expressed by an *if*-statement:

```
[ ]: n = 12
     if n % 2 == 0:
         C = n // 2
     else:
         C = 3*n + 1
```

```
print('C_{} = {}'.format(n, C))
```

The line starts with `if`, followed by the condition (a truth value, possibly depending on variables), followed by a colon (`:`). The next line(s), the *body* of the `if`-statement, are *indented* by four spaces. That means each such line starts with four spaces. The body can contain multiple lines (all indented). The body of the `if`-statement is evaluated only if the condition is true.

The `else:` (at the same level of indentation as the corresponding `if`) specifies what should be done if the condition is *not* true. Again, the body of the `else`-clause must be indented, and can contain multiple lines.

An `else`-clause is optional:

```
[ ]: n = 6
      print('A perfect number is', n)
      if n % 3 == 0:
          print("By the way, it's a threeven number")
```

It is also possible to switch between more than two alternatives, by adding

```
elif <other condition>:
    <other code>
```

after an `if`-statement.

### 3.12.1. Exercise

Define an integer variable `k` and print a string that says  $k$  is negative,  $k$  is zero, or  $k$  is positive (depending on the value of  $k$ ).

```
[ ]:
```

## 3.13. Range-based for loops

We can print the integers from 1 up to and including 10:

```
[ ]: for n in range(1,11):
      print(n)
```

Each line in the body of a `for`-loop is indented by four spaces. The body can contain multiple lines.

The `range(a,b)` consists of `[a, a+1, ..., b-1]` (excluding the endpoint). The notation `range(n)` is shorthand for `range(0,n)`.

We will see later why this convention (starting at 0 and excluding the endpoint) is convenient, e.g. in the context of lists.

The `range` function is from Python 3. In the loop above, `n` is a Python `int` object. To get a SageMath `Integer` we can use `srange`:

```
[ ]: for n in srange(1,11):
      print(n)
```

```
[ ]: for n in xrange(1,11):
      print('{}! = {}'.format(n, n.factorial()))
```

**Note:** The code `n.factorial()` does not work when `n` is a Python `int` (try it), but it does when `n` is a SageMath `Integer`.

If `n` is a Python `int`, then you can also write `Integer(n).factorial()`, i.e. first converting to a SageMath `Integer`.

### 3.13.1. Exercise

Print the squares of the first 10 non-negative integers.

```
[ ]:
```

## 3.14. Range-based for loops and conditionals

Which numbers  $y$  from 0 to 12 satisfy  $\gcd(y, 12) = 1$ ?

```
[ ]: for y in range(0,12):
      if gcd(y,12) == 1:
          print(y)
```

We will see later how we can store these in a list. (Now we can only look at them.)

### 3.14.1. Exercise

Print the numbers from 1 to 10 which are squarefree.

Hint: Use `m.is_squarefree()` for an `Integer` object `m` (in the body of a loop over `m`).

```
[ ]:
```

## 3.15. Range-based for loops and variables

Let's sum the first 100 positive integers:

```
[ ]: n = 100
      total = 0
      for k in range(1,n+1):
          total += k
      print('The sum of the first', n, 'positive integers is', total)
```

Here we used the `+=` operator to add `k` to `total`; this is equivalent to (but shorter than) `total = total + k`.

(Other shorthand notations are `-=`, `*=`, `%=`.)

Compare:

```
[ ]: n = 100
      n*(n+1)/2
```

### 3.15.1. Exercise

Define variables `n`, `oddsum`, `evensum`, and make a single `for`-loop (containing a conditional) so that, at the end `oddsum` is the sum of all odd integers from 1 to  $n$  (and similarly `evensum`). Add some nice print statements at the end.

[ ]:

## 3.16. While loops

Let's find the first square number greater than 200 (in a brute-force way).

[ ]:

```
n = 1
while n^2 < 200:
    n += 1
print(n, 'squared is', n^2)
print('Just to be sure:', n-1, 'squared is', (n-1)^2)
```

In a `while`-loop, the body (`n += 1` here) is repeatedly executed, as long as the condition (`n^2 < 200` here) is true.

**Warning:** Take care that your `while`-loops are written in such a way that they actually *terminate* in finite time.

To interrupt the execution of Sage code, press the “Stop” (black square) button in the notebook interface or press `Ctrl+C` on the command-line.

### 3.16.1. Exercise

Define a variable `n` and create a while loop to find the  $n$ th prime (e.g. in a naive way, trying every integer).

Hint: define a variable `k` which you increment (`k += 1`) with every iteration, use `is_prime()` to determine if you have a prime, and use another variable `prime_count` to keep track of how many primes have been seen so far. What should the `while`-condition be?

[ ]:

## 3.17. Break and continue

It's also possible to use the `break` statement in a loop (`for` or `while`) to exit the loop immediately (and proceed with the rest of the program).

[ ]:

```
n = 1
while True:
    if (n+1)^2 > 200:
        break
    n += 1
print(n, 'squared is', n^2)
```

You can also use the `continue` statement to skip the current iteration of a loop and continue with the next one.

```
[ ]: for n in range(10):
      if n % 3 == 0:
          continue
      print(n)
```

This is often useful to avoid “deep” indentation of the main body of your loop.

### 3.18. Lists

Lists can be defined by specifying their elements explicitly, between square brackets and separated by commas:

```
[ ]: Lst = [4,8,15,16,23,42]
      Lst
```

Or by a list comprehension:

```
[ ]: [n^2 for n in range(10)]
```

These may include a condition at the end:

```
[ ]: [a for a in range(12) if gcd(a,12) == 1]
```

Access individual elements:

```
[ ]: Lst[0]
```

```
[ ]: Lst[1]
```

```
[ ]: Lst[-1]
```

Test membership:

```
[ ]: 23 in Lst
```

Slicing:

```
[ ]: Lst[1:4]
```

```
[ ]: Lst[3:0:-1]
```

```
[ ]: Lst[-4:-1]
```

```
[ ]: Lst[::-1]
```

Lists may contain elements of different types, e.g. other lists.

```
[ ]: [ 1, 2, 3, [4,5], 6, True, None]
```

#### 3.18.1. Exercise

Create a list of even numbers up to some bound. Do the same for odd numbers. Then create a list containing both of these lists.

```
[ ]: 
```

### 3.19. Operations on lists

```
[ ]: len(Lst)
```

```
[ ]: sum(Lst)
```

```
[ ]: prod(Lst)
```

```
[ ]: sorted(Lst)
```

```
[ ]: [1,2,5] + [4,8,10,12]
```

```
[ ]: list(zip([0,1,2,3,4],[10,100,1000,10000,100000]))
```

### 3.20. Modifying lists

Lists can be modified.

```
[ ]: L = list(primes(20))  
L
```

```
[ ]: L[0] = 1  
L
```

```
[ ]: L[0] = 2  
L
```

```
[ ]: L.append(23)  
L
```

```
[ ]: L.pop(0)  
L
```

```
[ ]: L.remove(13)  
L
```

```
[ ]: L.insert(0, 2)  
L
```

#### 3.20.1. Exercise

Find out what else you can do with lists, by typing `L.` and pressing the **TAB** key.

```
[ ]: L.
```

To find out what a method does, enter its name followed by a question mark, as before. (The documentation of these Python methods is rather brief.)

### 3.21. Lists and loops

Let's build a list iteratively, using `append` in a range-based for loop.

```
[ ]: unitsmod12 = []
     for a in range(0,12):
         if gcd(a, 12) == 1:
             unitsmod12.append(a)
     print(unitsmod12)
```

You can loop over the elements of a list:

```
[ ]: L = [1,3,5,7,9]
     for x in L:
         print(x)
```

This is much more convenient than (but equivalent to) the index-based alternative:

```
[ ]: L = [1,3,5,7,9]
     for i in range(len(L)):
         print(L[i])
```

**Tip:** If you have written a range-based for loop where the index is *only* used to access elements of a list (as above), rewrite it as a loop over list elements to make it more readable.

There is also the following shorthand notation:

```
[ ]: for x,y in [ [1,2], [3,4] ]:
     print(x)
```

This is equivalent to:

```
[ ]: for P in [ [1,2], [3,4] ]:
     x,y = P
     print(x)
```

### 3.21.1. Exercise

Create a list of square numbers which are less than some (variable) bound.

```
[ ]:
```

## 3.22. Functions

A function can be defined as follows:

```
[ ]: def squared_plus_one(x):
     return x^2 + 1
```

The definition starts with **def**, followed by the name of the function, followed by the list of (input) arguments between parentheses, followed by a colon (:). The following lines (each indented by four spaces) form the *body* of the function. The body of a function can contain several statements. It is optional (but highly recommended) that a function *return* some result.

Calling a function executes the body, and the returned value is the result:

```
[ ]: squared_plus_one(3)
```

```
[ ]: squared_plus_one(5)
```

A function needs to be defined only once, and can then be called any number of times. (Re-defining a function overwrites the previous definition, just like it is with variables.)

Using functions is a great time saver. Instead of writing the code in the body of the function over and over again, we just write it once, and call it by its name.

You can turn any piece of code into a function by indenting it (adding four spaces in front of each line), giving it a name, specifying the arguments and adding a `return` statement. Recall this code from the section about lists:

```
[ ]: unitsmod12 = []
     for a in range(0,12):
         if gcd(a, 12) == 1:
             unitsmod12.append(a)
     print(unitsmod12)
```

Here it is as a function, now depending on the modulus `n` (which used to be the constant `12`):

```
[ ]: def units_mod(n):
     units = []
     for a in range(0,n):
         if gcd(a, n) == 1:
             units.append(a)
     return units
```

```
[ ]: units_mod(12)
```

**Big Tip:** If your function calculates something, *return* the result and don't just *print* it. If you do not return it, then you cannot use it in further calculations.

```
[ ]: list(reversed(units_mod(12)))
```

The above would not be possible if `units_mod` only printed the units instead of returning them.

For more information about printing inside functions, see Section [A](#) below.

### 3.22.1. Exercise

Define a function that returns the list of square numbers up to some bound (the bound should be the input). Name your function and its arguments appropriately.

Hint: Use the code you wrote in the exercise about lists and loops.

```
[ ]:
```

### 3.22.2. Exercise

Define another function, and call it.

```
[ ]:
```

```
[ ]:
```



## 4. Miscellaneous

This section contains other things which are nice to know.

### 4.1. Documenting your code

We have already seen the documentation of Sage's own functions, accessed using the question mark. We can also add documentation to our own functions. This is done by using a *docstring*, between triple quotes, starting at the first line in the definition of your function:

```
[ ]: def multiplicative_order_mod(a,n):
      """Return the multiplicative order of a mod n."""
      order = 1
      g = a % n
      while g != 1:
          g *= a          # shorthand for g = g * a
          g %= n         # shorthand for g = g % n
          order += 1
      return order
```

```
[ ]: multiplicative_order_mod(5,12)
```

The docstring can be accessed using the question mark:

```
[ ]: multiplicative_order_mod?
```

The docstrings in SageMath are written according to a format like this:

```
[ ]: def multiplicative_order_mod(a,n):
      r"""
      Return the multiplicative order of  $a$  mod  $n$ .

      INPUT:

      -  $a$  - integer
      -  $n$  - integer

      OUTPUT:

      - the multiplicative order of  $a$  mod  $n$ , i.e. the smallest
        positive integer  $k$  such that  $a^k = 1 \pmod n$ .

      ASSUMPTIONS:

      -  $a \pmod n$  is a unit, i.e.  $\gcd(a,n) = 1$ .

      EXAMPLES::

          sage: multiplicative_order_mod(5,12)
          2
          sage: multiplicative_order_mod(2,13)
          12
      """
      order = 1
```

```

g = a % n
while g != 1:
    g *= a          # shorthand for g = g * a
    g %= n         # shorthand for g = g % n
    order += 1
return order

```

```
[ ]: multiplicative_order_mod?
```

If you write a function which is interesting enough, then it is a good idea to add documentation like that.

```
[ ]:
```

## 4.2. Local variables

Typically, the behavior of a function should depend only on its arguments. Variables defined inside a function are called *local* to that function. These variables do not affect the values of variables with the same name which are defined outside the function.

```
[ ]: order = 'One pepperoni pizza, please.'
print('the multiplicative order of 3 mod 11 is', multiplicative_order_mod(3,11))
print(order)
```

The above works fine, even though `order` is also the name of a *local* variable inside the function `multiplicative_order_mod` (defined above).

That is the nice thing about functions which use local variables (and do not modify external variables): you can just use them without worrying about what happens inside them.

## 4.3. Benchmarking your code

Check how long your code takes by adding `%time` in front of a line:

```
[ ]: %time factor(randint(10^50, 10^51))
```

Or, by using `%timeit`, run the code several times and take the best time:

```
[ ]: %timeit factor(randint(10^50, 10^51))
```

## 4.4. Types and parents

We already mentioned in the very beginning that each object in SageMath has a *type* such as `Integer`, `Rational`, etc.

Most objects which are “elements” of some kind have a *parent* in SageMath.

```
[ ]: 5.parent()
```

The parent object `ZZ` represents the ring of integers  $\mathbb{Z}$ .

```
[ ]: ZZ
```

```
[ ]: 5 in ZZ
```

```
[ ]: 1/2 in ZZ
```

The parent of a polynomial is a polynomial ring.

```
[ ]: R.<x> = PolynomialRing(QQ)
      (x^2 + 1).parent()
```

```
[ ]: (x^2 + 1).parent() is R
```

The parent of a matrix is a matrix space.

```
[ ]: A = Matrix(QQ, [[1,1],[1,1]])
      A.parent()
```

```
[ ]: A.parent() is MatrixSpace(QQ, 2)
```

This system of “parents” makes e.g. the following `change_ring` functionality possible:

```
[ ]: B = A.change_ring(ZZ)
      B.parent()
```

```
[ ]: B.parent() is MatrixSpace(ZZ, 2)
```

See [Parents, Conversion and Coercion](#) in the Sage Tutorial if you are interested in the technical details.

## 4.5. L<sup>A</sup>T<sub>E</sub>X output

In the notebook interface, the `show` function displays an object using L<sup>A</sup>T<sub>E</sub>X in math mode, in a pretty way if possible. The `LatexExpr` function can be used to define custom LaTeX expressions.

```
[ ]: show(ZZ)
      show(QQ)
      show(LatexExpr(r'\LaTeX\text{ is }\LaTeX\text{, la la la la la.}'))
```

```
[ ]: R.<x> = QQ[]
      show(R)
      show(x^2 + 2)
      show(latex(x^2 + 2) + LatexExpr(r'\in') + latex(R))
```

```
[ ]: S.<x,y> = QQ[]
      show(S)
      show(x^2 + y^2 - 1)
      show(latex(x^2 + y^2 - 1) + LatexExpr(r'\in') + latex(S))
```

```
[ ]: A = Matrix(QQ, [[0,-1],[1,0]])
      show(A.parent())
      show(A)
      show(latex(A) + LatexExpr(r'\in') + latex(A.parent()))
```

## 4.6. Verbosity

In long computations it is sometimes useful to display intermediate results. This can of course be done using `print`. But sometimes you just want to use a function without seeing the intermediate results. How to get the best of both worlds? Use `verbose` and pass a `level`.

```
[ ]: def multiplicative_order_mod(a,n):
      order = 1
      g = a % n
      while g != 1:
          verbose("{}^{} = {}".format(a,order,g), level=2)
          g *= a          # shorthand for g = g * a
          g %= n          # shorthand for g = g % n
          order += 1
      return order
```

The function `verbose` prints its first argument only if the current verbosity level is at least `level`.

You can get the current verbosity level (default: `0`) with `get_verbose()` and set it with e.g. `set_verbose(100)`.

```
[ ]: get_verbose()
```

```
[ ]: set_verbose(100)
      multiplicative_order_mod(3,17)
```

```
[ ]: set_verbose(0)
      multiplicative_order_mod(3,17)
```

## 4.7. Copying lists

There is some subtlety involved in copying a list. Try to predict the output of the following code:

```
[ ]: L1 = [1,2,3]
      L2 = L1
      L1[0] = 5
      L2
```

What happens here is that `=` for lists is assignment by reference. Here's how to make a copy instead:

```
[ ]: L1 = [1,2,3]
      L2 = list(L1)
      L1[0] = 5
      L2
```

In case of e.g. nested lists even this is not enough, and you have to ensure that also the sub-objects are copied:

```
[ ]: from copy import deepcopy
      L1 = [ [0, 1], [-1, 0] ]
      L2 = deepcopy(L1)
```

```
L1[0][0] = 1
L2
```

Don't worry about this for now, but remember it if you run into trouble with lists.

## 4.8. Tuples

Tuples are like lists, but they cannot be modified and cannot contain modifiable elements.

```
[ ]: T = (1,2,3,4)
      T
```

```
[ ]: T[0]
```

```
[ ]: T[-1]
```

## 4.9. Accessing the source code

SageMath is free software. Its source code is freely available at e.g. <https://github.com/sagemath/sage>. In the notebook interface and on the command line, the source code of functions and methods can be accessed in a similar way to accessing the documentation, using `??` instead of `?`:

```
[ ]: n = 10
      n.factor??
```

At the end of the displayed source code, you find the name of the file where the method or function is defined. In the example above, it looks something like `~/src/SageMath-9.0/local/lib/python3.7/site-packages/sage/rings/integer.pyx`. You can use this path to find the file on your own machine or [on GitHub under `src/sage`](#). Continuing the example, that file would be `src/sage/rings/integer.pyx`. Seeing the whole file is useful e.g. if the source code uses some imported names (which you couldn't see by only looking at the function).

In the source code you will often find calls to external libraries, which are also open source, so you can continue your investigation by reading the source code of that library.

## 4.10. Error messages are your friends

Each cell in this section contains a mistake, and executing it will result in an error message in the output.

Don't be afraid, just try it! The error messages are intended to be informative and helpful (have a look at their **last line** first).

```
[ ]: (1+2
```

That was a syntax error, meaning the input was not well-formed. In particular, “unexpected EOF” (end-of-file) means that SageMath was expecting something more at the end of the input (i.e. the input was not complete). If you get this kind of error, check the balance of your parentheses and such things.

```
[ ]: 1/(1-2^0)
```

That was division by zero.

```
[ ]: 2 + ZZ
```

That error means that addition (the *operator* `+`) is not defined for the two objects (the *operands* of `+`) which we tried to add.

Usually for addition to make sense, the two operands should have a common parent object.

```
[ ]: numerical_approx(ZZ)
```

The code above tries to get a numerical approximation of  $\mathbb{Z}$ , which is nonsense because  $\mathbb{Z}$  is not a number.

```
[ ]: ZZ.n()
```

The code above tries to get a numerical approximation of  $\mathbb{Z}$  in another way. This time it doesn't work because `ZZ` does not have the method `n()`.

# Appendix B

## Kontsevich's star product $\star \bmod \bar{o}(\hbar^6)$

### B.1 Kontsevich's star product $\star \bmod \bar{o}(\hbar^4)$

**Encoding 1.** In the format described in Chapter 11, Implementation 1:

$\hbar^0$ :				2 4 1	0 1 0 1 1 2 2 3	-1/9
# 0 0				2 4 1	0 1 0 1 1 2 3 4	-1/18
2 0 1	1			2 4 1	0 1 0 2 1 2 1 2	1/30
$\hbar^1$ :				2 4 1	0 1 0 4 1 2 1 2	13/90
# 1 1				2 4 1	0 1 0 2 1 2 1 4	-1/45
2 1 1	0 1	1		2 4 1	0 1 0 4 1 2 1 4	-1/30
$\hbar^2$ :				2 4 1	0 1 0 4 1 5 1 2	1/90
# 1 1				2 4 1	0 1 0 2 1 2 1 3	1/30
2 2 1	0 3 1 2	-1/6		2 4 1	0 1 0 4 1 2 1 3	1/15
# 1 2				2 4 1	0 1 0 2 1 3 1 3	1/15
2 2 1	0 1 1 2	-1/3		2 4 1	0 1 0 2 1 3 1 4	1/90
# 2 1				# 3 2		
2 2 1	0 1 0 2	1/3		2 4 1	0 1 0 1 0 2 2 4	-1/6
# 2 2				2 4 1	0 1 0 4 1 3 0 4	-1/6
2 2 1	0 1 0 1	1/2		2 4 1	0 1 0 2 0 5 1 4	-1/18
$\hbar^3$ :				2 4 1	0 1 0 1 0 2 2 3	-1/9
# 1 1				2 4 1	0 1 0 1 0 2 3 4	-1/18
2 3 1	0 3 1 2 2 3	-1/6		2 4 1	0 1 0 2 0 2 1 2	-1/30
# 1 3				2 4 1	0 1 0 2 0 2 1 3	-13/90
2 3 1	0 1 1 2 1 2	1/6		2 4 1	0 1 0 2 0 3 1 2	1/45
# 2 3				2 4 1	0 1 0 2 0 3 1 3	1/30
2 3 1	0 1 0 1 1 2	-1/3		2 4 1	0 1 0 2 0 3 1 4	-1/90
# 3 1				2 4 1	0 1 0 2 0 5 1 2	-1/30
2 3 1	0 1 0 2 0 2	1/6		2 4 1	0 1 0 2 0 5 1 3	-1/15
# 3 2				2 4 1	0 1 0 4 1 2 0 4	-1/15
2 3 1	0 1 0 1 0 2	1/3		2 4 1	0 1 0 4 0 5 1 2	-1/90
# 3 3				# 1 3		
2 3 1	0 1 0 1 0 1	1/6		2 4 1	0 1 1 2 1 2 2 3	7/90
# 1 2				2 4 1	0 1 1 2 1 3 2 3	-1/90
2 3 1	0 1 1 2 2 3	-1/6		2 4 1	0 1 1 2 1 3 2 4	1/30
2 3 1	0 3 1 2 1 2	1/6		2 4 1	0 1 1 2 1 3 3 4	2/45
# 2 1				2 4 1	0 1 1 2 1 5 2 3	1/30
2 3 1	0 1 0 2 2 3	-1/6		2 4 1	0 1 1 2 1 5 2 4	1/45
2 3 1	0 3 1 2 0 3	-1/6		2 4 1	0 1 1 2 1 5 3 4	-1/90
# 2 2				2 4 1	0 1 1 4 1 5 2 3	-1/90
2 3 1	0 1 0 4 1 3	-1/6		2 4 1	0 1 1 4 1 5 2 4	-1/90
2 3 1	0 1 0 2 1 3	-1/6		2 4 1	0 1 1 4 2 3 1 4	1/30
2 3 1	0 1 0 4 1 2	-1/6		2 4 1	0 3 1 4 1 5 1 2	1/90
$\hbar^4$ :				2 4 1	0 3 1 4 1 2 1 3	1/90
# 3 4				2 4 1	0 3 1 2 1 3 1 4	1/90
2 4 1	0 1 0 1 0 1 1 2	-1/6		2 4 1	0 3 1 4 1 2 1 4	-1/30
# 4 3				2 4 1	0 3 1 4 1 2 1 2	-1/45
2 4 1	0 1 0 1 0 1 0 2	1/6		2 4 1	0 3 1 2 1 3 1 3	-1/60
# 4 4				2 4 1	0 3 1 2 1 2 1 3	-1/45
2 4 1	0 1 0 1 0 1 0 1	1/24		2 4 1	0 3 1 2 1 2 1 4	-1/45
# 2 4				2 4 1	0 3 1 2 1 2 1 2	-1/20
2 4 1	0 1 0 1 1 2 1 2	1/6		# 3 1		
2 4 1	0 1 0 1 1 2 1 3	1/18		2 4 1	0 1 0 2 0 2 2 3	-7/90
# 4 2				2 4 1	0 1 0 2 0 3 2 3	1/90
2 4 1	0 1 0 1 0 2 0 2	1/6		2 4 1	0 1 0 2 0 3 2 4	-1/30
2 4 1	0 1 0 1 0 2 0 3	1/18		2 4 1	0 1 0 2 0 3 3 4	-2/45
# 1 4				2 4 1	0 1 0 2 0 5 2 3	-1/30
2 4 1	0 1 1 2 1 2 1 2	-1/30		2 4 1	0 1 0 2 0 5 2 4	-1/45
2 4 1	0 1 1 2 1 2 1 3	-2/45		2 4 1	0 1 0 2 0 5 3 4	1/90
2 4 1	0 1 1 2 1 3 1 3	-1/30		2 4 1	0 1 0 4 0 5 2 3	1/90
2 4 1	0 1 1 2 1 3 1 4	1/45		2 4 1	0 1 0 4 0 5 2 4	1/90
# 3 3				2 4 1	0 1 0 4 2 3 0 4	-1/30
2 4 1	0 1 0 1 0 5 1 4	-1/12		2 4 1	0 3 0 4 0 5 1 2	1/90
2 4 1	0 1 0 1 0 2 1 4	-1/6		2 4 1	0 3 0 4 0 5 1 3	1/90
2 4 1	0 1 0 1 0 5 1 2	-1/6		2 4 1	0 3 0 4 0 5 1 4	1/90
2 4 1	0 1 0 1 0 2 1 3	-1/9		2 4 1	0 3 0 4 1 2 0 3	-1/30
# 4 1				2 4 1	0 3 0 4 1 2 0 4	-1/45
2 4 1	0 1 0 2 0 2 0 2	1/30		2 4 1	0 3 0 4 1 3 0 3	-1/60
2 4 1	0 1 0 2 0 2 0 3	2/45		2 4 1	0 3 0 4 1 3 0 4	-1/45
2 4 1	0 1 0 2 0 3 0 3	1/30		2 4 1	0 3 0 4 1 5 0 4	-1/45
2 4 1	0 1 0 2 0 3 0 4	-1/45		2 4 1	0 3 1 2 0 3 0 3	-1/20
# 2 3				# 1 1		
2 4 1	0 1 0 1 1 2 2 4	-1/6		2 4 1	0 3 1 2 2 3 2 3	-1/40
2 4 1	0 1 0 4 1 3 1 3	1/6		2 4 1	0 3 1 2 2 3 2 4	-1/72
2 4 1	0 1 0 4 1 3 1 2	1/18		2 4 1	0 3 1 2 2 3 3 4	1/72
				2 4 1	0 3 1 2 2 5 3 4	1/360
				2 4 1	0 3 1 4 2 3 2 3	-1/60
				2 4 1	0 3 2 4 1 2 2 4	-1/60



2 4 1	0 3 1 4 2 3 3 4	17/720	2 4 1	0 1 0 2 2 3 3 4	-1/30
2 4 1	0 3 2 4 1 2 2 3	-17/720	2 4 1	0 1 0 2 2 5 3 4	1/720
2 4 1	0 3 1 4 2 5 2 3	-1/180	2 4 1	0 1 0 4 2 3 2 3	19/720
2 4 1	0 3 4 5 1 2 2 4	1/180	2 4 1	0 1 0 4 2 3 2 4	-1/180
2 4 1	0 3 1 4 2 5 2 4	1/360	2 4 1	0 1 0 4 2 3 3 4	-13/360
2 4 1	0 3 4 5 1 2 3 4	-1/360	2 4 1	0 1 0 4 2 5 2 3	1/720
2 4 1	0 3 1 4 2 5 3 4	1/160	2 4 1	0 1 0 4 2 5 2 4	-1/360
2 4 1	0 3 4 5 1 2 2 3	1/160	2 4 1	0 1 0 4 2 5 3 4	-1/720
2 4 1	0 3 1 4 3 5 2 3	-17/1440	2 4 1	0 1 0 4 3 5 2 3	1/80
2 4 1	0 3 2 4 2 5 1 2	17/1440	2 4 1	0 1 0 4 3 5 2 4	1/180
2 4 1	0 3 1 4 3 5 2 4	-1/360	2 4 1	0 3 0 4 1 2 2 3	-1/36
2 4 1	0 3 2 4 3 5 1 2	1/360	2 4 1	0 3 0 4 1 2 2 4	-1/60
2 4 1	0 3 2 4 1 3 2 3	-13/720	2 4 1	0 3 0 4 1 2 3 4	1/720
2 4 1	0 3 2 4 1 3 3 4	-13/720	2 4 1	0 3 0 4 1 3 2 3	-1/45
2 4 1	0 3 2 4 1 3 2 4	-1/60	2 4 1	0 3 0 4 1 3 2 4	-17/720
2 4 1	0 3 2 4 1 5 2 3	-7/720	2 4 1	0 3 0 4 1 3 3 4	-13/360
2 4 1	0 3 4 5 1 5 2 4	7/720	2 4 1	0 3 0 4 1 5 2 3	-1/120
2 4 1	0 3 2 4 1 5 2 4	-1/180	2 4 1	0 3 0 4 1 5 2 4	-1/240
2 4 1	0 3 2 4 1 5 3 4	1/160	2 4 1	0 3 0 4 1 5 3 4	1/80
2 4 1	0 3 2 4 2 5 1 4	1/160	2 4 1	0 3 0 4 2 5 1 2	1/72
2 4 1	0 3 2 4 2 5 1 3	13/1440	2 4 1	0 3 0 4 2 5 1 3	1/90
2 4 1	0 3 4 5 1 3 2 4	-13/1440	2 4 1	0 3 0 4 2 5 1 4	1/180
2 4 1	0 3 2 4 3 5 1 3	-1/1440	2 4 1	0 3 0 4 3 5 1 2	1/360
2 4 1	0 3 4 5 1 3 2 3	1/1440	2 4 1	0 3 0 4 3 5 1 3	-1/720
2 4 1	0 3 2 4 3 5 1 4	1/360	2 4 1	0 3 0 4 3 5 1 4	1/180
2 4 1	0 3 4 5 1 5 2 3	1/240	2 4 1	0 3 1 2 0 3 2 3	-17/180
# 1 2			2 4 1	0 3 1 2 0 3 2 4	-1/72
2 4 1	0 1 1 2 2 3 2 3	-13/360	2 4 1	0 3 1 2 0 3 3 4	7/180
2 4 1	0 1 1 2 2 3 2 4	-1/720	2 4 1	0 3 1 2 0 5 2 3	-7/180
2 4 1	0 1 1 2 2 3 3 4	1/30	2 4 1	0 3 1 2 0 5 2 4	-1/180
2 4 1	0 1 1 2 2 5 3 4	-1/720	2 4 1	0 3 1 2 0 5 3 4	1/90
2 4 1	0 1 1 4 2 3 2 3	-19/720	2 4 1	0 3 1 4 0 5 2 3	-1/80
2 4 1	0 1 1 4 2 3 2 4	1/180	2 4 1	0 3 1 4 0 5 2 4	-1/120
2 4 1	0 1 1 4 2 3 3 4	13/360	2 4 1	0 3 1 4 0 5 3 4	1/40
2 4 1	0 1 1 4 2 5 2 3	-1/720	2 4 1	0 3 1 4 2 3 0 3	1/360
2 4 1	0 1 1 4 2 5 2 4	1/360	2 4 1	0 3 1 4 2 3 0 4	11/720
2 4 1	0 1 1 4 2 5 3 4	1/720	2 4 1	0 3 2 4 1 2 0 3	-7/240
2 4 1	0 1 1 4 3 5 2 3	-1/80	2 4 1	0 3 1 4 2 5 0 4	-1/180
2 4 1	0 1 1 4 3 5 2 4	-1/180	2 4 1	0 3 2 4 1 5 0 3	-1/60
2 4 1	0 3 1 4 1 2 3 4	-1/36	2 4 1	0 3 1 4 3 5 0 4	1/90
2 4 1	0 3 1 4 1 2 2 3	1/60	2 4 1	0 3 2 4 1 3 0 3	-1/60
2 4 1	0 3 1 4 1 2 2 4	-1/720	# 2 2		
2 4 1	0 3 1 2 1 3 3 4	1/45	2 4 1	0 1 0 4 1 3 3 4	-1/6
2 4 1	0 3 1 2 1 3 2 4	17/720	2 4 1	0 3 1 2 0 5 1 4	1/72
2 4 1	0 3 1 2 1 3 2 3	13/360	2 4 1	0 1 0 1 2 3 2 3	17/360
2 4 1	0 3 4 5 1 2 1 4	1/120	2 4 1	0 1 0 1 2 3 2 4	1/24
2 4 1	0 3 2 4 1 5 1 2	1/240	2 4 1	0 1 0 1 2 5 3 4	1/180
2 4 1	0 3 2 4 1 2 1 4	-1/80	2 4 1	0 1 0 2 1 2 2 3	2/45
2 4 1	0 3 1 4 1 5 2 3	-1/72	2 4 1	0 1 0 2 1 2 2 4	-2/45
2 4 1	0 3 1 4 2 5 1 3	-1/90	2 4 1	0 1 0 2 1 2 3 4	-1/30
2 4 1	0 3 2 4 1 5 1 3	-1/180	2 4 1	0 1 0 2 1 3 2 3	1/8
2 4 1	0 3 1 4 1 5 2 4	-1/360	2 4 1	0 1 0 4 1 2 2 4	-1/8
2 4 1	0 3 1 4 2 3 1 3	1/720	2 4 1	0 1 0 2 1 3 2 4	1/720
2 4 1	0 3 2 4 1 3 1 4	-1/180	2 4 1	0 1 0 4 1 2 2 3	-1/720
2 4 1	0 3 1 2 1 2 2 3	17/180	2 4 1	0 1 0 2 1 3 3 4	-11/180
2 4 1	0 3 1 2 1 2 3 4	-1/72	2 4 1	0 1 0 4 1 2 3 4	-11/180
2 4 1	0 3 1 2 1 2 2 4	7/180	2 4 1	0 1 0 2 1 5 2 3	1/36
2 4 1	0 3 1 2 1 5 2 3	7/180	2 4 1	0 1 0 4 2 5 1 2	-1/36
2 4 1	0 3 1 2 1 5 3 4	-1/180	2 4 1	0 1 0 2 1 5 2 4	1/90
2 4 1	0 3 1 2 1 5 2 4	1/90	2 4 1	0 1 0 4 2 3 1 2	-1/90
2 4 1	0 3 1 4 2 5 1 2	1/80	2 4 1	0 1 0 2 1 5 3 4	-1/180
2 4 1	0 3 1 4 3 5 1 2	1/120	2 4 1	0 1 0 4 3 5 1 2	-1/180
2 4 1	0 3 1 4 2 3 1 2	1/40	2 4 1	0 1 0 4 1 3 2 3	1/18
2 4 1	0 3 2 4 1 2 1 2	-1/360	2 4 1	0 1 0 4 1 3 2 4	-1/18
2 4 1	0 3 2 4 1 2 1 3	-11/720	2 4 1	0 1 0 4 1 5 2 3	1/144
2 4 1	0 3 1 4 2 3 1 4	7/240	2 4 1	0 1 0 4 2 5 1 3	-1/144
2 4 1	0 3 4 5 1 2 1 3	-1/180	2 4 1	0 1 0 4 1 5 2 4	-1/90
2 4 1	0 3 1 4 2 5 1 4	1/60	2 4 1	0 1 0 4 2 3 1 3	1/90
2 4 1	0 3 2 4 1 3 1 2	1/90	2 4 1	0 1 0 4 2 3 1 4	-1/60
2 4 1	0 3 2 4 1 3 1 3	1/60	2 4 1	0 1 0 4 2 5 1 4	1/60
# 2 1			2 4 1	0 3 0 4 1 2 1 2	-1/240
2 4 1	0 1 0 2 2 3 2 3	13/360	2 4 1	0 3 1 4 1 2 0 3	-1/240
2 4 1	0 1 0 2 2 3 2 4	1/720	2 4 1	0 3 0 4 1 2 1 3	-13/720

2 4 1	0 3 1 4 0 5 1 3	-13/720
2 4 1	0 3 0 4 1 2 1 4	-1/90
2 4 1	0 3 0 4 1 5 1 3	-1/90
2 4 1	0 3 0 4 1 3 1 2	1/60
2 4 1	0 3 1 2 0 5 1 3	1/60
2 4 1	0 3 0 4 1 3 1 3	1/30
2 4 1	0 3 1 2 0 3 1 3	1/30
2 4 1	0 3 0 4 1 3 1 4	-1/90
2 4 1	0 3 0 4 1 5 1 2	1/360
2 4 1	0 3 1 2 0 3 1 2	13/90
2 4 1	0 3 1 2 0 3 1 4	13/180
2 4 1	0 3 1 2 0 5 1 2	13/180
2 4 1	0 3 1 4 0 5 1 2	1/72

## B.2 Associativity of Kontsevich's $\star \bmod \bar{o}(\hbar^6)$

Here we show the associativity of Kontsevich's  $\star \bmod \bar{o}(\hbar^6)$ , up to  $\bar{o}(\hbar^6)$ .

First we split the associator into two parts, and show their vanishing separately.

The rational part vanishes because it can be expressed as a sum of Leibniz graphs (from the 0th layer):

```
[1]: from gcaops.graph.formality_graph_complex import FormalityGraphComplex
FGC = FormalityGraphComplex(QQ, lazy=True); FGC
n = 6
assoc = FGC.element_from_kgs_encoding(open('data/assoc{}_ratpart.txt'.format(n)).
    ↪read().rstrip())
assoc_n = assoc.homogeneous_part(3,n,2*n)
print('Number of Kontsevich graphs:', len(assoc_n), flush=True)
diff_orders = list(assoc_n.differential_orders())
print('Number of differential orders:', len(diff_orders), flush=True)
from gcaops.graph.leibniz_graph_expansion import
    ↪kontsevich_graph_sum_to_leibniz_graph_sum
from gcaops.graph.leibniz_graph_expansion import
    ↪leibniz_graph_sum_to_kontsevich_graph_sum
for diff_order in diff_orders:
    print(diff_order, end=' ', flush=True)
    part = assoc_n.part_of_differential_order(diff_order)
    part_leibniz = kontsevich_graph_sum_to_leibniz_graph_sum(part, verbose=True)
    print(leibniz_graph_sum_to_kontsevich_graph_sum(part_leibniz) == part, flush=True)
```

```
Number of Kontsevich graphs: 290243
Number of differential orders: 105
(3, 1, 4): 449K -> +220L -> +26K
True
(2, 2, 4): 829K -> +424L -> +71K
True
(2, 1, 4): 1524K -> +780L -> +115K
True
(1, 3, 4): 443K -> +220L -> +32K
True
(1, 2, 4): 1515K -> +780L -> +124K
True
(1, 1, 4): 2315K -> +1135L -> +281K
True
(3, 2, 4): 208K -> +98L -> +17K
True
(2, 3, 4): 203K -> +98L -> +22K
True
(1, 4, 4): 75K -> +36L -> +11K
True
(4, 1, 4): 82K -> +36L -> +4K
True
(4, 2, 4): 32K -> +14L -> +7K
True
(3, 3, 4): 38K -> +16L -> +7K
True
(2, 4, 4): 32K -> +14L -> +7K
True
(4, 2, 3): 208K -> +98L -> +17K
```

True  
 (4, 1, 3): 449K -> +220L -> +26K  
 True  
 (3, 3, 3): 362K -> +175L -> +30K  
 True  
 (3, 2, 3): 1424K -> +810L -> +161K  
 True  
 (3, 1, 3): 2612K -> +1475L -> +199K  
 True  
 (2, 4, 3): 199K -> +97L -> +26K  
 True  
 (2, 3, 3): 1423K -> +810L -> +162K  
 True  
 (2, 2, 3): 4984K -> +2947L -> +451K  
 True  
 (2, 1, 3): 7702K -> +4353L -> +618K  
 True  
 (1, 4, 3): 417K -> +215L -> +58K  
 True  
 (1, 3, 3): 2583K -> +1469L -> +216K  
 True  
 (1, 2, 3): 7659K -> +4350L -> +661K  
 True  
 (1, 1, 3): 10263K -> +5295L -> +1217K  
 True  
 (4, 3, 3): 38K -> +16L -> +7K  
 True  
 (3, 4, 3): 38K -> +16L -> +7K  
 True  
 (2, 5, 3): 10K -> +5L -> +5K  
 True  
 (1, 5, 3): 36K -> +14L -> +6K  
 True  
 (5, 2, 3): 15K -> +5L -> +0K  
 True  
 (5, 1, 3): 41K -> +14L -> +1K  
 True  
 (5, 3, 3): 3K -> +1L -> +0K  
 True  
 (4, 4, 3): 3K -> +1L -> +0K  
 True  
 (3, 5, 3): 3K -> +1L -> +0K  
 True  
 (1, 5, 4): 7K -> +4L -> +5K  
 True  
 (5, 1, 4): 12K -> +4L -> +0K  
 True  
 (5, 2, 4): 3K -> +1L -> +0K  
 True  
 (4, 3, 4): 3K -> +1L -> +0K  
 True  
 (3, 4, 4): 3K -> +1L -> +0K  
 True  
 (2, 5, 4): 3K -> +1L -> +0K  
 True  
 (4, 2, 2): 829K -> +424L -> +71K  
 True  
 (4, 1, 2): 1524K -> +780L -> +115K

True  
 (3, 3, 2): 1423K -> +810L -> +162K  
 True  
 (3, 2, 2): 4984K -> +2947L -> +451K  
 True  
 (3, 1, 2): 7702K -> +4353L -> +618K  
 True  
 (2, 3, 2): 4779K -> +2908L -> +631K  
 True  
 (2, 2, 2): 14046K -> +8416L -> +1618K  
 True  
 (2, 1, 2): 18894K -> +10298L -> +1904K  
 True  
 (1, 4, 2): 1338K -> +752L -> +266K  
 True  
 (1, 3, 2): 7297K -> +4282L -> +956K  
 True  
 (1, 2, 2): 19000K -> +10368L -> +1796K  
 True  
 (1, 1, 2): 22789K -> +10742L -> +2227K  
 True  
 (4, 3, 2): 203K -> +98L -> +22K  
 True  
 (3, 4, 2): 199K -> +97L -> +26K  
 True  
 (2, 4, 2): 758K -> +421L -> +141K  
 True  
 (1, 5, 2): 96K -> +43L -> +33K  
 True  
 (5, 2, 2): 53K -> +18L -> +1K  
 True  
 (5, 1, 2): 121K -> +43L -> +8K  
 True  
 (5, 3, 2): 15K -> +5L -> +0K  
 True  
 (4, 4, 2): 32K -> +14L -> +7K  
 True  
 (2, 5, 2): 43K -> +18L -> +11K  
 True  
 (3, 5, 2): 10K -> +5L -> +5K  
 True  
 (4, 2, 1): 1515K -> +780L -> +124K  
 True  
 (4, 1, 1): 2315K -> +1135L -> +281K  
 True  
 (3, 3, 1): 2583K -> +1469L -> +216K  
 True  
 (3, 2, 1): 7659K -> +4350L -> +661K  
 True  
 (3, 1, 1): 10263K -> +5295L -> +1217K  
 True  
 (2, 4, 1): 1338K -> +752L -> +266K  
 True  
 (2, 3, 1): 7297K -> +4282L -> +956K  
 True  
 (2, 2, 1): 19000K -> +10368L -> +1796K  
 True  
 (2, 1, 1): 22789K -> +10742L -> +2227K

True

(1, 4, 1): 2223K -> +1135L -> +373K

True

(1, 3, 1): 10068K -> +5290L -> +1412K

True

(1, 2, 1): 22591K -> +10736L -> +2424K

True

(1, 1, 1): 23814K -> +9358L -> +2709K

True

(4, 3, 1): 443K -> +220L -> +32K

True

(3, 4, 1): 417K -> +215L -> +58K

True

(2, 5, 1): 96K -> +43L -> +33K

True

(1, 5, 1): 234K -> +81L -> +8K

True

(5, 2, 1): 121K -> +43L -> +8K

True

(5, 1, 1): 234K -> +81L -> +8K

True

(5, 3, 1): 41K -> +14L -> +1K

True

(4, 4, 1): 75K -> +36L -> +11K

True

(3, 5, 1): 36K -> +14L -> +6K

True

(5, 4, 1): 12K -> +4L -> +0K

True

(5, 4, 2): 3K -> +1L -> +0K

True

(4, 5, 2): 3K -> +1L -> +0K

True

(1, 1, 5): 234K -> +81L -> +8K

True

(5, 1, 5): 3K -> +1L -> +0K

True

(4, 2, 5): 3K -> +1L -> +0K

True

(3, 3, 5): 3K -> +1L -> +0K

True

(2, 4, 5): 3K -> +1L -> +0K

True

(1, 5, 5): 3K -> +1L -> +0K

True

(2, 1, 5): 121K -> +43L -> +8K

True

(1, 2, 5): 121K -> +43L -> +8K

True

(4, 1, 5): 12K -> +4L -> +0K

True

(3, 2, 5): 15K -> +5L -> +0K

True

(2, 3, 5): 15K -> +5L -> +0K

True

(1, 4, 5): 12K -> +4L -> +0K

True

(3, 1, 5): 41K -> +14L -> +1K

```

True
(2, 2, 5): 53K -> +18L -> +1K
True
(1, 3, 5): 41K -> +14L -> +1K
True
(4, 5, 1): 7K -> +4L -> +5K
True
(5, 5, 1): 3K -> +1L -> +0K
True

```

The part proportional to  $\zeta(3)^2/\pi^6$  vanishes because it is a sum of Leibniz graphs (from the 0th layer, and from the 1st layer in 6 exceptional cases):

```

[2]: from gcaops.graph.formality_graph_complex import FormalityGraphComplex
FGC = FormalityGraphComplex(QQ, lazy=True); FGC
n = 6
assoc = FGC.element_from_kgs_encoding(open('data/assoc{}_zetapart.txt'.format(n)).
    ↪read().rstrip())
assoc_n = assoc.homogeneous_part(3,n,2*n)
print('Number of Kontsevich graphs:', len(assoc_n), flush=True)
diff_orders = list(assoc_n.differential_orders())
print('Number of differential orders:', len(diff_orders), flush=True)
from gcaops.graph.leibniz_graph_expansion import ↵
    ↪kontsevich_graph_sum_to_leibniz_graph_sum
from gcaops.graph.leibniz_graph_expansion import ↵
    ↪leibniz_graph_sum_to_kontsevich_graph_sum
for diff_order in diff_orders:
    print(diff_order, end=': ', flush=True)
    part = assoc_n.part_of_differential_order(diff_order)
    part_leibniz = kontsevich_graph_sum_to_leibniz_graph_sum(part, verbose=True)
    print(leibniz_graph_sum_to_kontsevich_graph_sum(part_leibniz) == part, flush=True)

```

```

Number of Kontsevich graphs: 194060
Number of differential orders: 28
(2, 1, 3): 4987K -> +3481L -> +1447K
True
(1, 1, 3): 8899K -> +5099L -> +2489K -> +648L -> +201K
True
(3, 1, 3): 732K -> +592L -> +488K
True
(2, 2, 2): 6240K -> +5047L -> +4038K
True
(3, 1, 2): 4987K -> +3481L -> +1447K
True
(2, 1, 2): 16100K -> +9665L -> +2912K -> +575L -> +84K
True
(1, 2, 2): 14200K -> +9001L -> +4579K -> +1224L -> +311K
True
(1, 1, 2): 21813K -> +10699L -> +3178K
True
(3, 2, 2): 988K -> +904L -> +844K
True
(4, 1, 2): 520K -> +392L -> +299K
True
(4, 1, 1): 1363K -> +876L -> +1061K
True

```

```

(3, 2, 1): 4173K -> +3076L -> +2051K
True
(3, 1, 1): 8899K -> +5099L -> +2489K -> +648L -> +201K
True
(2, 3, 1): 2620K -> +2084L -> +2797K
True
(2, 2, 1): 14200K -> +9001L -> +4579K -> +1224L -> +311K
True
(2, 1, 1): 21813K -> +10699L -> +3178K
True
(1, 3, 1): 5913K -> +3834L -> +4472K -> +1749L -> +1122K
True
(1, 2, 1): 20238K -> +10386L -> +4612K
True
(1, 1, 1): 23331K -> +9345L -> +3180K
True
(4, 2, 1): 520K -> +392L -> +299K
True
(3, 3, 1): 670K -> +566L -> +487K
True
(1, 1, 4): 1363K -> +876L -> +1061K
True
(1, 3, 2): 2620K -> +2084L -> +2797K
True
(1, 2, 3): 4173K -> +3076L -> +2051K
True
(2, 1, 4): 520K -> +392L -> +299K
True
(1, 2, 4): 520K -> +392L -> +299K
True
(2, 2, 3): 988K -> +904L -> +844K
True
(1, 3, 3): 670K -> +566L -> +487K
True

```

In fact the need of the 1st layer in the 6 exceptional cases is an artefact of our splitting of the associator. At each of those 6 tri-differential orders, taking the two parts together yields a factorization using Leibniz graphs from the 0th layer:

```

[ ]: from gcaops.graph.formality_graph_complex import FormalityGraphComplex
FGC = FormalityGraphComplex(SR, lazy=True); FGC
n = 6
#star = FGC.element_from_kgs_encoding(open('data/star{}.txt'.format(n)).read()).
↳rstrip()
#%time
#assoc = star.insertion(0, star, max_num_aerial=n) - star.insertion(1, star,
↳max_num_aerial=n)
#with open('data/assoc{}.txt'.format(n), 'w') as f:
#    f.write(assoc.kgs_encoding())
#%time
assoc = FGC.element_from_kgs_encoding(open('data/assoc{}.txt'.format(n)).read()).
↳rstrip()
assoc_n = assoc.homogeneous_part(3,n,2*n)
print('Number of Kontsevich graphs:', len(assoc_n), flush=True)
#diff_orders = list(assoc_n.differential_orders())
#print('Number of differential orders:', len(diff_orders), flush=True)

```



```

from gcaops.graph.leibniz_graph_expansion import 
    ↪kontsevich_graph_sum_to_leibniz_graph_sum
from gcaops.graph.leibniz_graph_expansion import 
    ↪leibniz_graph_sum_to_kontsevich_graph_sum

def coefficient_to_vector(c):
    f = QQ['zzz'](str(SR(c).expand()).replace('zeta(3)^2/pi^6', 'zzz'))
    return vector(QQ, [f.constant_coefficient(), f.monomial_coefficient(QQ['zzz']).
    ↪gen()])
def vector_to_coefficient(v):
    return v[0] + v[1]*zeta(3)^2/pi^6

for diff_order in reversed([(2,1,2), (1,2,2), (2,2,1), (1,1,3), (3,1,1), (1,3,1)]):
    print(diff_order, end=': ', flush=True)
    part = assoc_n.part_of_differential_order(diff_order)
    part_Leibniz = kontsevich_graph_sum_to_leibniz_graph_sum(part, 
    ↪coefficient_to_vector=coefficient_to_vector, 
    ↪vector_to_coefficient=vector_to_coefficient, verbose=True)
    print(leibniz_graph_sum_to_kontsevich_graph_sum(part_Leibniz) == part, flush=True)

```

```

Number of Kontsevich graphs: 290305
(1, 3, 1): 10068K -> +5290L -> +1412K
True
(3, 1, 1): 10264K -> +5295L -> +1216K
True
(1, 1, 3): 10264K -> +5295L -> +1216K
True
(2, 2, 1): 19006K -> +10368L -> +1790K
True
(1, 2, 2): 19006K -> +10368L -> +1790K
True
(2, 1, 2): 18901K -> +10298L -> +1897K
True

```

Hence Kontsevich's  $\star \bmod \bar{o}(\hbar^6)$  is associative; its associator up to  $\bar{o}(\hbar^6)$  is a sum of Leibniz graphs from the 0th layer.



# Appendix C

## Kontsevich affine star product $\star \bmod \bar{o}(\hbar^7)$

### C.1 Original expansion $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$

**Encoding 1.** In the format described in Chapter 11, Implementation 1:

```
h^0:
2 0 1      1
h^1:
2 1 1    0 1    1
h^2:
2 2 1    1 3 0 2    -1/6
2 2 1    1 3 0 1    -1/3
2 2 1    0 3 0 1     1/3
2 2 1    0 1 0 1     1/2
h^3:
2 3 1    1 4 0 1 0 1    -1/3
2 3 1    0 4 0 1 0 1     1/3
2 3 1    0 1 0 1 0 1     1/6
2 3 1    1 3 0 2 0 1    -1/6
2 3 1    1 3 0 4 0 1    -1/6
2 3 1    1 4 0 2 0 1    -1/6
h^4:
2 4 1    1 5 0 1 0 1 0 1    -1/6
2 4 1    0 5 0 1 0 1 0 1     1/6
2 4 1    0 1 0 1 0 1 0 1     1/24
2 4 1    1 5 1 4 0 1 0 1     1/18
2 4 1    0 5 0 4 0 1 0 1     1/18
2 4 1    1 4 1 5 1 3 0 1     1/45
2 4 1    1 3 0 2 0 1 0 1    -1/12
2 4 1    1 3 0 5 0 1 0 1    -1/6
2 4 1    1 5 0 2 0 1 0 1    -1/6
2 4 1    1 5 0 4 0 1 0 1    -1/9
2 4 1    0 4 0 5 0 3 0 1    -1/45
2 4 1    1 5 1 4 0 3 0 1     1/18
2 4 1    3 5 1 4 0 1 0 1     1/18
2 4 1    1 3 1 5 0 2 0 1     1/90
2 4 1    1 4 1 5 0 3 0 1     1/15
2 4 1    1 3 1 4 0 5 0 1     1/90
2 4 1    1 4 0 5 0 2 0 1    -1/18
2 4 1    3 5 0 4 0 1 0 1     1/18
2 4 1    1 3 0 4 0 5 0 1    -1/90
2 4 1    1 4 0 2 0 5 0 1    -1/15
2 4 1    1 5 0 4 0 2 0 1    -1/90
2 4 1    3 4 1 2 1 5 0 1     1/90
2 4 1    3 5 1 4 1 2 0 1     1/90
2 4 1    1 4 1 5 1 3 0 2     1/90
2 4 1    3 4 0 5 0 2 0 1     1/90
2 4 1    3 5 0 4 0 2 0 1    -1/90
2 4 1    1 3 0 5 0 2 0 4     1/90
2 4 1    1 5 1 4 0 3 0 2     1/72
```

2 4 1	3 5 2 4 0 1 0 1	1/180
2 4 1	3 4 1 2 0 5 0 1	1/180
2 4 1	3 4 1 5 0 2 0 1	1/180
2 4 1	4 5 1 2 0 3 0 1	-1/144
2 4 1	3 5 1 4 0 2 0 1	1/144
2 4 1	1 3 1 4 0 5 0 2	1/360
2 4 1	1 5 1 4 0 2 0 3	1/72
h <sup>5</sup> :		
2 5 1	0 1 0 1 0 1 0 1 0 1	1/120
2 5 1	1 6 0 1 0 1 0 1 0 1	-1/18
2 5 1	0 6 0 1 0 1 0 1 0 1	1/18
2 5 1	1 3 0 6 0 1 0 1 0 1	-1/12
2 5 1	1 6 0 2 0 1 0 1 0 1	-1/12
2 5 1	1 6 1 5 0 1 0 1 0 1	1/18
2 5 1	0 6 0 5 0 1 0 1 0 1	1/18
2 5 1	1 6 0 5 0 1 0 1 0 1	-1/9
2 5 1	1 4 1 6 1 3 0 1 0 1	1/45
2 5 1	0 4 0 6 0 3 0 1 0 1	-1/45
2 5 1	1 4 1 6 0 3 0 1 0 1	1/15
2 5 1	1 4 0 2 0 6 0 1 0 1	-1/15
2 5 1	3 6 1 5 0 1 0 1 0 1	1/18
2 5 1	3 6 0 5 0 1 0 1 0 1	1/18
2 5 1	4 5 1 2 1 6 0 1 0 1	-1/90
2 5 1	4 5 0 2 0 6 0 1 0 1	1/90
2 5 1	1 3 1 6 0 2 0 1 0 1	1/90
2 5 1	1 3 0 4 0 6 0 1 0 1	-1/90
2 5 1	1 5 1 6 0 3 0 1 0 1	1/18
2 5 1	1 4 0 5 0 6 0 1 0 1	-1/18
2 5 1	1 4 1 5 1 3 0 2 0 1	1/90
2 5 1	1 3 0 5 0 2 0 4 0 1	1/90
2 5 1	3 6 1 4 1 2 0 1 0 1	1/90
2 5 1	3 6 0 4 0 2 0 1 0 1	-1/90
2 5 1	1 4 1 5 1 3 0 6 0 1	1/90
2 5 1	1 6 0 5 0 2 0 4 0 1	1/90
2 5 1	1 3 1 4 0 6 0 1 0 1	1/90
2 5 1	1 6 0 4 0 2 0 1 0 1	-1/90
2 5 1	1 4 1 5 0 6 0 1 0 1	1/18
2 5 1	1 5 0 6 0 2 0 1 0 1	-1/18
2 5 1	3 6 1 5 1 2 0 4 0 1	-1/240
2 5 1	5 6 1 4 0 2 0 3 0 1	-1/240
2 5 1	1 5 1 4 0 2 0 3 0 1	1/72
2 5 1	3 6 1 4 1 5 0 1 0 1	-1/90
2 5 1	3 6 0 4 0 5 0 1 0 1	1/90
2 5 1	1 4 1 6 1 3 0 2 0 1	1/90
2 5 1	1 3 0 5 0 6 0 4 0 1	1/90
2 5 1	4 6 1 2 0 5 0 3 0 1	-1/720
2 5 1	3 6 1 4 1 5 0 2 0 1	-1/720
2 5 1	3 6 1 4 0 5 0 2 0 1	-1/1440
2 5 1	5 6 1 4 1 2 0 3 0 1	-1/1440
2 5 1	1 4 1 5 0 6 0 2 0 3	1/480
2 5 1	1 4 1 6 1 5 0 3 0 2	-1/480
2 5 1	1 6 1 5 0 3 0 2 0 1	43/1440
2 5 1	1 5 1 4 0 6 0 3 0 1	43/1440
2 5 1	3 6 1 4 0 5 0 1 0 1	7/360
2 5 1	4 6 1 5 0 3 0 1 0 1	-7/360
2 5 1	4 6 1 2 0 5 0 1 0 1	-11/1440
2 5 1	3 6 1 5 0 2 0 1 0 1	11/1440
2 5 1	3 6 4 5 1 2 0 1 0 1	-1/288
2 5 1	3 6 4 5 0 2 0 1 0 1	1/288
2 5 1	4 6 1 2 0 3 0 1 0 1	-1/144
2 5 1	3 6 1 4 0 2 0 1 0 1	1/144
2 5 1	3 4 1 6 1 5 0 2 0 1	-1/1440
2 5 1	4 5 1 2 0 6 0 3 0 1	-1/1440
2 5 1	1 4 1 6 0 5 0 3 0 1	19/1440
2 5 1	1 5 1 2 0 3 0 6 0 1	19/1440
2 5 1	1 4 1 2 0 5 0 3 0 1	1/360
2 5 1	3 5 1 4 0 2 0 6 0 1	1/240
2 5 1	3 5 1 6 1 2 0 4 0 1	-1/240
2 5 1	3 4 1 6 0 5 0 1 0 1	23/720
2 5 1	1 4 1 2 0 6 0 3 0 5	-1/480
2 5 1	1 4 1 6 1 3 0 2 0 5	1/480
2 5 1	3 5 1 4 1 2 0 6 0 1	1/180
2 5 1	3 4 1 6 0 5 0 2 0 1	-1/180

2 5 1	1 6 1 2 0 5 0 3 0 1	1/720
2 5 1	1 3 1 4 0 5 0 6 0 1	1/720
2 5 1	1 3 0 2 0 1 0 1 0 1	-1/36
2 5 1	1 4 1 6 0 2 0 1 0 1	1/18
2 5 1	1 4 0 6 0 2 0 1 0 1	-1/18
2 5 1	3 4 1 6 1 2 0 1 0 1	-1/90
2 5 1	3 4 0 6 0 2 0 1 0 1	1/90
2 5 1	3 6 2 5 0 1 0 1 0 1	1/180
2 5 1	4 5 1 6 0 3 0 2 0 1	-1/180
2 5 1	3 4 1 2 1 5 0 6 0 1	1/180
2 5 1	3 4 1 2 0 5 0 6 0 1	1/1440
2 5 1	3 5 1 4 1 6 0 2 0 1	-1/1440
2 5 1	1 6 1 4 0 3 0 2 0 1	1/36
2 5 1	1 5 1 4 0 3 0 6 0 1	1/36
2 5 1	3 6 2 4 1 5 0 1 0 1	-1/240
2 5 1	3 4 2 5 0 6 0 1 0 1	1/240
2 5 1	3 4 1 2 0 6 0 1 0 1	1/180
2 5 1	3 4 1 6 0 2 0 1 0 1	1/180
2 5 1	1 5 1 4 0 3 0 2 0 1	1/72
h <sup>6</sup> :		
2 6 1	0 1 0 1 0 1 0 1 0 1 0 1	1/720
2 6 1	1 7 0 1 0 1 0 1 0 1 0 1	-1/72
2 6 1	0 7 0 1 0 1 0 1 0 1 0 1	1/72
2 6 1	1 3 0 7 0 1 0 1 0 1 0 1	-1/36
2 6 1	1 7 0 2 0 1 0 1 0 1 0 1	-1/36
2 6 1	1 7 1 6 0 1 0 1 0 1 0 1	1/36
2 6 1	0 7 0 6 0 1 0 1 0 1 0 1	1/36
2 6 1	1 7 0 6 0 1 0 1 0 1 0 1	-1/18
2 6 1	1 4 1 7 1 3 0 1 0 1 0 1	1/90
2 6 1	0 4 0 7 0 3 0 1 0 1 0 1	-1/90
2 6 1	1 4 1 7 0 3 0 1 0 1 0 1	1/30
2 6 1	1 4 0 2 0 7 0 1 0 1 0 1	-1/30
2 6 1	1 3 1 4 0 7 0 1 0 1 0 1	1/180
2 6 1	1 7 0 4 0 2 0 1 0 1 0 1	-1/180
2 6 1	3 7 1 6 0 1 0 1 0 1 0 1	1/36
2 6 1	3 7 0 6 0 1 0 1 0 1 0 1	1/36
2 6 1	1 3 1 7 0 2 0 1 0 1 0 1	1/180
2 6 1	1 3 0 4 0 7 0 1 0 1 0 1	-1/180
2 6 1	1 6 1 7 0 3 0 1 0 1 0 1	1/18
2 6 1	1 4 0 7 0 6 0 1 0 1 0 1	-1/18
2 6 1	1 4 1 6 0 7 0 1 0 1 0 1	1/18
2 6 1	1 7 0 2 0 6 0 1 0 1 0 1	-1/18
2 6 1	4 6 1 2 1 7 0 1 0 1 0 1	-1/90
2 6 1	4 7 0 2 0 6 0 1 0 1 0 1	1/90
2 6 1	4 7 1 2 0 6 0 1 0 1 0 1	-11/1440
2 6 1	3 7 1 6 0 2 0 1 0 1 0 1	11/1440
2 6 1	3 7 1 4 1 2 0 1 0 1 0 1	1/180
2 6 1	4 7 0 2 0 3 0 1 0 1 0 1	-1/180
2 6 1	4 7 1 2 0 3 0 1 0 1 0 1	-1/288
2 6 1	3 7 1 4 0 2 0 1 0 1 0 1	1/288
2 6 1	1 5 1 4 0 2 0 3 0 1 0 1	1/144
2 6 1	1 4 1 2 0 5 0 3 0 1 0 1	1/720
2 6 1	1 6 1 7 1 5 1 3 1 4 0 1	-2/945
2 6 1	0 6 0 7 0 5 0 3 0 4 0 1	2/945
2 6 1	1 7 1 4 1 5 1 6 0 1 0 1	-1/135
2 6 1	0 7 0 4 0 5 0 6 0 1 0 1	-1/135
2 6 1	3 7 1 4 1 6 0 1 0 1 0 1	-1/90
2 6 1	3 7 0 4 0 6 0 1 0 1 0 1	1/90
2 6 1	1 6 1 7 1 5 1 3 1 4 0 2	-1/945
2 6 1	1 3 0 7 0 2 0 6 0 4 0 5	-1/945
2 6 1	1 4 1 7 1 3 0 2 0 1 0 1	1/90
2 6 1	1 3 0 5 0 7 0 4 0 1 0 1	1/90
2 6 1	1 6 1 4 1 5 1 7 0 3 0 1	-1/210
2 6 1	1 4 0 2 0 5 0 6 0 7 0 1	1/210
2 6 1	4 7 1 6 0 3 0 1 0 1 0 1	-7/360
2 6 1	3 7 1 4 0 6 0 1 0 1 0 1	7/360
2 6 1	1 3 1 4 0 5 0 7 0 1 0 1	1/720
2 6 1	1 7 1 2 0 5 0 3 0 1 0 1	1/720
2 6 1	1 4 1 7 0 5 0 3 0 1 0 1	19/1440
2 6 1	1 5 1 2 0 3 0 7 0 1 0 1	19/1440
2 6 1	3 4 1 7 0 6 0 1 0 1 0 1	23/720
2 6 1	1 5 1 4 0 7 0 3 0 1 0 1	43/1440
2 6 1	1 7 1 5 0 3 0 2 0 1 0 1	43/1440

2 6 1	3 4 1 5 1 7 1 6 1 2 0 1	-1/945
2 6 1	3 4 0 5 0 7 0 6 0 2 0 1	1/945
2 6 1	1 7 1 4 1 5 1 6 0 3 0 1	-1/270
2 6 1	1 4 0 7 0 5 0 6 0 2 0 1	1/270
2 6 1	1 4 1 7 1 5 1 6 0 3 0 1	-1/210
2 6 1	1 4 0 5 0 7 0 6 0 2 0 1	1/210
2 6 1	1 4 1 5 1 3 0 7 0 1 0 1	1/90
2 6 1	1 7 0 5 0 2 0 4 0 1 0 1	1/90
2 6 1	1 4 1 5 1 3 0 2 0 1 0 1	1/180
2 6 1	1 3 0 5 0 2 0 4 0 1 0 1	1/180
2 6 1	1 7 1 6 0 3 0 2 0 1 0 1	1/72
2 6 1	1 5 1 4 0 6 0 7 0 1 0 1	1/72
2 6 1	1 5 1 6 0 3 0 7 0 1 0 1	1/36
2 6 1	5 7 1 4 1 2 0 3 0 1 0 1	-1/1440
2 6 1	3 7 1 4 0 5 0 2 0 1 0 1	-1/1440
2 6 1	5 6 1 7 1 2 0 3 0 1 0 1	15/4*zeta(3)^2/pi^6-19/3240
2 6 1	3 6 1 4 0 7 0 2 0 1 0 1	15/4*zeta(3)^2/pi^6-19/3240
2 6 1	3 7 1 4 1 5 0 2 0 1 0 1	-1/720
2 6 1	4 7 1 2 0 5 0 3 0 1 0 1	-1/720
2 6 1	1 6 1 7 1 5 0 3 0 4 0 1	27/8*zeta(3)^2/pi^6-53/3024
2 6 1	1 5 1 6 0 3 0 7 0 2 0 1	-27/8*zeta(3)^2/pi^6+53/3024
2 6 1	1 7 1 6 1 5 0 3 0 4 0 1	-1/216
2 6 1	1 6 1 5 0 7 0 2 0 3 0 1	1/216
2 6 1	5 7 4 6 1 2 1 3 0 1 0 1	17/2*zeta(3)^2/pi^6-29/2268
2 6 1	5 7 4 6 0 2 0 3 0 1 0 1	17/2*zeta(3)^2/pi^6-29/2268
2 6 1	1 4 1 7 1 5 1 6 0 3 0 2	-33/2*zeta(3)^2/pi^6+103/4536
2 6 1	1 5 1 4 0 6 0 3 0 7 0 2	-33/2*zeta(3)^2/pi^6+103/4536
2 6 1	3 7 1 4 1 6 0 2 0 1 0 1	15/4*zeta(3)^2/pi^6-103/12960
2 6 1	4 7 1 2 0 5 0 6 0 1 0 1	15/4*zeta(3)^2/pi^6-103/12960
2 6 1	3 4 1 7 1 6 1 2 0 5 0 1	4*zeta(3)^2/pi^6-43/7560
2 6 1	5 6 1 4 0 2 0 7 0 3 0 1	-4*zeta(3)^2/pi^6+43/7560
2 6 1	4 7 1 2 1 5 0 6 0 1 0 1	15/4*zeta(3)^2/pi^6-157/12960
2 6 1	5 7 1 6 0 2 0 3 0 1 0 1	15/4*zeta(3)^2/pi^6-157/12960
2 6 1	3 7 4 6 1 2 0 1 0 1 0 1	-1/288
2 6 1	3 7 4 6 0 2 0 1 0 1 0 1	1/288
2 6 1	4 7 1 2 1 5 0 3 0 1 0 1	-1/240
2 6 1	5 7 1 4 0 2 0 3 0 1 0 1	-1/240
2 6 1	4 6 5 7 1 3 0 2 0 1 0 1	49/4*zeta(3)^2/pi^6-47/2520
2 6 1	1 7 1 6 1 5 0 3 0 2 0 4	-105/16*zeta(3)^2/pi^6+731/90720
2 6 1	1 7 1 6 1 5 0 1 0 1 0 1	-1/162
2 6 1	0 7 0 6 0 5 0 1 0 1 0 1	1/162
2 6 1	1 7 1 6 0 5 0 1 0 1 0 1	1/54
2 6 1	1 7 0 6 0 5 0 1 0 1 0 1	-1/54
2 6 1	1 5 1 6 1 7 0 4 0 1 0 1	-1/45
2 6 1	1 5 0 6 0 2 0 7 0 1 0 1	-1/45
2 6 1	1 7 1 4 1 5 0 6 0 1 0 1	-1/270
2 6 1	1 6 0 7 0 5 0 2 0 1 0 1	-1/270
2 6 1	4 5 1 6 1 7 0 1 0 1 0 1	-1/54
2 6 1	4 7 0 6 0 5 0 1 0 1 0 1	1/54
2 6 1	4 6 1 5 1 7 1 2 0 1 0 1	-1/270
2 6 1	4 6 0 5 0 7 0 2 0 1 0 1	-1/270
2 6 1	3 7 1 6 1 2 1 4 1 5 0 1	-1/945
2 6 1	3 7 0 6 0 2 0 4 0 5 0 1	1/945
2 6 1	1 5 1 4 1 7 1 6 0 3 0 1	-1/945
2 6 1	1 4 0 6 0 5 0 7 0 2 0 1	1/945
2 6 1	1 7 1 4 1 6 0 3 0 1 0 1	-1/270
2 6 1	1 4 0 7 0 5 0 6 0 1 0 1	-1/270
2 6 1	4 6 1 7 1 5 1 2 0 1 0 1	-1/270
2 6 1	4 6 0 7 0 5 0 2 0 1 0 1	-1/270
2 6 1	1 5 1 6 1 3 1 4 0 7 0 1	-1/945
2 6 1	1 7 0 6 0 2 0 4 0 5 0 1	1/945
2 6 1	1 4 1 7 1 3 0 6 0 1 0 1	1/135
2 6 1	1 6 0 5 0 7 0 4 0 1 0 1	1/135
2 6 1	3 4 1 6 1 5 1 2 1 7 0 1	1/945
2 6 1	3 4 0 6 0 5 0 7 0 2 0 1	1/945
2 6 1	1 5 1 6 0 7 0 3 0 1 0 1	1/45
2 6 1	1 6 1 5 0 3 0 7 0 1 0 1	1/45
2 6 1	1 3 1 5 0 6 0 7 0 1 0 1	1/270
2 6 1	1 7 1 6 0 5 0 3 0 1 0 1	1/270
2 6 1	3 7 1 6 0 5 0 1 0 1 0 1	1/54
2 6 1	4 7 1 6 0 5 0 1 0 1 0 1	-1/54
2 6 1	1 6 1 2 0 7 0 3 0 1 0 1	1/270
2 6 1	1 7 1 4 0 5 0 6 0 1 0 1	1/270

2 6 1	4 6 1 5 1 7 0 2 0 1 0 1	-3*zeta(3)^2/pi^6-13/6480
2 6 1	5 6 1 2 0 3 0 7 0 1 0 1	-3*zeta(3)^2/pi^6-13/6480
2 6 1	1 4 1 6 1 5 0 3 0 2 0 1	-1/480
2 6 1	1 4 1 5 0 6 0 2 0 3 0 1	1/480
2 6 1	4 5 1 2 1 7 0 3 0 1 0 1	-1/240
2 6 1	3 5 1 4 0 2 0 7 0 1 0 1	1/240
2 6 1	4 5 1 2 1 7 0 6 0 1 0 1	-59/12960
2 6 1	3 5 1 6 0 2 0 7 0 1 0 1	59/12960
2 6 1	3 4 1 7 1 5 0 2 0 1 0 1	-1/1440
2 6 1	4 5 1 2 0 7 0 3 0 1 0 1	-1/1440
2 6 1	1 6 1 4 1 5 0 7 0 3 0 1	15/8*zeta(3)^2/pi^6-173/30240
2 6 1	1 7 1 5 0 3 0 6 0 2 0 1	-15/8*zeta(3)^2/pi^6+173/30240
2 6 1	3 7 1 5 1 2 1 6 0 4 0 1	-273/16*zeta(3)^2/pi^6+4703/181440
2 6 1	5 7 1 4 0 2 0 6 0 3 0 1	273/16*zeta(3)^2/pi^6-4703/181440
2 6 1	4 6 1 2 1 7 0 3 0 1 0 1	-3*zeta(3)^2/pi^6-11/4320
2 6 1	5 6 1 4 0 2 0 7 0 1 0 1	-3*zeta(3)^2/pi^6-11/4320
2 6 1	3 7 1 6 1 5 1 2 0 4 0 1	287/16*zeta(3)^2/pi^6-1013/36288
2 6 1	5 7 1 4 0 6 0 3 0 2 0 1	-287/16*zeta(3)^2/pi^6+1013/36288
2 6 1	3 7 4 5 1 6 0 1 0 1 0 1	19/12960
2 6 1	3 7 4 5 0 6 0 1 0 1 0 1	-19/12960
2 6 1	1 5 1 4 1 7 1 6 0 3 0 2	23/2*zeta(3)^2/pi^6-1583/90720
2 6 1	1 5 1 4 0 7 0 6 0 3 0 2	23/2*zeta(3)^2/pi^6-1583/90720
2 6 1	5 7 1 4 1 2 0 6 0 1 0 1	-3*zeta(3)^2/pi^6+11/12960
2 6 1	3 7 1 6 0 5 0 2 0 1 0 1	-3*zeta(3)^2/pi^6+11/12960
2 6 1	5 6 1 7 1 2 0 4 0 1 0 1	1/432
2 6 1	3 6 1 5 0 7 0 2 0 1 0 1	1/432
2 6 1	4 6 1 7 1 5 0 2 0 1 0 1	-1/432
2 6 1	5 6 1 2 0 7 0 3 0 1 0 1	-1/432
2 6 1	1 4 1 6 1 5 0 3 0 7 0 1	-3*zeta(3)^2/pi^6+377/90720
2 6 1	1 7 1 5 0 6 0 2 0 3 0 1	3*zeta(3)^2/pi^6-377/90720
2 6 1	3 5 1 4 1 2 0 7 0 1 0 1	1/180
2 6 1	3 4 1 7 0 5 0 2 0 1 0 1	-1/180
2 6 1	3 4 1 5 1 6 1 2 0 7 0 1	-1/1890
2 6 1	4 5 1 7 0 6 0 3 0 2 0 1	1/1890
2 6 1	3 7 1 4 1 2 0 6 0 1 0 1	1/270
2 6 1	4 7 1 6 0 5 0 2 0 1 0 1	1/270
2 6 1	3 6 1 5 1 7 1 4 0 1 0 1	-1/270
2 6 1	3 6 0 5 0 7 0 4 0 1 0 1	-1/270
2 6 1	3 4 1 7 1 5 1 6 0 1 0 1	-1/270
2 6 1	3 4 0 7 0 5 0 6 0 1 0 1	-1/270
2 6 1	1 5 1 7 1 3 1 4 0 2 0 1	-1/945
2 6 1	1 3 0 6 0 7 0 4 0 5 0 1	1/945
2 6 1	3 7 4 5 1 6 1 2 0 1 0 1	37/16*zeta(3)^2/pi^6-701/181440
2 6 1	4 5 2 6 0 7 0 3 0 1 0 1	37/16*zeta(3)^2/pi^6-701/181440
2 6 1	3 7 1 4 1 5 0 6 0 1 0 1	-9/8*zeta(3)^2/pi^6-89/12960
2 6 1	4 7 1 6 0 5 0 3 0 1 0 1	-9/8*zeta(3)^2/pi^6-89/12960
2 6 1	3 5 4 7 1 2 1 6 0 1 0 1	-1/16*zeta(3)^2/pi^6+43/181440
2 6 1	5 6 2 4 0 7 0 3 0 1 0 1	-1/16*zeta(3)^2/pi^6+43/181440
2 6 1	3 4 1 7 1 5 0 6 0 1 0 1	3/4*zeta(3)^2/pi^6-1/360
2 6 1	4 5 1 6 0 7 0 3 0 1 0 1	3/4*zeta(3)^2/pi^6-1/360
2 6 1	3 6 1 5 1 2 1 7 0 4 0 1	11/8*zeta(3)^2/pi^6-53/30240
2 6 1	3 5 1 4 0 2 0 6 0 7 0 1	11/8*zeta(3)^2/pi^6-53/30240
2 6 1	3 5 1 4 1 6 0 7 0 1 0 1	-3/4*zeta(3)^2/pi^6-13/6480
2 6 1	3 4 1 7 0 5 0 6 0 1 0 1	3/4*zeta(3)^2/pi^6+13/6480
2 6 1	3 4 1 5 1 6 1 7 0 2 0 1	-11/8*zeta(3)^2/pi^6+137/45360
2 6 1	4 5 1 2 0 6 0 3 0 7 0 1	11/8*zeta(3)^2/pi^6-137/45360
2 6 1	1 4 1 6 1 3 0 2 0 7 0 1	-3/8*zeta(3)^2/pi^6+11/2268
2 6 1	1 4 1 7 0 6 0 3 0 5 0 1	3/8*zeta(3)^2/pi^6-11/2268
2 6 1	1 5 1 7 1 3 1 4 0 2 0 6	5*zeta(3)^2/pi^6-239/30240
2 6 1	1 4 1 2 0 7 0 3 0 5 0 6	5*zeta(3)^2/pi^6-239/30240
2 6 1	1 4 1 7 1 3 0 6 0 2 0 1	3/2*zeta(3)^2/pi^6-53/45360
2 6 1	1 3 1 4 0 6 0 7 0 5 0 1	-3/2*zeta(3)^2/pi^6+53/45360
2 6 1	6 7 1 5 1 2 1 4 0 3 0 1	-7*zeta(3)^2/pi^6+1963/181440
2 6 1	3 7 1 4 0 6 0 2 0 5 0 1	7*zeta(3)^2/pi^6-1963/181440
2 6 1	5 7 1 4 1 6 0 3 0 1 0 1	-9/8*zeta(3)^2/pi^6+7/6480
2 6 1	3 7 1 4 0 5 0 6 0 1 0 1	-9/8*zeta(3)^2/pi^6+7/6480
2 6 1	3 7 1 5 1 6 1 4 0 2 0 1	13/2*zeta(3)^2/pi^6-31/2880
2 6 1	4 7 1 2 0 6 0 3 0 5 0 1	-13/2*zeta(3)^2/pi^6+31/2880
2 6 1	1 4 1 6 1 7 0 3 0 2 0 1	-27/8*zeta(3)^2/pi^6+377/90720
2 6 1	1 5 1 4 0 6 0 3 0 7 0 1	27/8*zeta(3)^2/pi^6-377/90720
2 6 1	4 5 1 6 1 7 0 3 0 1 0 1	3/4*zeta(3)^2/pi^6-91/6480
2 6 1	3 5 1 4 0 6 0 7 0 1 0 1	-3/4*zeta(3)^2/pi^6+91/6480
2 6 1	1 4 1 7 1 6 0 2 0 3 0 1	9/8*zeta(3)^2/pi^6-173/30240

2 6 1	1 5 1 4 0 7 0 6 0 3 0 1	$-9/8*\zeta(3)^2/\pi^6+173/30240$
2 6 1	3 6 1 7 1 5 1 2 0 4 0 1	$-17/16*\zeta(3)^2/\pi^6+131/181440$
2 6 1	3 5 1 4 0 6 0 7 0 2 0 1	$-17/16*\zeta(3)^2/\pi^6+131/181440$
2 6 1	3 4 1 5 1 7 1 6 0 2 0 1	$35/16*\zeta(3)^2/\pi^6-37/12096$
2 6 1	4 5 1 2 0 6 0 7 0 3 0 1	$-35/16*\zeta(3)^2/\pi^6+37/12096$
2 6 1	4 6 1 7 1 5 0 3 0 1 0 1	$-9/8*\zeta(3)^2/\pi^6-13/1440$
2 6 1	5 6 1 4 0 7 0 3 0 1 0 1	$-9/8*\zeta(3)^2/\pi^6-13/1440$
2 6 1	3 7 4 6 1 5 1 2 0 1 0 1	$-41/4*\zeta(3)^2/\pi^6+163/11340$
2 6 1	4 7 2 6 0 5 0 3 0 1 0 1	$-41/4*\zeta(3)^2/\pi^6+163/11340$
2 6 1	1 6 1 4 1 7 0 3 0 5 0 1	$-9/8*\zeta(3)^2/\pi^6+11/18144$
2 6 1	1 5 1 2 0 3 0 6 0 7 0 1	$9/8*\zeta(3)^2/\pi^6-11/18144$
2 6 1	1 4 1 6 1 3 0 2 0 5 0 1	$1/480$
2 6 1	1 4 1 2 0 6 0 3 0 5 0 1	$-1/480$
2 6 1	1 4 1 7 1 5 0 6 0 2 0 1	$-1/1080$
2 6 1	1 5 1 2 0 7 0 6 0 3 0 1	$1/1080$
2 6 1	1 4 1 7 1 5 0 6 0 3 0 1	$-3/2*\zeta(3)^2/\pi^6+11/18144$
2 6 1	1 5 1 2 0 6 0 7 0 3 0 1	$3/2*\zeta(3)^2/\pi^6-11/18144$
2 6 1	3 7 5 6 1 2 0 4 0 1 0 1	$-115/16*\zeta(3)^2/\pi^6+487/45360$
2 6 1	3 7 4 6 1 5 0 2 0 1 0 1	$-115/16*\zeta(3)^2/\pi^6+487/45360$
2 6 1	5 7 1 2 1 6 0 4 0 3 0 1	$-379/32*\zeta(3)^2/\pi^6+3463/181440$
2 6 1	3 7 1 6 1 5 0 2 0 4 0 1	$379/32*\zeta(3)^2/\pi^6-3463/181440$
2 6 1	3 6 1 5 1 7 1 2 0 4 0 1	$-1/420$
2 6 1	3 5 1 4 0 7 0 6 0 2 0 1	$-1/420$
2 6 1	3 6 1 5 1 2 1 4 0 7 0 1	$-1/1890$
2 6 1	3 4 1 7 0 6 0 2 0 5 0 1	$-1/1890$
2 6 1	1 4 1 6 1 3 0 7 0 2 0 1	$1/270$
2 6 1	1 7 1 4 0 6 0 3 0 5 0 1	$-1/270$
2 6 1	5 6 1 4 1 2 0 7 0 3 0 1	$-59/32*\zeta(3)^2/\pi^6+779/362880$
2 6 1	3 4 1 6 1 7 0 2 0 5 0 1	$-59/32*\zeta(3)^2/\pi^6+779/362880$
2 6 1	5 6 1 7 1 2 0 3 0 4 0 1	$2*\zeta(3)^2/\pi^6-299/90720$
2 6 1	3 4 1 5 1 6 0 7 0 2 0 1	$-2*\zeta(3)^2/\pi^6+299/90720$
2 6 1	3 7 4 5 1 2 0 6 0 1 0 1	$11/32*\zeta(3)^2/\pi^6-811/362880$
2 6 1	3 7 4 5 1 6 0 2 0 1 0 1	$-11/32*\zeta(3)^2/\pi^6+811/362880$
2 6 1	1 7 1 4 1 6 0 3 0 2 0 5	$17/4*\zeta(3)^2/\pi^6-643/90720$
2 6 1	1 7 1 4 1 5 0 6 0 2 0 3	$17/4*\zeta(3)^2/\pi^6-643/90720$
2 6 1	3 5 1 6 1 2 0 4 0 7 0 1	$11/4*\zeta(3)^2/\pi^6-31/5040$
2 6 1	3 5 1 6 1 7 0 4 0 2 0 1	$11/4*\zeta(3)^2/\pi^6-31/5040$
2 6 1	3 6 1 4 1 5 0 2 0 7 0 1	$-25/32*\zeta(3)^2/\pi^6+13/13440$
2 6 1	4 5 1 2 1 7 0 6 0 3 0 1	$-25/32*\zeta(3)^2/\pi^6+13/13440$
2 6 1	3 7 1 4 1 5 0 6 0 2 0 1	$-151/32*\zeta(3)^2/\pi^6+191/25920$
2 6 1	5 7 1 4 1 2 0 6 0 3 0 1	$151/32*\zeta(3)^2/\pi^6-191/25920$
2 6 1	3 5 4 6 1 2 0 7 0 1 0 1	$-19/32*\zeta(3)^2/\pi^6-307/362880$
2 6 1	3 4 5 6 1 7 0 2 0 1 0 1	$-19/32*\zeta(3)^2/\pi^6-307/362880$
2 6 1	1 4 1 7 1 3 0 6 0 2 0 5	$-31/16*\zeta(3)^2/\pi^6+47/18144$
2 6 1	3 5 1 4 1 2 0 6 0 7 0 1	$3/4*\zeta(3)^2/\pi^6-53/90720$
2 6 1	3 5 1 4 1 7 0 6 0 2 0 1	$3/4*\zeta(3)^2/\pi^6-53/90720$
2 6 1	4 5 1 2 1 6 0 7 0 3 0 1	$-17/16*\zeta(3)^2/\pi^6-37/90720$
2 6 1	3 5 1 7 1 6 0 4 0 2 0 1	$-17/16*\zeta(3)^2/\pi^6-37/90720$
2 6 1	6 7 1 4 1 5 0 2 0 3 0 1	$-1/4*\zeta(3)^2/\pi^6+41/36288$
2 6 1	3 7 1 6 1 2 0 4 0 5 0 1	$1/4*\zeta(3)^2/\pi^6-41/36288$
2 6 1	1 4 1 6 1 3 0 7 0 5 0 1	$3/2*\zeta(3)^2/\pi^6-53/45360$
2 6 1	1 7 1 2 0 6 0 3 0 5 0 1	$-3/2*\zeta(3)^2/\pi^6+53/45360$
2 6 1	1 3 0 2 0 1 0 1 0 1 0 1	$-1/144$
2 6 1	1 4 1 7 0 2 0 1 0 1 0 1	$1/36$
2 6 1	1 3 0 2 0 7 0 1 0 1 0 1	$-1/36$
2 6 1	3 4 1 7 1 2 0 1 0 1 0 1	$-1/180$
2 6 1	3 4 0 2 0 7 0 1 0 1 0 1	$-1/180$
2 6 1	3 4 1 2 0 7 0 1 0 1 0 1	$1/360$
2 6 1	3 4 1 7 0 2 0 1 0 1 0 1	$1/360$
2 6 1	3 7 2 6 0 1 0 1 0 1 0 1	$1/360$
2 6 1	1 5 1 4 0 3 0 7 0 1 0 1	$1/36$
2 6 1	1 7 1 4 0 3 0 2 0 1 0 1	$1/36$
2 6 1	3 4 1 5 1 2 1 6 1 7 0 1	$1/945$
2 6 1	3 4 0 5 0 2 0 6 0 7 0 1	$-1/945$
2 6 1	1 4 1 6 1 5 1 7 0 3 0 1	$-1/270$
2 6 1	1 4 0 5 0 2 0 6 0 7 0 1	$1/270$
2 6 1	3 5 1 2 1 6 1 7 0 1 0 1	$-1/270$
2 6 1	4 5 0 7 0 6 0 2 0 1 0 1	$1/270$
2 6 1	3 5 1 4 1 7 0 2 0 1 0 1	$-1/1440$
2 6 1	3 4 1 2 0 5 0 7 0 1 0 1	$1/1440$
2 6 1	3 7 2 4 1 6 0 1 0 1 0 1	$-1/240$
2 6 1	3 4 2 6 0 7 0 1 0 1 0 1	$1/240$
2 6 1	1 7 1 4 1 5 1 6 0 3 0 2	$-1/540$



2 6 1	1 4 1 5 0 6 0 3 0 7 0 2	-1/540
2 6 1	3 4 1 2 1 6 1 7 0 5 0 1	-1/420
2 6 1	5 6 1 4 0 7 0 3 0 2 0 1	-1/420
2 6 1	3 4 1 5 1 2 0 7 0 1 0 1	-1/180
2 6 1	4 5 1 7 0 3 0 2 0 1 0 1	-1/180
2 6 1	1 7 1 5 1 6 0 3 0 2 0 4	-1/432
2 6 1	1 5 1 7 1 6 0 2 0 1 0 1	-1/108
2 6 1	1 5 0 7 0 6 0 2 0 1 0 1	-1/108
2 6 1	1 6 1 5 0 7 0 3 0 1 0 1	1/54
2 6 1	1 6 1 7 1 5 0 4 0 3 0 1	-1/90
2 6 1	1 6 1 5 0 3 0 7 0 2 0 1	1/90
2 6 1	4 5 1 7 1 2 0 6 0 1 0 1	-1/540
2 6 1	3 5 1 6 0 7 0 2 0 1 0 1	1/540
2 6 1	1 6 1 4 1 5 0 7 0 2 0 1	-1/540
2 6 1	1 7 1 4 0 3 0 6 0 2 0 1	1/540
2 6 1	4 5 1 6 1 7 0 2 0 1 0 1	-1/540
2 6 1	3 5 1 2 0 6 0 7 0 1 0 1	1/540
2 6 1	4 6 1 5 1 7 0 3 0 1 0 1	-1/108
2 6 1	5 6 1 4 0 3 0 7 0 1 0 1	-1/108
2 6 1	3 5 4 6 1 7 1 2 0 1 0 1	1/540
2 6 1	5 6 2 4 0 3 0 7 0 1 0 1	1/540
2 6 1	3 7 2 6 1 5 0 1 0 1 0 1	-1/540
2 6 1	3 7 2 6 0 5 0 1 0 1 0 1	1/540
2 6 1	3 6 1 2 1 5 1 7 0 4 0 1	-1/1890
2 6 1	3 5 1 4 0 6 0 2 0 7 0 1	-1/1890
2 6 1	1 6 1 4 1 7 0 3 0 2 0 1	-1/540
2 6 1	1 5 1 4 0 3 0 6 0 7 0 1	1/540
2 6 1	3 7 1 5 1 6 1 2 0 4 0 1	-1/540
2 6 1	5 7 1 4 0 3 0 6 0 2 0 1	1/540
2 6 1	3 4 1 5 1 2 1 6 0 7 0 1	1/1890
2 6 1	4 5 1 7 0 6 0 2 0 3 0 1	-1/1890
2 6 1	3 4 1 7 1 2 0 6 0 1 0 1	-1/270
2 6 1	4 5 1 6 0 7 0 2 0 1 0 1	-1/270
2 6 1	3 7 2 5 1 6 1 4 0 1 0 1	-3/8*zeta(3)^2/pi^6+17/90720
2 6 1	3 7 2 4 0 5 0 6 0 1 0 1	-3/8*zeta(3)^2/pi^6+17/90720
2 6 1	3 6 1 5 1 7 1 4 0 2 0 1	-3/4*zeta(3)^2/pi^6+53/90720
2 6 1	3 4 1 2 0 6 0 7 0 5 0 1	-3/4*zeta(3)^2/pi^6+53/90720
2 6 1	3 5 2 4 1 6 1 7 0 1 0 1	-9/8*zeta(3)^2/pi^6+1/360
2 6 1	3 5 2 4 0 6 0 7 0 1 0 1	-9/8*zeta(3)^2/pi^6+1/360
2 6 1	3 5 2 4 1 7 0 6 0 1 0 1	-15/8*zeta(3)^2/pi^6+13/6480
2 6 1	3 6 1 4 1 5 0 7 0 2 0 1	-3/4*zeta(3)^2/pi^6+11/36288
2 6 1	4 5 1 7 1 2 0 6 0 3 0 1	-3/4*zeta(3)^2/pi^6+11/36288
2 6 1	1 7 1 4 1 5 0 6 0 3 0 2	-1/2160
2 6 1	3 4 5 6 1 2 0 7 0 1 0 1	3/2*zeta(3)^2/pi^6-11/25920
2 6 1	3 5 4 6 1 7 0 2 0 1 0 1	3/2*zeta(3)^2/pi^6-11/25920
2 6 1	5 7 1 6 1 2 0 4 0 3 0 1	1/864
2 6 1	4 7 1 6 1 5 0 2 0 3 0 1	-1/864
2 6 1	3 7 2 4 1 5 0 6 0 1 0 1	-3/8*zeta(3)^2/pi^6-11/5040
2 6 1	3 7 2 5 1 6 0 4 0 1 0 1	-3/8*zeta(3)^2/pi^6-11/5040
2 6 1	4 5 1 7 1 6 0 2 0 3 0 1	-3/2*zeta(3)^2/pi^6+377/181440
2 6 1	3 5 1 2 1 6 0 4 0 7 0 1	-3/2*zeta(3)^2/pi^6+377/181440
2 6 1	5 6 1 4 1 7 0 2 0 3 0 1	-3/4*zeta(3)^2/pi^6+53/90720
2 6 1	3 4 1 6 1 2 0 7 0 5 0 1	-3/4*zeta(3)^2/pi^6+53/90720
2 6 1	1 5 1 4 0 3 0 2 0 1 0 1	1/144
2 6 1	3 4 1 7 1 2 1 6 0 5 0 1	1/540
2 6 1	5 6 1 4 0 3 0 7 0 2 0 1	-1/540
2 6 1	1 7 1 6 1 5 0 4 0 3 0 1	-1/216
2 6 1	1 6 1 5 0 7 0 3 0 2 0 1	1/216
2 6 1	3 7 2 6 1 5 0 4 0 1 0 1	-1/1080
2 6 1	4 5 1 6 1 7 0 2 0 3 0 1	-1/1080
2 6 1	4 5 1 6 1 2 0 7 0 3 0 1	-1/1080
2 6 1	1 7 1 6 1 5 0 4 0 3 0 2	-1/1296
h^7:		
2 7 1	0 1 0 1 0 1 0 1 0 1 0 1	1/5040
2 7 1	1 8 0 1 0 1 0 1 0 1 0 1	-1/360
2 7 1	0 8 0 1 0 1 0 1 0 1 0 1	1/360
2 7 1	1 3 0 8 0 1 0 1 0 1 0 1	-1/144
2 7 1	1 8 0 2 0 1 0 1 0 1 0 1	-1/144
2 7 1	1 8 1 7 0 1 0 1 0 1 0 1	1/108
2 7 1	0 8 0 7 0 1 0 1 0 1 0 1	1/108
2 7 1	1 8 0 7 0 1 0 1 0 1 0 1	-1/54
2 7 1	1 3 1 4 1 8 0 1 0 1 0 1	1/270
2 7 1	0 3 0 4 0 8 0 1 0 1 0 1	-1/270

2 7 1	1 4 1 8 0 3 0 1 0 1 0 1 0 1	1/90
2 7 1	1 4 0 2 0 8 0 1 0 1 0 1 0 1	-1/90
2 7 1	1 3 1 4 0 8 0 1 0 1 0 1 0 1	1/540
2 7 1	1 8 0 4 0 2 0 1 0 1 0 1 0 1	-1/540
2 7 1	3 8 1 7 0 1 0 1 0 1 0 1 0 1	1/108
2 7 1	3 8 0 7 0 1 0 1 0 1 0 1 0 1	1/108
2 7 1	1 3 1 8 0 2 0 1 0 1 0 1 0 1	1/540
2 7 1	1 3 0 4 0 8 0 1 0 1 0 1 0 1	-1/540
2 7 1	1 7 1 8 0 3 0 1 0 1 0 1 0 1	1/36
2 7 1	1 4 0 8 0 7 0 1 0 1 0 1 0 1	-1/36
2 7 1	1 4 1 7 0 8 0 1 0 1 0 1 0 1	1/36
2 7 1	1 8 0 2 0 7 0 1 0 1 0 1 0 1	-1/36
2 7 1	4 8 1 2 1 7 0 1 0 1 0 1 0 1	-1/180
2 7 1	4 8 0 2 0 7 0 1 0 1 0 1 0 1	1/180
2 7 1	4 8 1 2 0 7 0 1 0 1 0 1 0 1	-11/2880
2 7 1	3 8 1 7 0 2 0 1 0 1 0 1 0 1	11/2880
2 7 1	3 8 1 4 1 2 0 1 0 1 0 1 0 1	1/540
2 7 1	4 8 0 2 0 3 0 1 0 1 0 1 0 1	-1/540
2 7 1	4 8 1 2 0 3 0 1 0 1 0 1 0 1	-1/864
2 7 1	3 8 1 4 0 2 0 1 0 1 0 1 0 1	1/864
2 7 1	1 4 1 5 1 3 0 2 0 1 0 1 0 1	1/540
2 7 1	1 3 0 5 0 2 0 4 0 1 0 1 0 1	1/540
2 7 1	1 5 1 4 0 2 0 3 0 1 0 1 0 1	1/432
2 7 1	1 3 1 4 0 5 0 2 0 1 0 1 0 1	1/2160
2 7 1	3 8 1 4 1 7 0 1 0 1 0 1 0 1	-1/180
2 7 1	3 8 0 4 0 7 0 1 0 1 0 1 0 1	1/180
2 7 1	1 3 1 4 1 8 0 2 0 1 0 1 0 1	1/180
2 7 1	1 3 0 5 0 8 0 4 0 1 0 1 0 1	1/180
2 7 1	1 4 1 5 1 3 0 8 0 1 0 1 0 1	1/180
2 7 1	1 8 0 5 0 2 0 4 0 1 0 1 0 1	1/180
2 7 1	4 8 1 7 0 3 0 1 0 1 0 1 0 1	-7/720
2 7 1	3 8 1 4 0 7 0 1 0 1 0 1 0 1	7/720
2 7 1	1 3 1 4 0 5 0 8 0 1 0 1 0 1	1/1440
2 7 1	1 3 1 8 0 5 0 2 0 1 0 1 0 1	1/1440
2 7 1	1 5 1 8 0 3 0 4 0 1 0 1 0 1	19/2880
2 7 1	1 3 1 5 0 2 0 8 0 1 0 1 0 1	19/2880
2 7 1	3 4 1 8 0 7 0 1 0 1 0 1 0 1	23/1440
2 7 1	1 4 1 5 0 3 0 8 0 1 0 1 0 1	43/2880
2 7 1	1 5 1 8 0 2 0 3 0 1 0 1 0 1	43/2880
2 7 1	1 5 1 8 0 3 0 7 0 1 0 1 0 1	1/36
2 7 1	1 5 1 4 0 7 0 8 0 1 0 1 0 1	1/72
2 7 1	1 8 1 7 0 3 0 2 0 1 0 1 0 1	1/72
2 7 1	5 8 1 4 1 2 0 3 0 1 0 1 0 1	-1/2880
2 7 1	3 8 1 4 0 5 0 2 0 1 0 1 0 1	-1/2880
2 7 1	3 8 1 4 1 5 0 2 0 1 0 1 0 1	-1/1440
2 7 1	5 8 1 2 0 3 0 4 0 1 0 1 0 1	-1/1440
2 7 1	4 8 1 2 1 5 0 3 0 1 0 1 0 1	-1/480
2 7 1	5 8 1 4 0 2 0 3 0 1 0 1 0 1	-1/480
2 7 1	5 8 1 2 1 7 0 4 0 1 0 1 0 1	15/4*zeta(3)^2/pi^6-19/3240
2 7 1	3 8 1 5 0 2 0 7 0 1 0 1 0 1	15/4*zeta(3)^2/pi^6-19/3240
2 7 1	3 8 1 4 1 7 0 2 0 1 0 1 0 1	15/4*zeta(3)^2/pi^6-103/12960
2 7 1	4 8 1 2 0 5 0 7 0 1 0 1 0 1	15/4*zeta(3)^2/pi^6-103/12960
2 7 1	4 8 1 2 1 5 0 7 0 1 0 1 0 1	15/4*zeta(3)^2/pi^6-157/12960
2 7 1	5 8 1 7 0 2 0 3 0 1 0 1 0 1	15/4*zeta(3)^2/pi^6-157/12960
2 7 1	3 8 4 7 1 2 0 1 0 1 0 1 0 1	-1/576
2 7 1	3 8 4 7 0 2 0 1 0 1 0 1 0 1	1/576
2 7 1	4 7 5 8 1 3 0 2 0 1 0 1 0 1	49/4*zeta(3)^2/pi^6-47/2520
2 7 1	5 8 4 7 1 2 1 3 0 1 0 1 0 1	17/2*zeta(3)^2/pi^6-29/2268
2 7 1	5 8 4 7 0 2 0 3 0 1 0 1 0 1	17/2*zeta(3)^2/pi^6-29/2268
2 7 1	1 8 1 7 1 6 0 1 0 1 0 1 0 1	-1/162
2 7 1	0 8 0 7 0 6 0 1 0 1 0 1 0 1	1/162
2 7 1	1 8 1 7 0 6 0 1 0 1 0 1 0 1	1/54
2 7 1	1 8 0 7 0 6 0 1 0 1 0 1 0 1	-1/54
2 7 1	1 8 1 4 1 5 1 7 0 1 0 1 0 1	-1/135
2 7 1	0 8 0 4 0 5 0 7 0 1 0 1 0 1	-1/135
2 7 1	1 5 1 7 1 8 0 4 0 1 0 1 0 1	-1/45
2 7 1	1 5 0 7 0 2 0 8 0 1 0 1 0 1	-1/45
2 7 1	1 3 1 5 1 7 0 8 0 1 0 1 0 1	-1/270
2 7 1	1 7 0 8 0 5 0 2 0 1 0 1 0 1	-1/270
2 7 1	4 6 1 7 1 8 0 1 0 1 0 1 0 1	-1/54
2 7 1	4 8 0 7 0 6 0 1 0 1 0 1 0 1	1/54
2 7 1	1 4 1 7 1 8 0 2 0 1 0 1 0 1	-1/270
2 7 1	1 4 0 8 0 5 0 7 0 1 0 1 0 1	-1/270

2 7 1	1 3 1 4 1 7 0 8 0 1 0 1 0 1	1/135
2 7 1	1 7 0 5 0 8 0 4 0 1 0 1 0 1	1/135
2 7 1	4 8 1 7 0 6 0 1 0 1 0 1 0 1	-1/54
2 7 1	3 8 1 7 0 6 0 1 0 1 0 1 0 1	1/54
2 7 1	1 7 1 8 0 5 0 3 0 1 0 1 0 1	1/270
2 7 1	1 3 1 5 0 7 0 8 0 1 0 1 0 1	1/270
2 7 1	1 5 1 8 0 2 0 7 0 1 0 1 0 1	1/45
2 7 1	1 4 1 8 0 3 0 7 0 1 0 1 0 1	1/45
2 7 1	1 4 1 8 0 5 0 7 0 1 0 1 0 1	1/270
2 7 1	1 3 1 8 0 2 0 7 0 1 0 1 0 1	1/270
2 7 1	1 7 1 6 1 8 0 4 0 1 0 1 0 1	-1/108
2 7 1	1 5 0 7 0 6 0 8 0 1 0 1 0 1	-1/108
2 7 1	1 5 1 8 1 7 0 6 0 1 0 1 0 1	-1/108
2 7 1	1 6 0 8 0 7 0 2 0 1 0 1 0 1	-1/108
2 7 1	1 5 1 7 0 8 0 6 0 1 0 1 0 1	1/54
2 7 1	1 7 1 8 0 3 0 6 0 1 0 1 0 1	1/54
2 7 1	5 8 1 4 1 2 0 7 0 1 0 1 0 1	-3*zeta(3)^2/pi^6+11/12960
2 7 1	3 8 1 7 0 5 0 2 0 1 0 1 0 1	-3*zeta(3)^2/pi^6+11/12960
2 7 1	4 8 1 5 1 7 0 2 0 1 0 1 0 1	-3*zeta(3)^2/pi^6-13/6480
2 7 1	5 8 1 2 0 3 0 7 0 1 0 1 0 1	-3*zeta(3)^2/pi^6-13/6480
2 7 1	4 5 1 2 1 7 0 8 0 1 0 1 0 1	-59/12960
2 7 1	3 5 1 8 0 2 0 7 0 1 0 1 0 1	59/12960
2 7 1	4 8 1 2 1 7 0 3 0 1 0 1 0 1	-3*zeta(3)^2/pi^6-11/4320
2 7 1	5 8 1 4 0 2 0 7 0 1 0 1 0 1	-3*zeta(3)^2/pi^6-11/4320
2 7 1	3 8 4 6 1 7 0 1 0 1 0 1 0 1	19/12960
2 7 1	3 8 4 6 0 7 0 1 0 1 0 1 0 1	-19/12960
2 7 1	5 6 1 2 1 7 1 8 0 1 0 1 0 1	1/270
2 7 1	5 6 0 7 0 2 0 8 0 1 0 1 0 1	1/270
2 7 1	4 7 1 5 1 8 1 2 0 1 0 1 0 1	-1/270
2 7 1	4 7 0 5 0 8 0 2 0 1 0 1 0 1	-1/270
2 7 1	4 7 1 8 1 5 1 2 0 1 0 1 0 1	-1/270
2 7 1	4 7 0 8 0 5 0 2 0 1 0 1 0 1	-1/270
2 7 1	5 8 1 2 1 6 0 7 0 1 0 1 0 1	11/4320
2 7 1	3 8 1 7 0 2 0 6 0 1 0 1 0 1	11/4320
2 7 1	4 6 1 7 1 8 0 2 0 1 0 1 0 1	-11/4320
2 7 1	5 8 1 2 0 7 0 6 0 1 0 1 0 1	-11/4320
2 7 1	4 8 1 2 1 6 0 7 0 1 0 1 0 1	-1/270
2 7 1	5 8 1 7 0 2 0 6 0 1 0 1 0 1	-1/270
2 7 1	4 5 1 2 1 8 0 3 0 1 0 1 0 1	-1/480
2 7 1	3 5 1 4 0 2 0 8 0 1 0 1 0 1	1/480
2 7 1	3 4 1 5 1 8 0 2 0 1 0 1 0 1	1/2880
2 7 1	4 5 1 2 0 3 0 8 0 1 0 1 0 1	1/2880
2 7 1	3 5 1 4 1 2 0 8 0 1 0 1 0 1	1/360
2 7 1	3 5 1 8 0 2 0 4 0 1 0 1 0 1	-1/360
2 7 1	5 8 1 2 1 7 0 3 0 1 0 1 0 1	1/432
2 7 1	3 8 1 4 0 2 0 7 0 1 0 1 0 1	1/432
2 7 1	3 8 1 5 1 7 0 2 0 1 0 1 0 1	-1/432
2 7 1	4 8 1 2 0 3 0 7 0 1 0 1 0 1	-1/432
2 7 1	3 8 1 4 1 2 0 7 0 1 0 1 0 1	1/270
2 7 1	5 8 1 7 0 2 0 4 0 1 0 1 0 1	1/270
2 7 1	1 6 1 8 1 5 1 3 1 4 0 1 0 1	-2/945
2 7 1	0 6 0 8 0 5 0 3 0 4 0 1 0 1	2/945
2 7 1	1 6 1 4 1 5 1 8 0 3 0 1 0 1	-1/210
2 7 1	1 4 0 2 0 5 0 6 0 8 0 1 0 1	1/210
2 7 1	3 7 1 5 1 8 1 4 0 1 0 1 0 1	-1/270
2 7 1	3 7 0 5 0 8 0 4 0 1 0 1 0 1	-1/270
2 7 1	4 8 1 2 1 5 1 6 1 7 0 1 0 1	1/945
2 7 1	4 8 0 2 0 5 0 6 0 7 0 1 0 1	-1/945
2 7 1	3 4 1 8 1 5 1 7 0 1 0 1 0 1	-1/270
2 7 1	3 4 0 8 0 5 0 7 0 1 0 1 0 1	-1/270
2 7 1	1 5 1 8 1 3 1 4 0 2 0 1 0 1	-1/945
2 7 1	1 3 0 6 0 8 0 4 0 5 0 1 0 1	1/945
2 7 1	1 5 1 8 1 6 0 4 0 3 0 1 0 1	27/8*zeta(3)^2/pi^6-53/3024
2 7 1	1 5 1 6 0 3 0 8 0 2 0 1 0 1	-27/8*zeta(3)^2/pi^6+53/3024
2 7 1	1 3 1 5 1 6 0 4 0 8 0 1 0 1	-3*zeta(3)^2/pi^6+377/90720
2 7 1	1 8 1 5 0 6 0 2 0 3 0 1 0 1	3*zeta(3)^2/pi^6-377/90720
2 7 1	3 8 1 4 1 5 0 7 0 1 0 1 0 1	-9/8*zeta(3)^2/pi^6-89/12960
2 7 1	5 8 1 7 0 3 0 4 0 1 0 1 0 1	-9/8*zeta(3)^2/pi^6-89/12960
2 7 1	3 5 1 4 1 7 0 8 0 1 0 1 0 1	-3/4*zeta(3)^2/pi^6-13/6480
2 7 1	3 4 1 8 0 5 0 7 0 1 0 1 0 1	3/4*zeta(3)^2/pi^6+13/6480
2 7 1	1 3 1 8 1 2 0 6 0 4 0 1 0 1	3/2*zeta(3)^2/pi^6-53/45360
2 7 1	1 3 1 4 0 6 0 8 0 5 0 1 0 1	-3/2*zeta(3)^2/pi^6+53/45360
2 7 1	1 3 1 5 1 2 0 8 0 4 0 1 0 1	-3/8*zeta(3)^2/pi^6+11/2268

2 7 1	1 4 1 8 0 6 0 3 0 5 0 1 0 1	$3/8*\zeta(3)^2/\pi^6-11/2268$
2 7 1	3 4 1 5 1 7 0 8 0 1 0 1 0 1	$-3/4*\zeta(3)^2/\pi^6+1/360$
2 7 1	4 5 1 8 0 3 0 7 0 1 0 1 0 1	$-3/4*\zeta(3)^2/\pi^6+1/360$
2 7 1	5 8 1 4 1 7 0 3 0 1 0 1 0 1	$-9/8*\zeta(3)^2/\pi^6+7/6480$
2 7 1	3 8 1 4 0 5 0 7 0 1 0 1 0 1	$-9/8*\zeta(3)^2/\pi^6+7/6480$
2 7 1	4 5 1 7 1 8 0 3 0 1 0 1 0 1	$3/4*\zeta(3)^2/\pi^6-91/6480$
2 7 1	3 5 1 4 0 8 0 7 0 1 0 1 0 1	$-3/4*\zeta(3)^2/\pi^6+91/6480$
2 7 1	1 6 1 8 1 2 0 4 0 3 0 1 0 1	$9/8*\zeta(3)^2/\pi^6-173/30240$
2 7 1	1 5 1 4 0 8 0 6 0 3 0 1 0 1	$-9/8*\zeta(3)^2/\pi^6+173/30240$
2 7 1	1 8 1 2 1 6 0 4 0 3 0 1 0 1	$-27/8*\zeta(3)^2/\pi^6+377/90720$
2 7 1	1 5 1 4 0 6 0 3 0 8 0 1 0 1	$27/8*\zeta(3)^2/\pi^6-377/90720$
2 7 1	3 8 1 5 1 7 0 4 0 1 0 1 0 1	$-9/8*\zeta(3)^2/\pi^6-13/1440$
2 7 1	4 8 1 5 0 3 0 7 0 1 0 1 0 1	$-9/8*\zeta(3)^2/\pi^6-13/1440$
2 7 1	1 8 1 5 1 2 0 6 0 4 0 1 0 1	$-9/8*\zeta(3)^2/\pi^6+11/18144$
2 7 1	1 5 1 2 0 3 0 6 0 8 0 1 0 1	$9/8*\zeta(3)^2/\pi^6-11/18144$
2 7 1	1 3 1 5 1 8 0 6 0 4 0 1 0 1	$-3/2*\zeta(3)^2/\pi^6+11/18144$
2 7 1	1 5 1 2 0 6 0 8 0 3 0 1 0 1	$3/2*\zeta(3)^2/\pi^6-11/18144$
2 7 1	1 3 1 4 1 5 0 6 0 8 0 1 0 1	$3/2*\zeta(3)^2/\pi^6-53/45360$
2 7 1	1 8 1 2 0 6 0 3 0 5 0 1 0 1	$-3/2*\zeta(3)^2/\pi^6+53/45360$
2 7 1	1 6 1 5 1 3 0 8 0 4 0 1 0 1	$15/8*\zeta(3)^2/\pi^6-173/30240$
2 7 1	1 8 1 5 0 3 0 6 0 2 0 1 0 1	$-15/8*\zeta(3)^2/\pi^6+173/30240$
2 7 1	1 4 1 8 1 5 1 6 0 3 0 1 0 1	$-1/210$
2 7 1	1 4 0 5 0 8 0 6 0 2 0 1 0 1	$1/210$
2 7 1	1 8 1 4 1 5 1 6 0 3 0 1 0 1	$-1/270$
2 7 1	1 4 0 8 0 5 0 6 0 2 0 1 0 1	$1/270$
2 7 1	1 4 1 8 1 5 1 7 0 3 0 1 0 1	$-1/270$
2 7 1	1 4 0 5 0 8 0 6 0 7 0 1 0 1	$1/270$
2 7 1	3 4 1 8 1 5 1 6 1 2 0 1 0 1	$1/945$
2 7 1	3 4 0 8 0 5 0 6 0 2 0 1 0 1	$-1/945$
2 7 1	3 8 1 6 1 2 1 4 1 5 0 1 0 1	$-1/945$
2 7 1	3 8 0 6 0 2 0 4 0 5 0 1 0 1	$1/945$
2 7 1	4 7 1 5 1 8 1 6 1 2 0 1 0 1	$1/945$
2 7 1	4 7 0 5 0 8 0 6 0 2 0 1 0 1	$-1/945$
2 7 1	1 5 1 6 1 3 1 4 0 8 0 1 0 1	$-1/945$
2 7 1	1 8 0 6 0 2 0 4 0 5 0 1 0 1	$1/945$
2 7 1	1 4 1 6 1 5 1 7 0 8 0 1 0 1	$-1/270$
2 7 1	1 8 0 5 0 2 0 6 0 7 0 1 0 1	$1/270$
2 7 1	1 6 1 7 1 8 0 3 0 4 0 1 0 1	$-1/90$
2 7 1	1 6 1 5 0 3 0 7 0 8 0 1 0 1	$1/90$
2 7 1	1 3 1 5 1 7 0 8 0 4 0 1 0 1	$-1/540$
2 7 1	1 5 1 7 0 6 0 8 0 3 0 1 0 1	$1/540$
2 7 1	4 6 1 7 1 8 0 3 0 1 0 1 0 1	$-1/108$
2 7 1	5 8 1 4 0 7 0 6 0 1 0 1 0 1	$-1/108$
2 7 1	1 7 1 8 1 2 0 3 0 4 0 1 0 1	$-1/540$
2 7 1	1 4 1 5 0 8 0 6 0 7 0 1 0 1	$1/540$
2 7 1	1 5 1 6 1 7 0 8 0 4 0 1 0 1	$-1/90$
2 7 1	1 7 1 5 0 3 0 8 0 2 0 1 0 1	$1/90$
2 7 1	1 4 1 5 1 6 0 7 0 8 0 1 0 1	$-1/540$
2 7 1	1 7 1 8 0 6 0 2 0 3 0 1 0 1	$1/540$
2 7 1	4 8 1 5 1 6 0 7 0 1 0 1 0 1	$-1/108$
2 7 1	5 8 1 7 0 3 0 6 0 1 0 1 0 1	$-1/108$
2 7 1	1 7 1 5 1 2 0 8 0 4 0 1 0 1	$-1/540$
2 7 1	1 5 1 8 0 3 0 6 0 7 0 1 0 1	$1/540$
2 7 1	1 5 1 2 1 6 0 4 0 3 0 1 0 1	$-1/960$
2 7 1	1 4 1 5 0 6 0 2 0 3 0 1 0 1	$1/960$
2 7 1	1 3 1 5 1 2 0 6 0 4 0 1 0 1	$1/960$
2 7 1	1 4 1 2 0 6 0 3 0 5 0 1 0 1	$-1/960$
2 7 1	3 8 4 7 1 5 1 2 0 1 0 1 0 1	$-41/4*\zeta(3)^2/\pi^6+163/11340$
2 7 1	3 8 4 7 0 5 0 2 0 1 0 1 0 1	$-41/4*\zeta(3)^2/\pi^6+163/11340$
2 7 1	5 7 1 8 1 2 1 6 0 3 0 1 0 1	$3/16*\zeta(3)^2/\pi^6+943/362880$
2 7 1	6 7 1 4 0 8 0 2 0 3 0 1 0 1	$-3/16*\zeta(3)^2/\pi^6-943/362880$
2 7 1	3 8 4 7 1 5 0 2 0 1 0 1 0 1	$-115/16*\zeta(3)^2/\pi^6+487/45360$
2 7 1	3 8 5 7 1 2 0 4 0 1 0 1 0 1	$-115/16*\zeta(3)^2/\pi^6+487/45360$
2 7 1	6 8 1 5 1 2 1 4 0 3 0 1 0 1	$-7*\zeta(3)^2/\pi^6+1963/181440$
2 7 1	3 8 1 4 0 6 0 2 0 5 0 1 0 1	$7*\zeta(3)^2/\pi^6-1963/181440$
2 7 1	6 8 1 5 1 2 1 7 0 3 0 1 0 1	$1/16*\zeta(3)^2/\pi^6+257/362880$
2 7 1	3 8 1 4 0 6 0 2 0 7 0 1 0 1	$-1/16*\zeta(3)^2/\pi^6-257/362880$
2 7 1	3 8 1 5 1 6 1 4 0 2 0 1 0 1	$13/2*\zeta(3)^2/\pi^6-31/2880$
2 7 1	4 8 1 2 0 6 0 3 0 5 0 1 0 1	$-13/2*\zeta(3)^2/\pi^6+31/2880$
2 7 1	1 7 1 5 1 6 1 8 0 3 0 4 0 1	$65/16*\zeta(3)^2/\pi^6-437/72576$
2 7 1	1 5 1 6 0 3 0 7 0 2 0 8 0 1	$65/16*\zeta(3)^2/\pi^6-437/72576$
2 7 1	6 7 4 5 1 8 1 2 1 3 0 1 0 1	$149/16*\zeta(3)^2/\pi^6-5239/362880$
2 7 1	6 7 4 5 0 8 0 2 0 3 0 1 0 1	$-149/16*\zeta(3)^2/\pi^6+5239/362880$

2 7 1	3 4 1 6 1 7 1 2 1 8 0 5 0 1	-99/16*zeta(3)^2/pi^6+53/5760
2 7 1	5 6 1 4 0 2 0 7 0 3 0 8 0 1	-99/16*zeta(3)^2/pi^6+53/5760
2 7 1	1 7 1 6 1 8 1 4 1 5 0 3 0 2	27/16*zeta(3)^2/pi^6-289/120960
2 7 1	1 4 1 5 0 3 0 8 0 2 0 6 0 7	-27/16*zeta(3)^2/pi^6+289/120960
2 7 1	3 7 1 5 1 8 1 4 0 2 0 1 0 1	-31/16*zeta(3)^2/pi^6+131/51840
2 7 1	4 7 1 2 0 6 0 8 0 5 0 1 0 1	31/16*zeta(3)^2/pi^6-131/51840
2 7 1	3 7 1 4 1 5 0 8 0 2 0 1 0 1	-7/8*zeta(3)^2/pi^6-11/45360
2 7 1	5 8 1 2 1 7 0 6 0 4 0 1 0 1	7/8*zeta(3)^2/pi^6+11/45360
2 7 1	6 7 1 8 1 3 0 2 0 4 0 1 0 1	3/16*zeta(3)^2/pi^6+173/362880
2 7 1	4 8 1 2 1 5 0 6 0 7 0 1 0 1	-3/16*zeta(3)^2/pi^6-173/362880
2 7 1	3 8 1 6 1 7 0 2 0 4 0 1 0 1	-2*zeta(3)^2/pi^6+17/6048
2 7 1	6 7 1 2 1 5 0 8 0 4 0 1 0 1	2*zeta(3)^2/pi^6-17/6048
2 7 1	5 8 1 2 1 6 0 4 0 3 0 1 0 1	-379/32*zeta(3)^2/pi^6+3463/181440
2 7 1	3 8 1 6 1 5 0 2 0 4 0 1 0 1	379/32*zeta(3)^2/pi^6-3463/181440
2 7 1	3 8 1 4 1 5 0 6 0 2 0 1 0 1	-151/32*zeta(3)^2/pi^6+191/25920
2 7 1	5 8 1 2 1 3 0 6 0 4 0 1 0 1	151/32*zeta(3)^2/pi^6-191/25920
2 7 1	5 8 1 6 1 3 0 4 0 2 0 1 0 1	-1/4*zeta(3)^2/pi^6+41/36288
2 7 1	3 8 1 5 1 2 0 6 0 4 0 1 0 1	1/4*zeta(3)^2/pi^6-41/36288
2 7 1	1 8 1 6 1 5 0 3 0 4 0 1 0 1	-1/216
2 7 1	1 6 1 5 0 8 0 2 0 3 0 1 0 1	1/216
2 7 1	1 3 1 6 1 2 0 8 0 4 0 1 0 1	1/270
2 7 1	1 8 1 4 0 6 0 3 0 5 0 1 0 1	-1/270
2 7 1	1 8 1 5 1 3 0 6 0 4 0 1 0 1	-1/1080
2 7 1	1 5 1 2 0 8 0 6 0 3 0 1 0 1	1/1080
2 7 1	4 7 1 8 1 5 1 6 0 1 0 1 0 1	1/270
2 7 1	4 7 0 8 0 5 0 6 0 1 0 1 0 1	1/270
2 7 1	4 8 1 5 1 6 1 2 1 7 0 1 0 1	1/945
2 7 1	4 8 0 6 0 5 0 7 0 2 0 1 0 1	-1/945
2 7 1	1 8 1 4 1 5 1 7 0 3 0 1 0 1	-1/270
2 7 1	1 4 0 8 0 5 0 6 0 7 0 1 0 1	1/270
2 7 1	1 6 1 7 1 5 1 3 1 4 0 2 0 1	-1/945
2 7 1	1 3 0 7 0 2 0 6 0 4 0 5 0 1	-1/945
2 7 1	3 4 1 6 1 5 1 2 1 8 0 1 0 1	1/945
2 7 1	3 4 0 6 0 5 0 8 0 2 0 1 0 1	1/945
2 7 1	1 6 1 7 1 5 1 3 1 4 0 8 0 1	-1/945
2 7 1	1 8 0 7 0 2 0 6 0 4 0 5 0 1	-1/945
2 7 1	1 8 1 4 1 5 1 6 0 7 0 1 0 1	-1/270
2 7 1	1 7 0 8 0 5 0 6 0 2 0 1 0 1	1/270
2 7 1	1 4 1 5 1 6 1 8 0 3 0 1 0 1	-1/945
2 7 1	1 4 0 6 0 5 0 8 0 2 0 1 0 1	1/945
2 7 1	5 6 1 7 1 8 0 4 0 1 0 1 0 1	7/1080
2 7 1	3 8 1 5 0 7 0 6 0 1 0 1 0 1	7/1080
2 7 1	1 7 1 8 1 6 0 4 0 3 0 1 0 1	-43/4320
2 7 1	1 6 1 5 0 7 0 8 0 3 0 1 0 1	43/4320
2 7 1	3 8 1 4 1 6 0 7 0 1 0 1 0 1	-1/270
2 7 1	4 8 1 7 0 5 0 6 0 1 0 1 0 1	-1/270
2 7 1	1 3 1 7 1 2 0 8 0 4 0 1 0 1	1/270
2 7 1	1 4 1 7 0 6 0 8 0 5 0 1 0 1	-1/270
2 7 1	1 8 1 4 1 5 0 6 0 7 0 1 0 1	-1/2160
2 7 1	1 7 1 2 0 8 0 6 0 3 0 1 0 1	1/2160
2 7 1	1 8 1 5 1 7 0 6 0 4 0 1 0 1	-19/4320
2 7 1	1 6 1 2 0 3 0 7 0 8 0 1 0 1	19/4320
2 7 1	3 8 1 5 1 6 0 7 0 1 0 1 0 1	-7/1080
2 7 1	4 8 1 7 0 3 0 6 0 1 0 1 0 1	-7/1080
2 7 1	1 7 1 5 1 3 0 8 0 4 0 1 0 1	-19/4320
2 7 1	1 5 1 7 0 8 0 6 0 3 0 1 0 1	19/4320
2 7 1	1 7 1 8 1 3 0 6 0 4 0 1 0 1	-1/2160
2 7 1	1 3 1 5 0 8 0 6 0 7 0 1 0 1	1/2160
2 7 1	4 5 1 7 1 8 0 6 0 1 0 1 0 1	-23/2160
2 7 1	3 5 1 8 0 7 0 6 0 1 0 1 0 1	23/2160
2 7 1	1 3 1 4 1 6 0 7 0 8 0 1 0 1	1/270
2 7 1	1 8 1 7 0 6 0 3 0 5 0 1 0 1	-1/270
2 7 1	1 7 1 5 1 6 0 4 0 8 0 1 0 1	-43/4320
2 7 1	1 7 1 6 0 3 0 8 0 2 0 1 0 1	43/4320
2 7 1	5 6 1 2 1 3 0 8 0 4 0 1 0 1	-59/32*zeta(3)^2/pi^6+779/362880
2 7 1	3 4 1 8 1 5 0 8 0 2 0 1 0 1	59/32*zeta(3)^2/pi^6-779/362880
2 7 1	3 6 1 4 1 5 0 2 0 8 0 1 0 1	-25/32*zeta(3)^2/pi^6+13/13440
2 7 1	3 5 1 8 1 2 0 6 0 4 0 1 0 1	-25/32*zeta(3)^2/pi^6+13/13440
2 7 1	3 5 1 6 1 2 0 8 0 4 0 1 0 1	-17/16*zeta(3)^2/pi^6-37/90720
2 7 1	3 6 1 8 1 5 0 2 0 4 0 1 0 1	-17/16*zeta(3)^2/pi^6-37/90720
2 7 1	4 8 1 6 1 5 1 7 0 2 0 1 0 1	-35/8*zeta(3)^2/pi^6+121/15120
2 7 1	5 8 1 2 0 3 0 6 0 7 0 1 0 1	35/8*zeta(3)^2/pi^6-121/15120
2 7 1	1 4 1 7 1 5 1 6 0 3 0 2 0 1	-33/2*zeta(3)^2/pi^6+103/4536

2 7 1	1 5 1 4 0 6 0 3 0 7 0 2 0 1	$-33/2*\zeta(3)^2/\pi^6+103/4536$
2 7 1	4 6 1 2 1 5 1 8 0 3 0 1 0 1	$11/8*\zeta(3)^2/\pi^6-53/30240$
2 7 1	3 5 1 4 0 2 0 6 0 8 0 1 0 1	$11/8*\zeta(3)^2/\pi^6-53/30240$
2 7 1	4 6 1 2 1 5 1 7 0 8 0 1 0 1	$-5/4*\zeta(3)^2/\pi^6+131/72576$
2 7 1	3 5 1 8 0 2 0 6 0 7 0 1 0 1	$-5/4*\zeta(3)^2/\pi^6+131/72576$
2 7 1	3 4 1 5 1 6 1 8 0 2 0 1 0 1	$-11/8*\zeta(3)^2/\pi^6+137/45360$
2 7 1	4 5 1 2 0 6 0 3 0 8 0 1 0 1	$11/8*\zeta(3)^2/\pi^6-137/45360$
2 7 1	1 7 1 4 1 5 1 6 0 8 0 3 0 1	$-27/16*\zeta(3)^2/\pi^6+43/120960$
2 7 1	1 8 1 5 0 3 0 6 0 7 0 2 0 1	$-27/16*\zeta(3)^2/\pi^6+43/120960$
2 7 1	5 6 1 2 1 7 1 8 0 4 0 1 0 1	$1/2*\zeta(3)^2/\pi^6+1/4480$
2 7 1	3 6 1 4 0 7 0 2 0 8 0 1 0 1	$1/2*\zeta(3)^2/\pi^6+1/4480$
2 7 1	1 7 1 5 1 8 1 6 0 4 0 3 0 1	$-17/16*\zeta(3)^2/\pi^6+9/4480$
2 7 1	1 5 1 6 0 2 0 7 0 8 0 3 0 1	$-17/16*\zeta(3)^2/\pi^6+9/4480$
2 7 1	4 6 1 5 1 8 1 2 0 3 0 1 0 1	$-17/16*\zeta(3)^2/\pi^6+131/181440$
2 7 1	3 5 1 4 0 6 0 8 0 2 0 1 0 1	$-17/16*\zeta(3)^2/\pi^6+131/181440$
2 7 1	3 4 1 8 1 5 1 6 0 2 0 1 0 1	$-35/16*\zeta(3)^2/\pi^6+37/12096$
2 7 1	4 5 1 2 0 8 0 6 0 3 0 1 0 1	$35/16*\zeta(3)^2/\pi^6-37/12096$
2 7 1	4 7 1 5 1 8 1 6 0 2 0 1 0 1	$1/4*\zeta(3)^2/\pi^6+1/12960$
2 7 1	5 7 1 2 0 6 0 8 0 3 0 1 0 1	$-1/4*\zeta(3)^2/\pi^6-1/12960$
2 7 1	1 5 1 8 1 7 1 6 0 4 0 2 0 1	$1/1440$
2 7 1	1 5 1 6 0 8 0 7 0 2 0 3 0 1	$1/1440$
2 7 1	4 7 5 8 1 6 1 2 1 3 0 1 0 1	$-49/16*\zeta(3)^2/\pi^6+1849/362880$
2 7 1	4 7 5 8 0 6 0 2 0 3 0 1 0 1	$49/16*\zeta(3)^2/\pi^6-1849/362880$
2 7 1	5 7 4 8 1 6 1 3 0 1 0 1 0 1	$-31/8*\zeta(3)^2/\pi^6+893/90720$
2 7 1	4 7 5 8 0 3 0 6 0 1 0 1 0 1	$-31/8*\zeta(3)^2/\pi^6+893/90720$
2 7 1	3 8 1 5 1 2 1 6 1 7 0 4 0 1	$143/16*\zeta(3)^2/\pi^6-53/4032$
2 7 1	5 8 1 4 0 2 0 6 0 7 0 3 0 1	$143/16*\zeta(3)^2/\pi^6-53/4032$
2 7 1	3 8 1 7 1 5 1 6 1 2 0 4 0 1	$71/16*\zeta(3)^2/\pi^6-1/160$
2 7 1	5 8 1 4 0 6 0 3 0 7 0 2 0 1	$71/16*\zeta(3)^2/\pi^6-1/160$
2 7 1	4 8 1 2 1 5 1 7 0 3 0 1 0 1	$53/8*\zeta(3)^2/\pi^6-163/18144$
2 7 1	5 8 1 4 0 2 0 6 0 7 0 1 0 1	$-53/8*\zeta(3)^2/\pi^6+163/18144$
2 7 1	3 8 4 7 1 5 1 6 0 1 0 1 0 1	$29/8*\zeta(3)^2/\pi^6-41/9072$
2 7 1	3 8 4 6 0 5 0 7 0 1 0 1 0 1	$29/8*\zeta(3)^2/\pi^6-41/9072$
2 7 1	3 4 1 8 1 5 1 7 0 2 0 1 0 1	$5/4*\zeta(3)^2/\pi^6-1019/362880$
2 7 1	4 5 1 2 0 8 0 6 0 7 0 1 0 1	$-5/4*\zeta(3)^2/\pi^6+1019/362880$
2 7 1	1 5 1 6 1 7 1 8 1 4 0 2 0 3	$-\zeta(3)^2/\pi^6+293/181440$
2 7 1	1 4 1 5 0 7 0 8 0 2 0 3 0 6	$\zeta(3)^2/\pi^6-293/181440$
2 7 1	3 4 1 6 1 8 1 2 1 7 0 5 0 1	$95/8*\zeta(3)^2/\pi^6-1301/72576$
2 7 1	5 6 1 4 0 2 0 7 0 8 0 3 0 1	$95/8*\zeta(3)^2/\pi^6-1301/72576$
2 7 1	4 7 1 5 1 8 1 2 0 3 0 1 0 1	$-5*\zeta(3)^2/\pi^6+71/10080$
2 7 1	5 7 1 4 0 6 0 8 0 2 0 1 0 1	$5*\zeta(3)^2/\pi^6-71/10080$
2 7 1	3 4 1 8 1 7 1 6 1 2 0 5 0 1	$179/16*\zeta(3)^2/\pi^6-2029/120960$
2 7 1	5 6 1 4 0 7 0 8 0 3 0 2 0 1	$179/16*\zeta(3)^2/\pi^6-2029/120960$
2 7 1	3 5 4 7 1 8 1 6 0 1 0 1 0 1	$-3/8*\zeta(3)^2/\pi^6-109/60480$
2 7 1	3 5 4 7 0 8 0 6 0 1 0 1 0 1	$-3/8*\zeta(3)^2/\pi^6-109/60480$
2 7 1	3 8 1 4 1 7 0 6 0 2 0 1 0 1	$9/4*\zeta(3)^2/\pi^6-29/11340$
2 7 1	5 8 1 4 1 2 0 6 0 7 0 1 0 1	$-9/4*\zeta(3)^2/\pi^6+29/11340$
2 7 1	3 6 1 4 1 7 0 2 0 8 0 1 0 1	$1/4*\zeta(3)^2/\pi^6-11/13440$
2 7 1	4 5 1 2 1 8 0 6 0 7 0 1 0 1	$1/4*\zeta(3)^2/\pi^6-11/13440$
2 7 1	6 7 1 2 1 3 0 4 0 8 0 1 0 1	$-57/16*\zeta(3)^2/\pi^6+85/18144$
2 7 1	3 7 1 8 1 5 0 6 0 2 0 1 0 1	$57/16*\zeta(3)^2/\pi^6-85/18144$
2 7 1	5 7 4 8 1 6 0 3 0 1 0 1 0 1	$1/8*\zeta(3)^2/\pi^6+331/90720$
2 7 1	4 7 5 8 1 3 0 6 0 1 0 1 0 1	$1/8*\zeta(3)^2/\pi^6+331/90720$
2 7 1	6 7 1 8 1 2 0 4 0 3 0 1 0 1	$17/4*\zeta(3)^2/\pi^6-121/25920$
2 7 1	4 7 1 6 1 5 0 8 0 2 0 1 0 1	$-17/4*\zeta(3)^2/\pi^6+121/25920$
2 7 1	3 6 1 8 1 7 0 2 0 4 0 1 0 1	$5/4*\zeta(3)^2/\pi^6-227/51840$
2 7 1	4 6 1 2 1 5 0 7 0 8 0 1 0 1	$5/4*\zeta(3)^2/\pi^6-227/51840$
2 7 1	3 7 1 8 1 6 0 4 0 2 0 1 0 1	$-35/16*\zeta(3)^2/\pi^6+11/7560$
2 7 1	5 7 1 6 1 2 0 8 0 4 0 1 0 1	$35/16*\zeta(3)^2/\pi^6-11/7560$
2 7 1	3 8 5 6 1 7 0 4 0 1 0 1 0 1	$1/2*\zeta(3)^2/\pi^6+353/181440$
2 7 1	3 8 4 6 1 5 0 7 0 1 0 1 0 1	$1/2*\zeta(3)^2/\pi^6+353/181440$
2 7 1	3 7 1 6 1 2 0 4 0 8 0 1 0 1	$17/4*\zeta(3)^2/\pi^6-1709/181440$
2 7 1	5 8 1 6 1 7 0 4 0 2 0 1 0 1	$-17/4*\zeta(3)^2/\pi^6+1709/181440$
2 7 1	3 4 1 7 1 5 0 8 0 2 0 1 0 1	$\zeta(3)^2/\pi^6-43/17280$
2 7 1	5 6 1 2 1 7 0 8 0 4 0 1 0 1	$-\zeta(3)^2/\pi^6+43/17280$
2 7 1	3 7 1 8 1 2 0 6 0 4 0 1 0 1	$-15/8*\zeta(3)^2/\pi^6+13/5040$
2 7 1	6 7 1 4 1 5 0 2 0 8 0 1 0 1	$15/8*\zeta(3)^2/\pi^6-13/5040$
2 7 1	3 5 4 6 1 8 0 7 0 1 0 1 0 1	$3/16*\zeta(3)^2/\pi^6-23/13440$
2 7 1	3 4 5 6 1 8 0 7 0 1 0 1 0 1	$3/16*\zeta(3)^2/\pi^6-23/13440$
2 7 1	6 7 1 5 1 8 1 2 0 4 0 1 0 1	$-3/4*\zeta(3)^2/\pi^6+7/2592$
2 7 1	3 7 1 5 0 6 0 8 0 2 0 1 0 1	$3/4*\zeta(3)^2/\pi^6-7/2592$
2 7 1	6 7 1 8 1 5 1 2 0 4 0 1 0 1	$1/4320$
2 7 1	3 7 1 5 0 8 0 6 0 2 0 1 0 1	$-1/4320$

2 7 1	4 7 1 8 1 5 1 6 0 2 0 1 0 1	1/2160
2 7 1	5 7 1 2 0 8 0 6 0 3 0 1 0 1	-1/2160
2 7 1	1 5 1 8 1 7 1 6 0 4 0 3 0 1	-7/4*zeta(3)^2/pi^6+17/6720
2 7 1	1 5 1 6 0 7 0 8 0 2 0 3 0 1	-7/4*zeta(3)^2/pi^6+17/6720
2 7 1	4 6 1 8 1 5 1 2 0 3 0 1 0 1	-1/420
2 7 1	3 5 1 4 0 8 0 6 0 2 0 1 0 1	-1/420
2 7 1	3 4 1 5 1 7 1 6 1 2 0 8 0 1	-1/1890
2 7 1	4 5 1 8 0 6 0 3 0 7 0 2 0 1	-1/1890
2 7 1	4 7 1 8 1 5 1 2 0 3 0 1 0 1	-1/540
2 7 1	5 7 1 4 0 8 0 6 0 2 0 1 0 1	1/540
2 7 1	6 7 1 2 1 3 0 8 0 4 0 1 0 1	-1/4320
2 7 1	4 7 1 8 1 5 0 6 0 2 0 1 0 1	1/4320
2 7 1	3 7 1 4 1 6 0 8 0 2 0 1 0 1	-1/2160
2 7 1	5 7 1 8 1 2 0 6 0 4 0 1 0 1	1/2160
2 7 1	3 7 1 6 1 2 0 8 0 4 0 1 0 1	-1/720
2 7 1	6 7 1 8 1 5 0 2 0 4 0 1 0 1	1/720
2 7 1	3 5 1 6 1 2 0 4 0 8 0 1 0 1	11/4*zeta(3)^2/pi^6-31/5040
2 7 1	3 5 1 6 1 8 0 4 0 2 0 1 0 1	11/4*zeta(3)^2/pi^6-31/5040
2 7 1	3 4 1 5 1 6 0 8 0 2 0 1 0 1	-2*zeta(3)^2/pi^6+299/90720
2 7 1	5 6 1 8 1 2 0 3 0 4 0 1 0 1	2*zeta(3)^2/pi^6-299/90720
2 7 1	3 5 1 4 1 2 0 6 0 8 0 1 0 1	3/4*zeta(3)^2/pi^6-53/90720
2 7 1	3 5 1 4 1 8 0 6 0 2 0 1 0 1	3/4*zeta(3)^2/pi^6-53/90720
2 7 1	3 6 1 5 1 2 1 4 0 8 0 1 0 1	-1/1890
2 7 1	3 4 1 8 0 6 0 2 0 5 0 1 0 1	-1/1890
2 7 1	1 7 1 4 1 5 1 6 0 3 0 8 0 1	-1/540
2 7 1	1 8 1 4 0 6 0 2 0 7 0 3 0 1	-1/540
2 7 1	6 8 1 5 1 2 1 4 0 7 0 1 0 1	1/2*zeta(3)^2/pi^6-1/45360
2 7 1	3 8 1 7 0 6 0 2 0 5 0 1 0 1	-1/2*zeta(3)^2/pi^6+1/45360
2 7 1	1 4 1 7 1 5 1 6 0 3 0 8 0 1	1/2*zeta(3)^2/pi^6-89/30240
2 7 1	1 8 1 5 0 6 0 2 0 7 0 3 0 1	1/2*zeta(3)^2/pi^6-89/30240
2 7 1	6 7 1 8 1 2 0 3 0 4 0 1 0 1	1/864
2 7 1	4 7 1 5 1 6 0 8 0 2 0 1 0 1	-1/864
2 7 1	3 8 1 6 1 7 0 4 0 2 0 1 0 1	-1/864
2 7 1	5 7 1 6 1 2 0 4 0 8 0 1 0 1	1/864
2 7 1	3 8 4 5 1 7 1 2 0 1 0 1 0 1	37/16*zeta(3)^2/pi^6-701/181440
2 7 1	4 5 2 8 0 3 0 7 0 1 0 1 0 1	-37/16*zeta(3)^2/pi^6+701/181440
2 7 1	3 5 4 8 1 2 1 7 0 1 0 1 0 1	-1/16*zeta(3)^2/pi^6+43/181440
2 7 1	3 5 4 8 0 2 0 7 0 1 0 1 0 1	-1/16*zeta(3)^2/pi^6+43/181440
2 7 1	4 8 1 2 1 5 1 6 0 7 0 1 0 1	-45/16*zeta(3)^2/pi^6+1723/362880
2 7 1	5 8 1 7 0 2 0 6 0 3 0 1 0 1	45/16*zeta(3)^2/pi^6-1723/362880
2 7 1	4 5 1 2 1 8 1 6 0 7 0 1 0 1	-199/362880
2 7 1	5 6 1 7 0 2 0 8 0 3 0 1 0 1	199/362880
2 7 1	4 5 2 7 1 8 0 3 0 1 0 1 0 1	-11/32*zeta(3)^2/pi^6+811/362880
2 7 1	4 5 2 7 1 3 0 8 0 1 0 1 0 1	11/32*zeta(3)^2/pi^6-811/362880
2 7 1	3 4 5 7 1 8 0 2 0 1 0 1 0 1	-19/32*zeta(3)^2/pi^6-307/362880
2 7 1	3 5 4 7 1 2 0 8 0 1 0 1 0 1	-19/32*zeta(3)^2/pi^6-307/362880
2 7 1	5 7 1 8 1 2 1 6 0 4 0 1 0 1	1/720
2 7 1	6 7 1 5 0 8 0 2 0 3 0 1 0 1	-1/720
2 7 1	4 5 1 2 1 8 1 6 0 3 0 1 0 1	4*zeta(3)^2/pi^6-43/7560
2 7 1	5 6 1 4 0 2 0 8 0 3 0 1 0 1	-4*zeta(3)^2/pi^6+43/7560
2 7 1	4 8 1 5 1 6 1 2 0 3 0 1 0 1	287/16*zeta(3)^2/pi^6-1013/36288
2 7 1	5 8 1 4 0 6 0 3 0 2 0 1 0 1	-287/16*zeta(3)^2/pi^6+1013/36288
2 7 1	3 8 4 6 1 7 1 2 0 1 0 1 0 1	13/8*zeta(3)^2/pi^6-1/45360
2 7 1	3 8 5 6 0 2 0 7 0 1 0 1 0 1	13/8*zeta(3)^2/pi^6-1/45360
2 7 1	4 8 1 2 1 5 1 6 0 3 0 1 0 1	-273/16*zeta(3)^2/pi^6+4703/181440
2 7 1	5 8 1 4 0 2 0 6 0 3 0 1 0 1	273/16*zeta(3)^2/pi^6-4703/181440
2 7 1	5 7 1 6 1 2 1 8 0 4 0 1 0 1	23/4*zeta(3)^2/pi^6-29/4536
2 7 1	6 7 1 5 0 3 0 2 0 8 0 1 0 1	-23/4*zeta(3)^2/pi^6+29/4536
2 7 1	3 8 5 6 1 2 0 7 0 1 0 1 0 1	7/16*zeta(3)^2/pi^6+29/15120
2 7 1	3 8 4 6 1 7 0 2 0 1 0 1 0 1	7/16*zeta(3)^2/pi^6+29/15120
2 7 1	4 8 1 5 1 6 1 2 0 7 0 1 0 1	3/4*zeta(3)^2/pi^6-17/7560
2 7 1	5 8 1 7 0 6 0 3 0 2 0 1 0 1	-3/4*zeta(3)^2/pi^6+17/7560
2 7 1	3 8 5 6 1 7 1 2 0 1 0 1 0 1	1/864
2 7 1	3 8 4 6 0 2 0 7 0 1 0 1 0 1	1/864
2 7 1	4 8 1 6 1 5 1 2 0 7 0 1 0 1	-1/540
2 7 1	5 8 1 7 0 3 0 6 0 2 0 1 0 1	1/540
2 7 1	3 4 1 5 1 6 1 2 0 8 0 1 0 1	-1/1890
2 7 1	4 5 1 8 0 6 0 3 0 2 0 1 0 1	1/1890
2 7 1	3 8 5 6 1 7 0 2 0 1 0 1 0 1	-1/864
2 7 1	3 8 4 6 1 2 0 7 0 1 0 1 0 1	-1/864
2 7 1	6 7 5 8 1 3 1 2 0 4 0 1 0 1	443/32*zeta(3)^2/pi^6-7627/362880
2 7 1	4 8 6 7 1 5 0 3 0 2 0 1 0 1	-443/32*zeta(3)^2/pi^6+7627/362880
2 7 1	4 6 5 7 3 8 1 2 0 1 0 1 0 1	39/16*zeta(3)^2/pi^6-479/181440

2 7 1	4 6 5 7 3 8 0 2 0 1 0 1 0 1	$39/16*\zeta(3)^2/\pi^6-479/181440$
2 7 1	5 7 4 8 1 6 1 3 0 2 0 1 0 1	$-569/32*\zeta(3)^2/\pi^6+827/30240$
2 7 1	6 8 5 7 1 2 0 4 0 3 0 1 0 1	$569/32*\zeta(3)^2/\pi^6-827/30240$
2 7 1	3 8 1 7 1 2 1 6 0 4 0 5 0 1	$319/16*\zeta(3)^2/\pi^6-97/3360$
2 7 1	5 8 1 6 1 7 0 4 0 2 0 3 0 1	$319/16*\zeta(3)^2/\pi^6-97/3360$
2 7 1	5 7 6 8 1 3 1 4 0 2 0 1 0 1	$-21/8*\zeta(3)^2/\pi^6+493/120960$
2 7 1	5 8 4 7 1 2 0 6 0 3 0 1 0 1	$21/8*\zeta(3)^2/\pi^6-493/120960$
2 7 1	5 8 4 6 1 7 1 3 0 2 0 1 0 1	$143/32*\zeta(3)^2/\pi^6-4901/725760$
2 7 1	6 7 4 5 1 2 0 8 0 3 0 1 0 1	$143/32*\zeta(3)^2/\pi^6-4901/725760$
2 7 1	6 8 4 5 1 7 1 2 0 3 0 1 0 1	$203/32*\zeta(3)^2/\pi^6-7157/725760$
2 7 1	5 6 4 7 1 2 0 8 0 3 0 1 0 1	$-203/32*\zeta(3)^2/\pi^6+7157/725760$
2 7 1	1 8 1 5 1 7 1 6 0 2 0 3 0 4	$29/16*\zeta(3)^2/\pi^6-281/120960$
2 7 1	1 6 1 5 1 7 0 8 0 3 0 2 0 4	$-29/16*\zeta(3)^2/\pi^6+281/120960$
2 7 1	4 8 5 6 1 3 1 2 0 7 0 1 0 1	$29/16*\zeta(3)^2/\pi^6-53/20736$
2 7 1	5 8 4 6 1 7 0 3 0 2 0 1 0 1	$29/16*\zeta(3)^2/\pi^6-53/20736$
2 7 1	4 5 1 2 1 7 1 6 0 8 0 3 0 1	$1/4*\zeta(3)^2/\pi^6-481/725760$
2 7 1	6 7 1 8 1 5 0 2 0 4 0 3 0 1	$1/4*\zeta(3)^2/\pi^6-481/725760$
2 7 1	1 4 1 8 1 5 1 7 0 3 0 2 0 1	$-1/540$
2 7 1	1 5 1 4 0 6 0 8 0 7 0 3 0 1	$-1/540$
2 7 1	1 8 1 7 1 6 0 2 0 3 0 4 0 1	$-1/432$
2 7 1	1 5 1 7 1 6 0 8 0 3 0 4 0 1	$-1/432$
2 7 1	1 5 1 2 1 8 0 7 0 4 0 3 0 1	$-1/2160$
2 7 1	1 6 1 5 1 3 0 7 0 8 0 4 0 1	$-1/2160$
2 7 1	1 7 1 6 1 5 0 3 0 2 0 4 0 1	$-105/16*\zeta(3)^2/\pi^6+731/90720$
2 7 1	1 3 1 5 1 2 0 7 0 4 0 6 0 1	$-31/16*\zeta(3)^2/\pi^6+47/18144$
2 7 1	1 6 1 2 1 5 0 7 0 4 0 3 0 1	$17/4*\zeta(3)^2/\pi^6-643/90720$
2 7 1	1 5 1 2 1 6 0 7 0 3 0 4 0 1	$17/4*\zeta(3)^2/\pi^6-643/90720$
2 7 1	3 7 1 6 1 8 1 4 1 5 0 1 0 1	$1/945$
2 7 1	3 7 0 6 0 8 0 4 0 5 0 1 0 1	$-1/945$
2 7 1	1 6 1 8 1 5 1 3 1 4 0 2 0 1	$-1/945$
2 7 1	1 3 0 7 0 8 0 6 0 4 0 5 0 1	$-1/945$
2 7 1	4 5 2 7 1 8 1 6 1 3 0 1 0 1	$-55/8*\zeta(3)^2/\pi^6+1021/103680$
2 7 1	4 5 2 7 0 8 0 6 0 3 0 1 0 1	$55/8*\zeta(3)^2/\pi^6-1021/103680$
2 7 1	5 7 2 4 1 8 1 6 1 3 0 1 0 1	$57/16*\zeta(3)^2/\pi^6-3659/725760$
2 7 1	5 7 2 4 0 8 0 6 0 3 0 1 0 1	$-57/16*\zeta(3)^2/\pi^6+3659/725760$
2 7 1	4 7 1 8 1 5 1 6 0 3 0 1 0 1	$5/8*\zeta(3)^2/\pi^6+199/90720$
2 7 1	5 7 1 4 0 8 0 6 0 3 0 1 0 1	$-5/8*\zeta(3)^2/\pi^6-199/90720$
2 7 1	3 5 1 6 1 7 1 8 0 4 0 1 0 1	$1/16*\zeta(3)^2/\pi^6+41/181440$
2 7 1	5 6 1 4 0 8 0 7 0 3 0 1 0 1	$1/16*\zeta(3)^2/\pi^6+41/181440$
2 7 1	7 8 1 6 1 2 1 4 1 5 0 3 0 1	$-7/16*\zeta(3)^2/\pi^6+263/362880$
2 7 1	3 8 1 4 0 7 0 2 0 5 0 6 0 1	$-7/16*\zeta(3)^2/\pi^6+263/362880$
2 7 1	6 7 1 5 1 8 1 4 0 3 0 1 0 1	$2*\zeta(3)^2/\pi^6-37/22680$
2 7 1	3 7 1 4 0 6 0 8 0 5 0 1 0 1	$-2*\zeta(3)^2/\pi^6+37/22680$
2 7 1	3 8 1 6 1 7 1 4 1 5 0 2 0 1	$-25/8*\zeta(3)^2/\pi^6+25/5184$
2 7 1	4 8 1 2 0 7 0 3 0 5 0 6 0 1	$-25/8*\zeta(3)^2/\pi^6+25/5184$
2 7 1	1 4 1 7 1 5 1 8 0 3 0 2 0 1	$53/16*\zeta(3)^2/\pi^6-853/120960$
2 7 1	1 5 1 4 0 6 0 3 0 7 0 8 0 1	$53/16*\zeta(3)^2/\pi^6-853/120960$
2 7 1	3 7 1 5 1 6 1 8 1 2 0 4 0 1	$3*\zeta(3)^2/\pi^6-3131/725760$
2 7 1	3 5 1 4 0 7 0 6 0 8 0 2 0 1	$-3*\zeta(3)^2/\pi^6+3131/725760$
2 7 1	3 4 1 6 1 5 1 8 1 7 0 2 0 1	$-47/16*\zeta(3)^2/\pi^6+3229/725760$
2 7 1	4 5 1 2 0 7 0 6 0 8 0 3 0 1	$-47/16*\zeta(3)^2/\pi^6+3229/725760$
2 7 1	4 6 1 8 1 5 1 7 0 3 0 1 0 1	$67/45360$
2 7 1	3 5 1 4 0 8 0 6 0 7 0 1 0 1	$67/45360$
2 7 1	1 4 1 8 1 5 1 7 0 2 0 3 0 1	$21/16*\zeta(3)^2/\pi^6-1483/362880$
2 7 1	1 5 1 4 0 8 0 6 0 7 0 3 0 1	$21/16*\zeta(3)^2/\pi^6-1483/362880$
2 7 1	1 4 1 8 1 5 1 6 0 7 0 3 0 1	$1/2*\zeta(3)^2/\pi^6-157/90720$
2 7 1	1 5 1 2 0 6 0 8 0 7 0 3 0 1	$1/2*\zeta(3)^2/\pi^6-157/90720$
2 7 1	4 8 1 5 1 6 1 7 0 3 0 1 0 1	$-19/8*\zeta(3)^2/\pi^6+199/45360$
2 7 1	5 8 1 4 0 6 0 3 0 7 0 1 0 1	$19/8*\zeta(3)^2/\pi^6-199/45360$
2 7 1	4 8 2 7 1 6 1 3 1 5 0 1 0 1	$3*\zeta(3)^2/\pi^6-7/1728$
2 7 1	4 8 2 7 0 6 0 3 0 5 0 1 0 1	$-3*\zeta(3)^2/\pi^6+7/1728$
2 7 1	4 6 1 5 1 8 1 7 0 3 0 1 0 1	$-7/16*\zeta(3)^2/\pi^6+31/45360$
2 7 1	3 5 1 4 0 6 0 8 0 7 0 1 0 1	$-7/16*\zeta(3)^2/\pi^6+31/45360$
2 7 1	3 7 1 8 1 5 1 6 1 2 0 4 0 1	$-27/8*\zeta(3)^2/\pi^6+3961/725760$
2 7 1	3 5 1 4 0 6 0 8 0 7 0 2 0 1	$27/8*\zeta(3)^2/\pi^6-3961/725760$
2 7 1	1 5 1 4 1 8 1 7 0 2 0 3 0 1	$-65/16*\zeta(3)^2/\pi^6+2123/362880$
2 7 1	1 5 1 4 0 7 0 6 0 3 0 8 0 1	$-65/16*\zeta(3)^2/\pi^6+2123/362880$
2 7 1	3 4 1 5 1 8 1 6 1 7 0 2 0 1	$-37/8*\zeta(3)^2/\pi^6+5011/725760$
2 7 1	4 5 1 2 0 6 0 8 0 7 0 3 0 1	$-37/8*\zeta(3)^2/\pi^6+5011/725760$
2 7 1	1 5 1 7 1 3 1 4 0 2 0 6 0 1	$5*\zeta(3)^2/\pi^6-239/30240$
2 7 1	1 4 1 2 0 7 0 3 0 5 0 6 0 1	$5*\zeta(3)^2/\pi^6-239/30240$
2 7 1	1 7 1 4 1 5 1 8 0 3 0 6 0 1	$-35/16*\zeta(3)^2/\pi^6+37/17280$
2 7 1	1 5 1 2 0 3 0 6 0 7 0 8 0 1	$-35/16*\zeta(3)^2/\pi^6+37/17280$



2 7 1	5 8 2 4 1 6 1 3 1 7 0 1 0 1	-31/16*zeta(3)^2/pi^6+461/145152
2 7 1	5 8 2 4 0 6 0 3 0 7 0 1 0 1	31/16*zeta(3)^2/pi^6-461/145152
2 7 1	4 5 2 8 1 6 1 3 1 7 0 1 0 1	-29/8*zeta(3)^2/pi^6+733/145152
2 7 1	4 5 2 8 0 6 0 3 0 7 0 1 0 1	29/8*zeta(3)^2/pi^6-733/145152
2 7 1	3 4 1 5 1 6 1 7 0 8 0 1 0 1	3/4*zeta(3)^2/pi^6+1/1890
2 7 1	4 5 1 8 0 6 0 3 0 7 0 1 0 1	-3/4*zeta(3)^2/pi^6-1/1890
2 7 1	3 8 1 5 1 6 1 4 0 7 0 1 0 1	5/4*zeta(3)^2/pi^6-13/4320
2 7 1	4 8 1 7 0 6 0 3 0 5 0 1 0 1	-5/4*zeta(3)^2/pi^6+13/4320
2 7 1	3 7 1 5 1 2 1 6 1 8 0 4 0 1	-7/4*zeta(3)^2/pi^6+13/4480
2 7 1	3 5 1 4 0 2 0 6 0 7 0 8 0 1	7/4*zeta(3)^2/pi^6-13/4480
2 7 1	3 4 1 5 1 7 1 6 1 8 0 2 0 1	5/4*zeta(3)^2/pi^6-17/9072
2 7 1	4 5 1 2 0 6 0 3 0 7 0 8 0 1	5/4*zeta(3)^2/pi^6-17/9072
2 7 1	3 6 1 5 1 8 1 4 0 7 0 1 0 1	13/16*zeta(3)^2/pi^6-149/45360
2 7 1	3 4 1 7 0 6 0 8 0 5 0 1 0 1	13/16*zeta(3)^2/pi^6-149/45360
2 7 1	1 5 1 7 1 3 1 4 0 2 0 8 0 1	7/16*zeta(3)^2/pi^6-97/51840
2 7 1	1 4 1 8 0 7 0 3 0 5 0 6 0 1	7/16*zeta(3)^2/pi^6-97/51840
2 7 1	1 5 1 8 1 3 1 4 0 7 0 2 0 1	-5/8*zeta(3)^2/pi^6+7/8640
2 7 1	1 3 1 4 0 7 0 8 0 5 0 6 0 1	-5/8*zeta(3)^2/pi^6+7/8640
2 7 1	1 6 1 8 1 5 1 3 1 4 0 2 0 7	-11/16*zeta(3)^2/pi^6+281/362880
2 7 1	1 4 1 2 0 8 0 3 0 7 0 5 0 6	11/16*zeta(3)^2/pi^6-281/362880
2 7 1	4 6 2 5 1 3 1 7 1 8 0 1 0 1	3/16*zeta(3)^2/pi^6-71/362880
2 7 1	5 6 2 4 0 8 0 3 0 7 0 1 0 1	-3/16*zeta(3)^2/pi^6+71/362880
2 7 1	3 4 1 8 1 5 1 6 0 7 0 1 0 1	-9/16*zeta(3)^2/pi^6-11/8640
2 7 1	4 5 1 7 0 8 0 6 0 3 0 1 0 1	9/16*zeta(3)^2/pi^6+11/8640
2 7 1	1 8 1 6 1 7 0 4 0 2 0 3 0 1	25/8*zeta(3)^2/pi^6-131/12960
2 7 1	1 5 1 6 1 7 0 4 0 8 0 3 0 1	25/8*zeta(3)^2/pi^6-131/12960
2 7 1	1 5 1 2 1 8 0 7 0 3 0 4 0 1	-15/8*zeta(3)^2/pi^6+37/25920
2 7 1	1 5 1 6 1 3 0 7 0 8 0 4 0 1	-15/8*zeta(3)^2/pi^6+37/25920
2 7 1	1 8 1 2 1 6 0 4 0 7 0 3 0 1	-11/16*zeta(3)^2/pi^6+55/72576
2 7 1	1 6 1 5 1 3 0 7 0 4 0 8 0 1	-11/16*zeta(3)^2/pi^6+55/72576
2 7 1	3 8 1 4 1 5 0 6 0 7 0 1 0 1	-3/8*zeta(3)^2/pi^6-19/60480
2 7 1	5 7 1 8 1 3 0 6 0 4 0 1 0 1	3/8*zeta(3)^2/pi^6+19/60480
2 7 1	1 3 1 5 1 2 0 7 0 4 0 8 0 1	19/16*zeta(3)^2/pi^6-199/120960
2 7 1	1 5 1 8 1 3 0 7 0 4 0 6 0 1	19/16*zeta(3)^2/pi^6-199/120960
2 7 1	1 3 1 8 1 2 0 7 0 4 0 6 0 1	zeta(3)^2/pi^6-191/90720
2 7 1	1 4 1 5 1 3 0 7 0 8 0 6 0 1	zeta(3)^2/pi^6-191/90720
2 7 1	5 7 1 6 1 3 0 4 0 8 0 1 0 1	1/8*zeta(3)^2/pi^6+37/181440
2 7 1	3 8 1 5 1 7 0 6 0 4 0 1 0 1	-1/8*zeta(3)^2/pi^6-37/181440
2 7 1	3 5 1 6 1 7 0 4 0 8 0 1 0 1	-13/2520
2 7 1	1 6 1 2 1 7 0 4 0 8 0 3 0 1	7/4*zeta(3)^2/pi^6-163/36288
2 7 1	1 6 1 8 1 5 0 7 0 4 0 3 0 1	7/4*zeta(3)^2/pi^6-163/36288
2 7 1	1 6 1 2 1 8 0 7 0 4 0 3 0 1	3/8*zeta(3)^2/pi^6-61/45360
2 7 1	1 3 1 5 1 6 0 7 0 8 0 4 0 1	3/8*zeta(3)^2/pi^6-61/45360
2 7 1	3 7 1 6 1 5 0 8 0 4 0 1 0 1	3/8*zeta(3)^2/pi^6-149/30240
2 7 1	5 7 1 8 1 6 0 4 0 3 0 1 0 1	-3/8*zeta(3)^2/pi^6+149/30240
2 7 1	1 5 1 2 1 6 0 4 0 8 0 3 0 1	-5/2*zeta(3)^2/pi^6+13/7560
2 7 1	1 5 1 8 1 6 0 7 0 3 0 4 0 1	-5/2*zeta(3)^2/pi^6+13/7560
2 7 1	1 8 1 2 1 6 0 7 0 3 0 4 0 1	-2*zeta(3)^2/pi^6+59/20160
2 7 1	1 3 1 5 1 6 0 4 0 7 0 8 0 1	-2*zeta(3)^2/pi^6+59/20160
2 7 1	3 6 1 7 1 5 0 8 0 4 0 1 0 1	-1/8*zeta(3)^2/pi^6-703/120960
2 7 1	3 5 1 6 1 7 0 8 0 4 0 1 0 1	-1/8*zeta(3)^2/pi^6-703/120960
2 7 1	5 6 1 7 1 3 0 8 0 4 0 1 0 1	-1/16*zeta(3)^2/pi^6+47/90720
2 7 1	3 4 1 8 1 5 0 6 0 7 0 1 0 1	1/16*zeta(3)^2/pi^6-47/90720
2 7 1	1 3 1 6 1 2 0 7 0 8 0 4 0 1	-3/8*zeta(3)^2/pi^6+1/1440
2 7 1	1 3 1 5 1 8 0 7 0 4 0 6 0 1	-3/8*zeta(3)^2/pi^6+1/1440
2 7 1	3 6 1 4 1 5 0 7 0 8 0 1 0 1	-1/4*zeta(3)^2/pi^6-967/362880
2 7 1	3 5 1 8 1 7 0 6 0 4 0 1 0 1	-1/4*zeta(3)^2/pi^6-967/362880
2 7 1	3 8 4 7 1 6 1 2 0 5 0 1 0 1	57/16*zeta(3)^2/pi^6-599/120960
2 7 1	6 8 2 7 1 5 0 3 0 4 0 1 0 1	-57/16*zeta(3)^2/pi^6+599/120960
2 7 1	3 6 1 5 1 7 1 8 0 4 0 2 0 1	-7/8*zeta(3)^2/pi^6+1091/725760
2 7 1	3 5 1 6 1 2 0 7 0 4 0 8 0 1	7/8*zeta(3)^2/pi^6-1091/725760
2 7 1	6 8 1 5 1 7 1 4 0 3 0 2 0 1	65/8*zeta(3)^2/pi^6-4343/362880
2 7 1	3 8 1 5 1 2 0 7 0 4 0 6 0 1	65/8*zeta(3)^2/pi^6-4343/362880
2 7 1	6 7 1 5 1 8 1 2 0 3 0 4 0 1	-15/32*zeta(3)^2/pi^6+97/290304
2 7 1	3 4 1 5 1 6 0 7 0 8 0 2 0 1	-15/32*zeta(3)^2/pi^6+97/290304
2 7 1	4 7 1 8 1 5 1 6 0 2 0 3 0 1	-75/32*zeta(3)^2/pi^6+5129/1451520
2 7 1	3 5 1 6 1 2 0 7 0 8 0 4 0 1	75/32*zeta(3)^2/pi^6-5129/1451520
2 7 1	3 6 1 5 1 2 1 4 0 7 0 8 0 1	-1/4*zeta(3)^2/pi^6+13/45360
2 7 1	3 5 1 4 1 8 0 7 0 2 0 6 0 1	1/4*zeta(3)^2/pi^6-13/45360
2 7 1	6 8 1 5 1 2 1 4 0 7 0 3 0 1	-9/16*zeta(3)^2/pi^6+17/16128
2 7 1	3 8 1 4 1 5 0 7 0 2 0 6 0 1	-9/16*zeta(3)^2/pi^6+17/16128
2 7 1	4 6 1 2 1 5 1 8 0 7 0 3 0 1	45/32*zeta(3)^2/pi^6-3019/1451520
2 7 1	3 6 1 4 1 5 0 2 0 7 0 8 0 1	-45/32*zeta(3)^2/pi^6+3019/1451520

2 7 1	3 8 1 5 1 7 1 4 0 2 0 6 0 1	$5/16*\zeta(3)^2/\pi^6-379/725760$
2 7 1	5 8 1 2 1 3 0 7 0 4 0 6 0 1	$5/16*\zeta(3)^2/\pi^6-379/725760$
2 7 1	3 8 4 6 1 5 1 7 0 2 0 1 0 1	$9/32*\zeta(3)^2/\pi^6-205/290304$
2 7 1	3 8 4 5 1 2 0 6 0 7 0 1 0 1	$9/32*\zeta(3)^2/\pi^6-205/290304$
2 7 1	3 4 1 7 1 5 1 8 0 2 0 6 0 1	$25/32*\zeta(3)^2/\pi^6-181/161280$
2 7 1	5 6 1 2 1 3 0 7 0 4 0 8 0 1	$-25/32*\zeta(3)^2/\pi^6+181/161280$
2 7 1	3 5 6 7 1 8 1 4 0 2 0 1 0 1	$-1/32*\zeta(3)^2/\pi^6+169/483840$
2 7 1	4 8 2 5 1 3 0 6 0 7 0 1 0 1	$1/32*\zeta(3)^2/\pi^6-169/483840$
2 7 1	1 8 1 4 1 5 1 6 0 2 0 3 0 7	$-3/16*\zeta(3)^2/\pi^6-23/362880$
2 7 1	1 5 1 2 1 6 0 7 0 3 0 8 0 4	$3/16*\zeta(3)^2/\pi^6+23/362880$
2 7 1	1 8 1 4 1 5 1 6 0 7 0 2 0 3	$17/16*\zeta(3)^2/\pi^6-19/10368$
2 7 1	1 6 1 2 1 5 0 7 0 4 0 8 0 3	$-17/16*\zeta(3)^2/\pi^6+19/10368$
2 7 1	3 8 4 6 1 5 1 2 0 7 0 1 0 1	$-29/32*\zeta(3)^2/\pi^6+1573/1451520$
2 7 1	4 5 2 7 1 8 0 6 0 3 0 1 0 1	$-29/32*\zeta(3)^2/\pi^6+1573/1451520$
2 7 1	3 6 4 7 1 5 1 2 0 8 0 1 0 1	$-87/32*\zeta(3)^2/\pi^6+751/207360$
2 7 1	5 8 2 4 1 7 0 6 0 3 0 1 0 1	$87/32*\zeta(3)^2/\pi^6-751/207360$
2 7 1	3 8 5 6 1 7 1 2 0 4 0 1 0 1	$61/32*\zeta(3)^2/\pi^6-2939/1451520$
2 7 1	4 6 2 7 1 5 0 8 0 3 0 1 0 1	$61/32*\zeta(3)^2/\pi^6-2939/1451520$
2 7 1	4 6 1 8 1 7 1 2 0 5 0 3 0 1	$-23/16*\zeta(3)^2/\pi^6+451/145152$
2 7 1	3 6 1 7 1 5 0 8 0 4 0 2 0 1	$23/16*\zeta(3)^2/\pi^6-451/145152$
2 7 1	3 6 1 7 1 5 1 2 0 4 0 8 0 1	$9/8*\zeta(3)^2/\pi^6-2689/1451520$
2 7 1	3 5 1 6 1 8 0 4 0 7 0 2 0 1	$-9/8*\zeta(3)^2/\pi^6+2689/1451520$
2 7 1	3 4 1 7 1 5 1 6 0 2 0 8 0 1	$-5/8*\zeta(3)^2/\pi^6+271/290304$
2 7 1	5 6 1 8 1 2 0 7 0 3 0 4 0 1	$5/8*\zeta(3)^2/\pi^6-271/290304$
2 7 1	3 6 5 7 1 8 1 2 0 4 0 1 0 1	$-13/32*\zeta(3)^2/\pi^6+29/23040$
2 7 1	6 7 2 4 1 5 0 8 0 3 0 1 0 1	$13/32*\zeta(3)^2/\pi^6-29/23040$
2 7 1	4 5 1 8 1 7 1 6 0 2 0 3 0 1	$19/16*\zeta(3)^2/\pi^6-1243/725760$
2 7 1	6 7 1 2 1 5 0 8 0 4 0 3 0 1	$19/16*\zeta(3)^2/\pi^6-1243/725760$
2 7 1	3 5 4 7 1 2 1 6 0 8 0 1 0 1	$-3/8*\zeta(3)^2/\pi^6+19/32256$
2 7 1	6 8 2 5 1 7 0 4 0 3 0 1 0 1	$3/8*\zeta(3)^2/\pi^6-19/32256$
2 7 1	3 8 4 5 1 6 1 2 0 7 0 1 0 1	$3/8*\zeta(3)^2/\pi^6-1241/1451520$
2 7 1	5 6 2 7 1 8 0 3 0 4 0 1 0 1	$3/8*\zeta(3)^2/\pi^6-1241/1451520$
2 7 1	4 6 1 2 1 7 1 8 0 5 0 3 0 1	$45/16*\zeta(3)^2/\pi^6-899/241920$
2 7 1	3 6 1 7 1 5 0 2 0 4 0 8 0 1	$-45/16*\zeta(3)^2/\pi^6+899/241920$
2 7 1	1 5 1 4 1 7 1 6 0 3 0 2 0 1	$23/2*\zeta(3)^2/\pi^6-1583/90720$
2 7 1	1 5 1 4 0 7 0 6 0 3 0 2 0 1	$23/2*\zeta(3)^2/\pi^6-1583/90720$
2 7 1	3 8 1 6 1 5 1 2 1 7 0 4 0 1	$-39/4*\zeta(3)^2/\pi^6+1807/120960$
2 7 1	5 8 1 4 0 7 0 6 0 3 0 2 0 1	$-39/4*\zeta(3)^2/\pi^6+1807/120960$
2 7 1	3 6 1 7 1 2 0 4 0 8 0 1 0 1	$-3/2*\zeta(3)^2/\pi^6-479/362880$
2 7 1	3 6 1 7 1 5 0 2 0 8 0 1 0 1	$-3/2*\zeta(3)^2/\pi^6-479/362880$
2 7 1	4 6 1 5 1 8 1 2 0 7 0 1 0 1	$-409/181440$
2 7 1	3 5 1 7 0 6 0 8 0 2 0 1 0 1	$-409/181440$
2 7 1	5 6 1 2 1 7 1 8 0 3 0 1 0 1	$1/720$
2 7 1	3 6 1 5 0 7 0 2 0 8 0 1 0 1	$1/720$
2 7 1	3 5 1 6 1 7 1 8 0 2 0 1 0 1	$-1/4320$
2 7 1	5 6 1 2 0 8 0 7 0 3 0 1 0 1	$-1/4320$
2 7 1	1 5 1 4 1 7 1 6 0 8 0 2 0 1	$5/4*\zeta(3)^2/\pi^6-149/60480$
2 7 1	1 8 1 4 0 7 0 6 0 3 0 2 0 1	$5/4*\zeta(3)^2/\pi^6-149/60480$
2 7 1	4 6 1 8 1 5 1 2 0 7 0 1 0 1	$-1/540$
2 7 1	3 5 1 7 0 8 0 6 0 2 0 1 0 1	$-1/540$
2 7 1	3 4 1 6 1 5 1 2 1 7 0 8 0 1	$1/1890$
2 7 1	4 5 1 8 0 7 0 6 0 3 0 2 0 1	$-1/1890$
2 7 1	3 6 1 7 1 2 0 8 0 4 0 1 0 1	$-1/720$
2 7 1	4 6 1 7 1 5 0 2 0 8 0 1 0 1	$-1/720$
2 7 1	3 4 1 7 1 6 0 8 0 2 0 1 0 1	$-1/4320$
2 7 1	5 6 1 7 1 2 0 8 0 4 0 1 0 1	$1/4320$
2 7 1	3 6 1 4 1 2 0 7 0 8 0 1 0 1	$1/540$
2 7 1	4 5 1 7 1 8 0 6 0 2 0 1 0 1	$1/540$
2 7 1	1 5 1 4 1 8 1 6 0 7 0 3 0 1	$1/2*\zeta(3)^2/\pi^6-29/36288$
2 7 1	1 5 1 2 0 6 0 7 0 3 0 8 0 1	$1/2*\zeta(3)^2/\pi^6-29/36288$
2 7 1	1 4 1 8 1 5 1 6 0 7 0 2 0 1	$-1/1440$
2 7 1	1 5 1 2 0 8 0 6 0 7 0 3 0 1	$-1/1440$
2 7 1	1 5 1 2 1 7 0 4 0 8 0 3 0 1	$-1/1440$
2 7 1	1 8 1 5 1 6 0 7 0 3 0 4 0 1	$-1/1440$
2 7 1	1 3 1 5 1 2 0 7 0 8 0 4 0 1	$1/1440$
2 7 1	1 5 1 8 1 2 0 7 0 4 0 6 0 1	$1/1440$
2 7 1	3 5 4 8 1 6 1 7 0 2 0 1 0 1	$-11/16*\zeta(3)^2/\pi^6+1349/1451520$
2 7 1	3 6 5 7 1 2 0 4 0 8 0 1 0 1	$11/16*\zeta(3)^2/\pi^6-1349/1451520$
2 7 1	4 8 1 5 1 7 1 6 0 2 0 3 0 1	$-199/16*\zeta(3)^2/\pi^6+1961/103680$
2 7 1	6 8 1 2 1 5 0 7 0 4 0 3 0 1	$-199/16*\zeta(3)^2/\pi^6+1961/103680$
2 7 1	3 8 4 6 1 7 1 2 0 5 0 1 0 1	$-45/8*\zeta(3)^2/\pi^6+4237/483840$
2 7 1	4 6 2 7 1 5 0 3 0 8 0 1 0 1	$-45/8*\zeta(3)^2/\pi^6+4237/483840$
2 7 1	4 6 1 2 1 8 1 7 0 5 0 3 0 1	$-197/32*\zeta(3)^2/\pi^6+14659/1451520$

2 7 1	3 6 1 7 1 5 0 2 0 8 0 4 0 1	197/32*zeta(3)^2/pi^6-14659/1451520
2 7 1	7 8 1 5 1 2 1 6 0 4 0 3 0 1	-37/4*zeta(3)^2/pi^6+10153/725760
2 7 1	3 8 1 6 1 5 0 2 0 7 0 4 0 1	-37/4*zeta(3)^2/pi^6+10153/725760
2 7 1	3 4 1 7 1 8 1 6 0 2 0 5 0 1	-187/32*zeta(3)^2/pi^6+2495/290304
2 7 1	5 6 1 2 1 7 0 4 0 8 0 3 0 1	-187/32*zeta(3)^2/pi^6+2495/290304
2 7 1	3 8 6 7 1 5 1 2 0 4 0 1 0 1	-175/32*zeta(3)^2/pi^6+593/72576
2 7 1	4 8 2 7 1 5 0 6 0 3 0 1 0 1	175/32*zeta(3)^2/pi^6-593/72576
2 7 1	3 8 4 7 1 5 1 6 0 2 0 1 0 1	289/32*zeta(3)^2/pi^6-707/51840
2 7 1	3 8 5 7 1 2 0 6 0 4 0 1 0 1	-289/32*zeta(3)^2/pi^6+707/51840
2 7 1	3 5 6 8 1 2 1 7 0 4 0 1 0 1	153/32*zeta(3)^2/pi^6-9907/1451520
2 7 1	4 7 2 5 1 6 0 8 0 3 0 1 0 1	-153/32*zeta(3)^2/pi^6+9907/1451520
2 7 1	6 8 1 5 1 7 1 2 0 4 0 3 0 1	71/16*zeta(3)^2/pi^6-4519/725760
2 7 1	3 8 1 6 1 5 0 7 0 4 0 2 0 1	71/16*zeta(3)^2/pi^6-4519/725760
2 7 1	3 8 4 5 1 6 1 7 0 2 0 1 0 1	105/32*zeta(3)^2/pi^6-6407/1451520
2 7 1	3 7 5 6 1 2 0 8 0 4 0 1 0 1	105/32*zeta(3)^2/pi^6-6407/1451520
2 7 1	3 8 1 5 1 7 1 6 0 4 0 2 0 1	-15/16*zeta(3)^2/pi^6+479/241920
2 7 1	5 8 1 2 1 6 0 7 0 3 0 4 0 1	-15/16*zeta(3)^2/pi^6+479/241920
2 7 1	3 6 4 5 1 8 1 2 0 7 0 1 0 1	45/32*zeta(3)^2/pi^6-37/15120
2 7 1	5 6 2 4 1 7 0 8 0 3 0 1 0 1	-45/32*zeta(3)^2/pi^6+37/15120
2 7 1	3 4 1 7 1 5 1 6 0 8 0 2 0 1	-17/16*zeta(3)^2/pi^6+787/725760
2 7 1	5 6 1 8 1 2 0 7 0 4 0 3 0 1	17/16*zeta(3)^2/pi^6-787/725760
2 7 1	4 7 1 2 1 5 1 6 0 8 0 3 0 1	-29/16*zeta(3)^2/pi^6+2243/725760
2 7 1	3 5 1 6 1 8 0 7 0 2 0 4 0 1	29/16*zeta(3)^2/pi^6-2243/725760
2 7 1	3 5 4 6 1 2 1 7 0 8 0 1 0 1	15/32*zeta(3)^2/pi^6-17/40320
2 7 1	4 6 2 5 1 7 0 8 0 3 0 1 0 1	15/32*zeta(3)^2/pi^6-17/40320
2 7 1	3 5 4 6 1 8 1 7 0 2 0 1 0 1	1/8*zeta(3)^2/pi^6-299/362880
2 7 1	3 6 4 5 1 2 0 7 0 8 0 1 0 1	1/8*zeta(3)^2/pi^6-299/362880
2 7 1	1 5 1 4 1 6 1 7 0 8 0 3 0 2	-5/2*zeta(3)^2/pi^6+347/90720
2 7 1	1 6 1 2 1 5 0 8 0 7 0 4 0 3	5/2*zeta(3)^2/pi^6-347/90720
2 7 1	3 4 1 5 1 8 1 6 0 7 0 2 0 1	113/32*zeta(3)^2/pi^6-7801/1451520
2 7 1	5 6 1 2 1 3 0 7 0 8 0 4 0 1	113/32*zeta(3)^2/pi^6-7801/1451520
2 7 1	4 6 1 5 1 8 1 2 0 7 0 3 0 1	55/32*zeta(3)^2/pi^6-1303/483840
2 7 1	3 6 1 4 1 5 0 7 0 8 0 2 0 1	-55/32*zeta(3)^2/pi^6+1303/483840
2 7 1	3 6 1 5 1 7 1 2 0 4 0 8 0 1	1/4*zeta(3)^2/pi^6-89/60480
2 7 1	3 5 1 6 1 8 0 7 0 4 0 2 0 1	-1/4*zeta(3)^2/pi^6+89/60480
2 7 1	3 5 6 7 1 2 1 4 0 8 0 1 0 1	-1/4*zeta(3)^2/pi^6+1/90720
2 7 1	4 8 2 5 1 7 0 6 0 3 0 1 0 1	1/4*zeta(3)^2/pi^6-1/90720
2 7 1	4 6 1 8 1 5 1 2 0 7 0 3 0 1	1/4*zeta(3)^2/pi^6-157/181440
2 7 1	3 6 1 4 1 5 0 8 0 7 0 2 0 1	-1/4*zeta(3)^2/pi^6+157/181440
2 7 1	4 7 1 5 1 8 1 6 0 2 0 3 0 1	41/16*zeta(3)^2/pi^6-313/90720
2 7 1	3 5 1 6 1 2 0 8 0 7 0 4 0 1	-41/16*zeta(3)^2/pi^6+313/90720
2 7 1	3 6 4 7 1 8 1 2 0 5 0 1 0 1	21/8*zeta(3)^2/pi^6-23/6480
2 7 1	6 7 2 4 1 5 0 3 0 8 0 1 0 1	-21/8*zeta(3)^2/pi^6+23/6480
2 7 1	3 5 4 7 1 8 1 6 0 2 0 1 0 1	-19/8*zeta(3)^2/pi^6+73/24192
2 7 1	3 6 5 7 1 2 0 8 0 4 0 1 0 1	19/8*zeta(3)^2/pi^6-73/24192
2 7 1	4 6 1 2 1 5 1 7 0 8 0 3 0 1	-83/32*zeta(3)^2/pi^6+1427/362880
2 7 1	3 6 1 8 1 5 0 2 0 7 0 4 0 1	83/32*zeta(3)^2/pi^6-1427/362880
2 7 1	4 6 1 5 1 7 1 2 0 8 0 3 0 1	81/32*zeta(3)^2/pi^6-799/181440
2 7 1	3 6 1 8 1 5 0 7 0 4 0 2 0 1	-81/32*zeta(3)^2/pi^6+799/181440
2 7 1	6 7 1 5 1 2 1 4 0 3 0 8 0 1	-59/32*zeta(3)^2/pi^6+3701/1451520
2 7 1	3 4 1 5 1 8 0 7 0 2 0 6 0 1	-59/32*zeta(3)^2/pi^6+3701/1451520
2 7 1	6 7 1 5 1 2 1 8 0 3 0 4 0 1	19/16*zeta(3)^2/pi^6-19/11520
2 7 1	3 4 1 5 1 6 0 7 0 2 0 8 0 1	19/16*zeta(3)^2/pi^6-19/11520
2 7 1	3 6 1 7 1 5 1 8 0 4 0 2 0 1	19/16*zeta(3)^2/pi^6-419/241920
2 7 1	3 5 1 6 1 2 0 4 0 7 0 8 0 1	-19/16*zeta(3)^2/pi^6+419/241920
2 7 1	3 6 1 5 1 7 1 4 0 8 0 2 0 1	51/32*zeta(3)^2/pi^6-607/207360
2 7 1	3 5 1 8 1 2 0 7 0 4 0 6 0 1	-51/32*zeta(3)^2/pi^6+607/207360
2 7 1	1 5 1 8 1 3 1 4 0 7 0 2 0 6	-3/16*zeta(3)^2/pi^6+143/362880
2 7 1	1 3 1 5 1 2 0 8 0 4 0 6 0 7	3/16*zeta(3)^2/pi^6-143/362880
2 7 1	3 6 1 5 1 8 1 4 0 7 0 2 0 1	-1/2*zeta(3)^2/pi^6+191/181440
2 7 1	3 5 1 4 1 2 0 7 0 8 0 6 0 1	1/2*zeta(3)^2/pi^6-191/181440
2 7 1	4 5 2 7 3 8 1 6 0 1 0 1 0 1	45/16*zeta(3)^2/pi^6-1429/362880
2 7 1	3 5 4 8 2 6 0 7 0 1 0 1 0 1	45/16*zeta(3)^2/pi^6-1429/362880
2 7 1	4 8 1 2 1 5 1 6 0 7 0 3 0 1	-3/4*zeta(3)^2/pi^6+11/8640
2 7 1	5 8 1 6 1 3 0 7 0 2 0 4 0 1	-3/4*zeta(3)^2/pi^6+11/8640
2 7 1	3 4 1 7 1 8 1 2 0 5 0 6 0 1	-3/4*zeta(3)^2/pi^6+1/896
2 7 1	5 6 1 7 1 3 0 4 0 8 0 2 0 1	-3/4*zeta(3)^2/pi^6+1/896
2 7 1	3 8 1 7 1 5 1 2 0 4 0 6 0 1	-67/8*zeta(3)^2/pi^6+217/17280
2 7 1	5 8 1 6 1 3 0 4 0 7 0 2 0 1	-67/8*zeta(3)^2/pi^6+217/17280
2 7 1	1 5 1 7 1 3 1 4 0 8 0 6 0 1	-1/2*zeta(3)^2/pi^6+13/22680
2 7 1	1 8 1 2 0 7 0 3 0 5 0 6 0 1	-1/2*zeta(3)^2/pi^6+13/22680
2 7 1	5 7 2 4 1 6 1 8 1 3 0 1 0 1	-1/1890
2 7 1	5 7 2 4 0 6 0 8 0 3 0 1 0 1	1/1890

2 7 1	3 7 1 6 1 2 1 4 1 5 0 8 0 1	-1/1890
2 7 1	3 4 1 8 0 7 0 2 0 5 0 6 0 1	1/1890
2 7 1	3 4 1 7 1 5 1 2 0 8 0 6 0 1	$1/4*\zeta(3)^2/\pi^6-13/45360$
2 7 1	5 6 1 8 1 3 0 7 0 4 0 2 0 1	$-1/4*\zeta(3)^2/\pi^6+13/45360$
2 7 1	1 3 0 2 0 1 0 1 0 1 0 1 0 1	-1/720
2 7 1	1 4 1 8 0 2 0 1 0 1 0 1 0 1	1/108
2 7 1	1 3 0 2 0 8 0 1 0 1 0 1 0 1	-1/108
2 7 1	3 4 1 2 1 8 0 1 0 1 0 1 0 1	1/540
2 7 1	3 4 0 2 0 8 0 1 0 1 0 1 0 1	-1/540
2 7 1	3 4 1 2 0 8 0 1 0 1 0 1 0 1	1/1080
2 7 1	3 4 1 8 0 2 0 1 0 1 0 1 0 1	1/1080
2 7 1	3 8 2 7 0 1 0 1 0 1 0 1 0 1	1/1080
2 7 1	1 5 1 4 0 3 0 8 0 1 0 1 0 1	1/72
2 7 1	1 5 1 8 0 3 0 2 0 1 0 1 0 1	1/72
2 7 1	3 5 1 4 1 8 0 2 0 1 0 1 0 1	-1/2880
2 7 1	3 4 1 2 0 5 0 8 0 1 0 1 0 1	1/2880
2 7 1	3 4 2 7 1 8 0 1 0 1 0 1 0 1	-1/480
2 7 1	3 4 2 7 0 8 0 1 0 1 0 1 0 1	1/480
2 7 1	3 4 1 5 1 2 0 8 0 1 0 1 0 1	-1/360
2 7 1	4 5 1 8 0 3 0 2 0 1 0 1 0 1	-1/360
2 7 1	1 5 1 8 1 7 0 2 0 1 0 1 0 1	-1/108
2 7 1	1 5 0 8 0 7 0 2 0 1 0 1 0 1	-1/108
2 7 1	1 4 1 8 0 2 0 7 0 1 0 1 0 1	1/54
2 7 1	1 4 1 6 1 5 1 8 0 3 0 1 0 1	-1/270
2 7 1	1 4 0 5 0 2 0 6 0 8 0 1 0 1	1/270
2 7 1	3 5 1 2 1 7 1 8 0 1 0 1 0 1	-1/270
2 7 1	4 5 0 8 0 7 0 2 0 1 0 1 0 1	1/270
2 7 1	1 6 1 8 1 5 0 4 0 3 0 1 0 1	-1/90
2 7 1	1 6 1 5 0 3 0 8 0 2 0 1 0 1	1/90
2 7 1	3 5 1 2 1 7 0 8 0 1 0 1 0 1	-1/540
2 7 1	3 4 1 8 0 2 0 7 0 1 0 1 0 1	1/540
2 7 1	1 3 1 5 1 6 0 8 0 4 0 1 0 1	-1/540
2 7 1	1 8 1 4 0 3 0 6 0 2 0 1 0 1	1/540
2 7 1	4 5 1 7 1 8 0 2 0 1 0 1 0 1	-1/540
2 7 1	3 5 1 2 0 8 0 7 0 1 0 1 0 1	1/540
2 7 1	4 8 1 5 1 7 0 3 0 1 0 1 0 1	-1/108
2 7 1	5 8 1 4 0 3 0 7 0 1 0 1 0 1	-1/108
2 7 1	3 8 2 7 1 6 0 1 0 1 0 1 0 1	-1/540
2 7 1	3 8 2 7 0 6 0 1 0 1 0 1 0 1	1/540
2 7 1	1 8 1 2 1 5 0 4 0 3 0 1 0 1	-1/540
2 7 1	1 5 1 4 0 3 0 6 0 8 0 1 0 1	1/540
2 7 1	3 4 1 2 1 7 0 8 0 1 0 1 0 1	1/270
2 7 1	4 5 1 8 0 2 0 7 0 1 0 1 0 1	1/270
2 7 1	1 7 1 8 1 5 0 4 0 3 0 1 0 1	-1/108
2 7 1	1 6 1 4 0 3 0 7 0 8 0 1 0 1	1/108
2 7 1	1 7 1 5 1 6 0 8 0 4 0 1 0 1	-1/108
2 7 1	1 7 1 4 0 3 0 8 0 2 0 1 0 1	1/108
2 7 1	3 4 5 7 1 2 0 8 0 1 0 1 0 1	$3/2*\zeta(3)^2/\pi^6-11/25920$
2 7 1	3 5 4 7 1 8 0 2 0 1 0 1 0 1	$3/2*\zeta(3)^2/\pi^6-11/25920$
2 7 1	5 7 1 6 1 2 1 8 0 3 0 1 0 1	1/540
2 7 1	6 7 1 4 0 3 0 2 0 8 0 1 0 1	-1/540
2 7 1	3 5 4 7 1 8 1 2 0 1 0 1 0 1	1/540
2 7 1	3 4 5 8 0 2 0 7 0 1 0 1 0 1	1/540
2 7 1	4 8 1 6 1 5 1 2 0 3 0 1 0 1	-1/540
2 7 1	5 8 1 4 0 3 0 6 0 2 0 1 0 1	1/540
2 7 1	6 7 1 2 1 5 0 4 0 8 0 1 0 1	11/8640
2 7 1	3 7 1 8 1 5 0 4 0 2 0 1 0 1	-11/8640
2 7 1	6 8 1 2 1 5 0 4 0 3 0 1 0 1	1/864
2 7 1	3 8 1 6 1 5 0 4 0 2 0 1 0 1	-1/864
2 7 1	3 4 1 5 1 2 1 6 1 8 0 1 0 1	1/945
2 7 1	3 4 0 5 0 2 0 6 0 8 0 1 0 1	-1/945
2 7 1	3 6 1 5 1 8 1 4 0 2 0 1 0 1	$-3/4*\zeta(3)^2/\pi^6+53/90720$
2 7 1	3 4 1 2 0 6 0 8 0 5 0 1 0 1	$-3/4*\zeta(3)^2/\pi^6+53/90720$
2 7 1	3 8 2 5 1 7 1 4 0 1 0 1 0 1	$-3/8*\zeta(3)^2/\pi^6+17/90720$
2 7 1	3 4 2 8 0 5 0 7 0 1 0 1 0 1	$-3/8*\zeta(3)^2/\pi^6+17/90720$
2 7 1	3 5 2 4 1 7 1 8 0 1 0 1 0 1	$-9/8*\zeta(3)^2/\pi^6+1/360$
2 7 1	3 5 2 4 0 7 0 8 0 1 0 1 0 1	$-9/8*\zeta(3)^2/\pi^6+1/360$
2 7 1	3 6 1 2 1 5 0 8 0 4 0 1 0 1	$-3/2*\zeta(3)^2/\pi^6+377/181440$
2 7 1	3 5 1 6 1 8 0 2 0 4 0 1 0 1	$-3/2*\zeta(3)^2/\pi^6+377/181440$
2 7 1	3 4 2 7 1 5 0 8 0 1 0 1 0 1	$-3/8*\zeta(3)^2/\pi^6-11/5040$
2 7 1	3 5 2 7 1 8 0 4 0 1 0 1 0 1	$-3/8*\zeta(3)^2/\pi^6-11/5040$
2 7 1	3 5 2 4 1 8 0 7 0 1 0 1 0 1	$-15/8*\zeta(3)^2/\pi^6+13/6480$
2 7 1	3 5 1 2 1 8 0 6 0 4 0 1 0 1	$-3/4*\zeta(3)^2/\pi^6+11/36288$

2 7 1	3 6 1 4 1 5 0 8 0 2 0 1 0 1	-3/4*zeta(3)^2/pi^6+11/36288
2 7 1	3 4 1 2 1 5 0 6 0 8 0 1 0 1	3/4*zeta(3)^2/pi^6-53/90720
2 7 1	5 6 1 8 1 3 0 4 0 2 0 1 0 1	3/4*zeta(3)^2/pi^6-53/90720
2 7 1	4 5 1 8 1 6 1 2 0 3 0 1 0 1	1/420
2 7 1	5 6 1 4 0 8 0 3 0 2 0 1 0 1	-1/420
2 7 1	1 7 1 4 1 5 1 6 0 3 0 2 0 1	-1/540
2 7 1	1 4 1 5 0 6 0 3 0 7 0 2 0 1	-1/540
2 7 1	3 5 1 2 1 7 1 8 0 4 0 1 0 1	-1/540
2 7 1	5 6 1 4 0 8 0 7 0 2 0 1 0 1	-1/540
2 7 1	6 7 4 5 1 3 1 2 1 8 0 1 0 1	1/1890
2 7 1	6 7 4 5 0 3 0 2 0 8 0 1 0 1	-1/1890
2 7 1	3 4 1 5 1 2 1 6 0 8 0 1 0 1	1/1890
2 7 1	4 5 1 8 0 6 0 2 0 3 0 1 0 1	-1/1890
2 7 1	4 5 1 6 1 8 1 2 0 7 0 1 0 1	1/540
2 7 1	5 6 1 7 0 3 0 8 0 2 0 1 0 1	-1/540
2 7 1	3 6 1 8 1 7 0 4 0 2 0 1 0 1	-1/1080
2 7 1	3 6 1 2 1 5 0 7 0 8 0 1 0 1	-1/1080
2 7 1	3 8 2 7 1 6 0 4 0 1 0 1 0 1	-1/1080
2 7 1	3 8 2 7 1 5 0 6 0 1 0 1 0 1	-1/1080
2 7 1	3 6 1 7 1 5 0 8 0 2 0 1 0 1	-1/1080
2 7 1	3 6 1 2 1 7 0 4 0 8 0 1 0 1	-1/1080
2 7 1	1 7 1 5 1 6 0 3 0 2 0 4 0 1	-1/432
2 7 1	1 6 1 2 1 5 0 4 0 7 0 3 0 1	-1/2160
2 7 1	4 6 1 8 1 5 1 7 0 2 0 1 0 1	1/4320
2 7 1	3 5 1 2 0 8 0 6 0 7 0 1 0 1	1/4320
2 7 1	3 8 2 5 1 7 1 6 0 1 0 1 0 1	1/720
2 7 1	3 5 2 7 0 8 0 6 0 1 0 1 0 1	1/720
2 7 1	4 8 1 6 1 5 1 7 0 3 0 1 0 1	1/540
2 7 1	5 8 1 4 0 3 0 6 0 7 0 1 0 1	-1/540
2 7 1	5 8 2 4 1 3 1 6 1 7 0 1 0 1	-1/1890
2 7 1	5 8 2 4 0 3 0 6 0 7 0 1 0 1	1/1890
2 7 1	1 7 1 4 1 5 1 8 0 3 0 2 0 1	-1/540
2 7 1	1 5 1 4 0 3 0 6 0 7 0 8 0 1	-1/540
2 7 1	3 4 1 5 1 2 1 6 1 7 0 8 0 1	1/1890
2 7 1	4 5 1 8 0 6 0 2 0 7 0 3 0 1	1/1890
2 7 1	4 5 1 8 1 6 1 2 0 7 0 1 0 1	1/540
2 7 1	5 6 1 7 0 8 0 3 0 2 0 1 0 1	-1/540
2 7 1	1 7 1 4 1 5 1 6 0 8 0 2 0 1	-1/540
2 7 1	1 8 1 4 0 3 0 6 0 7 0 2 0 1	-1/540
2 7 1	4 6 1 5 1 2 1 8 0 3 0 1 0 1	-1/1890
2 7 1	3 5 1 4 0 6 0 2 0 8 0 1 0 1	-1/1890
2 7 1	6 7 1 8 1 5 0 4 0 3 0 1 0 1	7/2160
2 7 1	3 7 1 6 1 5 0 4 0 8 0 1 0 1	-7/2160
2 7 1	1 8 1 7 1 6 0 4 0 2 0 3 0 1	-43/8640
2 7 1	1 5 1 7 1 6 0 4 0 8 0 3 0 1	-43/8640
2 7 1	3 6 1 4 1 7 0 8 0 2 0 1 0 1	-1/4320
2 7 1	4 5 1 8 1 2 0 6 0 7 0 1 0 1	-1/4320
2 7 1	3 4 2 6 1 8 0 7 0 1 0 1 0 1	-1/720
2 7 1	3 5 2 6 1 8 0 7 0 1 0 1 0 1	-1/720
2 7 1	1 6 1 2 1 5 0 4 0 8 0 3 0 1	-19/8640
2 7 1	1 6 1 8 1 5 0 4 0 7 0 3 0 1	-19/8640
2 7 1	1 8 1 2 1 5 0 4 0 7 0 3 0 1	-1/4320
2 7 1	1 3 1 5 1 6 0 7 0 4 0 8 0 1	-1/4320
2 7 1	3 6 1 7 1 5 0 4 0 8 0 1 0 1	-23/4320
2 7 1	3 4 1 2 1 6 0 7 0 8 0 1 0 1	1/540
2 7 1	5 6 1 8 1 7 0 4 0 2 0 1 0 1	1/540
2 7 1	5 6 4 8 1 7 1 2 0 3 0 1 0 1	1/8*zeta(3)^2/pi^6+1/25920
2 7 1	6 7 4 5 1 2 0 3 0 8 0 1 0 1	1/8*zeta(3)^2/pi^6+1/25920
2 7 1	1 8 1 5 1 7 1 6 0 2 0 4 0 3	1/2880
2 7 1	1 6 1 5 1 7 0 8 0 2 0 3 0 4	-1/2880
2 7 1	5 6 4 8 1 7 1 3 0 2 0 1 0 1	15/16*zeta(3)^2/pi^6-13/10080
2 7 1	4 6 5 7 1 2 0 8 0 3 0 1 0 1	15/16*zeta(3)^2/pi^6-13/10080
2 7 1	5 7 4 8 2 3 1 6 0 1 0 1 0 1	25/16*zeta(3)^2/pi^6-7/1920
2 7 1	3 4 2 7 5 8 0 6 0 1 0 1 0 1	-25/16*zeta(3)^2/pi^6+7/1920
2 7 1	3 6 4 8 1 5 1 7 0 2 0 1 0 1	-9/8*zeta(3)^2/pi^6+29/22680
2 7 1	3 4 5 7 1 2 0 6 0 8 0 1 0 1	9/8*zeta(3)^2/pi^6-29/22680
2 7 1	3 5 6 7 1 8 1 2 0 4 0 1 0 1	3/8*zeta(3)^2/pi^6-7/5184
2 7 1	4 7 2 5 1 6 0 3 0 8 0 1 0 1	-3/8*zeta(3)^2/pi^6+7/5184
2 7 1	6 8 1 7 1 5 1 2 0 4 0 3 0 1	1/8640
2 7 1	3 8 1 6 1 5 0 4 0 7 0 2 0 1	1/8640
2 7 1	4 8 1 7 1 5 1 6 0 2 0 3 0 1	1/4320
2 7 1	6 8 1 2 1 5 0 4 0 7 0 3 0 1	1/4320
2 7 1	4 6 1 8 1 2 1 7 0 5 0 3 0 1	-7/8*zeta(3)^2/pi^6+17/13440

2 7 1	3 6 1 7 1 5 0 8 0 2 0 4 0 1	$7/8*\zeta(3)^2/\pi^6-17/13440$
2 7 1	6 8 4 5 1 2 1 3 0 7 0 1 0 1	$-1/4*\zeta(3)^2/\pi^6+1/90720$
2 7 1	4 8 5 6 1 7 0 3 0 2 0 1 0 1	$-1/4*\zeta(3)^2/\pi^6+1/90720$
2 7 1	3 4 1 7 1 2 1 6 0 8 0 5 0 1	$-1/4*\zeta(3)^2/\pi^6+89/60480$
2 7 1	5 6 1 8 1 7 0 4 0 2 0 3 0 1	$-1/4*\zeta(3)^2/\pi^6+89/60480$
2 7 1	3 8 1 7 1 2 1 6 0 5 0 4 0 1	$1/1440$
2 7 1	5 8 1 7 1 6 0 4 0 2 0 3 0 1	$1/1440$
2 7 1	3 5 4 7 1 6 1 2 0 8 0 1 0 1	$-3/8*\zeta(3)^2/\pi^6+17/15120$
2 7 1	6 8 2 5 1 7 0 3 0 4 0 1 0 1	$3/8*\zeta(3)^2/\pi^6-17/15120$
2 7 1	3 8 5 7 1 6 1 2 0 4 0 1 0 1	$1/1728$
2 7 1	6 8 2 7 1 5 0 4 0 3 0 1 0 1	$-1/1728$
2 7 1	3 7 1 6 1 8 1 4 1 5 0 2 0 1	$5/16*\zeta(3)^2/\pi^6-7/17280$
2 7 1	3 4 1 2 0 7 0 8 0 5 0 6 0 1	$-5/16*\zeta(3)^2/\pi^6+7/17280$
2 7 1	3 4 2 7 1 6 1 8 1 5 0 1 0 1	$15/16*\zeta(3)^2/\pi^6-31/30240$
2 7 1	3 4 2 7 0 6 0 8 0 5 0 1 0 1	$-15/16*\zeta(3)^2/\pi^6+31/30240$
2 7 1	3 4 2 5 1 8 1 6 1 7 0 1 0 1	$11/16*\zeta(3)^2/\pi^6-179/181440$
2 7 1	3 4 2 5 0 8 0 6 0 7 0 1 0 1	$-11/16*\zeta(3)^2/\pi^6+179/181440$
2 7 1	4 6 1 8 1 5 1 7 0 2 0 3 0 1	$-3/16*\zeta(3)^2/\pi^6+61/90720$
2 7 1	3 6 1 2 1 5 0 8 0 7 0 4 0 1	$3/16*\zeta(3)^2/\pi^6-61/90720$
2 7 1	3 5 2 6 1 8 1 7 0 4 0 1 0 1	$3/8*\zeta(3)^2/\pi^6+23/181440$
2 7 1	3 4 2 5 1 6 0 7 0 8 0 1 0 1	$-3/8*\zeta(3)^2/\pi^6-23/181440$
2 7 1	3 8 2 5 1 7 1 6 0 4 0 1 0 1	$3/4*\zeta(3)^2/\pi^6-1/40320$
2 7 1	3 7 2 5 1 6 0 4 0 8 0 1 0 1	$-3/4*\zeta(3)^2/\pi^6+1/40320$
2 7 1	6 7 1 5 1 8 1 4 0 3 0 2 0 1	$-1/2*\zeta(3)^2/\pi^6+191/181440$
2 7 1	3 4 1 5 1 2 0 7 0 8 0 6 0 1	$-1/2*\zeta(3)^2/\pi^6+191/181440$
2 7 1	3 4 1 7 1 2 1 8 0 5 0 6 0 1	$-1/4*\zeta(3)^2/\pi^6+157/181440$
2 7 1	5 6 1 7 1 3 0 4 0 2 0 8 0 1	$-1/4*\zeta(3)^2/\pi^6+157/181440$
2 7 1	3 8 2 6 1 5 1 7 0 4 0 1 0 1	$-15/16*\zeta(3)^2/\pi^6+319/181440$
2 7 1	3 8 2 4 1 5 0 6 0 7 0 1 0 1	$15/16*\zeta(3)^2/\pi^6-319/181440$
2 7 1	4 6 1 5 1 7 1 8 0 2 0 3 0 1	$\zeta(3)^2/\pi^6-59/40320$
2 7 1	3 6 1 2 1 5 0 7 0 4 0 8 0 1	$-\zeta(3)^2/\pi^6+59/40320$
2 7 1	3 5 2 6 1 8 1 4 0 7 0 1 0 1	$3/16*\zeta(3)^2/\pi^6-11/24192$
2 7 1	3 4 2 5 1 8 0 6 0 7 0 1 0 1	$-3/16*\zeta(3)^2/\pi^6+11/24192$
2 7 1	3 6 1 5 1 7 1 4 0 2 0 8 0 1	$3/16*\zeta(3)^2/\pi^6-1/2880$
2 7 1	3 5 1 2 1 8 0 7 0 4 0 6 0 1	$-3/16*\zeta(3)^2/\pi^6+1/2880$
2 7 1	3 5 2 4 1 6 1 7 0 8 0 1 0 1	$-1/8*\zeta(3)^2/\pi^6+121/90720$
2 7 1	3 5 2 6 1 7 0 8 0 4 0 1 0 1	$1/8*\zeta(3)^2/\pi^6-121/90720$
2 7 1	3 8 2 5 1 6 1 4 0 7 0 1 0 1	$5/8*\zeta(3)^2/\pi^6-353/362880$
2 7 1	3 8 2 5 1 7 0 6 0 4 0 1 0 1	$-5/8*\zeta(3)^2/\pi^6+353/362880$
2 7 1	3 5 4 6 1 8 1 2 0 7 0 1 0 1	$409/362880$
2 7 1	4 6 2 5 1 7 0 3 0 8 0 1 0 1	$409/362880$
2 7 1	4 6 1 7 1 8 1 2 0 5 0 3 0 1	$1/1440$
2 7 1	3 6 1 7 1 5 0 4 0 8 0 2 0 1	$-1/1440$
2 7 1	4 5 1 7 1 8 1 6 0 2 0 3 0 1	$1/8640$
2 7 1	6 7 1 2 1 5 0 4 0 8 0 3 0 1	$1/8640$
2 7 1	4 7 1 5 1 2 1 6 0 8 0 3 0 1	$5/8*\zeta(3)^2/\pi^6-149/120960$
2 7 1	3 5 1 6 1 8 0 2 0 7 0 4 0 1	$-5/8*\zeta(3)^2/\pi^6+149/120960$
2 7 1	4 6 1 7 1 5 1 2 0 8 0 3 0 1	$-1/1080$
2 7 1	3 6 1 8 1 5 0 4 0 7 0 2 0 1	$1/1080$
2 7 1	4 6 1 5 1 2 1 8 0 7 0 3 0 1	$1/4*\zeta(3)^2/\pi^6-29/72576$
2 7 1	3 6 1 4 1 5 0 7 0 2 0 8 0 1	$-1/4*\zeta(3)^2/\pi^6+29/72576$
2 7 1	1 8 1 4 1 5 1 6 0 7 0 3 0 2	$-1/2880$
2 7 1	1 6 1 2 1 5 0 4 0 7 0 8 0 3	$1/2880$
2 7 1	3 4 1 5 1 2 1 6 0 7 0 8 0 1	$1/4*\zeta(3)^2/\pi^6-13/45360$
2 7 1	5 6 1 8 1 3 0 7 0 2 0 4 0 1	$1/4*\zeta(3)^2/\pi^6-13/45360$
2 7 1	1 5 1 4 0 3 0 2 0 1 0 1 0 1	$1/432$
2 7 1	1 8 1 6 1 5 0 4 0 3 0 1 0 1	$-1/216$
2 7 1	1 6 1 5 0 8 0 3 0 2 0 1 0 1	$1/216$
2 7 1	4 5 1 6 1 8 1 2 0 3 0 1 0 1	$1/540$
2 7 1	5 6 1 4 0 3 0 8 0 2 0 1 0 1	$-1/540$
2 7 1	3 6 1 2 1 5 0 4 0 8 0 1 0 1	$-1/1080$
2 7 1	3 6 1 8 1 5 0 4 0 2 0 1 0 1	$-1/1080$
2 7 1	3 8 2 7 1 5 0 4 0 1 0 1 0 1	$-1/1080$
2 7 1	1 5 1 7 1 6 0 8 0 4 0 3 0 1	$-1/432$
2 7 1	1 8 1 7 1 6 0 2 0 4 0 3 0 1	$-1/432$
2 7 1	4 6 1 7 1 5 1 8 0 2 0 3 0 1	$1/8640$
2 7 1	3 6 1 2 1 5 0 4 0 7 0 8 0 1	$-1/8640$
2 7 1	3 8 2 5 1 6 1 7 0 4 0 1 0 1	$1/1440$
2 7 1	3 7 2 5 1 6 0 8 0 4 0 1 0 1	$-1/1440$
2 7 1	4 5 1 7 1 2 1 6 0 8 0 3 0 1	$-1/1080$
2 7 1	6 7 1 8 1 5 0 4 0 2 0 3 0 1	$-1/1080$
2 7 1	1 7 1 6 1 5 0 4 0 3 0 2 0 1	$-1/1296$

## C.2 Reduced expansion $\star_{\text{aff}}^{\text{red}} \bmod \bar{o}(\hbar^7)$

**Encoding 2.** In the format described in Chapter 11, Implementation 1:

h^0:				2 6 1	1 4 1 7 1 3 0 6 0 2 0 5	-2/567	
2 0 1	1			2 6 1	3 5 1 7 1 6 0 4 0 2 0 1	-31/3780	
h^1:				2 6 1	1 7 1 4 1 5 0 6 0 3 0 2	-1/360	
2 1 1	0 1	1		2 6 1	3 5 1 2 1 6 0 4 0 7 0 1	-7/1620	
h^2:				2 6 1	4 5 1 6 1 2 0 7 0 3 0 1	-11/2160	
2 2 1	1 3 0 2	-1/6		2 6 1	1 7 1 6 1 5 0 4 0 3 0 2	-1/1296	
2 2 1	1 3 0 1	-1/3		2 6 1	1 4 1 6 1 7 0 3 0 2 0 1	4/315	
2 2 1	0 3 0 1	1/3		2 6 1	4 6 1 7 1 5 0 3 0 1 0 1	-2/105	
2 2 1	0 1 0 1	1/2		2 6 1	1 6 1 4 1 7 0 3 0 5 0 1	-2/315	
h^3:				2 6 1	1 4 1 7 1 5 0 6 0 2 0 1	-1/180	
2 3 1	1 3 0 2 0 1	-1/6		2 6 1	1 4 1 7 1 5 0 6 0 3 0 1	-4/105	
2 3 1	1 4 0 2 0 1	-1/3		2 6 1	1 4 1 6 1 3 0 7 0 2 0 1	1/270	
2 3 1	1 4 0 1 0 1	-1/3		2 6 1	1 4 1 6 1 3 0 7 0 5 0 1	2/135	
2 3 1	0 4 0 1 0 1	1/3		2 6 1	4 6 1 5 1 7 0 3 0 1 0 1	-1/45	
2 3 1	0 1 0 1 0 1	1/6		2 6 1	3 7 2 6 1 5 0 1 0 1 0 1	-11/1080	
h^4:				2 6 1	1 6 1 4 1 7 0 3 0 2 0 1	-2/135	
2 4 1	1 5 1 4 0 3 0 2	1/72		2 6 1	1 7 1 6 1 5 0 4 0 3 0 1	-1/216	
2 4 1	1 3 1 4 0 5 0 2	-2/45		2 6 1	1 5 1 4 0 6 0 3 0 7 0 1	-4/315	
2 4 1	1 5 1 4 0 2 0 3	11/180		2 6 1	5 6 1 4 0 7 0 3 0 1 0 1	-2/105	
2 4 1	1 5 1 4 0 3 0 1	1/18		2 6 1	1 5 1 2 0 3 0 6 0 7 0 1	2/315	
2 4 1	1 3 1 5 0 2 0 1	-2/45		2 6 1	1 5 1 2 0 7 0 6 0 3 0 1	1/180	
2 4 1	1 3 1 4 0 5 0 1	2/15		2 6 1	1 5 1 2 0 6 0 7 0 3 0 1	4/105	
2 4 1	1 4 0 5 0 2 0 1	-1/18		2 6 1	1 7 1 4 0 6 0 3 0 5 0 1	-1/270	
2 4 1	1 3 0 4 0 5 0 1	2/45		2 6 1	1 7 1 2 0 6 0 3 0 5 0 1	-2/135	
2 4 1	1 5 0 4 0 2 0 1	-2/15		2 6 1	5 6 1 4 0 3 0 7 0 1 0 1	-1/45	
2 4 1	1 3 0 2 0 1 0 1	-1/12		2 6 1	3 7 2 6 0 5 0 1 0 1 0 1	11/1080	
2 4 1	1 5 0 2 0 1 0 1	-1/3		2 6 1	1 5 1 4 0 3 0 6 0 7 0 1	2/135	
2 4 1	1 5 0 4 0 1 0 1	-1/9		2 6 1	1 6 1 5 0 7 0 3 0 2 0 1	1/216	
2 4 1	1 5 0 1 0 1 0 1	-1/6		2 6 1	1 4 1 2 0 5 0 3 0 1 0 1	1/120	
2 4 1	0 5 0 1 0 1 0 1	1/6		2 6 1	3 4 1 7 0 6 0 1 0 1 0 1	2/45	
2 4 1	0 1 0 1 0 1 0 1	1/24		2 6 1	1 5 1 4 0 7 0 3 0 1 0 1	2/45	
2 4 1	1 5 1 4 0 1 0 1	1/18		2 6 1	1 7 1 5 0 3 0 2 0 1 0 1	2/45	
2 4 1	0 5 0 4 0 1 0 1	1/18		2 6 1	1 5 1 6 0 3 0 7 0 1 0 1	1/18	
2 4 1	1 4 1 5 1 3 0 1	1/45		2 6 1	3 7 1 6 0 5 0 1 0 1 0 1	2/45	
2 4 1	0 4 0 5 0 3 0 1	-1/45		2 6 1	4 7 1 6 0 5 0 1 0 1 0 1	-2/45	
2 4 1	1 4 1 5 1 3 0 2	1/90		2 6 1	1 6 1 2 0 7 0 3 0 1 0 1	4/135	
2 4 1	1 3 0 5 0 2 0 4	1/90		2 6 1	1 7 1 4 0 5 0 6 0 1 0 1	4/135	
h^5:				2 6 1	3 7 2 6 0 1 0 1 0 1 0 1	11/720	
2 5 1	1 4 1 2 0 5 0 3 0 1	1/60		2 6 1	1 7 1 4 0 3 0 2 0 1 0 1	1/18	
2 5 1	3 4 1 6 0 5 0 1 0 1	2/15		2 6 1	1 6 1 5 0 7 0 3 0 1 0 1	1/54	
2 5 1	1 6 1 2 0 5 0 3 0 1	2/45		2 6 1	1 5 1 4 0 3 0 2 0 1 0 1	1/144	
2 5 1	1 3 1 4 0 5 0 6 0 1	2/45		2 6 1	1 5 1 7 1 3 1 4 0 2 0 6	-1/378	
2 5 1	1 5 1 4 0 3 0 6 0 1	1/18		2 6 1	3 6 1 5 1 2 1 4 0 7 0 1	-31/11340	
2 5 1	3 4 1 6 0 2 0 1 0 1	11/360		2 6 1	1 7 1 4 1 5 1 6 0 3 0 2	-1/540	
2 5 1	1 5 1 4 0 3 0 2 0 1	1/72		2 6 1	3 5 2 4 1 6 1 7 0 1 0 1	1/567	
2 5 1	1 3 1 6 0 2 0 1 0 1	-2/45		2 6 1	1 4 1 2 0 7 0 3 0 5 0 6	-1/378	
2 5 1	1 3 1 4 0 6 0 1 0 1	2/15		2 6 1	3 4 1 7 0 6 0 2 0 5 0 1	-31/11340	
2 5 1	1 4 1 5 0 6 0 1 0 1	1/9		2 6 1	1 4 1 5 0 6 0 3 0 7 0 2	-1/540	
2 5 1	1 4 1 6 0 2 0 1 0 1	1/18		2 6 1	3 5 2 4 0 6 0 7 0 1 0 1	1/567	
2 5 1	1 3 0 4 0 6 0 1 0 1	2/45		2 6 1	1 7 0 6 0 1 0 1 0 1 0 1	-1/18	
2 5 1	1 6 0 4 0 2 0 1 0 1	-2/15		2 6 1	1 3 0 2 0 1 0 1 0 1 0 1	-1/144	
2 5 1	1 5 0 6 0 2 0 1 0 1	-1/9		2 6 1	1 3 0 7 0 1 0 1 0 1 0 1	-1/18	
2 5 1	1 4 0 6 0 2 0 1 0 1	-1/18		2 6 1	3 7 1 6 0 1 0 1 0 1 0 1	1/15	
2 5 1	1 6 0 2 0 1 0 1 0 1	-1/6		2 6 1	1 3 1 7 0 2 0 1 0 1 0 1	2/45	
2 5 1	1 6 0 5 0 1 0 1 0 1	-1/9		2 6 1	1 4 1 6 0 7 0 1 0 1 0 1	1/9	
2 5 1	1 3 0 2 0 1 0 1 0 1	-1/36		2 6 1	1 7 1 6 0 5 0 1 0 1 0 1	1/54	
2 5 1	1 4 1 5 1 3 0 2 0 1	1/90		2 6 1	1 4 1 7 0 2 0 1 0 1 0 1	1/36	
2 5 1	1 5 1 4 1 6 0 3 0 1	2/45		2 6 1	3 7 0 6 0 1 0 1 0 1 0 1	1/15	
2 5 1	1 4 1 6 1 5 0 3 0 1	-1/45		2 6 1	1 3 0 4 0 7 0 1 0 1 0 1	-2/45	
2 5 1	1 3 0 5 0 2 0 4 0 1	1/90		2 6 1	1 7 0 2 0 6 0 1 0 1 0 1	-1/9	
2 5 1	1 4 0 5 0 6 0 2 0 1	-1/45		2 6 1	2 6 1	1 7 0 6 0 5 0 1 0 1 0 1	-1/54
2 5 1	1 4 0 2 0 5 0 6 0 1	2/45		2 6 1	1 3 0 2 0 7 0 1 0 1 0 1	-1/36	
2 5 1	0 1 0 1 0 1 0 1 0 1	1/120		2 6 1	1 4 1 5 1 3 0 2 0 1 0 1	1/180	
2 5 1	1 6 0 1 0 1 0 1 0 1	-1/18		2 6 1	4 5 1 6 1 7 0 1 0 1 0 1	-2/45	
2 5 1	0 6 0 1 0 1 0 1 0 1	1/18		2 6 1	1 7 1 4 1 6 0 3 0 1 0 1	-4/135	
2 5 1	1 6 1 5 0 1 0 1 0 1	1/18		2 6 1	1 4 1 7 1 3 0 6 0 1 0 1	1/135	
2 5 1	0 6 0 5 0 1 0 1 0 1	1/18		2 6 1	1 5 1 7 1 6 0 2 0 1 0 1	-1/108	
2 5 1	1 4 1 6 1 3 0 1 0 1	1/45		2 6 1	1 5 1 4 1 7 0 3 0 1 0 1	2/45	
2 5 1	0 4 0 6 0 3 0 1 0 1	-1/45		2 6 1	1 4 1 7 1 5 0 3 0 1 0 1	-1/45	
h^6:				2 6 1	1 3 0 5 0 2 0 4 0 1 0 1	1/180	
				2 6 1	4 7 0 6 0 5 0 1 0 1 0 1	2/45	
				2 6 1	1 4 0 7 0 5 0 6 0 1 0 1	-4/135	
				2 6 1	1 6 0 5 0 7 0 4 0 1 0 1	1/135	



2 6 1	1 5 0 7 0 6 0 2 0 1 0 1	-1/108	2 7 1	4 8 1 5 1 7 0 3 0 1 0 1 0 1	-1/45
2 6 1	1 4 0 5 0 7 0 2 0 1 0 1	-1/45	2 7 1	3 8 2 7 1 6 0 1 0 1 0 1 0 1	-11/1080
2 6 1	1 4 0 2 0 5 0 7 0 1 0 1	2/45	2 7 1	1 8 1 2 1 5 0 4 0 3 0 1 0 1	-2/135
2 6 1	1 7 1 4 1 5 1 6 0 3 0 1	-1/270	2 7 1	1 7 1 5 1 6 0 8 0 4 0 1 0 1	-1/54
2 6 1	1 4 1 6 1 5 1 7 0 3 0 1	-1/270	2 7 1	1 8 1 6 1 5 0 4 0 3 0 1 0 1	-1/216
2 6 1	3 6 1 5 1 7 1 4 0 1 0 1	-2/315	2 7 1	1 7 1 6 1 2 0 8 0 4 0 1 0 1	2/135
2 6 1	1 5 1 4 1 7 1 6 0 3 0 1	-4/315	2 7 1	1 3 1 6 1 7 0 8 0 4 0 1 0 1	-1/135
2 6 1	1 4 0 7 0 5 0 6 0 2 0 1	1/270	2 7 1	1 5 1 4 0 6 0 3 0 8 0 1 0 1	-4/315
2 6 1	1 4 0 5 0 2 0 6 0 7 0 1	1/270	2 7 1	4 8 1 5 0 3 0 7 0 1 0 1 0 1	-2/105
2 6 1	3 6 0 5 0 7 0 4 0 1 0 1	-2/315	2 7 1	1 5 1 2 0 6 0 8 0 3 0 1 0 1	2/105
2 6 1	1 4 0 6 0 5 0 7 0 2 0 1	4/315	2 7 1	1 8 1 2 0 6 0 3 0 5 0 1 0 1	-8/945
2 6 1	0 1 0 1 0 1 0 1 0 1 0 1	1/720	2 7 1	1 8 1 5 0 3 0 6 0 2 0 1 0 1	2/105
2 6 1	1 7 0 1 0 1 0 1 0 1 0 1	-1/72	2 7 1	5 8 1 7 0 3 0 6 0 1 0 1 0 1	-2/45
2 6 1	0 7 0 1 0 1 0 1 0 1 0 1	1/72	2 7 1	1 5 1 8 0 3 0 6 0 7 0 1 0 1	4/135
2 6 1	1 7 1 6 0 1 0 1 0 1 0 1	1/36	2 7 1	1 8 1 4 0 6 0 3 0 5 0 1 0 1	-1/270
2 6 1	0 7 0 6 0 1 0 1 0 1 0 1	1/36	2 7 1	1 5 1 2 0 8 0 6 0 3 0 1 0 1	1/180
2 6 1	1 4 1 7 1 3 0 1 0 1 0 1	1/90	2 7 1	1 3 1 5 0 8 0 6 0 7 0 1 0 1	1/135
2 6 1	1 7 1 6 1 5 0 1 0 1 0 1	-1/162	2 7 1	3 5 1 8 0 7 0 6 0 1 0 1 0 1	1/45
2 6 1	0 4 0 7 0 3 0 1 0 1 0 1	-1/90	2 7 1	1 7 1 6 0 3 0 8 0 2 0 1 0 1	1/45
2 6 1	0 7 0 6 0 5 0 1 0 1 0 1	1/162	2 7 1	5 8 1 4 0 3 0 7 0 1 0 1 0 1	-1/45
2 6 1	1 6 1 7 1 5 1 3 1 4 0 1	-2/945	2 7 1	3 8 2 7 0 6 0 1 0 1 0 1 0 1	11/1080
2 6 1	0 6 0 7 0 5 0 3 0 4 0 1	2/945	2 7 1	1 5 1 4 0 3 0 6 0 8 0 1 0 1	2/135
2 6 1	1 7 1 4 1 5 1 6 0 1 0 1	-1/135	2 7 1	1 7 1 4 0 3 0 8 0 2 0 1 0 1	1/54
2 6 1	0 7 0 4 0 5 0 6 0 1 0 1	-1/135	2 7 1	1 6 1 5 0 8 0 3 0 2 0 1 0 1	1/216
2 6 1	1 6 1 7 1 5 1 3 1 4 0 2	-1/945	2 7 1	1 7 1 5 0 6 0 8 0 3 0 1 0 1	1/135
2 6 1	1 3 0 7 0 2 0 6 0 4 0 5	-1/945	2 7 1	1 8 1 5 0 3 0 6 0 7 0 1 0 1	-2/135
h^7:			2 7 1	1 4 1 8 1 5 1 7 0 3 0 2 0 1	-1/270
2 7 1	1 6 1 5 1 3 0 7 0 8 0 4 0 1	-1/180	2 7 1	1 5 1 7 1 3 1 4 0 2 0 6 0 1	11/7560
2 7 1	1 3 1 5 1 2 0 7 0 4 0 6 0 1	53/11340	2 7 1	1 7 1 4 1 5 1 8 0 3 0 6 0 1	-2/315
2 7 1	1 5 1 2 1 6 0 7 0 3 0 4 0 1	-31/3780	2 7 1	1 5 1 7 1 3 1 4 0 2 0 8 0 1	-4/315
2 7 1	1 3 1 5 1 6 0 4 0 7 0 8 0 1	2/135	2 7 1	3 4 1 8 1 5 1 6 0 7 0 1 0 1	-4/315
2 7 1	1 3 1 6 1 2 0 7 0 8 0 4 0 1	-8/945	2 7 1	1 5 1 4 1 7 1 6 0 3 0 2 0 1	-31/7560
2 7 1	1 3 1 5 1 8 0 7 0 4 0 6 0 1	-29/945	2 7 1	1 5 1 4 1 8 1 6 0 7 0 3 0 1	8/945
2 7 1	3 6 1 4 1 5 0 7 0 8 0 1 0 1	-46/945	2 7 1	1 5 1 7 1 3 1 4 0 8 0 6 0 1	-2/945
2 7 1	3 5 1 8 1 7 0 6 0 4 0 1 0 1	-32/945	2 7 1	1 7 1 4 1 5 1 6 0 3 0 2 0 1	-1/540
2 7 1	3 6 1 7 1 5 0 2 0 8 0 1 0 1	8/315	2 7 1	4 6 1 5 1 2 1 8 0 3 0 1 0 1	-37/15120
2 7 1	3 6 1 4 1 5 0 8 0 2 0 1 0 1	-191/22680	2 7 1	1 7 1 5 1 6 1 8 0 4 0 3 0 1	-1/135
2 7 1	3 6 1 2 1 7 0 4 0 8 0 1 0 1	-11/1080	2 7 1	1 7 1 5 1 8 1 6 0 4 0 2 0 1	1/270
2 7 1	1 6 1 2 1 5 0 4 0 7 0 3 0 1	-1/360	2 7 1	1 5 1 4 0 6 0 8 0 7 0 3 0 1	-1/270
2 7 1	1 8 1 2 1 5 0 4 0 7 0 3 0 1	-1/135	2 7 1	1 4 1 2 0 7 0 3 0 5 0 6 0 1	11/7560
2 7 1	1 3 1 5 1 6 0 7 0 4 0 8 0 1	-1/135	2 7 1	1 5 1 2 0 3 0 6 0 7 0 8 0 1	-2/315
2 7 1	3 6 1 7 1 5 0 4 0 8 0 1 0 1	-1/45	2 7 1	1 4 1 8 0 7 0 3 0 5 0 6 0 1	-4/315
2 7 1	3 8 2 7 1 5 0 4 0 1 0 1 0 1	-11/2160	2 7 1	4 5 1 7 0 8 0 6 0 3 0 1 0 1	4/315
2 7 1	1 8 1 7 1 6 0 2 0 4 0 3 0 1	-1/216	2 7 1	1 5 1 4 0 7 0 6 0 3 0 2 0 1	-31/7560
2 7 1	1 7 1 6 1 5 0 4 0 3 0 2 0 1	-1/1296	2 7 1	1 5 1 2 0 6 0 7 0 3 0 8 0 1	8/945
2 7 1	4 8 2 6 3 7 0 1 0 1 0 1 0 1	31/22680	2 7 1	1 8 1 2 0 7 0 3 0 5 0 6 0 1	-2/945
2 7 1	3 5 1 4 1 7 0 6 0 8 0 1 0 1	-16/945	2 7 1	1 4 1 5 0 6 0 3 0 7 0 2 0 1	-1/540
2 7 1	4 5 2 6 1 8 0 7 0 1 0 1 0 1	16/945	2 7 1	3 5 1 4 0 6 0 2 0 8 0 1 0 1	-37/15120
2 7 1	1 4 1 8 0 2 0 7 0 1 0 1 0 1	1/54	2 7 1	1 6 1 5 0 7 0 8 0 2 0 3 0 1	1/270
2 7 1	1 5 1 4 0 3 0 2 0 1 0 1 0 1	1/432	2 7 1	1 6 1 5 0 3 0 7 0 2 0 8 0 1	-1/135
2 7 1	1 5 1 4 0 2 0 3 0 1 0 1 0 1	1/360	2 7 1	1 8 0 2 0 1 0 1 0 1 0 1 0 1	-1/72
2 7 1	3 8 1 4 0 2 0 1 0 1 0 1 0 1	1/135	2 7 1	1 8 0 7 0 1 0 1 0 1 0 1 0 1	-1/54
2 7 1	1 5 1 8 0 3 0 4 0 1 0 1 0 1	1/15	2 7 1	1 3 0 2 0 1 0 1 0 1 0 1 0 1	-1/720
2 7 1	1 3 1 4 0 5 0 8 0 1 0 1 0 1	-1/45	2 7 1	3 8 1 7 0 1 0 1 0 1 0 1 0 1	1/45
2 7 1	1 5 1 8 0 3 0 7 0 1 0 1 0 1	1/18	2 7 1	1 3 1 8 0 2 0 1 0 1 0 1 0 1	2/135
2 7 1	1 7 1 8 0 5 0 3 0 1 0 1 0 1	4/135	2 7 1	1 4 1 7 0 8 0 1 0 1 0 1 0 1	1/18
2 7 1	4 8 1 7 0 6 0 1 0 1 0 1 0 1	-2/135	2 7 1	1 8 1 7 0 6 0 1 0 1 0 1 0 1	1/54
2 7 1	3 8 1 7 0 6 0 1 0 1 0 1 0 1	2/135	2 7 1	1 4 1 8 0 2 0 1 0 1 0 1 0 1	1/108
2 7 1	1 3 1 5 0 7 0 8 0 1 0 1 0 1	4/135	2 7 1	3 8 0 7 0 1 0 1 0 1 0 1 0 1	1/45
2 7 1	1 5 1 7 0 8 0 6 0 1 0 1 0 1	1/27	2 7 1	1 3 0 4 0 8 0 1 0 1 0 1 0 1	-2/135
2 7 1	1 5 1 4 0 3 0 8 0 1 0 1 0 1	1/36	2 7 1	1 8 0 2 0 7 0 1 0 1 0 1 0 1	-1/18
2 7 1	1 8 1 2 1 6 0 4 0 3 0 1 0 1	4/315	2 7 1	1 8 0 7 0 6 0 1 0 1 0 1 0 1	-1/54
2 7 1	3 8 1 5 1 7 0 4 0 1 0 1 0 1	-2/105	2 7 1	1 3 0 2 0 8 0 1 0 1 0 1 0 1	-1/108
2 7 1	1 3 1 5 1 8 0 6 0 4 0 1 0 1	-2/105	2 7 1	1 4 1 5 1 3 0 2 0 1 0 1 0 1	1/540
2 7 1	1 3 1 4 1 5 0 6 0 8 0 1 0 1	8/945	2 7 1	4 6 1 7 1 8 0 1 0 1 0 1 0 1	-2/45
2 7 1	1 6 1 5 1 3 0 8 0 4 0 1 0 1	-2/105	2 7 1	1 4 1 7 1 8 0 2 0 1 0 1 0 1	-4/135
2 7 1	4 8 1 5 1 6 0 7 0 1 0 1 0 1	-2/45	2 7 1	1 3 1 4 1 7 0 8 0 1 0 1 0 1	1/135
2 7 1	1 7 1 5 1 2 0 8 0 4 0 1 0 1	-4/135	2 7 1	1 5 1 8 1 7 0 6 0 1 0 1 0 1	-1/54
2 7 1	1 3 1 6 1 2 0 8 0 4 0 1 0 1	1/270	2 7 1	1 5 1 8 1 7 0 2 0 1 0 1 0 1	-1/108
2 7 1	1 8 1 5 1 3 0 6 0 4 0 1 0 1	-1/180	2 7 1	1 5 1 4 1 8 0 3 0 1 0 1 0 1	1/45
2 7 1	1 7 1 8 1 3 0 6 0 4 0 1 0 1	-1/135	2 7 1	1 3 1 5 1 8 0 4 0 1 0 1 0 1	-1/90
2 7 1	4 5 1 7 1 8 0 6 0 1 0 1 0 1	-1/45	2 7 1	1 3 0 5 0 2 0 4 0 1 0 1 0 1	1/540
2 7 1	1 7 1 5 1 6 0 4 0 8 0 1 0 1	-1/45	2 7 1	4 8 0 7 0 6 0 1 0 1 0 1 0 1	2/45

2 7 1	1 4 0 8 0 5 0 7 0 1 0 1 0 1	-4/135
2 7 1	1 7 0 5 0 8 0 4 0 1 0 1 0 1	1/135
2 7 1	1 6 0 8 0 7 0 2 0 1 0 1 0 1	-1/54
2 7 1	1 5 0 8 0 7 0 2 0 1 0 1 0 1	-1/108
2 7 1	1 4 0 5 0 8 0 2 0 1 0 1 0 1	-1/90
2 7 1	1 4 0 2 0 5 0 8 0 1 0 1 0 1	1/45
2 7 1	1 5 1 8 1 3 1 4 0 2 0 1 0 1	2/315
2 7 1	1 8 1 4 1 5 1 6 0 3 0 1 0 1	-1/270
2 7 1	1 4 1 6 1 5 1 7 0 8 0 1 0 1	-1/135
2 7 1	1 4 1 5 1 6 1 8 0 3 0 1 0 1	-2/105
2 7 1	1 4 1 6 1 5 1 8 0 3 0 1 0 1	-1/270
2 7 1	1 8 1 6 1 5 1 7 0 4 0 1 0 1	-2/135
2 7 1	1 5 1 7 1 8 1 6 0 4 0 1 0 1	1/135
2 7 1	4 8 1 2 1 5 1 7 0 1 0 1 0 1	2/315
2 7 1	1 3 0 6 0 8 0 4 0 5 0 1 0 1	-2/315
2 7 1	1 4 0 8 0 5 0 6 0 2 0 1 0 1	1/270
2 7 1	1 8 0 5 0 2 0 6 0 7 0 1 0 1	1/135
2 7 1	1 4 0 6 0 5 0 8 0 2 0 1 0 1	2/105
2 7 1	1 4 0 5 0 2 0 6 0 8 0 1 0 1	1/270
2 7 1	1 5 0 6 0 7 0 8 0 2 0 1 0 1	-1/135
2 7 1	1 5 0 8 0 2 0 6 0 7 0 1 0 1	2/135
2 7 1	4 8 0 2 0 5 0 7 0 1 0 1 0 1	2/315
2 7 1	1 6 1 7 1 5 1 3 1 4 0 2 0 1	-1/945
2 7 1	1 7 1 6 1 8 1 4 1 5 0 3 0 1	-4/945
2 7 1	3 4 1 8 1 5 1 6 1 7 0 1 0 1	2/945
2 7 1	1 6 1 4 1 8 1 7 1 5 0 3 0 1	4/945
2 7 1	1 8 1 6 1 7 1 4 1 5 0 2 0 1	-2/945
2 7 1	1 3 0 7 0 2 0 6 0 4 0 5 0 1	-1/945
2 7 1	1 4 0 2 0 7 0 8 0 5 0 6 0 1	-4/945
2 7 1	3 4 0 8 0 5 0 6 0 7 0 1 0 1	-2/945
2 7 1	1 3 0 8 0 7 0 2 0 5 0 6 0 1	-2/945
2 7 1	1 4 0 7 0 5 0 8 0 2 0 6 0 1	4/945
2 7 1	0 1 0 1 0 1 0 1 0 1 0 1 0 1	1/5040
2 7 1	1 8 0 1 0 1 0 1 0 1 0 1 0 1	-1/360
2 7 1	0 8 0 1 0 1 0 1 0 1 0 1 0 1	1/360
2 7 1	1 8 1 7 0 1 0 1 0 1 0 1 0 1	1/108
2 7 1	0 8 0 7 0 1 0 1 0 1 0 1 0 1	1/108
2 7 1	1 3 1 4 1 8 0 1 0 1 0 1 0 1	1/270
2 7 1	1 8 1 7 1 6 0 1 0 1 0 1 0 1	-1/162
2 7 1	0 3 0 4 0 8 0 1 0 1 0 1 0 1	-1/270
2 7 1	0 8 0 7 0 6 0 1 0 1 0 1 0 1	1/162
2 7 1	1 8 1 4 1 5 1 7 0 1 0 1 0 1	-1/135
2 7 1	0 8 0 4 0 5 0 7 0 1 0 1 0 1	-1/135
2 7 1	1 6 1 8 1 5 1 3 1 4 0 1 0 1	-2/945
2 7 1	0 6 0 8 0 5 0 3 0 4 0 1 0 1	2/945



















### C.3 Associativity of $\star_{\text{aff}}^{\text{red}} \bmod \bar{o}(\hbar^7)$

We inspect that the reduced star product  $\star_{\text{aff}}^{\text{red}} \bmod \bar{o}(\hbar^7)$  remains associative.

We read the star product:

```
[2]: from gcaops.graph.formality_graph_complex import FormalityGraphComplex
FGC = FormalityGraphComplex(SR, lazy=True); FGC
```

[2]: Formality graph complex over Symbolic Ring with Basis consisting of representatives of isomorphism classes of formality graphs with no automorphisms that induce an odd permutation on edges

```
[7]: affine_star7_reduced = FGC.element_from_kgs_encoding(open('data/affine_star7_reduced.
↳txt').read().rstrip())
```

We calculate the associator:

```
[9]: %time affine_assoc7_reduced = affine_star7_reduced.insertion(0, affine_star7_reduced,
↳max_num_aerial=7, max_aerial_in_degree=1) - affine_star7_reduced.insertion(1,
↳affine_star7_reduced, max_num_aerial=7, max_aerial_in_degree=1)
```

CPU times: user 49.7 s, sys: 0 ns, total: 49.7 s

Wall time: 49.3 s

We prove the associativity:

```
[12]: from gcaops.graph.leibniz_graph_expansion import
↳kontsevich_graph_sum_to_leibniz_graph_sum
from gcaops.graph.leibniz_graph_expansion import
↳leibniz_graph_sum_to_kontsevich_graph_sum

for order in range(2,8):
    print('h^{:}'.format(order))
    assoc_part = affine_assoc7_reduced.homogeneous_part(3, order, 2*order)
    diff_orders = list(assoc_part.differential_orders())
    print('Number of differential orders:', len(diff_orders))
    for diff_order in diff_orders:
        print(diff_order, end=': ', flush=True)
        component = assoc_part.part_of_differential_order(diff_order)
        component_leibniz = kontsevich_graph_sum_to_leibniz_graph_sum(component,
↳max_aerial_in_degree=1, verbose=True)
        print(leibniz_graph_sum_to_kontsevich_graph_sum(component_leibniz,
↳max_aerial_in_degree=1) == component)
```

h<sup>2</sup>:

Number of differential orders: 1

(1, 1, 1): 3K -> +1L -> +0K

True

h<sup>3</sup>:

Number of differential orders: 7

(1, 1, 2): 4K -> +3L -> +2K

True

(2, 1, 2): 3K -> +1L -> +0K

True

```

(1, 2, 2): 3K -> +1L -> +0K
True
(1, 1, 1): 2K -> +2L -> +2K
True
(2, 2, 1): 3K -> +1L -> +0K
True
(1, 2, 1): 4K -> +3L -> +2K
True
(2, 1, 1): 4K -> +3L -> +2K
True
h^4:
Number of differential orders: 22
(1, 1, 2): 4K -> +6L -> +5K
True
(1, 1, 3): 8K -> +6L -> +2K
True
(2, 1, 2): 18K -> +17L -> +8K
True
(1, 2, 2): 18K -> +18L -> +8K
True
(2, 1, 3): 7K -> +4L -> +2K
True
(1, 2, 3): 7K -> +4L -> +2K
True
(3, 1, 3): 3K -> +1L -> +0K
True
(2, 2, 3): 3K -> +1L -> +0K
True
(1, 3, 3): 3K -> +1L -> +0K
True
(3, 1, 2): 7K -> +4L -> +2K
True
(2, 2, 2): 16K -> +12L -> +8K
True
(1, 3, 2): 7K -> +4L -> +2K
True
(3, 1, 1): 8K -> +6L -> +2K
True
(2, 2, 1): 20K -> +18L -> +6K
True
(1, 3, 1): 9K -> +6L -> +1K
True
(2, 1, 1): 4K -> +6L -> +5K
True
(3, 2, 2): 3K -> +1L -> +0K
True
(2, 3, 2): 3K -> +1L -> +0K
True
(1, 2, 1): 2K -> +2L -> +4K -> +8L -> +6K
True
(2, 3, 1): 7K -> +4L -> +2K
True
(3, 2, 1): 7K -> +4L -> +2K
True
(3, 3, 1): 3K -> +1L -> +0K
True
h^5:
Number of differential orders: 50

```

(2, 1, 2): 8K -> +18L -> +18K  
 True  
 (1, 2, 2): 12K -> +30L -> +23K  
 True  
 (3, 1, 2): 40K -> +58L -> +35K  
 True  
 (2, 2, 2): 77K -> +110L -> +58K -> +37L -> +19K  
 True  
 (2, 1, 3): 40K -> +55L -> +33K  
 True  
 (1, 2, 3): 43K -> +61L -> +30K  
 True  
 (2, 2, 3): 65K -> +59L -> +23K  
 True  
 (3, 2, 2): 71K -> +62L -> +21K  
 True  
 (2, 3, 2): 68K -> +60L -> +20K  
 True  
 (1, 3, 2): 36K -> +50L -> +25K -> +11L -> +8K  
 True  
 (3, 1, 3): 31K -> +28L -> +13K  
 True  
 (1, 3, 3): 29K -> +28L -> +15K  
 True  
 (3, 2, 3): 22K -> +14L -> +8K  
 True  
 (2, 3, 3): 22K -> +14L -> +8K  
 True  
 (3, 1, 4): 7K -> +4L -> +2K  
 True  
 (2, 2, 4): 10K -> +5L -> +2K  
 True  
 (1, 3, 4): 7K -> +4L -> +2K  
 True  
 (4, 2, 3): 3K -> +1L -> +0K  
 True  
 (3, 3, 3): 3K -> +1L -> +0K  
 True  
 (4, 1, 3): 7K -> +4L -> +2K  
 True  
 (2, 4, 3): 3K -> +1L -> +0K  
 True  
 (1, 4, 3): 7K -> +4L -> +2K  
 True  
 (4, 1, 4): 3K -> +1L -> +0K  
 True  
 (3, 2, 4): 3K -> +1L -> +0K  
 True  
 (2, 3, 4): 3K -> +1L -> +0K  
 True  
 (1, 4, 4): 3K -> +1L -> +0K  
 True  
 (2, 1, 4): 14K -> +10L -> +4K  
 True  
 (1, 2, 4): 14K -> +10L -> +4K  
 True  
 (4, 2, 2): 10K -> +5L -> +2K  
 True

```

(3, 3, 2): 22K -> +14L -> +8K
True
(2, 4, 2): 10K -> +5L -> +2K
True
(4, 1, 2): 12K -> +9L -> +4K
True
(1, 4, 2): 13K -> +9L -> +3K
True
(1, 1, 4): 10K -> +9L -> +5K -> +1L -> +0K
True
(4, 2, 1): 12K -> +9L -> +4K
True
(4, 1, 1): 10K -> +9L -> +5K -> +1L -> +0K
True
(3, 3, 1): 33K -> +29L -> +11K
True
(3, 2, 1): 44K -> +61L -> +31K
True
(2, 4, 1): 13K -> +9L -> +3K
True
(2, 3, 1): 35K -> +50L -> +26K -> +11L -> +8K
True
(1, 4, 1): 8K -> +10L -> +7K
True
(1, 1, 3): 4K -> +10L -> +10K
True
(3, 1, 1): 4K -> +10L -> +10K
True
(2, 2, 1): 12K -> +30L -> +23K
True
(1, 3, 1): 6K -> +16L -> +12K
True
(4, 3, 2): 3K -> +1L -> +0K
True
(3, 4, 2): 3K -> +1L -> +0K
True
(3, 4, 1): 7K -> +4L -> +2K
True
(4, 3, 1): 7K -> +4L -> +2K
True
(4, 4, 1): 3K -> +1L -> +0K
True
h^6:
Number of differential orders: 95
(2, 2, 2): 46K -> +111L -> +104K -> +202L -> +130K
True
(1, 3, 2): 32K -> +80L -> +57K -> +96L -> +60K
True
(3, 1, 2): 27K -> +66L -> +74K -> +135L -> +66K
True
(3, 2, 2): 216K -> +423L -> +259K -> +205L -> +76K
True
(2, 3, 2): 225K -> +448L -> +239K
True
(4, 1, 2): 64K -> +109L -> +74K -> +53L -> +25K
True
(4, 2, 2): 164K -> +201L -> +90K -> +19L -> +10K
True

```

(3, 3, 2): 268K -> +362L -> +165K -> +55L -> +23K  
 True  
 (2, 2, 3): 219K -> +422L -> +251K -> +205L -> +83K  
 True  
 (1, 3, 3): 136K -> +246L -> +121K -> +73L -> +33K  
 True  
 (3, 1, 3): 113K -> +203L -> +141K -> +112L -> +36K  
 True  
 (3, 2, 3): 266K -> +358L -> +171K -> +57L -> +18K  
 True  
 (2, 3, 3): 270K -> +357L -> +160K -> +59L -> +26K  
 True  
 (4, 1, 3): 87K -> +106L -> +56K -> +18L -> +8K  
 True  
 (4, 2, 3): 99K -> +83L -> +30K  
 True  
 (3, 3, 3): 169K -> +139L -> +44K  
 True  
 (2, 4, 2): 147K -> +182L -> +78K -> +18L -> +11K  
 True  
 (1, 4, 2): 76K -> +126L -> +58K -> +32L -> +20K  
 True  
 (4, 3, 2): 99K -> +83L -> +30K  
 True  
 (3, 4, 2): 92K -> +78L -> +29K  
 True  
 (5, 1, 2): 24K -> +18L -> +8K -> +1L -> +0K  
 True  
 (5, 2, 2): 25K -> +17L -> +8K  
 True  
 (1, 4, 3): 74K -> +88L -> +41K -> +17L -> +10K  
 True  
 (2, 4, 3): 90K -> +77L -> +31K  
 True  
 (5, 2, 3): 14K -> +7L -> +2K  
 True  
 (4, 3, 3): 28K -> +16L -> +8K  
 True  
 (3, 4, 3): 28K -> +16L -> +8K  
 True  
 (3, 2, 4): 97K -> +81L -> +30K  
 True  
 (2, 3, 4): 95K -> +80L -> +32K  
 True  
 (1, 4, 4): 36K -> +31L -> +15K  
 True  
 (4, 1, 4): 41K -> +33L -> +14K  
 True  
 (4, 2, 4): 22K -> +14L -> +8K  
 True  
 (3, 3, 4): 28K -> +16L -> +8K  
 True  
 (2, 4, 4): 22K -> +14L -> +8K  
 True  
 (2, 5, 3): 10K -> +5L -> +2K  
 True  
 (1, 5, 3): 16K -> +10L -> +3K  
 True

(5, 1, 3): 18K -> +12L -> +5K  
 True  
 (5, 3, 3): 3K -> +1L -> +0K  
 True  
 (4, 4, 3): 3K -> +1L -> +0K  
 True  
 (3, 5, 3): 3K -> +1L -> +0K  
 True  
 (1, 5, 4): 7K -> +4L -> +2K  
 True  
 (5, 2, 4): 3K -> +1L -> +0K  
 True  
 (4, 3, 4): 3K -> +1L -> +0K  
 True  
 (3, 4, 4): 3K -> +1L -> +0K  
 True  
 (2, 5, 4): 3K -> +1L -> +0K  
 True  
 (2, 2, 4): 167K -> +202L -> +87K -> +18L -> +10K  
 True  
 (1, 3, 4): 91K -> +114L -> +52K  
 True  
 (3, 1, 4): 88K -> +106L -> +55K -> +18L -> +8K  
 True  
 (2, 5, 2): 20K -> +13L -> +5K  
 True  
 (1, 5, 2): 20K -> +17L -> +8K  
 True  
 (5, 3, 2): 14K -> +7L -> +2K  
 True  
 (4, 4, 2): 22K -> +14L -> +8K  
 True  
 (3, 5, 2): 10K -> +5L -> +2K  
 True  
 (1, 2, 4): 80K -> +129L -> +62K  
 True  
 (2, 1, 4): 64K -> +109L -> +74K -> +53L -> +25K  
 True  
 (4, 2, 1): 80K -> +129L -> +62K  
 True  
 (3, 3, 1): 134K -> +244L -> +125K -> +75L -> +30K  
 True  
 (2, 4, 1): 76K -> +126L -> +58K -> +32L -> +20K  
 True  
 (4, 3, 1): 91K -> +113L -> +52K  
 True  
 (3, 4, 1): 75K -> +92L -> +42K -> +13L -> +8K  
 True  
 (2, 5, 1): 20K -> +17L -> +8K  
 True  
 (1, 5, 1): 18K -> +15L -> +3K  
 True  
 (5, 2, 1): 24K -> +18L -> +8K -> +1L -> +0K  
 True  
 (5, 1, 1): 14K -> +15L -> +7K  
 True  
 (5, 3, 1): 18K -> +12L -> +5K  
 True



(4, 4, 1): 39K -> +32L -> +12K  
 True  
 (3, 5, 1): 16K -> +10L -> +3K  
 True  
 (1, 2, 3): 38K -> +92L -> +74K -> +112L -> +58K  
 True  
 (2, 1, 3): 28K -> +68L -> +75K -> +134L -> +65K  
 True  
 (3, 2, 1): 38K -> +93L -> +71K -> +109L -> +57K  
 True  
 (2, 3, 1): 32K -> +80L -> +57K -> +96L -> +60K  
 True  
 (1, 4, 1): 8K -> +18L -> +18K -> +34L -> +22K  
 True  
 (4, 1, 1): 12K -> +25L -> +19K  
 True  
 (5, 4, 2): 3K -> +1L -> +0K  
 True  
 (4, 5, 2): 3K -> +1L -> +0K  
 True  
 (1, 1, 4): 12K -> +25L -> +19K  
 True  
 (4, 1, 5): 9K -> +5L -> +2K  
 True  
 (3, 2, 5): 14K -> +7L -> +2K  
 True  
 (2, 3, 5): 14K -> +7L -> +2K  
 True  
 (1, 4, 5): 9K -> +5L -> +2K  
 True  
 (3, 1, 5): 20K -> +13L -> +5K  
 True  
 (2, 2, 5): 29K -> +19L -> +8K  
 True  
 (1, 3, 5): 20K -> +13L -> +5K  
 True  
 (2, 1, 5): 24K -> +18L -> +8K -> +1L -> +0K  
 True  
 (1, 2, 5): 24K -> +18L -> +8K -> +1L -> +0K  
 True  
 (1, 1, 5): 14K -> +15L -> +7K  
 True  
 (5, 1, 5): 3K -> +1L -> +0K  
 True  
 (4, 2, 5): 3K -> +1L -> +0K  
 True  
 (3, 3, 5): 3K -> +1L -> +0K  
 True  
 (2, 4, 5): 3K -> +1L -> +0K  
 True  
 (1, 5, 5): 3K -> +1L -> +0K  
 True  
 (5, 1, 4): 9K -> +5L -> +2K  
 True  
 (4, 5, 1): 7K -> +4L -> +2K  
 True  
 (5, 4, 1): 9K -> +5L -> +2K  
 True

```

(5, 5, 1): 3K -> +1L -> +0K
True
h^7:
Number of differential orders: 161
(3, 2, 2): 87K -> +314L -> +353K -> +915L -> +533K
True
(2, 3, 2): 107K -> +372L -> +341K -> +835L -> +468K
True
(3, 3, 2): 762K -> +1930L -> +1258K -> +1463L -> +475K
True
(2, 4, 2): 509K -> +1243L -> +699K -> +669L -> +269K
True
(4, 2, 2): 455K -> +1114L -> +766K -> +836L -> +295K
True
(4, 3, 2): 777K -> +1378L -> +672K -> +361L -> +125K
True
(3, 4, 2): 799K -> +1399L -> +640K -> +330L -> +113K
True
(5, 2, 2): 288K -> +438L -> +236K -> +109L -> +40K
True
(5, 3, 2): 264K -> +296L -> +133K -> +22L -> +5K
True
(4, 4, 2): 411K -> +506L -> +214K -> +61L -> +26K
True
(3, 2, 3): 638K -> +1612L -> +1293K -> +1732L -> +570K
True
(2, 3, 3): 760K -> +1919L -> +1251K -> +1457L -> +494K
True
(4, 2, 3): 736K -> +1333L -> +722K -> +421L -> +130K
True
(3, 3, 3): 1252K -> +2303L -> +1110K -> +658L -> +205K
True
(2, 4, 3): 795K -> +1392L -> +636K -> +336L -> +120K
True
(4, 3, 3): 632K -> +785L -> +334K -> +88L -> +30K
True
(3, 4, 3): 629K -> +781L -> +326K -> +86L -> +31K
True
(5, 2, 3): 262K -> +295L -> +135K -> +23L -> +5K
True
(5, 3, 3): 135K -> +107L -> +42K
True
(4, 4, 3): 208K -> +167L -> +57K
True
(3, 5, 2): 231K -> +259L -> +105K -> +21L -> +11K
True
(2, 5, 2): 297K -> +447L -> +196K -> +71L -> +33K
True
(5, 4, 2): 107K -> +86L -> +32K
True
(4, 5, 2): 101K -> +82L -> +30K
True
(6, 2, 2): 42K -> +30L -> +13K -> +1L -> +0K
True
(6, 3, 2): 28K -> +18L -> +8K
True
(2, 5, 3): 227K -> +254L -> +107K -> +26L -> +13K
True

```

(3, 5, 3): 117K -> +96L -> +40K  
 True  
 (5, 4, 3): 32K -> +18L -> +8K  
 True  
 (4, 5, 3): 28K -> +16L -> +8K  
 True  
 (4, 2, 4): 402K -> +494L -> +219K -> +63L -> +20K  
 True  
 (3, 3, 4): 641K -> +791L -> +330K -> +85L -> +29K  
 True  
 (2, 4, 4): 408K -> +499L -> +212K -> +68L -> +31K  
 True  
 (5, 2, 4): 103K -> +83L -> +32K  
 True  
 (4, 3, 4): 214K -> +169L -> +57K  
 True  
 (3, 4, 4): 210K -> +167L -> +57K  
 True  
 (2, 5, 4): 100K -> +81L -> +31K  
 True  
 (5, 3, 4): 32K -> +18L -> +8K  
 True  
 (4, 4, 4): 34K -> +18L -> +8K  
 True  
 (3, 5, 4): 28K -> +16L -> +8K  
 True  
 (3, 6, 3): 10K -> +5L -> +2K  
 True  
 (2, 6, 3): 23K -> +14L -> +5K  
 True  
 (6, 3, 3): 10K -> +5L -> +2K  
 True  
 (6, 2, 3): 28K -> +18L -> +8K  
 True  
 (6, 4, 3): 3K -> +1L -> +0K  
 True  
 (5, 5, 3): 3K -> +1L -> +0K  
 True  
 (4, 6, 3): 3K -> +1L -> +0K  
 True  
 (2, 6, 4): 10K -> +5L -> +2K  
 True  
 (6, 2, 4): 10K -> +5L -> +2K  
 True  
 (6, 3, 4): 3K -> +1L -> +0K  
 True  
 (5, 4, 4): 3K -> +1L -> +0K  
 True  
 (4, 5, 4): 3K -> +1L -> +0K  
 True  
 (3, 6, 4): 3K -> +1L -> +0K  
 True  
 (3, 2, 4): 733K -> +1340L -> +727K -> +411L -> +127K  
 True  
 (2, 3, 4): 779K -> +1389L -> +675K -> +352L -> +122K  
 True  
 (3, 6, 2): 23K -> +14L -> +5K  
 True

(2, 6, 2): 36K -> +27L -> +11K  
 True  
 (6, 4, 2): 10K -> +5L -> +2K  
 True  
 (5, 5, 2): 24K -> +15L -> +8K  
 True  
 (4, 6, 2): 10K -> +5L -> +2K  
 True  
 (2, 2, 4): 456K -> +1117L -> +772K -> +848L -> +289K  
 True  
 (5, 3, 1): 160K -> +243L -> +144K -> +78L -> +36K  
 True  
 (5, 2, 1): 106K -> +224L -> +152K -> +124L -> +58K  
 True  
 (4, 4, 1): 259K -> +452L -> +238K -> +124L -> +48K  
 True  
 (4, 3, 1): 289K -> +712L -> +422K -> +353L -> +134K  
 True  
 (3, 5, 1): 158K -> +240L -> +122K -> +52L -> +23K  
 True  
 (3, 4, 1): 280K -> +681L -> +392K -> +359L -> +142K  
 True  
 (2, 5, 1): 107K -> +226L -> +136K -> +94L -> +42K  
 True  
 (5, 4, 1): 114K -> +133L -> +63K  
 True  
 (4, 5, 1): 86K -> +99L -> +50K -> +17L -> +7K  
 True  
 (3, 6, 1): 24K -> +20L -> +10K  
 True  
 (2, 6, 1): 30K -> +28L -> +12K  
 True  
 (6, 3, 1): 28K -> +21L -> +10K -> +1L -> +0K  
 True  
 (6, 2, 1): 34K -> +32L -> +16K -> +2L -> +1K  
 True  
 (6, 4, 1): 15K -> +10L -> +4K  
 True  
 (5, 5, 1): 35K -> +30L -> +12K  
 True  
 (4, 6, 1): 16K -> +10L -> +3K  
 True  
 (2, 2, 3): 87K -> +318L -> +361K -> +922L -> +510K  
 True  
 (4, 2, 1): 42K -> +139L -> +115K -> +221L -> +128K  
 True  
 (3, 3, 1): 78K -> +276L -> +227K -> +453L -> +240K  
 True  
 (2, 4, 1): 49K -> +164L -> +125K -> +230L -> +135K  
 True  
 (1, 5, 2): 111K -> +233L -> +134K -> +87L -> +40K  
 True  
 (5, 1, 2): 94K -> +192L -> +162K -> +157L -> +60K  
 True  
 (1, 5, 3): 156K -> +237L -> +123K -> +55L -> +24K  
 True  
 (5, 1, 3): 147K -> +232L -> +160K -> +94L -> +42K  
 True

(6, 5, 2): 3K -> +1L -> +0K  
 True  
 (5, 6, 2): 3K -> +1L -> +0K  
 True  
 (1, 6, 2): 30K -> +28L -> +12K  
 True  
 (6, 1, 2): 34K -> +32L -> +16K -> +2L -> +1K  
 True  
 (1, 6, 3): 24K -> +20L -> +10K  
 True  
 (6, 1, 3): 28K -> +21L -> +10K -> +1L -> +0K  
 True  
 (3, 1, 4): 204K -> +481L -> +461K -> +600L -> +201K  
 True  
 (1, 3, 4): 287K -> +703L -> +423K -> +368L -> +135K  
 True  
 (4, 1, 5): 111K -> +127L -> +67K -> +15L -> +3K  
 True  
 (3, 2, 5): 261K -> +296L -> +135K -> +21L -> +4K  
 True  
 (2, 3, 5): 263K -> +297L -> +133K -> +20L -> +4K  
 True  
 (1, 4, 5): 113K -> +132L -> +65K  
 True  
 (3, 1, 5): 145K -> +231L -> +160K -> +94L -> +42K  
 True  
 (2, 2, 5): 296K -> +444L -> +234K -> +105L -> +38K  
 True  
 (1, 3, 5): 159K -> +243L -> +145K -> +77L -> +34K  
 True  
 (4, 1, 4): 231K -> +410L -> +265K -> +177L -> +72K  
 True  
 (1, 4, 4): 258K -> +451L -> +239K -> +125L -> +48K  
 True  
 (2, 1, 5): 93K -> +191L -> +162K -> +152L -> +55K  
 True  
 (1, 2, 5): 104K -> +223L -> +149K -> +119L -> +57K  
 True  
 (4, 1, 3): 202K -> +470L -> +456K -> +601L -> +205K  
 True  
 (1, 4, 3): 283K -> +693L -> +396K -> +355L -> +143K  
 True  
 (5, 1, 6): 9K -> +5L -> +2K  
 True  
 (4, 2, 6): 10K -> +5L -> +2K  
 True  
 (3, 3, 6): 10K -> +5L -> +2K  
 True  
 (2, 4, 6): 10K -> +5L -> +2K  
 True  
 (1, 5, 6): 9K -> +5L -> +2K  
 True  
 (4, 1, 6): 17K -> +11L -> +4K  
 True  
 (3, 2, 6): 32K -> +20L -> +8K  
 True  
 (2, 3, 6): 32K -> +20L -> +8K  
 True

```

(1, 4, 6): 17K -> +11L -> +4K
True
(5, 1, 5): 35K -> +29L -> +12K
True
(4, 2, 5): 104K -> +83L -> +29K
True
(3, 3, 5): 135K -> +108L -> +42K
True
(2, 4, 5): 107K -> +86L -> +30K
True
(1, 5, 5): 36K -> +30L -> +11K
True
(3, 1, 6): 28K -> +21L -> +10K -> +1L -> +0K
True
(2, 2, 6): 46K -> +32L -> +13K -> +1L -> +0K
True
(1, 3, 6): 28K -> +21L -> +10K -> +1L -> +0K
True
(5, 1, 4): 111K -> +127L -> +66K -> +14L -> +2K
True
(1, 5, 4): 86K -> +98L -> +48K -> +18L -> +9K
True
(2, 1, 6): 36K -> +33L -> +16K -> +2L -> +1K
True
(1, 2, 6): 36K -> +33L -> +16K -> +2L -> +1K
True
(1, 1, 6): 16K -> +18L -> +12K -> +3L -> +0K
True
(6, 1, 6): 3K -> +1L -> +0K
True
(5, 2, 6): 3K -> +1L -> +0K
True
(4, 3, 6): 3K -> +1L -> +0K
True
(3, 4, 6): 3K -> +1L -> +0K
True
(2, 5, 6): 3K -> +1L -> +0K
True
(1, 6, 6): 3K -> +1L -> +0K
True
(6, 1, 5): 9K -> +5L -> +2K
True
(5, 2, 5): 22K -> +14L -> +8K
True
(4, 3, 5): 32K -> +18L -> +8K
True
(3, 4, 5): 32K -> +18L -> +8K
True
(2, 5, 5): 24K -> +15L -> +8K
True
(1, 6, 5): 7K -> +4L -> +2K
True
(6, 1, 4): 15K -> +10L -> +4K
True
(1, 6, 4): 16K -> +10L -> +3K
True
(3, 1, 3): 32K -> +116L -> +161K -> +422L -> +272K
True

```

```

(1, 3, 3): 78K -> +277L -> +225K -> +442L -> +240K
True
(2, 1, 4): 21K -> +65L -> +72K -> +186L -> +137K
True
(1, 2, 4): 42K -> +139L -> +113K -> +219L -> +128K
True
(4, 1, 2): 21K -> +65L -> +72K -> +186L -> +137K
True
(1, 4, 2): 49K -> +164L -> +125K -> +226L -> +132K
True
(6, 2, 5): 3K -> +1L -> +0K
True
(5, 3, 5): 3K -> +1L -> +0K
True
(4, 4, 5): 3K -> +1L -> +0K
True
(3, 5, 5): 3K -> +1L -> +0K
True
(2, 6, 5): 3K -> +1L -> +0K
True
(6, 1, 1): 16K -> +18L -> +12K -> +3L -> +0K
True
(1, 6, 1): 12K -> +18L -> +15K
True
(1, 1, 5): 6K -> +16L -> +16K
True
(5, 1, 1): 6K -> +16L -> +16K
True
(1, 5, 1): 10K -> +28L -> +20K
True
(6, 6, 1): 3K -> +1L -> +0K
True
(5, 6, 1): 7K -> +4L -> +2K
True
(6, 5, 1): 9K -> +5L -> +2K
True

```

We conclude that the Kontsevich graph expansion of the reduced affine  $\star$ -product itself is associative up to  $\bar{o}(\hbar^7)$ ; the analytic formula which one writes for  $f \star_{\text{aff}}^{\text{red}} g \bmod \bar{o}(\hbar^7)$  with arbitrary arguments  $f, g \in C^\infty(M)$  and any affine Poisson structure  $P$  in  $\star_{\text{aff}} \bmod \bar{o}(\hbar^7)$  is, we establish in this appendix, identically equal to the formula  $f \star_{\text{aff}} g \bmod \bar{o}(\hbar^7)$ : all the coefficients of differential polynomials in  $f, g$  and  $P^{ij}$  are rational numbers, and  $\zeta(3)^2/\pi^6$  is not met at all.





# Appendix D

## Flows $Q_\gamma(P)$ and factorizations

$$\llbracket P, Q_\gamma(P) \rrbracket = \diamond_\gamma(P, \llbracket P, P \rrbracket)$$

Here are the flows  $Q_\gamma(P)$  and the respective solutions  $\diamond_\gamma(P, \llbracket P, P \rrbracket)$  of the Poisson cocycle factorization problems  $\llbracket P, Q_\gamma(P) \rrbracket = \diamond_\gamma(P, \llbracket P, P \rrbracket)$  for the tetrahedral and pentagon-wheel flows.

### D.1 The tetrahedral flow $Q_{\gamma_3}(P)$

The tetrahedral flow  $Q_{\gamma_3}$ :

$[(2,3), (2,5), (3,4), (3,5), (4,1), (4,2), (5,0), (5,4)]$	-576
$[(2,4), (2,5), (3,2), (3,5), (4,3), (4,5), (5,0), (5,1)]$	-192
$[(2,3), (2,4), (3,4), (3,5), (4,1), (4,5), (5,0), (5,2)]$	-576

The Poisson differential  $\llbracket P, Q_{\gamma_3} \rrbracket$ :

$[(3,4), (3,5), (4,5), (4,6), (5,6), (5,7), (6,2), (6,3), (7,0), (7,1)]$	-2304
$[(3,5), (3,6), (4,3), (4,6), (5,4), (5,6), (6,2), (6,7), (7,0), (7,1)]$	768
$[(3,5), (3,6), (4,3), (4,6), (5,4), (5,7), (6,2), (6,5), (7,0), (7,1)]$	-2304
$[(3,5), (3,7), (4,3), (4,7), (5,4), (5,6), (6,1), (6,2), (7,0), (7,5)]$	-2304
$[(3,5), (3,7), (4,3), (4,7), (5,4), (5,7), (6,1), (6,2), (7,0), (7,6)]$	768
$[(3,4), (3,5), (4,5), (4,7), (5,6), (5,7), (6,1), (6,2), (7,0), (7,3)]$	2304
$[(3,4), (3,7), (4,6), (4,7), (5,2), (5,3), (6,1), (6,3), (7,0), (7,6)]$	2304
$[(3,4), (3,7), (4,6), (4,7), (5,2), (5,4), (6,1), (6,3), (7,0), (7,6)]$	2304
$[(3,4), (3,7), (4,6), (4,7), (5,2), (5,6), (6,1), (6,3), (7,0), (7,6)]$	2304
$[(3,4), (3,7), (4,6), (4,7), (5,2), (5,7), (6,1), (6,3), (7,0), (7,6)]$	2304
$[(3,5), (3,7), (4,3), (4,7), (5,4), (5,7), (6,2), (6,5), (7,0), (7,1)]$	2304
$[(3,5), (3,7), (4,3), (4,7), (5,4), (5,7), (6,2), (6,7), (7,0), (7,1)]$	768
$[(3,4), (3,6), (4,6), (4,7), (5,2), (5,3), (6,1), (6,7), (7,0), (7,3)]$	2304
$[(3,4), (3,6), (4,6), (4,7), (5,2), (5,4), (6,1), (6,7), (7,0), (7,3)]$	2304
$[(3,4), (3,6), (4,6), (4,7), (5,2), (5,6), (6,1), (6,7), (7,0), (7,3)]$	2304
$[(3,4), (3,6), (4,6), (4,7), (5,2), (5,7), (6,1), (6,7), (7,0), (7,3)]$	2304
$[(3,4), (3,6), (4,5), (4,6), (5,2), (5,3), (6,1), (6,5), (7,0), (7,3)]$	2304
$[(3,4), (3,6), (4,5), (4,6), (5,2), (5,3), (6,1), (6,5), (7,0), (7,4)]$	2304
$[(3,4), (3,6), (4,5), (4,6), (5,2), (5,3), (6,1), (6,5), (7,0), (7,5)]$	2304
$[(3,4), (3,6), (4,5), (4,6), (5,2), (5,3), (6,1), (6,5), (7,0), (7,6)]$	2304
$[(3,5), (3,6), (4,3), (4,6), (5,4), (5,6), (6,1), (6,2), (7,0), (7,5)]$	2304
$[(3,5), (3,6), (4,3), (4,6), (5,4), (5,6), (6,1), (6,2), (7,0), (7,5)]$	2304
$[(3,5), (3,6), (4,3), (4,6), (5,4), (5,6), (6,1), (6,2), (7,0), (7,6)]$	768
$[(3,4), (3,5), (4,5), (4,6), (5,2), (5,6), (6,1), (6,3), (7,0), (7,3)]$	2304
$[(3,4), (3,5), (4,5), (4,6), (5,2), (5,6), (6,1), (6,3), (7,0), (7,4)]$	2304
$[(3,4), (3,5), (4,5), (4,6), (5,2), (5,6), (6,1), (6,3), (7,0), (7,5)]$	2304
$[(3,4), (3,5), (4,5), (4,6), (5,2), (5,6), (6,1), (6,3), (7,0), (7,6)]$	2304
$[(3,4), (3,5), (4,5), (4,6), (5,2), (5,6), (6,1), (6,3), (7,0), (7,6)]$	2304
$[(3,4), (3,5), (4,5), (4,6), (5,2), (5,6), (6,1), (6,3), (7,0), (7,2)]$	2304
$[(3,5), (3,6), (4,3), (4,6), (5,4), (5,6), (6,1), (6,7), (7,0), (7,2)]$	-768
$[(3,5), (3,6), (4,3), (4,6), (5,4), (5,7), (6,1), (6,5), (7,0), (7,2)]$	2304
$[(3,4), (3,7), (4,5), (4,7), (5,2), (5,3), (6,1), (6,3), (7,0), (7,5)]$	-2304
$[(3,4), (3,7), (4,5), (4,7), (5,2), (5,3), (6,1), (6,4), (7,0), (7,5)]$	-2304

$[(3,4), (3,7), (4,5), (4,7), (5,2), (5,3), (6,1), (6,5), (7,0), (7,5)]$	-2304
$[(3,4), (3,7), (4,5), (4,7), (5,2), (5,3), (6,1), (6,7), (7,0), (7,5)]$	-2304
$[(3,5), (3,7), (4,3), (4,7), (5,4), (5,7), (6,1), (6,5), (7,0), (7,2)]$	-2304
$[(3,5), (3,7), (4,3), (4,7), (5,4), (5,7), (6,1), (6,7), (7,0), (7,2)]$	-768
$[(3,4), (3,5), (4,5), (4,7), (5,2), (5,7), (6,1), (6,3), (7,0), (7,3)]$	-2304
$[(3,4), (3,5), (4,5), (4,7), (5,2), (5,7), (6,1), (6,4), (7,0), (7,3)]$	-2304
$[(3,4), (3,5), (4,5), (4,7), (5,2), (5,7), (6,1), (6,5), (7,0), (7,3)]$	-2304
$[(3,4), (3,5), (4,5), (4,7), (5,2), (5,7), (6,1), (6,7), (7,0), (7,3)]$	-2304

The Leibniz graph factorization  $\diamond_{\gamma_3}(P, \llbracket P, P \rrbracket)$ :

$[(3,4), (3,6), (4,5), (4,6), (5,1), (5,2), (5,3), (6,0), (6,5)]$	-2304
$[(3,5), (3,6), (4,3), (4,6), (5,4), (5,6), (6,0), (6,1), (6,2)]$	768
$[(3,4), (3,6), (4,5), (4,6), (5,2), (5,3), (6,0), (6,1), (6,5)]$	2304
$[(3,4), (3,5), (4,5), (4,6), (5,1), (5,2), (6,0), (6,3), (6,5)]$	-2304
$[(3,4), (3,5), (4,2), (4,6), (5,1), (5,4), (6,0), (6,3), (6,5)]$	2304
$[(3,4), (3,5), (3,6), (4,5), (4,6), (5,1), (5,2), (6,0), (6,5)]$	2304
$[(3,4), (3,5), (3,6), (4,2), (4,5), (5,1), (5,6), (6,0), (6,4)]$	-2304
$[(3,5), (3,6), (4,2), (4,3), (5,1), (5,4), (6,0), (6,4), (6,5)]$	-2304
$[(3,4), (3,6), (4,5), (4,6), (5,1), (5,3), (6,0), (6,2), (6,5)]$	-2304
$[(3,4), (3,5), (4,2), (4,5), (5,1), (5,6), (6,0), (6,3), (6,4)]$	2304
$[(3,4), (3,5), (3,6), (4,2), (4,6), (5,1), (5,4), (6,0), (6,5)]$	-2304
$[(3,4), (3,6), (4,2), (4,5), (5,1), (5,3), (6,0), (6,4), (6,5)]$	-2304
$[(3,4), (3,5), (4,5), (4,6), (5,1), (5,6), (6,0), (6,2), (6,3)]$	-2304
$[(3,4), (3,6), (4,5), (4,6), (5,1), (5,3), (5,6), (6,0), (6,2)]$	-2304
$[(3,4), (3,6), (4,2), (4,5), (5,1), (5,3), (5,6), (6,0), (6,4)]$	-2304
$[(3,4), (3,5), (3,6), (4,5), (4,6), (5,1), (5,6), (6,0), (6,2)]$	-2304
$[(3,5), (3,6), (4,2), (4,3), (5,1), (5,4), (5,6), (6,0), (6,4)]$	-2304
$[(3,4), (3,5), (4,5), (4,6), (5,1), (5,2), (5,6), (6,0), (6,3)]$	-2304
$[(3,4), (3,6), (4,2), (4,6), (5,1), (5,3), (5,4), (6,0), (6,5)]$	-2304
$[(3,4), (3,5), (4,2), (4,6), (5,1), (5,4), (5,6), (6,0), (6,3)]$	2304
$[(3,4), (3,5), (4,5), (4,6), (5,2), (5,6), (6,0), (6,1), (6,3)]$	2304
$[(3,4), (3,6), (4,5), (4,6), (5,2), (5,3), (5,6), (6,0), (6,1)]$	2304
$[(3,5), (3,6), (4,2), (4,3), (4,6), (5,1), (5,4), (6,0), (6,5)]$	2304
$[(3,4), (3,5), (3,6), (4,5), (4,6), (5,2), (5,6), (6,0), (6,1)]$	2304
$[(3,4), (3,6), (4,2), (4,5), (4,6), (5,1), (5,3), (6,0), (6,5)]$	2304
$[(3,5), (3,6), (4,2), (4,3), (4,5), (5,1), (5,6), (6,0), (6,4)]$	2304
$[(3,4), (3,5), (4,2), (4,5), (4,6), (5,1), (5,6), (6,0), (6,3)]$	2304

## D.2 The pentagon-wheel flow $Q_{\gamma_5}(P)$

The pentagon-wheel flow  $Q_{\gamma_5}$ :

$[(2,6), (2,7), (3,4), (3,6), (4,5), (4,7), (5,2), (5,3), (6,1), (6,5), (7,0), (7,5)]$	-7200
$[(2,5), (2,7), (3,6), (3,7), (4,3), (4,6), (5,4), (5,6), (6,1), (6,2), (7,0), (7,6)]$	7200
$[(2,3), (2,5), (3,5), (3,7), (4,6), (4,7), (5,4), (5,6), (6,1), (6,2), (7,0), (7,5)]$	-7200
$[(2,6), (2,7), (3,4), (3,6), (4,2), (4,6), (5,3), (5,7), (6,5), (6,7), (7,0), (7,1)]$	7200
$[(2,5), (2,7), (3,2), (3,5), (4,3), (4,6), (5,4), (5,7), (6,1), (6,5), (7,0), (7,6)]$	7200
$[(2,4), (2,5), (3,6), (3,7), (4,6), (4,7), (5,3), (5,6), (6,1), (6,2), (7,0), (7,6)]$	-7200
$[(2,5), (2,7), (3,5), (3,7), (4,3), (4,6), (5,4), (5,6), (6,1), (6,2), (7,0), (7,5)]$	7200
$[(2,5), (2,6), (3,2), (3,5), (4,3), (4,7), (5,4), (5,7), (6,1), (6,5), (7,0), (7,6)]$	-7200
$[(2,6), (2,7), (3,6), (3,7), (4,3), (4,6), (5,2), (5,4), (6,5), (6,7), (7,0), (7,1)]$	-7200
$[(2,5), (2,7), (3,5), (3,6), (4,3), (4,7), (5,4), (5,6), (6,1), (6,2), (7,0), (7,5)]$	7200
$[(2,5), (2,6), (3,5), (3,7), (4,2), (4,3), (5,4), (5,7), (6,1), (6,5), (7,0), (7,6)]$	-7200
$[(2,6), (2,7), (3,5), (3,7), (4,3), (4,7), (5,2), (5,7), (6,4), (6,7), (7,0), (7,1)]$	1440
$[(2,5), (2,6), (3,6), (3,7), (4,3), (4,6), (5,4), (5,6), (6,1), (6,7), (7,0), (7,2)]$	7200
$[(2,5), (2,7), (3,4), (3,5), (4,5), (4,6), (5,6), (5,7), (6,1), (6,2), (7,0), (7,3)]$	-7200
$[(2,4), (2,5), (3,5), (3,7), (4,3), (4,5), (5,6), (5,7), (6,1), (6,2), (7,0), (7,6)]$	7200
$[(2,4), (2,5), (3,6), (3,7), (4,6), (4,7), (5,3), (5,7), (6,1), (6,2), (7,0), (7,6)]$	-3600
$[(2,6), (2,7), (3,6), (3,7), (4,3), (4,7), (5,2), (5,4), (6,5), (6,7), (7,0), (7,1)]$	-3600
$[(2,5), (2,6), (3,2), (3,6), (4,3), (4,7), (5,4), (5,7), (6,1), (6,5), (7,0), (7,6)]$	-3600
$[(2,4), (2,7), (3,5), (3,6), (4,3), (4,5), (5,6), (5,7), (6,1), (6,2), (7,0), (7,6)]$	3600
$[(2,3), (2,6), (3,5), (3,6), (4,2), (4,7), (5,4), (5,6), (6,1), (6,7), (7,0), (7,5)]$	-3600
$[(2,5), (2,6), (3,2), (3,5), (4,3), (4,7), (5,6), (5,7), (6,4), (6,7), (7,0), (7,1)]$	3600
$[(2,6), (2,7), (3,4), (3,6), (4,5), (5,2), (5,3), (6,1), (6,4), (7,0), (7,6)]$	3600
$[(2,3), (2,6), (3,5), (3,7), (4,2), (4,7), (5,4), (5,6), (6,1), (6,4), (7,0), (7,5)]$	-3600
$[(2,3), (2,4), (3,5), (3,6), (4,5), (4,7), (5,6), (5,7), (6,1), (6,2), (7,0), (7,6)]$	3600
$[(2,6), (2,7), (3,4), (3,6), (4,5), (4,6), (5,2), (5,3), (6,1), (6,7), (7,0), (7,4)]$	3600
$[(2,3), (2,6), (3,4), (3,7), (4,5), (4,7), (5,2), (5,6), (6,1), (6,4), (7,0), (7,5)]$	3600

[(2,4),(2,5),(3,5),(3,7),(4,3),(4,6),(5,6),(5,7),(6,1),(6,2),(7,0),(7,2)]	3600
[(2,5),(2,6),(3,5),(3,7),(4,2),(4,3),(5,6),(5,7),(6,4),(6,7),(7,0),(7,1)]	3600
[(2,4),(2,7),(3,5),(3,6),(4,5),(4,6),(5,6),(5,7),(6,1),(6,2),(7,0),(7,3)]	3600
[(2,5),(2,6),(3,5),(3,6),(4,3),(4,7),(5,4),(5,6),(6,1),(6,7),(7,0),(7,2)]	3600
[(2,4),(2,5),(3,6),(3,7),(4,5),(4,7),(5,3),(5,6),(6,1),(6,4),(7,0),(7,2)]	-3600
[(2,4),(2,5),(3,6),(3,7),(4,3),(4,5),(5,6),(5,7),(6,1),(6,4),(7,0),(7,2)]	3600
[(2,5),(2,6),(3,2),(3,5),(4,3),(4,6),(5,4),(5,7),(6,1),(6,7),(7,0),(7,2)]	3600
[(2,4),(2,5),(3,2),(3,6),(4,3),(4,5),(5,6),(5,7),(6,1),(6,7),(7,0),(7,4)]	3600
[(2,6),(2,7),(3,5),(3,6),(4,3),(4,7),(5,2),(5,4),(6,1),(6,5),(7,0),(7,6)]	-3600
[(2,6),(2,7),(3,4),(3,6),(4,5),(4,7),(5,2),(5,3),(6,1),(6,4),(7,0),(7,5)]	-3600
[(2,4),(2,7),(3,6),(3,7),(4,5),(4,6),(5,3),(5,6),(6,1),(6,2),(7,0),(7,5)]	3600
[(2,5),(2,7),(3,5),(3,6),(4,3),(4,7),(5,4),(5,6),(6,1),(6,2),(7,0),(7,6)]	3600
[(2,3),(2,5),(3,5),(3,6),(4,6),(4,7),(5,4),(5,7),(6,1),(6,2),(7,0),(7,2)]	-3600
[(2,6),(2,7),(3,5),(3,6),(4,3),(4,7),(5,2),(5,4),(6,1),(6,4),(7,0),(7,5)]	-3600
[(2,5),(2,7),(3,2),(3,5),(4,3),(4,6),(5,4),(5,7),(6,1),(6,2),(7,0),(7,6)]	3600
[(2,5),(2,6),(3,5),(3,7),(4,2),(4,7),(5,6),(5,7),(6,3),(6,4),(7,0),(7,1)]	3600
[(2,5),(2,6),(3,5),(3,7),(4,2),(4,7),(5,4),(5,6),(6,3),(6,7),(7,0),(7,1)]	3600
[(2,4),(2,7),(3,6),(3,7),(4,5),(4,6),(5,3),(5,7),(6,1),(6,2),(7,0),(7,6)]	3600
[(2,6),(2,7),(3,4),(3,6),(4,2),(4,7),(5,3),(5,7),(6,5),(6,7),(7,0),(7,1)]	3600
[(2,6),(2,7),(3,2),(3,5),(4,3),(4,7),(5,4),(5,7),(6,5),(6,7),(7,0),(7,1)]	3600
[(2,5),(2,7),(3,6),(3,7),(4,3),(4,6),(5,4),(5,7),(6,1),(6,2),(7,0),(7,6)]	3600
[(2,4),(2,6),(3,5),(3,6),(4,3),(4,5),(5,6),(5,7),(6,1),(6,7),(7,0),(7,2)]	3600
[(2,4),(2,5),(3,5),(3,6),(4,3),(4,6),(5,6),(5,7),(6,1),(6,7),(7,0),(7,2)]	3600
[(2,6),(2,7),(3,5),(3,6),(4,3),(4,7),(5,2),(5,4),(6,1),(6,4),(7,0),(7,6)]	3600
[(2,4),(2,6),(3,2),(3,6),(4,5),(4,7),(5,3),(5,6),(6,1),(6,7),(7,0),(7,5)]	3600
[(2,4),(2,5),(3,6),(3,7),(4,3),(4,5),(5,6),(5,7),(6,1),(6,2),(7,0),(7,6)]	3600
[(2,4),(2,5),(3,5),(3,7),(4,3),(4,6),(5,6),(5,7),(6,1),(6,3),(7,0),(7,2)]	-3600
[(2,4),(2,5),(3,5),(3,7),(4,3),(4,6),(5,6),(5,7),(6,1),(6,2),(7,0),(7,6)]	3600
[(2,6),(2,7),(3,5),(3,6),(4,3),(4,6),(5,2),(5,4),(6,1),(6,7),(7,0),(7,4)]	3600
[(2,3),(2,6),(3,4),(3,6),(4,5),(4,7),(5,2),(5,7),(6,1),(6,5),(7,0),(7,6)]	3600
[(2,3),(2,4),(3,5),(3,6),(4,5),(4,7),(5,6),(5,7),(6,1),(6,4),(7,0),(7,2)]	-3600
[(2,5),(2,7),(3,2),(3,6),(4,3),(4,5),(5,6),(5,7),(6,4),(6,7),(7,0),(7,1)]	3600
[(2,6),(2,7),(3,4),(3,5),(4,2),(4,5),(5,6),(5,7),(6,3),(6,7),(7,0),(7,1)]	3600
[(2,4),(2,5),(3,5),(3,6),(4,6),(4,7),(5,6),(5,7),(6,1),(6,2),(7,0),(7,3)]	-3600
[(2,5),(2,6),(3,5),(3,7),(4,3),(4,6),(5,4),(5,6),(6,1),(6,7),(7,0),(7,2)]	3600
[(2,5),(2,6),(3,6),(3,7),(4,3),(4,6),(5,4),(5,7),(6,1),(6,5),(7,0),(7,2)]	-3600
[(2,4),(2,5),(3,4),(3,6),(4,5),(4,7),(5,6),(5,7),(6,1),(6,2),(7,0),(7,3)]	-3600
[(2,5),(2,6),(3,5),(3,6),(4,2),(4,3),(5,4),(5,7),(6,1),(6,2),(7,0),(7,3)]	-3600
[(2,5),(2,6),(3,2),(3,5),(4,3),(4,6),(5,4),(5,7),(6,1),(6,7),(7,0),(7,4)]	-3600
[(2,3),(2,5),(3,4),(3,6),(4,2),(4,5),(5,6),(5,7),(6,1),(6,7),(7,0),(7,4)]	3600
[(2,6),(2,7),(3,4),(3,6),(4,5),(4,7),(5,2),(5,3),(6,1),(6,5),(7,0),(7,6)]	-3600
[(2,5),(2,7),(3,6),(3,7),(4,3),(4,6),(5,4),(5,6),(6,1),(6,2),(7,0),(7,5)]	3600
[(2,3),(2,5),(3,5),(3,7),(4,6),(4,7),(5,4),(5,6),(6,1),(6,2),(7,0),(7,6)]	-3600
[(2,5),(2,6),(3,5),(3,7),(4,3),(4,6),(5,4),(5,7),(6,1),(6,3),(7,0),(7,2)]	-3600
[(2,4),(2,5),(3,6),(3,7),(4,6),(4,7),(5,3),(5,6),(6,1),(6,2),(7,0),(7,5)]	-3600
[(2,5),(2,7),(3,5),(3,7),(4,3),(4,6),(5,4),(5,6),(6,1),(6,2),(7,0),(7,6)]	3600
[(2,4),(2,5),(3,6),(3,7),(4,6),(4,7),(5,3),(5,6),(6,1),(6,2),(7,0),(7,2)]	-3600
[(2,5),(2,6),(3,5),(3,6),(4,3),(4,7),(5,4),(5,7),(6,1),(6,4),(7,0),(7,2)]	-3600
[(2,5),(2,6),(3,5),(3,7),(4,2),(4,3),(5,4),(5,7),(6,1),(6,3),(7,0),(7,6)]	-3600
[(2,5),(2,6),(3,2),(3,5),(4,3),(4,7),(5,4),(5,7),(6,1),(6,4),(7,0),(7,6)]	-3600
[(2,6),(2,7),(3,5),(3,7),(4,2),(4,5),(5,6),(5,7),(6,3),(6,4),(7,0),(7,1)]	-3600
[(2,6),(2,7),(3,5),(3,7),(4,2),(4,6),(5,4),(5,7),(6,3),(6,5),(7,0),(7,1)]	-3600
[(2,5),(2,7),(3,5),(3,7),(4,2),(4,3),(4,6),(5,4),(5,7),(6,3),(6,5),(7,0),(7,1)]	3600
[(2,4),(2,6),(3,6),(3,7),(4,5),(4,6),(5,3),(5,7),(6,1),(6,5),(7,0),(7,2)]	-3600
[(2,3),(2,5),(3,5),(3,6),(4,6),(4,7),(5,4),(5,6),(6,1),(6,7),(7,0),(7,2)]	3600
[(2,5),(2,7),(3,4),(3,6),(4,5),(4,6),(5,6),(5,7),(6,1),(6,2),(7,0),(7,3)]	-3600
[(2,3),(2,5),(3,4),(3,5),(4,6),(4,7),(5,4),(5,6),(6,1),(6,7),(7,0),(7,2)]	-3600
[(2,4),(2,6),(3,5),(3,7),(4,3),(4,5),(5,6),(5,7),(6,1),(6,4),(7,0),(7,2)]	-3600
[(2,5),(2,6),(3,5),(3,7),(4,3),(4,7),(5,4),(5,6),(6,1),(6,4),(7,0),(7,2)]	-3600
[(2,5),(2,6),(3,5),(3,7),(4,3),(4,7),(5,4),(5,6),(6,1),(6,3),(7,0),(7,2)]	-3600
[(2,4),(2,5),(3,5),(3,6),(4,3),(4,6),(5,4),(5,7),(6,1),(6,7),(7,0),(7,2)]	-3600
[(2,4),(2,5),(3,5),(3,6),(4,3),(4,7),(5,4),(5,6),(6,1),(6,7),(7,0),(7,2)]	3600
[(2,5),(2,6),(3,4),(3,7),(4,2),(4,5),(5,3),(5,6),(6,1),(6,7),(7,0),(7,4)]	-3600
[(2,5),(2,7),(3,4),(3,6),(4,2),(4,5),(5,3),(5,6),(6,1),(6,7),(7,0),(7,4)]	3600
[(2,5),(2,6),(3,5),(3,7),(4,2),(4,3),(5,4),(5,7),(6,1),(6,4),(7,0),(7,6)]	-3600
[(2,6),(2,7),(3,5),(3,7),(4,2),(4,5),(5,6),(5,7),(6,3),(6,4),(7,0),(7,1)]	-3600
[(2,6),(2,7),(3,5),(3,7),(4,2),(4,6),(5,4),(5,7),(6,3),(6,5),(7,0),(7,1)]	-3600
[(2,5),(2,7),(3,5),(3,7),(4,2),(4,3),(4,6),(5,4),(5,7),(6,1),(6,5),(7,0),(7,2)]	-7200
[(2,5),(2,6),(3,6),(3,7),(4,3),(4,7),(5,4),(5,7),(6,1),(6,7),(7,0),(7,2)]	7200
[(2,3),(2,5),(3,5),(3,6),(4,6),(4,7),(5,4),(5,7),(6,1),(6,5),(7,0),(7,2)]	-7200
[(2,5),(2,6),(3,2),(3,5),(4,3),(4,7),(5,4),(5,6),(6,1),(6,7),(7,0),(7,5)]	-7200
[(2,4),(2,5),(3,6),(3,7),(4,6),(4,7),(5,3),(5,7),(6,1),(6,7),(7,0),(7,2)]	7200
[(2,5),(2,6),(3,5),(3,6),(4,3),(4,7),(5,4),(5,7),(6,1),(6,5),(7,0),(7,2)]	-7200
[(2,5),(2,7),(3,2),(3,5),(4,3),(4,6),(5,4),(5,6),(6,1),(6,7),(7,0),(7,5)]	7200
[(2,5),(2,6),(3,5),(3,7),(4,3),(4,6),(5,4),(5,7),(6,1),(6,5),(7,0),(7,2)]	-7200

[(2,5),(2,6),(3,5),(3,7),(4,2),(4,3),(5,4),(5,6),(6,1),(6,7),(7,0),(7,5)]	-7200
[(2,5),(2,7),(3,6),(3,7),(4,3),(4,7),(5,4),(5,7),(6,1),(6,2),(7,0),(7,6)]	7200
[(2,4),(2,5),(3,5),(3,6),(4,5),(4,7),(5,6),(5,7),(6,1),(6,2),(7,0),(7,3)]	-7200
[(2,4),(2,5),(3,5),(3,6),(4,3),(4,5),(5,6),(5,7),(6,1),(6,7),(7,0),(7,2)]	7200
[(2,4),(2,5),(3,6),(3,7),(4,6),(4,7),(5,3),(5,6),(6,1),(6,7),(7,0),(7,2)]	3600
[(2,3),(2,6),(3,4),(3,7),(4,5),(4,7),(5,2),(5,6),(6,1),(6,7),(7,0),(7,5)]	3600
[(2,4),(2,6),(3,5),(3,7),(4,3),(4,5),(5,6),(5,7),(6,1),(6,7),(7,0),(7,2)]	3600
[(2,4),(2,6),(3,5),(3,7),(4,3),(4,7),(5,2),(5,7),(6,1),(6,5),(7,0),(7,6)]	3600
[(2,6),(2,7),(3,4),(3,7),(4,5),(4,6),(5,2),(5,3),(6,1),(6,7),(7,0),(7,4)]	3600
[(2,4),(2,6),(3,2),(3,7),(4,5),(4,7),(5,3),(5,6),(6,1),(6,4),(7,0),(7,5)]	3600
[(2,3),(2,4),(3,5),(3,6),(4,5),(4,7),(5,6),(5,7),(6,1),(6,7),(7,0),(7,2)]	-3600
[(2,6),(2,7),(3,4),(3,7),(4,5),(4,7),(5,2),(5,3),(6,1),(6,4),(7,0),(7,6)]	3600
[(2,5),(2,6),(3,2),(3,7),(4,3),(4,7),(5,4),(5,6),(6,1),(6,4),(7,0),(7,5)]	-3600
[(2,4),(2,5),(3,5),(3,6),(4,3),(4,7),(5,6),(5,7),(6,1),(6,2),(7,0),(7,2)]	3600
[(2,5),(2,7),(3,4),(3,6),(4,5),(4,7),(5,6),(5,7),(6,1),(6,2),(7,0),(7,3)]	3600
[(2,5),(2,7),(3,5),(3,7),(4,3),(4,6),(5,4),(5,7),(6,1),(6,2),(7,0),(7,6)]	-3600
[(2,4),(2,5),(3,6),(3,7),(4,5),(4,6),(5,3),(5,7),(6,1),(6,2),(7,0),(7,4)]	-3600
[(2,4),(2,5),(3,6),(3,7),(4,3),(4,5),(5,6),(5,7),(6,1),(6,2),(7,0),(7,4)]	-3600
[(2,5),(2,7),(3,2),(3,5),(4,3),(4,7),(5,4),(5,6),(6,1),(6,2),(7,0),(7,6)]	-3600
[(2,3),(2,5),(3,4),(3,5),(4,2),(4,7),(5,6),(5,7),(6,1),(6,3),(7,0),(7,6)]	3600
[(2,6),(2,7),(3,4),(3,6),(4,5),(4,7),(5,2),(5,3),(6,1),(6,7),(7,0),(7,5)]	-3600
[(2,6),(2,7),(3,4),(3,7),(4,5),(4,6),(5,2),(5,3),(6,1),(6,5),(7,0),(7,4)]	-3600
[(2,4),(2,6),(3,6),(3,7),(4,5),(4,7),(5,3),(5,7),(6,1),(6,5),(7,0),(7,2)]	3600
[(2,5),(2,6),(3,5),(3,7),(4,3),(4,6),(5,4),(5,7),(6,1),(6,7),(7,0),(7,2)]	-3600
[(2,3),(2,5),(3,5),(3,7),(4,6),(4,7),(5,4),(5,6),(6,1),(6,2),(7,0),(7,2)]	-3600
[(2,6),(2,7),(3,5),(3,7),(4,3),(4,6),(5,2),(5,4),(6,1),(6,5),(7,0),(7,4)]	-3600
[(2,5),(2,6),(3,2),(3,5),(4,3),(4,7),(5,4),(5,6),(6,1),(6,7),(7,0),(7,2)]	-3600
[(2,5),(2,6),(3,6),(3,7),(4,3),(4,6),(5,4),(5,7),(6,1),(6,7),(7,0),(7,2)]	3600
[(2,4),(2,6),(3,6),(3,7),(4,5),(4,6),(5,3),(5,7),(6,1),(6,7),(7,0),(7,2)]	3600
[(2,4),(2,7),(3,5),(3,7),(4,3),(4,5),(5,6),(5,7),(6,1),(6,2),(7,0),(7,6)]	3600
[(2,4),(2,5),(3,5),(3,7),(4,3),(4,7),(5,6),(5,7),(6,1),(6,2),(7,0),(7,6)]	3600
[(2,6),(2,7),(3,5),(3,7),(4,3),(4,6),(5,2),(5,4),(6,1),(6,7),(7,0),(7,4)]	3600
[(2,5),(2,6),(3,4),(3,7),(4,2),(4,7),(5,3),(5,7),(6,1),(6,5),(7,0),(7,6)]	-3600
[(2,4),(2,5),(3,6),(3,7),(4,3),(4,5),(5,6),(5,7),(6,1),(6,7),(7,0),(7,2)]	-3600
[(2,4),(2,5),(3,5),(3,6),(4,3),(4,7),(5,6),(5,7),(6,1),(6,2),(7,0),(7,3)]	-3600
[(2,4),(2,5),(3,5),(3,6),(4,3),(4,7),(5,6),(5,7),(6,1),(6,7),(7,0),(7,2)]	3600
[(2,6),(2,7),(3,5),(3,7),(4,3),(4,7),(5,2),(5,4),(6,1),(6,4),(7,0),(7,6)]	3600
[(2,5),(2,6),(3,2),(3,7),(4,3),(4,7),(5,4),(5,6),(6,1),(6,7),(7,0),(7,5)]	-3600
[(2,3),(2,4),(3,5),(3,7),(4,5),(4,6),(5,6),(5,7),(6,1),(6,2),(7,0),(7,4)]	-3600
[(2,5),(2,7),(3,4),(3,5),(4,6),(4,7),(5,6),(5,7),(6,1),(6,2),(7,0),(7,3)]	3600
[(2,5),(2,7),(3,5),(3,6),(4,3),(4,7),(5,4),(5,7),(6,1),(6,2),(7,0),(7,6)]	-3600
[(2,5),(2,7),(3,6),(3,7),(4,3),(4,7),(5,4),(5,6),(6,1),(6,2),(7,0),(7,5)]	-3600
[(2,4),(2,7),(3,4),(3,5),(4,5),(4,6),(5,6),(5,7),(6,1),(6,2),(7,0),(7,3)]	-3600
[(2,5),(2,7),(3,5),(3,7),(4,2),(4,3),(5,4),(5,6),(6,1),(6,3),(7,0),(7,6)]	3600
[(2,5),(2,7),(3,2),(3,5),(4,3),(4,7),(5,4),(5,6),(6,1),(6,4),(7,0),(7,6)]	3600
[(2,4),(2,5),(3,2),(3,5),(4,3),(4,7),(5,6),(5,7),(6,1),(6,3),(7,0),(7,6)]	3600
[(2,6),(2,7),(3,5),(3,6),(4,3),(4,7),(5,2),(5,4),(6,1),(6,7),(7,0),(7,5)]	-3600
[(2,5),(2,6),(3,6),(3,7),(4,3),(4,7),(5,4),(5,7),(6,1),(6,5),(7,0),(7,2)]	3600
[(2,3),(2,5),(3,5),(3,6),(4,6),(4,7),(5,4),(5,7),(6,1),(6,7),(7,0),(7,2)]	-3600
[(2,5),(2,7),(3,5),(3,6),(4,3),(4,7),(5,4),(5,6),(6,1),(6,2),(7,0),(7,3)]	3600
[(2,4),(2,5),(3,6),(3,7),(4,6),(4,7),(5,3),(5,7),(6,1),(6,5),(7,0),(7,2)]	3600
[(2,5),(2,7),(3,5),(3,6),(4,3),(4,7),(5,4),(5,6),(6,1),(6,2),(7,0),(7,3)]	3600
[(2,4),(2,5),(3,6),(3,7),(4,6),(4,7),(5,3),(5,7),(6,1),(6,5),(7,0),(7,2)]	3600
[(2,5),(2,7),(3,5),(3,6),(4,3),(4,7),(5,4),(5,6),(6,1),(6,2),(7,0),(7,5)]	-3600
[(2,5),(2,7),(3,6),(3,7),(4,3),(4,7),(5,4),(5,6),(6,1),(6,2),(7,0),(7,5)]	-3600
[(2,4),(2,7),(3,4),(3,5),(4,5),(4,6),(5,6),(5,7),(6,1),(6,2),(7,0),(7,5)]	-3600
[(2,3),(2,5),(3,5),(3,7),(4,6),(4,7),(5,4),(5,7),(6,1),(6,2),(7,0),(7,6)]	3600
[(2,4),(2,7),(3,5),(3,6),(4,5),(4,7),(5,6),(5,7),(6,1),(6,2),(7,0),(7,3)]	-3600
[(2,3),(2,5),(3,4),(3,5),(4,6),(4,7),(5,4),(5,7),(6,1),(6,2),(7,0),(7,6)]	-3600
[(2,4),(2,7),(3,5),(3,6),(4,5),(4,7),(5,6),(5,7),(6,1),(6,2),(7,0),(7,3)]	-3600
[(2,4),(2,7),(3,5),(3,6),(4,3),(4,5),(5,6),(5,7),(6,1),(6,2),(7,0),(7,4)]	-3600
[(2,5),(2,7),(3,5),(3,6),(4,3),(4,6),(5,4),(5,7),(6,1),(6,2),(7,0),(7,4)]	3600
[(2,5),(2,7),(3,5),(3,6),(4,2),(4,3),(5,4),(5,6),(6,1),(6,7),(7,0),(7,3)]	3600
[(2,5),(2,7),(3,2),(3,5),(4,3),(4,6),(5,4),(5,6),(6,1),(6,7),(7,0),(7,4)]	3600
[(2,4),(2,7),(3,6),(3,7),(4,5),(4,7),(5,3),(5,6),(6,1),(6,2),(7,0),(7,5)]	-3600
[(2,3),(2,5),(3,5),(3,7),(4,6),(4,7),(5,4),(5,7),(6,1),(6,2),(7,0),(7,6)]	3600
[(2,4),(2,7),(3,5),(3,6),(4,5),(4,7),(5,6),(5,7),(6,1),(6,2),(7,0),(7,3)]	-3600
[(2,3),(2,5),(3,4),(3,5),(4,6),(4,7),(5,4),(5,7),(6,1),(6,2),(7,0),(7,6)]	-3600
[(2,4),(2,7),(3,5),(3,6),(4,3),(4,5),(5,6),(5,7),(6,1),(6,2),(7,0),(7,4)]	-3600
[(2,5),(2,7),(3,5),(3,6),(4,3),(4,6),(5,4),(5,7),(6,1),(6,2),(7,0),(7,4)]	3600
[(2,5),(2,7),(3,5),(3,6),(4,3),(4,6),(5,4),(5,7),(6,1),(6,2),(7,0),(7,3)]	3600
[(2,4),(2,5),(3,5),(3,7),(4,3),(4,7),(5,4),(5,6),(6,1),(6,2),(7,0),(7,6)]	3600
[(2,4),(2,5),(3,5),(3,7),(4,3),(4,6),(5,4),(5,7),(6,1),(6,2),(7,0),(7,6)]	-3600
[(2,5),(2,7),(3,4),(3,6),(4,2),(4,5),(5,3),(5,7),(6,1),(6,4),(7,0),(7,6)]	3600
[(2,5),(2,6),(3,2),(3,5),(4,3),(4,7),(5,4),(5,7),(6,1),(6,3),(7,0),(7,6)]	-3600
[(2,5),(2,6),(3,5),(3,7),(4,2),(4,3),(5,4),(5,6),(6,1),(6,7),(7,0),(7,4)]	-3600

The Poisson differential  $\llbracket P, Q_{\gamma_5} \rrbracket$  and its Leibniz graph factorization  $\diamond_{\gamma_5}(P, \llbracket P, P \rrbracket)$  are available from <https://rburing.nl/gcaops>, namely as `[P,Q_gamma_5].txt` and `[P,Q_gamma_5]_leibniz.txt`

# Appendix E

## Graph cocycles

Encodings of graph cocycles  $\gamma_3, \gamma_5, \gamma_7, [\gamma_3, \gamma_5]$  in the Kontsevich unoriented graph complex with the vertex-expanding differential.

### E.1 The tetrahedron $\gamma_3$

```
1*UndirectedGraph(4, [(0,1), (0,2), (0,3), (1,2), (1,3), (2,3)])
```

### E.2 The pentagon-wheel cocycle $\gamma_5$

```
1*UndirectedGraph(6, [(0,3), (0,4), (0,5), (1,2), (1,4), (1,5), (2,3), (2,5), (3,5), (4,5)]) + \
(5/2)*UndirectedGraph(6, [(0,1), (0,3), (0,5), (1,2), (1,4), (2,4), (2,5), (3,4), (3,5), (4,5)])
```

### E.3 The heptagon-wheel cocycle $\gamma_7$

```
[(1,2), (1,4), (1,8), (2,3), (2,7), (3,5), (3,7), (4,6), (4,8), (5,7), (5,8), (6,7), (6,8), (7,8)] -21/8
[(1,3), (1,4), (1,8), (2,3), (2,5), (2,8), (3,7), (4,6), (4,8), (5,6), (5,7), (6,7), (6,8), (7,8)] -77/4
[(1,2), (1,3), (1,5), (2,4), (2,7), (3,5), (3,6), (4,6), (4,8), (5,7), (5,8), (6,7), (6,8), (7,8)] -35/8
[(1,2), (1,3), (1,8), (2,4), (2,6), (3,7), (3,8), (4,6), (4,7), (5,6), (5,7), (5,8), (6,8), (7,8)] 49/8
[(1,4), (1,7), (1,8), (2,3), (2,5), (2,6), (3,5), (3,7), (4,6), (4,8), (5,6), (5,8), (6,7), (7,8)] 77/8
[(1,2), (1,3), (1,8), (2,6), (2,7), (3,5), (3,8), (4,5), (4,6), (4,7), (5,6), (5,7), (6,8), (7,8)] -105/8
[(1,2), (1,4), (1,8), (2,3), (2,7), (3,6), (3,8), (4,6), (4,8), (5,6), (5,7), (5,8), (6,7), (7,8)] 7/8
[(1,2), (1,4), (1,5), (2,3), (2,7), (3,5), (3,6), (4,6), (4,8), (5,7), (5,8), (6,7), (6,8), (7,8)] 35/8
[(1,2), (1,3), (1,4), (2,7), (2,8), (3,6), (3,8), (4,6), (4,7), (5,6), (5,7), (5,8), (6,8), (7,8)] -49/8
[(1,2), (1,3), (1,8), (2,5), (2,7), (3,4), (3,6), (4,7), (4,8), (5,6), (5,8), (6,7), (6,8), (7,8)] 35/4
[(1,2), (1,3), (1,4), (2,5), (2,6), (3,6), (3,8), (4,5), (4,7), (5,7), (5,8), (6,7), (6,8), (7,8)] -119/16
[(1,2), (1,3), (1,5), (2,4), (2,8), (3,6), (3,8), (4,7), (4,8), (5,6), (5,7), (6,7), (6,8), (7,8)] 49/8
[(1,2), (1,3), (1,4), (2,3), (3,7), (4,6), (4,8), (5,6), (5,7), (5,8), (6,7), (6,8), (7,8)] 77/4
[(1,2), (1,5), (1,7), (2,5), (2,6), (3,5), (3,6), (3,8), (4,5), (4,7), (4,8), (6,7), (6,8), (7,8)] -49/8
[(1,3), (1,5), (1,8), (2,4), (2,6), (2,8), (3,7), (3,8), (4,6), (4,7), (5,6), (5,7), (6,8), (7,8)] -49/4
[(1,3), (1,4), (1,8), (2,5), (2,6), (2,8), (3,6), (3,8), (4,7), (4,8), (5,6), (5,7), (6,7), (7,8)] -49/4
[(1,2), (1,4), (1,8), (2,3), (2,8), (3,5), (3,7), (4,6), (4,8), (5,6), (5,7), (6,7), (6,8), (7,8)] -7
[(1,2), (1,4), (1,8), (2,3), (2,8), (3,6), (3,8), (4,6), (4,7), (5,6), (5,7), (5,8), (6,7), (7,8)] -7
[(1,2), (1,5), (1,6), (2,5), (2,7), (3,5), (3,6), (3,8), (4,6), (4,7), (4,8), (5,8), (6,7), (7,8)] 49/8
[(1,2), (1,4), (1,8), (2,3), (2,8), (3,6), (3,7), (4,6), (4,7), (5,6), (5,7), (5,8), (6,8), (7,8)] 49/8
[(1,2), (1,3), (1,5), (2,6), (2,7), (3,5), (3,6), (4,5), (4,7), (4,8), (5,8), (6,7), (6,8), (7,8)] -7
[(1,6), (1,7), (1,8), (2,3), (2,5), (2,8), (3,4), (3,8), (4,6), (4,8), (5,7), (5,8), (6,8), (7,8)] 1
[(1,2), (1,3), (1,8), (2,4), (2,8), (3,5), (3,8), (4,6), (4,7), (5,7), (5,8), (6,7), (6,8), (7,8)] 7
[(1,2), (1,3), (1,8), (2,5), (2,6), (3,7), (3,8), (4,5), (4,6), (4,7), (5,6), (5,7), (6,8), (7,8)] -7
[(1,2), (1,4), (1,6), (2,3), (2,5), (3,6), (3,7), (4,5), (4,8), (5,7), (5,8), (6,7), (6,8), (7,8)] 77/8
[(1,3), (1,6), (1,7), (2,4), (2,5), (2,6), (3,5), (3,7), (4,5), (4,8), (5,8), (6,7), (6,8), (7,8)] -7
[(1,4), (1,5), (1,7), (2,3), (2,6), (2,8), (3,7), (3,8), (4,6), (4,8), (5,6), (5,7), (6,8), (7,8)] 49/4
[(1,2), (1,6), (1,8), (2,7), (2,8), (3,4), (3,6), (3,8), (4,6), (4,7), (5,6), (5,7), (5,8), (7,8)] -147/8
[(1,2), (1,5), (1,6), (2,7), (2,8), (3,5), (3,6), (3,8), (4,5), (4,6), (4,7), (5,7), (6,8), (7,8)] -21/8
[(1,2), (1,4), (1,8), (2,3), (2,7), (3,5), (3,6), (4,5), (4,6), (5,7), (5,8), (6,7), (6,8), (7,8)] -35/8
[(1,4), (1,5), (1,6), (2,3), (2,6), (2,8), (3,7), (3,8), (4,6), (4,8), (5,7), (5,8), (6,7), (7,8)] -49/4
[(1,2), (1,5), (1,8), (2,3), (2,8), (3,4), (3,7), (4,6), (4,8), (5,6), (5,7), (6,7), (6,8), (7,8)] 105/8
```



```

(120)*UndirectedGraph(9, [(0,1), (0,2), (0,6), (1,3), (1,5), (2,5), (2,8), (3,5), (3,7), (4,6), (4,7), (4,8), (5,8), (6,7), (6,8), (7,8)]) + \
(-120)*UndirectedGraph(9, [(0,3), (0,4), (0,8), (1,2), (1,4), (1,8), (2,3), (2,5), (3,6), (4,7), (5,6), (5,7), (5,8), (6,7), (6,8), (7,8)]) + \
(120)*UndirectedGraph(9, [(0,1), (0,2), (0,7), (1,2), (1,6), (2,5), (3,5), (3,6), (3,8), (4,5), (4,7), (4,8), (5,8), (6,7), (6,8), (7,8)]) + \
(120)*UndirectedGraph(9, [(0,1), (0,2), (0,5), (1,3), (1,6), (2,5), (2,8), (3,5), (3,7), (4,6), (4,7), (4,8), (5,8), (6,7), (6,8), (7,8)]) + \
(120)*UndirectedGraph(9, [(0,1), (0,2), (0,8), (1,2), (1,5), (2,7), (3,4), (3,6), (3,8), (4,5), (4,6), (5,7), (5,8), (6,7), (6,8), (7,8)]) + \
(120)*UndirectedGraph(9, [(0,1), (0,4), (0,8), (1,3), (1,8), (2,3), (2,4), (2,7), (3,5), (4,6), (5,6), (5,7), (5,8), (6,7), (6,8), (7,8)]) + \
(120)*UndirectedGraph(9, [(0,1), (0,2), (0,6), (1,2), (1,5), (2,7), (3,4), (3,5), (3,8), (4,6), (4,8), (5,7), (5,8), (6,7), (6,8), (7,8)]) + \
(-60)*UndirectedGraph(9, [(0,1), (0,2), (0,8), (1,2), (1,8), (2,6), (3,4), (3,5), (3,6), (4,5), (4,7), (5,7), (5,8), (6,7), (6,8), (7,8)]) + \
(60)*UndirectedGraph(9, [(0,3), (0,4), (0,6), (1,2), (1,5), (1,8), (2,5), (2,8), (3,6), (3,7), (4,5), (4,7), (5,8), (6,7), (6,8), (7,8)]) + \
(120)*UndirectedGraph(9, [(0,4), (0,5), (0,8), (1,2), (1,3), (1,6), (2,3), (2,7), (3,8), (4,5), (4,6), (4,7), (5,7), (5,8), (6,7), (6,8)]) + \
(120)*UndirectedGraph(9, [(0,5), (0,7), (0,8), (1,4), (1,6), (1,8), (2,3), (2,5), (2,7), (3,6), (3,7), (4,5), (4,6), (4,8), (5,7), (6,8)]) + \
(-120)*UndirectedGraph(9, [(0,4), (0,5), (0,7), (1,3), (1,7), (1,8), (2,3), (2,6), (2,8), (3,6), (4,5), (4,7), (4,8), (5,6), (5,7), (6,8)]) + \
(-120)*UndirectedGraph(9, [(0,2), (0,6), (0,8), (1,3), (1,7), (1,8), (2,3), (2,5), (3,8), (4,5), (4,6), (4,7), (4,8), (5,6), (5,7), (6,7)]) + \
(-120)*UndirectedGraph(9, [(0,5), (0,7), (0,8), (1,2), (1,3), (1,7), (2,3), (2,8), (3,6), (4,5), (4,6), (4,7), (4,8), (5,6), (5,8), (6,7)]) + \
(-60)*UndirectedGraph(9, [(0,3), (0,4), (0,7), (1,2), (1,5), (1,8), (2,5), (2,8), (3,6), (3,8), (4,6), (4,7), (5,7), (5,8), (6,7), (6,8)]) + \
(-60)*UndirectedGraph(9, [(0,1), (0,2), (0,8), (1,2), (1,8), (2,5), (3,4), (3,6), (3,8), (4,5), (4,7), (5,6), (5,7), (6,7), (6,8), (7,8)]) + \
(60)*UndirectedGraph(9, [(0,1), (0,2), (0,7), (1,2), (1,7), (2,8), (3,4), (3,5), (3,8), (4,6), (4,7), (5,6), (5,8), (6,7), (6,8), (7,8)])

```





# Appendix F

## Reference documentation for the **gcaops** software

The following appendix has been generated from the documentation strings included in the source code of **gcaops**. In addition to an exhaustive listing of all available methods on all objects, some basic usage examples are included.

---

# Documentation of gcaops

*Release 1*

**Ricardo Buring**

**Jul 01, 2022**



## TABLE OF CONTENTS

<b>1</b>	<b>Algebra</b>	<b>579</b>
1.1	Superfunction algebra	579
1.2	Superfunction algebra operation	582
1.3	Polydifferential operator	584
1.4	Tensor product	587
1.5	Differential polynomial ring	588
1.6	Homogeneous differential polynomial equation solver	589
<b>2</b>	<b>Abstract base classes</b>	<b>591</b>
2.1	Graph basis	591
2.2	Graph vector	591
2.3	Graph complex	593
2.4	Graph vector (vector backend)	593
2.5	Graph vector (dictionary backend)	595
<b>3</b>	<b>Undirected graphs</b>	<b>599</b>
3.1	Undirected graph	599
3.2	Undirected graph basis	601
3.3	Undirected graph vector	603
3.4	Undirected graph operad	605
3.5	Undirected graph complex	605
<b>4</b>	<b>Directed graphs</b>	<b>609</b>
4.1	Directed graph	609
4.2	Directed graph basis	610
4.3	Directed graph vector	612
4.4	Directed graph operad	614
4.5	Directed graph complex	614
<b>5</b>	<b>Formality graphs</b>	<b>617</b>
5.1	Formality graph	617
5.2	Formality graph basis	622
5.3	Formality graph vector	625
5.4	Formality graph operad	628
5.5	Formality graph complex	629
5.6	Formality graph operator	631
<b>6</b>	<b>Graph cache</b>	<b>633</b>
6.1	Graph file view	633
6.2	Graph cache	635

**Python Module Index**

**637**

**Index**

**639**

## Documentation of gcaops, Release 1

---

Import the package:

```
sage: import gcaops
```

## 1.1 Superfunction algebra

Superfunction algebra

**class** gcaops.algebra.superfunction\_algebra.**Superfunction**(parent, monomial\_coefficients)

Bases: object

Superfunction on a coordinate chart of a  $Z_2$ -graded space.

A polynomial in the odd coordinates, with coefficients in the base ring (of even degree 0 functions).

**\_\_add\_\_**(other)

Return this superfunction added to other.

**\_\_eq\_\_**(other)

Return True if this superfunction equals other and False otherwise.

---

**Note:** This takes the difference and calls `is_zero()` on it. For comparison with zero it is faster to call `is_zero()` directly.

---

**\_\_getitem\_\_**(indices)

Return the coefficient of the monomial in the odd coordinates specified by indices.

**\_\_init\_\_**(parent, monomial\_coefficients)

Initialize this superfunction.

INPUT:

- parent - a *SuperfunctionAlgebra*
- monomial\_coefficients - a dictionary, taking a natural number  $m$  less than  $2^{\text{parent.ngens}()}$  to the coefficient of the monomial in the odd coordinates represented by  $m$

**\_\_mul\_\_**(other)

Return this superfunction multiplied by other.

**\_\_neg\_\_**()

Return the negative of this superfunction.

**\_\_pos\_\_**()

Return a copy of this superfunction.

**\_\_pow\_\_**(exponent)

Return this superfunction raised to the power exponent.

**\_\_radd\_\_**(*other*)

Return *other* added to this superfunction.

**\_\_repr\_\_**()

Return a string representation of this superfunction.

**\_\_rmul\_\_**(*other*)

Return *other* multiplied by this superfunction.

---

**Note:** This assumes that *other* commutes with this superfunction. It is justified because this function only gets called when *other* is even.

---

**\_\_rsub\_\_**(*other*)

Return *other* minus this superfunction.

**\_\_setitem\_\_**(*indices, new\_value*)

Set the coefficient of the monomial in the odd coordinates specified by *indices* to *new\_value*.

**\_\_sub\_\_**(*other*)

Return this superfunction minus *other*.

**\_\_truediv\_\_**(*other*)

Return this superfunction divided by *other*.

**bracket**(*other*)

Return the Schouten bracket (odd Poisson bracket) of this superfunction with *other*.

**copy**()

Return a copy of this superfunction.

**degree**()

Return the degree of this superfunction as a polynomial in the odd coordinates.

**degrees**()

Return an iterator over the degrees of the monomials (in the odd coordinates) of this superfunction.

**derivative**(\**args*)

Return the derivative of this superfunction with respect to *args*.

INPUT:

- *args* – an odd coordinate or an even coordinate, or a list of such

**diff**(\**args*)

Return the derivative of this superfunction with respect to *args*.

INPUT:

- *args* – an odd coordinate or an even coordinate, or a list of such

**homogeneous\_part**(*degree*)

Return the homogeneous part of this superfunction of total degree *degree* in the odd coordinates.

---

**Note:** Returns a *Superfunction* whose homogeneous component of degree *degree* is a *reference* to the respective component of this superfunction.

---

**indices**(*degree=None*)

Return an iterator over indices of this superfunction, i.e. a tuple of exponents for each monomial in the odd coordinates.



INPUT:

- `degree` (default: `None`) – if not `None`, yield only indices of degree `degree`

**is\_zero()**

Return `True` if this superfunction equals zero and `False` otherwise.

**map\_coefficients**(*f*, *new\_parent=None*)

Apply *f* to each of this superfunction's coefficients and return the resulting superfunction.

**parent()**

Return the parent *SuperfunctionAlgebra* that this superfunction belongs to.

**schouten\_bracket**(*other*)

Return the Schouten bracket (odd Poisson bracket) of this superfunction with *other*.

```
class gcaops.algebra.superfunction_algebra.SuperfunctionAlgebra(base_ring,
                                                                even_coordinates=None,
                                                                names='xi',          simplify=None,
                                                                is_zero='is_zero')
```

Bases: `object`

Supercommutative algebra of superfunctions on a coordinate chart of a  $Z_2$ -graded space.

Consisting of polynomials in the odd (degree 1) coordinates, with coefficients in the base ring (of even degree 0 functions). It is a free module over the base ring with a basis consisting of sorted monomials in the odd coordinates. The elements encode skew-symmetric multi-derivations of the base ring, or multi-vectors.

**\_\_call\_\_**(*arg*)

Return *arg* converted into an element of this superfunction algebra.

ASSUMPTIONS:

If *arg* is a *PolyDifferentialOperator*, it is assumed that its coefficients are skew-symmetric.

**\_\_init\_\_**(*base\_ring*, *even\_coordinates=None*, *names='xi'*, *simplify=None*, *is\_zero='is\_zero'*)

Initialize this superfunction algebra.

INPUT:

- `base_ring` – a commutative ring, considered as a ring of (even, degree 0) functions
- `even_coordinates` – (default: `None`) a list or tuple of elements of `base_ring`; if none is provided, then it is set to `base_ring.gens()`
- `names` – (default: `'xi'`) a list or tuple of strings or a comma separated string, consisting of names for the odd coordinates; or a single string consisting of a prefix that will be used to generate a list of numbered names
- `simplify` – (default: `None`) a string, containing the name of a method of an element of the base ring; that method should return a simplification of the element (will be used in each operation on elements that affects coefficients), or `None` (which amounts to no simplification).
- `is_zero` – (default: `'is_zero'`) a string, containing the name of a method of an element of the base ring; that method should return `True` when a simplified element of the base ring is equal to zero (will be used to decide equality of elements, to calculate the degree of elements, and to skip terms in some operations on elements)

**\_\_repr\_\_**()

Return a string representation of this superfunction algebra.

**base\_ring**()

Return the base ring of this superfunction algebra, consisting of (even, degree 0) functions.

**dimension**(*degree*)

Return the dimension of the graded component spanned by monomials of the given degree in the odd coordinates (as a module over the base ring).

INPUT:

- *degree* – a natural number

**even\_coordinate**(*i*)

Return the *i*-th even coordinate in the base ring of this superfunction algebra.

**even\_coordinates**()

Return the even coordinates in the base ring of this superfunction algebra.

**gen**(*i*)

Return the *i*-th odd coordinate of this superfunction algebra.

**gens**()

Return the tuple of odd coordinates of this superfunction algebra.

**graph\_operation**(*graph\_vector*)

Return the operation (on this superfunction algebra) defined by *graph\_vector*.

If the input is a graph cochain in a graph complex, then the operation that pre-symmetrizes the arguments is returned.

ASSUMPTION:

Assumes each graph in *graph\_vector* has the same number of vertices.

**ngens**()

Return the number of odd coordinates of this superfunction algebra.

**odd\_coordinate**(*i*)

Return the *i*-th odd coordinate of this superfunction algebra.

**odd\_coordinates**()

Return the tuple of odd coordinates of this superfunction algebra.

**one**()

Return the unit element of this superfunction algebra.

**schouten\_bracket**()

Return the Schouten bracket (odd Poisson bracket) on this superfunction algebra.

**tensor\_power**(*n*)

Return the *n*-th tensor power of this superfunction algebra.

**zero**()

Return the zero element of this superfunction algebra.

## 1.2 Superfunction algebra operation

Initialize self. See `help(type(self))` for accurate signature.

**class** `gcaops.algebra.superfunction_algebra_operation.SuperfunctionAlgebraDirectedGraphOperation`(*domain*, *codomain*, *graph\_vector*)

Bases: `gcaops.algebra.superfunction_algebra_operation.SuperfunctionAlgebraOperation`

A homogeneous *n*-ary multi-linear operation on a `SuperfunctionAlgebra`, defined by a `DirectedGraphVector`.

**degree()**

Return the degree of this operation.

**class** gcaops.algebra.superfunction\_algebra\_operation.**SuperfunctionAlgebraOperation**(*domain*,  
*codomain*)

Bases: abc.ABC

A homogeneous n-ary multi-linear operation acting on a *SuperfunctionAlgebra*.

**codomain()**

Return the codomain of this operation.

**degree()**

Return the degree of this operation.

**domain()**

Return the domain of this operation.

**class** gcaops.algebra.superfunction\_algebra\_operation.**SuperfunctionAlgebraSchoutenBracket**(*domain*,  
*codomain*)

Bases: *gcaops.algebra.superfunction\_algebra\_operation.SuperfunctionAlgebraSymmetricOperation*

Schouten bracket on a *SuperfunctionAlgebra*.

**degree()**

Return the degree of this operation.

**class** gcaops.algebra.superfunction\_algebra\_operation.**SuperfunctionAlgebraSymmetricBracketOperation**(\*args)

Bases: *gcaops.algebra.superfunction\_algebra\_operation.SuperfunctionAlgebraSymmetricOperation*

A homogeneous symmetric n-ary multi-linear operation acting on a *SuperfunctionAlgebra*, given by the Nijenhuis-Richardson bracket of two graded symmetric operations.

**degree()**

Return the degree of this operation.

**class** gcaops.algebra.superfunction\_algebra\_operation.**SuperfunctionAlgebraSymmetricDirectedGraphOperation**

Bases: *gcaops.algebra.superfunction\_algebra\_operation.SuperfunctionAlgebraDirectedGraphOperation*,  
*gcaops.algebra.superfunction\_algebra\_operation.SuperfunctionAlgebraSymmetricOperation*

A homogeneous symmetric n-ary multi-linear operation acting on a *SuperfunctionAlgebra*, defined by a *DirectedGraphVector*.

**class** gcaops.algebra.superfunction\_algebra\_operation.**SuperfunctionAlgebraSymmetricOperation**(*domain*,  
*codomain*)

Bases: *gcaops.algebra.superfunction\_algebra\_operation.SuperfunctionAlgebraOperation*

A homogeneous symmetric n-ary multi-linear operation acting on a *SuperfunctionAlgebra*.

**bracket**(*other*)

Return the Nijenhuis-Richardson bracket of this operation with the other operation.

**class** gcaops.algebra.superfunction\_algebra\_operation.**SuperfunctionAlgebraSymmetricUndirectedGraphOperation**

Bases: *gcaops.algebra.superfunction\_algebra\_operation.SuperfunctionAlgebraUndirectedGraphOperation*,  
*gcaops.algebra.superfunction\_algebra\_operation.SuperfunctionAlgebraSymmetricOperation*

A homogeneous n-ary multi-linear symmetric operation acting on a *SuperfunctionAlgebra*, defined by a *UndirectedGraphVector*.

**class** gcaops.algebra.superfunction\_algebra\_operation.**SuperfunctionAlgebraUndirectedGraphOperation**(*domain*, *codom*, *graph*)

Bases: *gcaops.algebra.superfunction\_algebra\_operation.SuperfunctionAlgebraOperation*

A homogeneous n-ary multi-linear operation acting on a *SuperfunctionAlgebra*, defined by a *UndirectedGraphVector*.

**degree**()  
Return the degree of this operation.

## 1.3 Polydifferential operator

Polydifferential operator

**class** gcaops.algebra.polydifferential\_operator.**PolyDifferentialOperator**(*parent*, *coefficients*)

Bases: object

Polydifferential operator on a coordinate chart.

A multi-linear polydifferential operator, with coefficients in the base ring (of functions).

**\_\_add\_\_**(*other*)  
Return this polydifferential operator added to *other*.

**\_\_eq\_\_**(*other*)  
Return True if this polydifferential operator equals *other* and False otherwise.

---

**Note:** This takes the difference and calls `is_zero()` on it. For comparison with zero it is faster to call `is_zero()` directly.

---

**\_\_getitem\_\_**(*multi\_indices*)  
Return the coefficient of the differential monomial specified by *multi\_indices*.

**\_\_init\_\_**(*parent*, *coefficients*)  
Initialize this polydifferential operator.

INPUT:

- *parent* - a *PolyDifferentialOperatorAlgebra* (which has an ordered basis of monomials in the odd coordinates)
- *coefficients* - a dictionary, mapping the arity *m* to a dictionary that maps *m*-tuples of multi-indices to elements in the base ring of *parent*

**\_\_mul\_\_**(*other*)  
Return this polydifferential operator multiplied by *other*.

---

**Note:** This is the pre-Lie product, a sum (with signs) of insertions of *other* into this polydifferential operator. For unary operators, it is simply composition.

---

**\_\_neg\_\_**()  
Return the negative of this polydifferential operator.

**\_\_pos\_\_**()  
Return a copy of this polydifferential operator.

**\_\_pow\_\_**(*exponent*)

Return this polydifferential operator raised to the power *exponent*.

**\_\_radd\_\_**(*other*)

Return *other* added to this polydifferential operator.

**\_\_repr\_\_**()

Return a string representation of this polydifferential operator.

**\_\_rmul\_\_**(*other*)

Return *other* multiplied by this polydifferential operator.

---

**Note:** This is only defined for elements of the base ring.

---

**\_\_rsub\_\_**(*other*)

Return *other* minus this polydifferential operator.

**\_\_setitem\_\_**(*multi\_indices*, *new\_value*)

Set the coefficient of the differential monomial specified by *multi\_indices* to *new\_value*.

**\_\_sub\_\_**(*other*)

Return this polydifferential operator minus *other*.

**\_\_truediv\_\_**(*other*)

Return this polydifferential operator divided by *other*.

**arity**()

Return the arity of this polydifferential operator.

ASSUMPTIONS:

Assumes this polydifferential operator is homogeneous.

**bracket**(*other*)

Return the Gerstenhaber bracket of this polydifferential operator with *other*.

**coefficient**(*variable*)

Return the coefficient of *variable* of this polydifferential operator.

**copy**()

Return a copy of this polydifferential operator.

**gerstenhaber\_bracket**(*other*)

Return the Gerstenhaber bracket of this polydifferential operator with *other*.

**hochschild\_differential**()

Return the Hochschild differential of this polydifferential operator, with respect to the multiplication operator of the parent.

**homogeneous\_part**(*arity*)

Return the homogeneous part of this polydifferential operator of arity *arity*.

---

**Note:** Returns a polydifferential operator whose homogeneous component of arity *arity* is a *reference* to the respective component of this polydifferential operator.

---

**insertion**(*position*, *other*)

Return the insertion of *other* into the *position*-th argument of this polydifferential operator.

**is\_zero**()

Return True if this polydifferential operator equals zero and False otherwise.

**map\_coefficients**(*f*, *new\_parent=None*)

Apply *f* to each of this polydifferential operator's coefficients and return the resulting polydifferential operator.

**multi\_indices**()

Return an iterator over the multi-indices of the terms in this polydifferential operator.

**parent**()

Return the parent *PolyDifferentialOperatorAlgebra* that this polydifferential operator belongs to.

**skew\_symmetrization**()

Return the polydifferential operator which is the skew-symmetrization of this polydifferential operator.

**subs**(\*args, \*\*kwargs)

Return this polydifferential operator with the subs method applied (with the given arguments) to each coefficient.

**symmetrization**()

Return the polydifferential operator which is the symmetrization of this polydifferential operator.

```
class gcaops.algebra.polydifferential_operator.PolyDifferentialOperatorAlgebra(base_ring,
                                                                              coordi-
                                                                              nates=None,
                                                                              names='ddx',
                                                                              sim-
                                                                              plify=None,
                                                                              is_zero='is_zero')
```

Bases: object

Noncommutative algebra of polydifferential operators on a coordinate chart.

**\_\_call\_\_**(\*args)

Return *arg* converted into an element of this polydifferential operator algebra.

**\_\_init\_\_**(*base\_ring*, *coordinates=None*, *names='ddx'*, *simplify=None*, *is\_zero='is\_zero'*)

Initialize this polydifferential operator algebra.

INPUT:

- *base\_ring* – a commutative ring, considered as a ring of functions
- *coordinates* – (default: *None*) a list or tuple of elements of *base\_ring*; if none is provided, then it is set to *base\_ring.gens()*
- *names* – (default: *'ddx'*) a list or tuple of strings or a comma separated string, consisting of names for the derivatives with respect to the coordinates; or a single string consisting of a prefix that will be used to generate a list of numbered names
- *simplify* – (default: *None*) a string, containing the name of a method of an element of the base ring; that method should return a simplification of the element (will be used in each operation on elements that affects coefficients), or *None* (which amounts to no simplification).
- *is\_zero* – (default: *'is\_zero'*) a string, containing the name of a method of an element of the base ring; that method should return *True* when a simplified element of the base ring is equal to zero (will be used to decide equality of elements, to calculate the arity of elements, and to skip terms in some operations on elements)

**\_\_repr\_\_**()

Return a string representation of this polydifferential operator algebra.

**base\_ring**()

Return the base ring of this polydifferential operator algebra, consisting of functions.

- coordinate**(*i*)  
Return the *i*-th even coordinate in the base ring of this polydifferential operator algebra.
- coordinates**()  
Return the coordinates in the base ring of this polydifferential operator algebra.
- derivative**(*i*)  
Return the *i*-th derivative of this polydifferential operator algebra.
- derivatives**()  
Return the tuple of derivatives of this polydifferential operator algebra.
- gen**(*i*)  
Return the *i*-th derivative of this polydifferential operator algebra.
- gens**()  
Return the tuple of derivatives of this polydifferential operator algebra.
- identity\_operator**()  
Return the (unary) identity operator of this polydifferential operator algebra.
- multiplication\_operator**()  
Return the (binary) multiplication operator of this polydifferential operator algebra.
- ngens**()  
Return the number of derivatives of this polydifferential operator algebra.
- tensor\_product**(\**args*)  
Return the tensor product of *args* as an element of this polydifferential operator algebra.
- zero**()  
Return the zero element of this polydifferential operator algebra.

## 1.4 Tensor product

Tensor product

- class** gcaops.algebra.tensor\_product.**TensorProduct**(*factors*)  
Bases: object  
Tensor product of vector spaces.
- factor**(*index*)  
Return the *index*-th factor of this tensor product.
- factors**()  
Return the list of factors of this tensor product.
- nfactors**()  
Return the number of factors of this tensor product.
- class** gcaops.algebra.tensor\_product.**TensorProductElement**(*parent*, *terms*)  
Bases: object  
Element of a tensor product of vector spaces.
- graded\_symmetrization**()  
Return the graded symmetrization of this tensor product element.  
ASSUMPTION:

Assumes each factor in each term of this tensor product element has a `degree` method and is homogeneous of that degree.

**parent()**

Return the parent `TensorProduct` that this tensor product element belongs to.

**terms()**

Return the list of terms of this tensor product element.

## 1.5 Differential polynomial ring

Differential polynomial ring

**class** `gcaops.algebra.differential_polynomial_ring.DifferentialPolynomial`(*parent, polynomial*)

Bases: object

Differential polynomial.

**derivative**(\*x)

Return the total derivative of this differential polynomial with respect to the base variables `x`.

**diff**(\*x)

Return the total derivative of this differential polynomial with respect to the base variables `x`.

**fibre\_degrees**()

Return the vector of degrees (with respect to each fibre variable) of this differential monomial.

**parent**()

Return the *DifferentialPolynomialRing* that this differential polynomial belongs to.

**partial\_derivative**(\*x)

Return the partial derivative of this differential polynomial with respect to the variables `x`.

**pdiff**(\*x)

Return the partial derivative of this differential polynomial with respect to the variables `x`.

**total\_derivative**(\*x)

Return the total derivative of this differential polynomial with respect to the base variables `x`.

**weights**()

Return the vector of weights of this differential monomial.

**class** `gcaops.algebra.differential_polynomial_ring.DifferentialPolynomialRing`(*base\_ring, fibre\_names, base\_names, max\_differential\_orders*)

Bases: object

Differential polynomial ring.

**\_\_init\_\_**(*base\_ring, fibre\_names, base\_names, max\_differential\_orders*)

Initialize this differential polynomial ring.

INPUT:

- `base_ring` – a ring, the ring of coefficients
- `fibre_names` – a list of strings, the names of the fibre variables
- `base_names` – a list of strings, the names of the base variables



- `max_differential_orders` – a list of natural numbers, the maximum differential order of each fibre variable

**base\_variables()**

Return the tuple of base variables of this differential polynomial ring.

**element\_class**

alias of *DifferentialPolynomial*

**fibre\_variables()**

Return the tuple of fibre variables of this differential polynomial ring.

**homogeneous\_monomials**(*fibre\_degrees, weights, max\_differential\_orders=None*)

Return the list of differential monomials with the given degrees and weights.

## 1.6 Homogeneous differential polynomial equation solver

Homogeneous differential polynomial equation solver

`gcaops.algebra.differential_polynomial_solver.solve_homogeneous_diffpoly`(*target, source, unknowns*)

Return a solution of a homogeneous differential polynomial equation.

INPUT:

- `target` – a homogeneous differential polynomial, the right-hand side of the equation
- `source` – a homogeneous differential polynomial, the left-hand side of the equation
- `unknowns` – a list of fibre variables, such that the total derivatives of those variables appear in `source`

ALGORITHM:

Builds an ansatz based on the homogeneity, and solves the arising linear system.



---

## ABSTRACT BASE CLASSES

### 2.1 Graph basis

Graph basis

**class** gcaops.graph.graph\_basis.**GraphBasis**

Bases: abc.ABC

Basis of a module spanned by graphs.

A basis consists of tuples `grading + (index, ...)` where e.g. `grading = (num_vertices, num_edges)` and `grading + (index,)` identifies the isomorphism class of the graph.

**graph\_properties()**

Return a dictionary containing the properties of the graphs in this basis.

**graph\_to\_key**(*graph*)

Return a tuple consisting of the key in this basis and the sign factor such that `graph` equals the sign times the graph identified by the key.

INPUT:

- `graph` – a graph

**key\_to\_graph**(*key*)

Return a tuple consisting of a graph and the sign factor such that the sign times the graph equals the graph identified by the key.

INPUT:

- `key` – a key in this basis

### 2.2 Graph vector

Graph vector

**class** gcaops.graph.graph\_vector.**GraphModule**

Bases: abc.ABC

Module spanned by graphs.

**\_\_call\_\_**(*arg*)

Convert `arg` into an element of this module.

**\_\_repr\_\_**()

Return a string representation of this module.

**base\_ring()**

Return the base ring of this module.

**basis()**

Return the basis of this module.

**zero()**

Return the zero vector in this module.

**class** gcaops.graph.graph\_vector.**GraphVector**

Bases: abc.ABC

Vector representing a linear combination of graphs.

**\_\_add\_\_**(*other*)

Return this graph vector added to *other*.

**\_\_eq\_\_**(*other*)

Return True if this graph vector is equal to *other* and False otherwise.

**\_\_iter\_\_**()

Returns an iterator over this graph vector, yielding tuples of the form (coeff, graph).

**\_\_len\_\_**()

Return the number of graphs with nonzero coefficients in this graph vector.

**\_\_mul\_\_**(*other*)

Return this graph vector multiplied by *other*.

**\_\_neg\_\_**()

Return the negative of this graph vector.

**\_\_pos\_\_**()

Return a copy of this graph vector.

**\_\_radd\_\_**(*other*)

Return *other* added to this graph vector.

**\_\_repr\_\_**()

Return a string representation of this graph vector.

**\_\_rmul\_\_**(*other*)

Return *other* multiplied by this graph vector.

**coefficient**(*monomial*)

Return the coefficient of *monomial* in this graph vector.

**copy**()

Return a copy of this graph vector.

**gradings**()

Return the set of grading tuples such that this graph vector contains terms with those gradings.

**homogeneous\_part**(\**grading*)

Return the homogeneous part of this graph vector consisting only of terms with the given grading.

**insertion**(*position*, *other*, *\*\*kwargs*)

Return the insertion of *other* into this graph vector at the vertex *position*.

**map\_coefficients**(*f*, *new\_parent=None*)

Apply *f* to each of this graph vector's coefficients and return the resulting graph vector.

**map\_graphs**(*f*, *new\_parent=None*)

Apply *f* to each of this graph vector's graphs and return the resulting graph vector.

**nedges()**

Return the number of edges in each graph in this graph vector.

ASSUMPTIONS:

Assumes all graphs in this graph vector have the same number of edges.

**nvertices()**

Return the number of vertices in each graph in this graph vector.

ASSUMPTIONS:

Assumes all graphs in this graph vector have the same number of vertices.

**parent()**

Return the parent GraphModule that this graph vector belongs to.

**plot(\*\*options)**

Return a plot of this graph vector.

**show(\*\*options)**

Show this graph.

## 2.3 Graph complex

Graph complex

**class** gcaops.graph.graph\_complex.**GraphCochain**

Bases: *gcaops.graph.graph\_vector.GraphVector*

Cochain of a *GraphComplex*.

**bracket(other)**

Return the graph Lie bracket of this graph cochain with *other*.

**differential()**

Return the graph differential of this graph cochain.

**class** gcaops.graph.graph\_complex.**GraphComplex**

Bases: *gcaops.graph.graph\_vector.GraphModule*

Graph complex.

## 2.4 Graph vector (vector backend)

Graph vector (vector backend)

**class** gcaops.graph.graph\_vector\_vector.**GraphModule\_vector**(*base\_ring*, *graph\_basis*,  
*vector\_constructor*, *matrix\_constructor*, *sparse=True*)

Bases: *gcaops.graph.graph\_vector.GraphModule*

Module spanned by graphs (with elements stored as dictionaries of vectors).

**\_\_call\_\_(arg)**

Convert *arg* into an element of this module.

**\_\_init\_\_(base\_ring, graph\_basis, vector\_constructor, matrix\_constructor, sparse=True)**

Initialize this graph module.

INPUT:

- `base_ring` – a ring, to be used as the ring of coefficients
- `graph_basis` – a *GraphBasis*
- `vector_constructor` – constructor of (sparse) vectors
- `matrix_constructor` – constructor of (sparse) matrices
- `sparse` – (default: `True`) a boolean, passed along to both constructors as a keyword argument

`__repr__()`

Return a string representation of this module.

`base_ring()`

Return the base ring of this module.

`basis()`

Return the basis of this module.

`zero()`

Return the zero vector in this module.

**class** `gcaops.graph.graph_vector_vector.GraphVector_vector`(*parent, vectors*)

Bases: `gcaops.graph.graph_vector.GraphVector`

Vector representing a linear combination of graphs (stored as a dictionary of vectors).

`__add__(other)`

Return this graph vector added to `other`.

`__eq__(other)`

Return `True` if this graph vector is equal to `other` and `False` otherwise.

`__init__(parent, vectors)`

Initialize this graph vector.

INPUT:

- `parent` – a *GraphModule*
- `vectors` – a dictionary, mapping gradings to (sparse) vectors of coefficients with respect to the basis of `parent`

`__iter__()`

Facilitates iterating over this graph vector, yielding tuples of the form (`coeff`, `graph`).

`__len__()`

Return the number of graphs with nonzero coefficients in this graph vector.

`__mul__(other)`

Return this graph vector multiplied by `other`.

`__neg__()`

Return the negative of this graph vector.

`__pos__()`

Return a copy of this graph vector.

`__radd__(other)`

Return `other` added to this graph vector.

`__repr__()`

Return a string representation of this graph vector.

**\_\_rmul\_\_(other)**  
Return *other* multiplied by this graph vector.

**\_\_rsub\_\_(other)**  
Return this graph vector subtracted from *other*.

**\_\_sub\_\_(other)**  
Return *other* subtracted from this graph vector.

**copy()**  
Return a copy of this graph vector.

**gradings()**  
Return the set of grading tuples such that this graph vector contains terms with those gradings.

**homogeneous\_part(\*grading)**  
Return the homogeneous part of this graph vector consisting only of terms with the given grading.

**insertion(position, other, \*\*kwargs)**  
Return the insertion of *other* into this graph vector at the vertex *position*.

**map\_coefficients(f, new\_parent=None)**  
Apply *f* to each of this graph vector's coefficients and return the resulting graph vector.

**parent()**  
Return the parent *GraphModule* that this graph vector belongs to.

**vector(\*grading)**  
Return the vector of coefficients of graphs with the given grading.

## 2.5 Graph vector (dictionary backend)

Graph vector (dictionary backend)

**class** `gcaops.graph.graph_vector_dict.GraphModule_dict`(*base\_ring*, *graph\_basis*)

Bases: `gcaops.graph.graph_vector.GraphModule`

Module spanned by graphs (with elements stored as dictionaries).

**\_\_call\_\_(arg)**

Convert *arg* into an element of this module.

**\_\_init\_\_(base\_ring, graph\_basis)**

Initialize this graph module.

INPUT:

- *base\_ring* – a ring, to be used as the ring of coefficients
- *graph\_basis* – a *GraphBasis*

**\_\_repr\_\_()**

Return a string representation of this module.

**base\_ring()**

Return the base ring of this module.

**basis()**

Return the basis of this module.

**zero()**

Return the zero vector in this module.

**class** gcaops.graph.graph\_vector\_dict.**GraphVector\_dict**(parent, vector)

Bases: *gcaops.graph.graph\_vector.GraphVector*

Vector representing a linear combination of graphs (stored as a dictionary).

**\_\_add\_\_**(other)

Return this graph vector added to other.

**\_\_eq\_\_**(other)

Return True if this graph vector is equal to other and False otherwise.

**\_\_init\_\_**(parent, vector)

Initialize this graph vector.

INPUT:

- parent – a *GraphModule*
- vector – a dictionary, representing a sparse vector of coefficients with respect to the basis of parent

**\_\_iter\_\_**()

Facilitates iterating over this graph vector, yielding tuples of the form (coeff, graph).

**\_\_len\_\_**()

Return the number of graphs with nonzero coefficients in this graph vector.

**\_\_mul\_\_**(other)

Return this graph vector multiplied by other.

**\_\_neg\_\_**()

Return the negative of this graph vector.

**\_\_pos\_\_**()

Return a copy of this graph vector.

**\_\_radd\_\_**(other)

Return other added to this graph vector.

**\_\_repr\_\_**()

Return a string representation of this graph vector.

**\_\_rmul\_\_**(other)

Return other multiplied by this graph vector.

**\_\_rsub\_\_**(other)

Return this graph vector subtracted from other.

**\_\_sub\_\_**(other)

Return other subtracted from this graph vector.

**copy**()

Return a copy of this graph vector.

**gradings**()

Return the set of grading tuples such that this graph vector contains terms with those gradings.

**homogeneous\_part**(\*grading)

Return the homogeneous part of this graph vector consisting only of terms with the given grading.

**insertion**(position, other, \*\*kwargs)

Return the insertion of other into this graph vector at the vertex position.

**map\_coefficients**(f, new\_parent=None)

Apply f to each of this graph vector's coefficients and return the resulting graph vector.



**parent()**

Return the parent *GraphModule* that this graph vector belongs to.



## UNDIRECTED GRAPHS

### 3.1 Undirected graph

Undirected graph

**class** `gcaops.graph.undirected_graph.UndirectedGraph`(*num\_vertices*, *edges*)

Bases: object

Undirected graph with vertices labeled by natural numbers and an ordered set of edges.

**\_\_eq\_\_**(*other*)

Return True if this graph equals *other*.

Note that this is *not* an isomorphism test, and the ordering of the list of edges is taken into account.

EXAMPLES:

```
sage: g = UndirectedGraph(3, [(0, 1), (1, 2), (2, 0)])
sage: h1 = UndirectedGraph(3, [(0, 1), (1, 2), (0, 2)])
sage: g == h1
True
sage: h2 = UndirectedGraph(3, [(0, 1), (0, 2), (1, 2)])
sage: g == h2
False
```

**\_\_init\_\_**(*num\_vertices*, *edges*)

Initialize this undirected graph.

INPUT:

- *num\_vertices* – a natural number, the number of vertices
- *edges* – a list of tuples of natural numbers

EXAMPLES:

1. Construct the graph consisting of a single edge:

```
sage: g = UndirectedGraph(2, [(0, 1)]); g
UndirectedGraph(2, [(0, 1)])
```

2. Construct the tetrahedron graph:

```
sage: g = UndirectedGraph(4, [(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)]); g
UndirectedGraph(4, [(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)])
```

**\_\_len\_\_**()

Return the number of vertices of this graph.

EXAMPLES:

```
sage: g = UndirectedGraph(3, [(0, 1), (1, 2), (2, 0)])
sage: len(g)
3
```

**\_\_repr\_\_()**

Return a string representation of this graph.

EXAMPLES:

```
sage: g = UndirectedGraph(3, [(0, 1), (1, 2), (2, 0)])
sage: repr(g)
'UndirectedGraph(3, [(0, 1), (1, 2), (0, 2)])'
```

**canonicalize\_edges()**

Lexicographically order the edges of this graph and return the sign of that edge permutation.

EXAMPLES:

```
sage: g = UndirectedGraph(3, [(1, 2), (0, 1)])
sage: g.canonicalize_edges()
-1
sage: g
UndirectedGraph(3, [(0, 1), (1, 2)])
```

**edges()**

Return the list of edges of this graph.

EXAMPLES:

```
sage: g = UndirectedGraph(3, [(0, 1), (1, 2)])
sage: g.edges()
[(0, 1), (1, 2)]
```

**get\_pos()**

Return the dictionary of positions of vertices in this graph (used for plotting).

EXAMPLES:

```
sage: g = UndirectedGraph(2, [(0, 1)])
sage: g.get_pos() is None
True
sage: g.set_pos({0: (0.0, 0.0), 1: (1.0, 0.0)})
sage: g.get_pos()
{0: (0.0000000000000000, 0.0000000000000000),
 1: (1.0000000000000000, 0.0000000000000000)}
```

**orientations()**

Return a generator producing the DirectedGraphs which are obtained by orienting this graph in all possible ways.

EXAMPLES:

```
sage: g = UndirectedGraph(2, [(0, 1)])
sage: list(g.orientations())
[DirectedGraph(2, [(0, 1)]), DirectedGraph(2, [(1, 0)]]
```

**plot(\*\*options)**

Return a plot of this graph.

EXAMPLES:

```
sage: g = UndirectedGraph(2, [(0, 1)])
sage: g.plot()
Graphics object consisting of 4 graphics primitives
```

**reabeled**(*relabeling*)

Return the graph obtained by relabeling this graph in the given way.

EXAMPLES:

```
sage: g = UndirectedGraph(3, [(0, 1), (1, 2)])
sage: g.reabeled({0: 1, 1: 0, 2: 2})
UndirectedGraph(3, [(0, 1), (0, 2)])
```

**set\_pos**(*new\_pos*)

Set the positions of vertices in this graph (used for plotting).

EXAMPLES:

```
sage: g = UndirectedGraph(2, [(0, 1)])
sage: g.set_pos({0: (0.0, 0.0), 1: (1.0, 0.0)})
sage: g.get_pos()
{0: (0.0000000000000000, 0.0000000000000000),
 1: (1.0000000000000000, 0.0000000000000000)}
```

**show**(*\*\*options*)

Show this graph.

EXAMPLES:

```
sage: g = UndirectedGraph(2, [(0, 1)])
sage: g.show()
```

## 3.2 Undirected graph basis

Undirected graph basis

**class** `gcaops.graph.undirected_graph_basis.UndirectedGraphBasis`

Bases: `gcaops.graph.graph_basis.GraphBasis`

Basis of a module spanned by undirected graphs.

A basis consists of keys  $(v, e, \text{index}, \dots)$  where  $(v, e, \text{index})$  identifies the isomorphism class of the graph.

**graph\_class**

alias of `gcaops.graph.undirected_graph.UndirectedGraph`

**class** `gcaops.graph.undirected_graph_basis.UndirectedGraphComplexBasis`(*connected=None*,  
*bicon-*  
*nected=None*,  
*min\_degree=0*)

Bases: `gcaops.graph.undirected_graph_basis.UndirectedGraphBasis`

Basis consisting of representatives of isomorphism classes of undirected graphs with no automorphisms that induce an odd permutation on edges

**cardinality**(*vertices, edges*)

Return the number of graphs in this basis with the given amount of vertices and edges.

**graph\_properties()**

Return a dictionary containing the properties of the graphs in this basis.

**graph\_to\_key(*graph*)**

Return a tuple consisting of the key in this basis and the sign factor such that *graph* equals the sign times the graph identified by the key.

INPUT:

- *graph* – an *UndirectedGraph*

OUTPUT:

Either (None, 1) if the input *graph* is not in the span of the basis, or a tuple consisting of a key and a sign, where a key is a tuple consisting of the number of vertices, the number of edges, and the index of the graph in the list.

**graphs(*vertices*, *edges*)**

Return the list of graphs in this basis with the given amount of *vertices* and *edges*.

**key\_to\_graph(*key*)**

Return a tuple consisting of an *UndirectedGraph* and the sign factor such that the sign times the graph equals the graph identified by the key.

INPUT:

- *key* – a key in this basis

OUTPUT:

Either (None, 1) if the input *key* is not in the basis, or a tuple consisting of an *UndirectedGraph* and a sign which is always +1.

**class gcaops.graph.undirected\_graph\_basis.UndirectedGraphOperadBasis**

Bases: *gcaops.graph.undirected\_graph\_basis.UndirectedGraphBasis*

Basis consisting of labeled undirected graphs with no automorphisms that induce an odd permutation on edges

**graph\_properties()**

Return a dictionary containing the properties of the graphs in this basis.

**graph\_to\_key(*graph*)**

Return a tuple consisting of the key in this basis and the sign factor such that *graph* equals the sign times the graph identified by the key.

INPUT:

- *graph* – an *UndirectedGraph*

OUTPUT:

Either (None, 1) if the input *graph* is not in the span of the basis, or a tuple consisting of a key and a sign, where a key is a tuple consisting of the number of vertices, the number of edges, the index of the graph in the list, followed by a permutation of vertices.

**key\_to\_graph(*key*)**

Return a tuple consisting of an *UndirectedGraph* and the sign factor such that the sign times the graph equals the graph identified by the key.

INPUT:

- *key* – a key in this basis

OUTPUT:

Either (None, 1) if the input key is not in the basis, or a tuple consisting of an *UndirectedGraph* and a sign.

### 3.3 Undirected graph vector

Undirected graph vector

**class** gcaops.graph.undirected\_graph\_vector.UndirectedGraphModule

Bases: *gcaops.graph.graph\_vector.GraphModule*

Module spanned by undirected graphs.

**class** gcaops.graph.undirected\_graph\_vector.UndirectedGraphModule\_dict(*base\_ring*,  
*graph\_basis*)

Bases: *gcaops.graph.undirected\_graph\_vector.UndirectedGraphModule*, *gcaops.graph.graph\_vector\_dict.GraphModule\_dict*

Module spanned by undirected graphs (with elements stored as dictionaries).

**\_\_init\_\_**(*base\_ring*, *graph\_basis*)

Initialize this undirected graph module.

INPUT:

- *base\_ring* – a ring, to be used as the ring of coefficients
- *graph\_basis* – an *UndirectedGraphBasis*

**class** gcaops.graph.undirected\_graph\_vector.UndirectedGraphModule\_vector(*base\_ring*,  
*graph\_basis*,  
*vector\_constructor*,  
*matrix\_constructor*,  
*sparse=True*)

Bases: *gcaops.graph.undirected\_graph\_vector.UndirectedGraphModule*, *gcaops.graph.graph\_vector\_vector.GraphModule\_vector*

Module spanned by undirected graphs (with elements stored as dictionaries of vectors).

**\_\_init\_\_**(*base\_ring*, *graph\_basis*, *vector\_constructor*, *matrix\_constructor*, *sparse=True*)

Initialize this undirected graph module.

INPUT:

- *base\_ring* – a ring, to be used as the ring of coefficients
- *graph\_basis* – an *UndirectedGraphBasis*
- *vector\_constructor* – constructor of (sparse) vectors
- *matrix\_constructor* – constructor of (sparse) matrices
- *sparse* – (default: True) a boolean, passed along to both constructors as a keyword argument

**class** gcaops.graph.undirected\_graph\_vector.UndirectedGraphVector

Bases: *gcaops.graph.graph\_vector.GraphVector*

Vector representing a linear combination of undirected graphs.

**class** gcaops.graph.undirected\_graph\_vector.UndirectedGraphVector\_dict(*parent*, *vector*)

Bases: *gcaops.graph.undirected\_graph\_vector.UndirectedGraphVector*, *gcaops.graph.graph\_vector\_dict.GraphVector\_dict*

Vector representing a linear combination of undirected graphs (stored as a dictionary).

**\_\_init\_\_**(*parent, vector*)

Initialize this undirected graph vector.

INPUT:

- *parent* – an *UndirectedGraphModule*
- *vector* – a dictionary, representing a sparse vector of coefficients with respect to the basis of *parent*

**nedges**()

Return the number of edges in each graph in this graph vector.

ASSUMPTIONS:

Assumes all graphs in this graph vector have the same number of edges.

**nvertices**()

Return the number of vertices in each graph in this graph vector.

ASSUMPTIONS:

Assumes all graphs in this graph vector have the same number of vertices.

**class** `gcaops.graph.undirected_graph_vector.UndirectedGraphVector_vector`(*parent, vectors*)

Bases: `gcaops.graph.undirected_graph_vector.UndirectedGraphVector`, `gcaops.graph.graph_vector_vector.GraphVector_vector`

Vector representing a linear combination of undirected graphs (stored as a dictionary of vectors).

**\_\_init\_\_**(*parent, vectors*)

Initialize this graph vector.

INPUT:

- *parent* – an *UndirectedGraphModule*
- *vectors* – a dictionary, mapping bi-gradings to (sparse) vectors of coefficients with respect to the basis of *parent*

**nedges**()

Return the number of edges in each graph in this graph vector.

ASSUMPTIONS:

Assumes all graphs in this graph vector have the same number of edges.

**nvertices**()

Return the number of vertices in each graph in this graph vector.

ASSUMPTIONS:

Assumes all graphs in this graph vector have the same number of vertices.



### 3.4 Undirected graph operad

Undirected graph operad

`gcaops.graph.undirected_graph_operad.UndirectedGraphOperad(base_ring)`  
Return the operad of undirected graphs over the given `base_ring`.

**class** `gcaops.graph.undirected_graph_operad.UndirectedGraphOperad_dict(base_ring)`  
Bases: `gcaops.graph.undirected_graph_vector.UndirectedGraphModule_dict`

Operad of undirected graphs (with elements stored as dictionaries).

`__init__(base_ring)`  
Initialize this graph operad.

`__repr__()`  
Return a string representation of this graph operad.

**class** `gcaops.graph.undirected_graph_operad.UndirectedGraphOperation_dict(parent, vector)`  
Bases: `gcaops.graph.undirected_graph_vector.UndirectedGraphVector_dict`

Element of an `UndirectedGraphOperad_dict` (stored as a dictionary).

`__init__(parent, vector)`  
Initialize this graph operation.

### 3.5 Undirected graph complex

Undirected graph complex

**class** `gcaops.graph.undirected_graph_complex.UndirectedGraphCochain`  
Bases: `gcaops.graph.graph_complex.GraphCochain`, `gcaops.graph.undirected_graph_vector.UndirectedGraphVector`

Cochain of an `UndirectedGraphComplex_`.

`bracket(other)`  
Return the graph Lie bracket of this graph cochain with `other`.

**class** `gcaops.graph.undirected_graph_complex.UndirectedGraphCochain_dict(parent, vector)`  
Bases: `gcaops.graph.undirected_graph_complex.UndirectedGraphCochain`, `gcaops.graph.undirected_graph_vector.UndirectedGraphVector_dict`

Cochain of an `UndirectedGraphComplex_dict` (stored as a dictionary).

`__init__(parent, vector)`  
Initialize this graph cochain.

`differential()`  
Return the graph differential of this graph cochain.

**class** `gcaops.graph.undirected_graph_complex.UndirectedGraphCochain_vector(parent, vector)`  
Bases: `gcaops.graph.undirected_graph_complex.UndirectedGraphCochain`, `gcaops.graph.undirected_graph_vector.UndirectedGraphVector_vector`

Cochain of an `UndirectedGraphComplex_vector` (stored as a dictionary of vectors).

`__init__(parent, vector)`  
Initialize this graph cochain.

**differential**(*use\_cache=True*)

Return the graph differential of this graph cochain.

**is\_coboundary**(*certificate=False*)

Return True if this graph cochain is a coboundary.

INPUT:

- *certificate* - if True, return a tuple where the first element is the truth value, and the second element is a graph cochain such that its differential is this graph cochain (or None).

`gcaops.graph.undirected_graph_complex.UndirectedGraphComplex`(*base\_ring*, *connected=None*, *biconnected=None*, *min\_degree=0*, *implementation='dict'*, *vector\_constructor=None*, *matrix\_constructor=None*, *sparse=True*)

Return the undirected graph complex over *base\_ring* with the given properties.

**class** `gcaops.graph.undirected_graph_complex.UndirectedGraphComplex_`

Bases: `gcaops.graph.graph_complex.GraphComplex`, `gcaops.graph.undirected_graph_vector.UndirectedGraphModule`

Undirected graph complex.

**class** `gcaops.graph.undirected_graph_complex.UndirectedGraphComplex_dict`(*base\_ring*, *connected=None*, *biconnected=None*, *min\_degree=0*)

Bases: `gcaops.graph.undirected_graph_complex.UndirectedGraphComplex_`, `gcaops.graph.undirected_graph_vector.UndirectedGraphModule_dict`

Undirected graph complex (with elements stored as dictionaries).

**\_\_init\_\_**(*base\_ring*, *connected=None*, *biconnected=None*, *min\_degree=0*)  
Initialize this graph complex.

**\_\_repr\_\_**()  
Return a string representation of this graph complex.

**class** `gcaops.graph.undirected_graph_complex.UndirectedGraphComplex_vector`(*base\_ring*, *vector\_constructor*, *matrix\_constructor*, *sparse=True*, *connected=None*, *biconnected=None*, *min\_degree=0*)

Bases: `gcaops.graph.undirected_graph_complex.UndirectedGraphComplex_`, `gcaops.graph.undirected_graph_vector.UndirectedGraphModule_vector`

Undirected graph complex (with elements stored as dictionaries of vectors).

**\_\_init\_\_**(*base\_ring*, *vector\_constructor*, *matrix\_constructor*, *sparse=True*, *connected=None*, *biconnected=None*, *min\_degree=0*)  
Initialize this graph complex.

INPUT:

- `base_ring` – a ring, to be used as the ring of coefficients
- `vector_constructor` – constructor of (sparse) vectors
- `matrix_constructor` – constructor of (sparse) matrices
- `sparse` – (default: `True`) a boolean, passed along to both constructors as a keyword argument

`__repr__()`

Return a string representation of this graph complex.

`cohomology_basis(vertices, edges)`

Return a basis of the cohomology in the given bi-grading (`vertices`, `edges`).



## DIRECTED GRAPHS

### 4.1 Directed graph

Directed graph

**class** `gcaops.graph.directed_graph.DirectedGraph`(*num\_vertices*, *edges*)

Bases: `object`

Directed graph with vertices labeled by natural numbers and an ordered set of edges.

**canonicalize\_edges()**

Lexicographically order the edges of this graph and return the sign of that edge permutation.

EXAMPLES:

```
sage: g = DirectedGraph(3, [(2, 1), (2, 0)])
sage: g.canonicalize_edges()
-1
sage: g
DirectedGraph(3, [(2, 0), (2, 1)])
```

**edges()**

Return the list of edges of this graph.

EXAMPLES:

```
sage: g = DirectedGraph(3, [(0, 1), (1, 2)])
sage: g.edges()
[(0, 1), (1, 2)]
```

**get\_pos()**

Return the dictionary of positions of vertices in this graph (used for plotting).

EXAMPLES:

```
sage: g = DirectedGraph(2, [(0, 1)])
sage: g.get_pos() is None
True
sage: g.set_pos({0: (0.0, 0.0), 1: (1.0, 0.0)})
sage: g.get_pos()
{0: (0.0000000000000000, 0.0000000000000000),
 1: (1.0000000000000000, 0.0000000000000000)}
```

**in\_degrees()**

Return the tuple of in-degrees of vertices of this graph.

EXAMPLES:

```
sage: g = DirectedGraph(4, [(1, 0), (2, 0), (3, 0), (1, 2), (2, 3), (3, 1)])
sage: g.in_degrees()
(3, 1, 1, 1)
```

**out\_degrees()**

Return the tuple of out-degrees of vertices of this graph.

EXAMPLES:

```
sage: g = DirectedGraph(4, [(1, 0), (2, 0), (3, 0), (1, 2), (2, 3), (3, 1)])
sage: g.out_degrees()
(0, 2, 2, 2)
```

**plot(\*\*options)**

Return a plot of this graph.

EXAMPLES:

```
sage: g = DirectedGraph(2, [(0, 1)])
sage: g.plot()
Graphics object consisting of 4 graphics primitives
```

**reabeled(relabeling)**

Return the graph obtained by relabeling this graph in the given way.

EXAMPLES:

```
sage: g = DirectedGraph(3, [(2, 0), (2, 1)])
sage: g.reabeled({0: 1, 1: 0, 2: 2})
DirectedGraph(3, [(2, 1), (2, 0)])
```

**set\_pos(new\_pos)**

Set the positions of vertices in this graph (used for plotting).

EXAMPLES:

```
sage: g = DirectedGraph(2, [(0, 1)])
sage: g.set_pos({0: (0.0, 0.0), 1: (1.0, 0.0)})
sage: g.get_pos()
{0: (0.0000000000000000, 0.0000000000000000),
 1: (1.0000000000000000, 0.0000000000000000)}
```

**show(\*\*options)**

Show this graph.

EXAMPLES:

```
sage: g = DirectedGraph(2, [(0, 1)])
sage: g.show()
```

## 4.2 Directed graph basis

Directed graph basis

**class** gcaops.graph.directed\_graph\_basis.DirectedGraphBasis

Bases: *gcaops.graph.graph\_basis.GraphBasis*

Basis of a module spanned by directed graphs.

A basis consists of keys  $(v, e, \text{index}, \dots)$  where  $(v, e, \text{index})$  identifies the isomorphism class of the graph.

**graph\_class**

alias of `gcaops.graph.directed_graph.DirectedGraph`

```
class gcaops.graph.directed_graph_basis.DirectedGraphComplexBasis(connected=None,
                                                                biconnected=None,
                                                                min_degree=0,
                                                                loops=True)
```

Bases: `gcaops.graph.directed_graph_basis.DirectedGraphBasis`

Basis consisting of representatives of isomorphism classes of directed graphs with no automorphisms that induce an odd permutation on edges

**cardinality**(*vertices, edges*)

Return the number of graphs in this basis with the given amount of *vertices* and *edges*.

**graph\_properties**()

Return a dictionary containing the properties of the graphs in this basis.

**graph\_to\_key**(*graph*)

Return a tuple consisting of the key in this basis and the sign factor such that *graph* equals the sign times the graph identified by the key.

INPUT:

- *graph* – a `DirectedGraph`

OUTPUT:

Either  $(None, 1)$  if the input *graph* is not in the span of the basis, or a tuple consisting of a key and a sign, where a key is a tuple consisting of the number of vertices, the number of edges, and the index of the graph in the list.

**graphs**(*vertices, edges*)

Return the list of graphs in this basis with the given amount of *vertices* and *edges*.

**key\_to\_graph**(*key*)

Return a tuple consisting of a `DirectedGraph` and the sign factor such that the sign times the graph equals the graph identified by the key.

INPUT:

- *key* – a key in this basis

OUTPUT:

Either  $(None, 1)$  if the input *key* is not in the basis, or a tuple consisting of a `DirectedGraph` and a sign which is always +1.

```
class gcaops.graph.directed_graph_basis.DirectedGraphOperadBasis
```

Bases: `gcaops.graph.directed_graph_basis.DirectedGraphBasis`

Basis consisting of labeled directed graphs with no automorphisms that induce an odd permutation on edges

**graph\_properties**()

Return a dictionary containing the properties of the graphs in this basis.

**graph\_to\_key**(*graph*)

Return a tuple consisting of the key in this basis and the sign factor such that *graph* equals the sign times the graph identified by the key.

INPUT:

- graph – a *DirectedGraph*

OUTPUT:

Either (None, 1) if the input graph is not in the span of the basis, or a tuple consisting of a key and a sign, where a key is a tuple consisting of the number of vertices, the number of edges, the index of the graph in the list, followed by a permutation of vertices.

**key\_to\_graph**(key)

Return a tuple consisting of a *DirectedGraph* and the sign factor such that the sign times the graph equals the graph identified by the key.

INPUT:

- key – a key in this basis

OUTPUT:

Either (None, 1) if the input key is not in the basis, or a tuple consisting of a *DirectedGraph* and a sign.

### 4.3 Directed graph vector

Directed graph vector

**class** gcaops.graph.directed\_graph\_vector.*DirectedGraphModule*

Bases: *gcaops.graph.graph\_vector.GraphModule*

Module spanned by directed graphs.

**\_\_call\_\_**(arg)

Convert arg into an element of this module.

**class** gcaops.graph.directed\_graph\_vector.*DirectedGraphModule\_dict*(base\_ring,  
graph\_basis)

Bases: *gcaops.graph.directed\_graph\_vector.DirectedGraphModule*, *gcaops.graph.graph\_vector\_dict.GraphModule\_dict*

Module spanned by directed graphs (with elements stored as dictionaries).

**\_\_init\_\_**(base\_ring, graph\_basis)

Initialize this directed graph module.

INPUT:

- base\_ring – a ring, to be used as the ring of coefficients
- graph\_basis – a *DirectedGraphBasis*

**class** gcaops.graph.directed\_graph\_vector.*DirectedGraphModule\_vector*(base\_ring,  
graph\_basis, vector\_constructor,  
matrix\_constructor,  
sparse=True)

Bases: *gcaops.graph.directed\_graph\_vector.DirectedGraphModule*, *gcaops.graph.graph\_vector\_vector.GraphModule\_vector*

Module spanned by directed graphs (with elements stored as dictionaries of vectors).

**\_\_init\_\_**(base\_ring, graph\_basis, vector\_constructor, matrix\_constructor, sparse=True)

Initialize this directed graph module.

INPUT:



- `base_ring` – a ring, to be used as the ring of coefficients
- `graph_basis` – a *DirectedGraphBasis*
- `vector_constructor` – constructor of (sparse) vectors
- `matrix_constructor` – constructor of (sparse) matrices
- `sparse` – (default: `True`) a boolean, passed along to both constructors as a keyword argument

**class** `gcaops.graph.directed_graph_vector.DirectedGraphVector`

Bases: `gcaops.graph.graph_vector.GraphVector`

Vector representing a linear combination of directed graphs.

**filter**(*max\_out\_degree=None*)

Return the graph vector which is the summand of this graph vector containing exactly those graphs that pass the filter.

**class** `gcaops.graph.directed_graph_vector.DirectedGraphVector_dict`(*parent, vector*)

Bases: `gcaops.graph.directed_graph_vector.DirectedGraphVector`, `gcaops.graph.graph_vector_dict.GraphVector_dict`

Vector representing a linear combination of directed graphs (stored as a dictionary).

**\_\_init\_\_**(*parent, vector*)

Initialize this directed graph vector.

INPUT:

- `parent` – a *DirectedGraphModule*
- `vector` – a dictionary, representing a sparse vector of coefficients with respect to the basis of `parent`

**filter**(*max\_out\_degree=None*)

Return the graph vector which is the summand of this graph vector containing exactly those graphs that pass the filter.

**nedges**()

Return the number of edges in each graph in this graph vector.

ASSUMPTIONS:

Assumes all graphs in this graph vector have the same number of edges.

**nvertices**()

Return the number of vertices in each graph in this graph vector.

ASSUMPTIONS:

Assumes all graphs in this graph vector have the same number of vertices.

**class** `gcaops.graph.directed_graph_vector.DirectedGraphVector_vector`(*parent, vectors*)

Bases: `gcaops.graph.directed_graph_vector.DirectedGraphVector`, `gcaops.graph.graph_vector_vector.GraphVector_vector`

Vector representing a linear combination of directed graphs (stored as a dictionary of vectors).

**\_\_init\_\_**(*parent, vectors*)

Initialize this graph vector.

INPUT:

- `parent` – a *DirectedGraphModule*
- `vectors` – a dictionary, mapping bi-gradings to (sparse) vectors of coefficients with respect to the basis of `parent`

**filter**(*max\_out\_degree=None*)

Return the graph vector which is the summand of this graph vector containing exactly those graphs that pass the filter.

**nedges**()

Return the number of edges in each graph in this graph vector.

ASSUMPTIONS:

Assumes all graphs in this graph vector have the same number of edges.

**nvertices**()

Return the number of vertices in each graph in this graph vector.

ASSUMPTIONS:

Assumes all graphs in this graph vector have the same number of vertices.

## 4.4 Directed graph operad

Directed graph operad

`gcaops.graph.directed_graph_operad.DirectedGraphOperad`(*base\_ring*)

Return the operad of directed graphs over the given *base\_ring*.

**class** `gcaops.graph.directed_graph_operad.DirectedGraphOperad_dict`(*base\_ring*)

Bases: `gcaops.graph.directed_graph_vector.DirectedGraphModule_dict`

Operad of directed graphs (with elements stored as dictionaries).

**\_\_init\_\_**(*base\_ring*)

Initialize this graph operad.

**\_\_repr\_\_**()

Return a string representation of this graph operad.

**class** `gcaops.graph.directed_graph_operad.DirectedGraphOperation_dict`(*parent, vector*)

Bases: `gcaops.graph.directed_graph_vector.DirectedGraphVector_dict`

Element of a `DirectedGraphOperad_dict` (stored as a dictionary).

**\_\_init\_\_**(*parent, vector*)

Initialize this graph operation.

## 4.5 Directed graph complex

Directed graph complex

**class** `gcaops.graph.directed_graph_complex.DirectedGraphCochain`

Bases: `gcaops.graph.graph_complex.GraphCochain`, `gcaops.graph.directed_graph_vector.DirectedGraphVector`

Cochain of a `DirectedGraphComplex_`.

**bracket**(*other*)

Return the graph Lie bracket of this graph cochain with *other*.

---

```

class gcaops.graph.directed_graph_complex.DirectedGraphCochain_dict(parent, vector)
    Bases: gcaops.graph.directed_graph_complex.DirectedGraphCochain, gcaops.graph.
           directed_graph_vector.DirectedGraphVector_dict
    Cochain of a DirectedGraphComplex_dict (stored as a dictionary).
    __init__(parent, vector)
        Initialize this graph cochain.
    differential()
        Return the graph differential of this graph cochain.
class gcaops.graph.directed_graph_complex.DirectedGraphCochain_vector(parent, vector)
    Bases: gcaops.graph.directed_graph_complex.DirectedGraphCochain, gcaops.graph.
           directed_graph_vector.DirectedGraphVector_vector
    Cochain of a DirectedGraphComplex_vector (stored as a dictionary of vectors).
    __init__(parent, vector)
        Initialize this graph cochain.
    differential(use_cache=True)
        Return the graph differential of this graph cochain.
    is_coboundary(certificate=False)
        Return True if this graph cochain is a coboundary.
    INPUT:
        • certificate - if True, return a tuple where the first element is the truth value, and the second
          element is a graph cochain such that its differential is this graph cochain (or None).
gcaops.graph.directed_graph_complex.DirectedGraphComplex(base_ring, connected=None, bi-
                                                         connected=None, min_degree=0,
                                                         loops=True, implementation='dict',
                                                         vector_constructor=None,
                                                         matrix_constructor=None,
                                                         sparse=True)
    Return the directed graph complex over base_ring with the given properties.
class gcaops.graph.directed_graph_complex.DirectedGraphComplex_
    Bases: gcaops.graph.graph_complex.GraphComplex, gcaops.graph.directed_graph_vector.
           DirectedGraphModule
    Directed graph complex.
class gcaops.graph.directed_graph_complex.DirectedGraphComplex_dict(base_ring, con-
                                                         nected=None, bi-
                                                         connected=None,
                                                         min_degree=0,
                                                         loops=True)
    Bases: gcaops.graph.directed_graph_complex.DirectedGraphComplex_, gcaops.graph.
           directed_graph_vector.DirectedGraphModule_dict
    Directed graph complex (with elements stored as dictionaries).
    __init__(base_ring, connected=None, biconnected=None, min_degree=0, loops=True)
        Initialize this graph complex.
    __repr__()
        Return a string representation of this graph complex.

```

```
class gcaops.graph.directed_graph_complex.DirectedGraphComplex_vector(base_ring, vector_constructor, matrix_constructor, sparse=True, connected=None, biconnected=None, min_degree=0, loops=True)
```

Bases: `gcaops.graph.directed_graph_complex.DirectedGraphComplex_`, `gcaops.graph.directed_graph_vector.DirectedGraphModule_vector`

Directed graph complex (with elements stored as dictionaries of vectors).

```
__init__(base_ring, vector_constructor, matrix_constructor, sparse=True, connected=None, biconnected=None, min_degree=0, loops=True)  
Initialize this graph complex.
```

```
__repr__()  
Return a string representation of this graph complex.
```

```
cohomology_basis(vertices, edges)  
Return a basis of the cohomology in the given bi-grading (vertices, edges).
```

## FORMALITY GRAPHS

## 5.1 Formality graph

Formality graph

**class** gcaops.graph.formality\_graph.**FormalityGraph**(*num\_ground\_vertices*, *num\_aerial\_vertices*,  
*edges*)

Bases: object

Directed graph with an ordered set of edges, and vertices labeled by natural numbers, the first of which are ordered ground vertices without outgoing edges.

**aerial\_product**(*other*)

Return the product of this graph with the *other* graph (i.e. the disjoint union followed by the identification of the ground vertices).

EXAMPLES:

```
sage: g = FormalityGraph(2, 1, [(2, 0), (2, 1)])
sage: g.aerial_product(g)
FormalityGraph(2, 2, [(2, 0), (2, 1), (3, 0), (3, 1)])
```

**automorphism\_group**()

Return the automorphism group of this graph.

EXAMPLES:

```
sage: g = FormalityGraph(2, 2, [(2, 0), (2, 1), (3, 0), (3, 1)])
sage: g.automorphism_group()
Permutation Group with generators [(2,3)]
```

**canonicalize\_edges**()

Lexicographically order the edges of this graph and return the sign of that edge permutation.

EXAMPLES:

```
sage: g = FormalityGraph(2, 1, [(2, 1), (2, 0)])
sage: g.canonicalize_edges()
-1
sage: g
FormalityGraph(2, 1, [(2, 0), (2, 1)])
```

**differential\_orders**()

Return the tuple of in-degrees of the ground vertices of this graph.

EXAMPLES:

```
sage: g = FormalityGraph(2, 1, [(2, 0), (2, 1)])
sage: g.differential_orders()
(1, 1)
sage: h = FormalityGraph(3, 2, [(3, 0), (3, 1), (3, 2), (4, 3), (4, 2)])
sage: h.differential_orders()
(1, 1, 2)
```

**edge\_contraction\_graph(*edge*)**

Return the *FormalityGraph* which is obtained by contracting the edge *edge* between aerial vertices in this graph.

EXAMPLES:

```
sage: g = FormalityGraph(3, 2, [(3, 0), (3, 1), (4, 3), (4, 2)])
sage: g.edge_contraction_graph((4, 3))
FormalityGraph(3, 1, [(3, 0), (3, 1), (3, 2)])
```

**edges()**

Return the list of edges of this graph.

EXAMPLES:

```
sage: g = FormalityGraph(2, 1, [(2, 0), (2, 1)])
sage: g.edges()
[(2, 0), (2, 1)]
```

**edges\_in\_air()**

Return the list of edges between aerial vertices of this graph.

EXAMPLES:

```
sage: g = FormalityGraph(2, 2, [(2, 0), (2, 1), (3, 2), (3, 1)])
sage: g.edges_in_air()
[(3, 2)]
sage: h = FormalityGraph(2, 1, [(2, 0), (2, 1)])
sage: h.edges_in_air()
[]
```

**static from\_kgs\_encoding(*kgs\_encoding*)**

Return a tuple consisting of a sign and a *FormalityGraph* built of wedges, as specified by the given encoding.

INPUT:

- *kgs\_encoding* – a string, containing a graph encoding as used in Buring’s *kontsevich\_graph\_series-cpp* programs

See also:

[\*kgs\\_encoding\(\)\*](#)

EXAMPLES:

```
sage: FormalityGraph.from_kgs_encoding('2 1 1 0 1')
(1, FormalityGraph(2, 1, [(2, 0), (2, 1)]))
sage: FormalityGraph.from_kgs_encoding('2 2 1 0 1 2 1')
(1, FormalityGraph(2, 2, [(2, 0), (2, 1), (3, 2), (3, 1)]))
```

**static from\_kontsevint\_encoding(*kontsevint\_encoding*)**

Return the *FormalityGraph* specified by the given encoding.

INPUT:

- `kontsevint_encoding` – a string, containing a graph encoding as used in Panzer’s `kontsevint` program

**See also:**

[`kontsevint\_encoding\(\)`](#)

EXAMPLES:

```
sage: FormalityGraph.from_kontsevint_encoding('[L, R]')
FormalityGraph(2, 1, [(2, 0), (2, 1)])
sage: FormalityGraph.from_kontsevint_encoding('[p1, p2, p3]')
FormalityGraph(3, 1, [(3, 0), (3, 1), (3, 2)])
sage: FormalityGraph.from_kontsevint_encoding('[L, R], [1, R]')
FormalityGraph(2, 2, [(2, 0), (2, 1), (3, 2), (3, 1)])
```

**get\_pos()**

Return the dictionary of positions of vertices in this graph (used for plotting).

EXAMPLES:

```
sage: g = FormalityGraph(2, 1, [(2, 0), (2, 1)])
sage: g.get_pos() is None
True
sage: g.set_pos({0: (0.0, 0.0), 1: (1.0, 0.0), 2: (0.5, 1.0)})
sage: g.get_pos()
{0: (0.0000000000000000, 0.0000000000000000),
 1: (1.0000000000000000, 0.0000000000000000),
 2: (0.5000000000000000, 1.0000000000000000)}
```

**ground\_relabeled(*relabeling*)**

Return a ground vertex relabeling of this graph.

EXAMPLES:

```
sage: g = FormalityGraph(2, 1, [(2, 0), (2, 1)])
sage: g.ground_relabeled({0: 1, 1: 0})
FormalityGraph(2, 1, [(2, 1), (2, 0)])
```

**has\_eye\_on\_ground()**

Return True if this graph contains a 2-cycle between two aerial vertices which are connected to the same ground vertex, and False otherwise.

EXAMPLES:

```
sage: g = FormalityGraph(1, 2, [(1, 2), (2, 1), (1, 0), (2, 0)])
sage: g.has_eye_on_ground()
True
sage: h = FormalityGraph(2, 1, [(2, 0), (2, 1)])
sage: h.has_eye_on_ground()
False
```

**has\_loops()**

Return True if this graph contains an edge which is a loop, and False otherwise.

EXAMPLES:

```
sage: g = FormalityGraph(1, 1, [(1, 1)])
sage: g.has_loops()
True
sage: h = FormalityGraph(2, 1, [(2, 0), (2, 1)])
sage: h.has_loops()
False
```

**has\_multiple\_edges()**

Return True if this graph contains multiple edges, and False otherwise.

EXAMPLES:

```
sage: g = FormalityGraph(1, 1, [(1, 0), (1, 0)])
sage: g.has_multiple_edges()
True
sage: h = FormalityGraph(2, 1, [(2, 0), (2, 1)])
sage: h.has_multiple_edges()
False
```

**has\_odd\_automorphism()**

Return True if this graph has an automorphism that induces an odd permutation on its ordered set of edges.

EXAMPLES:

```
sage: g = FormalityGraph(2, 3, [(2, 0), (2, 1), (3, 0), (3, 1), (4, 2), (4, 3)])
sage: g.has_odd_automorphism()
True
sage: h = FormalityGraph(2, 1, [(2, 0), (2, 1)])
sage: h.has_odd_automorphism()
False
```

**in\_degrees()**

Return the tuple of in-degrees of vertices of this graph.

EXAMPLES:

```
sage: g = FormalityGraph(2, 1, [(2, 0), (2, 1)])
sage: g.in_degrees()
(1, 1, 0)
sage: h = FormalityGraph(3, 2, [(3, 0), (3, 1), (3, 2), (4, 3), (4, 2)])
sage: h.in_degrees()
(1, 1, 2, 1, 0)
```

**kgs\_encoding()**

Return the encoding of this graph for use in Buring's `kontsevich_graph_series-cpp` programs.

ASSUMPTIONS:

Assumes that this graph is built of wedges (i.e. each aerial vertex has out-degree two).

See also:

[\*from\\_kgs\\_encoding\(\)\*](#)

EXAMPLES:

```
sage: g = FormalityGraph(2, 1, [(2, 0), (2, 1)])
sage: g.kgs_encoding()
'2 1 1 0 1'
sage: h = FormalityGraph(2, 2, [(2, 0), (2, 1), (3, 2), (3, 1)])
sage: h.kgs_encoding()
'2 2 1 0 1 2 1'
```

**kontsevint\_encoding()**

Return the encoding of this graph for use in Panzer's `kontsevint` program.

ASSUMPTIONS:

Assumes `len(self.edges()) == 2*self.num_aerial_vertices() - 2 + self.num_ground_vertices()`.

See also:



*from\_kontsevint\_encoding()*

EXAMPLES:

```
sage: g = FormalityGraph(2, 1, [(2, 0), (2, 1)])
sage: g.kontsevint_encoding()
'[[p1,p2]]'
sage: h = FormalityGraph(2, 2, [(2, 0), (2, 3), (3, 1), (3, 2)])
sage: h.kontsevint_encoding()
'[[p1,2],[p2,1]]'
```

**multiplicity()**

Return the number of formality graphs isomorphic to this one, under isomorphisms that preserve the ground vertices pointwise.

EXAMPLES:

```
sage: g1 = FormalityGraph(2, 1, [(2, 0), (2, 1)])
sage: g1.multiplicity()
2
sage: g2 = FormalityGraph(2, 2, [(2, 0), (2, 1), (3, 2), (3, 1)])
sage: g2.multiplicity()
8
sage: g3 = FormalityGraph(2, 2, [(2, 0), (2, 1), (3, 0), (3, 1)])
sage: g3.multiplicity()
4
```

**num\_aerial\_vertices()**

Return the number of aerial vertices of this graph.

EXAMPLES:

```
sage: g = FormalityGraph(2, 1, [(2, 0), (2, 1)])
sage: g.num_aerial_vertices()
1
```

**num\_ground\_vertices()**

Return the number of ground vertices of this graph.

EXAMPLES:

```
sage: g = FormalityGraph(2, 1, [(2, 0), (2, 1)])
sage: g.num_ground_vertices()
2
```

**out\_degrees()**

Return the tuple of out-degrees of vertices of this graph.

EXAMPLES:

```
sage: g = FormalityGraph(2, 1, [(2, 0), (2, 1)])
sage: g.out_degrees()
(0, 0, 2)
sage: h = FormalityGraph(3, 2, [(3, 0), (3, 1), (3, 2), (4, 3), (4, 2)])
sage: h.out_degrees()
(0, 0, 0, 3, 2)
```

**plot(\*\*options)**

Return a plot of this graph.

EXAMPLES:

```
sage: g = FormalityGraph(2, 1, [(2, 0), (2, 1)])
sage: g.plot()
Graphics object consisting of 6 graphics primitives
```

**reabeled**(*relabeling*)

Return a vertex relabeling of this graph.

EXAMPLES:

```
sage: g = FormalityGraph(2, 1, [(2, 0), (2, 1)])
sage: g.reabeled({0: 1, 1: 0, 2: 2})
FormalityGraph(2, 1, [(2, 1), (2, 0)])
```

**set\_pos**(*new\_pos*)

Set the positions of vertices in this graph (used for plotting).

EXAMPLES:

```
sage: g = FormalityGraph(2, 1, [(2, 0), (2, 1)])
sage: g.set_pos({0: (0.0, 0.0), 1: (1.0, 0.0), 2: (0.5, 1.0)})
sage: g.get_pos()
{0: (0.0000000000000000, 0.0000000000000000),
 1: (1.0000000000000000, 0.0000000000000000),
 2: (0.5000000000000000, 1.0000000000000000)}
```

**show**(*\*\*options*)

Show this graph.

EXAMPLES:

```
sage: g = FormalityGraph(2, 1, [(2, 0), (2, 1)])
sage: g.show()
```

## 5.2 Formality graph basis

Formality graph basis

**class** gcaops.graph.formality\_graph\_basis.**FormalityGraphBasis**

Bases: *gcaops.graph.graph\_basis.GraphBasis*

Basis of a module spanned by formality graphs.

A basis consists of keys (gv, av, e, index, ...) where (gv, av, e, index) identifies the isomorphism class of the graph.

**graph\_class**

alias of *gcaops.graph.formality\_graph.FormalityGraph*

**class** gcaops.graph.formality\_graph\_basis.**FormalityGraphComplexBasis**(*positive\_differential\_order=None, connected=None, loops=None*)

Bases: *gcaops.graph.formality\_graph\_basis.FormalityGraphBasis*

Basis consisting of representatives of isomorphism classes of formality graphs with no automorphisms that induce an odd permutation on edges.

**cardinality**(*num\_ground\_vertices, num\_aerial\_vertices, num\_edges*)

Return the number of graphs in this basis with the given num\_ground\_vertices, num\_aerial\_vertices and num\_edges.

**graph\_properties()**

Return a dictionary containing the properties of the graphs in this basis.

**graph\_to\_key(*graph*)**

Return a tuple consisting of the key in this basis and the sign factor such that *graph* equals the sign times the graph identified by the key.

INPUT:

- *graph* – a *FormalityGraph*

OUTPUT:

Either (None, 1) if the input *graph* is not in the span of the basis, or a tuple consisting of a key and a sign, where a key is a tuple consisting of the number of ground vertices, the number of aerial vertices, the number of edges, and the index of the graph in the list.

**graphs(*num\_ground\_vertices*, *num\_aerial\_vertices*, *num\_edges*)**

Return the list of graphs in this basis with the given *num\_ground\_vertices*, *num\_aerial\_vertices* and *num\_edges*.

**key\_to\_graph(*key*)**

Return a tuple consisting of a *FormalityGraph* and the sign factor such that the sign times the graph equals the graph identified by the key.

INPUT:

- *key* – a key in this basis

OUTPUT:

Either (None, 1) if the input *key* is not in the basis, or a tuple consisting of a *FormalityGraph* and a sign which is always +1.

**class** gcaops.graph.formality\_graph\_basis.**FormalityGraphComplexBasis\_lazy**(*positive\_differential\_order=None*,  
*con-*  
*nected=None*,  
*loops=None*)

Bases: *gcaops.graph.formality\_graph\_basis.FormalityGraphComplexBasis*

Basis consisting of representatives of isomorphism classes of formality graphs with no automorphisms that induce an odd permutation on edges.

**graph\_to\_key(*graph*)**

Return a tuple consisting of the key in this basis and the sign factor such that *graph* equals the sign times the graph identified by the key.

INPUT:

- *graph* – a *FormalityGraph*

OUTPUT:

Either (None, 1) if the input *graph* is not in the span of the basis, or a tuple consisting of a key and a sign, where a key is a tuple containing the number of ground vertices, the number of aerial vertices, and the number of edges, followed by all the edges in the graph.

**key\_to\_graph(*key*)**

Return a tuple consisting of a *FormalityGraph* and the sign factor such that the sign times the graph equals the graph identified by the key.

INPUT:

- *key* – a key in this basis

OUTPUT:

Either (None, 1) if the input `key` is not in the basis, or a tuple consisting of a *FormalityGraph* and a sign which is always +1.

**class** gcaops.graph.formality\_graph\_basis.**FormalityGraphOperadBasis**(*positive\_differential\_order=None, connected=None, loops=None*)

Bases: *gcaops.graph.formality\_graph\_basis.FormalityGraphBasis*

Basis consisting of labeled formality graphs with no automorphisms that induce an odd permutation on edges

**graph\_properties**()

Return a dictionary containing the properties of the graphs in this basis.

**graph\_to\_key**(*graph*)

Return a tuple consisting of the key in this basis and the sign factor such that `graph` equals the sign times the graph identified by the key.

INPUT:

- `graph` – a *FormalityGraph*

OUTPUT:

Either (None, 1) if the input `graph` is not in the span of the basis, or a tuple consisting of a key and a sign, where a key is a tuple consisting of the number of ground vertices, the number of aerial vertices, the number of edges, the index of the graph in the list, followed by a permutation of vertices.

**key\_to\_graph**(*key*)

Return a tuple consisting of a *FormalityGraph* and the sign factor such that the sign times the graph equals the graph identified by the key.

INPUT:

- `key` – a key in this basis

OUTPUT:

Either (None, 1) if the input `key` is not in the basis, or a tuple consisting of a *FormalityGraph* and a sign.

**class** gcaops.graph.formality\_graph\_basis.**KontsevichGraphBasis**(*positive\_differential\_order=None, connected=None, loops=None, mod\_ground\_permutations=False, max\_aerial\_in\_degree=None*)

Bases: *gcaops.graph.formality\_graph\_basis.QuantizationGraphBasis*

Basis consisting of representatives of isomorphism classes of Kontsevich graphs (built of wedges) with no automorphisms that induce an odd permutation on edges.

**flipping\_weight\_relations**(*num\_ground\_vertices, num\_aerial\_vertices*)

Return a matrix in which each row represents a linear relation between the weights of the graphs in the basis at the given bi-grading.

The relations are those induced by a single orientation-reversing coordinate change on the upper half-plane, applied to each factor of the configuration space.

ASSUMPTION:

Assumes `num_ground_vertices == 2`, and assumes that the weights are real-valued (e.g. defined using the harmonic propagators).

```
class gcaops.graph.formality_graph_basis.LeibnizGraphBasis(positive_differential_order=None,
                                                         connected=None, loops=None,
                                                         mod_ground_permutations=False,
                                                         max_aerial_in_degree=None)
```

Bases: `gcaops.graph.formality_graph_basis.QuantizationGraphBasis`

Basis consisting of representatives of isomorphism classes of Leibniz graphs (built of one tripod wedges) with no automorphisms that induce an odd permutation on edges.

## 5.3 Formality graph vector

Formality graph vector

```
class gcaops.graph.formality_graph_vector.FormalityGraphModule
```

Bases: `gcaops.graph.graph_vector.GraphModule`

Module spanned by formality graphs.

```
__call__(arg)
```

Return the result of converting `arg` into an element of this module.

```
element_from_kgs_encoding(kgs_encoding, hbar=1)
```

Return the linear combination of Kontsevich graphs specified by an encoding, as an element of this module.

INPUT:

- `kgs_encoding` – a string, containing an encoding of a graph series expansion as used in Buring’s `kontsevich_graph_series-cpp` program
- `hbar` (default: 1) – an element of the base ring, to be used as the graph series expansion parameter

```
class gcaops.graph.formality_graph_vector.FormalityGraphModule_dict(base_ring,
                                                                    graph_basis)
Bases:      gcaops.graph.formality_graph_vector.FormalityGraphModule,      gcaops.graph.
graph_vector_dict.GraphModule_dict
```

Module spanned by formality graphs (with elements stored as dictionaries).

```
__init__(base_ring, graph_basis)
```

Initialize this formality graph module.

INPUT:

- `base_ring` – a ring, to be used as the ring of coefficients
- `graph_basis` – a `FormalityGraphBasis`

```
class gcaops.graph.formality_graph_vector.FormalityGraphModule_vector(base_ring,
                                                                    graph_basis, vec-
                                                                    tor_constructor,
                                                                    ma-
                                                                    trix_constructor,
                                                                    sparse=True)
Bases:      gcaops.graph.formality_graph_vector.FormalityGraphModule,      gcaops.graph.
graph_vector_vector.GraphModule_vector
```

Module spanned by formality graphs (with elements stored as dictionaries of vectors).

```
__init__(base_ring, graph_basis, vector_constructor, matrix_constructor, sparse=True)
```

Initialize this formality graph module.

INPUT:

- `base_ring` – a ring, to be used as the ring of coefficients
- `graph_basis` – a *FormalityGraphBasis*
- `vector_constructor` – constructor of (sparse) vectors
- `matrix_constructor` – constructor of (sparse) matrices
- `sparse` – (default: `True`) a boolean, passed along to both constructors as a keyword argument

**class** `gcaops.graph.formality_graph_vector.FormalityGraphVector`

Bases: `gcaops.graph.graph_vector.GraphVector`

Vector representing a linear combination of formality graphs.

**attach\_to\_ground**(*degrees*)

Return the non-aerial graph vector that represents the polydifferential operator which results from evaluating this graph vector at multi-vectors of the given degrees.

ASSUMPTIONS:

Assumes that this graph vector is aerial.

**differential\_orders**()

Return an iterator over the tuples of in-degrees of ground vertices of graphs in this graph vector.

**filter**(*max\_aerial\_in\_degree=None*)

Return the graph vector which is the summand of this graph vector containing exactly those graphs that pass the filter.

**ground\_skew\_symmetrization**()

Return the skew-symmetrization (or anti-symmetrization) of this graph vector with respect to the ground vertices.

ASSUMPTIONS:

Assumes all graphs in this graph vector have the same number of ground vertices.

**ground\_symmetrization**()

Return the symmetrization of this graph vector with respect to the ground vertices.

ASSUMPTIONS:

Assumes all graphs in this graph vector have the same number of ground vertices.

**insertion**(*position, other, \*\*kwargs*)

Return the insertion of `other` into this graph vector at the vertex `position`.

**is\_aerial**()

Return `True` if this graph vector is aerial, and `False` otherwise.

**kgs\_encoding**()

Return an encoding of this graph vector for use in Buring's `kontsevich_graph_series-cpp` program.

ASSUMPTIONS:

Assumes all graphs in this graph vector are built of wedges (i.e. with each aerial vertex having out-degree two).

**nground**()

Return the number of ground vertices in each graph in this graph vector.

ASSUMPTIONS:

Assumes all graphs in this graph vector have the same number of ground vertices.

**part\_of\_differential\_order**(*diff\_order*)

Return the graph vector which is the summand of this graph vector containing only graphs such that the in-degrees of the ground vertices are *diff\_order*.

**set\_aerial**(*is\_aerial=True*)

Set this graph vector to be aerial if *is\_aerial* is True, respectively not aerial if *is\_aerial* is False.

**class** gcaops.graph.formality\_graph\_vector.**FormalityGraphVector\_dict**(*parent*, *vector*,  
*is\_aerial=False*)

Bases: [gcaops.graph.formality\\_graph\\_vector.FormalityGraphVector](#), [gcaops.graph.graph\\_vector\\_dict.GraphVector\\_dict](#)

Vector representing a linear combination of formality graphs (stored as a dictionary).

**\_\_init\_\_**(*parent*, *vector*, *is\_aerial=False*)

Initialize this formality graph vector.

INPUT:

- *parent* – a [FormalityGraphModule](#)
- *vector* – a dictionary, representing a sparse vector of coefficients with respect to the basis of *parent*
- *is\_aerial* – (default: False) a boolean, if True then this graph vector will be aerial

**filter**(*max\_aerial\_in\_degree=None*)

Return the graph vector which is the summand of this graph vector containing exactly those graphs that pass the filter.

**is\_aerial**()

Return True if this graph vector is aerial, and False otherwise.

**nedges**()

Return the number of edges in each graph in this graph vector.

ASSUMPTIONS:

Assumes all graphs in this graph vector have the same number of edges.

**nground**()

Return the number of ground vertices in each graph in this graph vector.

ASSUMPTIONS:

Assumes all graphs in this graph vector have the same number of ground vertices.

**nvertices**()

Return the number of vertices in each graph in this graph vector.

ASSUMPTIONS:

Assumes all graphs in this graph vector have the same number of vertices.

**set\_aerial**(*is\_aerial=True*)

Set this graph vector to be aerial if *is\_aerial* is True, respectively not aerial if *is\_aerial* is False.

**class** gcaops.graph.formality\_graph\_vector.**FormalityGraphVector\_vector**(*parent*, *vectors*,  
*is\_aerial=False*)

Bases: [gcaops.graph.formality\\_graph\\_vector.FormalityGraphVector](#), [gcaops.graph.graph\\_vector\\_vector.GraphVector\\_vector](#)

Vector representing a linear combination of formality graphs (stored as a dictionary of vectors).

**\_\_init\_\_**(*parent*, *vectors*, *is\_aerial=False*)

Initialize this graph vector.

INPUT:

- `parent` – a *FormalityGraphModule*
- `vectors` – a dictionary, mapping tri-gradings to (sparse) vectors of coefficients with respect to the basis of `parent`
- `is_aerial` – (default: False) a boolean, if True then this graph vector will be aerial

**filter**(*max\_aerial\_in\_degree=None*)

Return the graph vector which is the summand of this graph vector containing exactly those graphs that pass the filter.

**is\_aerial**()

Return True if this graph vector is aerial, and False otherwise.

**nedges**()

Return the number of edges in each graph in this graph vector.

ASSUMPTIONS:

Assumes all graphs in this graph vector have the same number of edges.

**nground**()

Return the number of ground vertices in each graph in this graph vector.

ASSUMPTIONS:

Assumes all graphs in this graph vector have the same number of ground vertices.

**nvertices**()

Return the number of vertices in each graph in this graph vector.

ASSUMPTIONS:

Assumes all graphs in this graph vector have the same number of vertices.

**set\_aerial**(*is\_aerial=True*)

Set this graph vector to be aerial if `is_aerial` is True, respectively not aerial if `is_aerial` is False.

## 5.4 Formality graph operad

Formality graph operad

`gcaops.graph.formality_graph_operad.FormalityGraphOperad`(*base\_ring*)

Return the operad of formality graphs over the given `base_ring`.

**class** `gcaops.graph.formality_graph_operad.FormalityGraphOperad_dict`(*base\_ring*)

Bases: `gcaops.graph.formality_graph_vector.FormalityGraphModule_dict`

Operad of formality graphs (with elements stored as dictionaries).

**\_\_init\_\_**(*base\_ring*)

Initialize this graph operad.

**\_\_repr\_\_**()

Return a string representation of this graph operad.

**class** `gcaops.graph.formality_graph_operad.FormalityGraphOperation_dict`(*parent, vector*)

Bases: `gcaops.graph.formality_graph_vector.FormalityGraphVector_dict`

Element of a *FormalityGraphOperad\_dict* (stored as a dictionary).



**\_\_init\_\_**(*parent, vector*)  
Initialize this graph operation.

## 5.5 Formality graph complex

Formality graph complex

**class** `gcaops.graph.formality_graph_complex.FormalityGraphCochain`

Bases: `gcaops.graph.graph_complex.GraphCochain`, `gcaops.graph.formality_graph_vector.FormalityGraphVector`

Cochain of a `FormalityGraphComplex_`.

**bracket**(*other, \*\*kwargs*)  
Return the graph Gerstenhaber bracket of this graph cochain with *other*.

**gerstenhaber\_bracket**(*other, \*\*kwargs*)  
Return the graph Gerstenhaber bracket of this graph cochain with *other*.

**schouten\_bracket**(*other, \*\*kwargs*)  
Return the graph analogue of the Schouten bracket (or Schouten-Nijenhuis bracket) of this graph cochain with *other*.

ASSUMPTIONS:

Assumes that this graph vector and *other* both are skew-symmetric and have differential order equal to one on each ground vertex.

**class** `gcaops.graph.formality_graph_complex.FormalityGraphCochain_dict`(*parent, vector*)

Bases: `gcaops.graph.formality_graph_complex.FormalityGraphCochain`, `gcaops.graph.formality_graph_vector.FormalityGraphVector_dict`

Cochain of a `FormalityGraphComplex_dict` (stored as a dictionary).

**\_\_init\_\_**(*parent, vector*)  
Initialize this graph cochain.

**differential**()  
Return the Hochschild differential of this graph cochain.

**hochschild\_differential**()  
Return the Hochschild differential of this graph cochain.

**class** `gcaops.graph.formality_graph_complex.FormalityGraphCochain_vector`(*parent, vector*)

Bases: `gcaops.graph.formality_graph_complex.FormalityGraphCochain`, `gcaops.graph.formality_graph_vector.FormalityGraphVector_vector`

Cochain of a `FormalityGraphComplex_vector` (stored as a dictionary of vectors).

**\_\_init\_\_**(*parent, vector*)  
Initialize this graph cochain.

**differential**(*use\_cache=False*)  
Return the graph differential of this graph cochain.

**hochschild\_differential**(*use\_cache=False*)  
Return the graph differential of this graph cochain.

`gcaops.graph.formality_graph_complex.FormalityGraphComplex`(*base\_ring*, *connected=None*, *loops=None*, *implementation='dict'*, *lazy=False*, *vector\_constructor=None*, *matrix\_constructor=None*, *sparse=True*)

Return the Formality graph complex over *base\_ring* with the given properties.

**class** `gcaops.graph.formality_graph_complex.FormalityGraphComplex_`  
 Bases: `gcaops.graph.graph_complex.GraphComplex`, `gcaops.graph.formality_graph_vector.FormalityGraphModule`

Formality graph complex.

**class** `gcaops.graph.formality_graph_complex.FormalityGraphComplex_dict`(*base\_ring*, *connected=None*, *loops=None*, *lazy=False*)  
 Bases: `gcaops.graph.formality_graph_complex.FormalityGraphComplex_`, `gcaops.graph.formality_graph_vector.FormalityGraphModule_dict`

Formality graph complex (with elements stored as dictionaries).

**\_\_init\_\_**(*base\_ring*, *connected=None*, *loops=None*, *lazy=False*)  
 Initialize this graph complex.

**\_\_repr\_\_**()  
 Return a string representation of this graph complex.

**class** `gcaops.graph.formality_graph_complex.FormalityGraphComplex_vector`(*base\_ring*, *vector\_constructor*, *matrix\_constructor*, *sparse=True*, *connected=None*, *loops=None*)  
 Bases: `gcaops.graph.formality_graph_complex.FormalityGraphComplex_`, `gcaops.graph.formality_graph_vector.FormalityGraphModule_vector`

Formality graph complex (with elements stored as dictionaries of vectors).

**\_\_init\_\_**(*base\_ring*, *vector\_constructor*, *matrix\_constructor*, *sparse=True*, *connected=None*, *loops=None*)  
 Initialize this graph complex.

**\_\_repr\_\_**()  
 Return a string representation of this graph complex.

**cohomology\_basis**(*ground\_vertices*, *aerial\_vertices*, *edges*)  
 Return a basis of the cohomology in the given tri-grading (*ground\_vertices*, *aerial\_vertices*, *edges*).

## 5.6 Formality graph operator

Formality graph operator

**class** gcaops.graph.formality\_graph\_operator.**FormalityGraphOperator**(*domain*, *codomain*,  
*graph\_vector*)

Bases: object

A homogeneous n-ary multi-linear operator on a *SuperfunctionAlgebra* with values in a *PolydifferentialOperatorAlgebra*, defined by a *FormalityGraphVector*.

**\_\_call\_\_**(\*args)

Return the evaluation of this operator at args.

**\_\_init\_\_**(*domain*, *codomain*, *graph\_vector*)

Initialize this operator.

**\_\_repr\_\_**()

Return a string representation of this operator.

**codomain**()

Return the codomain of this operator.

**domain**()

Return the domain of this operator.

**value\_at\_copies\_of**(*arg*)

Return the evaluation of this operator at copies of arg.

**class** gcaops.graph.formality\_graph\_operator.**FormalityGraphSymmetricOperator**

Bases: object

A homogeneous n-ary multi-linear symmetric operator on a *SuperfunctionAlgebra* with values in a *PolydifferentialOperatorAlgebra*, defined by a *FormalityGraphVector*.

**\_\_call\_\_**(\*args)

Return the evaluation of this operator at args.

gcaops.graph.formality\_graph\_operator.**formality\_graph\_operator**(*graph\_vector*, *domain*,  
*codomain*)

Factory.



## GRAPH CACHE

## 6.1 Graph file view

Graph file view

```
class gcaops.graph.graph_file.DirectedGraphFileView(filename, num_vertices, num_edges)
```

Bases: *gcaops.graph.graph\_file.GraphFileView*

Directed graph database file view

```
class gcaops.graph.graph_file.FormalityGraphFileView(filename, num_ground_vertices,  
num_aerial_vertices, num_edges)
```

Bases: *gcaops.graph.graph\_file.GraphFileView*

Formality graph database file view

```
__init__(filename, num_ground_vertices, num_aerial_vertices, num_edges)
```

Initialize this formality graph database file view.

INPUT:

- `filename` – a string, the path to an SQLite database file (the file will be created if it does not yet exist)
- `num_ground_vertices` – a natural number, the number of ground vertices in each graph
- `num_aerial_vertices` – a natural number, the number of aerial vertices in each graph
- `num_edges` – a natural number, the number of edges in each graph

```
class gcaops.graph.graph_file.GraphFileView(filename, num_vertices, num_edges)
```

Bases: `abc.ABC`

Graph database file view

```
__getitem__(index)
```

Return the graph at the given index in this database file.

```
__getstate__()
```

Return the state of this object as a dictionary (used for pickling).

```
__init__(filename, num_vertices, num_edges)
```

Initialize this graph database file view.

INPUT:

- `filename` – a string, the path to an SQLite database file (the file will be created if it does not yet exist)
- `num_vertices` – a natural number, the number of vertices in each graph
- `num_edges` – a natural number, the number of edges in each graph

**\_\_iter\_\_()**

Return an iterator over the graphs in this database file.

**\_\_len\_\_()**

Return the number of graphs in this database file.

**\_\_setstate\_\_(state\_dict)**

Set the state of this object from a dictionary (used for pickling).

**append(g)**

Insert the given graph into this database file.

**commit()**

Commit any changes to this database file.

**index(g)**

Return the index of the given graph in this database file.

**class gcaops.graph.graph\_file.UndirectedGraphFileView(filename, num\_vertices, num\_edges)**

Bases: [gcaops.graph.graph\\_file.GraphFileView](#)

Undirected graph database file view

**class gcaops.graph.graph\_file.UndirectedToDirectedGraphFileView(filename)**

Bases: object

Undirected to directed graph database file view

**\_\_getstate\_\_()**

Return the state of this object as a dictionary (used for pickling).

**\_\_init\_\_(filename)**

Initialize this “undirected to directed graph” database file view.

INPUT:

- `filename` – a string, the path to an SQLite database file (the file will be created if it does not yet exist)

**\_\_iter\_\_()**

Return an iterator over the rows in the “undirected to directed graph” database file.

**\_\_len\_\_()**

Return the number of rows in the “undirected to directed graph” database file.

**\_\_setstate\_\_(state\_dict)**

Set the state of this object from a dictionary (used for pickling).

**append(row)**

Insert a row into the “undirected to directed graph” database file.

**commit()**

Commit any changes to this database file.

**undirected\_to\_directed\_coeffs(undirected\_graph\_idx)**

Return an iterator over the (`directed_graph_idx`, `coefficient`) tuples related to the undirected graph with the given index.

## 6.2 Graph cache

Graph cache

**class** `gcaops.graph.graph_cache.DirectedGraphCache`(*undirected\_graph\_cache*)

Bases: `gcaops.graph.graph_cache.GraphCache`

Directed graph cache

**\_\_init\_\_**(*undirected\_graph\_cache*)

Initialize this directed graph cache.

INPUT:

- *undirected\_graph\_cache* – an `UndirectedGraphCache`

**canonicalize\_graph**(*graph*)

Return a tuple consisting the normal form of the graph and the sign factor relating the input graph to the normal form.

**file\_view**

alias of `gcaops.graph.graph_file.DirectedGraphFileView`

**graphs**(*bi\_grading*, *connected=False*, *biconnected=False*, *min\_degree=0*, *loops=True*, *has\_odd\_automorphism=True*)

Return a view (a list or a `GraphFileView`) of the graphs in the cache with the given options.

**class** `gcaops.graph.graph_cache.FormalityGraphCache`

Bases: `gcaops.graph.graph_cache.GraphCache`

Formality graph cache

**canonicalize\_graph**(*graph*)

Return a tuple consisting the normal form of the graph, an isomorphism from the normal form to the input graph, and the sign of the induced permutation on edges.

**file\_view**

alias of `gcaops.graph.graph_file.FormalityGraphFileView`

**graphs**(*tri\_grading*, *connected=None*, *max\_out\_degree=None*, *num\_verts\_of\_max\_out\_degree=None*, *sorted\_out\_degrees=None*, *max\_aerial\_in\_degree=None*, *loops=None*, *prime=None*, *has\_odd\_automorphism=None*, *positive\_differential\_order=None*, *mod\_ground\_permutations=False*)

Return a view (a list or a `GraphFileView`) of the graphs in the cache with the given options.

**class** `gcaops.graph.graph_cache.GraphCache`

Bases: `abc.ABC`

Graph cache

**canonicalize\_graph**(*graph*)

Return a tuple consisting the normal form of the graph, followed by data relating the input graph to the normal form (e.g. a sign factor).

**graphs**(*grading*, *\*\*options*)

Return a view (e.g. a list or a `GraphFileView`) of the graphs in the cache with the given options.

**class** `gcaops.graph.graph_cache.UndirectedGraphCache`

Bases: `gcaops.graph.graph_cache.GraphCache`

Undirected graph cache

**canonicalize\_graph**(*graph*)

Return a tuple consisting the normal form of the graph and the sign factor relating the input graph to the normal form.

**file\_view**

alias of *gcaops.graph.graph\_file.UndirectedGraphFileView*

**graphs**(*bi\_grading*, *connected=False*, *biconnected=False*, *min\_degree=0*,  
*has\_odd\_automorphism=True*)

Return a view (a list or a *GraphFileView*) of the graphs in the cache with the given options.



## PYTHON MODULE INDEX

### g

- `gcaops.algebra.differential_polynomial_ring`, 588
- `gcaops.algebra.differential_polynomial_solver`, 589
- `gcaops.algebra.polydifferential_operator`, 584
- `gcaops.algebra.superfunction_algebra`, 579
- `gcaops.algebra.superfunction_algebra_operation`, 582
- `gcaops.algebra.tensor_product`, 587
- `gcaops.graph.directed_graph`, 609
- `gcaops.graph.directed_graph_basis`, 610
- `gcaops.graph.directed_graph_complex`, 614
- `gcaops.graph.directed_graph_operad`, 614
- `gcaops.graph.directed_graph_vector`, 612
- `gcaops.graph.formality_graph`, 617
- `gcaops.graph.formality_graph_basis`, 622
- `gcaops.graph.formality_graph_complex`, 629
- `gcaops.graph.formality_graph_operad`, 628
- `gcaops.graph.formality_graph_operator`, 631
- `gcaops.graph.formality_graph_vector`, 625
- `gcaops.graph.graph_basis`, 591
- `gcaops.graph.graph_cache`, 635
- `gcaops.graph.graph_complex`, 593
- `gcaops.graph.graph_file`, 633
- `gcaops.graph.graph_vector`, 591
- `gcaops.graph.graph_vector_dict`, 595
- `gcaops.graph.graph_vector_vector`, 593
- `gcaops.graph.undirected_graph`, 599
- `gcaops.graph.undirected_graph_basis`, 601
- `gcaops.graph.undirected_graph_complex`, 605
- `gcaops.graph.undirected_graph_operad`, 605
- `gcaops.graph.undirected_graph_vector`, 603



## Symbols

- `__add__` () (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperator* method), 584
- `__add__` () (*gcaops.algebra.superfunction\_algebra.Superfunction* method), 579
- `__add__` () (*gcaops.graph.graph\_vector.GraphVector* method), 592
- `__add__` () (*gcaops.graph.graph\_vector\_dict.GraphVector\_dict* method), 596
- `__add__` () (*gcaops.graph.graph\_vector\_vector.GraphVector\_vector* method), 594
- `__call__` () (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperatorAlgebra* method), 586
- `__call__` () (*gcaops.algebra.superfunction\_algebra.SuperfunctionAlgebra* method), 581
- `__call__` () (*gcaops.graph.directed\_graph\_vector.DirectedGraphModule* method), 612
- `__call__` () (*gcaops.graph.formality\_graph\_operator.FormalityGraphOperator* method), 631
- `__call__` () (*gcaops.graph.formality\_graph\_operator.FormalityGraphSymmetricOperator* method), 631
- `__call__` () (*gcaops.graph.formality\_graph\_vector.FormalityGraphModule* method), 625
- `__call__` () (*gcaops.graph.graph\_vector.GraphModule* method), 591
- `__call__` () (*gcaops.graph.graph\_vector\_dict.GraphModule\_dict* method), 595
- `__call__` () (*gcaops.graph.graph\_vector\_vector.GraphModule\_vector* method), 593
- `__eq__` () (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperator* method), 584
- `__eq__` () (*gcaops.algebra.superfunction\_algebra.Superfunction* method), 579
- `__eq__` () (*gcaops.graph.graph\_vector.GraphVector* method), 592
- `__eq__` () (*gcaops.graph.graph\_vector\_dict.GraphVector\_dict* method), 596
- `__eq__` () (*gcaops.graph.graph\_vector\_vector.GraphVector\_vector* method), 594
- `__eq__` () (*gcaops.graph.undirected\_graph.UndirectedGraph* method), 599
- `__getitem__` () (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperator* method), 584
- `__getitem__` () (*gcaops.algebra.superfunction\_algebra.Superfunction* method), 579
- `__getitem__` () (*gcaops.graph.graph\_file.GraphFileView* method), 633
- `__getstate__` () (*gcaops.graph.graph\_file.GraphFileView* method), 633
- `__getstate__` () (*gcaops.graph.graph\_file.UndirectedToDirectedGraphFileView* method), 634
- `__init__` () (*gcaops.algebra.differential\_polynomial\_ring.DifferentialPolynomialRing* method), 588
- `__init__` () (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperator* method), 584
- `__init__` () (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperatorAlgebra* method), 586
- `__init__` () (*gcaops.algebra.superfunction\_algebra.Superfunction* method), 579
- `__init__` () (*gcaops.algebra.superfunction\_algebra.SuperfunctionAlgebra* method), 581
- `__init__` () (*gcaops.graph.directed\_graph\_complex.DirectedGraphCochain\_dict* method), 615
- `__init__` () (*gcaops.graph.directed\_graph\_complex.DirectedGraphCochain\_vector* method), 615
- `__init__` () (*gcaops.graph.directed\_graph\_complex.DirectedGraphComplex\_dict* method), 615
- `__init__` () (*gcaops.graph.directed\_graph\_complex.DirectedGraphComplex\_vector* method), 616
- `__init__` () (*gcaops.graph.directed\_graph\_operad.DirectedGraphOperad\_dict* method), 614
- `__init__` () (*gcaops.graph.directed\_graph\_operad.DirectedGraphOperation\_dict* method), 614

\_\_init\_\_ () (*gcaops.graph.directed\_graph\_vector.DirectedGraphModule\_dict* method), 612  
 \_\_init\_\_ () (*gcaops.graph.directed\_graph\_vector.DirectedGraphModule\_vector* method), 612  
 \_\_init\_\_ () (*gcaops.graph.directed\_graph\_vector.DirectedGraphVector\_dict* method), 613  
 \_\_init\_\_ () (*gcaops.graph.directed\_graph\_vector.DirectedGraphVector\_vector* method), 613  
 \_\_init\_\_ () (*gcaops.graph.formality\_graph\_complex.FormalityGraphCochain\_dict* method), 629  
 \_\_init\_\_ () (*gcaops.graph.formality\_graph\_complex.FormalityGraphCochain\_vector* method), 629  
 \_\_init\_\_ () (*gcaops.graph.formality\_graph\_complex.FormalityGraphComplex\_dict* method), 630  
 \_\_init\_\_ () (*gcaops.graph.formality\_graph\_complex.FormalityGraphComplex\_vector* method), 630  
 \_\_init\_\_ () (*gcaops.graph.formality\_graph\_operad.FormalityGraphOperad\_dict* method), 628  
 \_\_init\_\_ () (*gcaops.graph.formality\_graph\_operad.FormalityGraphOperation\_dict* method), 628  
 \_\_init\_\_ () (*gcaops.graph.formality\_graph\_operator.FormalityGraphOperator* method), 631  
 \_\_init\_\_ () (*gcaops.graph.formality\_graph\_vector.FormalityGraphModule\_dict* method), 625  
 \_\_init\_\_ () (*gcaops.graph.formality\_graph\_vector.FormalityGraphModule\_vector* method), 625  
 \_\_init\_\_ () (*gcaops.graph.formality\_graph\_vector.FormalityGraphVector\_dict* method), 627  
 \_\_init\_\_ () (*gcaops.graph.formality\_graph\_vector.FormalityGraphVector\_vector* method), 627  
 \_\_init\_\_ () (*gcaops.graph.graph\_cache.DirectedGraphCache* method), 635  
 \_\_init\_\_ () (*gcaops.graph.graph\_file.FormalityGraphFileView* method), 633  
 \_\_init\_\_ () (*gcaops.graph.graph\_file.GraphFileView* method), 633  
 \_\_init\_\_ () (*gcaops.graph.graph\_file.UndirectedToDirectedGraphFileView* method), 634  
 \_\_init\_\_ () (*gcaops.graph.graph\_vector\_dict.GraphModule\_dict* method), 595  
 \_\_init\_\_ () (*gcaops.graph.graph\_vector\_dict.GraphVector\_dict* method), 596  
 \_\_init\_\_ () (*gcaops.graph.graph\_vector\_vector.GraphModule\_vector* method), 593  
 \_\_init\_\_ () (*gcaops.graph.graph\_vector\_vector.GraphVector\_vector* method), 594  
 \_\_init\_\_ () (*gcaops.graph.undirected\_graph.UndirectedGraph* method), 599  
 \_\_init\_\_ () (*gcaops.graph.undirected\_graph\_complex.UndirectedGraphCochain\_dict* method), 605  
 \_\_init\_\_ () (*gcaops.graph.undirected\_graph\_complex.UndirectedGraphCochain\_vector* method), 605  
 \_\_init\_\_ () (*gcaops.graph.undirected\_graph\_complex.UndirectedGraphComplex\_dict* method), 606  
 \_\_init\_\_ () (*gcaops.graph.undirected\_graph\_complex.UndirectedGraphComplex\_vector* method), 606  
 \_\_init\_\_ () (*gcaops.graph.undirected\_graph\_operad.UndirectedGraphOperad\_dict* method), 605  
 \_\_init\_\_ () (*gcaops.graph.undirected\_graph\_operad.UndirectedGraphOperation\_dict* method), 605  
 \_\_init\_\_ () (*gcaops.graph.undirected\_graph\_vector.UndirectedGraphModule\_dict* method), 603  
 \_\_init\_\_ () (*gcaops.graph.undirected\_graph\_vector.UndirectedGraphModule\_vector* method), 603  
 \_\_init\_\_ () (*gcaops.graph.undirected\_graph\_vector.UndirectedGraphVector\_dict* method), 604  
 \_\_init\_\_ () (*gcaops.graph.undirected\_graph\_vector.UndirectedGraphVector\_vector* method), 604  
 \_\_iter\_\_ () (*gcaops.graph.graph\_file.GraphFileView* method), 633  
 \_\_iter\_\_ () (*gcaops.graph.graph\_file.UndirectedToDirectedGraphFileView* method), 634  
 \_\_iter\_\_ () (*gcaops.graph.graph\_vector.GraphVector* method), 592  
 \_\_iter\_\_ () (*gcaops.graph.graph\_vector\_dict.GraphVector\_dict* method), 596  
 \_\_iter\_\_ () (*gcaops.graph.graph\_vector\_vector.GraphVector\_vector* method), 594  
 \_\_len\_\_ () (*gcaops.graph.graph\_file.GraphFileView* method), 634  
 \_\_len\_\_ () (*gcaops.graph.graph\_file.UndirectedToDirectedGraphFileView* method), 634  
 \_\_len\_\_ () (*gcaops.graph.graph\_vector.GraphVector* method), 592  
 \_\_len\_\_ () (*gcaops.graph.graph\_vector\_dict.GraphVector\_dict* method), 596  
 \_\_len\_\_ () (*gcaops.graph.graph\_vector\_vector.GraphVector\_vector* method), 594  
 \_\_len\_\_ () (*gcaops.graph.undirected\_graph.UndirectedGraph* method), 599  
 \_\_mul\_\_ () (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperator* method), 584  
 \_\_mul\_\_ () (*gcaops.algebra.superfunction\_algebra.Superfunction* method), 579  
 \_\_mul\_\_ () (*gcaops.graph.graph\_vector.GraphVector* method), 592  
 \_\_mul\_\_ () (*gcaops.graph.graph\_vector\_dict.GraphVector\_dict* method), 596  
 \_\_mul\_\_ () (*gcaops.graph.graph\_vector\_vector.GraphVector\_vector* method), 594

---

`__neg__` () (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperator* method), 584  
`__neg__` () (*gcaops.algebra.superfunction\_algebra.Superfunction* method), 579  
`__neg__` () (*gcaops.graph.graph\_vector.GraphVector* method), 592  
`__neg__` () (*gcaops.graph.graph\_vector\_dict.GraphVector\_dict* method), 596  
`__neg__` () (*gcaops.graph.graph\_vector\_vector.GraphVector\_vector* method), 594  
`__pos__` () (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperator* method), 584  
`__pos__` () (*gcaops.algebra.superfunction\_algebra.Superfunction* method), 579  
`__pos__` () (*gcaops.graph.graph\_vector.GraphVector* method), 592  
`__pos__` () (*gcaops.graph.graph\_vector\_dict.GraphVector\_dict* method), 596  
`__pos__` () (*gcaops.graph.graph\_vector\_vector.GraphVector\_vector* method), 594  
`__pow__` () (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperator* method), 584  
`__pow__` () (*gcaops.algebra.superfunction\_algebra.Superfunction* method), 579  
`__radd__` () (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperator* method), 585  
`__radd__` () (*gcaops.algebra.superfunction\_algebra.Superfunction* method), 579  
`__radd__` () (*gcaops.graph.graph\_vector.GraphVector* method), 592  
`__radd__` () (*gcaops.graph.graph\_vector\_dict.GraphVector\_dict* method), 596  
`__radd__` () (*gcaops.graph.graph\_vector\_vector.GraphVector\_vector* method), 594  
`__repr__` () (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperator* method), 585  
`__repr__` () (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperatorAlgebra* method), 586  
`__repr__` () (*gcaops.algebra.superfunction\_algebra.Superfunction* method), 580  
`__repr__` () (*gcaops.algebra.superfunction\_algebra.SuperfunctionAlgebra* method), 581  
`__repr__` () (*gcaops.graph.directed\_graph\_complex.DirectedGraphComplex\_dict* method), 615  
`__repr__` () (*gcaops.graph.directed\_graph\_complex.DirectedGraphComplex\_vector* method), 616  
`__repr__` () (*gcaops.graph.directed\_graph\_operad.DirectedGraphOperad\_dict* method), 614  
`__repr__` () (*gcaops.graph.formality\_graph\_complex.FormalityGraphComplex\_dict* method), 630  
`__repr__` () (*gcaops.graph.formality\_graph\_complex.FormalityGraphComplex\_vector* method), 630  
`__repr__` () (*gcaops.graph.formality\_graph\_operad.FormalityGraphOperad\_dict* method), 628  
`__repr__` () (*gcaops.graph.formality\_graph\_operator.FormalityGraphOperator* method), 631  
`__repr__` () (*gcaops.graph.graph\_vector.GraphModule* method), 591  
`__repr__` () (*gcaops.graph.graph\_vector.GraphVector* method), 592  
`__repr__` () (*gcaops.graph.graph\_vector\_dict.GraphModule\_dict* method), 595  
`__repr__` () (*gcaops.graph.graph\_vector\_dict.GraphVector\_dict* method), 596  
`__repr__` () (*gcaops.graph.graph\_vector\_vector.GraphModule\_vector* method), 594  
`__repr__` () (*gcaops.graph.graph\_vector\_vector.GraphVector\_vector* method), 594  
`__repr__` () (*gcaops.graph.undirected\_graph.UndirectedGraph* method), 600  
`__repr__` () (*gcaops.graph.undirected\_graph\_complex.UndirectedGraphComplex\_dict* method), 606  
`__repr__` () (*gcaops.graph.undirected\_graph\_complex.UndirectedGraphComplex\_vector* method), 607  
`__repr__` () (*gcaops.graph.undirected\_graph\_operad.UndirectedGraphOperad\_dict* method), 605  
`__rmul__` () (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperator* method), 585  
`__rmul__` () (*gcaops.algebra.superfunction\_algebra.Superfunction* method), 580  
`__rmul__` () (*gcaops.graph.graph\_vector.GraphVector* method), 592  
`__rmul__` () (*gcaops.graph.graph\_vector\_dict.GraphVector\_dict* method), 596  
`__rmul__` () (*gcaops.graph.graph\_vector\_vector.GraphVector\_vector* method), 594  
`__rsub__` () (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperator* method), 585  
`__rsub__` () (*gcaops.algebra.superfunction\_algebra.Superfunction* method), 580  
`__rsub__` () (*gcaops.graph.graph\_vector\_dict.GraphVector\_dict* method), 596  
`__rsub__` () (*gcaops.graph.graph\_vector\_vector.GraphVector\_vector* method), 595  
`__setitem__` () (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperator* method), 585  
`__setitem__` () (*gcaops.algebra.superfunction\_algebra.Superfunction* method), 580  
`__setstate__` () (*gcaops.graph.graph\_file.GraphFileView* method), 634

`__setstate__()` (*gcaops.graph.graph\_file.UndirectedToDirectedGraphFileView* method), 634  
`__sub__()` (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperator* method), 585  
`__sub__()` (*gcaops.algebra.superfunction\_algebra.Superfunction* method), 580  
`__sub__()` (*gcaops.graph.graph\_vector\_dict.GraphVector\_dict* method), 596  
`__sub__()` (*gcaops.graph.graph\_vector\_vector.GraphVector\_vector* method), 595  
`__truediv__()` (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperator* method), 585  
`__truediv__()` (*gcaops.algebra.superfunction\_algebra.Superfunction* method), 580

## A

`aerial_product()` (*gcaops.graph.formality\_graph.FormalityGraph* method), 617  
`append()` (*gcaops.graph.graph\_file.GraphFileView* method), 634  
`append()` (*gcaops.graph.graph\_file.UndirectedToDirectedGraphFileView* method), 634  
`arity()` (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperator* method), 585  
`attach_to_ground()` (*gcaops.graph.formality\_graph\_vector.FormalityGraphVector* method), 626  
`automorphism_group()` (*gcaops.graph.formality\_graph.FormalityGraph* method), 617

## B

`base_ring()` (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperatorAlgebra* method), 586  
`base_ring()` (*gcaops.algebra.superfunction\_algebra.SuperfunctionAlgebra* method), 581  
`base_ring()` (*gcaops.graph.graph\_vector.GraphModule* method), 591  
`base_ring()` (*gcaops.graph.graph\_vector\_dict.GraphModule\_dict* method), 595  
`base_ring()` (*gcaops.graph.graph\_vector\_vector.GraphModule\_vector* method), 594  
`base_variables()` (*gcaops.algebra.differential\_polynomial\_ring.DifferentialPolynomialRing* method), 589  
`basis()` (*gcaops.graph.graph\_vector.GraphModule* method), 592  
`basis()` (*gcaops.graph.graph\_vector\_dict.GraphModule\_dict* method), 595  
`basis()` (*gcaops.graph.graph\_vector\_vector.GraphModule\_vector* method), 594  
`bracket()` (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperator* method), 585  
`bracket()` (*gcaops.algebra.superfunction\_algebra.Superfunction* method), 580  
`bracket()` (*gcaops.algebra.superfunction\_algebra\_operation.SuperfunctionAlgebraSymmetricOperation* method), 583  
`bracket()` (*gcaops.graph.directed\_graph\_complex.DirectedGraphCochain* method), 614  
`bracket()` (*gcaops.graph.formality\_graph\_complex.FormalityGraphCochain* method), 629  
`bracket()` (*gcaops.graph.graph\_complex.GraphCochain* method), 593  
`bracket()` (*gcaops.graph.undirected\_graph\_complex.UndirectedGraphCochain* method), 605

## C

`canonicalize_edges()` (*gcaops.graph.directed\_graph.DirectedGraph* method), 609  
`canonicalize_edges()` (*gcaops.graph.formality\_graph.FormalityGraph* method), 617  
`canonicalize_edges()` (*gcaops.graph.undirected\_graph.UndirectedGraph* method), 600  
`canonicalize_graph()` (*gcaops.graph.graph\_cache.DirectedGraphCache* method), 635  
`canonicalize_graph()` (*gcaops.graph.graph\_cache.FormalityGraphCache* method), 635  
`canonicalize_graph()` (*gcaops.graph.graph\_cache.GraphCache* method), 635  
`canonicalize_graph()` (*gcaops.graph.graph\_cache.UndirectedGraphCache* method), 635  
`cardinality()` (*gcaops.graph.directed\_graph\_basis.DirectedGraphComplexBasis* method), 611  
`cardinality()` (*gcaops.graph.formality\_graph\_basis.FormalityGraphComplexBasis* method), 622  
`cardinality()` (*gcaops.graph.undirected\_graph\_basis.UndirectedGraphComplexBasis* method), 601  
`codomain()` (*gcaops.algebra.superfunction\_algebra\_operation.SuperfunctionAlgebraOperation* method), 583  
`codomain()` (*gcaops.graph.formality\_graph\_operator.FormalityGraphOperator* method), 631  
`coefficient()` (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperator* method), 585  
`coefficient()` (*gcaops.graph.graph\_vector.GraphVector* method), 592



cohomology\_basis() (*gcaops.graph.directed\_graph\_complex.DirectedGraphComplex\_vector method*), 616  
 cohomology\_basis() (*gcaops.graph.formality\_graph\_complex.FormalityGraphComplex\_vector method*), 630  
 cohomology\_basis() (*gcaops.graph.undirected\_graph\_complex.UndirectedGraphComplex\_vector method*), 607  
 commit() (*gcaops.graph.graph\_file.GraphFileView method*), 634  
 commit() (*gcaops.graph.graph\_file.UndirectedToDirectedGraphFileView method*), 634  
 coordinate() (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperatorAlgebra method*), 586  
 coordinates() (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperatorAlgebra method*), 587  
 copy() (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperator method*), 585  
 copy() (*gcaops.algebra.superfunction\_algebra.Superfunction method*), 580  
 copy() (*gcaops.graph.graph\_vector.GraphVector method*), 592  
 copy() (*gcaops.graph.graph\_vector\_dict.GraphVector\_dict method*), 596  
 copy() (*gcaops.graph.graph\_vector\_vector.GraphVector\_vector method*), 595

## D

degree() (*gcaops.algebra.superfunction\_algebra.Superfunction method*), 580  
 degree() (*gcaops.algebra.superfunction\_algebra\_operation.SuperfunctionAlgebraDirectedGraphOperation method*), 582  
 degree() (*gcaops.algebra.superfunction\_algebra\_operation.SuperfunctionAlgebraOperation method*), 583  
 degree() (*gcaops.algebra.superfunction\_algebra\_operation.SuperfunctionAlgebraSchoutenBracket method*), 583  
 degree() (*gcaops.algebra.superfunction\_algebra\_operation.SuperfunctionAlgebraSymmetricBracketOperation method*), 583  
 degree() (*gcaops.algebra.superfunction\_algebra\_operation.SuperfunctionAlgebraUndirectedGraphOperation method*), 584  
 degrees() (*gcaops.algebra.superfunction\_algebra.Superfunction method*), 580  
 derivative() (*gcaops.algebra.differential\_polynomial\_ring.DifferentialPolynomial method*), 588  
 derivative() (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperatorAlgebra method*), 587  
 derivative() (*gcaops.algebra.superfunction\_algebra.Superfunction method*), 580  
 derivatives() (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperatorAlgebra method*), 587  
 diff() (*gcaops.algebra.differential\_polynomial\_ring.DifferentialPolynomial method*), 588  
 diff() (*gcaops.algebra.superfunction\_algebra.Superfunction method*), 580  
 differential() (*gcaops.graph.directed\_graph\_complex.DirectedGraphCochain\_dict method*), 615  
 differential() (*gcaops.graph.directed\_graph\_complex.DirectedGraphCochain\_vector method*), 615  
 differential() (*gcaops.graph.formality\_graph\_complex.FormalityGraphCochain\_dict method*), 629  
 differential() (*gcaops.graph.formality\_graph\_complex.FormalityGraphCochain\_vector method*), 629  
 differential() (*gcaops.graph.graph\_complex.GraphCochain method*), 593  
 differential() (*gcaops.graph.undirected\_graph\_complex.UndirectedGraphCochain\_dict method*), 605  
 differential() (*gcaops.graph.undirected\_graph\_complex.UndirectedGraphCochain\_vector method*), 605  
 differential\_orders() (*gcaops.graph.formality\_graph.FormalityGraph method*), 617  
 differential\_orders() (*gcaops.graph.formality\_graph\_vector.FormalityGraphVector method*), 626  
 DifferentialPolynomial (*class in gcaops.algebra.differential\_polynomial\_ring*), 588  
 DifferentialPolynomialRing (*class in gcaops.algebra.differential\_polynomial\_ring*), 588  
 dimension() (*gcaops.algebra.superfunction\_algebra.SuperfunctionAlgebra method*), 581  
 DirectedGraph (*class in gcaops.graph.directed\_graph*), 609  
 DirectedGraphBasis (*class in gcaops.graph.directed\_graph\_basis*), 610  
 DirectedGraphCache (*class in gcaops.graph.graph\_cache*), 635  
 DirectedGraphCochain (*class in gcaops.graph.directed\_graph\_complex*), 614  
 DirectedGraphCochain\_dict (*class in gcaops.graph.directed\_graph\_complex*), 614  
 DirectedGraphCochain\_vector (*class in gcaops.graph.directed\_graph\_complex*), 615  
 DirectedGraphComplex() (*in module gcaops.graph.directed\_graph\_complex*), 615  
 DirectedGraphComplex\_ (*class in gcaops.graph.directed\_graph\_complex*), 615

DirectedGraphComplex\_dict (class in *gcaops.graph.directed\_graph\_complex*), 615  
 DirectedGraphComplex\_vector (class in *gcaops.graph.directed\_graph\_complex*), 615  
 DirectedGraphComplexBasis (class in *gcaops.graph.directed\_graph\_basis*), 611  
 DirectedGraphFileView (class in *gcaops.graph.graph\_file*), 633  
 DirectedGraphModule (class in *gcaops.graph.directed\_graph\_vector*), 612  
 DirectedGraphModule\_dict (class in *gcaops.graph.directed\_graph\_vector*), 612  
 DirectedGraphModule\_vector (class in *gcaops.graph.directed\_graph\_vector*), 612  
 DirectedGraphOperad() (in module *gcaops.graph.directed\_graph\_operad*), 614  
 DirectedGraphOperad\_dict (class in *gcaops.graph.directed\_graph\_operad*), 614  
 DirectedGraphOperadBasis (class in *gcaops.graph.directed\_graph\_basis*), 611  
 DirectedGraphOperation\_dict (class in *gcaops.graph.directed\_graph\_operad*), 614  
 DirectedGraphVector (class in *gcaops.graph.directed\_graph\_vector*), 613  
 DirectedGraphVector\_dict (class in *gcaops.graph.directed\_graph\_vector*), 613  
 DirectedGraphVector\_vector (class in *gcaops.graph.directed\_graph\_vector*), 613  
 domain() (*gcaops.algebra.superfunction\_algebra\_operation.SuperfunctionAlgebraOperation* method), 583  
 domain() (*gcaops.graph.formality\_graph\_operator.FormalityGraphOperator* method), 631

## E

edge\_contraction\_graph() (*gcaops.graph.formality\_graph.FormalityGraph* method), 618  
 edges() (*gcaops.graph.directed\_graph.DirectedGraph* method), 609  
 edges() (*gcaops.graph.formality\_graph.FormalityGraph* method), 618  
 edges() (*gcaops.graph.undirected\_graph.UndirectedGraph* method), 600  
 edges\_in\_air() (*gcaops.graph.formality\_graph.FormalityGraph* method), 618  
 element\_class (*gcaops.algebra.differential\_polynomial\_ring.DifferentialPolynomialRing* attribute), 589  
 element\_from\_kgs\_encoding() (*gcaops.graph.formality\_graph\_vector.FormalityGraphModule* method), 625  
 even\_coordinate() (*gcaops.algebra.superfunction\_algebra.SuperfunctionAlgebra* method), 582  
 even\_coordinates() (*gcaops.algebra.superfunction\_algebra.SuperfunctionAlgebra* method), 582

## F

factor() (*gcaops.algebra.tensor\_product.TensorProduct* method), 587  
 factors() (*gcaops.algebra.tensor\_product.TensorProduct* method), 587  
 fibre\_degrees() (*gcaops.algebra.differential\_polynomial\_ring.DifferentialPolynomial* method), 588  
 fibre\_variables() (*gcaops.algebra.differential\_polynomial\_ring.DifferentialPolynomialRing* method), 589  
 file\_view (*gcaops.graph.graph\_cache.DirectedGraphCache* attribute), 635  
 file\_view (*gcaops.graph.graph\_cache.FormalityGraphCache* attribute), 635  
 file\_view (*gcaops.graph.graph\_cache.UndirectedGraphCache* attribute), 636  
 filter() (*gcaops.graph.directed\_graph\_vector.DirectedGraphVector* method), 613  
 filter() (*gcaops.graph.directed\_graph\_vector.DirectedGraphVector\_dict* method), 613  
 filter() (*gcaops.graph.directed\_graph\_vector.DirectedGraphVector\_vector* method), 613  
 filter() (*gcaops.graph.formality\_graph\_vector.FormalityGraphVector* method), 626  
 filter() (*gcaops.graph.formality\_graph\_vector.FormalityGraphVector\_dict* method), 627  
 filter() (*gcaops.graph.formality\_graph\_vector.FormalityGraphVector\_vector* method), 628  
 flipping\_weight\_relations() (*gcaops.graph.formality\_graph\_basis.KontsevichGraphBasis* method), 624  
 formality\_graph\_operator() (in module *gcaops.graph.formality\_graph\_operator*), 631  
 FormalityGraph (class in *gcaops.graph.formality\_graph*), 617  
 FormalityGraphBasis (class in *gcaops.graph.formality\_graph\_basis*), 622  
 FormalityGraphCache (class in *gcaops.graph.graph\_cache*), 635  
 FormalityGraphCochain (class in *gcaops.graph.formality\_graph\_complex*), 629  
 FormalityGraphCochain\_dict (class in *gcaops.graph.formality\_graph\_complex*), 629  
 FormalityGraphCochain\_vector (class in *gcaops.graph.formality\_graph\_complex*), 629



FormalityGraphComplex() (in module *gcaops.graph.formality\_graph\_complex*), 629  
 FormalityGraphComplex\_ (class in *gcaops.graph.formality\_graph\_complex*), 630  
 FormalityGraphComplex\_dict (class in *gcaops.graph.formality\_graph\_complex*), 630  
 FormalityGraphComplex\_vector (class in *gcaops.graph.formality\_graph\_complex*), 630  
 FormalityGraphComplexBasis (class in *gcaops.graph.formality\_graph\_basis*), 622  
 FormalityGraphComplexBasis\_lazy (class in *gcaops.graph.formality\_graph\_basis*), 623  
 FormalityGraphFileView (class in *gcaops.graph.graph\_file*), 633  
 FormalityGraphModule (class in *gcaops.graph.formality\_graph\_vector*), 625  
 FormalityGraphModule\_dict (class in *gcaops.graph.formality\_graph\_vector*), 625  
 FormalityGraphModule\_vector (class in *gcaops.graph.formality\_graph\_vector*), 625  
 FormalityGraphOperad() (in module *gcaops.graph.formality\_graph\_operad*), 628  
 FormalityGraphOperad\_dict (class in *gcaops.graph.formality\_graph\_operad*), 628  
 FormalityGraphOperadBasis (class in *gcaops.graph.formality\_graph\_basis*), 624  
 FormalityGraphOperation\_dict (class in *gcaops.graph.formality\_graph\_operad*), 628  
 FormalityGraphOperator (class in *gcaops.graph.formality\_graph\_operator*), 631  
 FormalityGraphSymmetricOperator (class in *gcaops.graph.formality\_graph\_operator*), 631  
 FormalityGraphVector (class in *gcaops.graph.formality\_graph\_vector*), 626  
 FormalityGraphVector\_dict (class in *gcaops.graph.formality\_graph\_vector*), 627  
 FormalityGraphVector\_vector (class in *gcaops.graph.formality\_graph\_vector*), 627  
 from\_kgs\_encoding() (*gcaops.graph.formality\_graph.FormalityGraph* static method), 618  
 from\_kontsevint\_encoding() (*gcaops.graph.formality\_graph.FormalityGraph* static method), 618

## G

*gcaops.algebra.differential\_polynomial\_ring*  
 module, 588  
*gcaops.algebra.differential\_polynomial\_solver*  
 module, 589  
*gcaops.algebra.polydifferential\_operator*  
 module, 584  
*gcaops.algebra.superfunction\_algebra*  
 module, 579  
*gcaops.algebra.superfunction\_algebra\_operation*  
 module, 582  
*gcaops.algebra.tensor\_product*  
 module, 587  
*gcaops.graph.directed\_graph*  
 module, 609  
*gcaops.graph.directed\_graph\_basis*  
 module, 610  
*gcaops.graph.directed\_graph\_complex*  
 module, 614  
*gcaops.graph.directed\_graph\_operad*  
 module, 614  
*gcaops.graph.directed\_graph\_vector*  
 module, 612  
*gcaops.graph.formality\_graph*  
 module, 617  
*gcaops.graph.formality\_graph\_basis*  
 module, 622  
*gcaops.graph.formality\_graph\_complex*

module, 629

`gcaops.graph.formality_graph_operad`  
module, 628

`gcaops.graph.formality_graph_operator`  
module, 631

`gcaops.graph.formality_graph_vector`  
module, 625

`gcaops.graph.graph_basis`  
module, 591

`gcaops.graph.graph_cache`  
module, 635

`gcaops.graph.graph_complex`  
module, 593

`gcaops.graph.graph_file`  
module, 633

`gcaops.graph.graph_vector`  
module, 591

`gcaops.graph.graph_vector_dict`  
module, 595

`gcaops.graph.graph_vector_vector`  
module, 593

`gcaops.graph.undirected_graph`  
module, 599

`gcaops.graph.undirected_graph_basis`  
module, 601

`gcaops.graph.undirected_graph_complex`  
module, 605

`gcaops.graph.undirected_graph_operad`  
module, 605

`gcaops.graph.undirected_graph_vector`  
module, 603

`gen()` (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperatorAlgebra* method), 587

`gen()` (*gcaops.algebra.superfunction\_algebra.SuperfunctionAlgebra* method), 582

`gens()` (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperatorAlgebra* method), 587

`gens()` (*gcaops.algebra.superfunction\_algebra.SuperfunctionAlgebra* method), 582

`gerstenhaber_bracket()` (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperator* method), 585

`gerstenhaber_bracket()` (*gcaops.graph.formality\_graph\_complex.FormalityGraphCochain* method), 629

`get_pos()` (*gcaops.graph.directed\_graph.DirectedGraph* method), 609

`get_pos()` (*gcaops.graph.formality\_graph.FormalityGraph* method), 619

`get_pos()` (*gcaops.graph.undirected\_graph.UndirectedGraph* method), 600

`graded_symmetrization()` (*gcaops.algebra.tensor\_product.TensorProductElement* method), 587

`gradings()` (*gcaops.graph.graph\_vector.GraphVector* method), 592

`gradings()` (*gcaops.graph.graph\_vector\_dict.GraphVector\_dict* method), 596

`gradings()` (*gcaops.graph.graph\_vector\_vector.GraphVector\_vector* method), 595

`graph_class` (*gcaops.graph.directed\_graph\_basis.DirectedGraphBasis* attribute), 611

`graph_class` (*gcaops.graph.formality\_graph\_basis.FormalityGraphBasis* attribute), 622

`graph_class` (*gcaops.graph.undirected\_graph\_basis.UndirectedGraphBasis* attribute), 601

`graph_operation()` (*gcaops.algebra.superfunction\_algebra.SuperfunctionAlgebra* method), 582

`graph_properties()` (*gcaops.graph.directed\_graph\_basis.DirectedGraphComplexBasis* method), 611

`graph_properties()` (*gcaops.graph.directed\_graph\_basis.DirectedGraphOperadBasis* method), 611

graph\_properties() (*gcaops.graph.formality\_graph\_basis.FormalityGraphComplexBasis* method), 622  
 graph\_properties() (*gcaops.graph.formality\_graph\_basis.FormalityGraphOperadBasis* method), 624  
 graph\_properties() (*gcaops.graph.graph\_basis.GraphBasis* method), 591  
 graph\_properties() (*gcaops.graph.undirected\_graph\_basis.UndirectedGraphComplexBasis* method), 601  
 graph\_properties() (*gcaops.graph.undirected\_graph\_basis.UndirectedGraphOperadBasis* method), 602  
 graph\_to\_key() (*gcaops.graph.directed\_graph\_basis.DirectedGraphComplexBasis* method), 611  
 graph\_to\_key() (*gcaops.graph.directed\_graph\_basis.DirectedGraphOperadBasis* method), 611  
 graph\_to\_key() (*gcaops.graph.formality\_graph\_basis.FormalityGraphComplexBasis* method), 623  
 graph\_to\_key() (*gcaops.graph.formality\_graph\_basis.FormalityGraphComplexBasis\_lazy* method), 623  
 graph\_to\_key() (*gcaops.graph.formality\_graph\_basis.FormalityGraphOperadBasis* method), 624  
 graph\_to\_key() (*gcaops.graph.graph\_basis.GraphBasis* method), 591  
 graph\_to\_key() (*gcaops.graph.undirected\_graph\_basis.UndirectedGraphComplexBasis* method), 602  
 graph\_to\_key() (*gcaops.graph.undirected\_graph\_basis.UndirectedGraphOperadBasis* method), 602  
 GraphBasis (class in *gcaops.graph.graph\_basis*), 591  
 GraphCache (class in *gcaops.graph.graph\_cache*), 635  
 GraphCochain (class in *gcaops.graph.graph\_complex*), 593  
 GraphComplex (class in *gcaops.graph.graph\_complex*), 593  
 GraphFileView (class in *gcaops.graph.graph\_file*), 633  
 GraphModule (class in *gcaops.graph.graph\_vector*), 591  
 GraphModule\_dict (class in *gcaops.graph.graph\_vector\_dict*), 595  
 GraphModule\_vector (class in *gcaops.graph.graph\_vector\_vector*), 593  
 graphs() (*gcaops.graph.directed\_graph\_basis.DirectedGraphComplexBasis* method), 611  
 graphs() (*gcaops.graph.formality\_graph\_basis.FormalityGraphComplexBasis* method), 623  
 graphs() (*gcaops.graph.graph\_cache.DirectedGraphCache* method), 635  
 graphs() (*gcaops.graph.graph\_cache.FormalityGraphCache* method), 635  
 graphs() (*gcaops.graph.graph\_cache.GraphCache* method), 635  
 graphs() (*gcaops.graph.graph\_cache.UndirectedGraphCache* method), 636  
 graphs() (*gcaops.graph.undirected\_graph\_basis.UndirectedGraphComplexBasis* method), 602  
 GraphVector (class in *gcaops.graph.graph\_vector*), 592  
 GraphVector\_dict (class in *gcaops.graph.graph\_vector\_dict*), 595  
 GraphVector\_vector (class in *gcaops.graph.graph\_vector\_vector*), 594  
 ground\_relabeled() (*gcaops.graph.formality\_graph.FormalityGraph* method), 619  
 ground\_skew\_symmetrization() (*gcaops.graph.formality\_graph\_vector.FormalityGraphVector* method), 626  
 ground\_symmetrization() (*gcaops.graph.formality\_graph\_vector.FormalityGraphVector* method), 626

## H

has\_eye\_on\_ground() (*gcaops.graph.formality\_graph.FormalityGraph* method), 619  
 has\_loops() (*gcaops.graph.formality\_graph.FormalityGraph* method), 619  
 has\_multiple\_edges() (*gcaops.graph.formality\_graph.FormalityGraph* method), 619  
 has\_odd\_automorphism() (*gcaops.graph.formality\_graph.FormalityGraph* method), 620  
 hochschild\_differential() (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperator* method), 585  
 hochschild\_differential() (*gcaops.graph.formality\_graph\_complex.FormalityGraphCochain\_dict* method), 629  
 hochschild\_differential() (*gcaops.graph.formality\_graph\_complex.FormalityGraphCochain\_vector* method), 629  
 homogeneous\_monomials() (*gcaops.algebra.differential\_polynomial\_ring.DifferentialPolynomialRing* method), 589  
 homogeneous\_part() (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperator* method), 585  
 homogeneous\_part() (*gcaops.algebra.superfunction\_algebra.Superfunction* method), 580  
 homogeneous\_part() (*gcaops.graph.graph\_vector.GraphVector* method), 592

homogeneous\_part() (*gcaops.graph.graph\_vector\_dict.GraphVector\_dict method*), 596  
homogeneous\_part() (*gcaops.graph.graph\_vector\_vector.GraphVector\_vector method*), 595

## I

identity\_operator() (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperatorAlgebra method*), 587  
in\_degrees() (*gcaops.graph.directed\_graph.DirectedGraph method*), 609  
in\_degrees() (*gcaops.graph.formality\_graph.FormalityGraph method*), 620  
index() (*gcaops.graph.graph\_file.GraphFileView method*), 634  
indices() (*gcaops.algebra.superfunction\_algebra.Superfunction method*), 580  
insertion() (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperator method*), 585  
insertion() (*gcaops.graph.formality\_graph\_vector.FormalityGraphVector method*), 626  
insertion() (*gcaops.graph.graph\_vector.GraphVector method*), 592  
insertion() (*gcaops.graph.graph\_vector\_dict.GraphVector\_dict method*), 596  
insertion() (*gcaops.graph.graph\_vector\_vector.GraphVector\_vector method*), 595  
is\_aerial() (*gcaops.graph.formality\_graph\_vector.FormalityGraphVector method*), 626  
is\_aerial() (*gcaops.graph.formality\_graph\_vector.FormalityGraphVector\_dict method*), 627  
is\_aerial() (*gcaops.graph.formality\_graph\_vector.FormalityGraphVector\_vector method*), 628  
is\_coboundary() (*gcaops.graph.directed\_graph\_complex.DirectedGraphCochain\_vector method*), 615  
is\_coboundary() (*gcaops.graph.undirected\_graph\_complex.UndirectedGraphCochain\_vector method*), 606  
is\_zero() (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperator method*), 585  
is\_zero() (*gcaops.algebra.superfunction\_algebra.Superfunction method*), 581

## K

key\_to\_graph() (*gcaops.graph.directed\_graph\_basis.DirectedGraphComplexBasis method*), 611  
key\_to\_graph() (*gcaops.graph.directed\_graph\_basis.DirectedGraphOperadBasis method*), 612  
key\_to\_graph() (*gcaops.graph.formality\_graph\_basis.FormalityGraphComplexBasis method*), 623  
key\_to\_graph() (*gcaops.graph.formality\_graph\_basis.FormalityGraphComplexBasis\_lazy method*), 623  
key\_to\_graph() (*gcaops.graph.formality\_graph\_basis.FormalityGraphOperadBasis method*), 624  
key\_to\_graph() (*gcaops.graph.graph\_basis.GraphBasis method*), 591  
key\_to\_graph() (*gcaops.graph.undirected\_graph\_basis.UndirectedGraphComplexBasis method*), 602  
key\_to\_graph() (*gcaops.graph.undirected\_graph\_basis.UndirectedGraphOperadBasis method*), 602  
kgs\_encoding() (*gcaops.graph.formality\_graph.FormalityGraph method*), 620  
kgs\_encoding() (*gcaops.graph.formality\_graph\_vector.FormalityGraphVector method*), 626  
KontsevichGraphBasis (*class in gcaops.graph.formality\_graph\_basis*), 624  
kontsevint\_encoding() (*gcaops.graph.formality\_graph.FormalityGraph method*), 620

## L

LeibnizGraphBasis (*class in gcaops.graph.formality\_graph\_basis*), 624

## M

map\_coefficients() (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperator method*), 585  
map\_coefficients() (*gcaops.algebra.superfunction\_algebra.Superfunction method*), 581  
map\_coefficients() (*gcaops.graph.graph\_vector.GraphVector method*), 592  
map\_coefficients() (*gcaops.graph.graph\_vector\_dict.GraphVector\_dict method*), 596  
map\_coefficients() (*gcaops.graph.graph\_vector\_vector.GraphVector\_vector method*), 595  
map\_graphs() (*gcaops.graph.graph\_vector.GraphVector method*), 592  
module  
    gcaops.algebra.differential\_polynomial\_ring, 588  
    gcaops.algebra.differential\_polynomial\_solver, 589  
    gcaops.algebra.polydifferential\_operator, 584

gcaops.algebra.superfunction\_algebra, 579  
 gcaops.algebra.superfunction\_algebra\_operation, 582  
 gcaops.algebra.tensor\_product, 587  
 gcaops.graph.directed\_graph, 609  
 gcaops.graph.directed\_graph\_basis, 610  
 gcaops.graph.directed\_graph\_complex, 614  
 gcaops.graph.directed\_graph\_operad, 614  
 gcaops.graph.directed\_graph\_vector, 612  
 gcaops.graph.formality\_graph, 617  
 gcaops.graph.formality\_graph\_basis, 622  
 gcaops.graph.formality\_graph\_complex, 629  
 gcaops.graph.formality\_graph\_operad, 628  
 gcaops.graph.formality\_graph\_operator, 631  
 gcaops.graph.formality\_graph\_vector, 625  
 gcaops.graph.graph\_basis, 591  
 gcaops.graph.graph\_cache, 635  
 gcaops.graph.graph\_complex, 593  
 gcaops.graph.graph\_file, 633  
 gcaops.graph.graph\_vector, 591  
 gcaops.graph.graph\_vector\_dict, 595  
 gcaops.graph.graph\_vector\_vector, 593  
 gcaops.graph.undirected\_graph, 599  
 gcaops.graph.undirected\_graph\_basis, 601  
 gcaops.graph.undirected\_graph\_complex, 605  
 gcaops.graph.undirected\_graph\_operad, 605  
 gcaops.graph.undirected\_graph\_vector, 603  
 multi\_indices() (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperator* method), 586  
 multiplication\_operator() (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperatorAlgebra*  
*method*), 587  
 multiplicity() (*gcaops.graph.formality\_graph.FormalityGraph* method), 621

## N

nedges() (*gcaops.graph.directed\_graph\_vector.DirectedGraphVector\_dict* method), 613  
 nedges() (*gcaops.graph.directed\_graph\_vector.DirectedGraphVector\_vector* method), 614  
 nedges() (*gcaops.graph.formality\_graph\_vector.FormalityGraphVector\_dict* method), 627  
 nedges() (*gcaops.graph.formality\_graph\_vector.FormalityGraphVector\_vector* method), 628  
 nedges() (*gcaops.graph.graph\_vector.GraphVector* method), 592  
 nedges() (*gcaops.graph.undirected\_graph\_vector.UndirectedGraphVector\_dict* method), 604  
 nedges() (*gcaops.graph.undirected\_graph\_vector.UndirectedGraphVector\_vector* method), 604  
 nfactors() (*gcaops.algebra.tensor\_product.TensorProduct* method), 587  
 ngens() (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperatorAlgebra* method), 587  
 ngens() (*gcaops.algebra.superfunction\_algebra.SuperfunctionAlgebra* method), 582  
 nground() (*gcaops.graph.formality\_graph\_vector.FormalityGraphVector* method), 626  
 nground() (*gcaops.graph.formality\_graph\_vector.FormalityGraphVector\_dict* method), 627  
 nground() (*gcaops.graph.formality\_graph\_vector.FormalityGraphVector\_vector* method), 628  
 num\_aerial\_vertices() (*gcaops.graph.formality\_graph.FormalityGraph* method), 621  
 num\_ground\_vertices() (*gcaops.graph.formality\_graph.FormalityGraph* method), 621  
 nvertices() (*gcaops.graph.directed\_graph\_vector.DirectedGraphVector\_dict* method), 613  
 nvertices() (*gcaops.graph.directed\_graph\_vector.DirectedGraphVector\_vector* method), 614  
 nvertices() (*gcaops.graph.formality\_graph\_vector.FormalityGraphVector\_dict* method), 627



nvertices() (*gcaops.graph.formality\_graph\_vector.FormalityGraphVector\_vector* method), 628  
nvertices() (*gcaops.graph.graph\_vector.GraphVector* method), 593  
nvertices() (*gcaops.graph.undirected\_graph\_vector.UndirectedGraphVector\_dict* method), 604  
nvertices() (*gcaops.graph.undirected\_graph\_vector.UndirectedGraphVector\_vector* method), 604

## O

odd\_coordinate() (*gcaops.algebra.superfunction\_algebra.SuperfunctionAlgebra* method), 582  
odd\_coordinates() (*gcaops.algebra.superfunction\_algebra.SuperfunctionAlgebra* method), 582  
one() (*gcaops.algebra.superfunction\_algebra.SuperfunctionAlgebra* method), 582  
orientations() (*gcaops.graph.undirected\_graph.UndirectedGraph* method), 600  
out\_degrees() (*gcaops.graph.directed\_graph.DirectedGraph* method), 610  
out\_degrees() (*gcaops.graph.formality\_graph.FormalityGraph* method), 621

## P

parent() (*gcaops.algebra.differential\_polynomial\_ring.DifferentialPolynomial* method), 588  
parent() (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperator* method), 586  
parent() (*gcaops.algebra.superfunction\_algebra.Superfunction* method), 581  
parent() (*gcaops.algebra.tensor\_product.TensorProductElement* method), 588  
parent() (*gcaops.graph.graph\_vector.GraphVector* method), 593  
parent() (*gcaops.graph.graph\_vector\_dict.GraphVector\_dict* method), 596  
parent() (*gcaops.graph.graph\_vector\_vector.GraphVector\_vector* method), 595  
part\_of\_differential\_order() (*gcaops.graph.formality\_graph\_vector.FormalityGraphVector* method), 626  
partial\_derivative() (*gcaops.algebra.differential\_polynomial\_ring.DifferentialPolynomial* method), 588  
pdiff() (*gcaops.algebra.differential\_polynomial\_ring.DifferentialPolynomial* method), 588  
plot() (*gcaops.graph.directed\_graph.DirectedGraph* method), 610  
plot() (*gcaops.graph.formality\_graph.FormalityGraph* method), 621  
plot() (*gcaops.graph.graph\_vector.GraphVector* method), 593  
plot() (*gcaops.graph.undirected\_graph.UndirectedGraph* method), 600  
PolyDifferentialOperator (class in *gcaops.algebra.polydifferential\_operator*), 584  
PolyDifferentialOperatorAlgebra (class in *gcaops.algebra.polydifferential\_operator*), 586

## R

reabeled() (*gcaops.graph.directed\_graph.DirectedGraph* method), 610  
reabeled() (*gcaops.graph.formality\_graph.FormalityGraph* method), 622  
reabeled() (*gcaops.graph.undirected\_graph.UndirectedGraph* method), 601

## S

schouten\_bracket() (*gcaops.algebra.superfunction\_algebra.Superfunction* method), 581  
schouten\_bracket() (*gcaops.algebra.superfunction\_algebra.SuperfunctionAlgebra* method), 582  
schouten\_bracket() (*gcaops.graph.formality\_graph\_complex.FormalityGraphCochain* method), 629  
set\_aerial() (*gcaops.graph.formality\_graph\_vector.FormalityGraphVector* method), 627  
set\_aerial() (*gcaops.graph.formality\_graph\_vector.FormalityGraphVector\_dict* method), 627  
set\_aerial() (*gcaops.graph.formality\_graph\_vector.FormalityGraphVector\_vector* method), 628  
set\_pos() (*gcaops.graph.directed\_graph.DirectedGraph* method), 610  
set\_pos() (*gcaops.graph.formality\_graph.FormalityGraph* method), 622  
set\_pos() (*gcaops.graph.undirected\_graph.UndirectedGraph* method), 601  
show() (*gcaops.graph.directed\_graph.DirectedGraph* method), 610  
show() (*gcaops.graph.formality\_graph.FormalityGraph* method), 622  
show() (*gcaops.graph.graph\_vector.GraphVector* method), 593  
show() (*gcaops.graph.undirected\_graph.UndirectedGraph* method), 601

skew\_symmetrization() (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperator* method), 586  
 solve\_homogeneous\_diffpoly() (*in module gcaops.algebra.differential\_polynomial\_solver*), 589  
 subs() (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperator* method), 586  
 Superfunction (*class in gcaops.algebra.superfunction\_algebra*), 579  
 SuperfunctionAlgebra (*class in gcaops.algebra.superfunction\_algebra*), 581  
 SuperfunctionAlgebraDirectedGraphOperation (*class in gcaops.algebra.superfunction\_algebra\_operation*),  
 582  
 SuperfunctionAlgebraOperation (*class in gcaops.algebra.superfunction\_algebra\_operation*), 583  
 SuperfunctionAlgebraSchoutenBracket (*class in gcaops.algebra.superfunction\_algebra\_operation*), 583  
 SuperfunctionAlgebraSymmetricBracketOperation (*class in gcaops.algebra.superfunction\_algebra\_operation*),  
 583  
 SuperfunctionAlgebraSymmetricDirectedGraphOperation (*class in gcaops.algebra.superfunction\_algebra\_operation*),  
 583  
 SuperfunctionAlgebraSymmetricOperation (*class in gcaops.algebra.superfunction\_algebra\_operation*), 583  
 SuperfunctionAlgebraSymmetricUndirectedGraphOperation (*class in gcaops.algebra.superfunction\_algebra\_operation*),  
 583  
 SuperfunctionAlgebraUndirectedGraphOperation (*class in gcaops.algebra.superfunction\_algebra\_operation*),  
 583  
 symmetrization() (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperator* method), 586

## T

tensor\_power() (*gcaops.algebra.superfunction\_algebra.SuperfunctionAlgebra* method), 582  
 tensor\_product() (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperatorAlgebra* method), 587  
 TensorProduct (*class in gcaops.algebra.tensor\_product*), 587  
 TensorProductElement (*class in gcaops.algebra.tensor\_product*), 587  
 terms() (*gcaops.algebra.tensor\_product.TensorProductElement* method), 588  
 total\_derivative() (*gcaops.algebra.differential\_polynomial\_ring.DifferentialPolynomial* method), 588

## U

undirected\_to\_directed\_coeffs() (*gcaops.graph.graph\_file.UndirectedToDirectedGraphFileView* method),  
 634  
 UndirectedGraph (*class in gcaops.graph.undirected\_graph*), 599  
 UndirectedGraphBasis (*class in gcaops.graph.undirected\_graph\_basis*), 601  
 UndirectedGraphCache (*class in gcaops.graph.graph\_cache*), 635  
 UndirectedGraphCochain (*class in gcaops.graph.undirected\_graph\_complex*), 605  
 UndirectedGraphCochain\_dict (*class in gcaops.graph.undirected\_graph\_complex*), 605  
 UndirectedGraphCochain\_vector (*class in gcaops.graph.undirected\_graph\_complex*), 605  
 UndirectedGraphComplex() (*in module gcaops.graph.undirected\_graph\_complex*), 606  
 UndirectedGraphComplex\_ (*class in gcaops.graph.undirected\_graph\_complex*), 606  
 UndirectedGraphComplex\_dict (*class in gcaops.graph.undirected\_graph\_complex*), 606  
 UndirectedGraphComplex\_vector (*class in gcaops.graph.undirected\_graph\_complex*), 606  
 UndirectedGraphComplexBasis (*class in gcaops.graph.undirected\_graph\_basis*), 601  
 UndirectedGraphFileView (*class in gcaops.graph.graph\_file*), 634  
 UndirectedGraphModule (*class in gcaops.graph.undirected\_graph\_vector*), 603  
 UndirectedGraphModule\_dict (*class in gcaops.graph.undirected\_graph\_vector*), 603  
 UndirectedGraphModule\_vector (*class in gcaops.graph.undirected\_graph\_vector*), 603  
 UndirectedGraphOperad() (*in module gcaops.graph.undirected\_graph\_operad*), 605  
 UndirectedGraphOperad\_dict (*class in gcaops.graph.undirected\_graph\_operad*), 605  
 UndirectedGraphOperadBasis (*class in gcaops.graph.undirected\_graph\_basis*), 602  
 UndirectedGraphOperation\_dict (*class in gcaops.graph.undirected\_graph\_operad*), 605

UndirectedGraphVector (*class in gcaops.graph.undirected\_graph\_vector*), 603  
UndirectedGraphVector\_dict (*class in gcaops.graph.undirected\_graph\_vector*), 603  
UndirectedGraphVector\_vector (*class in gcaops.graph.undirected\_graph\_vector*), 604  
UndirectedToDirectedGraphFileView (*class in gcaops.graph.graph\_file*), 634

## V

value\_at\_copies\_of() (*gcaops.graph.formality\_graph\_operator.FormalityGraphOperator method*), 631  
vector() (*gcaops.graph.graph\_vector\_vector.GraphVector\_vector method*), 595

## W

weights() (*gcaops.algebra.differential\_polynomial\_ring.DifferentialPolynomial method*), 588

## Z

zero() (*gcaops.algebra.polydifferential\_operator.PolyDifferentialOperatorAlgebra method*), 587  
zero() (*gcaops.algebra.superfunction\_algebra.SuperfunctionAlgebra method*), 582  
zero() (*gcaops.graph.graph\_vector.GraphModule method*), 592  
zero() (*gcaops.graph.graph\_vector\_dict.GraphModule\_dict method*), 595  
zero() (*gcaops.graph.graph\_vector\_vector.GraphModule\_vector method*), 594