Computer-algebraische und analytische Methoden zur Berechnung von Vertexfunktionen im Standardmodell

Dissertation

zur Erlangung des Grades
„Doktor der Naturwissenschaften“
am Fachbereich Physik
der Johannes Gutenberg-Universität in Mainz

Alexander Frink

geboren in Wiesbaden

Mainz 2000

Datum der mündlichen Prüfung: 26.05.2000

Zusammenfassung

Das Standardmodell der elektroschwachen Wechselwirkung hat in den vergangenen Jahrzehnten beachtliche Erfolge erzielt. Die vorhergesagten Teilchen wurden, bis auf das Higgs-Boson, experimentell nachgewiesen. Bei Streu- und Zerfallsprozessen stimmen die experimentell gemessenen und die theoretisch berechneten Wirkungsquerschnitte im Rahmen ihrer jeweiligen Genauigkeit sehr gut überein. Um Abweichungen vom Standardmodell zu finden und um so Hinweise auf eine umfassendere Theorie zu erhalten, muß die Präzision sowohl im Experiment als auch bei den theoretischen Berechnungen erhöht werden.

Auf der theoretischen Seite ist dies gleichbedeutend mit dem Schritt zu der nächsthöheren Ordnung der Störungsreihe, im Falle des Standardmodells ist dies die Zweiloop-Ordnung. Wegen der großen Zahl von Teilchen im Standardmodell tragen viele Feynman-Graphen, in der Regel mehrere Tausend, zu einem Prozeß bei. Die Berechnung jedes einzelnen dieser Graphen ist wegen der vielen verschiedenen Massenskalen auch noch deutlich schwieriger als beispielsweise in der Quantenelektrodynamik. Diese Aufgabe läßt sich nur noch mit Computer-Unterstützung lösen.

Es existieren zwar bereits Programmpakete, mit denen Prozesse mehr oder weniger automatisch berechnet werden können, jedoch nur zur Baumgraphen- oder Einloop-Ordnung. Zur Zweiloop-Ordnung können noch nicht einmal alle einzelnen Diagramme in ihrer vollen Allgemeinheit berechnet werden. Lediglich Zweipunkt-Funktionen können methodisch als abgeschlossen betrachtet werden.

Ziel dieser Arbeit ist es daher, zum einen analytische Methoden zu entwickeln, mit denen beliebige Dreipunkt-Feynman-Graphen berechnet werden können. Diese werden z.B. für die Behandlung von Zerfallsprozessen benötigt. Zum anderen soll ein Rahmen geschaffen werden, der eine Einbettung dieser Methoden in ein Programmpaket zur automatisierten Berechnung von Prozessen erlaubt.

Kapitel 1 behandelt die Entwicklung der Methoden zur Berechnung von Zweiloop-Dreipunkt-Integralen für den Fall beliebiger Massen und Impulse mit Hilfe der Parallel-/Orthogonalraum-Integrationstechnik. Die zugehörigen konvergenten skalaren Integrale sind bereits berechnet worden. Der Schwerpunkt liegt daher auf der Behandlung der verschiedenen Divergenzen, mit denen man in physikalischen Anwendungen konfrontiert wird, nämlich den ultraviolett, infrarot und kollinear divergenten Beiträgen. Besonders wichtig sind die Integrale mit Tensorstruktur, für die ein Verfahren vorgestellt wird, das diese auf eine übersichtliche Basismenge von Standardintegralen reduziert. Weiterhin wird gezeigt, wie die Integrale der Basismenge mit Hilfe der Parallel-/Orthogonalraum-Methode berechnet werden können.

Für die Implementierung eines Programmpakets zur automatisierten Berechnung von Ein- und Zweiloop-Feynman-Diagrammen, die gleichermaßen symbolische wie numerische Manipulationen von Ausdrücken erfordern, haben sich konventionelle Computer-Algebra-Systeme wie Maple oder Mathematica als nicht geeignet erwiesen. In Kapitel 2 wird daher die Programmbibliothek GiNaC vorgestellt, die die Programmiersprache C++ um Fähigkeiten zur Verarbeitung symbolischer Ausdrücke erweitert.

Mit diesen Bausteinen, den allgemeinen Berechnungsmethoden von Zweiloop-Dreipunkt-Funktionen und einer geeigneten Programmierumgebung, wird in Kapitel 3 ein Konzept für ein neues Paket zur automatisierten Berechnung von Feynman-Graphen eingeführt. Es basiert auf Ideen des in Mainz entwickelten Paketes *xloops*, dessen Weiterentwicklung und Ergänzung um Zweiloop-Dreipunkt-Methoden jedoch nicht mehr sinnvoll erschien. Dort werden auch noch einige Komponenten beschrieben, die bisher in *xloops* fehlten.

Inhaltsverzeichnis

Einleitung

Die Elementarteilchenphysik hat sich zur Aufgabe gesetzt, die grundlegenden Bausteine der Materie und ihre Wechselwirkungen qualitativ und quantitativ zu beschreiben. Als erfolgreiches, gemeinsames Konzept haben sich dabei relativistische Quantenfeldtheorien — eine Verbindung von klassischer Feldtheorie, Relativitätstheorie und Quantenmechanik — herausgestellt. Ein wesentlicher Bestandteil ist dabei auch die Idee der Erweiterung einer globalen Symmetrie auf eine lokale Symmetrie. Dabei müssen neue Felder, die sogenannten Eichfelder, eingeführt werden, die sich dann als Träger einer Wechselwirkung darstellen.

Die älteste dieser Eichtheorien ist die Quantenelektrodynamik (QED), die die Wechselwirkung zwischen Elektronen, deren Antiteilchen Positronen und den Wechselwirkungsteilchen des elektromagnetischen Feldes, den Photonen, beschreibt. Die Dirac-Gleichung für freie Elektronen und Positronen ist invariant unter einer globalen Änderung der Phase der Wellenfunktionen, was einer Symmetrie unter Transformationen aus der Gruppe U(1) entspricht. Eine lokale, also ortsabhängige, Phasenänderung zerstört zunächst die Invarianz der Dirac-Gleichung. Diese kann aber durch Einführung eines masselosen Photon-Feldes, das mit den Elektronen wechselwirkt, restauriert werden. Die Theorie der QED ist nicht exakt lösbar und muß störungstheoretisch als Entwicklung nach einem kleinen Parameter, der Kopplungskonstante α, behandelt werden. Erst durch das Konzept der Renormierung, durch die der Zusammenhang zwischen den in der Theorie auftretenden Parametern und den physikalisch beobachtbaren Größen wie Massen und Kopplungskonstanten uminterpretiert wird, wurde es möglich, die in höheren Ordnungen der Störungsreihe (sogenannten *Loops*) auftretenden Divergenzen systematisch zu behandeln und so sinnvolle physikalische Vorhersagen zu erhalten. Eine wichtige Rolle spielt dabei die graphische Darstellung der zu berechnenden Größen in Form sogenannter Feynman-Diagramme, mit deren Hilfe die komplizierten Rechnungen in übersichtlicher Weise organisiert werden können. Für die Entwicklung der theoretischen Grundlagen der QED wurden Tomonaga, Schwinger und Feynman 1965 mit dem Nobelpreis ausgezeichnet. Die QED zählt heute zu den am genauesten berechneten und experimentell verifizierten physikalischen Theorien (Nobelpreis 1989, unter anderem für die Messung des anomalen magnetischen Moments des Elektrons mit einer Genauigkeit von 12 Stellen, von denen die ersten 10 direkt mit der Theorie übereinstimmen).

Seit 1930 wurde versucht, auch die schwache Wechselwirkung, die unter anderem den radioaktiven Beta-Zerfall verursacht, als Quantenfeldtheorie zu formulieren. Die Fermi-Theorie als Punktwechselwirkung von vier Fermionen lieferte zwar eine gute phänomenologische Beschreibung, verletzte aber bei hohen Energien die Unitarität, also die Wahrscheinlichkeitsinterpretation der Quantentheorie. Sie war auch nicht renormierbar, so daß keine sinnvolle Störungsentwicklung möglich war. Auch die Einführung massiver geladener W-Bosonen als Träger der Wechselwirkung in Analogie zum Photon konnte diese Probleme

zunächst nicht lösen.

1954 schlugen Yang und Mills vor, die globale SU(2)-Isospin-Symmetrie der Nukleonen als lokale Eichsymmetrie zu formulieren. Da die SU(2) eine nicht-abelsche (nichtkommutative) Gruppe ist, spricht man in diesem Zusammenhang auch von einer nichtabelschen Eichtheorie. Die Theorie sagte die Existenz dreier masseloser Wechselwirkungs-Bosonen mit Spin 1 und den Ladungen $+1$, 0 und -1 voraus, die als die ρ-Mesonen ρ^+, ρ^0 und ρ^- identifiziert werden können. Die Isospin-Symmetrie ist aber gebrochen, Proton und Neutron haben unterschiedliche Massen, und die ρ-Mesonen sind massiv. Die einfache Yang-Mills-Theorie war damit kein Kandidat für eine phänomenologisch anwendbare Theorie.

Auf der Grundlage dieses Modells gelang es Ende der 60er Jahre Glashow, Salam und Weinberg, eine Eichtheorie zu formulieren, die die schwache mit der elektromagnetischen Wechselwirkung auf Basis einer SU(2)×U(1)-Symmetrie vereint. Eines der größten Probleme, nämlich, daß die zugehörigen vier Wechselwirkungsteilchen zum Teil massiv sein müssen, um die scheinbare Punktwechselwirkung des Fermi-Modells bei niedrigen Energien erklären zu können, lösten sie durch Postulierung eines spontan gebrochenen Vakuum-Grundzustandes mit Hilfe des sogenannten Higgs-Mechanismus. Ein komplexes Higgs-Dublett, ein Feld mit vier Freiheitsgraden, kann mit drei seiner Freiheitsgrade drei der Wechselwirkungsteilchen (bzw. Linearkombinationen davon) Masse geben, während der vierte Freiheitsgrad als physikalisches Higgs-Boson mit Spin 0 übrig bleibt. Die drei massiven Wechselwirkungsteilchen nennt man W^+, W^- und Z^0, das vierte, das aus der elektromagnetischen Wechselwirkung bekannte Photon, ist masselos. Der Higgs-Mechanismus kann zusätzlich den im Modell vorhandenen Fermionen über die sogenannte Yukawa-Kopplung Masse geben. Für dieses *Standardmodell der elektroschwachen Wechselwirkung* wurde Glashow, Salam und Weinberg 1979 der Nobelpreis verliehen.

Das Standardmodell wurde in den ersten Jahren nur zögerlich akzeptiert, da es nicht renormierbar und somit keine Berechnungen in höheren Ordnungen der Störungstheorie zu erlauben schien. Erst 't Hooft und Veltman gelang 1971 der Beweis der Renormierbarkeit nicht-abelscher Eichtheorien in allen Ordnungen der Störungstheorie und damit die Berechnung von Prozessen bis zur Einloop-Ordnung, wofür ihnen 1999 der Nobelpreis verliehen wurde.

In den vergangenen Jahrzehnten hat das Standardmodell, erweitert um die Quantenchromodynamik (QCD), die die starke Wechselwirkung der Quarks als Eichtheorie zur SU(3)-Farbsymmetrie beschreibt, beachtliche Erfolge erzielt, so durch die Beobachtung schwacher neutraler Ströme 1973 (ein erster indirekter Nachweis der Existenz des Z^0), den Nachweis von Gluonen in Drei-Jet-Ereignissen 1979, den direkten Nachweis der massiven W- und Z-Bosonen 1983 (Nobelpreis 1984) und die Vervollständigung der drei Quark- und Lepton-Familien, zuletzt durch das Top-Quark 1995. Da die zuvor unbekannte Masse des Top-Quarks durch Berücksichtigung von höheren Ordnungen der Störungstheorie in Übergangsamplituden eingeht, ließ sich ein erlaubter Bereich der Masse durch Vergleich mit dem Experiment vorhersagen, der gut mit den gemessenen 175 GeV übereinstimmt. Einzig das Higgs-Boson konnte noch nicht experimentell nachgewiesen werden, bisher können nur Ausschlußgrenzen angegeben werden. Die Untergrenze seiner möglichen Masse liegt zur Zeit bei 95.2 GeV [77], die im wesentlichen daher stammt, daß das Higgs-Boson nicht direkt bei LEP2 (zweite Ausbaustufe des *Large Electron Positron Collider*) beobachtet werden konnte. Eine obere Schranke wird von einigen unabhängigen theoretischen Argumenten wie Unitarität, Gitterrechnungen oder Konvergenz der Störungsreihe geliefert und liegt zwischen 700 GeV und 1 TeV [32]. Auch aus den Experimenten erhält man, wie zu-

vor beim Top-Quark, indirekt eine obere Schranke bei einigen hundert GeV [73]. Diese ist jedoch mit großen theoretischen Unsicherheiten behaftet, da die Higgsmasse nur logarithmisch in Looprechnungen eingeht und daher z.B. experimentelle Fehler in α_S und $\alpha(M_Z)$ einen großen Einfluß haben. Der erlaubte Massenbereich des Higgs-Bosons kann aber vom LHC (*Large Hadron Collider*) abgedeckt werden, so daß das Higgs-Boson dort mit großer Wahrscheinlichkeit entdeckt werden kann, wenn man annimmt, daß das Standardmodell gültig ist.

Auf der anderen Seite gibt es einige theoretische Gründe, die das Standardmodell unvollständig erscheinen lassen. So stört beispielsweise die hohe Zahl der unabhängigen freien Parameter (25 bzw. 18, wenn man die Neutrinos als masselos annimmt), die experimentell angepaßt werden müssen. Das Standardmodell erklärt nicht die Herkunft der unterschiedlichen schwachen, elektromagnetischen und starken Ladungen und die Hierarchie der Massen der in ihm enthaltenen Teilchen und ist auch nicht in der Lage, die Gravitation als vierte fundamentale Wechselwirkung einzubeziehen.

Verschiedene Theorien wurden vorgeschlagen, um das Standardmodell zu erweitern, beispielsweise supersymmetrische Modelle (SUSY) oder *Grand Unified Theories* (GUT). Ihr Einfluß auf physikalische Prozesse ist aber bei den heute erreichbaren Beschleunigerenergien klein, da die typischen Massenskalen der neu vorhergesagten Teilchen groß sind. Als grundlegende Vorgehensweise ergibt sich daraus, daß einerseits die experimentelle Präzision und gleichzeitig die Genauigkeit der theoretischen Berechnungen erhöht werden müssen, um eventuelle Abweichungen von den Vorhersagen des Standardmodells feststellen zu können. Damit erhofft man sich Hinweise auf die Eigenschaften erweiterter Theorien.

Während reine QED-Prozesse inzwischen bis zum Vierloop-Niveau berechnet werden können und so eine relative Genauigkeit besser als 10^{-8} erreichen, konnten theoretische Berechnungen im Standardmodell bis auf wenige Ausnahmen noch nicht über die Einloop-Ordnung hinaus verbessert werden. Gleichzeitig besteht schon jetzt durch die präzisen Messungen von LEP in der Nähe der Z^0-Resonanz ein großes Interesse an genaueren theoretischen Vorhersagen.

Einer der Gründe für die Probleme, mit denen sich Berechnungen konfrontiert sehen, ist die große Anzahl der zu Zweiloop-Prozessen beitragenden Feynman-Graphen. Sie liegt bei Selbstenergien in der Größenordnung $\mathcal{O}(10^2)$ (z.B. Photon-Selbstenergie: 272 Diagramme in der unitären Eichung, gegenüber 3 in der QED), bei Zerfällen bei $\mathcal{O}(10^3)$ (z.B. $Z \to b\bar{b}$: 1534 Diagramme) und bei Streuprozessen bei $\mathcal{O}(10^4)$ (z.B. $\gamma\gamma \to W^+W^-$: 15876 Diagramme). Ohne Computerunterstützung ist weder die Erzeugung und Verwaltung noch die Berechnung der Diagramme denkbar. Dazu kommt, daß die Berechnung jedes einzelnen dieser Diagramme wegen der Vielzahl der auftretenden Massenskalen deutlich komplizierter als in der QED ist. Es ist bekannt, daß sich nicht alle Diagramme analytisch berechnen lassen [80], so daß hier die Verwendung von Näherungsmethoden wie numerischer Integration oder Reihenentwicklungen unausweichlich wird. Dies schließt jedoch nicht aus, daß für einzelne Diagramme, beispielsweise bei gleichen Massen, analytische Ergebnisse angegeben werden können. Im folgenden wird daher immer vom schwierigsten Fall beliebiger Massen und Impulse ausgegangen.

Es existieren bereits Programmpakete, die automatisch alle Schritte ausführen, die zur Berechnung von Prozessen im Standardmodell nötig sind, angefangen bei der Erzeugung der Graphen über die Berechnung der einzelnen Diagramme bis zur Phasenraumintegration unter Berücksichtigung experimentell notwendiger Schnitte. Das Paket CompHEP [4] beschränkt sich dabei auf Baumgraphen, ist aber durch das Zusatzpaket LanHEP [81] sehr

flexibel in der Wahl der zugrundeliegenden Quantenfeldtheorie, während grace [45] auch Einloop-Korrekturen berücksichtigen kann. Das in Mainz entwickelte Paket *xloops* [9] beherrscht Einloop-Korrekturen und auch Zweiloop-Korrekturen zu Selbstenergie-Graphen, berechnet aber nur die einzelnen Diagramme. Weitere Einloop-Programme, die jeweils nur Teilaspekte behandeln, sind FeynArts [57] (Erzeugen und Zeichnen von Feynman-Diagrammen, Reduzierung auf Tensor-Integrale), FeynCalc [66] (Reduzierung von Tensor-Integralen auf skalare Integrale) und FF [70] (numerische Evaluierung von skalaren Integralen). Eine Übersicht findet sich in [40].

Für die Erzeugung aller Graphen, die zu Zweiloop-Prozessen beitragen, können QGRAF [68] und grace verwendet werden. Für Berechnungen auf Zweiloop-Niveau existiert aber noch kein vollständiges Programmpaket. Bisher gibt es nur eine Reihe von methodischen Vorarbeiten, so für Zweipunkt-Funktionen

- die in Mainz entwickelte Parallel-/Orthogonalraum-Integrationsmethode, mit der Zweiloop-Zweipunkt-Diagramme vollständig berechnet werden können und die auf eine numerische Zweifach-Integration führt [54] (bereits in *xloops* implementiert),
- Reduzierung von Tensor-Integralen auf skalare Diagramme durch ein iteriertes Passarino-Veltman-Verfahren (Paket TwoCalc) [90], Berechnung der skalaren Integrale mit einer numerischen Einfach-Integration [1] (C++-Programme s2lse und master),
- Berechnung mit Hilfe von Differentialgleichungen im Quadrat des äußeren Impulses [78],
- asymptotische Entwicklungen nach Massen oder Impulsen mit Padé-Approximation zur Verbesserung der Konvergenz [6, 21, 26].

Zur Berechnung von skalaren Dreipunkt-Funktionen gibt es ebenfalls einige Ansätze:

- Berechnung der Integrale mit der Parallel-/Orthogonalraum-Methode [16, 31, 56], die auch hier auf eine numerische Zweifach-Integration führt,
- asymptotische Entwicklung in einem äußeren Impuls, wobei zur Zeit nur eingeschränkte Impulskonfigurationen behandelt werden können [22],
- höherdimensionale numerische Integration über Feynman-Parameter mit Monte-Carlo-Methoden [33].

Auch für Vierpunkt-Funktionen liefert die Parallel-/Orthogonalraum-Methode Ergebnisse in Form einer Dreifach-Integraldarstellung für die skalaren Diagramme [50].

Ziel dieser Arbeit ist es, weitere methodische Vorarbeiten zu leisten, die für eine Berechnung allgemeiner Zweiloop-Dreipunkt-Funktionen notwendig sind, sowie einen Rahmen zu schaffen, der eine Einbettung dieser Methoden in ein Paket zur automatisierten Berechnung von Prozessen erlaubt.

Kapitel 1 behandelt die Entwicklung der Methoden zur Berechnung von Zweiloop-Dreipunkt-Integralen für den Fall beliebiger Massen und Impulse mit Hilfe der Parallel-/Orthogonalraum-Integrationstechnik. Der Schwerpunkt liegt auf der Behandlung der verschiedenen Divergenzen, mit denen man in physikalischen Anwendungen konfrontiert wird, nämlich den ultraviolett, infrarot und kollinear divergenten Beiträgen.

Für die Implementierung eines Programmpakets zur automatisierten Berechnung von Ein- und Zweiloop-Feynman-Diagrammen, die gleichermaßen symbolische wie numerische

Manipulationen von Ausdrücken erfordern, haben sich konventionelle Computer-Algebra-Systeme wie Maple oder Mathematica als nicht geeignet erwiesen. In Kapitel 2 wird daher die Programmbibliothek GiNaC vorgestellt, die die Programmiersprache C++ um Fähigkeiten zur Verarbeitung symbolischer Ausdrücke erweitert.

Mit diesen Bausteinen, den allgemeinen Berechnungsmethoden von Zweiloop-Dreipunkt-Funktionen und einer geeigneten Programmierumgebung, wird in Kapitel 3 ein Konzept für ein neues Paket zur automatisierten Berechnung von Feynman-Graphen, das auf den Ideen des *xloops*-Pakets aufbaut, eingeführt. Dort werden auch noch einige Komponenten beschrieben, die bisher in *xloops* fehlten.

Kapitel 1

Zweiloop-Dreipunkt-Funktionen

Die Berechnung von Zweiloop-Dreipunkt-Funktionen für den allgemeinen Fall beliebiger innerer Massen und äußerer Impulse durch Aufspaltung der Integrationen in Komponenten aus einem Parallelraum, der von den äußeren Impulsen aufgespannt wird, und einen dazu senkrechten Orthogonalraum wurde zuerst in [56] demonstriert. Eine Ausarbeitung der Details der dort gefundenen Dreifach-Integraldarstellung und die Möglichkeit der Reduktion auf eine besser handhabbare, numerisch stabilere Zweifach-Integraldarstellung der skalaren planaren Dreipunkt-Funktion (Abb. 1.1a) wurde in [16] vorgenommen. Eine Verallgemeinerung auf die skalaren Topologien in Abb. 1.1b und c findet sich in [30, 31].

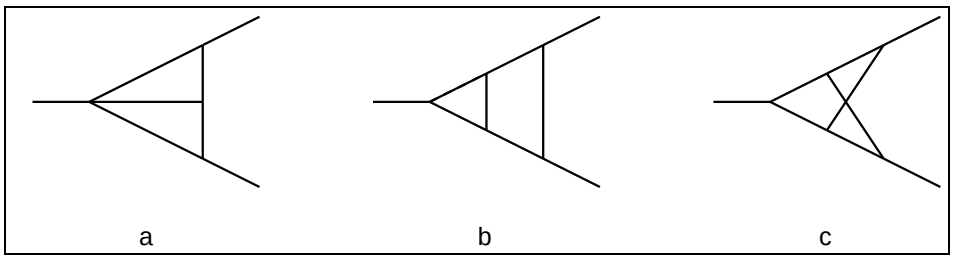


Abbildung 1.1: Dreipunkt-Topologien, deren skalare Diagramme bereits erfolgreich mit der Parallel-/Orthogonalraum-Methode berechnet wurden

Die bisherigen Arbeiten decken aber nur den Fall ab, daß die Integrale konvergent sind. Ultraviolette oder infrarote Divergenzen, mit denen man in physikalischen Anwendungen konfrontiert wird, können nicht direkt berechnet werden, sie benötigen zuvor eine Regularisierung und Abspaltung der divergenten Anteile.

In diesem Kapitel werden Verfahren vorgestellt, mit denen man auch divergente Zweiloop-Dreipunkt-Funktionen für den Fall beliebiger Massen und Impulse (bzw. spezieller Massen und Impulse, bei denen infrarote oder kollineare Divergenzen auftreten) mit Hilfe von Parallel-/Orthogonalraum-Integrationen berechnen kann. Weiterhin finden sich andere Resultate im Zusammenhang mit dieser Methode zur Berechnung von Feynman-Graphen.

1.1 Parallel-/Orthogonalraum-Integrationsmethode

Da die Berechnung divergenter Dreipunkt-Funktionen auf den zugehörigen konvergenten skalaren Diagrammen aufbaut, wird im folgenden eine Einführung in die Parallel-/Orthogonalraum-Integrationsmethode am Beispiel der planaren Dreipunkt-Funktion [16] gegeben sowie erläutert, welche Probleme mit dieser Methode bei divergenten Dreipunkt-Funktionen auftreten und wie sie gelöst werden können.

1.1.1 Konvergente Zweiloop-Dreipunkt-Funktionen

Für die planare Dreipunkt-Funktion Abb. 1.2 lautet das zu berechnende konvergente Integral

$$V_P = \int d^4l \int d^4k \, \frac{1}{P_1 P_2 P_3 P_4 P_5 P_6} \tag{1.1}$$

mit Propagatoren $P_1(l+q_1, m_1)$, $P_2(l-q_2, m_2)$, $P_3(l+k, m_3)$, $P_4(k-q_1, m_4)$, $P_5(k+q_2, m_5)$ und $P_6(k, m_6)$, wobei $P_i(p, m) = p^2 - m^2 + i\eta$.

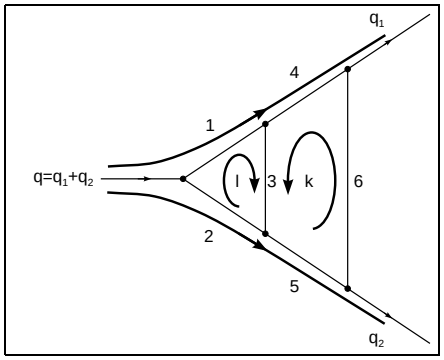


Abbildung 1.2: Impulsfluß durch die planare Dreipunkt-Funktion und Numerierung der Propagatoren

1.1.1.1 Wahl des Bezugssystems

Die Impulse werden im folgenden so interpretiert, daß ein reelles Teilchen mit Impuls $q = q_1 + q_2$ (also $q^2 > 0$) in zwei andere reelle Teilchen mit Impulsen q_1 und q_2 ($q_1^2 > 0$, $q_2^2 > 0$) zerfällt. Dies ist der physikalisch interessanteste Fall. Die Parallel-/Orthogonalraum-Methode nutzt nun aus, daß die Amplitude in Gl. (1.1) lorentzinvariant ist und daher in einem beliebigen Bezugssystem berechnet werden kann. Als geeignet erweist sich das Ruhesystem des zerfallenden Teilchens, das so gedreht wird, daß sich die beiden auslaufenden Zerfallsprodukte entlang einer Raumachse bewegen. Die Impulse q_1 und q_2 lassen sich dann mit

$$\begin{aligned} q_1^\mu &= (e_1, q_z, 0, 0) \\ q_2^\mu &= (e_2, -q_z, 0, 0) \end{aligned} \tag{1.2}$$

parametrisieren, wobei wegen der Symmetrie $q_1 \leftrightarrow q_2$ o.B.d.A. $q_z > 0$ gewählt werden kann. Die Bedingungen $q_{1,2}^2 > 0$ implizieren dann

$$e_1 > q_z > 0 \quad \text{und} \quad e_2 > q_z > 0\,. \tag{1.3}$$

Raumartige Impulse $q_1^2 < 0$ und/oder $q_2^2 < 0$ können auch berechnet werden, jedoch muß in der folgenden Rechnung an einigen Stellen eine Aussage über das Vorzeichen von z.B. $e_1 - q_z$ gemacht werden können. Ein raumartiger Impuls $q^2 = (q_1 + q_2)^2 < 0$ kann durch die Parametrisierung $q_1^\mu = (e, q_{z,1}, 0, 0)$, $q_2^\mu = (-e, q_{z,2}, 0, 0)$ behandelt werden.

1.1.1.2 Aufspaltung in Parallel- und Orthogonalraum

Die Impulse q_1 und q_2 spannen den sog. Parallelraum (parallel zu den äußeren Impulsen) in den 0- und 1-Komponenten der Raum-Zeit auf, der hier also zweidimensional ist. Die verbleibenden 2- und 3-Komponenten bilden den — ebenfalls zweidimensionalen — sog. Orthogonalraum, das orthogonale Komplement zum Parallelraum.

Teilt man auch die Loopimpulse l und k in Parallel- und Orthogonalraum-Komponenten auf

$$\begin{aligned} l^\mu &= (l_0, l_1, l_2, l_3) = (l_0, l_1, \vec{l}_\perp) \\ k^\mu &= (k_0, k_1, k_2, k_3) = (k_0, k_1, \vec{k}_\perp)\,, \end{aligned} \tag{1.4}$$

so hängt der Integrand nur noch von l_0, l_1, k_0, k_1, $l_\perp^2 = |\vec{l}_\perp|^2$, $k_\perp^2 = |\vec{k}_\perp|^2$ und $l_\perp k_\perp z = \vec{l}_\perp \cdot \vec{k}_\perp$ mit $z = \cos(\sphericalangle(\vec{l}_\perp, \vec{k}_\perp))$ ab, und man kann die Orthogonalraum-Komponenten l_2, l_3, k_2 und k_3 durch neue Variablen, im wesentlichen zweidimensionale Polarkoordinaten für $\vec{l}_\perp$ und $\vec{k}_\perp$, substituieren, in denen das Volumenelement

$$\int d^4l \int d^4k = \pi \int_{-\infty}^{+\infty} dl_0\, dk_0\, dl_1\, dk_1 \int_0^{+\infty} dt \int_0^{+\infty} ds \int_{-1}^{+1} \frac{dz}{\sqrt{1-z^2}} \tag{1.5}$$

mit

$$\begin{aligned} s &= l_\perp^2 \\ t &= k_\perp^2 \\ z &= \frac{\vec{l}_\perp \cdot \vec{k}_\perp}{|\vec{l}_\perp||\vec{k}_\perp|} = \cos\sphericalangle(\vec{l}_\perp, \vec{k}_\perp)\,. \end{aligned} \tag{1.6}$$

lautet. Über den Winkel im Orthogonalraum, den man z.B. als absolute Richtung von $\vec{l}_\perp$ interpretieren kann und von dem das Integral nicht abhängt, wurde bereits integriert.

1.1.1.3 Linearisierung der Propagatoren

Die Propagatoren $P_1, \ldots, P_6$ sind quadratische Funktionen in l_1 und k_1. Durch eine Transformation der Parallelraum-Komponenten

$$\begin{aligned} l_0 &\to l_0 + l_1 \\ k_0 &\to k_0 + k_1\,, \end{aligned} \tag{1.7}$$

die die Integrationsgrenzen unverändert läßt, werden dank der Metrik $(1, -1, -1, -1)$ des Minkowskiraums alle Propagatoren linear in l_1 und k_1, z.B.

$$\begin{aligned} P_1 &= (l_0 + e_1)^2 + 2(l_0 + e_1 - q_z)l_1 - q_z^2 - s - m_1^2 + i\eta \\ P_3 &= (l_0 + k_0)^2 + 2(l_0 + k_0)(l_1 + k_1) - s - t - \sqrt{s}\sqrt{t}\,z - m_3^2 + i\eta\,. \end{aligned} \tag{1.8}$$

1.1.1.4 Integration über z

Der einzige Propagator, der z, den Kosinus des Winkels zwischen $\vec{l}_\perp$ und $\vec{k}_\perp$, enthält, ist P_3, und zwar in der Form

$$P_3 = A + Bz + i\eta\,, \tag{1.9}$$

wobei $B < 0$ und A reell, aber von unbestimmtem Vorzeichen ist. Damit ergibt die z-Integration

$$\int_{-1}^{1} \frac{dz}{\sqrt{1-z^2}} \frac{1}{A + Bz + i\eta} = \frac{\pi}{\sqrt[c]{(A+i\eta)^2 - B^2}}\,. \tag{1.10}$$

$\sqrt[c]{x}$ bezeichnet dabei die komplexe Wurzelfunktion, die entgegen der häufig gewählten Konvention ihren Schnitt entlang der positiven reellen Achse hat.

1.1.1.5 Integration über l_1

Die Integration über l_1 wird mit Hilfe des Residuensatzes ausgeführt. Da der Integrand in alle Richtungen für $|l_1| \to \infty$ wie $1/|l_1|^3$ abfällt, trägt ein Halbkreis im Unendlichen nicht bei. Der Schnitt der Wurzel aus Gl. (1.10) diktiert jedoch, in welcher Halbebene die Kontur geschlossen werden kann, ohne den Schnitt zu kreuzen. Dazu sucht man die komplexen l_1-Werte, für die $(A+i\eta)^2 - B^2 = (A+i\eta+B)(A+i\eta-B)$ eine positive reelle Zahl ergibt. Man findet

$$\mathrm{Im}(l_1^{\mathrm{Schnitt}}) = -\frac{1}{2}\frac{\eta}{l_0 + k_0}\,, \tag{1.11}$$

woraus folgt, daß man für $l_0 + k_0 > 0$ in der oberen und für $l_0 + k_0 < 0$ in der unteren komplexen l_1-Halbebene schließen muß, um Werte von l_1 auf dem Schnitt zu vermeiden.

Nur die Propagatoren P_1 und P_2 haben Pole in der komplexen l_1-Ebene. Der Pol von P_1 liegt in der oberen Halbebene, wenn $l_0 + e_1 - q_z < 0$ ist, und in der unteren Halbebene, wenn $l_0 + e_1 - q_z > 0$ ist. Analog findet man die Lage des Pols von P_2 in Abhängigkeit vom Vorzeichen von $l_0 - e_2 - q_z$.

1.1.1.6 Integration über k_1

Auch die Integration über k_1 kann mit dem Residuensatz ausgeführt werden. Der Schnitt der Wurzel gibt, wie bei der l_1-Integration, in Abhängigkeit vom Vorzeichen von $l_0 + k_0$ die Halbebene vor, in der die Kontur ohne Kreuzen des Schnittes geschlossen werden kann. Für die Lage der Pole von P_4, P_5 und P_6 findet man Bedingungen der Form $k_0 - y \lessgtr 0$.

Diese drei Bedingungen, $l_0 + k_0 \gtrless 0$ vom Schnitt, $l_0 - x \lessgtr 0$ aus der l_1-Integration und $k_0 - y \lessgtr 0$ aus der k_1-Integration, sind teilweise unvereinbar. Sie lassen sich jedoch in den vier Dreiecken in Abb. 1.3 erfüllen.

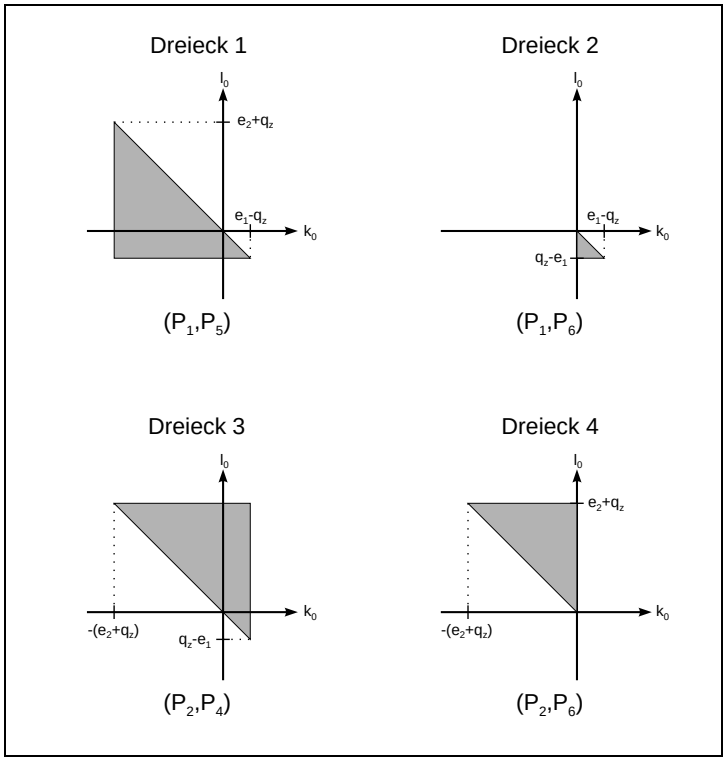


Abbildung 1.3: Einzelne Integrationsgebiete für die planare Dreipunkt-Funktion

1.1.1.7 Integration über s

Nach den Residuenintegrationen erhält man eine Vierfach-Integraldarstellung für die planare Dreipunkt-Funktion

$$V_P = \sum_{j=1}^{4} \iint\limits_{T_j} dl_0\, dk_0 \int\limits_0^\infty dt \int\limits_0^\infty ds\, C_j \, \frac{1}{(t+t_{0,j}-i\eta)(t+t'_{0,j}-i\eta)} \times \frac{1}{s+s_{0,j}-i\eta} \, \frac{1}{\sqrt[c]{(a_j t+b_j+i\eta+c_j s)^2-4st}}\,, \tag{1.12}$$

wobei die T_j die Dreiecke aus Abb. 1.3 bezeichnen und die Koeffizienten $s_{0,j}$, $t_{0,j}$, $t'_{0,j}$, a_j, b_j und c_j rationale Funktionen der Massen $m_1, \ldots, m_6$, Impulskomponenten e_1, e_2 und q_z und der verbleibenden Parallelraum-Variablen l_0 und k_0 sind. Für das Argument der Wurzel schreibt man (der Index j wird im folgenden weggelassen)

$$(at+b+i\eta+cs)^2-4st = c^2(s-\sigma_1(t))(s-\sigma_2(t)) = c^2 R(s,t)\,. \tag{1.13}$$

Vor der s-Integration sollte der Integrand in Real- und Imaginärteil zerlegt werden. Quellen für einen Imaginärteil sind zum einen negative Argumente der Wurzel, zum anderen Integrationen über den Pol bei $s = -s_{0,j}$, die mit den Sokhotsky-Plemelj-Relationen

[15]

$$\lim_{\eta\to+0}\int\limits_{s_1}^{s_2}\frac{1}{s+s_0-i\eta}\,f(s)\,ds=\text{P.V.}\int\limits_{s_1}^{s_2}\frac{1}{s+s_0}\,f(s)\,ds+i\pi\int\limits_{s_1}^{s_2}\delta(s+s_0)\,f(s)\,ds \tag{1.14}$$

abgespalten werden können. Das Argument der Wurzel $R(s,t)$ wird innerhalb einer Ellipse in der (s,t)-Ebene negativ. Sind für einen festen positiven Wert von t die Nullstellen $\sigma_1(t)$ und $\sigma_2(t)$ reell und positiv, so teilt man das s-Integral von 0 bis ∞ in die Intervalle $[0,\sigma_1]$, $[\sigma_1,\sigma_2]$ und $[\sigma_2,\infty[$ auf.

Die s-Integration kann mit Hilfe der Eulerschen Substitutionen [20] $s\to u$ (der Ersten in den Bereichen, in denen das Argument der Wurzel positiv ist, der Zweiten bei negativem Argument)

$$\begin{aligned} s &= \frac{u^2-\sigma_1\sigma_2}{2u-(\sigma_1+\sigma_2)} && \text{1. Eulersche Substitution}\\ s &= \frac{\sigma_2+\sigma_1 u^2}{u^2+1} && \text{2. Eulersche Substitution,} \end{aligned} \tag{1.15}$$

die die Wurzel rational machen, in beiden Fällen auf Integrale vom Typ

$$\int\frac{1}{Au^2+Bu+C}\,du=\begin{cases} \frac{1}{\sqrt{B^2-4AC}}\ln\left|\frac{2Au+B-\sqrt{B^2-4AC}}{2Au+B+\sqrt{B^2-4AC}}\right| \\ \qquad\text{falls } \Delta\equiv B^2-4AC>0 \\ \frac{2}{\sqrt{4AC-B^2}}\arctan\left(\frac{2Au+B}{\sqrt{4AC-B^2}}\right) \\ \qquad\text{falls } \Delta<0 \\ -\frac{2}{2Au+B} \\ \qquad\text{falls } \Delta=0 \end{cases} \tag{1.16}$$

zurückgeführt werden.

1.1.1.8 Integration über t

Nach der s-Integration und einer Partialbruchzerlegung von $1/(t+t_0-i\eta)/(t+t_0'-i\eta)$ erhält man für die letzte analytisch ausführbare Integration über t Integrale der Form

$$\int dt\,\frac{1}{t+t_0-i\eta}\,\frac{1}{\sqrt{R_0(t)}}\ln\left|\frac{s_0+(at+b)/c+\sqrt{R_0(t)}}{s_0+(at+b)/c-\sqrt{R_0(t)}}\right| \tag{1.17}$$

und

$$\int dt\,\frac{1}{t+t_0-i\eta}\,\frac{1}{\sqrt{-R_0(t)}}\arctan\left(\frac{s_0+(at+b)/c}{\sqrt{-R_0(t)}}\right) \tag{1.18}$$

mit

$$R_0(t)=R(-s_0,t)=(s_0+\sigma_1(t))(s_0+\sigma_2(t)), \tag{1.19}$$

einer quadratischen Funktion in t, sowie einfachere Terme ohne Logarithmus bzw. Arkustangens.

Auch die t-Integration läßt sich nach Abtrennung des Imaginärteils aus der Integration über den Pol $1/(t+t_0-i\eta)$ mit den Sokhotsky-Plemelj-Relationen mit Hilfe der Eulerschen Substitutionen $t \to v$ ausführen. Dies führt auf Integrale vom Typ

$$\mathrm{P.V.} \int \frac{\ln|v^2+pv+q|}{v^2+rv+s}\,dv\,, \tag{1.20}$$

die sich durch Dilogarithmen und Clausen-Funktionen [64] ausdrücken lassen.

Die verbleibenden Integrationen über l_0 und k_0 über endliche Gebiete können nur noch numerisch ausgeführt werden.

1.1.2 Divergente Zweiloop-Dreipunkt-Funktionen

Die Berechnung divergenter Integrale erfordert eine Regularisierung der Divergenzen, z.B. durch Abschneiden der Loopintegration für alle $k^2 > \Lambda^2$, das Pauli-Villars-Verfahren [72] oder analytische Regularisierung [83], wobei sich heute überwiegend die dimensionale Regularisierung [14, 61] durchgesetzt hat, da sie z.B. die Eichinvarianz von Feynman-Amplituden nicht verletzt. Die Integrale werden dann nicht mehr in vier Dimensionen, sondern in $D = 4 - 2\varepsilon$ Dimensionen berechnet, und die Divergenzen zeigen sich in der Laurent-Entwicklung der Amplitude in Form von Polen in ε, wenn $D \to 4$, also $\varepsilon \to 0$, geht.

Das Volumenelement in Gl. (1.5) ist aber offensichtlich nur für Integrale in einer vierdimensionalen Raum-Zeit ausgelegt. In D Dimensionen lautet die Zerlegung des Volumenelementes in Parallel- und Orthogonalraum-Komponenten entsprechend

$$\begin{aligned}\int d^Dl \int d^Dk &= V \int\limits_{-\infty}^{+\infty} dl_0\,dk_0\,dl_1\,dk_1 \int\limits_0^{+\infty} k_\perp^{D-3} dk_\perp \int\limits_0^{+\infty} l_\perp^{D-3} dl_\perp \int\limits_{-1}^{+1} (1-z^2)^{\frac{D-5}{2}} dz \\ &= \frac{V}{4} \int\limits_{-\infty}^{+\infty} dl_0\,dk_0\,dl_1\,dk_1 \int\limits_0^{+\infty} t^{\frac{D-4}{2}} dt \int\limits_0^{+\infty} s^{\frac{D-4}{2}} ds \int\limits_{-1}^{+1} (1-z^2)^{\frac{D-5}{2}} dz\,. \end{aligned} \tag{1.21}$$

Dabei ist V, das man aus der Integration über die Winkel erhält, von denen das Integral nicht abhängt, das Produkt der Oberflächen je einer Hyperkugel in $D-2$ und $D-3$ Dimensionen:

$$V = K_{D-2} K_{D-3} \tag{1.22}$$

mit

$$K_D = \frac{2\pi^{D/2}}{\Gamma(D/2)}\,. \tag{1.23}$$

Für $D = 4$ ist $V/4 = \pi$, was mit dem Vorfaktor aus Gl. (1.5) übereinstimmt. Falls der übrige Integrand nicht von z abhängt, gibt die z-Integration

$$\int\limits_{-1}^{+1} (1-z^2)^{\frac{D-5}{2}} dz = \sqrt{\pi}\,\frac{\Gamma((D-3)/2)}{\Gamma((D-2)/2)} = \frac{K_{D-3}}{K_{D-2}}\,, \tag{1.24}$$

und man erhält das quadrierte Einloop-Ergebnis $(K_{D-3})^2$ für das Volumenelement.

In der Regel enthält der Integrand aber z. Die erste nicht-triviale Integration in D Dimension hat dann als Verallgemeinerung von Gl. (1.10) das Ergebnis

$$\int_{-1}^{+1}(1-z^2)^{\frac{D-5}{2}}\frac{1}{A+Bz}dz = \frac{\sqrt{\pi}\,\Gamma(\frac{D-3}{2})\,{}_2F_1(\frac{1}{2},1;\frac{D-2}{2};\frac{B^2}{A^2})}{A\,\Gamma(\frac{D-2}{2})}\,. \tag{1.25}$$

Das Auftreten der hypergeometrischen Funktion ${}_2F_1$ macht weitere analytische Integrationen nach den Residuenintegrationen de facto unmöglich.

Die Parallel-/Orthogonalraum-Methode ist also nicht direkt zum Berechnen von divergenten Feynman-Graphen in dimensionaler Regularisierung geeignet. Die Strategie, die im folgenden bei divergenten Integralen angewendet wird, lautet statt dessen: Finde geeignete Abzugsterme, die subtrahiert und wieder addiert werden, mit der Eigenschaft, daß

- die Differenz endlich ist und in vier Dimensionen berechnet werden kann,
- die Abzugsterme eine ähnliche Struktur wie der divergente Term haben (sich z.B. nur in Massen oder äußeren Impulsen unterscheiden) und deshalb mit derselben Methode berechnet werden können und
- sie einfach genug sind, um mit anderen Methoden in dimensionaler Regularisierung berechnet zu werden.

Die Abzugsterme müssen dabei nicht vollständig analytisch in D Dimensionen berechnet werden können, aber die Koeffizienten der *divergenten* Terme in der Laurent-Entwicklung sollten analytisch bekannt sein (Ausnahme in Abschnitt 1.4).

1.2 Ultraviolett divergente skalare Diagramme

Als erstes Beispiel divergenter Dreipunkt-Funktionen soll das skalare Diagramm aus Abb. 1.4 mit der Parallel-/Orthogonalraum-Methode und Abzugstermen berechnet werden (vgl. auch Abschnitt 1.5.9 für eine Methode, die für diese Art von Diagrammen besser geeignet ist).

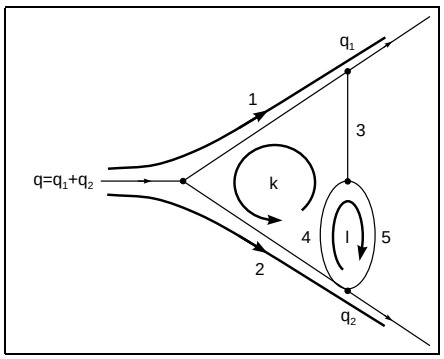


Abbildung 1.4: Impulsfluß durch die UV divergente Dreipunkt-Funktion mit einem Zweipunkt-Untergraphen und Numerierung der Propagatoren

1.2.1 Bestimmung der Abzugsterme

Als Impulsfluß durch das Diagramm wird

$$\begin{aligned} P_1 &= (k-q_1)^2 - m_1^2 + i\eta \\ P_2 &= (k+q_2)^2 - m_2^2 + i\eta \\ P_3 &= k^2 - m_3^2 + i\eta \\ P_4 &= (l+k)^2 - m_4^2 + i\eta \\ P_5 &= l^2 - m_5^2 + i\eta \end{aligned} \tag{1.26}$$

gewählt. Die Integration des Diagramms

$$V_{\mathrm{UV}} = \int d^D l \int d^D k \, \frac{1}{P_1 P_2 P_3 P_4 P_5} \tag{1.27}$$

ist in vier Dimensionen logarithmisch divergent, wenn $l \to \infty$ geht, da im Nenner vier Potenzen von l stehen, während das Integrationsmaß im Zähler ebenfalls vier Potenzen von l beisteuert. Das Integral verhält sich also wie $\int dl/l$. Man bezeichnet dies als Ultraviolett-Divergenz. Gesucht wird nun ein Abzugsterm, der das *Powercounting* in l — das Bestimmen des Divergenzgrades durch Abzählen der Potenzen von l im Zähler und Nenner — um (mindestens) eine Potenz verbessert, während die konvergente Integration in k unverändert bleiben soll. Als solchen kann man das Diagramm wählen, bei dem die l-Propagatoren masselos sind, d.h.

$$\begin{aligned} \tilde{P}_1 &= P_1 \\ \tilde{P}_2 &= P_2 \\ \tilde{P}_3 &= P_3 \\ \tilde{P}_4 &= P_4\big|_{m_4=0} = (l+k)^2 + i\eta \\ \tilde{P}_5 &= P_5\big|_{m_5=0} = l^2 + i\eta \,. \end{aligned} \tag{1.28}$$

In der Differenz ist

$$\begin{aligned} &\int d^D l \left(\frac{1}{P_4 P_5} - \frac{1}{\tilde{P}_4 \tilde{P}_5} \right) = \\ &\qquad \int d^D l \, \frac{(l+k)^2 m_5^2 + l^2 m_4^2 - m_4^2 m_5^2}{((l+k)^2 - m_4^2 + i\eta)(l^2 - m_5^2 + i\eta)((l+k)^2 + i\eta)(l^2 + i\eta)} \end{aligned} \tag{1.29}$$

endlich, wenn $D \to 4$ gesetzt wird, und es gilt

$$\begin{aligned} V_{\mathrm{UV}} &= \int d^4 l \int d^4 k \, \frac{1}{P_1 P_2 P_3} \left(\frac{1}{P_4 P_5} - \frac{1}{\tilde{P}_4 \tilde{P}_5} \right) + \int d^D l \int d^D k \, \frac{1}{P_1 P_2 P_3 \tilde{P}_4 \tilde{P}_5} + \mathcal{O}(\varepsilon) \\ &= V_{\mathrm{UV,endl}} + V_{\mathrm{UV,div}} + \mathcal{O}(\varepsilon) \,. \end{aligned} \tag{1.30}$$

Bei der Erzeugung von Abzugstermen für ultraviolett divergente Integrale ist darauf zu achten, daß keine künstlichen Infrarot-Divergenzen (Abschnitt 1.3) eingeführt werden. Dies ist hier gewährleistet.

1.2.2 Berechnung des endlichen Anteils

Analog zu Abschnitt 1.1.1 können nun in $V_{UV,\mathrm{endl}}$ die Propagatoren (auch $\tilde{P}_4$ und $\tilde{P}_5$) in l_1 und k_1 durch die Transformation $l_0 \to l_0 + l_1$ und $k_0 \to k_0 + k_1$ linearisiert werden. Die Integration über z (nur in P_3 enthalten) gibt eine Wurzel mit einem Schnitt in der komplexen l_1- bzw. k_1-Ebene, dessen Lage vom Vorzeichen von $l_0 + k_0$ abhängt. Um die l_1- und k_1-Integrationen mit Hilfe des Residuensatzes ausführen zu können, schließt man daher die Kontur für $l_0 + k_0 > 0$ in der oberen, für $l_0 + k_0 < 0$ in der unteren Halbebene. *Powercounting* zeigt, daß für die l_1-Integration der Abzugsterm noch nicht benötigt wird, die Integrationen sind individuell endlich. Man findet einen Beitrag von (P_1, P_5) (bzw. $(P_1, \tilde{P}_5)$) im Bereich $l_0 + k_0 > 0$, $l_0 < 0$ und $k_0 - e_1 + q_z < 0$ sowie einen Beitrag von (P_2, P_5) (bzw. $(P_2, \tilde{P}_5)$) im Bereich $l_0 + k_0 < 0$, $l_0 > 0$ und $k_0 + e_2 + q_z > 0$, also zwei Dreiecke (Abb. 1.5).

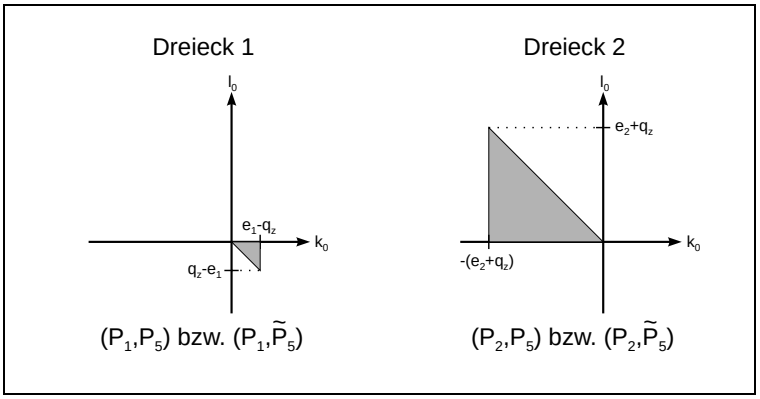


Abbildung 1.5: Dreiecke, die bei der UV divergenten Dreipunkt-Funktion mit einem Zweipunkt-Untergraphen beitragen

Damit erhält man als Zwischenergebnis eine Vierfach-Integraldarstellung für $V_{\mathrm{UV,endl}}$:

$$V_{\mathrm{UV,endl}} = \sum_{j=1}^{2} \iint\limits_{T_j} dl_0\, dk_0 \int\limits_0^\infty dt \int\limits_0^\infty ds\, C_j \frac{1}{(t + t_{0,j} - i\eta)(t + t'_{0,j} - i\eta)} \left(\frac{1}{\sqrt[c]{(a_j t + b_j + i\eta + c_j s)^2 - 4st}} - \frac{1}{\sqrt[c]{(a_j t + \tilde{b}_j + i\eta + c_j s)^2 - 4st}} \right). \quad (1.31)$$

Die Indizes j werden im folgenden wieder weggelassen. Im Vergleich zu Gl. (1.12) fehlt hier der Term mit $1/(s + s_0 - i\eta)$, und die s-Integration wäre ohne den Abzugsterm divergent, da

$$\sqrt[c]{(at + b + i\eta + cs)^2 - 4st} = (at + b + cs - 2\frac{t}{c} + \mathcal{O}(\frac{1}{s})) \quad (1.32)$$

für $s \to \infty$ ($c < 0$ und $\sqrt[c]{x} = -\sqrt{|x|}$ im Integrationsgebiet). Die Differenz der beiden inversen Wurzeln hingegen fällt wie $1/s^2$ für $s \to \infty$ ab und ist daher integrabel. Entscheidend dafür ist, daß der Vorfaktor a in beiden Termen derselbe ist. Dies ist dadurch garantiert, daß a und c nur von l_0, k_0 und den Komponenten der äußeren Impulse abhängen, während die Abhängigkeit von den Massen nur in t_0, t'_0, b und $\tilde{b}$ steckt.

Integrale analog zu Gl. (1.31) treten auch wieder in Abschnitt 1.5.6 auf. Zerlegt man sie wieder wie in Abschnitt 1.1.1.7 in Real- und Imaginärteil, so erweist es sich zur Vermeidung zusätzlicher Fallunterscheidungen als vorteilhaft, die Terme der Differenz einzeln zu integrieren. So erhält man für die s-Integration

$$\begin{aligned}\mathcal{S} &= \int_0^\Lambda ds\, \frac{|c|}{\sqrt[c]{(at+b+i\eta+cs)^2-4st}} \\ &= \int_0^\Lambda ds\, \frac{1}{\sqrt[c]{s-(\sigma_1+\sigma_2)s+\sigma_1\sigma_2}} \\ &= \int_0^\Lambda ds\, \frac{1}{\sqrt[c]{(s-\sigma_1)(s-\sigma_2)}}\,. \end{aligned} \tag{1.33}$$

Dabei ist Λ ein Abschneideparameter mit $\sigma_1 < \sigma_2 < \Lambda$, wenn $\sigma_{1/2}$ reell sind. Später wird in der Differenz der Grenzwert $\Lambda \to \infty$ genommen. Zuerst wird jedoch die komplexe Wurzel aufgelöst, wobei man drei Klassen unterscheiden muß (siehe [30] für Details):

- Klasse A, $b > 0$ und $bc/(1-ac) < t < \infty$ oder $b < 0$ und $0 < t < \infty$: $\sigma_1 = \sigma_2^*$ oder $\sigma_1 < 0$ und $\sigma_2 < 0$
- Klasse B, $b > 0$ und $-b/a < t < bc/(1-ac)$: $0 < \sigma_1 < \sigma_2$
- Klasse C, $b > 0$ und $0 < t < -b/a$: $0 < \sigma_1 < \sigma_2$

Ein Imaginärteil tritt nur bei Klasse B und C auf, wenn $\sigma_1 < s < \sigma_2$ möglich ist:

$$\mathrm{Im}(\mathcal{S}^{B,C}) = -\int_{\sigma_1}^{\sigma_2} ds\, \frac{1}{\sqrt{(s-\sigma_1)(\sigma_2-s)}} = -\pi\,. \tag{1.34}$$

Für den Realteil unterscheidet man zunächst

$$\begin{aligned}\mathrm{Re}(\mathcal{S}^A) &= -\int_0^\Lambda ds\, \frac{1}{\sqrt{s-(\sigma_1+\sigma_2)s+\sigma_1\sigma_2}} \\ &= \ln\Big(2\sqrt{\sigma_1\sigma_2}-(\sigma_1+\sigma_2)\Big) - \ln\Big(2\Lambda+2\sqrt{\Lambda^2-(\sigma_1+\sigma_2)\Lambda+\sigma_1\sigma_2}-(\sigma_1+\sigma_2)\Big) \\ &= \ln\Big(2\sqrt{\sigma_1\sigma_2}-(\sigma_1+\sigma_2)\Big) - \ln(4\Lambda) + \mathcal{O}(\frac{1}{\Lambda})\,, \end{aligned} \tag{1.35}$$

$$\begin{aligned}\mathrm{Re}(\mathcal{S}^B) &= -\int_0^{\sigma_1} ds\, \frac{1}{\sqrt{(\sigma_1-s)(\sigma_2-s)}} - \int_{\sigma_2}^\Lambda ds\, \frac{1}{\sqrt{(s-\sigma_1)(s-\sigma_2)}} \\ &= \Big[\ln\Big((\sigma_1+\sigma_2)-2\sqrt{\sigma_1\sigma_2}\Big) - \ln(\sigma_2-\sigma_1)\Big] + \Big[\ln(\sigma_2-\sigma_1) - \ln(4\Lambda+\mathcal{O}(\Lambda^0))\Big] \\ &= \ln\Big((\sigma_1+\sigma_2)-2\sqrt{\sigma_1\sigma_2}\Big) - \ln(4\Lambda) + \mathcal{O}(\frac{1}{\Lambda}) \quad \text{und} \end{aligned} \tag{1.36}$$

$$\begin{aligned}
\mathrm{Re}(\mathcal{S}^C) &= +\int_0^{\sigma_1} ds\, \frac{1}{\sqrt{(\sigma_1-s)(\sigma_2-s)}} - \int_{\sigma_2}^{\Lambda} ds\, \frac{1}{\sqrt{(s-\sigma_1)(s-\sigma_2)}} \\
&= \left[\ln(\sigma_2-\sigma_1) - \ln\left((\sigma_1+\sigma_2) - 2\sqrt{\sigma_1\sigma_2}\right)\right] + \left[\ln(\sigma_2-\sigma_1) - \ln(4\Lambda + \mathcal{O}(\Lambda^0))\right] \\
&= \ln\left((\sigma_2-\sigma_1)^2/\left((\sigma_1+\sigma_2) - 2\sqrt{\sigma_1\sigma_2}\right)\right) - \ln(4\Lambda + \mathcal{O}(\Lambda^0)) \\
&= \ln\left((\sigma_1+\sigma_2) + 2\sqrt{\sigma_1\sigma_2}\right) - \ln(4\Lambda) + \mathcal{O}(\frac{1}{\Lambda})\,. \qquad (1.37)
\end{aligned}$$

Die Argumente der Logarithmen sind in allen Fällen reell und positiv. Da

$$\begin{aligned}
\sigma_1+\sigma_2 &= 2\frac{t(2-ac)-bc}{c^2} \\
\sigma_1\sigma_2 &= \left(\frac{at+b}{c}\right)^2, \qquad (1.38)
\end{aligned}$$

ist

$$\sqrt{\sigma_1\sigma_2} = \begin{cases} +\frac{at+b}{c} & \text{Klasse A und B} \\ -\frac{at+b}{c} & \text{Klasse C,} \end{cases} \qquad (1.39)$$

und man kann die Ergebnisse der Klassen zusammenfassen

$$\mathrm{Re}(\mathcal{S}^{A,B,C}) = \ln\left|4\frac{(1-ac)t-bc}{c^2}\right| - \ln(4\Lambda) + \mathcal{O}(\frac{1}{\Lambda})\,. \qquad (1.40)$$

Damit erhält man

$$\begin{aligned}
&\int_0^{\infty} ds \left(\frac{1}{\sqrt[c]{(at+b+i\eta+cs)^2-4st}} - \frac{1}{\sqrt[c]{(at+\tilde{b}+i\eta+cs)^2-4st}}\right) \\
&= \frac{1}{|c|}\left[\ln\left|\frac{t-\frac{bc}{1-ac}}{t-\frac{\tilde{b}c}{1-ac}}\right| - i\pi\left(\Theta\Big(0<t<\frac{bc}{1-ac}\Big) - \Theta\Big(0<t<\frac{\tilde{b}c}{1-ac}\Big)\right)\right], \qquad (1.41)
\end{aligned}$$

wobei die Abhängigkeit von Λ im Limes $\Lambda\to\infty$ herausfällt. Die t-Integration läßt sich nun nach Partialbruchzerlegung und Abspalten des Imaginärteils mit Gl. (1.14) durch Dilogarithmen ausdrücken [30, 64]:

$$\mathrm{P.V.}\int dt\, \frac{\ln|t-\beta|}{t-t_0} = \begin{cases} \ln|t_0-\beta|\ln|t-t_0| - \mathrm{Re}\left(\mathrm{Li}_2\left(\frac{t-t_0}{\beta-t_0}\right)\right) & \text{für } \beta\neq t_0 \\ \frac{1}{2}\ln^2|t-\beta| & \text{für } \beta = t_0\,. \end{cases} \qquad (1.42)$$

Zusätzlich gibt es noch einfachere Integrale rationaler Funktionen von t aus den Imaginärteilen.

1.2.3 Berechnung des divergenten Anteils

Zur Berechnung des divergenten Anteils wird ein semi-numerisches Verfahren benutzt, das auch in Abschnitt 1.4.2 zur Anwendung kommt. Zunächst wird die k-Integration ausgeführt:

$$\begin{aligned}
\int d^Dk\, \frac{1}{(k^2+i\eta)((l+k)^2+i\eta)} &= i\pi^{2-\varepsilon}\frac{\Gamma(\varepsilon)\Gamma^2(1-\varepsilon)}{\Gamma(2-2\varepsilon)}(-l^2-i\eta)^{-\varepsilon} \\
&= \left(\frac{i\pi^2}{\varepsilon} - i\pi^2(\gamma-2+\ln\pi) + \mathcal{O}(\varepsilon)\right)(-l^2-i\eta)^{-\varepsilon}\,. \qquad (1.43)
\end{aligned}$$

Da die Integration über l selbst endlich ist (streng genommen muß man gleichförmige Konvergenz zeigen), kann man diese vorher in ε entwickeln:

$$\begin{aligned} V_{\mathrm{UV,div}} &= C \int d^D l \, \frac{1}{P_1 P_2 P_3} (-l^2 - i\eta)^{-\varepsilon} \\ &= C \, K_{D-2} \int dl_0 \int dl_1 \int ds \, (1 - \varepsilon \ln s + \mathcal{O}(\varepsilon^2) \\ &\quad \times \frac{1 - \varepsilon \ln(-l^2 - i\eta) + \mathcal{O}(\varepsilon^2))}{P_1 P_2 P_3} \,, \end{aligned} \tag{1.44}$$

mit $C = i\pi^{2-\varepsilon}\Gamma(\varepsilon)\Gamma^2(1-\varepsilon)/\Gamma(2-2\varepsilon)$ aus Gl. (1.43) und K_{D-2} aus Gl. (1.23). Während der $\mathcal{O}(\varepsilon^0)$-Anteil der l-Integration analytisch angegeben werden kann, wird der $\mathcal{O}(\varepsilon)$-Anteil (der wegen der Multiplikation mit C einen Beitrag zur Ordnung $\mathcal{O}(\varepsilon^0)$ von $V_{\mathrm{UV,div}}$ gibt) numerisch bestimmt. Analog zur Berechnung von Zweiloop-Dreipunkt-Funktionen kann man nach einer Linearisierung der Propagatoren durch die Transformation $l_0 \to l_0 + l_1$ die l_1-Integration mit Hilfe des Residuensatzes ausführen. Der Term mit $\ln(-l_0^2 - 2l_0 l_1 + s - i\eta)$ im Zähler bestimmt durch den Schnitt des Logarithmus auf der negativen reellen Achse die Halbebene, in der die Integrationskontur geschlossen werden muß, nämlich für $l_0 > 0$ in der oberen und für $l_0 < 0$ in der unteren Halbebene. Als Realteil für die s-Integration erhält man dann Integrale der Form

$$\mathrm{P.V.} \int ds \frac{\ln|s + s_1|}{s + s_0} \,, \tag{1.45}$$

die sich durch Dilogarithmen ausdrücken lassen [30]. Die verbleibende Integration über l_0 über endliche Intervalle wird numerisch ausgeführt.

1.2.4 Andere ultraviolett divergente skalare Diagramme

Außer der Topologie in Abb. 1.4 treten noch die Topologien in Abb. 1.6 mit einem UV divergenten Zweipunkt-Untergraphen auf. Abb. 1.6a läßt sich für $m_3 \neq m_4$ durch Partialbruchzerlegung in den Propagatoren P_3 und P_4 auf zwei Graphen vom Typ Abb. 1.4 reduzieren. Der Fall $m_3 = m_4$ ist effektiv Graph Abb. 1.4 mit einem quadrieren Propagator, den man durch Ableiten nach m_3 unmittelbar vor der numerischen Integration aus obigem Ergebnis erhalten kann. Dabei ist zu beachten, daß in Gl. (1.42) sowohl β, t_0 als auch der Vorfaktor der Partialbruchzerlegung von m_3 abhängen können. Alternativ dazu kann man alle Schritte der Parallel-/Orthogonalraum-Integration auch mit einem quadrierten Propagator durchführen. Quadrierte Pole tragen dann nicht zur Residuenintegration bei, und zusätzlich zu Gl. (1.42) treten Integrale vom Typ

$$\mathrm{P.V.} \int dt \, \frac{\ln|t-\beta|}{(t-t_0)^2} = \begin{cases} \frac{1}{(t_0-\beta)(t-t_0)} \left((t-t_0) \ln|t-t_0| - (t-\beta) \ln|t-\beta| \right) & \text{für } \beta \neq t_0 \\ -\frac{1}{t-\beta} (\ln|t-\beta| + 1) & \text{für } \beta = t_0 \end{cases} \tag{1.46}$$

auf.

Die Topologie in Abb. 1.6b ist schwieriger, da sie — wenn der Impulsfluß so gewählt wird, daß die Propagatoren im Zweipunkt-Untergraphen die Impulse l und $l+k$ haben — sowohl für $l \to \infty$ als auch für $l + k \to \infty$ divergent wird. Diese Topologie läßt sich aber ohnehin effektiver mit dem Verfahren aus [75] (vgl. Abschnitt 1.5.9) berechnen, so daß hier auf Details verzichtet wird.

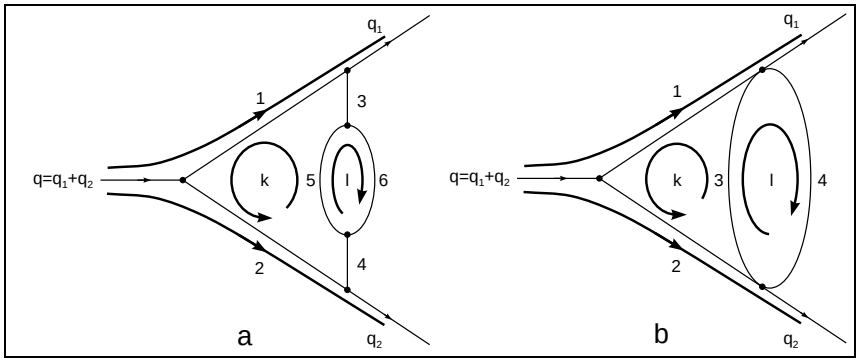


Abbildung 1.6: Andere UV divergente Dreipunkt-Funktionen mit einem Zweipunkt-Untergraphen

1.3 Infrarot divergente Diagramme

Infrarote Divergenzen treten in einigen Diagrammen auf, wenn der Integrand für einen bestimmten Wert eines Loopimpulses $k = k_{\mathrm{IR}} + \Delta k$ in der Umgebung von k_{IR} mindestens wie $1/(\Delta k)^4$ divergiert und daher nicht mehr in vier Dimensionen integrabel ist. Üblicherweise transformiert man diesen Loopimpuls so, daß $k_{\mathrm{IR}} = 0$ ist und dann die Infrarot-Divergenz für $k \to 0$ auftritt.

Im Gegensatz zu UV-Divergenzen, deren Existenz für große Loopimpulse nur von der Zählerstruktur und der Topologie, also dem Nenner, abhängt und sofort durch *Powercounting* festgestellt werden kann, treten IR-Divergenzen nur für bestimmte Werte der Massen der inneren Teilchen und der äußeren Impulse auf. Die Entscheidung, ob eine Infrarot-Divergenz vorliegt, ist daher mittels IR-*Powercounting* erst möglich, wenn Massen und Impulse bekannt sind. Das einfachste nicht-triviale Beispiel ist die Einloop-Dreipunkt-Funktion aus Abb. 1.7 für den Fall $q_1^2 = m_1^2$, $q_2^2 = m_2^2$ und $m_3 = 0$ mit den Propagatoren

$$\begin{aligned} P_1 &= (k+q_1)^2 - m_1^2 + i\eta \;=\; k^2 + 2k\cdot q_1 + i\eta \\ P_2 &= (k-q_2)^2 - m_2^2 + i\eta \;=\; k^2 - 2k\cdot q_2 + i\eta \\ P_3 &= k^2 - m_3^2 + i\eta \;=\; k^2 + i\eta\,, \end{aligned} \tag{1.47}$$

die für $k \to 0$, wo man k^2 gegenüber $k \cdot q_{1/2}$ vernachlässigen kann, wie $1/k^4$ divergiert.

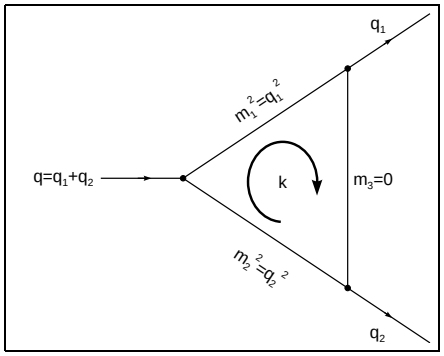


Abbildung 1.7: Infrarot divergente Einloop-Dreipunkt-Funktion

Die klassische Methode der Behandlung von Infrarot-Singularitäten in der QED und im Standardmodell arbeitet statt mit masselosen Teilchen mit einer Masse $m_3 = \lambda$ als Regulator und betrachtet dann den Grenzwert $\lambda \to 0$ [93]. Infrarot-Divergenzen können aber auch wie UV-Divergenzen mit Hilfe von dimensionaler Regularisierung [14] in $D = 4 - 2\varepsilon$ Dimensionen kontrolliert werden. Diese Methode wird üblicherweise in der QCD verwendet. Es ist dabei nicht notwendig, zwischen einem $\varepsilon_{\mathrm{UV}} > 0$ für UV-Divergenzen und einem $\varepsilon_{\mathrm{IR}} < 0$ für Infrarot-Divergenzen zu unterscheiden. Die Schwierigkeit von IR-Divergenzen gegenüber UV-Divergenzen besteht darin, daß sich nicht auf einfache algorithmische Weise Abzugsterme erzeugen lassen, die sich auch mit der Parallel-/Orthogonalraum-Methode berechnen lassen [52].

1.3.1 IR-Divergenz in einem Loopimpuls

Auf Zweiloop-Niveau zeigt z.B. der Graph aus Abb. 1.8 mit $q_1^2 = m_4^2$, $q_2^2 = m_5^2$ und $m_6 = 0$ eine Infrarot-Divergenz analog zum Einloop-Graphen in Abb. 1.7, wenn $k \to 0$ geht. Die Propagatoren lauten damit hier

$$\begin{aligned} P_1 &= (l+q_1)^2 - m_1^2 + i\eta \\ P_2 &= (l-q_2)^2 - m_2^2 + i\eta \\ P_3 &= (l+k)^2 - m_3^2 + i\eta \\ P_4 &= k^2 - 2k\cdot q_1 + i\eta \\ P_5 &= k^2 + 2k\cdot q_2 + i\eta \\ P_6 &= k^2 + i\eta\,. \end{aligned} \tag{1.48}$$

Der Fall, daß auch die Integration in l eine Divergenz für $l \to 0$ hat, wird in Abschnitt 1.3.2 behandelt.

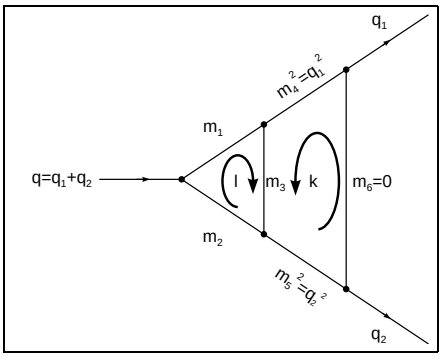


Abbildung 1.8: Die in einem Loopimpuls infrarot divergente planare Zweiloop-Dreipunkt-Funktion

1.3.1.1 Bestimmung der Abzugsterme

Einen Abzugsterm, der das *Powercounting* in der Differenz für $k \to 0$ um eine Potenz verbessert und so das Integral in $D = 4$ Dimensionen endlich macht, findet man in dem Integral, bei dem P_3 durch

$$\tilde{P}_3 = P_3\big|_{k=0} = l^2 - m_3^2 + i\eta \tag{1.49}$$

ersetzt wurde, denn die Differenz

$$\frac{1}{P_3} - \frac{1}{\tilde{P}_3} = -\frac{k^2 + 2k \cdot l}{P_3 \tilde{P}_3} \tag{1.50}$$

gibt einen zusätzlichen Faktor $\sim k \cdot l$ im Zähler, wenn $k \to 0$ geht. Damit gilt

$$\begin{aligned} V_{\mathrm{IR}} &= \int d^4l \int d^4k \, \frac{1}{P_1 P_2} \left(\frac{1}{P_3} - \frac{1}{\tilde{P}_3} \right) \frac{1}{P_4 P_5 P_6} \\ &\quad + \int d^Dl \, \frac{1}{P_1 P_2 \tilde{P}_3} \int d^Dk \, \frac{1}{P_4 P_5 P_6} + \mathcal{O}(\varepsilon) \\ &= V_{\mathrm{IR,endl}} + V_{\mathrm{IR,div}} + \mathcal{O}(\varepsilon) \, . \end{aligned} \tag{1.51}$$

Der Abzugsterm faktorisiert in l und k, wobei der Term in k offensichtlich auch infrarot divergent ist, da es sich um den Einloop-Graphen aus Gl. (1.47) handelt, und somit die gesamte Infrarot-Divergenz von V_{IR} enthält.

1.3.1.2 Berechnung des endlichen Anteils

Da es sich bei infraroten Divergenzen immer um Endpunkt-Singularitäten handelt, tritt die Divergenz für $k \to 0$ erst in der letzten Integration über eine Komponente von k auf. Dies ist nach dem Vorgehen aus Abschnitt 1.1.1 k_0, über das numerisch integriert wird. Das bedeutet, daß alle Schritte für die vorherigen analytischen Integrationen für den ursprünglichen divergenten Term und seinen ebenfalls infrarot divergenten Abzugsterm unabhängig voneinander ausgeführt werden können. Der Abzugsterm garantiert dabei, daß die numerische Integration für $k_0 \to 0$ endlich bleibt und einen wohldefinierten Grenzwert hat. Da die Infrarot-Divergenz nur logarithmisch ist, stellt dies numerisch kein Problem dar.

An den analytischen Integrationen zur Berechnung des Beitrags von der planaren Dreipunkt-Funktion (mit P_3) ändert sich daher nichts gegenüber Abschnitt 1.1.1.

Obwohl alle analytischen Integrationen für den ursprünglichen und den Abzugsterm unabhängig voneinander ausgeführt werden können, müssen dennoch dieselben Schritte (z.B. Variablentransformationen) angewendet werden, damit die numerische Integration für $k_0 \to 0$ endlich bleibt. Das bedeutet für den Abzugsterm (mit $\tilde{P}_3$):

- Linearisierung der Propagatoren durch $l_0 \to l_0 + l_1$, $k_0 \to k_0 + k_1$.
- Die z-Integration ist trivial und gibt einen Faktor π, da kein Propagator von z abhängt.
- Die l_1- und k_1-Integrationen werden mit Hilfe des Residuensatzes ausgeführt. Da es aus der z-Integration keine Wurzel mit einem Schnitt in der komplexen l_1- und k_1-Ebene gibt, hat man die Freiheit, die Kontur beliebig zu schließen. Eine Möglichkeit ist, so wie für die planare Dreipunkt-Funktion zu schließen, also für $l_0 + k_0 > 0$ in der oberen und für $l_0 + k_0 < 0$ in der unteren Halbebene. Dann erhält man aus der l_1-Integration von den Propagatoren, die l enthalten, Bedingungen der Form $l_0 \lesseqgtr x$ und aus der k_1-Integration von den k-Propagatoren Bedingungen der Form $k_0 \lesseqgtr y$. Daraus resultieren wieder beitragende Dreiecke, die in Tabelle 1.1 bzw. Abb. 1.9 aufgeführt sind. Die Vereinigung der Dreiecksgebiete stimmt mit der Vereinigung der Dreiecke der planaren Dreipunkt-Funktion aus Abb. 1.3 überein.

Propagatoren	Bedingungen		
(P_1, P_5)	$l_0 + k_0 < 0$	$l_0 + e_1 - q_z > 0$	$k_0 + e_2 + q_z > 0$
(P_1, P_6)	$l_0 + k_0 < 0$	$l_0 + e_1 - q_z > 0$	$k_0 > 0$
(P_2, P_4)	$l_0 + k_0 > 0$	$l_0 - e_2 - q_z < 0$	$k_0 - e_1 + q_z < 0$
(P_2, P_6)	$l_0 + k_0 > 0$	$l_0 - e_2 - q_z < 0$	$k_0 < 0$
(P_3, P_4)	$l_0 + k_0 > 0$	$l_0 < 0$	$k_0 - e_1 + q_z < 0$
(P_3, P_5)	$l_0 + k_0 < 0$	$l_0 > 0$	$k_0 + e_2 + q_z > 0$

Tabelle 1.1: Beitragende Dreiecke für den Abzugsterm der infrarot divergenten planaren Dreipunkt-Funktion

Alternativ dazu kann man auch z.B. beide Integrationen in der oberen Halbebene schließen, was ein äquivalentes Ergebnis überlappender Rechtecke gibt, wenn man berücksichtigt, daß die Summe aller Residuen verschwindet, wenn alle Pole in der oberen Halbebene liegen [49].

- Die s- und t-Integrationen faktorisieren und sind von der Form

$$\begin{aligned}
&\int_0^\infty ds\, \frac{1}{(s + s_0 - i\eta)(s + s_0' - i\eta)} \\
&\quad = \quad \frac{1}{s_0' - s_0}\Big(\ln|s_0'| - \ln|s_0| + i\pi\big(\Theta(-s_0) - \Theta(-s_0')\big)\Big) \quad \text{und} \\
&\int_0^\infty dt\, \frac{1}{(t + t_0 - i\eta)(t + t_0' - i\eta)} \\
&\quad = \quad \frac{1}{t_0' - t_0}\Big(\ln|t_0'| - \ln|t_0| + i\pi\big(\Theta(-t_0) - \Theta(-t_0')\big)\Big)\,. \qquad (1.52)
\end{aligned}$$

1.3.1.3 Berechnung des divergenten Anteils

Der divergente Anteil $V_{IR,\mathrm{div}}$ ist einfach das Produkt zweier Einloop-Dreipunkt-Funktionen. Da der Faktor mit der k-Integration infrarot divergent ist und seine Entwicklung in ε mit $1/\varepsilon$ beginnt, muß der Faktor mit der l-Integration bis zur Ordnung $\mathcal{O}(\varepsilon)$ entwickelt werden, um das Produkt bis zu $\mathcal{O}(\varepsilon^0)$ zu bekommen.

Das Abzugsverfahren für infrarot divergente Graphen ist auch für numerisch problematische Fälle der planare Dreipunkt-Funktion geeignet, die nahe an der Infrarot-Singularität sind, z.B. $q_1^2 = m_4^2$, $q_2^2 = m_5^2$, m_6 klein, aber endlich. Der Abzugsterm liefert dann die führenden Logarithmen, während die numerische Stabilität von $V_{\mathrm{IR,endl}}$ verbessert wird.

1.3.2 IR-Divergenz in beiden Loopimpulsen

Setzt man im Graphen aus Abb. 1.8 auch $m_1^2 = q_1^2$, $m_2^2 = q_2^2$ und $m_3 = 0$, so gibt es zusätzlich zur Infrarot-Divergenz bei $k \to 0$ noch eine Über-Alles-Divergenz, wenn $l \to 0 \wedge k \to 0$ gehen. Für diesen individuellen Graphen läßt sich kein geeigneter Abzugsterm finden, aber in der Summe mit dem zugehörigen gekreuzten Graphen, der ebenfalls eine Über-Alles-Divergenz hat, faktorisieren diese bei geeigneter Wahl des Impulsflusses und einer

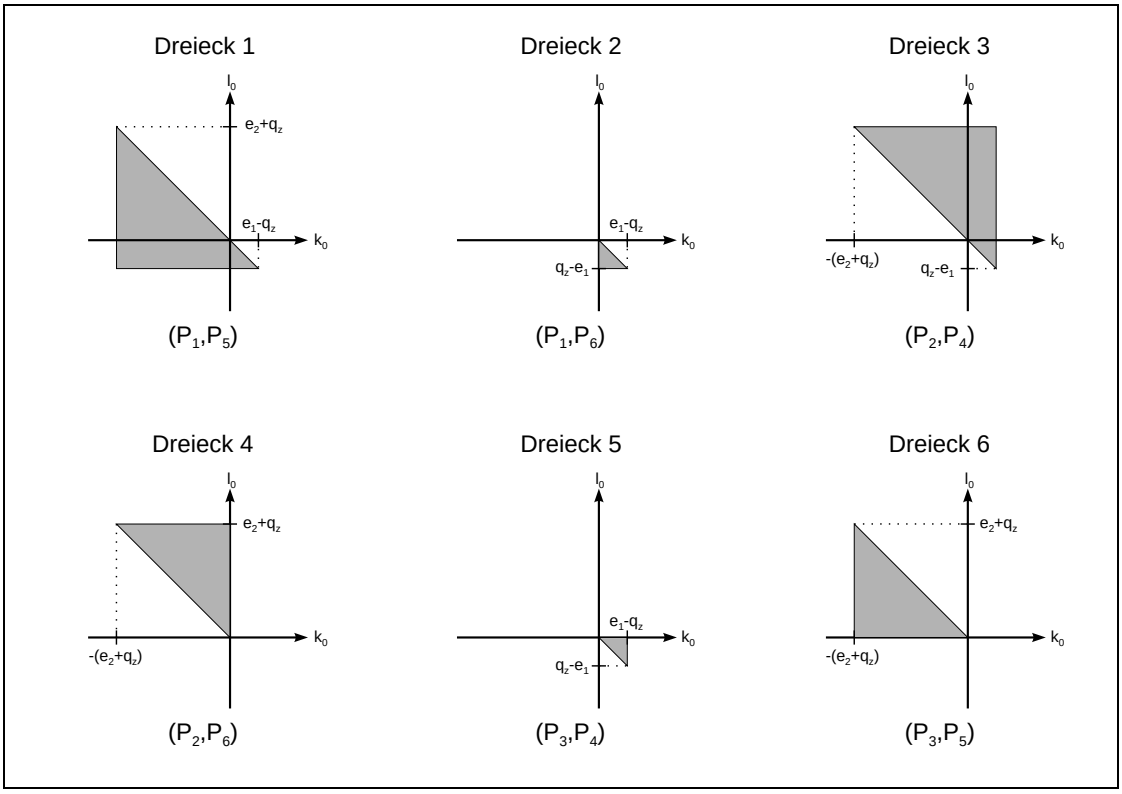


Abbildung 1.9: Beitragende Dreiecke für den Abzugsterm der infrarot divergenten planaren Dreipunkt-Funktion

Symmetrisierung in einen infrarot endlichen und einen infrarot divergenten Anteil, der das Produkt zweier infrarot divergenter Einloop-Graphen (Abb. 1.10) ist. Derartige Vereinfachungen in der Summe infrarot divergenter Beiträge erwartet man aus dem Kinoshita-Lee-Nauenberg-Theorem [46, 60].

Konkret wählt man auch für die planare Dreipunkt-Funktion einen Impulsfluß analog zur gekreuzten Dreipunkt-Funktion, bei dem zwei Propagatoren, nämlich P_1 und P_2, die Summe $l + k$ enthalten. Mit dieser Wahl tragen die beiden masselosen Propagatoren jeweils die Impulse l und k. Betrachtet man die Integranden $\mathcal{I}_P$ der planaren und $\mathcal{I}_C$ der gekreuzten Dreipunkt-Funktion (ohne das implizit vorhandene $i\eta$ in den Propagatoren)

$$\begin{aligned}
\mathcal{I}_P &= \frac{1}{((l+k)^2 - 2(l+k)\cdot q_1)((l+k)^2 + 2(l+k)\cdot q_2)} \\
&\quad \times \frac{1}{(k^2 - 2k\cdot q_1)(k^2 + 2k\cdot q_2) l^2 k^2} \\
\mathcal{I}_C &= \frac{1}{((l+k)^2 - 2(l+k)\cdot q_1)((l+k)^2 + 2(l+k)\cdot q_2)} \\
&\quad \times \frac{1}{(l^2 - 2l\cdot q_1)(k^2 + 2k\cdot q_2) l^2 k^2}\,,
\end{aligned} \tag{1.53}$$

so gilt für die in l und k symmetrisierte Summe der beiden, die man betrachten darf, da

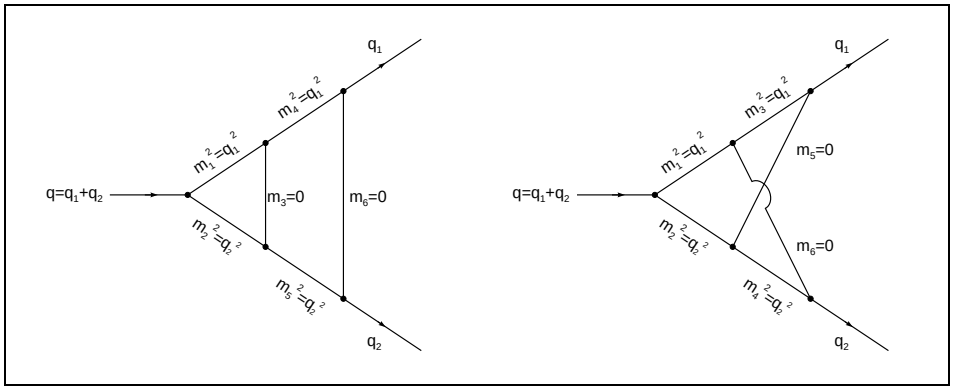


Abbildung 1.10: Die planare und die gekreuzte Dreipunkt-Funktion im Falle einer Über-Alles-Infrarot-Divergenz

über l und k integriert wird,

$$\begin{aligned} 2\mathcal{I}_{\mathcal{P}+\mathcal{C}} &= \mathcal{I}_P + \mathcal{I}_P\big|_{l\leftrightarrow k} + \mathcal{I}_C + \mathcal{I}_C\big|_{l\leftrightarrow k} \\ &= \frac{1}{(l^2-2l\cdot q_1)(l^2+2l\cdot q_2)l^2} \times \frac{1}{(k^2-2k\cdot q_1)(k^2+2k\cdot q_2)k^2} \\ &\quad \times \left(1-\frac{2l\cdot k}{(l+k)^2-2(l+k)\cdot q_1}\right) \times \left(1-\frac{2l\cdot k}{(l+k)^2+2(l+k)\cdot q_2}\right) . \end{aligned} \tag{1.54}$$

Die letzten beiden Faktoren gehen offensichtlich gegen 1, wenn l und/oder k gegen 0 gehen. Die Infrarot-Divergenz steckt ausschließlich in den ersten beiden Faktoren, die jeweils wie $1/l^4$ bzw. $1/k^4$ für l bzw. $k \to 0$ gegen Unendlich gehen.

Damit ist mit

$$\begin{aligned} V_P + V_C &= \int d^D l \int d^D k\, \mathcal{I}_P + \int d^D l \int d^D k\, \mathcal{I}_C \\ &= \int d^4 l \int d^4 k\, \frac{1}{2}\left(\mathcal{I}_P + \mathcal{I}_P\big|_{l\leftrightarrow k} + \mathcal{I}_C + \mathcal{I}_C\big|_{l\leftrightarrow k} - \mathcal{I}_{1,l}\mathcal{I}_{1,k}\right) \\ &\quad + \frac{1}{2}\int d^D l\, \mathcal{I}_{1,l} \int d^D k\, \mathcal{I}_{1,k} + \mathcal{O}(\varepsilon) \end{aligned} \tag{1.55}$$

ein Ausdruck gefunden, dessen schwierigster Teil in vier Dimensionen mit der Parallel-/Orthogonalraum-Methode berechnet werden kann (Abb. 1.11). Der infrarot divergente Anteil ist das Produkt zweier infrarot divergenter Einloop-Graphen

$$\begin{aligned} \mathcal{I}_{1,l} &= \frac{1}{(l^2-2l\cdot q_1)(l^2+2l\cdot q_2)l^2} \\ \mathcal{I}_{1,k} &= \mathcal{I}_{1,l}\big|_{l\to k}\,, \end{aligned} \tag{1.56}$$

die analytisch bekannt sind [42].

Für die Berechnung des endlichen Anteils muß man berücksichtigen, daß wegen der Symmetrisierung in l und k beide Impulse in allen Integrationsschritten aus Abschnitt 1.1.1 gleich behandelt werden müssen. Dies ist bei der gekreuzten Dreipunkt-Funktion in [30] nicht gewährleistet, da dort zusätzlich zu der Linearisierungs-Transformation Gl. (1.7) noch $l_1 \to l_1 + q_z$ verschoben wurde, um die Propagatoren einfacher zu machen, während

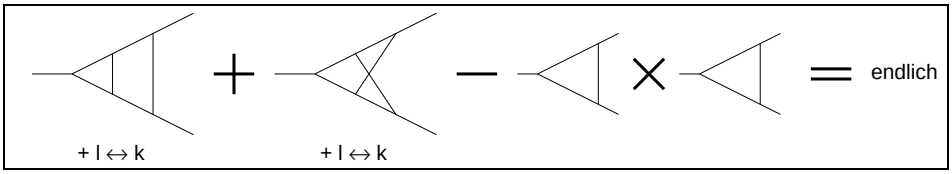


Abbildung 1.11: Zieht man von der symmetrisierten Summe aus planarer und gekreuzter Dreipunkt-Funktion ein Produkt aus zwei Einloop-Graphen ab, so ist das Ergebnis infrarot endlich.

k_1 unverändert blieb. Dies hat einen Einfluß auf die Koeffizienten der Vierfach-Integraldarstellung (analog zu Gl. (1.12)) und die beitragenden Gebiete, die entlang der l_0-Achse um q_z nach unten verschoben werden müssen (Abb. 1.12). Die Gebiete des symmetrisierten Terms $l \leftrightarrow k$ erhält man durch Spiegelung an der Winkelhalbierenden der 1. und 3. Quadranten der (l_0, k_0)-Ebene.

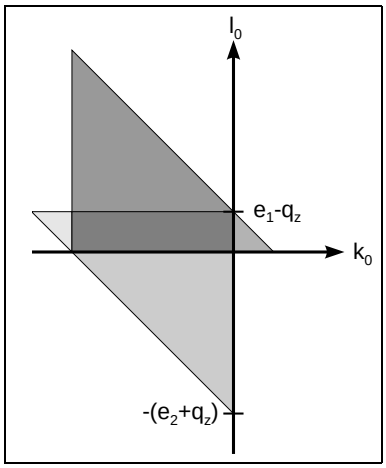


Abbildung 1.12: Vereinigungsmenge der Integrationsgebiete für die gekreuzte Dreipunkt-Funktion

Für die planare Dreipunkt-Funktion darf man nicht die Ergebnisse aus Abschnitt 1.1 übernehmen, da hier der Impulsfluß so gewählt wurde, daß zwei Propagatoren $l + k$ enthalten (vgl. auch die Abzugsterme in Abschnitt 1.5.7). Vor der z-Integration (Abschnitt 1.1.1.4) muß daher eine Partialbruchzerlegung in diesen beiden Propagatoren gemacht werden, wodurch man Pole auf der reellen l_1- bzw. k_1-Achse erhält. Die Rechenschritte sind ähnlich zur gekreuzten Dreipunkt-Funktion, mit Dreiecken und zunächst einseitig unbeschränkten Gebieten (Abb. 1.13), die sich außerhalb der in Abb. 1.14 dargestellten zu Null addieren.

Für den Abzugsterm, das Produkt zweier Einloop-Graphen, lassen sich die Ergebnisse aus Abschnitt 1.3.1 benutzen, wenn man berücksichtigt, daß die Ersetzung $l \to -l$ vorgenommen werden muß.

In der Summe aller Terme muß man über das Gebiet aus Abb. 1.15 in der (l_0, k_0)-Ebene integrieren. Man sieht, daß auf der beitragenden Strecke der beiden kritischen Geraden $l_0 = 0$ und $k_0 = 0$ alle Terme $\mathcal{I}_P$, $\mathcal{I}_P\big|_{l\leftrightarrow k}$, $\mathcal{I}_C$, $\mathcal{I}_C\big|_{l\leftrightarrow k}$ und $\mathcal{I}_{1,l}\mathcal{I}_{1,k}$ vertreten sind.

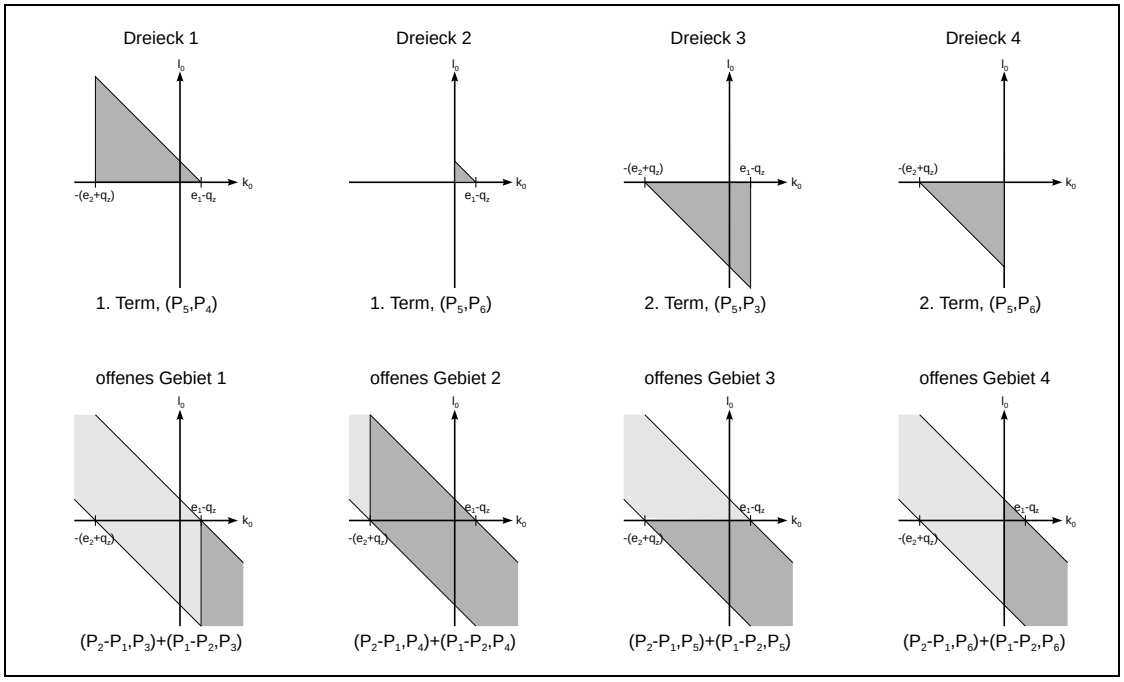


Abbildung 1.13: Einzelne Integrationsgebiete für die planare Dreipunkt-Funktion

Aus bisher noch nicht geklärten Gründen ist die gefundene Zweifach-Integraldarstellung aber nicht numerisch stabil. In der Umgebung von $l_0 = 0 \wedge k_0 = 0$ divergiert der Integrand in Polarkoordinaten $k_0 = r\cos\varphi$ und $l_0 = r\sin\varphi$ stark oszillierend in φ wie $1/r^2$. Dabei muß noch näher untersucht werden, ob es sich um ein prinzipielles Problem in der Bestimmung des Abzugsterms handelt (der aber wegen des *Powercounting*-Arguments richtig erscheint) oder ein Spezialfall in der s- oder t-Integration übersehen wurde.

1.4 Kollinear divergente Diagramme

Betrachtet man in dem infrarot divergenten Einloop-Dreipunkt-Graphen aus Abb. 1.7 den Grenzwert $m_1 \to 0$, $m_2 \to 0$, so stellt man fest, daß der Koeffizient des divergenten $1/\varepsilon$-Terms selbst divergent mit $\log(m_1^2/q^2)$ bzw. $\log(m_2^2/q^2)$ ist, d.h. nicht als Grenzwert der bereits in ε entwickelten infrarot divergenten Dreipunkt-Funktion existiert. Eine korrekte Berechnung der masselosen Dreipunkt-Funktion in dimensionaler Regularisierung hingegen liefert als Ergebnis einen Term proportional zu

$$\frac{\Gamma^2(\frac{D-4}{2})\Gamma(\frac{6-D}{2})}{\Gamma(D-3)}(-q^2 - i\eta)^{\frac{D-6}{2}}\,, \tag{1.57}$$

dessen Entwicklung in ε Pole in $1/\varepsilon$, wie andere UV oder IR divergente Einloop-Graphen auch, aber zusätzlich noch Pole in $1/\varepsilon^2$ hat. Man nennt diesen Graphen kollinear divergent, da der Integrand — im Gegensatz zu einem infrarot divergenten Graphen, bei dem der Integrand divergiert, wenn der Loopimpuls l gegen Null geht — schon divergiert, wenn l parallel zu den äußeren lichtartigen Impulsen q_1 und q_2 ist (dies wird in Gl. (1.62) deutlich).

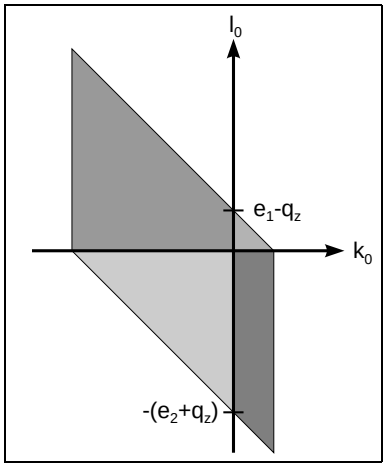


Abbildung 1.14: Vereinigungsmenge der Integrationsgebiete für die planare Dreipunkt-Funktion

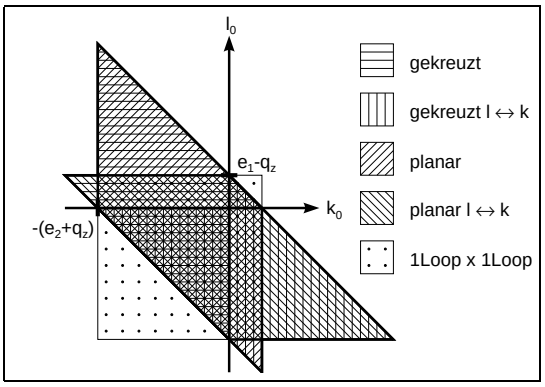


Abbildung 1.15: Integrationsgebiete für die Summe aus gekreuzter und planarer Dreipunkt-Funktion, jeweils symmetrisiert, und den Abzugsterm

Auch auf Zweiloop-Niveau, wo ultraviolett oder infrarot divergente Graphen normalerweise Pole in $1/\varepsilon^2$ haben (bei Divergenzen in l und k, sonst nur $1/\varepsilon$), muß der kollinear divergente Fall gesondert behandelt werden, da hier Divergenzen bis zu $\mathcal{O}(1/\varepsilon^4)$ auftreten können [53]. Physikalisch relevant ist dies, wenn man die leichten Quarks als masselos annimmt, z.B. im Graphen in Abb. 1.16, der im Prozeß $Z \to d\,\bar{d}$ beiträgt. In jedem Fall ist die Masse des d-Quarks klein gegenüber den anderen Massenskalen m_t, m_Z und m_W.

Im folgenden soll die planare Dreipunkt-Funktion für den kollinear divergenten Fall Abb. 1.17 ($q_1^2 = m_4^2 = 0$, $q_2^2 = m_5^2 = 0$ und $m_6 = 0$) mit der Parallel-/Orthogonalraum-Technik für den Fall allgemeiner Massen m_1, m_2 und m_3 sowie Impuls q^2 berechnet werden. Wie schon im Falle der infrarot divergenten Graphen (Abschnitt 1.3) ist das größte Problem zunächst die Bestimmung der Abzugsterme, die einerseits den Integranden endlich machen, andererseits aber auch selbst in dimensionaler Regularisierung berechnet werden können. Obwohl es dafür keinen allgemeinen Algorithmus gibt, kann man diese auch im kollinearen Fall finden. Dazu muß das Verfahren aber leicht modifiziert werden, indem eine andere Transformation zur Linearisierung der Propagatoren eingesetzt wird. Weiter-

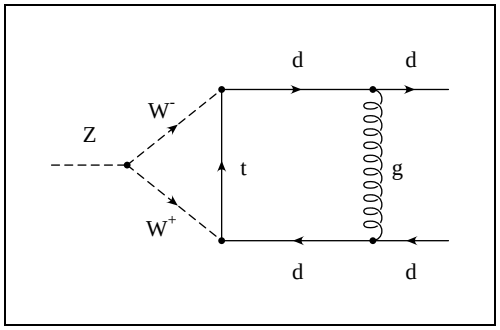


Abbildung 1.16: Ein Graph aus dem Prozeß $Z \to d\bar{d}$, der bei $m_d = 0$ kollinear divergent ist

hin findet man eine Abweichung von der Regel, daß die letztlich numerisch auszuführende Zweifach-Integration der Integraldarstellung über endliche Gebiete in der (l_0, k_0)-Ebene geht [49].

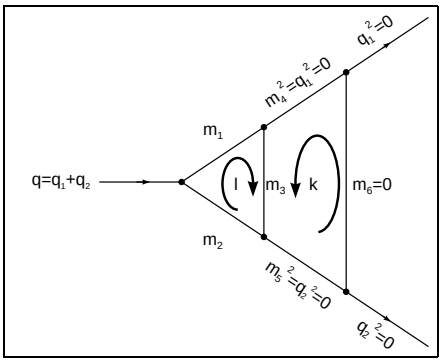


Abbildung 1.17: Kollinear divergente planare Dreipunkt-Funktion

1.4.1 Bestimmung der Abzugsterme

Das zu berechnende Integral aus Abb. 1.17 lautet

$$\begin{aligned} V_{\text{col}} &= \int d^D k \int d^D k \, \frac{1}{P_1 P_2 P_3 P_4 P_5 P_6} \\ &= \int d^D k \, \frac{1}{P_4 P_5 P_6} C(k) \,, \end{aligned} \tag{1.58}$$

wobei

$$C(k) = C(k; q_1, q_2; m_1, m_2, m_3) = \int d^D l \, \frac{1}{P_1 P_2 P_3} \tag{1.59}$$

den inneren Einloop-Dreipunkt-Untergraphen bezeichnet, der die Integration über den Impuls l zusammenfaßt und dessen „äußere“ Impulse $q_1 - k$ und $q_2 + k$ gerade um k gegenüber denen des ganzen Graphen verschoben sind (Abb. 1.18).

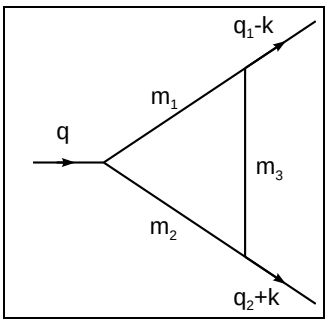


Abbildung 1.18: Einloop-Dreipunkt-Untergraph $C(k)$

Für die weitere Berechnung wird wieder das Bezugssystem aus Abschnitt 1.1.1.1 gewählt. Zusammen mit der Bedingung, daß sich die auslaufenden Teilchen auf dem Lichtkegel befinden, also $q_1^2 = q_2^2 = 0$, können die äußeren Impulse durch einen einzigen Parameter e ausgedrückt werden:

$$\begin{aligned} q = q_1 + q_2 &= (2e, 0, \vec{0}) \\ q_1 &= (e, e, \vec{0}) \\ q_2 &= (e, -e, \vec{0})\,. \end{aligned} \tag{1.60}$$

Abweichend von der Berechnung der planaren Dreipunkt-Funktion in Abschnitt 1.1.1 oder der gekreuzten Dreipunkt-Funktion [30] wird hier eine symmetrischere Transformation der Parallelraum-Komponenten der Loopimpulse $l^\mu = (l_0, l_1, \vec{l}_\perp)$ und $k^\mu = (k_0, k_1, \vec{k}_\perp)$ benutzt, die die Propagatoren linear in allen Parallelraum-Komponenten macht (normalerweise bringt dies keinen Vorteil, da nach der Residuenintegration über l_1 und k_1 die Linearität in l_0 und k_0 wieder verloren geht):

$$\begin{aligned} l_0 &\to l_0 + l_1 \\ l_1 &\to l_1 - l_0 \\ k_0 &\to k_0 + k_1 \\ k_1 &\to k_1 - k_0\,. \end{aligned} \tag{1.61}$$

Diese Transformation, deren Jacobi-Determinante jeweils einen Faktor 2 für l und k liefert, ist äquivalent dazu, das Integral in Lichtkegel-Variablen $l \cdot q_{1,2}$ und $k \cdot q_{1,2}$ auszudrücken. Damit lauten die linearisierten Propagatoren

$$\begin{aligned} P_1 &= 4l_0(l_1 + e) - s - m_1^2 + i\eta \\ P_2 &= 4l_1(l_0 - e) - s - m_2^2 + i\eta \\ P_3 &= 4(l_0 + k_0)(l_1 + k_1) - s - t - \sqrt{s}\,\sqrt{t}\,z - m_3^2 + i\eta \\ P_4 &= 4k_0(k_1 - e) - t + i\eta \\ P_5 &= 4k_1(k_0 + e) - t + i\eta \\ P_6 &= 4k_0k_1 - t - m_6^2 + i\eta\,. \end{aligned} \tag{1.62}$$

m_6 wird zunächst für die Berechnung des endlichen Anteils als Regulator von Null verschieden belassen. Der Grenzwert $m_6 \to 0$ wird später in Abschnitt 1.4.3 analytisch ausgeführt.

Die Berechnung ist gültig für den Fall beliebiger Massen m_1, m_2 und m_3, außer für den trivialen, doppelt kollinear divergenten Fall $m_1 = m_2 = m_3 = 0$.

In der Darstellung von Gl. (1.62) sieht man gut, wie sich die kollineare Divergenz von einer infraroten unterscheidet, bei der erst die letzte Integration bei $k \to 0$ divergent wird. Hier gibt es eine Divergenz entlang der Linien $k_0 = 0$ (von P_4 und P_6) und $k_1 = 0$ (von P_5 und P_6), wenn $t = 0$ ist, also schon wenn $k^2 \to 0$ geht bzw. wenn k parallel („kollinear“) zu q_1 oder q_2 ist.

Einen Abzugsterm, der die kollineare Divergenz regularisiert, findet man durch Modifikation von $C(k) = C(k_0, k_1, t)$:

$$\begin{aligned} C(k_0,k_1,t) \;=\; & C(k_0,k_1,t) - C(k_0,0,0) - C(0,k_1,0) + C(0,0,0) && \text{(I)} \\ & + C(k_0,0,0) - C(0,0,0) && \text{(II)} \\ & + C(0,k_1,0) - C(0,0,0) && \text{(III)} \\ & + C(0,0,0) && \text{(IV).} \end{aligned} \tag{1.63}$$

Beitrag (I) ist endlich und kann in $D = 4$ Dimensionen berechnet werden, da mit $C(k_0, 0, 0)$ ein Term subtrahiert wird, der sich ähnlich wie $C(k_0, k_1, t)$ für $k_1 \to 0$ bei $t = 0$ verhält und so das *Powercounting* in k_1 um eine Potenz verbessert. Ebenso verbessert $C(0, k_1, 0)$ das *Powercounting* in k_0, während die Addition von $C(0, 0, 0)$ dafür sorgt, daß die zusätzlich vorhandene normale Infrarot-Divergenz bei $k = 0$ (vgl. Abschnitt 1.3) nur einmal subtrahiert wird. Die Beiträge (II) und (III) regularisieren die kollineare Divergenz. Sie sind trotz des Abzugsterms $C(0, 0, 0)$ noch divergent und müssen daher in D Dimensionen berechnet werden. Ihre Entwicklung in ε beginnt mit einem Pol $1/\varepsilon$. Beitrag (IV) enthält sowohl die infraroten als auch die kollinearen Divergenzen des zugehörigen Einloop-Graphen in k. Seine Divergenz beginnt entsprechend mit $1/\varepsilon^2$.

1.4.2 Berechnung der divergenten Anteile

1.4.2.1 Beitrag (IV)

Beitrag (IV) läßt sich am einfachsten berechnen, da er in ein Produkt zweier Einloop-Dreipunkt-Graphen faktorisiert, wobei der k-Graph vollständig masselos ist:

$$\begin{aligned} I_{\mathrm{IV}} &= \int d^D k \, \frac{1}{P_4 P_5 P_6} \times \int d^D l \, \frac{1}{P_1 P_2 \tilde{P}_3} \\ &= -i\pi^{D/2} \frac{\Gamma^2\left(\frac{D-4}{2}\right)\Gamma\left(\frac{6-D}{2}\right)}{\Gamma(D-3)} (-q^2 - i\eta)^{\frac{D-6}{2}} C(0,0,0) \end{aligned} \tag{1.64}$$

mit

$$\tilde{P}_3 = P_3\big|_{k=0} \,. \tag{1.65}$$

Die Entwicklung des Produkts der Γ-Funktionen beginnt mit $1/\varepsilon^2$. Daher muß $C(0, 0, 0)$ bis $\mathcal{O}(\varepsilon^2)$ berechnet werden. Da eine analytische Entwicklung der Einloop-Dreipunkt-Funktion für beliebige Massen aber nur bis $\mathcal{O}(\varepsilon)$ bekannt ist [67], muß die letzte Ordnung numerisch berechnet werden. Diese liefert aber nach Multiplikation mit dem $1/\varepsilon^2$-Pol nur einen Beitrag zu $\mathcal{O}(\varepsilon^0)$, die ohnehin in Beitrag (I) numerisch berechnet wird. Für die Entwicklung bis $\mathcal{O}(\varepsilon^2)$ kann man ausnutzen, daß $C(0, 0, 0)$ endlich ist und daher bereits

vor der Integration entwickelt werden kann:

$$\begin{aligned} C(0,0,0) &= \int d^D l \, \frac{1}{P_1 P_2 \tilde{P}_3} \\ &= K_{D-2} \int\limits_{-\infty}^{+\infty} dl_0 \int\limits_{-\infty}^{+\infty} dl_1 \int\limits_{0}^{+\infty} dl_\perp \, l_\perp^{D-3} \frac{1}{P_1 P_2 \tilde{P}_3} \\ &= \frac{K_{D-2}}{2} \int\limits_{-\infty}^{+\infty} dl_0 \int\limits_{-\infty}^{+\infty} dl_1 \int\limits_{0}^{+\infty} ds \, s^{-\varepsilon} \frac{1}{P_1 P_2 \tilde{P}_3} \, . \\ &= \frac{K_{D-2}}{2} \int\limits_{-\infty}^{+\infty} dl_0 \int\limits_{-\infty}^{+\infty} dl_1 \int\limits_{0}^{+\infty} ds \, (1 - \varepsilon \ln s + \frac{\varepsilon^2}{2} \ln^2 s + \mathcal{O}(\varepsilon^3)) \frac{1}{P_1 P_2 \tilde{P}_3} \end{aligned} \tag{1.66}$$

mit K_{D-2} aus Gl. (1.23), das auch noch in ε entwickelt werden muß. Nach Linearisierung der Propagatoren in l_1 durch $l_0 \to l_0 + l_1$ und Ausführen der l_1-Integration mit dem Residuensatz ist die s-Integration für den $\mathcal{O}(\varepsilon^2)$-Anteil von der Form Gl. (1.45) und

$$\int ds \frac{\ln^2 s}{s + s_0} = \ln^2 s \ln \frac{s + s_0}{s_0} + 2 \ln s \, \mathrm{Li}_2(-\frac{s}{s_0}) - 2 \, \mathrm{Li}_3(-\frac{s}{s_0}) \, , \tag{1.67}$$

wobei die Trilogarithmen im Endergebnis herausfallen. Die verbleibende Integration über k_0 (über ein endliches Intervall) wird numerisch ausgeführt.

1.4.2.2 Beitrag (II)

Beitrag (II) ist schwieriger zu berechnen, da die Integrationen in l und k nicht faktorisieren. Da in beiden Termen aber $t = 0$ gesetzt ist, hängt der Integrand nicht von z, dem Kosinus des Winkels zwischen $\vec{l}_\perp$ und $\vec{k}_\perp$, ab. Daher hat

$$I_{\mathrm{II}} = \frac{\pi^{\frac{D-2}{2}}}{2\Gamma\left(\frac{D-2}{2}\right)} \int\limits_{-\infty}^{+\infty} dk_0 \big(C(k_0,0,0) - C(0,0,0)\big) \int\limits_{0}^{\infty} t^{\frac{D-4}{2}} dt \int\limits_{-\infty}^{+\infty} dk_1 \, \frac{1}{P_4 P_5 P_6} \tag{1.68}$$

keinen Schnitt in der komplexen k_1-Ebene, und man kann die k_1-Integration ohne Probleme mit dem Residuensatz ausführen. Schließt man die Kontur in der unteren Halbebene, so gibt es nur einen Beitrag vom Pol von P_5 im Intervall $-e < k_0 < 0$. Außerhalb dieses Intervalls liegen alle Pole auf derselben Seite der reellen Achse, und es gibt keinen Beitrag. Damit ist

$$\int\limits_{-\infty}^{+\infty} dk_1 \, \frac{1}{P_4 P_5 P_6} = \frac{-2\pi i}{4e^2} \frac{k_0 + e}{t(t + 4k_0(k_0 + e) - i\eta)} \Theta(-e < k_0 < 0) \, . \tag{1.69}$$

Die Integration über t gibt dann

$$\int\limits_{0}^{\infty} dt \, \frac{t^{\frac{D-6}{2}}}{t + 4k_0(k_0 + e) - i\eta} = (4k_0(k_0 + e) - i\eta)^{\frac{D-6}{2}} \Gamma\left(\frac{D-4}{2}\right) \Gamma\left(\frac{6-D}{2}\right) \, . \tag{1.70}$$

$k_0(k_0+e)$ ist dabei immer negativ. Mit einer Substitution $k_0 = -xe$ erhält man

$$\begin{aligned} I_{\mathrm{II}} &= -i\pi^{D/2}\frac{\Gamma\left(\frac{D-4}{2}\right)\Gamma\left(\frac{6-D}{2}\right)}{\Gamma\left(\frac{D-2}{2}\right)}(-q^2-i\eta)^{\frac{D-6}{2}} \\ &\quad\times\int_0^1 dx\frac{1}{x}\left[x(1-x)\right]^{\frac{D-4}{2}}\left(C(-xe,0,0)-C(0,0,0)\right)\,, \end{aligned} \tag{1.71}$$

mit $q^2 = 4e^2$.

Die Entwicklung der Γ-Funktionen beginnt mit einem Pol in $1/\varepsilon$, daher muß das Integral über x bis zu $\mathcal{O}(\varepsilon)$ ausgewertet werden. Aber schon seine Ordnung $\mathcal{O}(\varepsilon^0)$, die nach Multiplikation mit dem $1/\varepsilon$-Pol aus den Γ-Funktionen zum divergenten Anteil von V_{col} beiträgt, muß numerisch berechnet werden. Daher können im kollinearen Fall nicht alle divergenten Anteile analytisch angegeben werden. Da es sich aber nur um eine eindimensionale Integration handelt, kann sie zu relativ hoher Genauigkeit ausgeführt werden.

1.4.2.3 Beitrag (III)

Beitrag (III) kann analog zu Beitrag (II) berechnet werden, indem man die Symmetrie des Integranden in k_0 und k_1 ausnutzt und zuerst die k_0-Integration mit dem Residuensatz ausführt. Es gibt dann nur einen Beitrag von P_4 im Intervall $0 < k_1 < e$, wenn die Kontur in der oberen Halbebene geschlossen wird. Man erhält

$$\begin{aligned} I_{\mathrm{III}} &= -i\pi^{D/2}\frac{\Gamma\left(\frac{D-4}{2}\right)\Gamma\left(\frac{6-D}{2}\right)}{\Gamma\left(\frac{D-2}{2}\right)}(-q^2-i\eta)^{\frac{D-6}{2}} \\ &\quad\int_0^1 dx\frac{1}{x}\left[x(1-x)\right]^{\frac{D-4}{2}}\left(C(0,xe,0)-C(0,0,0)\right)\,. \end{aligned} \tag{1.72}$$

Für den symmetrischen Fall $m_1 = m_2$ ist $I_{\mathrm{III}} = I_{\mathrm{II}}$.

1.4.3 Berechnung des endlichen Anteils

Der schwierigste Teil ist die Berechnung von Beitrag (I), des endlichen Anteils. Im Prinzip kann man analog zu Abschnitt 1.1.1 vorgehen, wenn man berücksichtigt, daß die einzelnen Terme divergent sind. Bei der Integration über z stellt man fest, daß nur der erste Term mit $C(k_0,k_1,t)$ einen nicht-trivialen Beitrag gibt. Die anderen Terme hängen, abgesehen vom Integrationsmaß $1/\sqrt{1-z^2}$, nicht von z ab und haben daher keinen Schnitt in den komplexen l_1- und k_1-Ebenen.

Die l_1- und k_1-Integrationen werden mit Hilfe des Residuensatzes ausgeführt. Jeder einzelne Term ist für sich konvergent, wenn l_1 bzw. k_1 nach Unendlich gehen (die kollineare Divergenz befindet sich ja bei $k_1 = 0$ bzw. $k_0 = 0$). Daher können die Konturen der Terme unabhängig voneinander geschlossen werden. Für den $C(k_0,k_1,t)$-Term muß der Schnitt der Wurzel berücksichtigt werden, der in der oberen Halbebene liegt, wenn $l_0 + k_0 < 0$ ist, und in der unteren Halbebene, wenn $l_0 + k_0 > 0$ ist. Überprüft man die Halbebenen, in denen die Propagatoren Pole haben, so findet man drei beitragende Dreiecke in der (l_0,k_0)-Ebene von den Paaren (P_1,P_5), (P_2,P_4) und (P_2,P_6), die in Abb. 1.19 abgebildet sind.

Für den zweiten Term mit $C(k_0, 0, 0)$ kann man die Kontur beliebig schließen, da er keinen Schnitt hat. Schließt man sie wie bei $C(k_0, k_1, t)$, also abhängig vom Vorzeichen von $l_0 + k_0$, was äquivalent dazu ist, Pole vom gemischten Propagator P_3 zu vermeiden, so findet man dieselben Dreiecke wie oben.

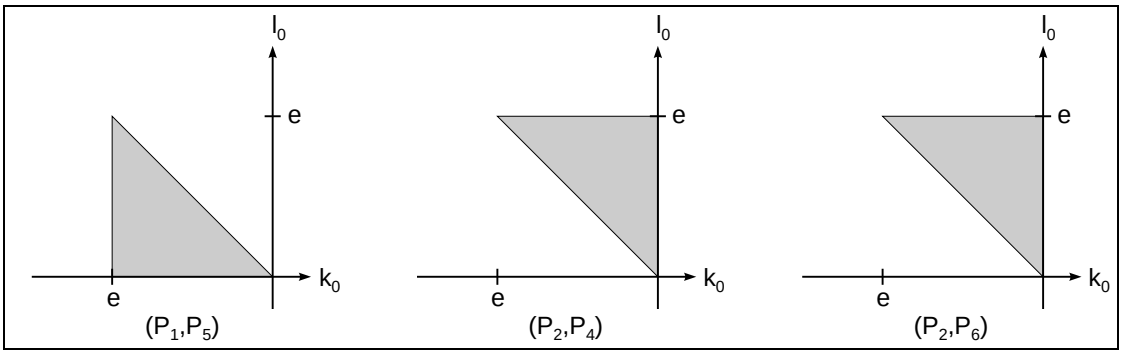


Abbildung 1.19: Beitragende Gebiete von $C(k_0, k_1, t)$ und $C(k_0, 0, 0)$

Mit dem $C(0, k_1, 0)$-Term tritt etwas Neues bei der Parallel-/Orthogonalraum-Methode auf: Unabhängig davon, wie die Kontur geschlossen wird, bleibt am Ende eine Integration über ein unbegrenztes Gebiet in der (l_0, k_0)-Ebene. Fährt man mit der Strategie fort, Pole von P_3 zu vermeiden, so muß die Kontur für die l_1-Integration in der oberen Halbebene geschlossen werden, wenn $l_0 > 0$ ist, und in der unteren Halbebene, wenn $l_0 < 0$ ist. Weiterhin wird die Kontur der k_1-Integration immer in der oberen Halbebene geschlossen. Damit erhält man die drei Gebiete in Abb. 1.20 von (P_2, P_4), (P_2, P_5) und (P_2, P_6), wobei sich die drei Beiträge im Bereich $k_0 < -e$ *nicht* zu Null addieren.

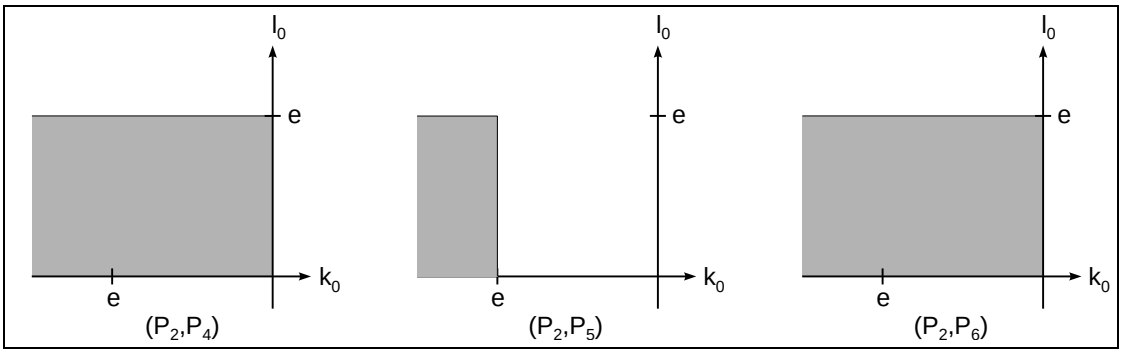


Abbildung 1.20: Beitragende Gebiete von $C(0, k_1, 0)$

Der letzte Term mit $C(0, 0, 0)$ faktorisiert in l und k. Man kann die Kontur so schließen, daß in der l_1-Integration nur P_2 und in der k_1-Integration nur P_5 beiträgt und erhält als Integrationsgebiet ein Quadrat (Abb. 1.21).

Nun können die Integrationen über t und danach s analytisch ausgeführt werden. Da m_6 zunächst endlich gelassen wurde, sind nach einer Partialbruchzerlegung die Integrale

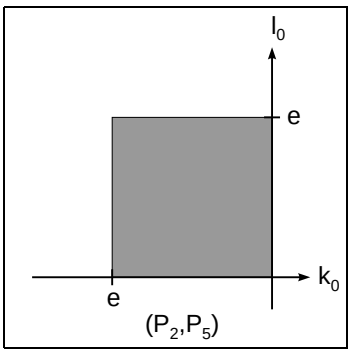


Abbildung 1.21: Beitragende Gebiete von $C(0,0,0)$

von der Form

$$\int_0^\infty ds \int_0^\infty dt \frac{1}{s+s_0-i\eta}\frac{1}{t+t_0-i\eta}\frac{1}{\sqrt[c]{(at+b+i\eta+cs)^2-4st}} \quad \text{von } C(k_0,k_1,t)$$

$$\int_0^\infty ds \int_0^\infty dt \frac{1}{s+s_0-i\eta}\frac{1}{t+t_0-i\eta}\frac{1}{at+b+i\eta+cs} \quad \text{von } C(k_0,0,0) \text{ und } C(0,k_1,0)$$

$$\int_0^\infty ds \int_0^\infty dt \frac{1}{s+s_0-i\eta}\frac{1}{s+s_0'-i\eta}\frac{1}{t+t_0-i\eta}\frac{1}{t+t_0'-i\eta} \quad \text{von } C(0,0,0) \quad (1.73)$$

mit nicht-verschwindenden t_0. Da einige dieser t_0 aber proportional zu m_6^2 sind, divergiert das zugehörige Integral entsprechend im Grenzwert $m_6 \to 0$ proportional zu $\log m_6^2$. Man kann aber zeigen, daß jeweils die Summe von Paaren dieser Integrale endlich ist, wenn $m_6 \to 0$ geht und daher die Abzugsterme richtig gewählt wurden. Die Paare selbst hängen von der Position in der (l_0, k_0)-Ebene ab, die in die beitragenden Gebiete A, B und C aus Abb. 1.22 zerlegt werden kann.

In der Praxis vertauscht man gegenüber Abschnitt 1.1.1.7 und 1.1.1.8 die Reihenfolge der s- und t- Integration und entwickelt die Integrale aus Gl. (1.73) für $t_0 \to 0$. Zuletzt sammelt man alle Terme proportional zu $\log t_0$, wobei alle verschwindenden t_0 dieselbe Proportionalitätskonstante zu m_6^2 haben. Diese Terme heben sich gegenseitig auf. Die übrigen Terme bereiten keine Probleme in der anschließenden analytischen s-Integration und der numerischen Integration über l_0 und k_0.

1.4.4 Ergebnisse und Vergleich

Die Tabellen 1.2-1.4 zeigen numerische Ergebnisse für drei verschiedene Massenkombinationen in Abhängigkeit von $q^2 = (q_1+q_2)^2$, die mit obiger Methode berechnet wurden, sowie zum Vergleich dieselben Werte mit Hilfe von asymptotischer Entwicklung in q^2 und anschließender Padé-Approximation [21, 24, 25]. Die Tabellen enthalten die Summe aller endlichen Anteile (von Beitrag (I)-(IV)). Der Fehler ist ± 1 in der letzten angegebenen Stelle.

- Tabelle 1.2: Drei gleiche Massen, $m_1 = m_2 = m_3 = m$. Dieser Fall wird z.B. für $H \to gg$ über einen Top-Loop (Abb. 1.23) benötigt. Zahlen für diesen Fall wurden

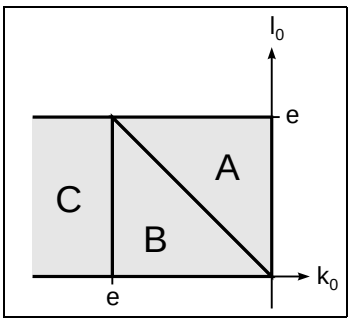


Abbildung 1.22: Integrationsgebiete in der (l_0, k_0)-Ebene für die kollinear divergente planare Dreipunkt-Funktion

unabhängig von M. Spira bereitgestellt [84], die mit den hier gezeigten übereinstimmen.

- Tabelle 1.3: Zwei verschiedene Massen, $m_1 = m_2 = 80\,\mathrm{GeV}$ ($\sim m_\mathrm{W}$), $m_3 = 180\,\mathrm{GeV}$ ($\sim m_\mathrm{top}$). Dies entspricht dem Graphen aus Abb. 1.16.
- Tabelle 1.4: $m_1 = m_2 = 0$, m_3 endlich (z.B. Abb. 1.24).

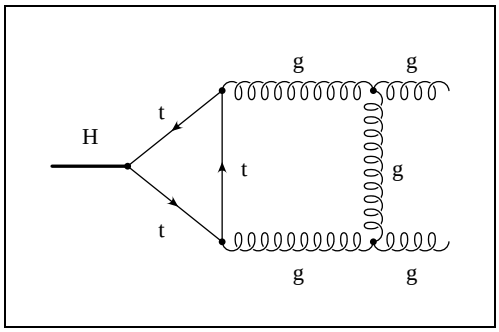


Abbildung 1.23: Ein kollinear divergenter Graph aus dem Prozeß $H \to g\,g$, Ergebnisse des endlichen Anteils des zugehörigen skalaren Graphen in Tabelle 1.2

Die numerische Stabilität der Ergebnisse ist im allgemeinen nicht so gut wie bei der konvergenten planaren Dreipunkt-Funktion, kann aber oberhalb der Schwellen mit der Genauigkeit der asymptotischen Entwicklung konkurrieren. Insbesondere der Fall $m_1 = m_2 = 0$ gibt für große q^2 nur eine Genauigkeit von rund 1%. Ein großes q^2 ist nach einer Skalentransformation aber äquivalent zu einem kleinen m_3, und dann geht der Graph in den Grenzfall $m_3 \to 0$ über, der auch in l kollinear divergent ist und daher wie $\log^2 \frac{q^2}{m_3^2}$ divergiert. Dieser Fall müßte gesondert regularisiert werden.

Ein Vergleich des CPU-Zeitbedarfs beider Methoden ist schwierig. Die Entwicklung in q^2 benötigt einmalig viel Zeit, um die Taylor-Koeffizienten und die Padé-Approximation für bestimmte Massen m_1, m_2 und m_3 zu berechnen. Das Berechnen des Integrals für einen gewünschten Wert von q^2 ist danach nur eine Sache von Sekunden. Eine Erhöhung der Genauigkeit zieht eine Neuberechnung aller Koeffizienten nach sich. Die Komplexität

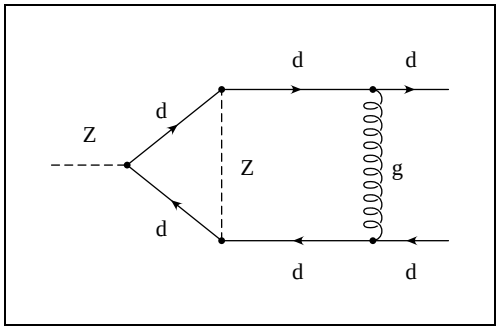


Abbildung 1.24: Ein weiterer Graph aus dem Prozeß $Z \to d\,\bar{d}$, der bei $m_d = 0$ kollinear divergent ist und zu dem die Ergebnisse des endlichen Anteils seines skalaren Graphen in Tabelle 1.4 aufgeführt sind

	asympt. Entwicklung		P/O-Raum-Integration	
q^2/m^2	Re	Im	Re	Im
1.0	2.764761286	−0.1635351018	2.76477	−0.16353
2.0	1.619771096	0.4534319337	1.61977	0.45344
3.0	1.477131552	0.5068403285	1.47713	0.50684
3.9	3.035456900	0.2892013177	3.03546	0.28920
3.99	5.17259	0.014233	5.17253	0.014236
4.01	6.187	2.749	6.18778	2.74948
4.1	3.27645	3.92263	3.27646	3.92263
5.0	−0.438728258	2.545472409	−0.438728	2.54548
6.0	−0.8927774793	1.492146568	−0.892777	1.49215
7.0	−0.9057171008	0.9272065484	−0.905717	0.927207
8.0	−0.8281584836	0.5982049870	−0.828159	0.598205
10.0	−0.6509229304	0.2610928562	−0.650923	0.261093
40.0	−0.0713343755	−0.0467805683	−0.0713345	−0.0467805
400.0	−0.00055586	−0.00274078	−0.000555895	−0.00274078

Tabelle 1.2: Endlicher Anteil der skalaren kollinear divergenten Dreipunkt-Funktion für $m_1 = m_2 = m_3 = m$

nimmt mit der Anzahl verschiedener Massen zu. Der Fall dreier verschiedener Massen wurde noch nicht berechnet.

Die Parallel-/Orthogonalraum-Methode hingegen hat in der Regel mehr Probleme, wenn Spezialfälle, wie etwa gleiche Massen, auftreten. Der Zeitbedarf ist in etwa konstant für jedes Tupel (q^2, m_1, m_2, m_3), da nichts im voraus berechnet werden kann, und wegen der Abzugsterme etwa eine Größenordnung höher als für eine vergleichbare infrarot und kollinear endliche planare Dreipunkt-Funktion. Die Genauigkeit kann für jedes Tupel (q^2, m_1, m_2, m_3) unabhängig voneinander gewählt werden.

Der Vergleich hat gezeigt, daß beide Methoden konsistente Ergebnisse liefern und damit die realistische Berechnung des Zerfalls $Z \to d\,\bar{d}$ zur Zweiloop-Ordnung möglich wird.

q^2/m_1^2	asympt. Entwicklung Re	Im	P/O-Raum-Integration Re	Im
1.0	1.255976034	0.4303855857	1.255974	0.430385
2.0	0.6127661221	0.4991658334	0.61277	0.499164
3.0	0.4346988621	0.4502907526	0.43470	0.450292
3.9	0.458394486	0.343784488	0.4584	0.34379
3.99	0.49254	0.25004761	0.4925	0.25003
4.01	0.6359	0.1902	0.63598	0.18994
4.1	0.73504	0.43260	0.73526	0.43271
5.0	0.322110	0.855622	0.32219	0.85569
6.0	0.0075200	0.8023528	0.00756	0.80237
7.0	−0.161777	0.682110	−0.16176	0.68214
8.0	−0.252547	0.564693	−0.25255	0.56469
10.0	−0.3256	0.37987	−0.32559	0.37975
40.0	−0.0695667	−0.029828	−0.06956	−0.02983
100.0	−0.013493	−0.015279	−0.01349	−0.01528
400.0	−0.00071	−0.002537	−0.00071	−0.00254

Tabelle 1.3: Endlicher Anteil der skalaren kollinear divergenten Dreipunkt-Funktion für $m_1 = m_2 = 80\,\mathrm{GeV}$, $m_3 = 180\,\mathrm{GeV}$

q^2/m_3^2	asympt. Entwicklung Re	Im	P/O-Raum-Integration Re	Im
0.5	81.17501719	12.06458720	81.1750	12.0644
1.0	17.7659	19.97799834	17.7658	19.9779
2.0	−0.6047	7.3759	−0.604	7.376
3.0	−1.7543	3.1815	−1.754	3.182
4.0	−1.595	1.603	−1.595	1.604
5.0	−1.327	0.880	−1.326	0.880
6.0	−1.096	0.503	−1.095	0.503
7.0	−0.914	0.289	−0.913	0.288
8.0	−0.773	0.159	−0.771	0.159
10.0	− 0.572	0.023	− 0.570	0.025

Tabelle 1.4: Endlicher Anteil der skalaren kollinear divergenten Dreipunkt-Funktion für $m_1 = m_2 = 0$

1.4.5 Kollineare Divergenz bei einem lichtartigen äußeren Impuls

Betrachtet man die planare Dreipunkt-Funktion für den Fall $q_1^2 = m_4^2 = 0$ und $m_6 = 0$, aber $q_2^2 = m_5^2 \neq 0$, so gibt es, zusätzlich zu der Infrarot-Singularität bei $k \to 0$, nur eine kollineare Divergenz, wenn k parallel zu q_1 ist bzw. das mit Gl. (1.61) transformierte $k_0 \to 0$ geht, während der Integrand bei $k_1 \to 0$ integrabel bleibt. Ausgehend von Gl. (1.63) muß man daher hier nur die Divergenz in k_0 abziehen:

$$\begin{aligned} C(k_0,k_1,t) \;=\; & C(k_0,k_1,t) - C(0,k_1,0) && \text{(I')} \\ & + C(0,k_1,0) - C(0,0,0) && \text{(III')} \\ & + C(0,0,0) && \text{(IV').} \end{aligned} \tag{1.74}$$

Die Berechnung von Beitrag (III') umfaßt mehr Terme aus der Residuenintegration, führt aber auch auf eine numerische Einfach-Integration.

Während bei der kollinearen Divergenz aus dem vorigen Abschnitt mit zwei lichtartigen Impulsen der Grenzfall $m_1^2 = q_1^2 = 0$, $m_2^2 = q_2^2 = 0$ und $m_3 = 0$, also einer zusätzlichen Infrarot-Divergenz bei $l \to 0$ (Abschnitt 1.3.2) in Verbindung mit einer kollinearen Divergenz für $l \parallel q_1$ und $l \parallel q_2$, analytisch bekannt ist [53], ist $m_1^2 = m_4^2 = q_1^2 = 0$, $m_2^2 = m_5^2 = q_2^2 \neq 0$ z.B. für den Myon-Zerfall von physikalischem Interesse. Bevor dies in Angriff genommen werden kann, müssen jedoch erst die in Abschnitt 1.3.2 erwähnten Probleme gelöst werden.

1.5 Tensor-Integrale

Sobald man Feynman-Graphen für eine realistische Quantenfeldtheorie — jenseits der Demonstrationsmodelle ϕ^3- oder ϕ^4-Theorie, bei denen nur skalare Diagramme

$$\int \prod_{l=1}^{L} d^D k_l \prod_{i=1}^{N} \frac{1}{p_i^2 - m_i^2} \tag{1.75}$$

mit L Loops und N Propagatoren auftreten, wobei die p_i Linearkombinationen der äußeren Impulse q_j und der Loopimpulse k_l sind — berechnen will, erhält man im Integranden eine zusätzliche Zählerstruktur aus den Feynmanregeln für Propagatoren und Vertizes in Form von Polynomen der Impulse, z.B.

$$\frac{1}{\not{p} - m} = \frac{p^\mu \gamma_\mu + m}{p^2 - m^2} \tag{1.76}$$

bei Fermion-Propagatoren oder

$$-g_s f_{abc} \left((p_1 - p_2)_\rho \, g_{\mu\nu} + (p_2 - p_3)_\mu \, g_{\nu\rho} + (p_3 - p_1)_\nu \, g_{\rho\mu} \right) \tag{1.77}$$

beim Drei-Gluon-Vertex in der QCD. Eine Zählerstruktur erhält man auch bei verschiedenen Drei-Boson-Vertizes und Vertizes mit Geist-Feldern im Standardmodell.

Expandiert man den Zähler und zieht konstante Faktoren (skalare Kopplungskonstanten, Massen, äußere Impulse, Elemente der Clifford- oder SU(3)-Algebra etc.) aus dem Integral, so sind alle zu lösenden Integrale von der Form

$$T^{[n_1,\ldots,n_L]} = \int \left(\prod_{l=1}^{L} d^D k_l \, k_l^{\mu_{l,1}} \cdots k_l^{\mu_{l,n_l}} \right) \prod_{i=1}^{N} \frac{1}{p_i^2 - m_i^2} \, . \tag{1.78}$$

In den folgenden Abschnitten werden zunächst allgemeine, bekannte Verfahren zur Berechnung von Tensor-Integralen vorgestellt sowie erläutert, warum sie nicht zur Berechnung von Zweiloop-Dreipunkt-Funktionen geeignet sind. Dabei wird auch auf die Besonderheiten in Verbindung mit der Parallel-/Orthogonalraum-Methode hingewiesen. Danach wird ein Verfahren vorgestellt, das aus zwei Stufen besteht, nämlich dem Kürzen von Propagatoren, um die Integrale auf eine Basismenge von letztlich zu berechnenden Funktionen zu reduzieren, und daran anschließend ein Abzugsverfahren, das die Berechnung in vier Dimensionen mit der Parallel-/Orthogonalraum-Methode ermöglicht. Zuletzt folgt eine detaillierte Betrachtung der verschiedenen auftretenden Dreipunkt-Topologien.

1.5.1 Bekannte Verfahren zur Berechnung von Tensor-Integralen

Tensor-Integrale vom Typ Gl. (1.78) werden in der Regel dadurch berechnet, daß man sie auf einen möglichst kleinen Satz von zu berechnenden Basisintegralen zurückführt. Das bekannteste Verfahren stammt von Passarino und Veltman [71]. Dort macht man zunächst einen Ansatz für den allgemeinsten Tensor, der sich aus den äußeren Impulsen bilden läßt, kontrahiert dann mit den einzelnen Termen des Ansatzes und löst das entstehende lineare Gleichungssystem nach den unbekannten Koeffizienten. Für einen Tensor 2. Stufe bei einer Einloop-Zweipunkt-Funktion mit einem unabhängigen Impuls q lautet dieser Ansatz z.B.

$$T^{\mu\nu} = \int d^D l \, \frac{l^\mu l^\nu}{((l+q)^2 - m_1^2)(l^2 - m_2^2)} = \int d^D l \, \frac{l^\mu l^\nu}{P_1 P_2} = A g^{\mu\nu} + B q^\mu q^\nu \tag{1.79}$$

mit der Lösung

$$\begin{aligned} A &= \frac{1}{(D-1)q^2} \int d^D l \, \frac{l^2 q^2 - (l\cdot q)^2}{P_1 P_2} \\ B &= \frac{1}{(D-1)(q^2)^2} \int d^D l \, \frac{D(l\cdot q)^2 - l^2 q^2}{P_1 P_2} \,. \end{aligned} \tag{1.80}$$

Damit hat man im Zähler der Integrale die freien Lorentzindizes eliminiert und hat dort nur noch Skalarprodukte der Loopimpulse und der äußeren Impulse. Diese Skalarprodukte kann man nun soweit wie möglich durch Linearkombinationen von Propagatoren, Massen und äußeren Impulsen ausdrücken, z.B.

$$l \cdot q = \frac{1}{2}(P_1 - P_2 + m_1^2 - m_2^2 - q^2) \,. \tag{1.81}$$

Dadurch kann man die Integrale durch Integrale gleicher Zählerstruktur, aber geringerer Anzahl Propagatoren, oder gleicher Anzahl Propagatoren, aber einfacherer Zählerstruktur, ausdrücken.

Das Passarino-Veltman-Verfahren ist weder in der Höhe des Tensorgrades, noch in der Anzahl äußerer Beine, noch in der Anzahl der Loopimpulse begrenzt. Die Komplexität nimmt jedoch wegen des Lösens des linearen Gleichungssystems stark mit dem Tensorgrad zu. Zudem kann es zu numerischen Instabilitäten kommen. Besonders geeignet ist es auf Einloop-Niveau, da für beliebige Anzahl äußerer Beine alle Tensor-Integrale durch die zugehörigen skalaren Integrale ausgedrückt werden können. Auf Zweiloop-Niveau ist dies im allgemeinen nicht mehr möglich, hier müssen zusätzliche Integrale berechnet werden.

Bei Zweiloop-Zweipunkt-Funktionen können die Tensor-Integrale der Master-Zweiloop-Zweipunkt-Funktion (Abb. 1.25a) auf die skalare Master-Zweiloop-Zweipunkt-Funktion

und Integrale mit weniger Propagatoren reduziert werden. Auf faktorisierende Topologien (z.B. Abb. 1.25e) läßt sich das Passarino-Veltman-Verfahren direkt anwenden, am schwierigsten sind die Topologien mit einem Zweipunkt-Untergraphen (Abb. 1.25b-d). Dort muß das Passarino-Veltman-Verfahren iterativ zuerst auf den Zweipunkt-Untergraphen angewendet werden, wobei man den Tensor-Ansatz statt mit dem äußeren mit dem zweiten Loopimpuls macht [90]. Die Determinanten beim Lösen des Gleichungssystem entsprechen zusätzlichen Propagatoren. Bei allgemeinen Zweiloop-Dreipunkt-Funktionen ist eine Reduzierung auf skalare Integrale mit diesem Verfahren nicht möglich.

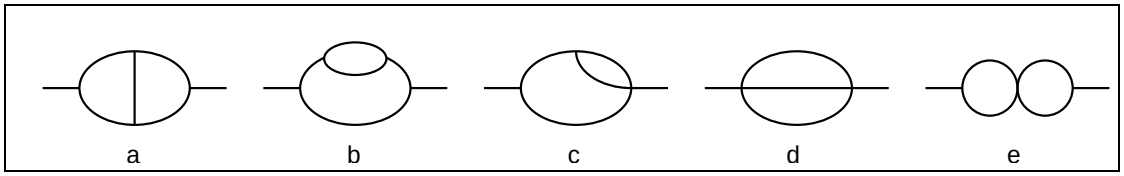


Abbildung 1.25: Einige Zweipunkt-Topologien

Eine Berechnung von Tensor-Integralen durch Verschieben der Raum-Zeit-Dimension D um Vielfache von 2 [23] ist bisher auch nur für Zweiloop-Zweipunkt-Funktionen ausgearbeitet worden. Eine Verbindung mit der hier benutzten Parallel-/Orthogonalraum-Methode führt zu Schwierigkeiten, da alle Rechenschritte eng mit der Dimension $D = 4$ verbunden sind.

In [52] wurde ein Verfahren vorgestellt, das die Berechnung von Zweiloop-Tensor-Integralen (bzw. beliebiger UV divergenter Zweiloop-Funktionen) für beliebige n-Punkt-Funktionen in einen in $D = 4$ Dimensionen konvergenten Beitrag und einen divergenten Beitrag aufteilt, der einfach genug ist, um in dimensionaler Regularisierung berechnet zu werden. Diese Aufteilung wurde so gewählt, daß der konvergente Beitrag mit der Parallel-/Orthogonalraum-Methode berechnet werden kann (vgl. Abschnitt 1.1.2). Die Idee ist dabei, für ein allgemeines Zweiloop-Integral mit Zählerstruktur

$$T = \int d^D l \int d^D k \, \frac{Z^{(z_l,z_k)}}{N_l M N_k}, \tag{1.82}$$

wobei N_l das Produkt aller n_l Propagatoren ist, die nur l enthalten, entsprechend N_k das Produkt aller n_k Propagatoren, die nur k enthalten, und M das Produkt aller m Propagatoren, die sowohl l als auch k enthalten (die Impulse können immer so gewählt werden, daß sie in der Kombination $l + k$ auftreten), und $Z^{(z_l,z_k)}$ ein Monom mit z_l Komponenten von l und z_k Komponenten von k (jeweils freie Lorentzindizes l^μ oder Parallel-/Orthogonalraum-Komponenten wie l_0). Das Integral ist dann divergent in vier Dimensionen, wenn mindestens einer der Divergenzgrade in den einzelnen Loopimpulsen ω_l, ω_k oder der Über-Alles-Divergenzgrad ω mit

$$\begin{aligned} \omega_l &= 2n_l + 2m - 4 - z_l \\ \omega_k &= 2n_k + 2m - 4 - z_k \\ \omega &= 2n_k + 2n_k + 2m - 8 - z_l - z_k \end{aligned} \tag{1.83}$$

Null (logarithmische Divergenz) oder negativ ist. Man führt nun masse- und impulsfreie

Propagatoren

$$\begin{aligned} \tilde{N}_l &= N_l\big|_{m_i=0,q_i=0} \\ \tilde{N}_k &= N_k\big|_{m_i=0,q_i=0} \\ \tilde{M} &= M\big|_{m_i=0,q_i=0} \end{aligned} \tag{1.84}$$

(q_i sind die äußeren Impulse) ein, definiert

$$\begin{aligned} L^{j_l} &= \frac{(\tilde{N}_l\tilde{M} - N_l M)^{j_l}}{(\tilde{N}_l\tilde{M})^{j_l}} = \left(1 - \frac{N_l M}{\tilde{N}_l\tilde{M}}\right)^{j_l} \\ K^{j_k} &= \frac{(\tilde{N}_k\tilde{M} - N_k M)^{j_k}}{(\tilde{N}_k\tilde{M})^{j_k}} = \left(1 - \frac{N_k M}{\tilde{N}_k\tilde{M}}\right)^{j_k} \end{aligned} \tag{1.85}$$

und wählt ganzzahlige j_l und j_k so, daß

$$\begin{aligned} \omega + j_l + j_k &> 0 \\ \omega_l + j_l &> 0 \\ \omega_k + j_k &> 0\,. \end{aligned} \tag{1.86}$$

Weiterhin definiert man $\hat{L}^{j_l}\hat{K}^{j_k}$ als nur die Terme in der Expandierung von $L^{j_l}K^{j_k}$, die in den folgenden Integralen nach *Powercounting* einzeln divergente Beiträge geben würden. Dann ist in

$$T = T_{\text{konv}} + T_{\text{div}} \tag{1.87}$$

mit

$$\begin{aligned} T_{\text{konv}} &= \int d^D l \int d^D k\, \frac{Z^{(z_l,z_k)}\hat{L}^{j_l}\hat{K}^{j_k}}{N_l M N_k} = \int d^4 l \int d^4 k\, \frac{Z^{(z_l,z_k)}\hat{L}^{j_l}\hat{K}^{j_k}}{N_l M N_k} + \mathcal{O}(\varepsilon) \\ T_{\text{div}} &= \int d^D l \int d^D k\, \frac{Z^{(z_l,z_k)}(1 - \hat{L}^{j_l}\hat{K}^{j_k})}{N_l M N_k} \end{aligned} \tag{1.88}$$

T_{konv} endlich und kann in $D = 4$ Dimensionen berechnet werden. Im Abzugsterm T_{div} treten dann nur Beiträge auf, bei denen alle Propagatoren, die l enthalten (N_l und M), oder alle Propagatoren, die k enthalten (N_k und M) durch $\tilde{N}_l$ und $\tilde{M}$ bzw. $\tilde{N}_k$ und $\tilde{M}$ ersetzt sind, gegebenenfalls auch in höheren Potenzen. Dadurch wird die l- oder die k-Integration vergleichsweise einfach, und auch die zweite Loopintegration läßt sich analytisch ausführen. Das Weglassen konvergenter Beiträge aus der Expandierung von $L^{j_l}K^{j_k}$ sowohl in T_{konv} als auch in T_{div} durch Einführung von $\hat{L}^{j_l}\hat{K}^{j_k}$ vermeidet außer mehr Rechenaufwand auch künstlich eingeführte Infrarot-Divergenzen durch masselose Propagatoren.

Der Abzugsterm, der in Abschnitt 1.2 für die UV divergente Dreipunkt-Funktion aus Abb. 1.4 gewählt wurde, ist ein Beispiel für diese Abzugsprozedur mit

$$\begin{aligned} \omega &= 2 \text{ (über-alles konvergent)} \\ \omega_l &= 4 \text{ (konvergent in } l) \\ \omega_k &= 0 \text{ (logarithmisch divergent in } k) \\ \Rightarrow j_l &= 0 \\ j_k &= 1\,. \end{aligned} \tag{1.89}$$

1.5.2 Probleme bei Zweiloop-Dreipunkt-Funktionen

Während das oben beschriebene Verfahren bereits prinzipiell erfolgreich in *xloops* für Zweiloop-Zweipunkt-Funktionen benutzt wird, gibt es einige Gründe, die gegen einen Einsatz bei Dreipunkt-Funktionen sprechen.

Mit der Erzeugung von Abzugstermen nach Gl. (1.88) wird die Berechnung von Tensor-Integralen nur insoweit vereinfacht, als die entstehenden Ausdrücke konvergent sind und daher nicht in D Dimensionen berechnet werden müssen, wodurch die in Abschnitt 1.1.2 beschriebenen Probleme vermieden werden. Eine Reduzierung der Anzahl der Integrale wird nicht erreicht, ebenso wie keine Vorschrift zum weiteren Vorgehen gegeben wird. Eine direkte Berechnung der Tensor-Integrale führt aber zu Problemen, wie das folgende Beispiel zeigt.

Mit der Parallel-/Orthogonalraum-Methode erhält man beim Berechnen von Feynman-Integralen, wie in Abschnitt 3.3.2 gezeigt wird, als Zählerstruktur keine Komponenten der Loopimpulse l und k mit freien Lorentzindizes l^μ, k^ν etc., sondern direkt die Parallel-/Orthogonalraum-Komponenten

$$T^{(p_0,p_1,p_\perp,r_0,r_1,r_\perp,u)} = \int d^D l \int d^D k \, \frac{l_0^{p_0} l_1^{p_1} (l_\perp^2)^{p_\perp} k_0^{r_0} k_1^{r_1} (k_\perp^2)^{r_\perp} (l_\perp k_\perp z)^u}{N_l M N_k} \,. \tag{1.90}$$

Der naheliegendste Ansatz ist, die Tensor-Integrale direkt zu berechnen, d.h. alle Schritte aus Abschnitt 1.1.1 auf den vollständigen Integranden inklusive Zähler anzuwenden, beispielsweise auf die planare Dreipunkt-Funktion mit Zähler

$$T_P^{(1,0,0,1,1,0,0)} = \int d^D l \int d^D k \, \frac{l_0 k_0 k_1}{P_1(l+q_1)P_2(l-q_2)P_3(l+k)P_4(k-q_1)P_5(k+q_2)P_6(k)} \tag{1.91}$$

mit

$$P_i(p) = p^2 - m_i^2 + i\eta \,, \tag{1.92}$$

die bereits konvergent in $D = 4$ Dimensionen ist und daher keinen Abzugsterm benötigt. Auf den Zähler $Z = l_0 k_0 k_1$ werden dann dieselben Transformationen wie auf den Nenner angewendet:

- Linearisierung der Propagatoren durch Gl. (1.7): $Z \to (l_0 + l_1)(k_0 + k_1)k_1$.
- Residuenintegration in l_1 und k_1, Ersetzen von l_1 und k_1 durch die Nullstelle des Propagators, z.B. für das Paar (P_1, P_5)

$$\begin{aligned} l_{1,\mathrm{Pol}} &= -\frac{1}{2} \frac{l_0^2 + 2 l_0 e_1 + e_1^2 - q_z^2 - s - m_1^2 + i\eta}{l_0 + e_1 - q_z} \\ k_{1,\mathrm{Pol}} &= -\frac{1}{2} \frac{k_0^2 + 2 k_0 e_2 + e_2^2 - q_z^2 - t - m_5^2 + i\eta}{k_0 + e_2 + q_z} \end{aligned} \tag{1.93}$$

$$\begin{aligned} Z \;\to\; & \frac{\left(-{k_0}^2 - 2 k_0 e_2 - {e_2}^2 + {q_z}^2 + t + m_5^2\right)\left({k_0}^2 - {e_2}^2 + {q_z}^2 + t + m_5^2 + 2 q_z k_0\right)}{4\left(k_0 + q_z + e_2\right)^2} \\ & \times \frac{\left(-2 l_0 q_z + {l_0}^2 - {e_1}^2 + {q_z}^2 + s + m_1^2\right)}{2\left(l_0 - q_z + e_1\right)} \,. \end{aligned} \tag{1.94}$$

- Die nun folgenden s- und t-Integrationen sind von der Form

$$\int_0^\infty dt\, \frac{(t+t_1-i\eta)(t+t_1'-i\eta)}{(t+t_0-i\eta)(t+t_0'-i\eta)} \int_0^\infty ds\, \frac{s+s_1-i\eta}{s+s_0-i\eta} \frac{1}{\sqrt[c]{(at+b+i\eta+cs)^2-4st}} \tag{1.95}$$

  und offensichtlich divergent sowohl für $s \to +\infty$ als auch für $t \to +\infty$.

Die direkte Berechnung der Tensoren ist daher nur mit einer zusätzlichen Regularisierung der s- und t-Integrale möglich.

Es gibt noch weitere Gründe, die gegen eine Anwendung des Verfahrens aus Gl. (1.88) sprechen:

- Wenn z.B. $N_l = ((l+q_1)^2 - m_1^2 + i\eta)(l^2 - m_2^2 + i\eta)$ ist, dann enthält ein Abzugsterm mit $\tilde{N}_l = (l^2+i\eta)^2$ einen quadrierten Propagator. Dieser führt bei der s-Integration auf Integrale vom Typ

$$\int_0^\infty ds \frac{1}{(s+s_0-i\eta)^2} \frac{1}{\sqrt{s^2-(\sigma_1+\sigma_2)s+\sigma_1\sigma_2}}\,. \tag{1.96}$$

  Nach einer Eulerschen Substitution $s \to u$ durch Gl. (1.15) transformiert sich das Integral zu

$$\int 2 \frac{2u-(\sigma_1+\sigma_2)}{(u^2+2s_0u-s_0(\sigma_1+\sigma_2)-\sigma_1\sigma_2)^2}\, du\,, \tag{1.97}$$

  also eine rationale Funktion deutlich höherer Komplexität verglichen mit Gl. (1.16). Für jede Potenz von $s+s_0-i\eta$ erhöht sich der Zählergrad um 1, der Nennergrad um 2. Während man hier das Ergebnis noch durch Ableiten des überschaubaren analytischen Ergebnisses nach s_0 erhalten kann, bereitet dies für die t-Integration große Probleme, da diese zwar analytisch berechnet wird, aber durch die Vielzahl von Fallunterscheidungen nicht in einer zum Ableiten geeigneten Form vorliegt.

- Wird in den gemischten Propagatoren M der Impuls Null gesetzt, so bedeutet dies, daß für den Ausgangs- und den Abzugsterm u.U. die Integrationskontur bei der l_1- und k_1-Integration unterschiedlich geschlossen werden muß, z.B. bei der gekreuzten Dreipunkt-Funktion abhängig vom Vorzeichen von $l_0+k_0-e_1$, für einen Abzugsterm dann abhängig vom Vorzeichen von l_0+k_0. Dadurch ist evtl. die Konvergenz dieser Integrationen nicht mehr manifest ohne Einführung eines zusätzlichen Regulators gesichert.

- Wird in den gemischten Propagatoren M der Impuls Null gesetzt, so gibt es auch Abzugsterme mit $M\tilde{M}$, die aufwendig partialbruchzerlegt werden müssen, mit allen zusätzlichen Problemen durch Pole auf der reellen l_1- bzw. k_1-Achse, die von der gekreuzten Dreipunkt-Funktion bekannt sind [30].

- Enthält ein Abzugsterm nur noch impulslose Propagatoren $\tilde{N}_l$ und $\tilde{N}_k$, so verschwinden alle Dreiecke, über die die numerischen l_0- und k_0-Integrationen ausgeführt werden müssen, was als Ergebnis naiverweise Null bedeuten würde, da die Konvergenz der Residuenintegration nicht überprüft wurde. Dieser Fall wird im nächsten Abschnitt näher untersucht.

1.5.3 Die Master-Zweipunkt-Funktion und der Grenzfall verschwindenden Impulses

Betrachtet man die Master-Zweiloop-Zweipunkt-Funktion Abb. 1.26(links)

$$I_M = \int d^D l \int d^D k \frac{1}{P_1 P_2 P_3 P_4 P_5} \tag{1.98}$$

mit Propagatoren

$$\begin{aligned} P_1 &= l^2 - m_1^2 + i\eta \\ P_2 &= (l-q)^2 - m_2^2 + i\eta \\ P_3 &= (l+k)^2 - m_3^2 + i\eta \\ P_4 &= k^2 - m_4^2 + i\eta \\ P_5 &= (k+q)^2 - m_5^2 + i\eta \end{aligned} \tag{1.99}$$

für den Grenzfall $q \to 0$, so nähert sich I_M dem analytisch bekannten, endlichen Wert der Zweiloop-Vakuumtopologie [17] Abb. 1.26(rechts) (nach einer Partialbruchzerlegung in m_1, m_2 bzw. m_4, m_5).

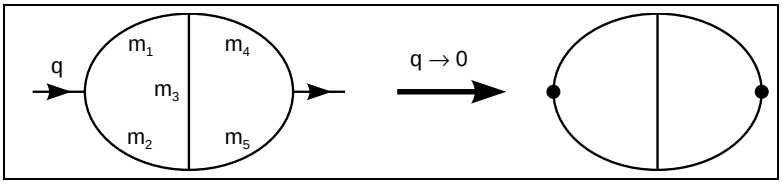


Abbildung 1.26: Die Master-Zweiloop-Zweipunkt-Topologie geht im Grenzfall $q \to 0$ in einen Vakuumgraphen über.

Setzt man aber in Gl. (1.99) direkt $q = 0$ und versucht, das Integral mit der Parallel-/Orthogonalraum-Methode zu berechnen, so gibt P_3 mit dem Vorzeichen von $l_0 + k_0 \gtrless 0$ die Halbebene vor, in der die Residuenintegrationen geschlossen werden. Von P_1 und P_2 erhält man dann als Bedingungen $l_0 \lesseqgtr 0$, von P_4 und P_5 Bedingungen $k_0 \lesseqgtr 0$. Alle Bedingungen für beitragende Gebiete sind also von der Form

$$\begin{array}{rcl} l_0 + k_0 & > & 0 \\ l_0 & < & 0 \\ k_0 & < & 0 \end{array} \quad \text{oder} \quad \begin{array}{rcl} l_0 + k_0 & < & 0 \\ l_0 & > & 0 \\ k_0 & > & 0\,, \end{array} \tag{1.100}$$

die jeweils zu einem Punkt entartete Dreiecke darstellen und daher scheinbar nicht beitragen, wie in Abschnitt 1.1.1.6 die Bedingungen aus dem Paar (P_1, P_4). Da $I_M\big|_{q=0}$ aber nicht verschwinden darf, muß der Integrand entsprechend divergieren, während die Fläche der Dreiecke bei $q \to 0$ schrumpft.

In der Tat erhält man, wenn man zunächst mit endlichem q rechnet, z.B. vom Propagatorpaar (P_1, P_5), das im Dreieck $l_0 + k_0 < 0$, $l_0 > 0$, $k_0 > -q$ beiträgt:

$$\begin{aligned} I_{M,1} &= C \int_0^q dl_0 \int_{-q}^{-l_0} dk_0 \int_0^\infty dt \, \frac{1}{t + t_0 - i\eta} \\ &\qquad \int_0^\infty ds \, \frac{1}{s + s_0 - i\eta} \frac{1}{\sqrt[c]{(at + b + i\eta + cs)^2 - 4st}} \,. \end{aligned} \tag{1.101}$$

Mit dem zweiten, ähnlichen Beitrag $I_{M,2}$ vom Propagatorpaar (P_2, P_4) ist $I_M = I_{M,1}+I_{M,2}$. Die relevanten Koeffizienten sind

$$\begin{aligned} C &= \frac{1}{4q^2} \\ s_0 &= \frac{l_0^2 q + (m_2^2 - m_1^2 - q^2) l_0 + m_1^2 q}{q} \\ t_0 &= \frac{k_0^2 q + (m_4^2 - m_5^2 + q^2) k_0 + m_4^2 q}{q} \,. \end{aligned} \tag{1.102}$$

Substituiert man

$$l_0 = q l_0' \quad \text{und} \quad k_0 = q k_0' \,, \tag{1.103}$$

so kann man den Grenzwert $q \to 0$ bilden:

$$\begin{aligned} I_{M,1}\big|_{q=0} &= C' \int_0^1 dl_0' \int_{-1}^{-l_0'} dk_0' \int_0^\infty dt \, \frac{1}{t + t_0' - i\eta} \\ &\qquad \int_0^\infty ds \, \frac{1}{s + s_0' - i\eta} \, \frac{1}{\sqrt[c]{(a't + b' + i\eta + c's)^2 - 4st}} \end{aligned} \tag{1.104}$$

mit

$$\begin{aligned} C' &= \frac{1}{4} \\ s_0' &= (m_2^2 - m_1^2) l_0' + m_1^2 \\ t_0' &= (m_4^2 - m_4^2) k_0' + m_4^2 \,, \end{aligned} \tag{1.105}$$

und man erhält ein endliches Ergebnis aus der numerischen Integration, das mit [17] übereinstimmt.

1.5.4 Kürzen von Propagatoren

Da das direkte Berechnen der Tensoren selbst für konvergente Dreipunkt-Funktionen wegen Gl. (1.95) nicht möglich ist, wird statt dessen für die Topologien in Abb. 1.27 zunächst versucht, durch Kürzen von Propagatoren problematische Fälle zu vermeiden.

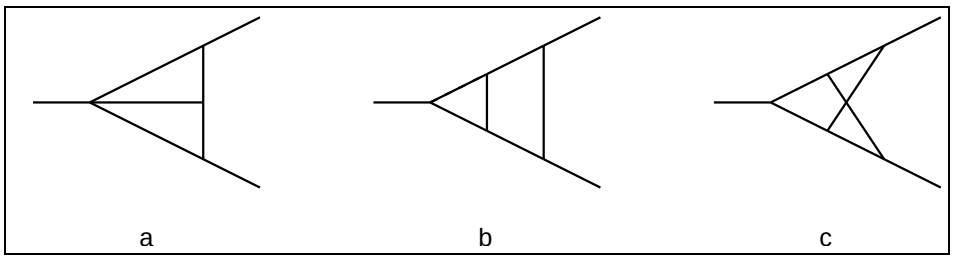


Abbildung 1.27: Die „schwierigen“ echten Zweiloop-Dreipunkt-Topologien

Die Zählerstruktur einer Zweiloop-Dreipunkt-Funktion, die von zwei linear unabhängigen äußeren Impulsen q_1 und q_2 abhängt, ist immer ein Polynom der sieben Lorentzinvarianten

$$l^2, \quad l \cdot q_1, \quad l \cdot q_2, \quad k^2, \quad k \cdot q_1, \quad k \cdot q_2, \quad l \cdot k\,. \tag{1.106}$$

Äquivalent dazu sind die sieben Parallel-/Orthogonalraum-Komponenten

$$l_0, \quad l_1, \quad l_\perp^2, \quad k_0, \quad k_1, \quad k_\perp^2, \quad l_\perp k_\perp z\,, \tag{1.107}$$

die mit den Invarianten aus Gl. (1.106) durch nichtlineare Transformationen in Verbindung stehen, ebenso beliebige unabhängige Kombinationen daraus.

Diesen sieben Invarianten stehen im ungünstigsten Fall (der reduzierten planaren Topologie Abb. 1.27a) aber nur fünf Propagatoren zum Kürzen entgegen, so daß man, wie bereits erwähnt, bei Zweiloop-Dreipunkt-Funktionen nicht alle Tensoren durch Kürzen auf skalare Integrale reduzieren kann.

Als günstigste Wahl für die beiden nicht-kürzbaren Invarianten erweisen sich die Linearkombinationen

$$(l_0 - l_1)^p \quad \text{und} \quad (k_0 - k_1)^r\,. \tag{1.108}$$

Diese haben den Vorteil, daß sie sich nach der Linearisierung der Propagatoren mit Gl. (1.7) zu l_0^p und k_0^r transformieren. Dieser Zähler bleibt während der l_1- und k_1-Integrationen unverändert und hat daher auch keinen Einfluß auf die Konvergenz und Durchführung der s- und t-Integrationen. Er muß erst bei der numerischen Integration über l_0 und k_0 berücksichtigt werden.

Das Kürzen der Tensoren wird in zwei Stufen durchgeführt. Zuerst werden die Orthogonalraum-Komponenten gekürzt, danach die Parallelraum-Komponenten.

1.5.4.1 Kürzen der Orthogonalraum-Komponenten

Alle Orthogonalraum-Komponenten $l_\perp^2$, $k_\perp^2$ und $l_\perp k_\perp z$ können für alle Topologien in Abb. 1.27 und Abb. 1.28 gekürzt werden, da N_l, N_k und M jeweils mindestens einen Propagator enthalten.

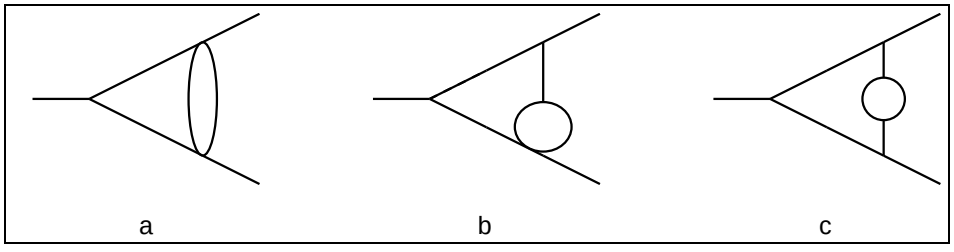


Abbildung 1.28: Die einfacheren echten Zweiloop-Dreipunkt-Topologien mit einem Zweipunkt-Untergraphen

Im ersten Schritt werden alle Vorkommnisse von $l_\perp k_\perp z$ gekürzt. Jeder Propagator in M ist von der Form $P_{l+k} = (l_0 + k_0 + p_0)^2 - (l_1 + k_1 + p_1)^2 - l_\perp^2 - k_\perp^2 - l_\perp k_\perp z - m^2$, und damit ist

$$l_\perp k_\perp z = (l_0 + k_0 + p_0)^2 - (l_1 + k_1 + p_1)^2 - l_\perp^2 - k_\perp^2 - m^2 - P_{l+k}\,, \tag{1.109}$$

und man erhält Terme gleicher Tensorstufe, aber in anderen Komponenten, Terme geringerer Tensorstufe bei gleichem Nenner und Terme geringerer Tensorstufe mit weniger Propagatoren, die in der Regel einfacher zu berechnen sind (bzw. rekursiv mit dem hier vorgestellten Verfahren).

Im zweiten Schritt wird $l_\perp^2$ gegen einen beliebigen Propagator aus N_l gekürzt, der von der Form $P_l = (l_0 + p_0)^2 - (l_1 + p_1)^2 - l_\perp^2 - m^2$ ist:

$$l_\perp^2 = (l_0 + p_0)^2 - (l_1 + p_1)^2 - m^2 - P_l \,. \tag{1.110}$$

Im dritten und letzten Schritt wird $k_\perp^2$ gegen einen Propagator aus N_k gekürzt:

$$k_\perp^2 = (k_0 + p_0)^2 - (k_1 + p_1)^2 - m^2 - P_k \,. \tag{1.111}$$

Nach diesen Schritten wurden sämtliche Orthogonalraum-Komponenten eliminiert, indem sie in die Parallelraum-Komponenten l_0, l_1, k_0 und k_1 verschoben wurden.

1.5.4.2 Kürzen der Parallelraum-Komponenten

Ein weiteres Kürzen in den Parallelraum-Komponenten k_0, k_1 ist nur möglich, wenn N_k mindestens zwei Propagatoren enthält, die nicht impulsentartet sind, d.h. daß die Differenz noch Komponenten von k enthält. Dasselbe gilt für l_0, l_1 mit N_l. Alle Topologien in Abb. 1.27 haben mindestens zwei nicht-entartete Propagatoren in N_l und N_k.

N_k enthalte die zwei Propagatoren

$$\begin{aligned} P_{k,a} &= (k + p_a)^2 - m_a^2 = (k_0 + p_{a,0})^2 - (k_1 + p_{a,1})^2 - k_\perp^2 - m_a^2 \\ P_{k,b} &= (k + p_b)^2 - m_b^2 = (k_0 + p_{b,0})^2 - (k_1 + p_{b,1})^2 - k_\perp^2 - m_b^2 \,, \end{aligned} \tag{1.112}$$

wobei p_a und p_b beliebige verschiedene Linearkombinationen der Impulse q_1 und q_2 sind (es ist auch ein $p_{a,b} = 0$ erlaubt).

Seien $\hat{n}_0 = (1, 0, \vec{0})$ und $\hat{n}_1 = (0, -1, \vec{0})$ die Einheitsvektoren entlang der beiden Parallelraum-Komponenten, also $k_0 = k \cdot \hat{n}_0$ und $k_1 = k \cdot \hat{n}_1$. Weiter sei $n = (1, 1, \vec{0})$ so gewählt, daß $k \cdot n = k_0 - k_1$ auf die gewünschte nicht-kürzbare Komponente aus Gl. (1.108) projiziert, und $p = p_a - p_b = (p_0, p_1, \vec{0})$. Alle diese Vektoren haben nur Komponenten im zweidimensionalen Parallelraum. Sowohl das Paar $\hat{n}_0$, $\hat{n}_1$ als auch n, p bildet eine Basis dieses Parallelraums (der Spezialfall, daß n und p linear abhängig sind, wird weiter unten behandelt). Durch eine Basistransformation

$$\begin{aligned} \hat{n}_0 &= \frac{p_1}{p_1 - p_0} n - \frac{1}{p_1 - p_0} p \\ \hat{n}_1 &= \frac{p_0}{p_1 - p_0} n - \frac{1}{p_1 - p_0} p \end{aligned} \tag{1.113}$$

lassen sich die verbleibenden Tensoren in den Parallelraum-Komponenten (vor der Linea-

risierung durch Gl. (1.7)) transformieren:

$$\begin{aligned}
k_0^{r_0} k_1^{r_1} &= (k\cdot\hat{n}_0)^{r_0}(k\cdot\hat{n}_1)^{r_1} \\
&= \left(k\cdot\left(\frac{p_1}{p_1-p_0}n-\frac{1}{p_1-p_0}p\right)\right)^{r_0}\left(k\cdot\left(\frac{p_0}{p_1-p_0}n-\frac{1}{p_1-p_0}p\right)\right)^{r_1} \\
&= \left(\sum_{i=0}^{r_0}\binom{r_0}{i}\left(\frac{p_1}{p_1-p_0}k\cdot n\right)^i\left(-\frac{1}{p_1-p_0}k\cdot p\right)^{r_0-i}\right) \\
&\quad\times\left(\sum_{j=0}^{r_1}\binom{r_1}{j}\left(\frac{p_0}{p_1-p_0}k\cdot n\right)^j\left(-\frac{1}{p_1-p_0}k\cdot p\right)^{r_1-j}\right) \\
&= \left(\frac{1}{p_1-p_0}\right)^{r_0+r_1}\sum_{l=0}^{r_0+r_1}(-1)^{r_0+r_1-l}(k\cdot n)^l(k\cdot p)^{r_0+r_1-l} \\
&\quad\times\sum_{m=\max(0,l-r_1)}^{\min(r_0,l)}\binom{r_0}{m}\binom{r_1}{l-m}p_1^m p_0^{l-m}\,. \qquad (1.114)
\end{aligned}$$

Da in der Differenz der beiden Propagatoren $P_{k,a}$ und $P_{k,b}$

$$\begin{aligned}
P_{k,a}-P_{k,b} &= 2k\cdot(p_a-p_b)-m_a^2+m_b^2 \\
&= 2k_0(p_{a,0}-p_{b,0})-2k_1(p_{a,1}-p_{b,1})-m_a^2+m_b^2 \qquad (1.115)
\end{aligned}$$

alle Orthogonalraum-Komponenten herausfallen, hat man nun die Möglichkeit, die Kombination $k\cdot p = k\cdot(p_a-p_b)$ gegen eine Differenz von Propagatoren zu kürzen

$$k\cdot p=\frac{1}{2}(P_{k,a}-P_{k,b}+m_a^2-m_b^2)\,, \qquad (1.116)$$

ohne neue Orthogonalraum-Komponenten einzuführen, und es bleiben nur noch Tensoren der Form $(k\cdot n)^r=(k_0-k_1)^r$ übrig. Dasselbe gilt für die Komponenten $l_0^{p_0}l_1^{p_1}$, die auf die Kombination $(l_0-l_1)^p$ reduziert werden können.

Enthält N_k drei oder mehr Propagatoren $P_{k,a}=(k+p_a)^2-m_a^2$, $P_{k,b}=(k+p_b)^2-m_b^2$ und $P_{k,c}=(k+p_c)^2-m_c^2$ und sind $p=p_a-p_b$ und $p'=p_a-p_c$ linear unabhängig, so kann man $k_0^{r_0}k_1^{r_1}$ nach $(k\cdot p)^r(k\cdot p')^{r'}$ entwickeln und alle Tensoren in k kürzen.

Der Spezialfall, daß n und p linear abhängig sind, bedeutet, daß $p=(p_0,p_1)\sim n=(1,1)$ ist, also $p_0=p_1$. Dies kann auftreten, wenn $p=q_1=(e_1,q_z)$ ist und der Grenzfall $q_1^2\to 0\Rightarrow q_z\to e_1$ betrachtet wird. In diesem Fall kann man statt nach den Basisvektoren p und n nach p und $n'=(1,-1)$ entwickeln. Als Transformation, die die Propagatoren linearisiert (Gl. (1.7)), wählt man dann $l_0\to l_0-l_1$ bzw. $k_0\to k_0-k_1$. Die Transformation kann für l und k separat gewählt werden. Welche die geeignete ist, wird in den Abschnitten über die einzelnen Topologien angegeben.

1.5.4.3 Zusammenfassung

Alle Tensor-Integrale vom Typ Gl. (1.90) für die Topologien aus Abb. 1.27 lassen sich durch Kürzen auf Integrale vom Typ

$$T^{(p_0,p_1,p_\perp,r_0,r_1,r_\perp,u)}\to\int d^Dl\int d^Dk\,\frac{(l_0-l_1)^{p_0+p_1+2p_\perp+u}\,(k_0-k_1)^{r_0+r_1+2r_\perp+u}}{N_lMN_k} \qquad (1.117)$$

derselben Topologie sowie Tensor-Integrale von Topologien mit weniger Propagatoren und geringerer Tensorstufe — für jeden gekürzten Propagator mindestens eine Stufe weniger — reduzieren. Für die gekürzten Topologien können folgende Fälle auftreten:

- wieder eine Topologie aus Abb. 1.27 (genau genommen kann nur Abb. 1.27a auftreten): Das Verfahren kann rekursiv angewendet werden (Abb. 1.29).
- eine Topologie aus Abb. 1.28: Diese lassen sich effizienter mit Hilfe von Dispersionsrelationen berechnen (Abschnitt 1.5.9). (Abb. 1.29)
- eine effektive Zweipunkt-Topologie wie z.B. Abb. 1.30(rechts): Nach einer Reduzierung des Parallelraums (Abschnitt 3.2) können diese mit der Zweipunkt-Methode [27] berechnet werden.
- eine faktorisierende Topologie wie z.B. Abb. 1.30(links): Diese Topologien sind vergleichsweise trivial und bereiten keine weiteren Probleme.

Das Kürzen kann mit den in Abschnitt 3.1 erzeugten Informationen voll automatisiert werden.

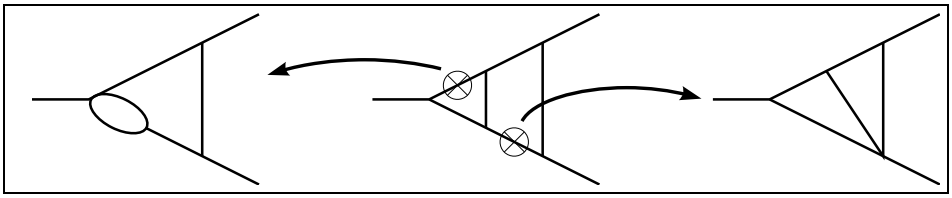


Abbildung 1.29: Beispiel für das Kürzen von Propagatoren, bei dem das Ergebnis eine Topologie mit Zweipunkt-Untergraph (links) oder die reduzierte planare Topologie (rechts) ist

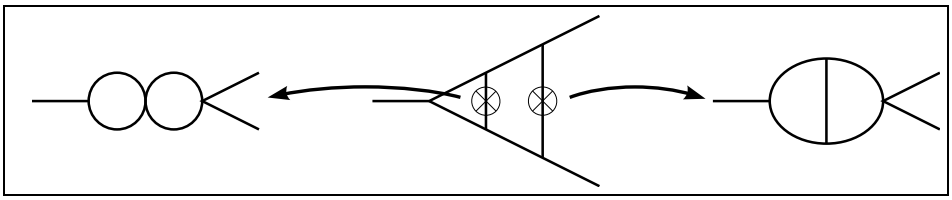


Abbildung 1.30: Beispiel für das Kürzen von Propagatoren, bei dem das Ergebnis eine faktorisierende Topologie (links) oder eine effektive Zweiloop-Zweipunkt-Topologie (rechts) ist

1.5.5 Das Abzugsverfahren

Das Abzugsverfahren für divergente Zweiloop-Dreipunkt-Funktionen zu Topologien aus Abb. 1.27 sollte zum einen sicherstellen, daß keine quadrierten Propagatoren entstehen (wegen Gl. (1.97)), zum anderen, daß die Konvergenz der l_1-, k_1-, s- und t-Integrationen nicht gestört wird und zuletzt der Grenzübergang $q_i \to 0$ kontrolliert werden kann. Außerdem soll unnötiger Rechenaufwand durch überflüssige Abzugsterme vermieden werden.

An der Feststellung der Divergenzgrade ω_l, ω_k und ω in l, k und über-alles ändert sich nichts gegenüber Gl. (1.83), ebenso an der Notwendigkeit, die Größen j_l und j_k wie in Gl. (1.86) zu wählen.

Anstelle der Multiplikatoren L^{j_l} und K^{j_k} in Gl. (1.85) werden jedoch Größen eingeführt, die keine quadrierten Propagatoren erzeugen. L^{j_l} wurde so konstruiert, daß in allen Abzugstermen sowohl die Propagatoren, die nur l enthalten, als auch die, die $l+k$ enthalten, frei von Impulsen und Massen sind, damit die l- und die anschließende k-Integration analytisch ausgeführt werden können. Dies ist aber nach Abschnitt 1.1.2 nicht unbedingte Voraussetzung für Abzugsterme, solange diese noch mit anderen Methoden berechnet werden können.

Teilt man

$$N_l = N_l^{(p)} N_l^{(0)} = \prod_{i=1}^{n_l^{(p)}} ((l+p_i)^2 - m_i^2) \prod_{j=1}^{n_l^{(0)}} (l^2 - m^2_{n_l^{(p)}+j}) \tag{1.118}$$

in die Propagatoren mit einem äußeren Impuls ($p_i \neq 0$) und die ohne auf (in den relevanten Fällen ist $n_l^{(0)} \leq 1$) und konstruiert man

$$N_{l,r}^{(p)} = \prod_{i=1}^{n_l^{(p)}} ((l+\lambda_r p_i)^2 - m_{i,r}^2) \tag{1.119}$$

sowie

$$\tilde{L}^{(j_l)} = \prod_{r=1}^{j_l} \frac{N_{l,r}^{(p)} - N_l^{(p)}}{N_{l,r}^{(p)}} = \prod_{r=1}^{j_l} \left(1 - \frac{N_l^{(p)}}{N_{l,r}^{(p)}}\right) , \tag{1.120}$$

dann verbessern sich durch Multiplikation mit $\tilde{L}^{(j_l)}$ die Divergenzgrade ω und ω_l um mindestens j_l. Die Massen $m_{i,r}$ werden alle verschieden, aber ansonsten beliebig gewählt, um quadrierte Propagatoren zu vermeiden. Die Unabhängigkeit des Verfahrens von der konkreten Wahl der Massen gibt eine Kontrollmöglichkeit des Ergebnisses. λ_r wird während der l_1- und k_1-Integration endlich gehalten. Am Ende wird der Grenzübergang $\forall r = 1, \ldots, j_l : \lim_{\lambda_r \to 0} \tilde{L}^{(j_l)}$ gebildet.

Analog ist

$$\begin{aligned}
N_k &= N_k^{(p)} N_k^{(0)} = \prod_{i=1}^{n_k^{(p)}} ((k+p_i)^2 - m_i^2) \prod_{j=1}^{n_k^{(0)}} (k^2 - m^2_{n_k^{(p)}+j}) \\
N_{k,r}^{(p)} &= \prod_{i=1}^{n_k^{(p)}} ((k+\kappa_r p_i)^2 - m_{i,r}^2) \\
\tilde{K}^{(j_k)} &= \prod_{r=1}^{j_k} \frac{N_{k,r}^{(p)} - N_k^{(p)}}{N_{k,r}^{(p)}} = \prod_{r=1}^{j_k} \left(1 - \frac{N_k^{(p)}}{N_{k,r}^{(p)}}\right) ,
\end{aligned} \tag{1.121}$$

so daß das Produkt

$$\frac{1}{N_l M N_k} \tilde{L}^{(j_l)} \tilde{K}^{(j_k)} \tag{1.122}$$

bei Integration in $D = 4$ Dimensionen endlich ist. Damit wird ein Tensor-Integral analog zu Gl. (1.88) in einen konvergenten und einen divergenten Anteil aufgespalten:

$$T = T_{\text{konv}} + T_{\text{div}} \tag{1.123}$$

mit

$$\begin{aligned} T_{\text{konv}} &= \int d^D l \int d^D k \, \frac{Z^{(z_l,z_k)} \hat{\tilde{L}}^{(j_l)} \hat{\tilde{K}}^{(j_k)}}{N_l M N_k} = \int d^4 l \int d^4 k \, \frac{Z^{(z_l,z_k)} \hat{\tilde{L}}^{(j_l)} \hat{\tilde{K}}^{(j_k)}}{N_l M N_k} + \mathcal{O}(\varepsilon) \\ T_{\text{div}} &= \int d^D l \int d^D k \, \frac{Z^{(z_l,z_k)} \left(1 - \hat{\tilde{L}}^{(j_l)} \hat{\tilde{K}}^{(j_k)}\right)}{N_l M N_k} \,. \end{aligned} \tag{1.124}$$

Konvergente Terme in der Expandierung von $\tilde{L}^{(j_l)} \tilde{K}^{(j_k)}$ werden wieder in T_{konv} und T_{div} weggelassen. Bei endlichen $m_{i,r}$ besteht keine Gefahr von Infrarot-Divergenzen.

Beim Multiplizieren mit $\tilde{L}^{(j_l)}$ gibt es für $j_l > 1$ durch die Differenz $N_{l,r}^{(p)} - N_l^{(p)}$ eine zusätzliche Zählerstruktur $l \cdot p = l_0 p_0 - l_1 p_1$ in den Abzugstermen, die sich aber nicht mehr kürzen läßt, da hier alle l-Propagatoren frei von Impulsen sind. Durch das Einsetzen der Pole aus der l_1-Residuenintegration gibt es im Zähler auch zusätzliche Potenzen von s. Diese stören aber die Konvergenz der s-Integration nicht, da sie immer nur in Verbindung mit einem zusätzlichen $1/(s + s_0')$ Term von $1/N_{l,r}^{(p)}$ auftreten. Eine Partialbruchzerlegung

$$\frac{s + s_1}{(s + s_0)(s + s_0')} = \frac{s_0 - s_1}{(s_0 - s_0')(s + s_0)} - \frac{s_0' - s_1}{(s_0 - s_0')(s + s_0')} \tag{1.125}$$

eliminiert das zusätzliche s im Zähler. Alternativ dazu kann man bereits vor der l_1-Integration eine Partialbruchzerlegung in l_1 machen, die auf dasselbe Ergebnis führt.

Da die Topologien in Abb. 1.27 jeweils mindestens zwei Propagatoren in N_l und N_k haben, sind alle Abzugsterme in l oder k frei von äußeren Impulsen und damit impulsentartet. Da außerdem alle Massen verschieden gewählt wurden, kann man diese Propagatoren partialbruchzerlegen. Somit haben alle Abzugsterme mindestens einen Propagator weniger und sind daher einfacher zu berechnen. In den folgenden Abschnitten werden die nichtfaktorisierenden echten Dreipunkt-Topologien nach absteigender Anzahl von Propagatoren sortiert behandelt.

1.5.6 Die planare Dreipunkt-Funktion

Bei der planaren Dreipunkt-Funktion (Abb. 1.27b), mit den sechs Propagatoren $P_1(l+q_1)$, $P_2(l - q_2)$, $P_3(l + k)$, $P_4(k - q_1)$, $P_5(k + q_2)$ und $P_6(k)$, tritt im Standardmodell der elektroschwachen Wechselwirkung in der Feynman-Eichung mit $\xi = 1$ höchstens ein Tensor 5. Stufe auf, z.B. bei dem Graphen in Abb. 1.31, wobei maximal ein Tensor 3. Stufe auf den Loopimpuls l und maximal 4. Stufe auf k entfällt. Da N_k drei Propagatoren enthält, kann man alle k-Tensorkomponenten kürzen. N_l enthält nur zwei Propagatoren, und es bleiben Integrale vom Typ

$$T_P = \int d^D l \int d^D k \, \frac{(l_0 - l_1)^p}{P_1 P_2 P_3 P_4 P_5 P_6} \,, \quad 0 \le p \le 3 \tag{1.126}$$

sowie Integrale mit weniger Propagatoren (Abb. 1.29 und Abb. 1.30). Die Divergenzgrade berechnen sich zu $\omega = 4 - p$, $\omega_l = 2 - p$ und $\omega_k = 4$, damit wählt man $j_l = \max(0, p - 1)$,

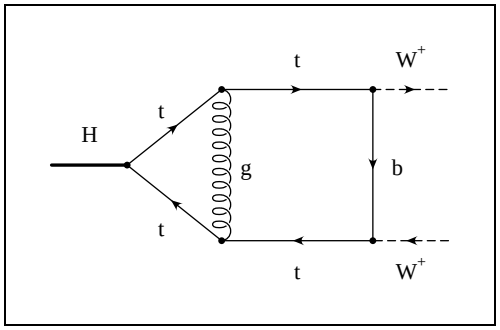


Abbildung 1.31: Beispiel für einen Graphen der Tensorstufe 5 bei der planaren Topologie

$j_k = 0$. Im Standardmodell mit maximal $p = 3$ ist $\omega = 1$, $\omega_l = -1$ und $\omega_k = 4$, also $j_l = 2$ und $j_k = 0$. Höhere Tensoren lassen sich natürlich auch berechnen.

Beim Grenzübergang $\lambda_r \to 0$ entarten in den Abzugstermen, die mit Parallel-/Orthogonalraum-Integration berechnet werden, einige Dreiecke, z.B. von $(\tilde{P}_1, P_6)$, zu einem Punkt, ohne daß der Integrand singulär wird. Dafür tragen nun andere, vorher entartete Paare bei, z.B. $(\tilde{P}_1, P_4)$. Eine Komplikation entsteht noch dadurch, daß im Abzugsterm

$$\tilde{P}_2 = \tilde{P}_1 + m_{1,r}^2 - m_{2,r}^2 + \mathcal{O}(\lambda_r) \tag{1.127}$$

ist. Setzt man dann bei der l_1-Residuenintegration (Abschnitt 1.1.1.5) den Pol von P_1, $l_1^{(1)}$, in P_2 ein

$$\tilde{P}_2\big|_{l_1 = l_1^{(1)}} = m_{1,r}^2 - m_{2,r}^2 + \mathcal{O}(\lambda_r)\,, \tag{1.128}$$

so erhält man im Limes $\lambda_r \to 0$ keinen Term mit $1/(s + s_0 - i\eta)$, sondern Integrale vom Typ Gl. (1.31), wie bei der ultraviolett divergenten Dreipunkt-Funktion aus Abschnitt 1.2. Der Term mit der zweiten Wurzel kommt dabei nicht von einem explizit konstruierten Abzugsterm mit masselosen Teilchen, sondern von einem anderen Propagatorpaar, das im selben Dreieck beiträgt. Alle beitragenden Gebiete für den Abzugsterm bei $j_l = 1$ im Limes $\lambda_r \to 0$ findet man in Abb. 1.32. Außerdem erhält aber ab $j_l \geq 2$ in den Abzugstermen eine zusätzliche Zählerstruktur mit $l \cdot q_1$ und $l \cdot q_2$, die nicht gekürzt werden kann.

Mit $N_l = P_1 P_2$ ist $p = q_1 + q_2 = (e_1 + e_2, 0, \vec{0})$. Daher tritt der Fall $n \sim p$ nicht auf. Bei der gedrehten Variante der planaren Dreipunkt-Funktion [30] hingegen ist $N_l = P_1(l) P_2(l + q_1)$ und somit $p \sim q_1$. Hier sollte deshalb $l_0 \to l_0 - l_1$ zum Linearisieren benutzt werden. Ansonsten verhält sie sich ähnlich wie die nicht-gedrehte Variante.

In der Berechnung des divergenten Teils T_{div} treten nur Dreipunkt-Funktionen mit einem Zweipunkt-Untergraphen (Abschnitt 1.5.9) auf, wenn man eine Partialbruchzerlegung in den verschiedenen Propagatoren der Form $1/(l^2 - m_{i,r}^2)$ macht. Dies erkennt man, wenn man die Propagatoren neu anordnet (Abb. 1.33).

1.5.7 Die gekreuzte Dreipunkt-Funktion

Bei der gekreuzten Dreipunkt-Funktion (Abb. 1.27c), die ebenfalls sechs Propagatoren $P_1(l + k - q_1)$, $P_2(l + k + q_2)$, $P_3(l - q_1)$, $P_4(k + q_2)$, $P_5(l)$ und $P_6(k)$ hat, treten im

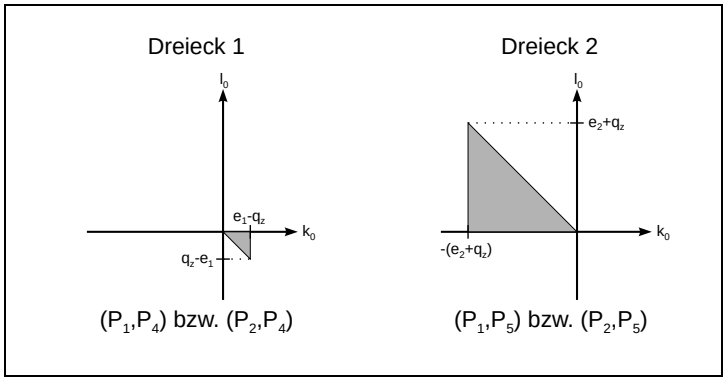


Abbildung 1.32: Einzelne Integrationsgebiete für den Abzugsterm zu $j_l = 1$ für die planare Dreipunkt-Funktion

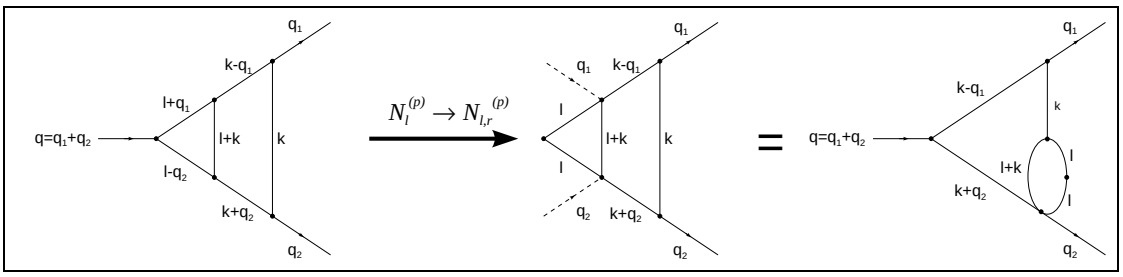


Abbildung 1.33: Die Abzugsterme der planaren Dreipunkt-Funktion haben einen Zweipunkt-Untergraphen

Standardmodell ebenfalls höchstens Tensoren 5. Stufe auf, z.B. in Abb. 1.34. Dabei kann jeweils ein Tensor 4. Stufe für l oder k auftreten. Hier lassen sich weder alle k- noch l-Tensoren kürzen, da N_l und N_k jeweils nur zwei Propagatoren enthalten. Da man mit $P_3(l - q_1)$ und $P_5(l)$ nur $l \cdot q_1$ kürzen kann, $l \cdot q_1$ aber proportional zu $n = (1, 1)$ werden kann, sollte man, abweichend von [30], die Propagatoren mit $l_0 \to l_0 - l_1$ und $k_0 \to k_0 + k_1$ linearisieren. Damit bleiben nach dem Kürzen Integrale der Form

$$T_C = \int d^D l \int d^D k \, \frac{(l_0 + l_1)^p \, (k_0 - k_1)^r}{P_1 P_2 P_3 P_4 P_5 P_6} \,, \quad 0 \le p \le 4 \,, \quad 0 \le r \le 4 \,, \quad p + r \le 5 \qquad (1.129)$$

sowie Integrale mit einem Propagator weniger, die wegen der Symmetrie der gekreuzten Dreipunkt-Funktion alle vom Typ der reduzierten planaren Dreipunkt-Funktion sind. Die Divergenzgrade sind $\omega = 4 - p - r$, $\omega_l = 4 - p$ und $\omega_k = 4 - r$. Der divergenteste Fall, $p = 4$ und $r = 1$ (analog $p = 1$ und $r = 4$), gibt $\omega = -1$, $\omega_l = 0$ und $\omega_k = 3$. Die Wahl $j_l = 1$ und $j_k = 1$ macht das Integral daher endlich. Dies gilt auch für die anderen Fälle mit $p + r = 5$. Für $p + r = 4$ wählt man $j_l = 1$ und $j_k = 0$, wenn $p \ge r$, ansonsten $j_l = 0$ und $j_k = 1$. Die übrigen Tensoren mit $p + r \le 3$ sind konvergent.

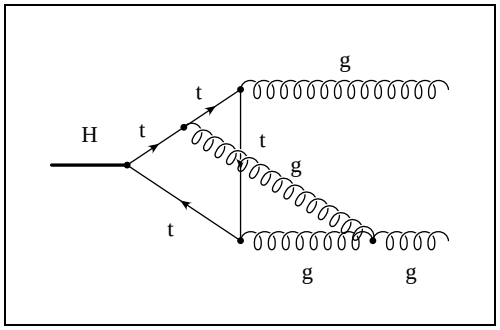


Abbildung 1.34: Beispiel für einen Graphen der Tensorstufe 5 bei der gekreuzten Topologie

Die Abzugsterme 1 und 2 in

$$\begin{aligned}\frac{\tilde{L}^{(1)}\tilde{K}^{(1)}}{P_1P_2P_3P_4P_5P_6} &= \frac{1}{P_1P_2P_3P_4P_5P_6}\left(1-\frac{P_3(l)}{\tilde{P}_3(l-q_1)}\right)\left(1-\frac{P_4(k+q_2)}{\tilde{P}_4(k)}\right)\\ &= \underbrace{\frac{1}{P_1P_2P_3P_4P_5P_6}}_{\text{normal}}-\underbrace{\frac{1}{P_1P_2\tilde{P}_3P_4P_5P_6}}_{\text{Abzug1}}\\ &\quad -\underbrace{\frac{1}{P_1P_2P_3\tilde{P}_4P_5P_6}}_{\text{Abzug2}}+\underbrace{\frac{1}{P_1P_2\tilde{P}_3\tilde{P}_4P_5P_6}}_{\text{Abzug3}}\end{aligned} \tag{1.130}$$

(Abb. 1.35) sind vom Typ der reduzierten planaren Dreipunkt-Funktion (Abschnitt 1.5.8) mit einem quadrierten Propagator, die hier im Gegensatz zu [30] mit zwei Propagatoren, die $l+k$ enthalten, berechnet werden muß (vgl. auch Abschnitt 1.3.2). Das Programm zum Bestimmen der beitragenden Gebiete und Koeffizienten wird in Anhang B beschrieben. Für Abzugsterm 1 beispielsweise erhält man die Gebiete aus Abb. 1.36. Abzugsterm 3 hat, nach Partialbruchzerlegung, einen Zweipunkt-Untergraphen (Abschnitt 1.5.9).

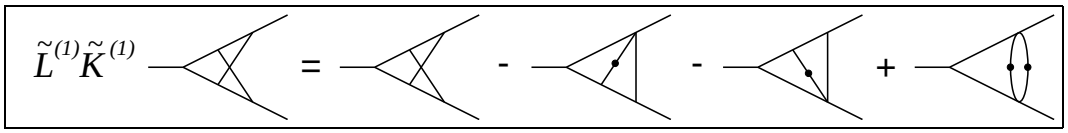


Abbildung 1.35: Abzugsterme bei der gekreuzten Dreipunkt-Funktion

1.5.8 Die reduzierte planare Dreipunkt-Funktion

Die reduzierte planare Dreipunkt-Funktion (Abb. 1.27a), mit fünf Propagatoren $P_1(l+q_1)$, $P_2(l)$, $P_3(l+k)$, $P_4(k+q_2)$ und $P_5(k)$, ist in Bezug auf die Berechnung von Tensor-Integralen die schwierigste Dreipunkt-Topologie, da sie von den Topologien in Abb. 1.27 zum einen die wenigsten Propagatoren hat und damit die geringsten Möglichkeiten zum Kürzen bietet, zum anderen mit dem höchsten Divergenzgrad auftritt. Man erhält sie nicht nur in direkten Graphen mit dieser Topologie — durch die Vierpunkt-Kopplung tritt sie im Vergleich zur

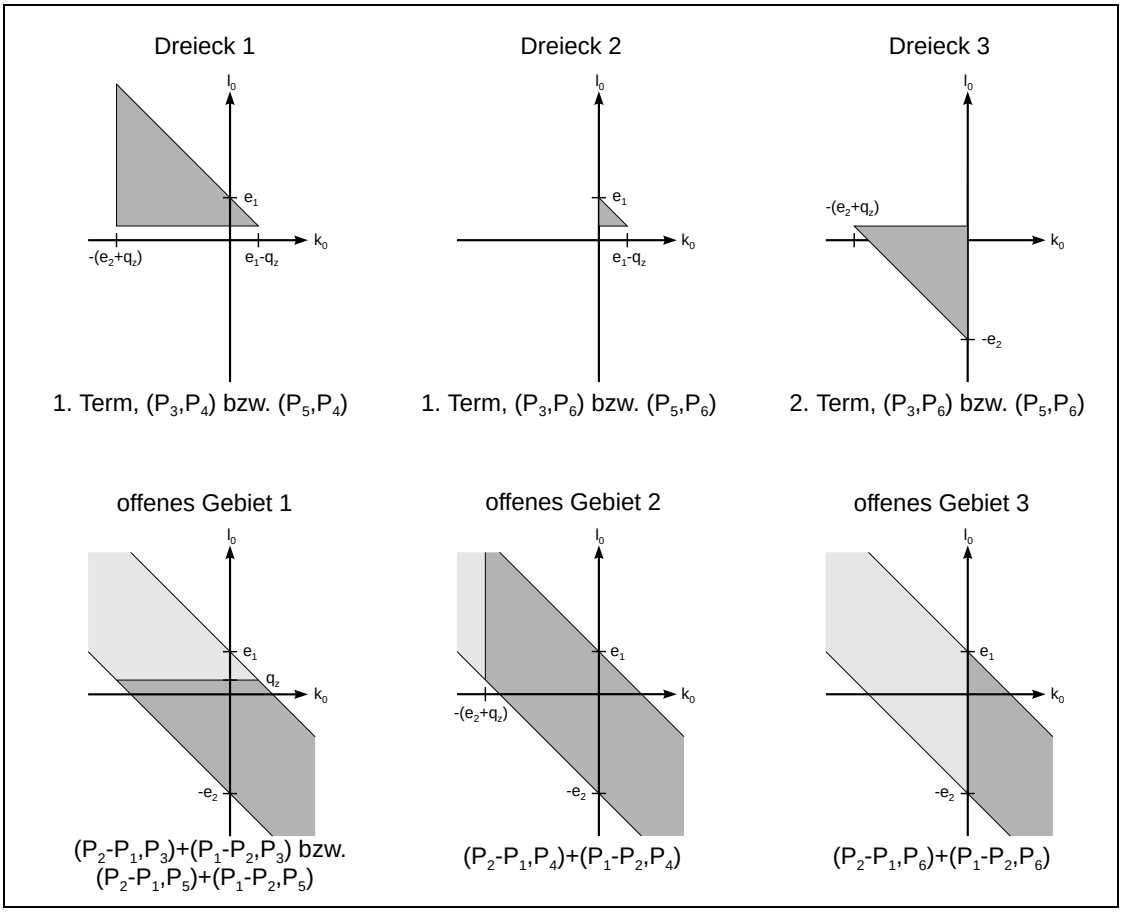


Abbildung 1.36: Einzelne Integrationsgebiete für Abzugsterm 1 zu $j_l = 1$, $j_k = 1$ für die gekreuzte Dreipunkt-Funktion

planaren oder gekreuzten Topologie relativ selten auf, da sie keinen Fermionloop enthalten kann —, sondern auch durch das Kürzen von Propagatoren (P_4 und P_5 bei der planaren, eines beliebigen bei der gekreuzten Dreipunkt-Funktion) und als Abzugsterm divergenter gekreuzter Dreipunkt-Funktionen.

Die kürzbaren Impulse in N_l und N_k legen, ebenfalls abweichend von [30], eine Linearisierung mit $l_0 \to l_0 - l_1$ und $k_0 \to k_0 + k_1$ nahe. Im schwierigsten Fall tritt im Standardmodell ein Tensor 4. Stufe auf, wobei maximal ein Tensor 3. Stufe je auf l oder k entfallen kann. Damit ist z.B. $\omega = -2$, $\omega_l = -1$ und $\omega_k = 1$, so daß mit $j_l = 2$, $j_k = 1$ das Integral endlich wird. Damit bleiben Integrale

$$T_R = \int d^D l \int d^D k \, \frac{(l_0 + l_1)^p \, (k_0 - k_1)^r}{P_1 P_2 P_3 P_4 P_5} \,, \quad 0 \le p \le 3 \,, \quad 0 \le r \le 3 \,, \quad p + r \le 4 \,. \quad (1.131)$$

Als Abzugsterm der gekreuzten Dreipunkt-Funktion erhält man zwar sogar einen Tensor 5. Stufe, der dann vollständig in l oder k sein kann, da eine Impulstransformation, z.B. $l \to l - k$ ausgeführt werden muß, um einen der beiden $l + k$-Propagatoren zu eliminieren. Der Abzugsterm hat aber einen zusätzlichen Propagator, so daß dort effektiv ω nur -1 ist und keine höheren $\tilde{L}^{(j_l)}$ benötigt werden.

Die Abzugsterme, die durch Multiplikation mit $\tilde{L}^{(j_l)}$ bzw. $\tilde{K}^{(j_k)}$ entstehen, sind nach

Partialbruchzerlegung wieder Dreipunkt-Funktionen (die nur noch entweder von q_1 oder von q_2 abhängen und somit sogar effektiv nur Zweipunkt-Funktionen sind) mit einem Zweipunkt-Untergraphen, die im folgenden Abschnitt behandelt werden bzw. Vakuum-Graphen, die von keinem äußeren Impuls mehr abhängen [17] (Abb. 1.37).

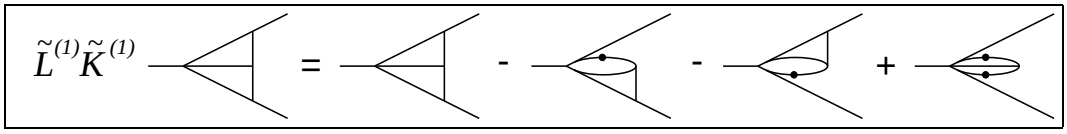


Abbildung 1.37: Abzugsterme bei der reduzierten planaren Dreipunkt-Funktion

Analog zu Abschnitt 1.5.3 treten bei der Berechnung mit Parallel-/Orthogonalraum-Integrationen Terme auf, die im Grenzfall $\lambda_r \to 0 \wedge \kappa_r \to 0$ nur noch Beiträge in zu Punkten schrumpfenden Dreiecken geben. Hier sind sie jedoch ungefährlich, da ihr Zähler mindestens einen Faktor l_0 oder k_0 (nach Linearisierung) enthält, der nach einer Substitution analog zu Gl. (1.103) eine zusätzliche Potenz von λ_r bzw. κ_r im Zähler gibt.

1.5.9 Dreipunkt-Funktionen mit einem Zweipunkt-Untergraphen

Bei den drei verbleibenden nicht-faktorisierenden Dreipunkt-Topologien aus Abb. 1.28, von denen a und b auch als Abzugsterme der Topologien aus Abb. 1.27 auftreten, lassen sich zwar die skalaren Diagramme mit dem Parallel-/Orthogonalraum-Verfahren berechnen (Abschnitt 1.2). Mit einer zusätzlichen Zählerstruktur tritt jedoch das Problem auf, daß noch weniger Propagatoren als in Abschnitt 1.5.8 zum Kürzen zur Verfügung stehen, so daß z.B. bei der Topologie Abb. 1.28a drei nicht-kürzbare Invarianten aus Gl. (1.106) bzw. Gl. (1.107) übrig bleiben. Die direkte Berechnung der Tensor-Integrale ist jedoch wegen Gl. (1.95) nicht möglich.

Dies ist aber auch gar nicht nötig, da es für die Zweipunkt- und Dreipunkt-Topologien mit einem Zweipunkt-Untergraphen ein Verfahren gibt, das auf eine eindimensionale numerische Integration, im Vergleich zur zweidimensionalen bei der Parallel-/Orthogonalraum-Methode, führt und in [75] für die Topologie in Abb. 1.28a vorgestellt wurde. Die wesentlichen Schritte sind:

- Betrachtet werden divergente Integrale vom Typ

$$T^{(\alpha_0,\alpha_1,\beta)} = \int d^D l \int d^D k \frac{k_0^{\alpha_0} k_1^{\alpha_1} l_0^{\beta}}{P_1(l+q_1)P_2(l-q_2)P_3(l+k)P_4(k)} \,. \tag{1.132}$$

- Mit Hilfe einer Taylor-Entwicklung in den Impulsen

$$\mathcal{T}^{(r)}_{p_1,p_2,\dots} \frac{Z(p_1,p_2,\dots)}{N(p_1,p_2,\dots)} = \sum_{j=0}^{r-1} \frac{1}{j!} \left(\frac{d}{d\rho}\right)^j \left.\frac{Z(p_1,p_2,\dots)}{N(\rho p_1,\rho p_2,\dots)}\right|_{\rho=0} , \tag{1.133}$$

  die zuerst die Divergenz in k und dann in l abzieht, ist

$$\begin{aligned} T^{(\alpha_0,\alpha_1,\beta)}_{\text{konv}} &= (1-\mathcal{T}^{(1+\alpha_0+\alpha_1+\beta)}_{q_1,q_2}) \int d^D l \, \frac{l_0^{\beta}}{P_1(l+q_1)P_2(l-q_2)} \\ &\quad \times (1-\mathcal{T}^{(\alpha_0+\alpha_1+1)}_{l}) \int d^D k \, \frac{k_0^{\alpha_0} k_1^{\alpha_1}}{P_3(l+k)P_4(k)} \end{aligned} \tag{1.134}$$

endlich und kann in $D = 4$ Dimensionen berechnet werden.

- Die k-Integration kann in ein Dispersionsintegral umgewandelt werden:

$$(1 - \mathcal{T}_l^{(\alpha_0+\alpha_1+1)}) \int d^D k \, \frac{k_0^{\alpha_0} k_1^{\alpha_1}}{P_3(l+k)P_4(k)} = \int\limits_{(m_3+m_4)^2}^{\infty} d\zeta \frac{\sqrt{\lambda(\zeta, m_3^2, m_4^2)} f_{\alpha_0,\alpha_1}(\zeta, m_3^2, m_4^2, l)}{l^2 - \zeta}, \qquad (1.135)$$

  wobei $\lambda(\zeta, m_3^2, m_4^2) = (\zeta - m_3^2 - m_4^2)^2 - 4m_3^2 m_4^2$ die Källén-Funktion und f_{α_0,α_1} eine rationale Funktion von ζ, m_3^2 und m_4^2 mit Komponenten von l (l_0 und l_1) im Zähler ist.

- Alle Komponenten von l können gegen Propagatoren gekürzt werden, und es bleiben skalare Einloop-Dreipunkt-Funktionen

$$C(\zeta) = \int d^D l \, \frac{1}{P_1(l+q_1)P_2(l-q_2)(l^2-\zeta)}, \qquad (1.136)$$

  deren eine Masse der Dispersionsparameter ζ ist. Außerdem gibt es einfachere Tensor-Integrale von Einloop-Zwei- und -Einpunkt-Funktionen.

- Die verbleibende Integration über den Dispersionsparameter ζ wird numerisch ausgeführt. Durch eine partielle Integration kann man $C(\zeta)$ durch $dC/d\zeta$ ersetzen und so die Berechnung von Dilogarithmen vermeiden.

- Die durch die Taylor-Entwicklung abgezogenen Terme sind entweder faktorisierende Dreipunkt-Topologien oder Vakuumgraphen.

Gegenüber [75] wird diese Topologie zusätzlich mit ganzzahligen Potenzen der Propagatoren benötigt. Dies verringert die benötigte Ordnung in der Taylor-Entwicklung um 2 je zusätzlicher Potenz. Potenzen von P_3 und P_4 lassen sich berechnen, indem man f_{α_0,α_1} nach m_3^2 bzw. m_4^2 differenziert. Potenzen von P_1 und P_2 führen auf Einloop-Dreipunkt-Funktionen mit Potenzen von Propagatoren. Dies verkompliziert die partielle Integration vor der numerischen Integration, was aber nur ein kleiner Nachteil ist, da der Integrand sowieso je nach Divergenzgrad mit hoher Genauigkeit (≈ 150 Stellen) berechnet werden muß und man so den Geschwindigkeitsvorteil einer fest in die CPU eingebauten Funktion wie $\log(x)$ gegenüber einer benutzerimplementierten wie $\mathrm{Li}_2(x)$, für die eine effiziente Reihenentwicklung existiert, nicht nutzen kann.

Die Erweiterung des Verfahrens auf die Topologien Abb. 1.28b und Abb. 1.28c bereitet keine Schwierigkeiten. An der k-Integration ändert sich nichts, bei der l-Integration hat man einen oder zwei zusätzliche Propagatoren $1/(l^2 - m^2)$, so daß man nach einer Partialbruchzerlegung dieser und des $1/(l^2 - \zeta)$ Terms zusätzlich zu $C(\zeta)$ noch Einloop-Dreipunkt-Funktionen, die von ζ unabhängig sind, erhält.

In [75] wurde das Romberg-Verfahren [85] zur numerischen Integration über ζ benutzt. Etwas bessere Ergebnisse erhält man mit Gauss-Integrationsverfahren [76] mit einer Gewichtsfunktion, die auf die spezielle Form des Integranden angepaßt ist. Mit der Substitution

$$\zeta(t) = (m_3 + m_4)^2 + \frac{t^{2/3}}{(1-t)^\alpha}, \quad t \in [0, 1[\qquad (1.137)$$

aus [75] wird der Integrand für $\alpha = 1$ am besten durch eine Gewichtsfunktion

$$W(t) = a - b\ln(1-t) \tag{1.138}$$

beschrieben, deren Gewichte und Abszissenwerte sich mit Hilfe der Legendre-Polynome berechnen lassen [76].

1.6 Anomale Schwellen

In [16, 30] wurde untersucht, wie das Auftreten physikalischer Schwellen (und Pseudo-Schwellen) bei der Berechnung der planaren und der gekreuzten Dreipunkt-Funktion mit der Parallel-/Orthogonalraum-Methode mit Bedingungen an die Koeffizienten der Vierfach-Integraldarstellung Gl. (1.12) (von denen noch zwei analytisch ausgeführt werden können) in Verbindung gebracht werden kann.

Im wesentlichen haben die einzelnen Terme der Integrale die Form

$$\iint dl_0 dk_0 \int_0^\infty \frac{dt}{t + t_0 - i\eta} \int_0^\infty \frac{ds}{s + s_0 - i\eta} \frac{1}{\sqrt{(at + b + i\eta + cs)^2 - 4st}}\,, \tag{1.139}$$

wobei die Koeffizienten a, b, c, t_0 und s_0 rationale Funktionen der Impulse der äußeren und Massen der inneren Teilchen sowie der verbleibenden Integrationsvariablen l_0 und k_0 sind. Physikalische Schwellen sind äquivalent zum Auftreten eines Imaginärteils im Integral. Die Bedingungen dazu sind immer von der Form

$$q_i^2 \geq (m_{i_1} + \cdots + m_{i_n})^2\,, \tag{1.140}$$

d.h. die Impulse werden groß genug, um die virtuellen inneren Teilchen reell werden zu lassen. Man kann also einen Schnitt durch das Diagramm machen und erhält einen erlaubten physikalischen Prozeß.

Bei den Zweiloop-Dreipunkt-Funktionen findet man, daß

- die Existenz eines Vorzeichenwechsels eines t_0 oder s_0 vom Positiven ins Negative für Werte von l_0 und k_0 innerhalb des Integrationsbereichs (der entsprechend Gl. (1.14) einen Beitrag zum Imaginärteil ergibt) einem Zwei-Teilchen-Schnitt entspricht,

- die Existenz eines Vorzeichenwechsel eines b vom Negativen ins Positive (da dann das Argument der Wurzel negativ werden kann) einem Drei-Teilchen-Schnitt entspricht,

- bei der gekreuzten Dreipunkt-Funktion einzelne Koeffizienten scheinbar Beiträge zum Imaginärteil liefern, die sich wegen der Partialbruchzerlegung in der Summe aber wegheben.

Sowohl Zwei- als auch Drei-Teilchen-Schnitte bzw. deren Widerspiegeln im Vorzeichenwechsel der Koeffizienten machen sich in den analytischen s- und t-Integrationen durch eine Reihe von Fallunterscheidungen bemerkbar (allein bei der s-Integration der gekreuzten Dreipunkt-Funktion sind 10 Fälle zu unterscheiden).

1.6.1 Landau-Gleichungen

Physikalische Schwellen erhält man als Lösungen der Landau-Gleichungen [58], die notwendige Bedingungen für das Auftreten von Singularitäten in Feynman-Graph-Amplituden geben. Die Landau-Gleichungen haben aber auch Lösungen, die keiner physikalischen Schwelle entsprechen und man daher anomale Schwellen nennt. Im folgenden werden die Landau-Gleichungen vorgestellt und mit ihnen das Verhalten von Zweiloop-Dreipunkt-Funktionen an anomalen Schwellen untersucht sowie gezeigt, daß die Parallel-/Orthogonalraum-Integraldarstellung anomale Schwellen korrekt behandelt.

Zunächst werden Bedingungen für Singularitäten in Amplituden A zu beliebigen L-Loop-Feynman-Graphen

$$A = \int \prod_{l=1}^{L} d^D k_l \, Z \prod_{i=1}^{n} \frac{1}{r_i^2 - m_i^2} \,. \tag{1.141}$$

gesucht. Die r_i sind dabei Linearkombinationen der Loopimpulse k_l und der äußeren Impulse p_j. Die Zählerstruktur Z, ein Polynom der Impulse, hat keinen Einfluß auf die Singularitäten und kann daher im folgenden ignoriert werden, indem nur $Z = 1$ betrachtet wird. Führt man Feynman-Parameter im Integral ein, wird die Amplitude proportional zu

$$A \sim \int \prod_{l=1}^{L} d^D k_l \int_0^1 \prod_{j=1}^{n} d\alpha_j \, \frac{\delta(1 - \sum \alpha_i)}{(\sum \alpha_i (r_i^2 - m_i^2))^n} \,. \tag{1.142}$$

Der Nenner kann nach einer linearen Transformation der Loopimpulse $k_l \to k'_l$ auf die Form

$$\sum_{i=1}^{n} \alpha_i (r_i^2 - m_i^2) = \phi + K(k'_1, k'_2, \dots) \tag{1.143}$$

gebracht werden, wobei ϕ eine Funktion der Massen und äußeren Impulse p_i und K eine homogene quadratische Funktion der transformierten Loopimpulse k'_l ist. K ist positiv definit für euklidische Impulse. Eine Singularität kann daher dann auftreten, wenn es Werte von α_i gibt, für die das Minimum von ϕ als Funktion der Massen und äußeren Impulse Null werden kann.

Diese Überlegungen führen auf die notwendigen Bedingungen für Singularitäten

$$\alpha_i (r_i^2 - m_i^2) = 0 \quad \text{für alle} \quad i = 1, \dots, n \tag{1.144}$$

zusammen mit

$$\sum_{i=1}^{n} \alpha_i r_i \cdot \frac{\partial r_i}{\partial k_l} = \sum_{i \in \mathcal{L}_l} (\pm) \alpha_i r_i = 0 \,, \quad l = 1, \dots, L \,, \tag{1.145}$$

wobei $\mathcal{L}_l$ die Menge aller Propagatoren ist, die den Loopimpuls k_l enthalten. Diese Bedingungen nennt man Landau-Gleichungen.

Gl. (1.144) kann so interpretiert werden, daß alle Propagatoren bei einer Singularität auf der Massenschale sein müssen ($r_i^2 = m_i^2$) oder der Propagator nicht beachtet wird ($\alpha_i = 0$), d.h. man betrachtet den um den Propagator i gekürzten Graph, dessen Singularitäten eine Untermenge der Singularitäten des betrachteten Graphen bilden.

1.6.2 Einloop-Beispiel

Bei Zweipunkt-Funktionen erhält man aus den Landau-Gleichungen sofort die Bedingungen für physikalische Schwellen

$$p^2 = \left(\sum_i m_i\right)^2 . \tag{1.146}$$

Bei dem Einloop-Dreipunkt-Graphen aus Abb. 1.38 findet man zum einen Lösungen, bei denen eines der $\alpha_i = 0$ ist, effektiv also ein Zweipunkt-Graph betrachtet wird. Dies ergibt die physikalischen Schwellen $p_1^2 = (m_2 + m_3)^2$ und Permutationen. Sucht man nach Lösungen, bei denen alle α_i nicht-verschwindend sind, so fordert man

$$r_i^2 = m_i^2 , \quad i = 1, \ldots, 3 \tag{1.147}$$

aus Gl. (1.144), und Gl. (1.145) lautet

$$\alpha_1 r_1 + \alpha_2 r_2 + \alpha_3 r_3 = 0 . \tag{1.148}$$

Durch Multiplikation mit den r_i führt dies auf ein Gleichungssystem

$$\begin{aligned} \alpha_1 r_1^2 + \alpha_2 r_1 \cdot r_2 + \alpha_3 r_1 \cdot r_3 &= 0 \\ \alpha_1 r_1 \cdot r_2 + \alpha_2 r_2^2 + \alpha_3 r_2 \cdot r_3 &= 0 \\ \alpha_1 r_1 \cdot r_3 + \alpha_2 r_2 \cdot r_3 + \alpha_3 r_3^2 &= 0 \end{aligned} \tag{1.149}$$

mit

$$r_1 \cdot r_2 = \frac{1}{2}\left(r_1^2 + r_2^2 - (r_1 - r_2)^2\right) = \frac{1}{2}(m_1^2 + m_2^2 - p_3^2) \tag{1.150}$$

und Permutationen. Das Gleichungssystem hat nicht-triviale Lösungen, wenn die Determinante verschwindet:

$$1 + 2\mu_1\mu_2\mu_3 - \mu_1^2 - \mu_2^2 - \mu_3^2 = 0 \tag{1.151}$$

mit der Abkürzung

$$\mu_j = \frac{(m_1^2 + m_2^2 + m_3^2 - m_j^2 - p_j^2)\, m_j}{2 m_1 m_2 m_3}, \quad j = 1, 2, 3 . \tag{1.152}$$

Diese Bedingung läßt sich auch im physikalischen Bereich (d.h. reelle Impulse p_1^2, p_2^2 und p_3^2 mit reellen Impulskomponenten im Parallelraum) erfüllen, z.B. in beliebiger Massendimension mit

$$m_1 = m_2 = m_3 = 1 , \tag{1.153}$$

und

$$\begin{aligned} p_1^2 &= 6 \\ p_2^2 &= 5 \\ p_3^2 &= -4 - \sqrt{15} \approx -7.873 . \end{aligned} \tag{1.154}$$

Derartige Lösungen der Landau-Gleichungen, die nicht einer auf einen Zweipunkt-Graphen reduzierten physikalischen Schwelle entsprechen, nennt man anomale Schwellen.

Abb. 1.39 zeigt das Verhalten von Real- und Imaginärteil der Einloop-Dreipunkt-Funktion in der Nähe der anomalen Schwelle, wenn p_3^2 variiert wird. Der Realteil zeigt eine Diskontinuität, während der Imaginärteil eine logarithmische Divergenz hat.

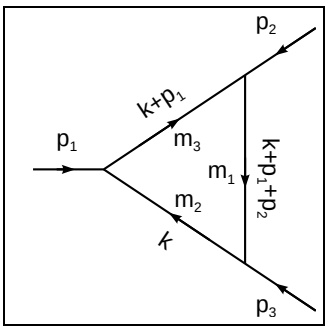


Abbildung 1.38: Impuls- und Massenkonventionen zur Untersuchung der anomalen Schwellen eines Einloop-Graphen

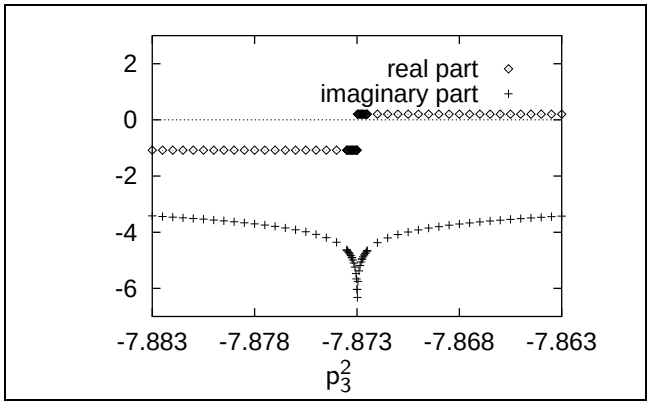


Abbildung 1.39: Real- und Imaginärteil der Einloop-Dreipunkt-Funktion in der Umgebung der anomalen Schwelle

1.6.3 Verhalten von Zweiloop-Funktionen

Das Verhalten der Zweiloop-Dreipunkt-Funktionen an anomalen Schwellen soll am Beispiel des Graphen in Abb. 1.40 gezeigt werden. Wie in [58] gezeigt, haben die zugehörigen Landau-Gleichungen keine Lösungen, bei denen alle sechs α_i nicht-verschwindend sind. Statt einer systematischen Untersuchung aller möglichen Fälle soll hier nur exemplarisch der Fall

$$\alpha_1 = \alpha_2 = \alpha_3 = 0 \tag{1.155}$$

behandelt werden, bei dem man eine anomale Schwelle bei derselben Lösung wie in Gl. (1.151) für den Einloop-Graphen erwartet, wenn man m_1, m_2 und m_3 durch m_6, m_5 und m_4 ersetzt. Für die äußeren Impulse werden in Übereinstimmung mit der bisherigen Zweiloop-Dreipunkt-Konvention q, q_1 und q_2 statt p_1, p_2 und p_3 verwendet.

Mit Gl. (1.2) in Parallel-/Orthogonalraum-Komponenten ausgedrückt sind die quadrierten Impulse $q^2 = (e_1 + e_2)^2$, $q_1^2 = e_1^2 - q_z^2$ und $q_2^2 = e_2^2 - q_z^2$. Gl. (1.151) ist dann eine

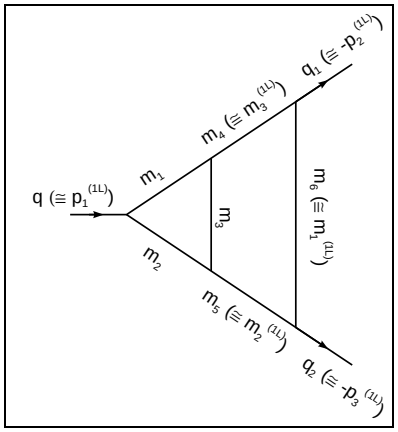


Abbildung 1.40: Impuls- und Massenkonventionen zur Untersuchung der anomalen Schwellen eines Zweiloop-Graphen

biquadratische Funktion in q_z mit der Lösung (q_z wird immer als positiv angenommen):

$$\begin{aligned} q_z^2 \;=\; \frac{1}{2(e_1+e_2)^2}\Big(\; & {m_4}^4 - 2\,{m_5}^2 e_1\,e_2 + {e_1}^4 + {e_2}^4 - 2\,{e_2}^2 {m_6}^2 + 2\,{e_2}^2 {e_1}^2 \\ & + {m_5}^4 + 2\,e_1\,{e_2}^3 + 2\,{e_1}^3 e_2 - 4\,e_1\,e_2\,{m_6}^2 - 2\,e_1\,e_2\,{m_4}^2 \\ & - 2\,{m_5}^2 {e_2}^2 - 2\,{m_5}^2 {m_4}^2 - 2\,{m_4}^2 {e_1}^2 - 2\,{e_1}^2 {m_6}^2 \\ & \pm\; \sqrt{(e_2+e_1-m_4-m_5)\,(e_2+e_1+m_4-m_5)} \\ & \times \sqrt{(e_2+e_1-m_4+m_5)\,(e_2+e_1+m_4+m_5)} \\ & \times \sqrt{(-2\,e_2\,m_6 - 2\,e_1\,m_6 + {e_1}^2 - {m_4}^2 + {m_5}^2 - {e_2}^2)} \\ & \times \sqrt{(2\,e_2\,m_6 + 2\,e_1\,m_6 + {e_1}^2 - {m_4}^2 + {m_5}^2 - {e_2}^2)}\Big) \end{aligned} \tag{1.156}$$

Für eines der vier Dreiecke in der Vierfach-Integraldarstellung der planaren Dreipunkt-Funktion aus Gl. (1.12), über die integriert wird (und das gewählt wurde, weil es gerade bei der Konfiguration Gl. (1.154) die im folgenden beschriebenen Korrelationen erfüllt), lauten die Koeffizienten t_0 und t_0'

$$\begin{aligned} t_0 &= \frac{e_2 - q_z}{e_2 + q_z} k_0^2 + \frac{m_6^2 - m_5^2 + e_2^2 - q_z^2}{e_2 + q_z} k_0 + m_6^2 \\ t_0' &= \frac{e_1 + q_z}{e_1 - q_z} k_0^2 + \frac{m_4^2 - m_6^2 - e_1^2 + q_z^2}{e_1 - q_z} k_0 + m_6^2 \,. \end{aligned} \tag{1.157}$$

t_0 und t_0' sind also quadratisch in k_0. Bezeichnet man die beiden Nullstellen von t_0 und t_0' als Funktion von k_0 mit $k_0^{(1,2)}$ bzw. ${k_0'}^{(1,2)}$ und setzt man dort für q_z den Wert mit der positiven Wurzel aus Gl. (1.156) ein, so findet man, daß auf der anomalen Schwelle zwei Nullstellen im Integrationsbereich zusammenfallen:

$$k_0^{(2)} = {k_0'}^{(1)} \,. \tag{1.158}$$

Für den Beweis wird zunächst $m_4 = m_5 = m_6 = 1$ gesetzt, da sonst die Ausdrücke sehr lang werden. Die Verallgemeinerung auf beliebige Massen folgt unten. Damit vereinfacht

sich Gl. (1.156) unter Berücksichtigung nur der reellen Lösung von q_z zu

$$\begin{aligned} q_z^2 &= \frac{1}{2}\left(e_1^2+e_2^2-4+\sqrt{(e_1+e_2+2)(e_1+e_2-2)(e_1-e_2+2)(e_1-e_2-2)}\right) \\ &= \frac{1}{2}\left(e_1^2+e_2^2-4+\sqrt{D}\right). \end{aligned} \tag{1.159}$$

Weiter ist

$$\begin{aligned} k_0^{(2)} &= -\frac{1}{2}\frac{e_2^2-q_z^2+\sqrt{(e_2^2-q_z^2)(e_2^2-q_z^2-4)}}{e_2-q_z} \\ &= -\frac{1}{2}\frac{e_2^2-q_z^2+\sqrt{Q_2}}{e_2-q_z}, \end{aligned} \tag{1.160}$$

mit $Q_2 = q_2^2(q_2^2-4) > 0$ für die Werte aus Gl. (1.154), und

$$\begin{aligned} k_0'^{(1)} &= \frac{1}{2}\frac{e_1^2-q_z^2+\sqrt{(e_1^2-q_z^2)(e_1^2-q_z^2-4)}}{e_1+q_z} \\ &= \frac{1}{2}\frac{e_1^2-q_z^2+\sqrt{Q_1}}{e_1+q_z}, \end{aligned} \tag{1.161}$$

mit $Q_1 = q_1^2(q_1^2-4) > 0$ (Gl. (1.154)). Die Bedingung $k_0^{(2)} = k_0'^{(1)}$ ist dann gleichbedeutend mit

$$(e_1+e_2)(e_1e_2-q_z^2)+(e_2-e_1+\sqrt{Q_2}-\sqrt{Q_1})q_z+\sqrt{Q_2}e_1+\sqrt{Q_1}e_2 = 0\,. \tag{1.162}$$

Mit einer Substitution für e_1 und e_2 [79]

$$\begin{aligned} x^2 &= \frac{e_1+e_2-2}{e_1+e_2+2} \\ y^2 &= \frac{e_1-e_2-2}{e_1-e_2+2} \end{aligned} \tag{1.163}$$

und der Rücktransformation

$$\begin{aligned} e_1 &= 2\frac{1-x^2y^2}{(1-x)(1+x)(1-y)(1+y)} \\ e_2 &= 2\frac{x^2-y^2}{(1-x)(1+x)(1-y)(1+y)} \end{aligned} \tag{1.164}$$

werden alle Wurzeln rational. Zunächst ist

$$D = \left(\frac{16xy}{(1-x)(1+x)(1-y)(1+y)}\right)^2, \tag{1.165}$$

wobei für die betrachteten Werte der Impulse aus Gl. (1.154)

$$\begin{aligned} 0 < x < 1 \\ 0 < y < x \end{aligned} \tag{1.166}$$

gilt, und man erhält

$$q_z^2 = \left(\frac{2(x+y)(1-xy)}{(1-x)(1+x)(1-y)(1+y)}\right)^2. \tag{1.167}$$

Weiterhin sind

$$Q_1 = \left(\frac{4(1-xy)(x-y)}{(1-x)(1+x)(1-y)(1+y)} \right)^2 \tag{1.168}$$

und

$$Q_2 = \left(\frac{4(1+xy)(x+y)}{(1-x)(1+x)(1-y)(1+y)} \right)^2 . \tag{1.169}$$

Damit sind alle Terme in Gl. (1.162) rational, und man prüft leicht nach, daß die linke Seite Null ist.

Für den Fall verschiedener Massen benutzt man statt Gl. (1.163) die Substitution

$$\begin{aligned} x^2 &= \frac{e_1+e_2-m_5-m_4}{e_1+e_2+m_5+m_4} \\ y^2 &= \frac{(e_1^2-e_2^2-2(e_1+e_2)m_6+m_5^2-m_4^2)(e_1+e_2+m_5-m_4)}{(e_1^2-e_2^2+2(e_1+e_2)m_6+m_5^2-m_4^2)(e_1+e_2-m_5+m_4)} , \end{aligned} \tag{1.170}$$

die für $m_4 = m_5 = m_6 = 1$ mit Gl. (1.163) identisch ist und mit der D aus Gl. (1.165) ebenfalls zu einem Quadrat wird. q_z^2, Q_1 und Q_2 sind jeweils Quadrate mit einem zusätzlichen Faktor $W^2 = (x^2 m_5 + m_4)(x^2 m_4 + m_5) > 0$. Das Äquivalent zu Gl. (1.162) ist dann eine rationale Funktion mit einer Wurzel $R(x, y, m_4, m_5, m_6; W)$, deren Zähler eine Linearkombination $A(x, y, m_4, m_5, m_6) + B(x, y, m_4, m_5, m_6)\, W$ ist. Durch Einsetzen findet man $A = 0$ und $B = 0$.

Alternativ dazu kann man die Größen

$$\begin{aligned} e_+ &= e_1 + e_2 \\ e_- &= e_1 - e_2 \\ m_+ &= m_5 + m_4 \\ m_- &= m_5 - m_4 \end{aligned} \tag{1.171}$$

einführen und die Substitution für m_+, m_- und e_-

$$\begin{aligned} x^2 &= \frac{e_+ - m_+}{e_+ + m_+} \\ y^2 &= \frac{e_+ e_- - 2e_+ m_6 + m_+ m_-}{e_+ e_- + 2e_+ m_6 + m_+ m_-} \\ z^2 &= \frac{e_+ - m_-}{e_+ + m_-} \end{aligned} \tag{1.172}$$

benutzen, die *alle* auftretenden Wurzeln rational macht.

Genau auf der anomalen Schwelle gibt es also einen Wert von k_0, bei dem sowohl t_0 als auch t_0' gleichzeitig Null werden. Da Gl. (1.12) aber vor dem weiteren Ausführen der analytischen s- und t-Integrationen in t partialbruchzerlegt wird, bedeutet dies für die einzelnen Terme, die von der Form

$$\tilde{C} \int_0^\infty \frac{dt}{t + t_0 - i\eta} \int_0^\infty \frac{ds}{s + s_0 - i\eta} \frac{1}{\sqrt{(at+b+cs)^2 - 4st}} \tag{1.173}$$

sind, keine weitere Fallunterscheidung. Der Vorfaktor $\tilde{C}$, der proportional zu $\frac{1}{t_0-t_0'}$ ist, wird jedoch in der Nähe der anomalen Schwelle für Werte von k_0 um die Nullstelle groß, und man zieht große Terme voneinander ab. Es handelt sich um eine logarithmische, integrable Singularität.

Abb. 1.41 zeigt daher auch das erwartete Verhalten der planaren Zweiloop-Dreipunkt-Funktion, analog zu dem der Einloop-Dreipunkt-Funktion, in der Umgebung der anomalen Schwelle mit

$$\begin{aligned} q^2 &= 6 \\ q_1^2 &= 5 \\ q_2^2 &= -4-\sqrt{15} \approx -7.873 \\ m_i &= 1\,, \quad i=1,\ldots,6\,. \end{aligned} \tag{1.174}$$

Lediglich die numerische Stabilität wird in unmittelbarer Nähe der Schwelle schlechter.

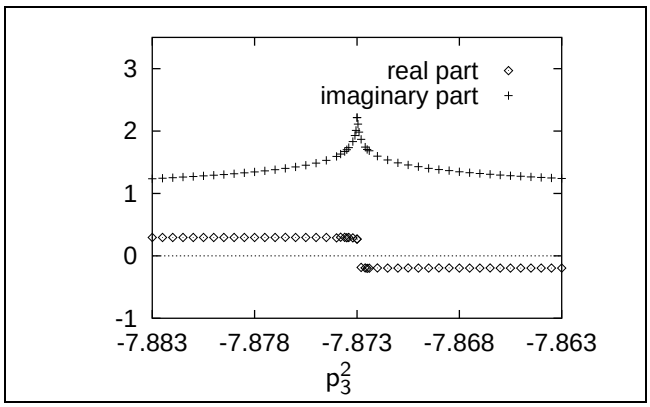


Abbildung 1.41: Real- und Imaginärteil der Zweiloop-Dreipunkt-Funktion in der Umgebung der anomalen Schwelle

Kapitel 2

GiNaC — eine Bibliothek für Computer-Algebra in C++

Dieses Kapitel beschreibt die Entwicklung der Programmbibliothek GiNaC [2] (ein rekursives Akronym für *GiNaC is not a CAS*), die es erlaubt, in C++-Programmen symbolische Berechnungen auszuführen. Diese Bibliothek bildet die Grundlage für die neue Generation des Programmpaketes *xloops* zur automatischen Berechnung von Feynman-Graphen (siehe Kapitel 3). Sie ist allgemein genug, um in anderen Bereichen, auch außerhalb der Physik, einsetzbar zu sein. Dieses Kapitel setzt einige Grundkenntnisse in der Programmiersprache C++ voraus. Eine Einführung in C++ gibt [7, 86].

2.1 Motivation

Die bisherige Version von *xloops* [9] beruhte auf Programmteilen, die Anfang der 90er Jahre für ein Paket zum Berechnen von Einloop-Feynman-Graphen entwickelt wurden und später um Zweiloop-Zweipunkt-Graphen erweitert wurden. Eine Weiterentwicklung der bestehenden Teile um Dreipunkt-Funktionen, also im wesentlichen die Umsetzung der Ergebnisse aus [16, 30] und Kapitel 1 zur automatisierten Berechnung von Feynman-Graphen, erschien nicht mehr sinnvoll (die Gründe werden detaillierter in Abschnitt 3.3 aufgeführt), so daß eine komplette Neuprogrammierung beschlossen wurde. Dadurch bekam man auch die Freiheit, die Programmierumgebung Maple [11, 12], in der *xloops* geschrieben war, zur Disposition zu stellen. Gegen Maple konnten einige Gründe aufgeführt werden:

- Neue Programmversionen von Maple (Releases) führen regelmäßig Inkompatibilitäten ein (beispielsweise in der Behandlung lokaler und globaler Variablen), mit der Folge, daß Programme für die alte Release nur mit großen Änderungen zum Laufen zu bringen sind. Dadurch entstehen zum Teil subtile Fehler, die sich nur schwer entdecken und beheben lassen.

  Die Anpassung von *xloops* auf die 1996 erschienene Maple Release 4 ist gerade erst abgeschlossen. Anfang 2000 erscheint bereits Release 6.

- Eigene Fehler in Maple-Programmen lassen sich nur schwer finden, da der eingebaute (erst seit Release 4 überhaupt vorhandene) Debugger sehr rudimentär ist.

- Aufgrund der internen Arbeitsweise von Maple kann ein und derselbe Funktionsaufruf

abhängig von vorherigen Berechnungen verschiedene Ergebnisse liefern, beispielsweise bei der Entwicklung von Laurent-Reihen.

- Programme lassen sich nur unzureichend in abgeschlossene, voneinander unabhängige Module aufteilen. Dies ist aber notwendig, da *χloops* mittlerweile einen Umfang von 600kB erreicht hat.

- Die einzige Datenstruktur in Maple ist die Liste. Beispielsweise muß die Information über alle Topologien in einer sechsfach geschachtelten Liste gespeichert werden. Zugriffe auf diese Listen in der Form `op(i,op(j,op(k,L)))` machen Programme schnell unlesbar.

- Die fehlenden Datenstrukturen verleiten dazu, statt Funktionsparametern globale Variablen zu benutzen und somit versteckte Abhängigkeiten zu erzeugen.

- Es gibt keine effiziente numerische Integration, z.B. mit Hilfe von Monte-Carlo-Methoden. Dafür ist es notwendig, ein eigenes C-Programm aufzurufen. Maple bietet aber keine Kommunikationsmöglichkeiten mit anderen Programmen außer über den (betriebssystemabhängigen) Befehl `system()`.

- Maple ist als kommerzielles Programm nicht überall verfügbar. Dies stand u.a. einer weiten Verbreitung von *χloops* entgegen.

Andere Computer-Algebra-Systeme (CAS) haben dieselben und weitere Nachteile:

- In Mathematica [92] ist die Wahrscheinlichkeit, daß eigene Programme unter einer neuen Release fehlerfrei laufen, höher als bei Maple. Man wird jedoch durch die befristete Vergabe von Lizenzen zu einem Update gezwungen und kann bei Problemen die alte Version nicht weiternutzen. Außerdem ist selbst die Programmiersprache listenorientiert (z.B. `For[start, test, incr, body]`), wodurch die Lesbarkeit der Programme gestört wird. Positiv ist anzumerken, daß mit dem Zusatzprogramm MathLink eine Kommunikation mit anderen Programmen möglich ist.

- Reduce [41] läuft nicht stabil bei großen Ausdrücken. Die Benutzeroberfläche ist sehr rudimentär.

- MuPAD [69] hebt sich, seit es nicht mehr frei verfügbar ist, nicht deutlich von den übrigen CAS ab.

- Über andere CAS lagen keine Erfahrungen vor, da sie nicht auf den Rechnern der Arbeitsgruppe installiert waren.

Demgegenüber stehen die Vorteile, die eine strukturierte Programmiersprache wie C++ bietet:

- Programmiersprache: Die standardisierte Syntax der Programmiersprache ist nicht nur einem kleinen Kreis der CAS-Anwender bekannt. Der Sprache fehlen auch keine wichtigen Elemente der strukturierten Programmierung wie `do/while`.

- Strukturierte Datentypen: Es gibt strukturierte Datentypen (Klassen), auf deren Elemente man mit Namen zugreifen kann, statt unbenannter n-fach geschachtelter Listen.

- Typsicherheit: Daten, die an Funktionen übergeben werden, werden schon zur Kompilierzeit auf ihren Typ überprüft, fehlerhafte Konstrukte werden abgewiesen.
- Entwicklungstools: Es gibt eine Reihe mächtiger Entwicklungstools für C++, wie Editoren, Debugger, Tools für Laufzeitanalyse (Profiling), graphische Darstellung der Klassen-Hierarchien, automatische Erstellung der Dokumentation aus dem Quellcode etc.
- Modularisierung: Programme können durch die Trennung in Schnittstelle und Implementation leichter modularisiert werden. Die Verwendung sogenannter Namespaces kann verhindern, daß Variablennamen, die in verschiedenen Modulen mehrfach verwendet werden, Konflikte verursachen.
- Effizienz: Kompilierte Programme laufen schneller als interpretierte (Maple und Mathematica sind interpretierte Sprachen). Dies ist vor allem in Prozeduren erkennbar, die einen geringen symbolischen Anteil haben.
- Integration: In eigenen Programmen ist es möglich, nahtlos andere Bibliotheken einzubinden, z.B. für numerische Berechnungen, graphische Ausgaben, Benutzeroberflächen.
- Erweiterbarkeit: Ein objektorientierter Entwurf erlaubt es, dem Programm neue Funktionalität zu geben, die zum Entwicklungszeitpunkt nicht vorhergesehen war, ohne daß an der bisherigen Bibliothek Änderungen vorgenommen werden müssen.
- Preis: Ein C++-Compiler ist auf vielen Systemen bereits verfügbar oder kann kostenlos installiert werden. Außerdem besteht die Möglichkeit, Programme in Binärform zu verteilen, die keine weiteren installierten Programme voraussetzen.

Die Berechnung von Feynman-Graphen erfordert zwingend, zumindest auf Einloop-Niveau, wo sich alle Ergebnisse analytisch ausdrücken lassen, symbolische Ausdrücke zu manipulieren. Weder C++ noch eine andere traditionelle Programmiersprache bieten aber standardmäßig diese Möglichkeit. Eine Analyse der *xloops*-Programme zeigte jedoch, daß nur Teile der Fähigkeiten von Maple benutzt wurden, so vor allem

- Zusammensetzen von Ausdrücken aus Symbolen und elementaren Funktionen ($\sin(x)$ etc.) mit den Grundrechenarten $+$, $-$, $\times$, $\div$, Potenzen und einem nicht-kommutativen Produkt,
- Handhabung beliebig großer ganzer Zahlen (z.B. in Koeffizienten von Rekursionsrelationen),
- numerische Evaluierung mit beliebiger Genauigkeit (bei der Berechnung von Feynman-Graphen werden häufig große Terme voneinander abgezogen, so daß man intern mit einer Genauigkeit in der Größenordnung von 100 Stellen rechnen muß, um ein Ergebnis auf 10 Stellen genau zu erhalten),
- Zusammenfassen gleicher Terme ($2x + 3x \to 5x$),
- Vereinfachung gebrochen-rationaler Funktionen ($\frac{x^2-1}{x-1} \to x + 1$),
- symbolische Ableitungen (z.B. nach Massen, um Feynman-Diagramme mit Nennerpotenzen > 1 zu berechnen),

- Laurent-Entwicklung von Funktionen ($\frac{1}{\epsilon^n}$-Terme in dimensionaler Regularisierung),
- Definition eigener Funktionen mit Ableitung und Laurent-Entwicklung (z.B. $\mathrm{Li}_2(x)$),
- Lösen von linearen Gleichungssystemen.

Insbesondere werden *nicht* gebraucht

- symbolische Integration,
- Faktorisierung von Ausdrücken,
- Grenzwertberechnung,
- Lösen von nicht-linearen Gleichungssystemen,
- erweiterte Vereinfachungen (z.B. $\sin(2x) - 2\sin(x)\cos(x) \to 0$),
- der größte Teil der Bibliotheksfunktionen.

  Viele Funktionen wurden in *xloops* ohnehin neu geschrieben, da die Bibliotheksfunktionen nur eingeschränkt benutzbar oder fehlerhaft waren oder erst in neueren Releases hinzugekommen sind, z.B. Dilogarithmus (keine komplexen Argumente), Logarithmus (falscher Schnitt bei $\log(x - iy)$ falls $x < 0$ und $y \ll x$), Lorentztensoren.

Die Umsetzung dieser Anforderungen erschien machbar. Die Wahl der Implementierungssprache fiel auf C++, da es C++ erlaubt, Operatoren wie die Grundrechenarten $+$, $-$, $\times$, $\div$ auf eigene Klassen wirken zu lassen. Dadurch wird es möglich, einen Ausdruck wie $3x + 5y$ in lesbarer Form als `3*x+5*y` einzugeben. Umständlichere Schreibweisen wie `x.times(3).plus(y.times(5))` (Java-Stil) oder `plus(times(x,3),times(y,5))` (C- oder Pascal-Stil) können damit vermieden werden.

Die Idee, eine primär für die Verarbeitung von Zahlen in Maschinengenauigkeit und Zeichenketten gedachte Programmiersprache um Fähigkeiten zur Manipulation symbolischer Ausdrücke zu erweitern, ist bereits Anfang der 60er Jahre mit dem Paket Formula ALGOL [74] verwirklicht worden. Ende der 60er Jahre erschien das Paket ALTRAN [39], das mittels eines Präprozessors in einer Teilmenge von FORTRAN einen symbolischen Datentyp zur Verfügung stellt, der jedoch auf rationale Funktionen beschränkt ist. Ein symbolischer Datentyp für C++ wird in [82] eingeführt. Die Implementation dort eignet sich jedoch für nicht viel mehr als zu Demonstrationszwecken.

2.2 Übersicht über GiNaC

GiNaC verfolgt einen objektorientieren Ansatz, da sich gerade damit mathematische Ausdrücke und Operationen darauf sehr gut beschreiben lassen. So kann man z.B. algebraische Ausdrücke aus Objekten für Summen, Produkte und Potenzen von ebendiesen Objekten sowie Symbolen (Variablen) und Zahlen aufbauen. Eine Operation wie z.B. eine Ableitung muß dann nur für das jeweilige Objekt ohne Rücksicht auf andere Objekte definiert werden: Produkte wenden die Produktregel zum Differenzieren an, die Ableitung einer Zahl ist Null, die eines Symbols 1 oder 0 etc. Ähnliches gilt z.B. für das Expandieren von Ausdrücken, bei dem sich Produkte anders verhalten als Potenzen.

Viele Probleme in der Physik können aber besser durch prozedurale statt objektorientierte Lösungsansätze beschrieben werden. GiNaC bietet daher durch Kapselung aller

wichtigen Methoden auch die Möglichkeit, ohne Kenntnisse der objektorientierten Erweiterungen von C++ im Vergleich zu C, in prozeduralen Programmen benutzt zu werden, ähnlich wie das mit dem Standard-Datentyp `complex` möglich ist.

Sowohl das Verhalten als auch die Namensgebung grundlegender Funktionen (`eval()`, `evalf()`, `subs()`, `nops()`, `op()` etc.) wurde stark von Maple beeinflußt. Dadurch fällt es leichter, Maple-Programme nach GiNaC zu portieren. In gewissem Maße läßt sich dies sogar automatisieren (siehe Abschnitt 2.5.1), wenngleich die konvertierten Programme dann nicht von den Vorteilen der Strukturierung in C++ profitieren können.

2.2.1 Klassenhierarchie

Alle Ausdrücke werden in GiNaC aus Objekten zusammengesetzt, die von einer Basisklasse `basic` abgeleitet sind (Abb. 2.1). Diese Klasse enthält bereits alle Methoden, die auf eine größere Anzahl von Objekten sinnvoll anwendbar sind. Ein häufiger Fall ist es, daß Objekte wieder andere Ausdrücke als Unterobjekte enthalten, entweder eine feste (Potenz, Relation) oder eine variable Zahl (Summe, Produkt, Liste). Für die Verwaltung solcher Kollektionen von Objekten bietet C++ bereits im Sprachstandard die Standard Template Library (STL) [44], die doppelt verkettete Listen (sequentieller Zugriff), Vektoren (wahlfreier Zugriff über einen Index), Maps (Zugriff über einen beliebigen Schlüssel, z.B. Zeichenkette) u.a. zur Verfügung stellt.

Die STL kann jedoch keine Kollektionen von sogenannten polymorphen Objekten, d.h. einer Hierarchie von Objekten, die von einem Basisobjekt abgeleitet sind, bilden. Sie erfordert wegen ihrer internen Speicherverwaltung Objekte konstanter Größe. Kollektionen von Zeigern auf polymorphe Objekte würden zwar die Forderung nach konstanter Größe erfüllen, jedoch kann die STL den Speicher, den die Objekte in der Kollektion belegen, nicht wieder freigeben. Eine Lösung wäre die Verwendung des Templates `auto_ptr`, ebenfalls Teil der STL.

GiNaC benutzt ein ähnliches Konzept wie `auto_ptr`, das diesem jedoch in Bezug auf Speichereffizienz überlegen ist (siehe Abschnitt 2.2.2) und außerdem noch die korrekte Evaluierung der Ausdrücke übernimmt. Zunächst genügt es jedoch zu wissen, daß die Klasse `ex` im wesentlichen aus einem Zeiger auf ein `basic`- (oder davon abgeleitetes) Objekt besteht, sich um die Freigabe belegten Speichers kümmert und die STL-Forderung nach konstanter Objektgröße erfüllt.

GiNaC unterscheidet sich von anderen CAS wie Maple, Mathematica oder Reduce dadurch, daß es sich bei Variablen, denen Ausdrücke zugewiesen werden können, und unveränderlichen terminalen[1] Symbolen um verschiedene Objekte handelt. Eine Variable vom Typ `ex` hat *immer* einen Wert, entweder den beim Erzeugen angegebenen, einen später zugewiesenen oder den Wert Null. Ein Symbol (ein Objekt vom Typ `symbol`, Abschnitt 2.3.2) hingegen hat als Wert immer nur sich selbst. Um einem Symbol c den Wert $3 \cdot 10^8$ „zuzuweisen“, muß man die Methode `subs()` verwenden. Dies verhindert Probleme mit inkonsistent evaluierten Ausdrücken, wie dies gelegentlich in Maple auftritt (siehe Abschnitt 2.2.3).

Die wichtigsten, in sich abgeschlossenen Konzepte, nämlich die Speicherverwaltung, die Evaluierung von Ausdrücken und die Äquivalenz- und Ordnungsrelation, werden in den folgenden Abschnitten erläutert. Daran schließt sich eine Übersicht über die auf Objekten

[1] Terminale Ausdrücke wie Symbole, Konstanten (Abschnitt 2.3.2), Zahlen (Abschnitt 2.3.1) und Indizes (Abschnitt 2.4.2) haben keine weiteren Unterausdrücke.

vom Typ `basic` definierten Methoden sowie eine Diskussion der einzelnen Klassen der GiNaC-Klassenhierarchie aus Abb. 2.1 an.

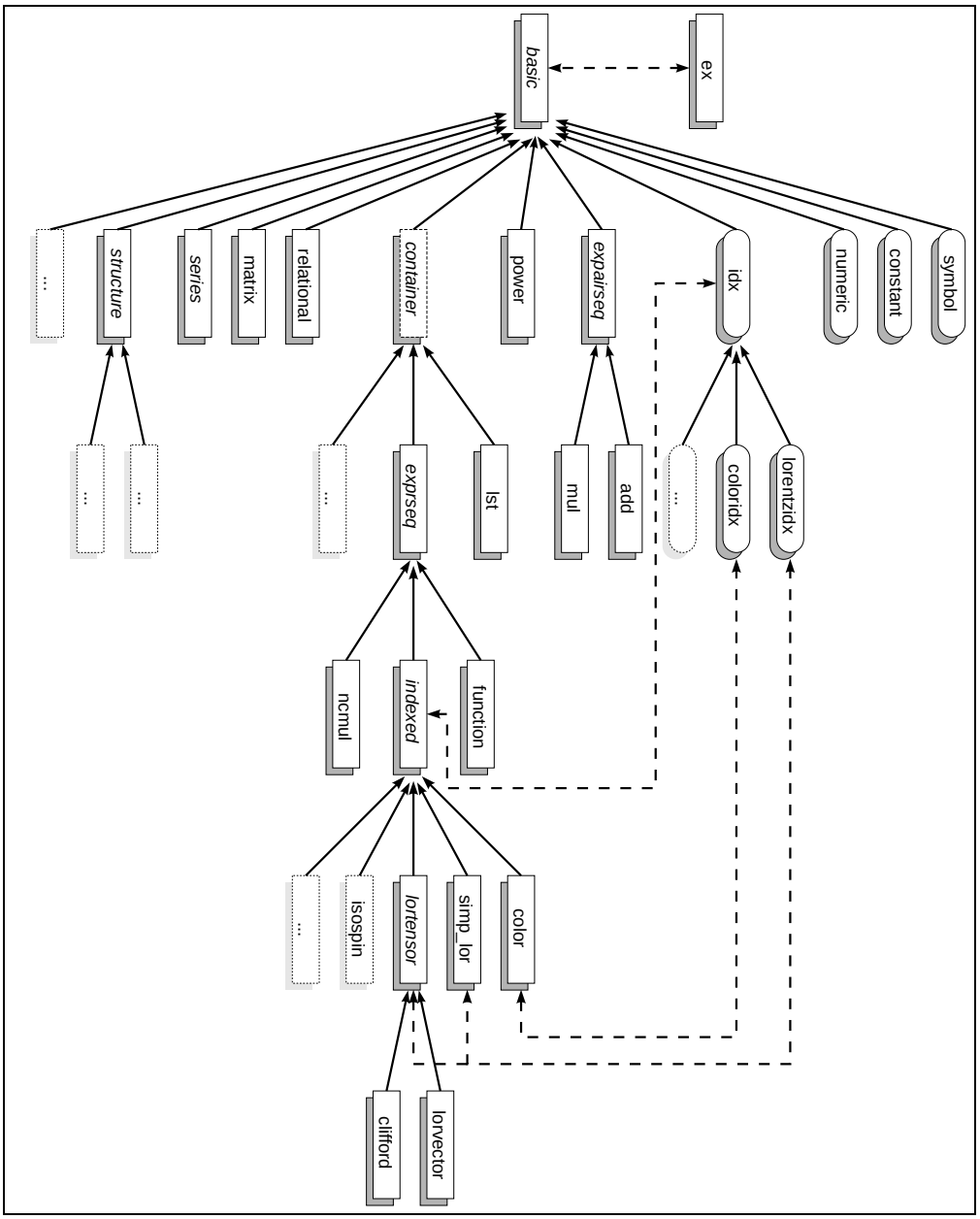


Abbildung 2.1: Die GiNaC-Objekthierarchie: Klassen in ovalen Kästen sind terminale Objekte, Klassen in rechteckigen Kästen können Unterausdrücke enthalten. Mit kursiv geschriebenen Klassen kommt der Endanwender normalerweise nicht in Berührung, da es sich um interne Klassen oder abstrakte Basisklassen handelt. Hellgraue Kästen zeigen die Stellen, an denen GiNaC erweitert werden kann. Eine durchgezogene Linie steht für Vererbung, eine gestrichelte für eine sonstige Beziehung zwischen den Klassen. Bei `container` handelt es sich nicht um eine Klasse im eigentlichen Sinn, sondern um ein Perl-Script, das die von ihm abgeleiteten Klassen erzeugt.

2.2.2 Speicherverwaltung

GiNaC benutzt eine Speicherverwaltungstechnik, die als *Reference Counting* mit *Copy On Write* bezeichnet wird [13]. Das bedeutet, daß sich identische Objekte denselben Speicherplatz teilen können, indem sie mitzählen, wie oft sie verwendet werden. Soll eines von ihnen verändert werden, wird automatisch eine Kopie des zu verändernden Objektes angelegt, während sich die übrigen weiterhin den Speicherplatz teilen. Dies geschieht transparent für den Benutzer. Die Objekte können beliebig zusammengesetzte Ausdrücke sein. Berechnet man z.B. von

$$f(x,y) = \sin^2((x+y)^3) \tag{2.1}$$

die Ableitung

$$\frac{\partial}{\partial x} f(x,y) = 6(x+y)^2 \sin((x+y)^3)\cos((x+y)^3)\,, \tag{2.2}$$

so belegt der viermal auftretende Ausdruck $x+y$ (inklusive in der Ausgangsfunktion Gl. (2.1)) nur einmal Speicher. Dadurch wird der vorhandene Speicher sehr effizient ausgenutzt. Die Speicherbelegung für dieses Beispiel ist in Abb. 2.2 graphisch dargestellt. Die Objekte sind vereinfacht dargestellt und zeigen nicht alle enthaltene Information. Man beachte, daß der Exponent 2 des Sinus in der Ausgangsfunktion Gl. (2.1) nicht den Speicher mit dem Exponenten von $(x+y)^2$ teilt, da sie unabhängig voneinander erzeugt werden.

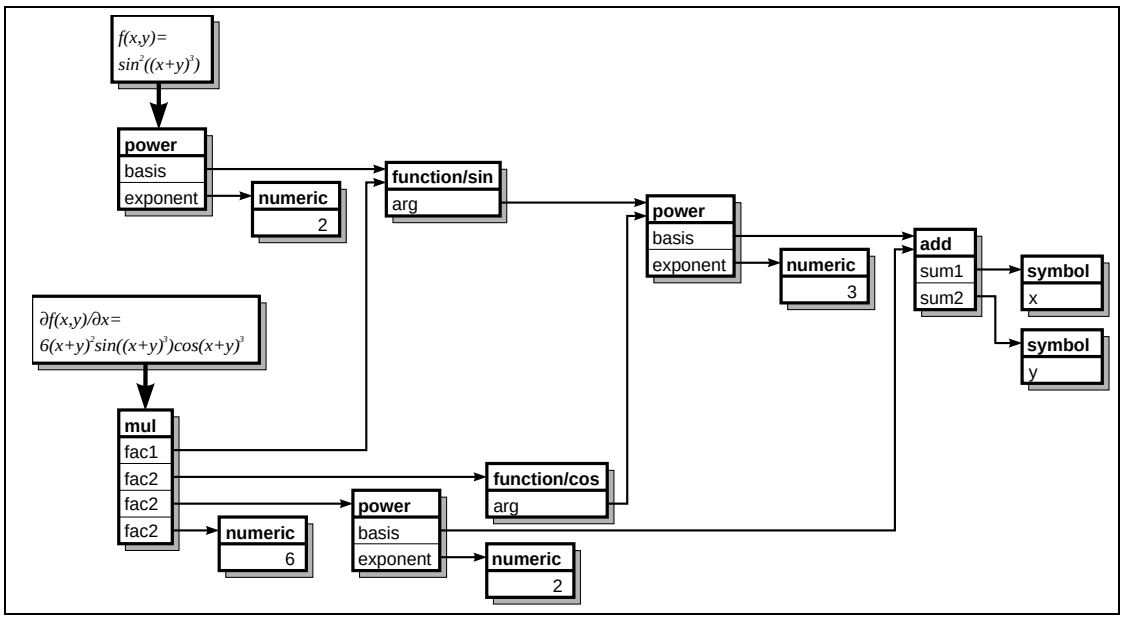


Abbildung 2.2: Speicherbelegung mit *Reference Counting* am Beispiel der Ausdrücke $f(x,y) = \sin^2((x+y)^3)$ und $\frac{\partial}{\partial x} f(x,y) = 6(x+y)^2 \sin((x+y)^3)\cos((x+y)^3)$

2.2.3 Evaluierung

GiNaC versucht, Ausdrücke so früh wie möglich automatisch zu vereinfachen, sofern dies mit vertretbarem Rechenaufwand möglich ist, in der Regel, indem der Ausdruck in eine

„kanonische“ Form umgewandelt wird (z.B. durch Sortieren der Argumente). Als vertretbar gilt ein Aufwand, der nicht viel schneller steigt als die Größe des zu vereinfachenden Ausdrucks, also z.B. $\mathcal{O}(n \log n)$. Insbesondere sollen alle Vereinfachungen, die konstante Zeit benötigen, ausgeführt werden. Beispiele aus verschiedenen Klassen sind:

- $x+y+z+0 \to x+y+z$
- $x^3y^4zx^2 \to x^5y^4z$
- $x^1 \to x$
- $g^{\nu\mu}g_{\mu\nu} \to g^{\mu\nu}g_{\mu\nu}$ (Ausnutzung der Symmetrie der Indizes)
- $f_{aac} \to 0$ (Strukturkonstanten der SU(3))
- $\sin(\pi) \to 0$

Nicht automatisch ausgeführt werden sollten Vereinfachungen, die $\mathcal{O}(n^2)$ oder mehr Zeit brauchen, z.B. Kontrahieren von Lorentzindizes.

Einige Vereinfachungen werden schon beim Konstruieren des Objektes ausgeführt. Da der Ergebnistyp einer Vereinfachung aber nicht mehr der des gerade konstruierten Objektes sein muß (z.B. bei der Vereinfachung von einem `power`-Objekt x^1 zu einem `symbol` x), sondern einen beliebigen Typ annehmen kann, werden diese Vereinfachungen beim Konstruieren eines `ex`-Objektes aus einem von `basic` abgeleiteten Typ gemacht. Dazu wird die virtuelle Methode `eval()` des jeweiligen Objektes aufgerufen, die ihrerseits davon ausgehen kann, daß eventuell vorhandene Unterobjekte vom Typ `ex` bereits evaluiert sind, da `ex`-Objekte durch diese Konstruktion immer evaluiert sein sollten. In den meisten Fällen ist es daher auch nicht notwendig, eine explizite Evaluierung mit der Methode `eval()` durchzuführen. Eine Ausnahme gibt es bei Funktionen, deren Ergebnis auf Seiteneffekten beruht, beispielsweise dem Wert globaler Variablen, auf die innerhalb der Funktion zugegriffen wird.

2.2.4 Äquivalenz- und Ordnungsrelation

Ein weiteres wichtiges Konzept in GiNaC ist die Äquivalenz- und Ordnungsrelation auf Ausdrücken. Um Ausdrücke miteinander vergleichen zu können (z.B. um gleiche Terme zusammenfassen zu können, siehe Abschnitt 2.3.3), muß nämlich eine Äquivalenzrelation $\simeq$ definiert werden, die die Axiome

$$\begin{aligned} \forall e \in \mathcal{E} : e \simeq e & & & \text{(Reflexivität)}, \\ e_1 \simeq e_2 & \Rightarrow & e_2 \simeq e_1 & \text{(Symmetrie) und} \\ (e_1 \simeq e_2) \wedge (e_2 \simeq e_3) & \Rightarrow & e_1 \simeq e_3 & \text{(Transitivität)} \end{aligned} \tag{2.3}$$

erfüllt. Dabei ist $\mathcal{E}$ die Menge aller in GiNaC erzeugbaren Ausdrücke. Diese Äquivalenzrelation muß und kann nicht mathematischer Gleichheit zweier Ausdrücke entsprechen, denn für einen effizienten Vergleich müßten beide Seiten zunächst auf eine kanonische Form gebracht werden. Man kann aber zeigen, daß man nicht für alle Klassen von Ausdrücken eine kanonische Form finden kann [34]. Es wird daher nicht gefordert, daß z.B. $(a+b)(a-b) \simeq a^2-b^2$, andererseits ist es zwingend erforderlich, daß Objekte, die denselben Speicherplatz teilen, die Äquivalenzrelation erfüllen. Wünschenswert ist aber auch, daß

triviale Zusammenhänge wie $ab \simeq ba$ erkannt werden, um Ausdrücke wirksam vereinfachen zu können.

Im Vorgriff auf Abschnitt 2.3.3 ist es für diese Forderung notwendig, auch eine vollständige Ordnungsrelation $\preceq$ auf den Objekten zu definieren, also

$$\begin{array}{rcl} \forall e \in \mathcal{E} : e \preceq e\,, & & \\ (e_1 \preceq e_2) \wedge (e_2 \preceq e_1) & \Rightarrow & e_1 \simeq e_2\,, \\ (e_1 \preceq e_2) \wedge (e_2 \preceq e_3) & \Rightarrow & e_1 \preceq e_3\,. \end{array} \tag{2.4}$$

Die konkrete Realisierung der Ordnungsrelation ist beliebig, solange diese Axiome erfüllt sind. Insbesondere muß keine Beziehung zwischen $\preceq$ und der Ordnungsrelation auf reellen Zahlen bestehen, so daß z.B. $3 \preceq 2$ gelten kann.

2.2.4.1 Grundlagen einer Hashfunktion

Da das Vergleichen von Ausdrücken mit $\simeq$ und $\preceq$ eine häufig benutzte Funktion ist (auch ohne daß der Anwender sie explizit aufruft), sollte sie möglichst effizient implementiert sein. Andererseits sollte aber der Anwender, der neue Klassen zu GiNaC hinzufügen will, sich nicht mehr als notwendig mit den Details befassen müssen.

In GiNaC wird dies mit Hilfe einer Hashfunktion [47, 65] implementiert. Eine Hashfunktion ist eine Abbildung der Ausdrücke auf eine endliche Indexmenge, auf der eine vollständige Ordnungsrelation definiert ist

$$\begin{array}{rcl} h : \mathcal{E} & \rightarrow & I \\ e & \mapsto & h(e)\,, \end{array} \tag{2.5}$$

mit der Eigenschaft

$$e_1 \simeq e_2 \;\Rightarrow\; h(e_1) = h(e_2)\,. \tag{2.6}$$

In der Praxis wählt man meistens $I = \{0, \ldots, N-1\}$ mit dem Vergleich natürlicher Zahlen als Ordnungsrelation und $N = 2^b$, wo b die Breite eines Maschinenwortes ist.

Da es unendlich viele Ausdrücke in $\mathcal{E}$ gibt, kann $h(e)$ nicht injektiv sein. Ein erlaubtes Beispiel ist daher die triviale Hashfunktion

$$e \;\mapsto\; h_t(e) = 0\,. \tag{2.7}$$

h_t ist aber keine *gute* Hashfunktion. An gute Hashfunktionen stellt man die Forderung, daß eine Hashkollision

$$(e_1 \not\simeq e_2) \wedge (h(e_1) = h(e_2)) \tag{2.8}$$

selten auftritt. Dies kann erreicht werden, indem man N groß wählt (z.B. $N = 2^{32}$) und h so konstruiert, daß alle Werte in I quasi-zufällig und gleichverteilt angenommen werden. Die Wahl der Hashfunktion ist immer von den Daten, von denen ein Hashwert berechnet werden soll, abhängig. Gute Hashfunktionen für einen Datentyp können schlechte Hashwerte für andere Typen liefern.

2.2.4.2 TypeIDs

Bevor die in GiNaC benutzte Hashfunktion in Abschnitt 2.2.4.3 vorgestellt wird, muß noch das Konzept der TypeIDs in GiNaC eingeführt werden. C++ bietet als im Standard [44] definiertes Sprachelement Typidentifikation zur Laufzeit mit Hilfe der Funktion `typeid()`. Sie gibt `type_info`-Objekte zurück, auf denen eine vollständige Ordnungrelation definiert ist. Da der Datentyp aber opak ist, kann er nicht als Parameter in der Hashwertberechnung verwendet werden. Beim GNU C Compiler ist der Typ `type_info` als Zeichenkette mit lexikographischer Ordnung repräsentiert. Vergleiche von Zeichenketten sind aber relativ zeitintensiv. Daher implementiert GiNaC eine eigene Typidentifikation über eine Variable `unsigned tinfo_key`, die für jede Klasse eindeutig sein muß. Es liegt in der Verantwortung der einzelnen Klassen, `tinfo_key` im Konstruktor mit dem richtigen Wert zu füllen. Werte für die Basisklassen von GiNaC sind in `tinfos.h` definiert.

2.2.4.3 Die Hashfunktion in GiNaC

Die standardmäßig in GiNaC verwendete Hashfunktion benutzt als Baustein die Abbildung

$$\begin{aligned} g : \mathbb{N}_0 &\to [0,1[\\ n &\mapsto m - [m] \quad \text{mit} \quad m = \frac{\sqrt{5}-1}{2} n \,. \end{aligned} \tag{2.9}$$

$[m]$ ist die größte ganze Zahl, die kleiner oder gleich m ist. Betrachtet man die Funktionswerte $g(1), \ldots, g(n)$, so teilen sie das Intervall $[0,1[$ in $n+1$ Teilintervalle auf. $g(n+1)$ hat dann die Eigenschaft, daß es innerhalb des größten Teilintervalls liegt [47]. Die Funktionswerte werden also gleichmäßig auf das Intervall $[0,1[$ verteilt.

Daraus kann man eine Hashfunktion

$$\begin{aligned} h_{gr} : \mathbb{N}_0 &\to \{0, \ldots, N-1\} \\ n &\mapsto [Ng(n)] \end{aligned} \tag{2.10}$$

auf den natürlichen Zahlen definieren. In der Praxis wählt man $N = 2^{32}$ und $n \in \{0, \ldots, N-1\}$. $g(n)$ muß dann intern m mit mindestens 64 Bit Genauigkeit (Mantisse) berechnen, damit $m - [m]$ noch 32 signifikante Bits hat. ANSI C++ garantiert *nicht* die Existenz eines geeigneten Datentyps. Auf den meisten Systemen existiert aber ein Datentyp `long double`, `unsigned long` oder `unsigned long long` (kein ANSI Typ), der diese Eigenschaften besitzt. $\frac{\sqrt{5}-1}{2} = 0,618\ldots$ ist der Goldene Schnitt, daher bezeichnet man die Hashfunktion auch als Goldener-Schnitt-Hashfunktion.

Damit läßt sich nun eine Hashfunktion auf Ausdrücken definieren. Sei $e \in \mathcal{E}$ ein Ausdruck mit Unterausdrücken e_i, $i = 1, \ldots, k > 0$ (z.B. $e = \arctan(x,y) \Rightarrow e_1 = x, e_2 = y$). Dann wird h rekursiv definiert als

$$\begin{aligned} h : \mathcal{E} &\to \{0, \ldots, N-1\} \\ e &\mapsto \mathrm{rot}(\ldots \mathrm{rot}(\mathrm{rot}(h_{gr}(\mathrm{tinfo\_key}(e))) \operatorname{xor} h_{gr}(e_1)) \operatorname{xor} h_{gr}(e_2) \\ &\qquad \ldots) \operatorname{xor} h_{gr}(e_k) \end{aligned} \tag{2.11}$$

mit der Links-Rotation der $b = \log_2 N$ Bits von n

$$\mathrm{rot}(n) = (2n) \bmod N + [2n/N] \tag{2.12}$$

und der binären Exklusiv-Oder Verknüpfung $x \operatorname{xor} y$.

Terminale Ausdrücke, also solche ohne Unterausdrücke, wie z.B. Symbole, können einen beliebigen Hashwert zurückgeben. Jede Klasse hat die Möglichkeit, eine an ihre Bedürfnisse angepaßte Hashfunktion zu verwenden.

2.2.4.4 Vergleich von Ausdrücken

Der in GiNaC benutzte Vergleichsalgorithmus für zwei Ausdrücke $e_1, e_2 \in \mathcal{E}$ lautet damit wie folgt:

- vergleiche $h(e_1)$ mit $h(e_2)$.
  - ▷ $h(e_1) < h(e_2) \Rightarrow e_1 \prec e_2$
  - ▷ $h(e_1) > h(e_2) \Rightarrow e_1 \succ e_2$
  - ▷ $h(e_1) = h(e_2) \Rightarrow$ gehe zum nächsten Punkt
- vergleiche tinfo_key(e_1) mit tinfo_key(e_2)
  - ▷ tinfo_key(e_1) < tinfo_key(e_2) $\Rightarrow e_1 \prec e_2$
  - ▷ tinfo_key(e_1) > tinfo_key(e_2) $\Rightarrow e_1 \succ e_2$
  - ▷ tinfo_key(e_1) = tinfo_key(e_2) $\Rightarrow$ gehe zum nächsten Punkt
- e_1 und e_2 sind vom selben Typ. Es liegt nun in der Verantwortung der Klasse, eine Ordnungsrelation, die Gl. (2.3) und Gl. (2.4) erfüllt, auf zwei Objekten gleichen Typs zu definieren. Die durchgängig verwendete Strategie ist:
  - ▷ vergleiche weitere Datenfelder, die nicht Unterausdrücke sind (beispielsweise Flags) und entscheide beim ersten verschiedenen $e_1 \prec e_2$ oder $e_1 \succ e_2$,
  - ▷ vergleiche die Anzahl der Unterausdrücke (sofern vorhanden),
  - ▷ vergleiche die Unterausdrücke rekursiv,
  - ▷ wenn alle Datenfelder und Unterausdrücke gleich sind, ist $e_1 \simeq e_2$.

$e_1 \prec e_2$ bedeutet dabei $(e_1 \preceq e_2) \wedge (e_1 \not\simeq e_2)$.

GiNaC-intern werden Ausdrücke mit den Methoden `compare()` und `is_equal()` verglichen, die zunächst die Hashes und dann die TypeIDs vergleichen. Konnte danach noch nicht ausgeschlossen werden, daß beide Ausdrücke gleich sind, werden die virtuellen Funktionen `compare_same_type()` und `is_equal_same_type` aufgerufen, die dann von der jeweiligen Klasse implementiert werden müssen. Es genügt, `compare_same_type()` zu implementieren, da `is_equal_same_type` standardmäßig `compare_same_type()` aufruft, sofern eine Elternklasse nicht bereits `is_equal_same_type` überschrieben hat.

2.2.5 Methoden

GiNaC stellt eine Reihe von virtuellen Funktionen in `basic` zur Verfügung, die von abgeleiteten Klassen überschrieben werden müssen oder können. Funktionsargumente werden nur angegeben, wenn in der Erläuterung Bezug auf sie genommen wird. Eine vollständige Referenz findet sich in der automatisch erstellten GiNaC-Dokumentation [3].

2.2.5.1 Interne Methoden

Einige Methoden müssen von jeder abgeleiteten Klasse überschrieben werden, damit GiNaC zuverlässig arbeitet:

- `duplicate()`: Diese Methode simuliert einen virtuellen Copy-Konstruktor, indem sie einen Zeiger auf eine dynamische Kopie des aktuellen Objektes zurückgibt. Sie wird für die Speicherverwaltung mit *Reference Counting* (Abschnitt 2.2.2) benötigt, die mit dynamischen Objekten arbeitet.

- `compare_same_type()`: Jede Klasse muß, wie im Abschnitt 2.2.4 beschrieben, eine vollständige (aber beliebige) Ordnungsrelation auf ihren eigenen Objekten aufbauen. Der Rückgabewert ist -1, 0 oder 1, je nachdem, ob das erste kleiner, gleich oder größer dem zweiten verglichenen Objekt ist.

- `is_equal_same_type()`: Da es in einigen Fällen einfacher ist, Ungleichheit als eine Sortierreihenfolge festzustellen, gibt es auch diese Methode. `is_equal_same_type()` muß äquivalent zu `compare_same_type()==0` sein.

- `calchash()`: Berechnet die in Abschnitt 2.2.4.3 beschriebene Hashfunktion. Insbesondere terminale Objekte (Symbole, Indizes, Zahlen) sollten die in `basic` definierte Standardmethode aus Gl. (2.11) überschreiben, da sie keine Unterausdrücke haben und daher beispielsweise alle Symbole denselben Hashwert erhalten würden.

Diese Liste wird in Zukunft noch um Methoden zur Archivierung von Ausdrücken (Abschnitt 2.6) erweitert werden.

2.2.5.2 Methoden zur Ausgabe

GiNaC-Ausdrücke können in verschiedener Form auf dem Bildschirm oder in Dateien ausgegeben werden:

- `print()`: Dies ist das Standardformat der Ausgabe. Sie wird verwendet, wenn man `ex`-Objekte mit `<<` in ein `ostream`-Objekt einfügt. Das Format sollte vor allem lesbar sein und sich auf wesentliche Informationen beschränken. Überflüssige Klammern werden durch ein Prioritätensystem vermieden (z.B. Potenzen vor Multiplikation vor Addition vor Relationen).

- `printraw()`: Gibt alle in einem Objekt gespeicherte Information (inklusive Statusinformationen und Hashwerte) in einer Zeile aus. Dient ausschließlich zum Debuggen.

- `printtree()`: Ähnlich wie `printraw()`, aber Ausgabe in Baumform durch Einrükkung der Unterausdrücke.

- `printcsrc()`: Gibt einen Ausdruck in einer Form aus, die es erlaubt, ihn in ein C-Programm einzubetten, z.B. x^y als `pow(x,y)`. Zahlen können als `double` oder im CLN-Format ausgegeben werden.

Keine dieser Ausgabeformen ist zur Zeit geeignet, Ausdrücke auch wieder aus einer Datei einzulesen. Dies ist jedoch für die Zukunft geplant (Abschnitt 2.6). Ebenso ist ein spezielles Ausgabeformat für LaTeX denkbar.

2.2.5.3 Zugriff auf Unterausdrücke

Häufig enthalten Ausdrücke wieder Unterausdrücke, z.B. ein `power`-Objekt Basis und Exponenten, oder ein `add`-Objekt seine Summanden. Für deren Handhabung werden folgende Methoden zur Verfügung gestellt:

- `nops()`: Gibt die Anzahl der Unterausdrücke zurück.
- `op(i)`: Gibt eine *Kopie* des i-ten Unterausdrucks zurück, wobei $0 \leq i <$ `nops()` ist. Beispielsweise ist bei einer Relation (Abschnitt 2.3.6) `op(0)` die linke und `op(1)` die rechte Seite.
- `let_op()`: Gibt eine *Referenz* auf den i-ten Unterausdruck zurück, der dadurch verändert werden kann. Da dadurch die Evaluierungslogik (Abschnitt 2.2.3) umgangen wird, sollte sie nur bei Klassen verwendet werden können, die keine weitere Evaluierung benötigen, z.B. `exprseq` oder `lst`.
- `operator[]()`: wie `op()`, erlaubt aber auch Indizierung über einen beliebigen Ausdruck (standardmäßig jedoch nur über einen Index zwischen 0 und `nops()-1`).

2.2.5.4 Methoden auf ganz- und gebrochen-rationalen Funktionen

- `expand()`: Bringt einen Ausdruck auf total expandierte Form.
- `degree(s)`: Gibt den Exponenten der höchsten vorkommenden Potenz (den Grad) eines multivariaten Polynoms in einem Symbol s zurück. Der Ausdruck muß nicht expandiert sein.
- `ldegree(s)`: Entsprechend für die niedrigste Potenz.
- `coeff(s,n)`: Gibt den Koeffizienten von s^n zurück. Der Ausdruck muß dazu expandiert sein.
- `collect(s)`: Sammelt die Koeffizienten eines multivariaten Polynoms nach Potenzen der Variable s.
- `normal()`: Vereinfacht rationale Funktionen, indem Zähler und Nenner expandiert und gemeinsame Faktoren gekürzt werden. Zähler und Nenner des Ergebnisses sind also teilerfremd.

2.2.5.5 Ableitungen

- `diff(s)`: Berechnet das totale Differential $\frac{d}{ds}$.
- `series(s,x,n)`: Berechnet die Taylor- bzw. Laurent-Entwicklung eines Ausdrucks in der Variablen s um den Punkt x bis zur Ordnung $\mathcal{O}(s^n)$.

2.2.5.6 Sonstige Methoden

- `info()`: Dient zum Abfragen verschiedener Informationen über einen Ausdruck, z.B. über den Typ, Zustände innerhalb eines Typs (positiv oder negativ bei `numeric`) oder Eigenschaften, die den Ausdruck als ganzes betreffen (z.B. rationale Funktion).

- `has()`: Gibt an, ob ein Ausdruck gleich einem anderen ist oder diesen als Unterausdruck enthält. `has()` testet nur einzelne Operanden, nicht Kombinationen davon. Nach dieser Definition ist $y + z$ *nicht* in $x + y + z$ enthalten.
- `subs()`: Ersetzt eine Liste von Symbolen oder Indizes durch beliebige andere Ausdrücke. Der entstehende Ausdruck ist, im Gegensatz zu einer Veränderung durch `let_op()`, evaluiert.
- `eval()`: Evaluiert einen Ausdruck, d.h. es werden einige Vereinfachungen ausgeführt. Diese Methode wird automatisch von GiNaC aufgerufen (Abschnitt 2.2.3).
- `evalf()`: Wertet einen Ausdruck soweit wie möglich in der eingestellten Genauigkeit numerisch als Gleitkommazahl aus.
- `get_indices()`: Gibt eine Liste aller Indizes eines Ausdrucks zurück, z.B. bei einem `mul`-Objekt die Vereinigung der Indizes der Faktoren.
- `return_type()`: Ausdrücke in GiNaC können kommutativ oder nicht-kommutativ sein. Nicht-kommutative Ausdrücke können entweder nur mit anderen Ausdrücken vom selben Typ oder mit allen nicht-kommutativen Ausdrücken nicht kommutieren, beispielsweise ein noch nicht evaluierter Vertex in einem Feynman-Graphen, der ein Produkt aus einem Clifford- und einem SU(3)-Algebra-Element sein kann (Details siehe Abschnitt 2.4.1). `return_type()` gibt daher einen der Werte `commutative`, `noncommutative` oder `noncommutative_composite` zurück.
- `return_type_tinfo()`: Gibt für nicht-kommutative Objekte die effektive TypeID (siehe Abschnitt 2.4.1) zurück, die von der TypeID des Objektes selbst abweichen kann. Beispielsweise hat eine Funktion `slash(p)` ($= p^\mu \gamma_\mu$) die TypeID `function`, aber `return_type_tinfo()` gibt die TypeID eines `clifford`-Objektes zurück.

Weiterhin gibt es noch eine Reihe virtueller Methoden, die als Hilfsfunktionen für andere dienen und hier nicht näher erläutert werden sollen.

2.3 Grundlegende Klassen

Einfache Ausdrücke werden in GiNaC durch Kombination der terminalen Objekte *Symbol*, *Konstante* und *Zahl* mit den Grundrechenarten $+$, $-$, $\times$, $\div$ und x^y, Anwenden eingebauter oder benutzerdefinierter Funktionen darauf oder Bilden von Relationen erzeugt.

2.3.1 Ganzzahl- und Gleitkomma-Arithmetik

Für den überwiegenden Teil der numerischen Berechnungen bedient sich GiNaC der C++-Bibliothek CLN [37], die wiederum auf der C-Bibliothek GNU MP [36] beruht. GNU MP stellt Datentypen für ganze Zahlen beliebiger Größe, rationale Zahlen und Gleitkommazahlen beliebiger Genauigkeit sowie die Grundrechenarten darauf zur Verfügung. GNU MP ist portabel, effizient in allen Genauigkeitsbereichen (sowohl bei 100 als auch bei 1.000.000 Stellen Genauigkeit), benutzt optimierte Assembler-Routinen auf den meisten verfügbaren Mikroprozessoren und ist frei im Quellcode unter der GNU Lesser General Public License (LGPL) [29] verfügbar. GNU MP ist weit verbreitet, da es Grundlage für die Implementierung des RSA Verschlüsselungsalgorithmus in dem Paket ssh ist, mit dem abhörsichere Verbindungen zwischen Computern hergestellt werden können.

CLN erweitert GNU MP um komplexe Zahlen, deren Real- und Imaginärteil ganze, rationale oder Gleitkommazahlen sein können, sowie um effiziente Algorithmen für transzendente Funktionen, exakte n-te Wurzeln u.v.m. Weiterhin kapselt CLN die GNU MP Datentypen in polymorphe C++-Klassen und definiert überladene Operatoren, die eine Benutzung sehr einfach machen. CLN ist frei im Quellcode unter der GNU General Public License (GPL) [28] verfügbar und gehört z.B. zum Lieferumfang der Debian-Linux-Distribution [18].

Als Nachteile sind anzumerken, daß CLN nur mit dem GNU C Compiler übersetzt werden kann. Dieser ist jedoch für fast alle Betriebssysteme und Prozessorarchitekturen verfügbar. Weiterhin erlaubt die Bibliothek nicht die Verwendung in parallelisierten Programmen.

Die GiNaC-Klasse `numeric` kapselt wiederum ein polymorphes CLN-Objekt. `numeric` kann dementsprechend sowohl eine ganze Zahl, eine rationale Zahl oder eine Gleitkommazahl, jeweils mit optionalem Imaginärteil, enthalten. Den aktuellen Typ kann man mit der Methode `info()` erfragen. Verknüpft man zwei `numeric`-Objekte, von denen mindestens eines eine Gleitkommazahl ist, mit $+$, $-$, $\times$, $\div$ oder x^y, so ist das Ergebnis wieder eine Gleitkommazahl, wobei die Genauigkeit durch das ungenaueste Objekt vorgegeben wird. Ansonsten wird exakt weitergerechnet. Die Genauigkeit stellt man mit Hilfe der globalen Variablen `Digits` ein.

`numeric`-Objekte lassen sich durch expliziten Konstruktor-Aufruf aus den verschiedenen C `int`-Typen (Ganzzahl), zwei `long` (rationale Zahlen) oder einem `double` (Gleitkommazahl) erzeugen. Weiterhin können Zeichenketten und der CLN-Typ `cl_N` in ein GiNaC-`numeric` umgewandelt werden.

Zur Berechnung des Hashwertes von `numeric`-Objekten wird auf die CLN-Funktion `cl_equal_hashcode()` zurückgegriffen.

2.3.2 Symbole und Konstanten

Ein Symbol x wird in GiNaC durch eine Anweisung

```
symbol x("sym_x");
```

definiert. Dabei muß man drei Aspekte des Symbols unterscheiden. Zum einen gibt es damit ein Objekt `x` vom Typ `symbol`, mit dem im folgenden Programm gerechnet wird, z.B. in einer Anweisung

```
ex e=diff(sin(x),x);
```

Außerdem gibt es den beschreibenden Namen des Symbols `"sym_x"`, der *ausschließlich* bei Ausgabe verwendet wird:

```
cout << e << endl; // druckt 'cos(sym_x)'
```

Zuletzt bekommt jedes Symbol eine eindeutige Seriennummer, die intern von GiNaC bei jedem neu erzeugten Symbol inkrementiert wird. Nur diese Seriennummer wird dazu verwendet zu entscheiden, ob zwei Symbole identisch oder verschieden sind. Die Seriennummer geht auch in die Berechnung des Hashwertes (Abschnitt 2.2.4) mit ein.

In der Regel sollte man Symbolen denselben beschreibenden Namen wie der Variablen selbst geben. Der Name ist jedoch optional. Läßt man ihn weg, wie in

```
symbol y;
```

so erzeugt GiNaC aus der Seriennummer einen Namen wie z.B. `"symbol17"`.

Symbolen kann kein Wert zugewiesen werden[2]. Ausdrücke muß man Variablen vom Typ `ex` zuweisen.

Konstanten können in Ausdrücken ähnlich wie Symbole benutzt werden. Zusätzlich geben sie bei Anwendung der Methode `evalf()` ihren numerischen Wert zurück. Dieser Wert kann in einer festgelegten Genauigkeit hinterlegt werden (z.B. für physikalische Konstanten), beispielsweise

```
constant qe("qe",numeric(1.602e-19));
cout << qe << endl;          // druckt 'qe'
cout << evalf(qe) << endl;   // druckt '1.602e-19'
```

oder durch Aufruf einer Gleitkomma-Evaluierungsfunktion zur aktuell eingestellten Genauigkeit berechnet werden. GiNaC definiert die Konstanten `Pi=3.141593...`, `Catalan=0.9159656...` und `EulerGamma=0.5772157...`, die mit einer beliebigen Genauigkeit evaluiert werden können. Die imaginäre Einheit i ist keine Konstante im Sinne der `constant`-Klasse, sondern ein `const numeric I`.

2.3.3 Summen und Produkte

GiNaC kennt keine von `basic` abgeleiteten Datentypen, die direkt Differenzen bzw. Quotienten von beliebigen Ausdrücken speichern. Statt dessen arbeitet GiNaC intern immer mit Summen bzw. Produkten. Eine Differenz $x - y$ wird daher als $x + (-1)y$ dargestellt, ebenso ein Quotient $\frac{x}{y}$ als xy^{-1}.

Untersucht man systematisch die einzelnen algorithmischen Schritte beim Zusammenfassen von Termen in Summen (und Differenzen)

$$\begin{aligned} 3x+5y+z-y+4z-2x &\rightarrow 3x+(-2)x+5y+(-1)y+1z+4z \\ &\rightarrow (3-2)x+(5-1)y+(1+4)z \\ &\rightarrow 1x+4y+5z \\ &\rightarrow x+4y+5z \end{aligned} \tag{2.13}$$

und Produkten (sowie Quotienten)

$$\begin{aligned} \frac{x^3y^5zz^4}{x^2y} &\rightarrow x^3x^{-2}y^5y^{-1}z^1z^4 \\ &\rightarrow x^{3-2}y^{5-1}z^{1+4} \\ &\rightarrow x^1y^4z^5 \\ &\rightarrow xy^4z^5\,, \end{aligned} \tag{2.14}$$

so findet man folgende Gemeinsamkeiten:

- Terme ohne Vorfaktor bzw. Potenz kann man zunächst um einen Faktor 1 bzw. eine Potenz 1 erweitern.
- Die Terme können sortiert werden, um passende Terme identifizieren zu können, da sowohl Summe als auch Produkt kommutativ sind.

[2]Die interaktive GiNaC-Shell ginsh (Abschnitt 2.5.2) weist intern auch Symbolen Werte zu. Diese Option sollte aber nicht von anderen Programmen benutzt werden, da die automatische Evaluierung von Ausdrücken gestört wird.

- Bei zusammenfaßbaren Termen addiert man die Vorfaktoren bzw. die Potenzen.
- Überflüssige Faktoren bzw. Potenzen 1 können im Ergebnis weggelassen werden.

Weiterhin sind sowohl Summe als auch Produkt assoziativ. Um Klammern zu vermeiden, sollten diese automatisch aufgelöst werden:

$$\begin{aligned} a+(b+c)+d &\rightarrow a+b+c+d \\ a(bc)d &\rightarrow abcd \end{aligned} \tag{2.15}$$

2.3.3.1 Die Klasse `expairseq`

Diese Ähnlichkeit legt es nahe, die gemeinsame Funktionalität in einer Basisklasse für Summe und Produkt zu implementieren, die sich wie folgt charakterisieren läßt:

- Alle Terme werden in Paare aus einem numerischen Koeffizienten und einem nichtnumerischen Rest zerlegt. Gibt es keinen solchen Koeffizienten, wird er auf 1 gesetzt.
- Ist einer der Summanden einer Summe bzw. der Faktoren eines Produkts selbst eine Summe bzw. ein Produkt, so werden deren Terme einzeln übernommen (Assoziativität).
- Auf den Paaren aus Koeffizient und Rest kann mit Hilfe der Ordnungsrelation auf `ex` ebenfalls eine Ordnungsrelation definiert werden (auch die numerischen Koeffizienten c gehören zur Menge aller Ausdrücke $\mathcal{E}$):

$$\begin{aligned} (r_1,c_1) \simeq (r_2,c_2) &\Leftrightarrow (r_1 \simeq r_2) \wedge (c_1 \simeq c_2) \\ (r_1,c_1) \prec (r_2,c_2) &\Leftrightarrow r_1 \prec r_2 \vee ((r_1 \simeq r_2) \wedge (c_1 \prec c_2))\,. \end{aligned} \tag{2.16}$$

  Damit werden die Terme sortiert.
- Paare (r_1,c_1) und (r_2,c_2) mit $r_1 \simeq r_2$ stehen nun nebeneinander und können durch ein Paar (r_1, c_1+c_2) ersetzt werden.
- Rein numerische Terme können gesondert behandelt werden. Sie müssen nicht in ein Paar zerlegt werden, sondern können direkt addiert bzw. multipliziert werden.

Speziell bei Produkten kann es vorkommen, daß nach dem Durchgang des Zusammenfassens diese Schritte erneut durchgeführt werden müssen, z.B. bei

$$\begin{aligned} (xy)^{1/2}x(xy)^{3/2} &\rightarrow (xy)^{1/2}(xy)^{3/2}x^1 \\ &\rightarrow (xy)^{1/2+3/2}x^1 \\ &\rightarrow (xy)^2x^1\,. \end{aligned} \tag{2.17}$$

Um das gewünschte Ergebnis x^3y^2 zu erhalten, muß zunächst $(xy)^2 \rightarrow x^2y^2$ umgewandelt (siehe Abschnitt 2.3.4) und dann erneut zusammengefaßt werden.

Diese Funktionalität wird von der „abstrakten“ GiNaC-Klasse `expairseq`[3] zur Verfügung gestellt. `expairseq`-Objekte haben zwei wichtige Konstruktoren. Zum einen ein

[3] `expairseq` ist keine abstrakte Klasse im Sinne der C++-Definition. Es können Objekte von diesem Typ angelegt werden, aber sie haben keinen praktischen Nutzen.

Konstruktor, der *zwei* Ausdrücke des Typs `ex` als Argument hat, zum anderen ein Konstruktor von einer beliebigen Anzahl von Objekten des Typs `ex`. Der erste Konstruktor wird beim Verknüpfen von Ausdrücken mit + und × (und damit auch − und ÷) benutzt. Es gibt optimierte Routinen, falls einer oder beide der Operanden selbst vom Typ Summe bzw. Produkt sind. In diesem Fall wird ausgenutzt, daß mindestens einer der beiden Operanden schon vorsortiert ist (siehe auch Abschnitt 2.3.3.4). Der zweite Konstruktor wird meist von internen Methoden wie `expand()` genutzt, wo erst alle expandierten Summanden gesammelt werden und dann zu einer Summe verknüpft werden.

Die interne Datenstruktur der Klasse benutzt die Hilfsklasse `expair`, die zwei Unterobjekte vom Typ `ex` für den numerischen Koeffizienten und den nicht-numerischen Rest enthält, mit der Ordnungsrelation aus Gl. (2.16). In einer STL-Kollektion `vector<expair>` kann dann eine variable Zahl von Paaren gespeichert werden. Hinzu kommt eine Variable `overall_coeff`, die die direkt zusammenfaßbaren numerischen Anteile speichert.

Eine von `expairseq` abgeleitete Klasse muß folgende Information (in Form von virtuellen Funktionen) zur Verfügung stellen:

- den Startwert für `overall_coeff` (0 bei Summe, 1 bei Produkt),
- eine Regel zum Verknüpfen des `overall_coeff` mit einem neuen numerischen Operanden (addieren bzw. multiplizieren),
- Aufteilen eines Operanden in ein Paar aus Koeffizient und Rest,
- Rekombinieren eines Paares in den ursprünglichen Operanden.

2.3.3.2 Die Klasse `mul`

Die von `expairseq` abgeleitete Klasse `mul` repräsentiert Produkte. Potenzen mit beliebigen numerischen Exponenten werden in Paare aus Exponent und Basis für Koeffizienten und Rest umgewandelt. Andere Ausdrücke bilden ein Paar aus Ausdruck und 1.

An Vereinfachung werden in `eval()` ausgeführt:

- Wenn das Produkt keine Paare von Ausdrücken mehr hat, ist das Ergebnis der `overall_coeff` (z.B. $x^2 \cdot 3 \cdot x^{-2} \to 3$).
- Wenn das Produkt nur noch aus einem Paar und dem `overall_coeff` 1 besteht, ist das Ergebnis das rekombinierte letzte Paar (z.B. $x^3 x^{-2} \to x$).
- Wenn der `overall_coeff` Null ist, ist das ganze Produkt Null.
- Wenn das Produkt nur noch ein Paar aus einer Summe und einem Koeffizienten (=Potenz) 1 ist, wird der `overall_coeff` auf die Summanden verteilt, das Ergebnis ist vom Typ `add`, beispielsweise in $5(x+3y-2z+4) \to 5x+15y-10z+20$. Warum nur bei Zahlen das Distributivgesetz eine tatsächliche Vereinfachung auf Objektebene ist, zeigt Abb. 2.3. Auch Maple führt diese Vereinfachung durch, da es intern eine ähnliche Datenstruktur benutzt [34].

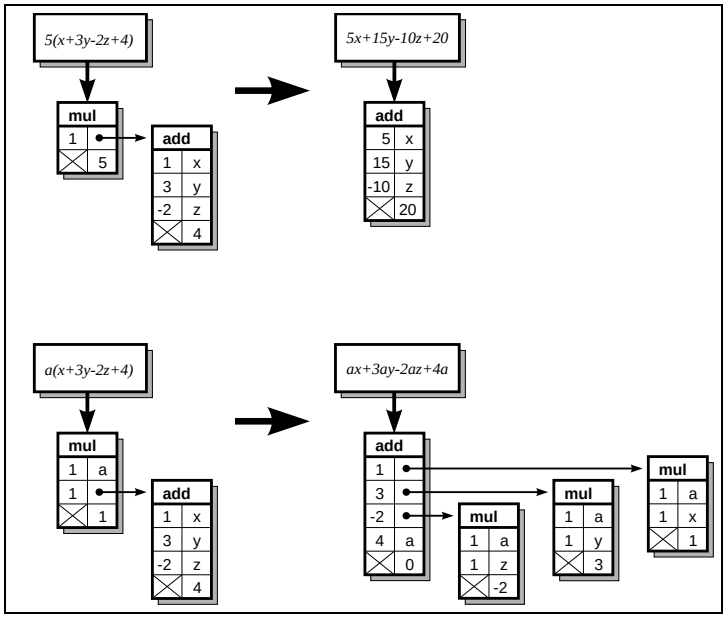


Abbildung 2.3: Anwendung des Distributivgesetzes auf $5(x + 3y - 2z + 4)$ (oben) und $a(x + 3y - 2z + 4)$ (unten). Nur im ersten Fall ist der umgewandelte Ausdruck einfacher als der ursprüngliche.

2.3.3.3 Die Klasse `add`

Die ebenfalls von `expairseq` abgeleitete Klasse `add` zur Repräsentation von Summen teilt Ausdrücke vom Typ `mul` in Koeffizienten und Rest auf, indem sie dessen `overall_coeff` als Koeffizienten nimmt und ein neues, kopiertes `mul`-Objekt, dessen `overall_coeff` auf 1 gesetzt wird, als Rest. Alle anderen Ausdrücke ergeben wieder ein Paar aus diesem Ausdruck und Koeffizienten 1.

Da die grundlegenden Vereinfachungen, die man von einem Summen-Objekt erwartet, bereits im Konstruktor der Basisklasse `expairseq` durchgeführt werden, bleibt in der Methode `eval()` nicht viel zu tun:

- Wenn die Summe keine Paare von Ausdrücken mehr hat, ist das Ergebnis keine Summe mehr, sondern nur noch der `overall_coeff` (z.B. bei $x + 2 - x \to 2$).

- Wenn die Summe nur noch aus einem Paar und dem `overall_coeff` Null besteht, ist das Ergebnis das rekombinierte letzte Paar (z.B. $x + 3 + 2y - 1 - x - 2 \to 2y$).

2.3.3.4 Zusammenfassen durch Hash-Tabellen

Das in Abschnitt 2.3.3.1 vorgestellte Verfahren zum Zusammenfassen von n Summanden in einer Summe oder Faktoren in einem Produkt benötigt $\mathcal{O}(n \log n)$ Zeit, die dominiert wird vom Sortieren der Terme. Das anschließende Vergleichen benachbarter Terme benötigt nur $\mathcal{O}(n)$. Dies kann theoretisch noch beschleunigt werden, wenn man Hash-Tabellen mit Verkettung bei Kollisionen benutzt [47, 65]. Der Algorithmus dazu lautet:

- Zerlege, wie in Abschnitt 2.3.3.1, alle Terme in Paare (r, c).

- Initialisiere eine Hash-Tabelle der Größe $M = 2^m$ mit $m = [\log_2 n] + k$, $k \in \mathbb{Z}$. Die Größe der Hash-Tabelle orientiert sich also an der Zahl der Terme und ist in etwa proportional zu ihr. k ist ein freier Parameter, der empirisch angepaßt werden kann. Je größer k ist, desto geringer ist die Wahrscheinlichkeit einer Hashkollision Gl. (2.8), aber um so größer ist der Zeitbedarf für die Initialisierung der Hash-Tabelle. Die Tabelle besteht aus einem Vektor von Listen von Zeigern auf Koeffizienten/Rest-Paare.

- Berechne zu jedem Paar (r, c) den Hashindex $i = h(r) \bmod M$, der nicht vom Koeffizienten abhängt.

- Falls es in der Hash-Tabelle zu i noch keinen Eintrag gibt, lege einen Zeiger auf das aktuelle Paar an.

- Falls es bereits mindestens einen Eintrag gibt, vergleiche das aktuelle Paar mit allen in der Liste und fasse gegebenenfalls zusammen oder hänge einen Zeiger auf dieses Paar an.

Dieses Verfahren hat eine Zeitkomplexität von $\mathcal{O}(M) \approx \mathcal{O}(n)$ zum Initialisieren der Hash-Tabelle und $\mathcal{O}((1 + \frac{\beta}{2})n)$ für n Zugriffe auf die Hash-Tabelle, wenn $\beta = \frac{n}{M}$ die Auslastung der Hash-Tabelle ist, unter der Voraussetzung, daß alle Einträge in der Hash-Tabelle gleichmäßig und unkorreliert gefüllt werden, die Hashfunktion also quasi-zufällig die Ausdrücke aus $\mathcal{E}$ auf die Indexmenge $\{0, \ldots, N-1\}$ bzw. $\{0, \ldots, M-1\}$ abbildet. Ein einzelner Zugriff braucht unter diesen Voraussetzungen im schlimmsten Fall eine Zeit von $\mathcal{O}(\frac{\log n}{\log \log n})$ (maximal erwartete Anzahl von Einträgen in der verketteten Liste).

Da $M \approx 2^k n$, ist $\beta = \mathcal{O}(1)$, und das Zusammenfassen von Termen sollte insgesamt $\mathcal{O}(n)$ sein. Dennoch hat das Verfahren auch Nachteile:

- Der Aufwand pro Term ist höher als beim Sortieren und Vergleichen benachbarter Terme, die Proportionalitätskonstante in $\mathcal{O}(n)$ ist also relativ groß.

- Beim Kopieren von Objekten muß die Hash-Tabelle mitkopiert oder neu angelegt werden.

- Das Vergleichen ($\simeq$ und $\prec$) von zwei `expairseq`-Objekten ist aufwendiger, da die Unterausdrücke nicht sortiert sind (kein lexikographischer Vergleich möglich).

- Fordert man zum einfacheren Vergleichen, daß immer $M = f(n)$ ist, so muß gegebenenfalls die Hash-Tabelle neu eingerichtet werden, wenn zwei Ausdrücke zusammengefaßt wurden und $f(n-1) \neq f(n)$.

- Der häufige Fall, daß ein `expairseq`-Objekt aus zwei anderen `expairseq`-Objekten bzw. einem `expairseq` und einem nicht-`expairseq`-Objekt erzeugt wird, braucht auch mit dem Verfahren aus Abschnitt 2.3.3.1 nur $\mathcal{O}(n)$, da die Objekte schon vorsortiert sind.

In der Praxis hat sich empirisch gezeigt, daß Hash-Tabellen zwar tatsächlich bei großen Ausdrücken asymptotisch schneller sind, die bisher benötigten Programme liefen jedoch langsamer. Etwas bessere Ergebnisse wurden mit einem hybriden Verfahren erzielt, bei dem für n unter einer einstellbaren Schwelle Sortieren und oberhalb Hash-Tabellen verwendet wurden. Die aktuelle GiNaC-Distribution benutzt *keine* Hash-Tabellen.

Algorithmus	**Zeitbedarf**
paarweises Vergleichen	$\mathcal{O}(n^2)$
Sortieren, Vergleichen benachbarter Ausdrücke	$\mathcal{O}(n \log n)$
Hash-Tabellen	$\mathcal{O}((1+\frac{\beta}{2})n)$

Tabelle 2.1: Zeitbedarf für das Zusammenfassen einer Summe mit n Summanden für verschiedene Algorithmen

2.3.4 Potenzen

Potenzen werden in GiNaC in Objekten vom Typ `power` gespeichert, die je ein Unterobjekt vom Typ `ex` für Basis und Exponent haben. Die Basis muß ein kommutatives Objekt sein (siehe Abschnitt 2.4.1.2).

Die Methode `eval()` führt folgende Vereinfachungen aus (x ist ein beliebiger Ausdruck):

- $x^0 \to 1$, Ausnahmefehler (Exception), wenn $x = 0$
- $x^1 \to x$
- $0^x \to 0$, wenn $x \in \mathbb{C}$, aber Ausnahmefehler (Exception), wenn $\mathrm{Re}(x) \leq 0$
- $1^x \to 1$
- Falls sowohl Basis b als auch Exponent e numerisch ($b, e \in \mathbb{C}$) sind:
  - Falls $b \notin \mathbb{Q}(i) \vee e \notin \mathbb{Q}(i)$, ist das Ergebnis $b^e \notin \mathbb{Q}(i)$ und wird von CLN als Gleitkommazahl berechnet. Dabei bezeichnet $\mathbb{Q}(i) \subset \mathbb{C}$ die Menge aller komplexen Zahlen $z = x + iy$ mit rationalem Real- und Imaginärteil, $x \in \mathbb{Q} \wedge y \in \mathbb{Q}$.
  - Falls $b \in \mathbb{Q}(i) \wedge e \in \mathbb{Q}(i)$, wird probeweise b^e in CLN berechnet. Wenn auch $b^e \in \mathbb{Q}(i)$, kann CLN dafür ein exaktes Ergebnis zurückgeben. Ansonsten werden die weiteren Regeln bearbeitet.
  - falls $e = \frac{n}{m} \in \mathbb{Q}$ mit $\frac{n}{m} = q + \frac{n'}{m}, q \in \mathbb{Z}, 0 < \frac{n'}{m} < 1$, wird als Ergebnis $b^q b^{\frac{n'}{m}}$ zurückgegeben, wobei b^q von CLN berechnet wird.
  - In allen anderen Fällen wird b^e unevaluiert zurückgegeben.
- $(x^a)^b \to x^{ab}$ mit $b \in \mathbb{Z} \vee -1 < a \leq 1$
- $(\prod_i x_i)^n = \prod_i x_i^n$ mit $n \in \mathbb{Z}$
- $(ax)^b \to a^b x^b$ mit $a \in \mathbb{R}_0^+ \wedge b \in \mathbb{C}$
- $(ax)^b \to |a|^b (-x)^b$ mit $a \in \mathbb{R}^- \wedge b \in \mathbb{C}$

Den Beweis für die letzten vier Regeln findet man in Anhang A.

2.3.5 Funktionen

Normale Funktionen in C++ sind nicht geeignet, um Funktionen auf symbolischen Ausdrücken zu definieren, die sich wie in anderen CAS verhalten. Sei x ein Symbol, dann ist $\sin(x)$ ein Ausdruck, den man ohne weitere Kenntnisse über x (z.B. Vielfaches von π) nicht weiter evaluieren kann. Wäre `ex sin(ex const & x) {...}` eine C++-Funktion, so müßte sie nach Analyse des Arguments „sich selbst“ mit `return sin(x);` zurückgeben. Dies würde aber zu einer unendlichen Rekursion führen.

GiNaC stellt daher eine Klasse `function` bereit, die über die Hilfsklasse `exprseq`[4] von `basic` abgeleitet ist und damit zu einem in Summen, Produkten etc. verwendbaren Ausdruck wird. `function`-Objekte werden durch einen eindeutigen Funktions-Index (z.B. Sinus=1, Kosinus=2, ...) unterschieden, der beim Initialisieren des Programms vergeben wird und bei jedem Konstruieren eines Objektes mitgegeben werden muß. Um einen Funktionsaufruf $\sin(x)$ auch als `sin(x)` und nicht als `function(1,x)` schreiben zu können, bedient GiNaC sich des Präprozessors, der eine entsprechende `inline`-Funktion erzeugt.

Um beispielsweise eine GiNaC-Funktion `sqr(x)` zu deklarieren, die numerische Argumente quadriert, symbolische aber beläßt, schreibt man in die Header-Datei:

```
DECLARE_FUNCTION_1P(sqr)
```

`_1P` zeigt dabei an, daß die Funktion von einem Parameter abhängt. Die Implementation sieht dann wie folgt aus:

```
ex sqr_eval(ex const & x)
{
    if (is_ex_exactly_of_type(x,numeric)) return x*x;
    return sqr(x).hold();
}

REGISTER_FUNCTION(sqr,eval_func(sqr_eval))
```

Dem Makro `REGISTER_FUNCTION` können noch weitere Optionen übergeben werden, beispielsweise eine Funktion zur numerischen Evaluierung (`evalf_func()`), zur Berechnung von Ableitungen (`diff_func()`), `return_type()`-Information, Evaluierungsreihenfolge der Argumente bei `evalf()` oder ein Name zur LaTeX-Ausgabe. Verzichtet man auf die Makros, kann man auch überladene Funktionen definieren (z.B. die Riemannsche Zetafunktion $\zeta(x)$ und $\zeta(n,x)$).

GiNaC kennt zur Zeit die folgenden Funktionen (mit numerischer Evaluierung, Ableitungen und Laurent-Entwicklungen): Sinus, Kosinus, Tangens, Exponentialfunktion, natürlicher Logarithmus, inverse trigonometrische Funktionen (u.a. Arkustangens mit einem und zwei Argumenten), hyperbolische Funktionen, inverse hyperbolische Funktionen, Dilogarithmus, Trilogarithmus, Riemannsche Zetafunktion $\zeta(x)$ und deren Ableitungen $\zeta(n,x)$. Gamma-, Polygamma und Beta-Funktion existieren für ganz- und halbzahlige Argumente.

[4] Die Klasse `exprseq` basiert auf der STL Kollektion `vector<ex>`, die wahlfreien Zugriff auf eine variable Zahl von Unterausdrücken erlaubt. Sie definiert die Methoden `eval()`, `evalf()`, `subs()` etc. durch Wirken auf die Unterausdrücke. `exprseq` dient auch als „abstrakte“ Basisklasse für `ncmul` (Abschnitt 2.4.1.2) und `indexed` (Abschnitt 2.4.2).

2.3.6 Relationen

Gleichungen und Ungleichungen werden in GiNaC durch ein `relational`-Objekt dargestellt und mit Hilfe der in C++ üblichen Operatoren `==` (für $=$), `!=` (für $\neq$), `<`, `<=`, `>` und `>=` gebildet.

Relationen führen keinerlei Vereinfachungen auf beiden Seiten durch. Es gibt jedoch eine automatische Konvertierungsfunktion von `relational` nach `bool`, durch die es möglich wird,

```
if (expand(pow(a+b,2))==a*a+2*a*b+b*b) {
    ...
}
```

zu schreiben. `==` testet dabei Gleichheit im Sinne der Äquivalenzrelation aus Abschnitt 2.2.4, indem die rechte von der linken Seite abgezogen wird und das Ergebnis mit Null verglichen wird. Die Vereinfachung wird vom Konstruktor der Klasse `expairseq` und der Methode `eval()` der Klasse `add` (siehe Abschnitt 2.3.3) durchgeführt. Insbesondere wird *nicht* die Methode `normal()` (Abschnitt 2.2.5.4) aufgerufen. Relationen mit `<`, `<=`, `>` oder `>=` innerhalb von `if (...)` sind nur sinnvoll, wenn die Differenz numerisch und reell ist. Diese Operatoren haben keine Verbindung zu der Ordnungsrelation auf Ausdrücken $\preceq$ (Abschnitt 2.2.4).

GiNaC kann Systeme von *linearen* Gleichungen mit Hilfe der Funktion `lsolve()` lösen, die intern das Gleichungssystem in eine Matrix transformiert und dann Methoden der Klasse `matrix` (Abschnitt 2.4.3) anwendet.

2.4 Spezielle Klassen

Neben den bisher vorgestellten grundlegenden Klassen in GiNaC, mit denen die Basisfunktionen eines typischen CAS modelliert werden, gibt es noch eine Reihe weiterer spezieller Klassen, die zum Teil sehr konkret im Hinblick auf Anwendungen in der Hochenergiephysik (siehe Kapitel 3) entworfen wurden, andererseits aber auch vom Konzept her flexibel genug für weitere Anwendungen in der Physik und Mathematik sind.

2.4.1 Nicht-kommutative Produkte

Eines der wesentlichen Konzepte beim Vereinfachen von Produkten ist das Sortieren der Faktoren bezüglich der definierten Ordnungsrelation Gl. (2.4) und danach Zusammenfassen nun benachbarter Terme (vgl. Abschnitt 2.3.3). Dabei wird implizit vorausgesetzt, daß die Faktoren kommutieren. Möchte man jedoch auch mit nicht-kommutierenden Objekten arbeiten (z.B. Matrizen, Elemente der SU(N)- oder Clifford-Algebra), muß man außer einer Klasse für nicht-kommutative Produkte auch eine Methode zum Unterscheiden zwischen kommutierenden und nicht-kommutierenden Objekten einführen.

2.4.1.1 `return_type`-Information

Die Methode `return_type()` gibt Auskunft darüber, ob ein Ausdruck kommutativ ist oder nicht. Es gibt folgende Rückgabewerte:

- `commutative`: Alle Terme des Ausdrucks sind kommutativ.

- `noncommutative`: Alle nicht-kommutativen Terme des Ausdrucks gehören zur selben nicht-kommutativen Algebra (einige Terme können auch kommutativ sein). Die TypeID (Abschnitt 2.2.4.2) dieser Algebra kann dann mit der `return_type_tinfo()`-Methode abgefragt werden. `return_type_tinfo()` gibt die *effektive* TypeID eines Ausdrucks zurück. Bei einer Summe $\gamma_5\gamma_0 + \gamma_1$ beispielsweise wird angenommen, daß alle Summanden zur selben Algebra gehören, und es wird die effektive TypeID des ersten Summanden zurückgegeben, in diesem Fall also die TypeID der Clifford-Algebra statt der eines `add`- oder `mul`-Objektes.
- `noncommutative_composite`: Der Ausdruck ist ein Produkt aus Elementen aus verschiedenen Algebren, z.B. $\gamma_\mu T_a$, oder es kann keine Aussage darüber getroffen werden, zu welcher Algebra der Ausdruck gehört.

2.4.1.2 Die Klasse `ncmul`

Nicht-kommutative Produkte werden mit Hilfe der Klasse `ncmul` gebildet, die die einzelnen Faktoren zunächst unter Berücksichtigung ihrer Reihenfolge in einem Vektor speichert. In der zugehörigen Methode `eval()` wird dann versucht, Beiträge verschiedener nicht-kommutativer Algebren so weit wie möglich zu trennen. Dabei wird implizit angenommen, daß verschiedene effektive TypeIDs verschiedene Algebren repräsentieren und untereinander kommutieren. Im einzelnen werden folgende Vereinfachungen in `eval()` ausgeführt:

- Zuerst wird Assoziativität ausgenutzt, indem alle Unterprodukte (sowohl `ncmul`- als auch `mul`-Objekte) des zu vereinfachenden `ncmul`-Objekts unter Beibehaltung ihrer Reihenfolge aufgelöst werden. Bei der wiederholten Anwendung geht Information über bereits nach Algebren sortierte Faktoren verloren gehen (Abb. 2.4), so daß zukünftige GiNaC-Versionen diese Regel optimieren sollten.
- Ein `ncmul` mit nur einem Faktor wird zu diesem Faktor.
- Ein `ncmul` ohne Faktoren ist 1.
- Kommutative Faktoren (bei denen `return_type==commutative` ist) werden aus dem nicht-kommutativen Produkt herausgezogen und in ein kommutatives Produkt aus diesen Faktoren und ein nicht-kommutatives Produkt aus den verbleibenden Faktoren umgewandelt.
- Ist mindestens einer der Faktoren vom `return_type==noncommutative_composite`, so wird das `ncmul`-Objekt nicht weiter vereinfacht.
- Alle verbliebenen Faktoren, die nun vom `return_type==noncommutative` sind, werden nach ihrem `return_type_tinfo` sortiert und in ein kommutatives Produkt aus nicht-kommutativen Produkten der Beiträge der einzelnen nicht-kommutierenden Algebren umgeformt.
- Wenn ein `ncmul` nur noch Faktoren einer Algebra hat, wird die Algebra-spezifische Funktion `simplify_ncmul()` aufgerufen, die als Argument einen Vektor von Algebra-Elementen hat. Dieser Vektor wird standardmäßig ohne weitere Evaluierung wieder als evaluiertes `ncmul` zurückgegeben (siehe z.B. die Klasse `color`, Abschnitt 2.4.2.1, für eine etwas komplexere Evaluierung). Der Zeitaufwand sollte auch hier $\mathcal{O}(n \log n)$ nicht übersteigen.

Ein Beispiel für die Evaluierung eines nicht-kommutativen Produkts findet sich in Abb. 2.4.

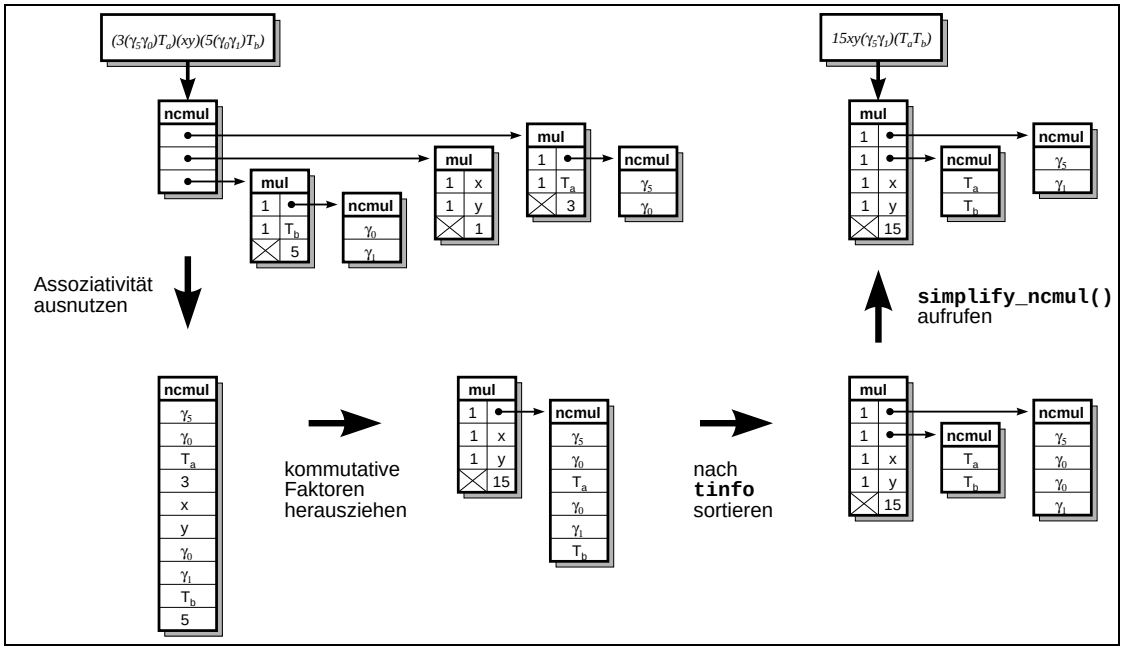


Abbildung 2.4: Vereinfachung nicht-kommutativer Produkte am Beispiel $(3(\gamma_5\gamma_0)T_a)(xy)(5(\gamma_0\gamma_1)T_b) \rightarrow 15xy(\gamma_5\gamma_1)(T_aT_b)$. Die γ_i sollen Elemente der Clifford-Algebra, die T_j Elemente der SU(3)-Algebra sein. γ_i und T_j kommutieren. In `simplify_ncmul()` wird $\gamma_0^2 = \mathbb{1}_\gamma$ ausgenutzt.

Der Operator `*` erzeugt immer kommutative Produkte (`mul`). Um auch nicht-kommutative Produkte bequem schreiben zu können, kann der Operator `%` verwendet werden, der dieselbe Operatorpriorität wie `*` hat und normalerweise für die modulo-Operation steht, die aber auf Ausdrücken nicht sinnvoll definiert werden kann. Für ganzzahlige Potenzen nicht-kommutativer Ausdrücke gibt es die Funktion `ncpower()`. `pow()` bzw. `power()` erwarten eine kommutative Basis.

2.4.2 Indizes und indizierte Objekte

Indizes werden in GiNaC durch die Klasse[5] `idx` und davon abgeleitete Klassen, zur Zeit `coloridx` (Abschnitt 2.4.2.1) und `lorentzidx` (Abschnitt 2.4.2.2), repräsentiert. Obwohl `idx` als Basisklasse für eine Hierarchie von Index-Klassen gedacht ist, ist sie nicht abstrakt und kann direkt für eigene Indizes benutzt werden. Indizes können sowohl einen symbolischen als auch einen numerischen Wert haben. Symbolische Indizes bestehen analog zu Symbolen (Abschnitt 2.3.2) aus einem Namen, der nur zur Ausgabe verwendet wird, und einer internen Seriennummer, die Indizes voneinander unterscheidet. Numerische Indizes sind eigenständige Objekte, gehören also nicht zur Klasse `numeric`. Weiterhin kann man ko- und kontravariante Indizes erzeugen. Standardmäßig werden kontravariante Indizes erzeugt. Die Methode `toggle_covariant()` erzeugt aus einem kontravarianten Index seinen kovarianten Partner (mit demselben Namen und derselben Seriennummer) und umgekehrt.

[5] Der Name `index` konnte wegen einer Namenskollision mit der C-Funktion `index()` nicht verwendet werden.

Die Klasse `idx` selbst kennt keine Summenkonvention. Dies wird den Klassen überlassen, die Indizes benutzen, insbesondere auch die Entscheidung, ob überhaupt zwischen ko- und kontravarianten Indizes unterschieden werden soll.

Die in `basic` definierte Methode `get_indices()` gibt einen Vektor aller Indizes in einem Ausdruck zurück. In einer Summe wird implizit angenommen, daß alle Summanden dieselben Indizes tragen. Die Menge der Indizes eines Produktes ist die Vereinigungsmenge der Indizes der einzelnen Faktoren.

Die Klasse `indexed` dient als Basisklasse für indextragende Objekte. Sie ist von `exprseq` abgeleitet, und alle Argumente des Konstruktors sollten von Typ `idx` oder einer abgeleiteten Klasse sein[6]. `indexed`-Objekte sind standardmäßig nicht-kommutativ und geben als `return_type_tinfo()` ihren eigenen `tinfo_key` zurück.

2.4.2.1 SU(3)-Algebra

GiNaC implementiert eine SU(3)-Algebra, die in *xloops* für die Feynman-Regeln der QCD benötigt wird (Kapitel 3). Daher wird im folgenden synonym von SU(3)- und Farbalgebra gesprochen.

Die QCD-Feynman-Regeln können auf zwei Arten angegeben werden: mit expliziten Farbmatrixelementen oder mit Farbmatrizen (also Elementen der SU(3)-Algebra) und zusätzlichen Regeln. Mit expliziten Matrixelementen lauten die SU(3)-Faktoren der Feynman-Regeln für die beiden Graphen in Abb. 2.5

- für jeden Vertex, an dem ein Quark mit Farbe i (auslaufend), ein Quark mit Farbe j (einlaufend) und ein Gluon mit Index a zusammentreffen, ein $(T_a)_{ij}$,
- für jeden Quark-Propagator mit Indizes i und j an den Enden ein $\delta^{(3)}_{ij}$,
- für jeden Gluon-Propagator mit Indizes a und b an den Enden ein $\delta^{(8)}_{ab}$,
- die auftretenden Matrixelemente $(T_a)_{ij}$ sind skalare Größen, die beliebig kommutieren,
- Drei- und Vier-Gluonen-Vertizes, die Kombinationen der Strukturkonstanten f_{abc} beitragen, werden in den folgenden Beispielen nicht benötigt.

Damit erhält man als Farbanteil zu der amputierten Amplitude für die Streuung eines Quarks mit Farbe j an einem Antiquark mit Farbe i in ein Quark/Antiquark-Paar mit Farben j' und i', Abb. 2.5(links),

$$\begin{aligned} & (T_a)_{il}\delta^{(3)}_{lk}(T_b)_{kj}(T_c)_{i'l'}\delta^{(3)}_{l'k'}(T_d)_{k'j'}\delta^{(8)}_{ac}\delta^{(8)}_{bd} \\ = \; & (T_a)_{ik}(T_b)_{kj}(T_a)_{i'k'}(T_b)_{k'j'} \\ = \; & (T_aT_b)_{ij}(T_aT_b)_{i'j'}\,, \end{aligned} \tag{2.18}$$

und für die Gluon-Selbstenergie, Abb. 2.5(rechts),

$$\begin{aligned} & (T_b)_{ji}\delta^{(3)}_{il}(T_a)_{lk}\delta^{(3)}_{kj} \\ = \; & (T_b)_{jl}(T_a)_{lj} \\ = \; & (T_bT_a)_{jj} \\ = \; & \mathrm{Tr}^{(3)}\,T_bT_a\,. \end{aligned} \tag{2.19}$$

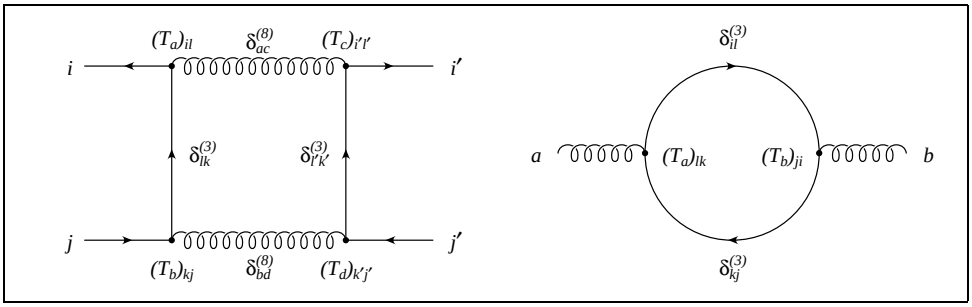


Abbildung 2.5: Beispiele für QCD-Graphen mit Feynman-Regeln, die explizite SU(3)-Matrixelemente benutzen

Quarks und Gluonen sind immer nur in farblosen, also farbgemischten Zuständen beobachtbar. Bei der Berechnung von Feynman-Diagrammen ist daher immer über eventuell auftretende Farbindizes zu summieren. Die explizite Angabe von Farbindizes wird tatsächlich nie benötigt. Eine äquivalente Formulierung der Feynman-Regeln besteht aus diesem Grund in der Verwendung von Farbmatrizen, wobei man zusätzliche Regeln fordert, die die Nichtvertauschbarkeit der Farbmatrizen berücksichtigen:

- für jeden Quark-Quark-Gluon-Vertex ein T_a, wenn das Gluon den Index a trägt,
- der Quark-Propagator hat keine weitere Farbstruktur,
- für jeden Gluon-Propagator mit Indizes a und b an den Enden wie oben ein $\delta^{(8)}_{ab}$,
- die Farbmatrizen müssen entlang der Fermionlinien (entgegen der Pfeilrichtung) aneinander multipliziert werden,
- Beiträge nicht-verbundener Fermionlinien sind zu unterscheiden, da sie zu verschiedenen Repräsentationen der Farbalgebra gehören (T_a, T'_a, ...),
- über geschlossene Fermionlinien ist die Spur $\mathrm{Tr}^{(3)}(\dots)$ zu bilden.
- Ist man trotzdem an der Übergangsamplitude für ein Quark der Farbe i nach j interessiert, projiziert man von der resultierenden Farbmatrix die i-j-Komponente heraus.

Damit erhält man für die beiden Graphen aus Abb. 2.6 die Amplituden $T_aT_bT'_aT'_b$ bzw. $\mathrm{Tr}^{(3)}\, T_bT_a$, und man sieht, daß beide Methoden äquivalent sind.

GiNaC wählt den zweiten Weg, da die Information über zusammenhängende Fermionlinien aus der Datenstruktur für die Topologien (Abschnitt 3.1) genommen werden kann und so die Anzahl der Indizes reduziert wird. Zunächst wird eine von `idx` abgeleitete Klasse `coloridx` eingeführt. Sie fügt zu `idx` keine neue Funktionalität hinzu und dient nur der Wahrung der Typsicherheit. Automatisch generierte Namen haben als Präfix jedoch `color` statt `index`.

Die Klasse `color`, von `indexed` abgeleitet, ist nicht identisch mit der SU(3)-Algebra, sondern repräsentiert alle Objekte, die SU(3)-Indizes tragen, also die Erzeugenden T_a (und

[6] Nur wenn die GiNaC-Bibliothek mit speziellen Compiler-Optionen kompiliert wird, geben Argumente, die nicht vom Typ `idx` sind, einen Laufzeitfehler.

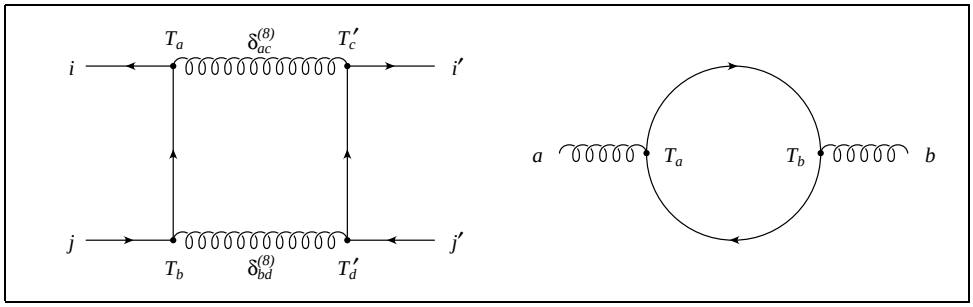


Abbildung 2.6: Beispiele für QCD-Graphen mit Feynman-Regeln, die nur mit SU(3)-Matrizen arbeiten

Einheitsmatrizen $\mathbb{1}$) in unterschiedlichen Repräsentationen (T_a, T'_a, ...), die Strukturkonstanten f_{abd} und d_{abc} und $\delta^{(8)}_{ab}$. `color`-Objekte werden nicht direkt über Konstruktoren erzeugt, sondern über die `friend`-Funktionen `color_T()`, `color_ONE()`, `color_f()`, `color_d()` und `color_h()`, wobei h_{abc} sofort in $d_{abc} + if_{abc}$ zerlegt wird.

Die Methode `eval()` führt folgende Umformungen aus [5]:

- Es wird die Einsteinsche Summenkonvention vorausgesetzt.
- Die Indizes in f_{abc}, d_{abc} und $\delta^{(8)}_{ab}$ werden in eine kanonische Reihenfolge — entsprechend der Ordnungsrelation auf Ausdrücken (Abschnitt 2.2.4) — gebracht. Bei f_{abc} wird die Antisymmetrie in Form eines Vorzeichens bei einer ungeraden Permutation der Indizes und des Verschwindens bei gleichen Indizes berücksichtigt.
- Sind bei $\delta^{(8)}_{ab}$ beide Indizes numerisch, so ist das Ergebnis 0 oder 1. Sind beide symbolisch und identisch, ist das Ergebnis 8.
- $d_{aac} \equiv \sum_a d_{aac} \to 0$.
- d_{abc} mit numerischen a, b und c werden durch ihren Wert ersetzt, z.B. $d_{146} = \frac{1}{2}$.
- f_{abc} mit numerischen a, b und c werden durch ihren Wert ersetzt, z.B. $f_{123} = 1$.

Die Methode `simplify_ncmul()` (Abschnitt 2.4.1.2) vereinfacht zusätzlich:

- Die Kette von `color`-Objekten wird sortiert in die Beiträge von f_{abc}, d_{abc}, $\delta^{(8)}_{ab}$, T_a und $\mathbb{1}$ der einzelnen Repräsentationen, die untereinander kommutieren, sofern alle Elemente eindeutig zugeordnet werden können (z.B. keine Funktion mit unbestimmtem Rückgabewert `color`).
- Überflüssige $\mathbb{1}$ werden entfernt.

Eine `friend`-Funktion `simplify_color()` expandiert einen beliebigen Ausdruck, der nicht nur `color`-Objekte enthalten muß, und wendet Vereinfachungsregeln an, die mindestens $\mathcal{O}(n^2)$ Zeit brauchen, wenn n die Anzahl der `color`-Objekte ist:

- $\delta^{(8)}_{ab}$ werden soweit wie möglich kontrahiert.
- $d_{akl} f_{bkl} = 0$.

- $d_{akl}d_{bkl} = \frac{5}{3}\delta^{(8)}_{ab}$.
- $f_{akl}f_{bkl} = 3\delta^{(8)}_{ab}$.
- $d_{abc}T_bT_c = \frac{5}{6}T_a$.
- $f_{abc}T_bT_c = \frac{3}{2}iT_a$.

Die Funktion `color_trace()` bildet die Spur $\mathrm{Tr}^{(3)}(\dots)$ in einer bestimmten Repräsentation mit der Formel [5]

$$\begin{aligned}\mathrm{Tr}^{(3)}(T_{a_1}\dots T_{a_n}) &= \frac{1}{6}\delta^{(8)}_{a_{n-1}a_n}\,\mathrm{Tr}^{(3)}(T_{a_1}\dots T_{a_{n-2}}) \\ &\quad +\frac{1}{2}h_{a_{n-1}a_nk}\,\mathrm{Tr}^{(3)}(T_{a_1}\dots T_{a_{n-2}}T_k)\end{aligned} \tag{2.20}$$

2.4.2.2 Lorentzvektoren

Zum gegenwärtigen Zeitpunkt bietet GiNaC nur eine rudimentäre Unterstützung für die Verarbeitung von Lorentzvektoren an, die für die Anwendung der Parallel-/Orthogonalraum-Reduzierung (Abschnitt 3.2) ausreicht. Eine weitergehende Funktionalität ist zur Zeit in Entwicklung (Abschnitt 2.6).

Die Klasse `simp_lor` kennt nur einfache, kommutierende Lorentzvektoren und den metrischen Tensor $g^{\mu\nu}$ und unterscheidet zwischen ko- und kontravarianten Indizes. Die begleitende Klasse `lorentzidx` hingegen ist allgemein genug, um auch für die zukünftige Klasse `lortensor` als Indizes zu dienen. Sie wurde in Hinblick auf Anwendungen in *χloops* mit Parallel-/Orthogonalraumzerlegung in dimensionaler Regularisierung entworfen, ist aber nicht darauf beschränkt. Symbolische Lorentzindizes können daher über alle Raum-Zeit-Dimensionen oder nur über die des Orthogonalraums laufen. Bei Orthogonalraum-Indizes muß man die Dimension des Parallelraums mit angeben, während die Festlegung der Gesamt-Dimension der Raum-Zeit Aufgabe der Klasse ist, die die Lorentzindizes benutzt (zur Zeit also `simp_lor`). Dadurch kann `lorentzidx` auch in Anwendungen verwendet werden, die eine vierdimensionale Raum-Zeit erwarten.

Lorentzvektoren (genau genommen eine Komponente eines Lorentzvektors) werden mit der `friend`-Funktion `lor_vec()` aus einem Namen und einem Lorentzindex, der symbolisch oder numerisch sein kann, erzeugt. Im Gegensatz zu Symbolen und Indizes werden Lorentzvektoren anhand ihres Namens unterschieden, nicht durch eine Seriennummer.

Der metrische Tensor (bzw. eine Komponente davon) wird mit der Funktion `lor_g()` aus zwei Lorentzindizes erzeugt. In der Methode `eval()` werden die Indizes in eine kanonische Reihenfolge sortiert und im Falle zweier numerischer Indizes durch den Wert des metrischen Tensors ersetzt,

$$g^{00} = 1\,,\quad g^{11} = g^{22} = \dots = -1\,,\quad g^{\mu\nu} = 0\,,\quad \text{falls}\quad \mu \neq \nu\,. \tag{2.21}$$

Bei einem ko-/kontravarianten Paar von symbolischen Indizes wird die Summe ausgeführt,

$$g^{\mu}{}_{\mu} = D\,,\quad g^{\mu_\perp}{}_{\mu_\perp} = D_\perp\,. \tag{2.22}$$

Die Dimension der Raum-Zeit D erhält man aus der Funktion `Dim()`, die ein klasseninternes Symbol zurückgibt. $D_\perp$ ist `Dim()` minus der Parallelraum-Dimension, die im Orthogonalraum-Index $\mu_\perp$ enthalten ist.

Die `friend`-Funktion `simplify_simp_lor()` expandiert einen Ausdruck und kontrahiert $g^{\mu\nu}$ sowie Paare von Lorentzvektoren. Diese muß man einem Objekt der Klasse `scalar_products` bekannt machen, das dann als Parameter an `simplify_simp_lor()` übergeben wird.

2.4.3 Matrizen

GiNaC stellt eine Klasse `matrix` zur Darstellung von $n \times m$-Matrizen zur Verfügung. Matrizen können addiert, subtrahiert und multipliziert werden. Es gibt Methoden, um Matrizen zu invertieren oder zu transponieren, sowie die Determinante, die Spur oder das charakteristische Polynom zu berechnen.

Die zur Zeit wichtigste Methode ist jedoch `fraction_free_elim()`, mit dem sowohl über- als auch unterbestimmte lineare Gleichungssysteme gelöst werden können. Sie benutzt die sogenannte *fraction-free* Gauss-Eliminationsmethode [34], die beim Umformen der Matrix auf Echelon-Form Divisionen vermeidet. Das Gleichungssystem in Matrixform wird mit der Funktion `lsolve()` aus einer Liste von Relationen (Abschnitt 2.3.6) gewonnen.

2.4.4 Strukturen

Einer der größten Nachteile von Maple oder Mathematica ist der Mangel an strukturierten Datentypen. In GiNaC kann man zwar auf C-Strukturen und C++-Klassen zurückgreifen, diese Datentypen sind jedoch nicht geeignet, um als Argumente an GiNaC-Funktionen (Abschnitt 2.3.5) übergeben zu werden, da alle GiNaC-Objekte von der Basisklasse `basic` abgeleitet werden müssen. Dafür gibt es in GiNaC ein Perl-Script `structure.pl`, das mit einem einfachen Parser eine normale C-Struktur in eine GiNaC-Klasse umwandelt, die von der Struktur-Basisklasse `structure` abgeleitet ist. `structure` verwaltet die TypeIDs für die automatisch erzeugten Klassen. Beispielsweise läßt sich damit eine Klasse `momenta3pt` mit Komponenten `q10`, `q20` und `q21` erzeugen, die die für einen Dreipunkt-Feynman-Graphen notwendigen Impulskomponenten enthält. Auf die Komponenten kann man dann mit `mom.q10()` etc. (nur lesend) zugreifen, wenn `mom` eine Variable vom Typ `momenta3pt` ist.

2.5 Hilfsprogramme

Einige Hilfsprogramme sollen die Einarbeitung in GiNaC und die Konvertierung bestehender Programme erleichtern.

2.5.1 Maple-GiNaC-Konverter

Maple-Programme haben eine ähnliche Syntax für Ausdrücke und Kontrollstrukturen wie C++-Programme. Dies legt es nahe, daß man einfache Maple-Programme automatisch in C++-Programme konvertieren kann. Allerdings ist die Anwendung eines solchen Konverters nur in Grenzen möglich.

Die Konvertierung besteht im wesentlichen aus den beiden Schritten

- Analyse der syntaktischen Elemente eines Maple-Programms mit lex und yacc,
- Synthese eines C++-Programms aus den analysierten Daten.

In Anhang B.9 ist dies detaillierter aufgeführt.

Einige in Maple benutzbare Sprachelemente haben kein Äquivalent in C++, beispielsweise die Verkettung von Variablennamen mit dem Punkt-Operator. Ein konvertiertes Programm braucht außerdem in fast allen Fällen eine Nachbehandlung. Viele Konstrukte lassen sich aber ohnehin in C++ eleganter als in Maple formulieren.

2.5.2 GiNaC interaktiv

Der höheren Ausführungsgeschwindigkeit kompilierter Sprachen wie C++ im Vergleich mit interpretierten Sprachen wie BASIC, LOGO oder CAS wie Maple und Mathematica steht der Nachteil gegenüber, daß man kleine Programme nicht schnell testen kann, sondern immer den Zyklus „Editieren, Kompilieren, Ausführen“ durchlaufen muß. Interpretierte Sprachen erlauben in der Regel das interaktive Ausführen einzelner Kommandos mit direkter Kontrolle der Ausgabe.

Mit der GiNaC-Shell ginsh kann man einfache Ausdrücke auf einer Kommandozeile eingeben und auswerten lassen und das Ergebnis Symbolen zuweisen. Die eingegebene Zeile wird mit flex und bison zerlegt, analysiert und ausgeführt. Man kann Ausdrücke expandieren, sammeln, differenzieren, Laurent-Reihen bilden und Funktionen aufrufen, die auch aus anderen Bibliotheken (z.B. *xloops*) stammen können. Der Zugriff auf die meisten der erweiterten GiNaC-Klassen aus Abschnitt 2.4 ist jedoch nicht möglich, da als terminale Objekte nur Zahlen und Symbole unterstützt werden und so ein Arbeiten mit Indizes nicht möglich ist. Auch können keine Kontrollstrukturen wie Schleifen oder Bedingungen benutzt werden.

Mit Hilfe einer speziell angepaßten Version des C++-Interpreters CINT [35] kann man diese Beschränkungen jedoch umgehen. CINT ist Teil des ROOT-Projektes [10] am CERN, in dem Datenerfassung und -analyse der NA49- und LHC-Experimente programmiert werden (und das daher noch mindestens 15 Jahre gepflegt wird), und erlaubt das interaktive Ausführen von ANSI (mit kleinen Einschränkungen) C- und C++-Programmen. Ein Zusatzprogramm erlaubt es, beliebige C++-Klassenbibliotheken wie z.B. GiNaC mit CINT zu einem eigenständigen Programm zu kapseln. Die aktuelle CINT-Version (5.14.26, Dezember 1999) hat noch Probleme mit *Namespaces*, Überladen von Funktionen, expliziten Konstruktoren und einigen STL-Komponenten, die GiNaC vermeiden kann und nach Aussage des Autors in kommenden CINT-Versionen behoben sein werden.

Durch Zugriff auf undokumentierte CINT-Funktionen ist es sogar möglich, eine eigene Oberfläche GiNaC-cint zu schreiben, die Eingaben interaktiv von der Tastatur entgegennimmt, an CINT zur Ausführung durchreicht und den Rückgabewert (der letzten Anweisung, wenn mehrere durch Semikolon getrennte eingegeben wurden) auf dem Bildschirm ausgibt und zur weiteren Verwendung in den fortlaufenden Variablen `Out`*n* und `LAST`, `LLAST` und `LLLAST` für das jeweils letzte, vorletzte und vorvorletzte Ergebnis speichert, falls er vom Typ `ex` ist. Eine kurze Beispielsitzung mit GiNaC-cint findet sich in Anhang B.10.

2.6 Zukünftige Erweiterungen

Die Entwicklung von GiNaC ist noch nicht abgeschlossen, und da es sich um ein offenes System handelt, das geeignet ist, in eine Vielzahl von Programmen — auch außerhalb der Hochenergiephysik — eingebettet zu werden, ist ein Ende auch noch nicht abzusehen. Der für die automatisierte Berechnung von Feynman-Graphen benötigte Teil (Kapitel 3) ist

jedoch bis auf zwei Klassen, die im folgenden beschrieben werden, vollständig implementiert.

2.6.1 Lorentztensoren

Die Klasse `simp_lor` (Abschnitt 2.4.2.2) ist nicht ausreichend, um alle Anforderungen, die *χloops* stellt, zu erfüllen. Es fehlen folgende Eigenschaften:

- höhere Tensoren, z.B. um die Photon-Selbstenergie $\Pi^{\mu\nu}(q^2)$ zu repräsentieren,
- der total antisymmetrische Tensor $\epsilon^{\mu\nu\rho\sigma}$,
- Lorentzvektoren mit expliziten Parallelraum-Komponenten, z.B. ${q_2}^\mu = (q_{20}, q_{21}, \vec{0})$ oder $l^\mu = (l_0, l_1, \vec{l_\perp})$,
- Skalarprodukte $p \cdot q = p_\mu q^\mu$,
- Algebren mit Lorentzindizes (z.B. Clifford-Algebra) und nicht-kommutative Skalarprodukte (z.B. $\not{p} = p_\mu \gamma^\mu$).

Die in Entwicklung befindliche Klasse `lortensor` (von `indexed` abgeleitet) ist eine Weiterentwicklung von `simp_lor`, die Lorentztensoren beliebig hohen Ranges (genau genommen Komponenten dieser Tensoren) repräsentiert, mit den Spezialfällen $g^{\mu\nu}$ und $\epsilon^{\mu\nu\rho\sigma}$. Lorentztensoren werden, wie Symbole und Indizes, anhand einer Seriennummer unterschieden werden und einen Namen für die Ausgabe haben. Eine davon abgeleitete Klasse `lorvector` wird kovariante Vektoren und Vektoren im Parallel-/Orthogonalraum darstellen. Parallelraum-Komponenten werden sofort ersetzt (${q_2}^\mu|_{\mu=0} \rightarrow q_{20}$), Orthogonalraum-Komponenten werden am Ende durch Kombinationen des metrischen Tensors im Orthogonalraum ($g^{\mu_\perp \nu_\perp}$) ersetzt.

2.6.2 Clifford-Algebra

Für die Clifford-Algebra gelten ähnliche Überlegungen wie für die SU(3)-Algebra (Abschnitt 2.4.2.1). Man kann mit kommutierenden Matrixelementen $(\gamma^\mu)_{ij}$ (wobei i und j über die vier Spinorkomponenten laufen) oder mit nicht-kommutierenden Clifford-Algebra-Elementen γ^μ und zusätzlichen Regeln (Vertizes und Propagatoren entlang der Fermionlinien einsetzen, Spurbildung über geschlossene Fermionlinien) rechnen. Die Gründe, die bei der SU(3)-Algebra für die zweite Möglichkeit gesprochen haben, gelten entsprechend auch für die Clifford-Algebra. Die Klasse `clifford`, von `lortensor` abgeleitet, stellt allgemeine γ^μ-Objekte sowie Parallel-/Orthogonalraum-Objekte γ^0, γ^1, ... , $\gamma^{\mu_\perp}$ und die speziellen Objekte $\mathbb{1}$ und γ_5 dar, jeweils in verschiedenen Repräsentationen zu verschiedenen Fermionlinien. Die Methode `simplify_ncmul()` der `clifford`-Klasse wird, wenn alle Clifford-Elemente γ^μ durch Parallel-/Orthogonalraum-Komponenten ausgedrückt wurden, die Antikommutativität der Clifford-Algebra

$$\begin{aligned} \{\gamma^\mu, \gamma^\nu\} &= 2g^{\mu\nu}\mathbb{1} \\ \{\gamma_5, \gamma^\mu\} &= 0 \end{aligned} \tag{2.23}$$

ausnutzen und den Ausdruck auf die Form

$$(\gamma_5)^{n_5}(\gamma^0)^{n_0}(\gamma^1)^{n_1}\ldots\gamma^{\mu_{\perp 1}}\ldots\gamma^{\mu_{\perp n}} \tag{2.24}$$

bringen, wobei n_5, n_0, n_1, ... $\in \{0, 1\}$ sind.

2.6.3 Sonstiges

Eine Reihe von Erweiterung ist geplant oder wünschenswert:

- Ein-/Ausgabe von Ausdrücken in Dateien, bei dem das *Reference Counting* erhalten bleibt. Dies ist beispielsweise sinnvoll, um häufig benötigte Einloop-Funktionen zu speichern.
- Ausgabe von Ausdrücken im LaTeX-Format.
- Thread-Sicherheit der GiNaC-Bibliothek, d.h. die Fähigkeit, GiNaC parallel auf Mehrprozessorrechnern zu benutzen. Dies wird zur Zeit vor allem durch die mangelnde Thread-Sicherheit der CLN-Bibliothek verhindert.
- Multithreading wäre auch notwendig, um GiNaC innerhalb von Funktionen zu benutzen, die mit PVEGAS [48] numerisch integriert werden sollen.
- Ein „inertes“ (nicht evaluiertes) `diff()`, z.B. `f(x).diff(x) = Diff(f(x),x)`, wenn `f(x)` keine Methode `diff()` registriert hat.
- Spezielle Funktionen wie Bessel, elliptische Integrale, höhere Polylogarithmen, hypergeometrische Funktionen etc.
- Arithmetik in endlichen Körpern ($\mathbb{Z}/p\mathbb{Z}$).

GiNaC fehlen noch viele Eigenschaften, die andere CAS besitzen, aber nicht für *xloops* benötigt werden. Für einen direkten Vergleich mit Mathematica, Maple, Reduce etc. wie in [91] fehlen u.a.

- Faktorisierung von ganzen Zahlen und Ausdrücken,
- weitergehende Vereinfachung von Ausdrücken,
- Grenzwertberechnung,
- Annahmen über Variablen und logische Folgerungen (z.B. $x \geq y,\ y \geq z,\ z \geq x \Rightarrow x = z$),
- symbolische Integration.

Diese Forderungen werden aber selbst von diesen kommerziellen CAS nicht immer korrekt erfüllt.

Kapitel 3

Automatisierte Berechnung von Feynman-Graphen

Eines der Probleme, auf das man beim Berechnen von Prozessen auf Zweiloop-Niveau stößt, ist die große Anzahl von beitragenden Feynman-Graphen. Durch die vielen Teilchen im Standardmodell werden beispielsweise in der Äquivalenzmodell-Näherung [59, 88] für den Zerfall $H \to Z\,Z$ 157 Einteilchen-irreduzible Graphen [32], im vollen Standardmodell aber bereits in der unitären Eichung 1852 Graphen benötigt. Solche Berechnungen können nur noch mit Computer-Unterstützung ausgeführt werden. Dabei besteht die Schwierigkeit darin, allgemeine und eindeutige Algorithmen zu finden. Dieses Kapitel stellt einige Komponenten vor, die für ein Programmpaket zur automatisierten Berechnung von Feynman-Graphen benötigt werden.

3.1 Topologie-Erzeugung

Es existieren bereits Programmpakete, die zu einem gegebenen Modell alle beitragenden Feynman-Graphen erzeugen, z.B. QGRAF [68] oder grace [45]. Für eine Berechnung der Graphen ist die Ausgabe dieser Programme aber nur bedingt geeignet, da es sich nur um eine Aneinanderreihung von Propagatoren und Vertizes handelt. Diese enthält zwar alle nötige Information (Impulse, Teilchen), um daraus von Hand den vollständigen Graphen zu rekonstruieren, aber die Graphen werden nicht notwendigerweise in einer erkennbaren Reihenfolge mit festen Impulskonventionen erzeugt. QGRAF selbst bietet auch gar keine Möglichkeit zum Berechnen von Graphen, grace beschränkt sich auf Baum- und Einloop-Graphen, wo die Anzahl der Basistopologien klein ist (drei Einteilchen-irreduzible bei Einloop-Dreipunkt). Im folgenden werden immer nur Einteilchen-irreduzible Topologien betrachtet.

Auf Zweiloop-Niveau hingegen gibt es bei Dreipunkt 23 und bei Vierpunkt schon 54 Basistopologien (Abschnitt 3.1.7). Für eine automatisierte Berechnung von Prozessen ist es daher unerläßlich, alle Topologien selbst entsprechend der gewünschten Konventionen zu erzeugen und zu klassifizieren, da die Topologie die Propagatorstruktur des zugehörigen skalaren Graphen bestimmt und diese wiederum die Berechnungsmethode (Kapitel 1) festlegt.

Die hier vorgestellte Methode nutzt aus, daß man alle Einloop-n-Punkt-Graphen erhält, indem man n äußere Beine mit den Impulsen $q_1 \dots q_n$ auf *alle* möglichen Arten an einen Vakuum-Loop anheftet. Dabei ist Impulserhaltung ($\sum_i q_i = 0$) zu beachten. Entspre-

chend lassen sich alle Zweiloop-n-Punkt-Graphen erzeugen, indem man n Beine an die beiden grundlegenden Vakuum-Topologien in Abb. 3.1, die faktorisierende und die nicht-faktorisierende Vakuum-Topologie, anheftet.

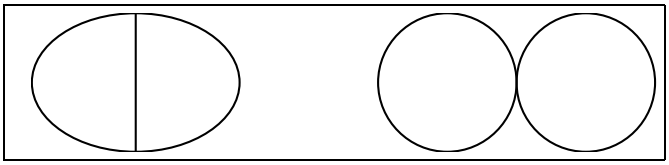


Abbildung 3.1: Die beiden Zweiloop-Vakuum-Topologien, die nicht-faktorisierende (links) und die faktorisierende (rechts)

Dabei werden offensichtlich zunächst einige Topologien erzeugt, die wegen Symmetrien in den Vakuum-Topologien identisch mit anderen sind. So sind z.B. die Dreipunkt-Graphen in Abb. 3.2a und b identisch, während c verschieden von a ist, da sie durch die Benennung der äußeren Impulse $q_1 \leftrightarrow q_2$ unterscheidbar sind (der Impuls, der am Vierervertex einfließt, ist ausgezeichnet).

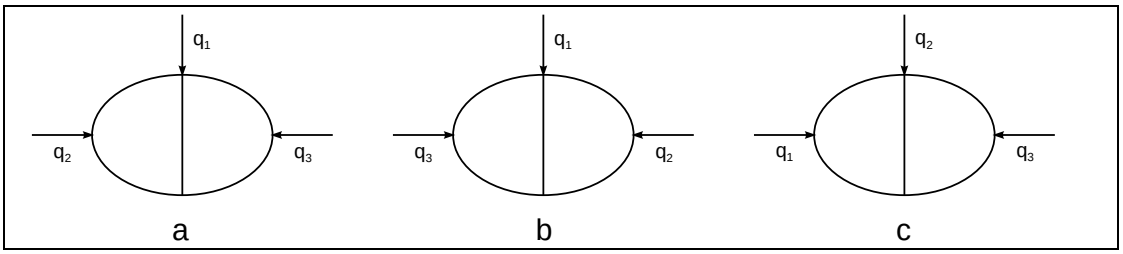


Abbildung 3.2: Drei Topologien, die durch Anheften von jeweils drei Beinen an die nicht-faktorisierende Vakuum-Topologie entstehen

Im folgenden wird zunächst die Erzeugung der nicht-faktorisierenden Topologien, ausgehend von Abb. 3.1(links), betrachtet. Die (einfacheren) faktorisierenden werden im Anschluß behandelt.

3.1.1 Nicht-faktorisierende Topologien

In jedem nicht-faktorisierenden Zweiloop-Graphen sind zwei Vertizes ausgezeichnet, an denen drei innere Propagatoren aufeinandertreffen. Diese werden im folgenden als *innere* Vertizes, symbolisch mit a und b, bezeichnet. Sie haben drei innere Beine (vier bei der faktorisierenden Vakuum-Topologie) und können auch äußere Beine haben, müssen dies aber nicht. Weiter werden potentielle *äußere* Vertizes eingeführt, die immer nur zwei innere Beine und mindestens ein äußeres Bein haben und an denen die äußeren Impulse einlaufen. Beim Erzeugen von n-Punkt-Graphen benötigt man höchstens n dieser Vertizes in jedem der drei Zweige, die die Vertizes a und b verbinden. Diese werden mit $c_1 \dots c_n$ im linken Zweig (Zweig 1), $d_1 \dots d_n$ im rechten (Zweig 2) und $e_n \dots e_1$ im mittleren (Zweig 3) bezeichnet, jeweils gerichtet a mit b verbindend (Abb. 3.3).

Nun bringt man auf alle möglichen Arten n äußere Impulse $q_1 \dots q_n$ an den Vertizes $v_1 \dots v_n$ an, wobei $v_i \in \{a, b, c_1, \dots, c_n, d_1, \dots, d_n, e_1, \dots, e_n\}$ ist. Dabei ist zu berücksichtigen, daß auch $v_i = v_j$ sein kann, also mehr als ein Impuls am selben Vertex angreifen kann,

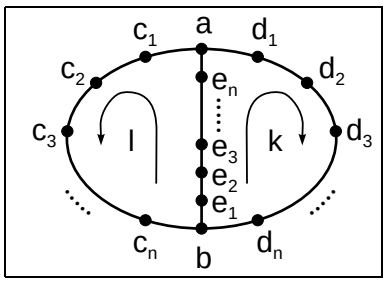


Abbildung 3.3: Benennung der Vertizes und Impulsfluß bei der nichtfaktorisierenden Vakuum-Topologie

wodurch sich die effektive Anzahl der Beine verringert, vgl. Abschnitt 3.1.5. Während es in renormierbaren Eichtheorien nur Vertizes mit drei und vier Beinen gibt, werden z.B. in der chiralen Störungstheorie bei genügend hoher Ordnung auch Vertizes mit beliebig vielen Beinen benötigt [75]. Außerdem treten Graphen mit Vertizes mit mehr als vier Beinen als künstliche Abzugsterme und beim Kürzen von Propagatoren auf (Abschnitt 1.5). Die bisherige *xloops* Version konnte nur Graphen mit Vertizes mit maximal vier Beinen erzeugen.

Man hat die Freiheit, Weg und Orientierung der beiden Loopimpulse l und k beliebig festzulegen sowie diese wegen der Translationsinvarianz der Loopintegration [14] um einen konstanten Wert zu verschieben. Folgende Konventionen legen den Impulsfluß im Graphen eindeutig fest und machen gleichzeitig die Impulse der einzelnen Graphen vergleichbar:

- Weg und Orientierung der beiden Loopimpulse l und k im Vakuum-Graphen werden so festgelegt, daß $l + k$ im mittleren Zweig „nach oben“, l zurück entlang des linken und k entlang des rechten Zweiges fließt (Abb. 3.3).

- Der Propagator $P_{e_n,a}$, der die beiden Vertizes e_n und a verbindet, wird auf den Impuls $l + k$ normiert.

- Alle am Vertex a einlaufenden Impulse fließen mit l entlang des linken Zweiges weiter.

- Alle am Vertex b einlaufenden Impulse fließen mit $l+k$ entlang des mittleren Zweiges.

- Alle anderen einlaufenden Impulse fließen weiter in Richtung des Vakuum-Loopimpulses im entsprechenden Zweig (l, k bzw. $l + k$).

- Alle einlaufenden Impulse werden letztendlich im $l + k$-Zweig gesammelt und laufen dort „nach oben“. Spätestens bei e_n sind alle n Impulse in den Graphen eingeflossen, so daß dieser Propagator den Impuls $l + k + q_1 + \cdots + q_n$ trägt. Dies ist nach Impulserhaltung aber $l + k$, in Übereinstimmung mit der zweiten Regel.

- Äußere Vertizes, an denen kein äußerer Impuls angeheftet wurde, werden ignoriert. Die temporäre Einführung überzähliger Vertizes erleichtert die Implementierung des Verfahrens.

Damit erhält man eine Menge $\Gamma = \{P_{v_i,v_j}(p_k)\}$ von gerichteten Propagatoren mit ihrem zugehörigen Impuls, die Paare von Vertizes verbinden. Γ beschreibt die Topologie

des Graphen vollständig. Im Beispiel von Abb. 3.4a mit $(v_1, v_2, v_3) = (a, c_1, b)$ erhält man

$$\Gamma = \{P_{a,c_1}(l + q_1), P_{c_1,b}(l + q_1 + q_2), P_{a,b}(k), P_{b,a}(l + k)\}\,. \tag{3.1}$$

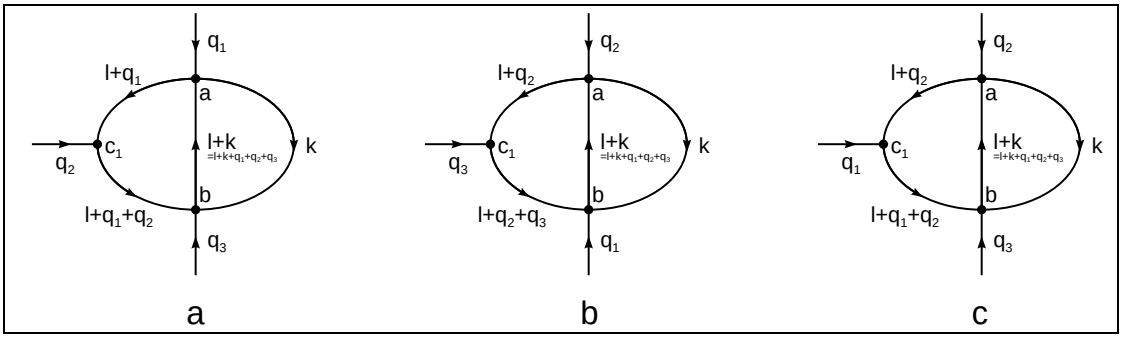


Abbildung 3.4: Impulsfluß für die Vertexkombinationen a: $(v_1, v_2, v_3) = (a, c_1, b)$, b: (b, a, c_1), c: (c_1, a, b)

Alle möglichen Vertexkombinationen, an die äußere Beine angehängt werden können, erhält man aus

$$(v_1 \dots v_n) \in [a, b, c_1, \dots, c_n, d_1, \dots, d_n, e_1, \dots, e_n]^n\,. \tag{3.2}$$

Damit erzeugt man aber zunächst identische Topologien mehrfach. Sei $E(v)$ die Anzahl der äußeren Impulse am Vertex v. Sei $x_1 \dots x_n$ einer der drei Zweige $c_1 \dots c_n$, $d_1 \dots d_n$ bzw. $e_1 \dots e_n$. Dann können alle Graphen, die die Bedingung

$$E(x_{i+1}) = m > 0 \wedge E(x_i) = 0, \quad i \in 1, \dots, n-1 \tag{3.3}$$

erfüllen, ignoriert werden, da sie dieselbe Menge von Propagatoren erzeugen wie die Kombination

$$E(x_{i+1}) = 0 \wedge E(x_i) = m\,. \tag{3.4}$$

Diese Ausschlußbedingung läßt sich auch formulieren als

$$\max\{i \,|\, E(x_i) > 0\} < \min\{i \,|\, E(x_i) = 0\} \tag{3.5}$$

Desweiteren lassen sich die Symmetrien des Vakuumgraphen ausnutzen, zum einen die Spiegelsymmetrie an der Horizontalen, zum anderen der Austausch der drei Zweige $1 \leftrightarrow 2$, $1 \leftrightarrow 3$ und $2 \leftrightarrow 3$. Damit können Graphen mit

$$E(a) > E(b) \quad \text{(Spiegelsymmetrie)} \tag{3.6}$$

sowie

$$\begin{aligned} |\{i \,|\, E(c_i) > 0\}| &< |\{i \,|\, E(d_i) > 0\}| \quad (1 \leftrightarrow 2) \\ |\{i \,|\, E(c_i) > 0\}| &< |\{i \,|\, E(e_i) > 0\}| \quad (1 \leftrightarrow 3) \\ |\{i \,|\, E(d_i) > 0\}| &< |\{i \,|\, E(e_i) > 0\}| \quad (2 \leftrightarrow 3) \end{aligned} \tag{3.7}$$

entfallen, wo $|\{\dots\}|$ die Anzahl der Elemente einer Menge bezeichnet. Falls $|\{i \,|\, E(x_i) > 0\}| = |\{i \,|\, E(y_i) > 0\}|$ ist, läßt sich noch ein lexikographisches Ausschlußkriterium formulieren:

$$\begin{aligned} &\exists i \geq 1 : \forall j < i : E(c_j) = E(d_j) \wedge E(c_i) < E(d_i) \\ &\exists i \geq 1 : \forall j < i : E(c_j) = E(e_j) \wedge E(c_i) < E(e_i) \\ &\exists i \geq 1 : \forall j < i : E(d_j) = E(e_j) \wedge E(d_i) < E(e_i) \end{aligned} \tag{3.8}$$

Diese Strategie zur Erzeugung der nicht-faktorisierenden Zweiloop-Topologien hat folgende Eigenschaften:

- Es werden möglichst wenig Propagatoren erzeugt, die l und k gemeinsam enthalten. Das ist wünschenswert, da in der Regel Graphen mit gemischten Propagatoren in der Berechnung schwieriger sind [30] (sie werden aber beispielsweise in Abschnitt 1.3.2 und 1.5 benötigt).
- Wenn l und k gemeinsam vorkommen, dann immer in der Kombination $l + k$.
- Fast alle Topologien haben nur einen $l + k$-Propagator, der auch frei von äußeren Impulsen q_i ist. Es gibt nur eine Dreipunkt- und zwei Vierpunkt-Topologien, die zwei $l + k$-Propagatoren haben, von denen einer frei von äußeren Impulsen ist.
- Wenn die Topologie einen Einloop-Zweipunkt-Untergraphen hat, dann haben dessen zwei Propagatoren immer die Impulse $l + k$ und k.
- Alle Topologien in Abb. 1.27, auch die gedrehten Varianten, werden abweichend von Abschnitt 1.5 so erzeugt, daß $n_l^{(0)} = 1$ (aus Gl. (1.118)) und $n_k^{(0)} = 1$ (aus Gl. (1.121)) ist.
- Die Propagatoren werden für jeden Zweig getrennt, innerhalb des Zweiges aber in ihrer logischen Reihenfolge, gesammelt. Dadurch läßt sich später beim Einsetzen der Feynman-Regeln feststellen, wo zusammenhängende Fermionlinien bzw. -loops sind.

Bei allen Kombinationen $(v_1, \dots, v_n)$, die nicht von vornherein durch diese Kriterien augeschlossen werden, muß nun geprüft werden, ob der dadurch bestimmte Graph Γ durch eine lineare Transformation der Loopimpulse l und k in einen der bereits als verschieden erkannten Graphen $\{\Gamma_i\}$ überführt werden kann. Zwei Graphen Γ und Γ' lassen sich nach folgendem Algorithmus vergleichen:

- Vergleiche die Anzahl N_P der Propagatoren bzw. die Anzahl N_V der Vertizes der beiden Graphen. N_V setzt sich zusammen aus den beiden inneren Vertizes a und b sowie $n - 2 \leq N_{V_e} = N_V - 2 \leq n$ äußeren Vertizes $x_1, \dots, x_{N_{V_e}}$.
- Vergleiche die charakteristischen Polynome der beiden Graphen. Das charakteristische Polynom eines ungerichteten Graphen ist das charakteristische Polynom $\det(A - \lambda \mathbb{1})$ seiner Adjazenzmatrix [38]. Die (symmetrische) Adjazenzmatrix A ist eine $N_V \times N_V$ Matrix, wo A_{ij} die Anzahl der Propagatoren ist, die den Vertex i mit dem Vertex j verbinden. Ein Propagator von i nach i zählt nur einfach. Gleichheit der charakteristischen Polynome ist eine notwendige, aber nicht hinreichende Bedingung für Identität zwischen Graphen, da Information über die Richtung (bzw. allgemeiner bei Feynman-Graphen die Impulse) nicht eingeht.

Dieser Schritt ist nicht zwingend notwendig, verbessert aber die Ausführungsgeschwindigkeit des Verfahrens, da er ein schnelles Entscheidungskriterium für topologisch verschiedene Graphen darstellt.

- Betrachte alle Permutationen von Vertizes

$$\sigma_V \in S(\{a,b\}) \times S(\{x_1,\ldots,x_{N_{V_e}}\}) \tag{3.9}$$

(da die Vertizes a und b ausgezeichnet sind) und überprüfe, ob es jeweils passende Paare von Propagatoren mit gleichem bzw. vertauschtem Anfangs- und Endvertex gibt. Da die Reihenfolge der Propagatoren in Γ bzw. Γ' voneinander abweichen kann, muß man auch alle Permutationen

$$\sigma_P \in S(\{1,\ldots,N_P\}) \tag{3.10}$$

betrachten. Für vertauschte Anfangs- und Endvertizes merkt man sich ein Vorzeichen

$$\pi \in \{-1,1\}^{N_P}\,. \tag{3.11}$$

- Gibt es ein solches Tripel (σ_V,σ_P,π) von Vertex- und Propagatorpermutationen sowie Propagatorrichtungen, muß nun festgestellt werden, ob es eine Impulstransformation gibt, die die Impulse an den einzelnen Propagatoren exakt ineinander überführt. Das führt zu einem linearen $N_P \times N_P$ Gleichungssystem für die Loopimpulse l und k, das immer überbestimmt ist, da $N_P > 2$:

$$p_i(l',k') = \pi(i)\, p_{\sigma_P(i)}(l,k)\,, \quad i = 1,\ldots,N_P\,. \tag{3.12}$$

Dabei ist p_i der Impuls des i-ten Propagators. Beim Lösen des Systems muß man noch die Impulserhaltung $q_n = -\sum_{i=1}^{n-1} q_i$ berücksichtigen (wegen Abschnitt 3.1.2 ist es vorteilhaft, q_n nicht schon zu Beginn durch die negative Summe der übrigen Impulse ersetzt werden).

- Falls das Gleichungssystem lösbar ist, ist die Impulstransformation von der Form

$$\begin{aligned} k &= \alpha k' + \beta l' + q_x \\ l &= \gamma k' + \delta l' + q_y \\ &\quad \alpha,\beta,\gamma,\delta \in \{-1,0,1\} \\ &\quad q_x, q_y \ \text{ eine Linearkombination der } \ q_1,\ldots,q_n\,. \end{aligned} \tag{3.13}$$

Als Beispiel dient der Vergleich der beiden Graphen Γ aus Gl. (3.1), Abb. 3.4a und der Graph aus Abb. 3.4b mit $(v_1',v_2',v_3') = (b,a,c_1)$,

$$\Gamma' = \{P_{a,c_1}(l+q_3), P_{c_1,b}(l+q_2+q_3), P_{a,b}(k), P_{b,a}(l+k)\}\,. \tag{3.14}$$

Beide Graphen haben dieselbe Anzahl von Propagatoren $N_P = 4$ und Vertizes $N_V = 3$. Die Adjazenzmatrix ist in beiden Fällen

$$A = \begin{pmatrix} 0 & 2 & 1 \\ 2 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}\,, \tag{3.15}$$

und damit ist auch das charakteristische Polynom $\lambda^3 - 6\lambda - 4$ gleich.

Die erste Permutation der Vertizes, die Identität $\sigma_V = (a, b, c_1)$, führt mit der Identität $\sigma_P = (1, 2, 3, 4)$ und $\pi = (1, 1, 1, 1)$ auf das Gleichungssystem

$$\begin{aligned} l' + q_1 &= l + q_3 \\ l' + q_1 + q_2 &= l + q_2 + q_3 \\ k' &= k \\ l' + k' &= l + k\,, \end{aligned} \tag{3.16}$$

das keine Lösung für l und k hat.

Die zweite Vertexpermutation $\sigma_V = (b, a, c_1)$ mit der Propagatorpermutation $\sigma_P = (2, 1, 3, 4)$ und $\pi = (-1, -1, -, 1, -1)$ ergibt das Gleichungssystem

$$\begin{aligned} l' + q_1 &= -(l + q_2 + q_3) \\ l' + q_1 + q_2 &= -(l + q_3) \\ k' &= -k \\ l' + k' &= -(l + k) \end{aligned} \tag{3.17}$$

mit der Lösung

$$\begin{aligned} k &= -k' \\ l &= -l'\,, \end{aligned} \tag{3.18}$$

unter Berücksichtigung von $q_3 = -q_1 - q_2$.

Damit ist gezeigt, daß Γ und Γ' dieselbe Topologie beschreiben, d.h. die Vertexkombination (b, a, c_1) erzeugt denselben Graphen wie (a, b, c_1) und kann daher ignoriert werden.

3.1.2 Einteilung in Haupt- und Subtopologien

Für den Graphen

$$\Gamma'' = \{P_{a,c_1}(l + q_2), P_{c_1,b}(l + q_1 + q_2), P_{a,b}(k), P_{b,a}(l + k)\} \tag{3.19}$$

aus Abb. 3.4c mit $(v_1'', v_2'', v_3'') = (c_1, a, b)$ gibt es keine Impulstransformation $l(l', k')$ und $k(k', l')$, die ihn in Γ aus Abb. 3.4a übergehen läßt. Er unterscheidet sich von Γ aber nur durch die Vertauschung der beiden äußeren Impulse q_1 und q_2, ist also topologisch eng mit ihm verwandt. Graphen, die *nach* einer Permutation der äußeren Impulse

$$\sigma_q \in S(q_1, \ldots, q_n) \tag{3.20}$$

ineinander übergehen, werden Subtopologien zur selben Haupttopologie genannt. Alle Topologien, die zu derselben Hauptopologie gehören, lassen sich sehr ähnlich mit der Parallel-/Orthogonalraum-Methode berechnen. In der Regel ändert sich nur die Lage, aber nicht die Anzahl der beitragenden Gebiete, und die Koeffizienten sind von vergleichbarer Komplexität. Die Erzeugung aller Topologien im vorangegangenen Abschnitt garantiert, daß es zu jeder Permutation σ_q *genau einen* Graphen gibt, der zu dem impulspermutierten Graphen identisch ist. Dieser kann natürlich auch der betrachtete Graph selbst sein (trivial im Falle $\sigma_q = \mathrm{id}$).

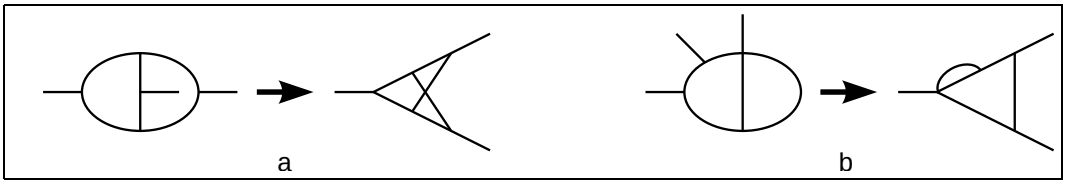


Abbildung 3.5: Beispiele für Topologien mit maximaler (a) und minimaler (b) Symmetrie in den äußeren Beinen

Die Anzahl der Subtopologien zu einer Haupttopologie mit n Beinen liegt zwischen 1 (z.B. Abb. 3.5a, dort führen alle σ_q den Graphen in sich selbst über) und $n!$ (z.B. Abb. 3.5b, dort sind alle q_i ausgezeichnet und damit unterscheidbar).

Der Algorithmus zum Finden der zusammengehörigen Subtopologien entspricht dem im letzten Abschnitt vorgestellten. Die Permutationen lassen sich leichter ausführen, wenn nicht schon zu Beginn q_n durch $q_n = -\sum_{i=1}^{n-1} q_i$ eliminiert wurde.

3.1.3 Faktorisierende Topologien

Die faktorisierenden Zweiloop-Topologien lassen sich analog zu den nicht-faktorisierenden aus der Vakuum-Topologie in Abb. 3.6 erzeugen, mit folgenden Unterschieden:

- Es gibt nur einen inneren Vertex a.
- Es gibt nur zwei Zweige, den linken mit l und den rechten mit k.
- Die n potentiellen Vertizes im l-Zweig werden mit $b_1, \ldots, b_n$ bezeichnet, die im k-Zweig mit $c_1, \ldots, c_n$.
- Der Propagator P_{a,b_1}, sofern existent, trägt als Impuls l plus die Summe der bei a einlaufenden Impulse. Ansonsten gilt dies für den Propagator $P_{a,a}$ des l-Loops.
- Der Propagator P_{a,c_1}, bzw. der Propagator $P_{a,a}$ des k-Loops, falls dieser nicht existent, trägt den Impuls k.
- Die Vertexkombinationen durchlaufen

$$(v_1, \ldots, v_n) \in [a, b_1, \ldots, b_n, c_1, \ldots, c_n]^n \,. \tag{3.21}$$

- Die Bedingungen für zu ignorierende Kombinationen reduzieren sich auf

$$\begin{aligned}
\max\{i \,|\, E(b_i) > 0\} &< \min\{i \,|\, E(b_i) = 0\} \\
\max\{i \,|\, E(c_i) > 0\} &< \min\{i \,|\, E(c_i) = 0\} \\
|\{i \,|\, E(b_i) > 0\}| &< |\{i \,|\, E(c_i) > 0\}| \\
|\{i \,|\, E(b_i) > 0\}| &= |\{i \,|\, E(c_i) > 0\}| \Longrightarrow \\
&\quad \exists i \geq 1 : \forall j < i : E(b_j) = E(c_j) \wedge E(b_i) < E(c_i) \,.
\end{aligned} \tag{3.22}$$

- Diese Ausschlußbedingungen garantieren, daß ein eventuell vorhandener Tadpole-Loop, der frei von äußeren Impulsen ist, den Impuls k hat.

- Zum Vergleichen von Graphen müssen nur die äußeren Vertizes permutiert werden, da Vertex a der einzige ausgezeichnete Vertex ist:

$$\sigma_V \in S(\{x_1, \ldots, x_n\})\,. \tag{3.23}$$

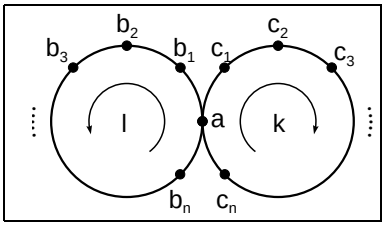


Abbildung 3.6: Benennung der Vertizes und Impulsfluß bei der faktorisierenden Vakuum-Topologie

3.1.4 Kürzen von Propagatoren

Eine große Bedeutung kommt bei der Berechnung von Feynman-Graphen dem Kürzen von Propagatoren zu (beispielsweise in Abschnitt 1.5.4 für Dreipunkt-Topologien). Dadurch kann man die Anzahl der zu berechnenden Integrale auf eine kleinere Menge von Basisintegralen reduzieren. Auf Einloop-Niveau können alle Tensor-Integrale durch Kürzen auf skalare Integrale reduziert werden [71], ebenso alle Zweiloop-Zweipunkt-Funktionen [90]. Aufgrund der großen Anzahl von Topologien (Abschnitt 3.1.7) ist es unerläßlich, das Kürzen zu automatisieren. Dabei muß man folgende Fälle unterscheiden:

- Die zu kürzende Topologie ist nicht-faktorisierend, der zu kürzende Propagator ist nicht der letzte in seinem Zweig: die gekürzte Topologie ist wieder nicht-faktorisierend.

- Die zu kürzende Topologie ist nicht-faktorisierend und der zu kürzende Propagator ist der letzte in seinem Zweig: die gekürzte Topologie ist faktorisierend.

- Die zu kürzende Topologie ist faktorisierend, der zu kürzende Propagator ist nicht der letzte in seinem Zweig: die gekürzte Topologie ist wieder faktorisierend.

- Die zu kürzende Topologie ist nicht-faktorisierend und der zu kürzende Propagator ist der letzte in seinem Zweig: die gekürzte Topologie ist effektiv ein Einloop-Graph, aber das resultierende Integral verschwindet wegen $\int d^D k\,(k^2)^\alpha = 0$, da die zweite Loopintegration nur noch Zählervariablen beinhaltet.

- Aus einer zu kürzenden Einloop-Topologie wird wieder eine Einloop-Topologie, sofern noch mindestens ein Propagator übrig bleibt. Ansonsten verschwindet das Integral.

Zum Finden der gekürzten Topologie geht man ähnlich wie bei der Suche nach Subtopologien vor. Zu jeder Topologie und jedem gekürzten Propagator findet man *genau eine* Topologie, sofern das Integral nicht verschwindet (beim Kürzen von faktorisierenden Topologien können Einloop-Topologien mit quadrierten Propagatoren entstehen). Im Unterschied zu den Subtopologien ist man hier aber nicht nur an der Zieltopologie interessiert,

sondern auch an der Propagatorpermutation und der zugehörigen Impulstransformation $l'(l,k)$ und $k'(l,k)$, mit der man den Zähler des Graphen transformieren muß.

3.1.5 Effektive Anzahl äußerer Beine

Die Komplexität der Berechnung eines Feynman-Graphen steigt mit der Anzahl seiner äußeren Beine. Daher ist es sinnvoll, n-Punkt-Topologien, bei denen zwei (oder mehr) äußere Beine am selben Vertex angreifen, möglichst früh zu einer $(n-1)$-Punkt-Topologie zu vereinfachen, da der Graph nur von der Summe der an einem Vertex angreifenden Impulse abhängt. Die Anzahl der verschiedenen Vertizes, an denen äußere Beine angreifen, nennt man die effektive Anzahl äußerer Beine eines Graphen. Erzeugt man n-Punkt-Topologien nach aufsteigendem n, so wurde der effektive Graph bereits früher erzeugt. Hier ist man zusätzlich zur Zieltopologie, der Transformation der Loopimpulse und der Permutation der Vertizes noch an der Transformation der äußeren Impulse interessiert. Da dabei auch die äußeren Impulse beliebig umbenannt werden können, sind auch alle anderen Subtopologien zur selben Haupttopologie wie eine gefundene Lösung ebenfalls Lösungen.

3.1.6 Symmetriefaktoren

Aus den erzeugten Topologien erhält man durch Zuordnung von Teilchen zu den Propagatoren Feynman-Graphen. Weist dieser eine topologische Symmetrie auf, muß man in der Berechnung einen Symmetriefaktor berücksichtigen. Notwendige Voraussetzung dafür ist, daß die nackte Vakuumtopologie ohne äußere Beine schon eine entsprechende Symmetrie ausweist. Die faktorisierende Zweiloop-Topologie hat die Symmetrien

- Austausch der Loopimpulse $l \leftrightarrow k$ (Spiegelung an der Vertikalen),
- Umdrehen der Impulsrichtung $l \to -l$ (Spiegelung des l-Loops an der Horizontalen) und
- Umdrehen der Impulsrichtung $k \to -k$ (Spiegelung des k-Loops an der Horizontalen).

Die nicht-faktorisierende Topologie hat die Symmetrien

- Austausch der Zweige $l \leftrightarrow k$,
- Austausch der Zweige $l \leftrightarrow -(l+k)$,
- Austausch der Zweige $k \leftrightarrow -(l+k)$ und
- Umdrehen der Impulsrichtungen $l \to -l$ gleichzeitig mit $k \to -k$ (Spiegelung an der Horizontalen).

Zu jeder *Topologie* kann man nun testen, ob diese Symmetrien auch nach Anhängen der äußeren Beine erhalten bleiben. In der Berechnung von *Feynman-Graphen* wird aber nur dann ein Symmetriefaktor berücksichtigt, wenn zusätzlich die Teilchen in den die Symmetrie bestimmenden Propagatoren übereinstimmen (bzw. Teilchen-Antiteilchen-Paare sind, wenn die Impulsrichtung umgedreht wird).

Bei der faktorisierenden Topologie trägt jede der drei Symmetrien unabhängig mit einem Faktor 2 bei. Den Gesamt-Symmetriefaktor erhält man durch Multiplikation aller

	1-Loop	2-Loop faktorisierend	2-Loop nicht-faktorisierend
1-Punkt	1	2	2
2-Punkt	2	6	7
3-Punkt	5	26	36
4-Punkt	17	153	257

Tabelle 3.1: Anzahl der erzeugten Topologien

	1-Loop	2-Loop faktorisierend	2-Loop nicht-faktorisierend
1-Punkt	1	2	2
2-Punkt	2	5	6
3-Punkt	3	10	13
4-Punkt	5	22	32

Tabelle 3.2: Anzahl der erzeugten Haupttopologien

Symmetriefaktoren. Bei der nicht-faktorisierenden Topologie notiert man einen Faktor 2, wenn nur eine der drei Zweig-Austausch-Symmetrien besteht, und einen Faktor 6(= 3!), wenn alle drei bestehen. Die Spiegelung an der Horizontalen gibt einen unabhängigen Faktor 2.

Für die Praxis kann man automatisch zu jeder Topologie und jeder gefundenen Symmetrie eine Liste mit Propagatoren, deren Teilchen auf Gleichheit (bzw. auf Teilchen-Antiteilchen-Paar) zu testen sind, und dem zugehörigen Symmetriefaktor erstellen.

Da aber zur Zeit nicht geplant ist, in *xloops* vollständige Prozesse zu erzeugen, sondern die Ausgabe von QGRAF zu verwenden (siehe Anhang B), die bereits den Symmetriefaktor enthält, besteht zur Zeit keine Notwendigkeit, die Bestimmung des Symmetriefaktors zu implementieren. Die erzeugte Datenstruktur enthält aber alle Informationen, um dies bei Bedarf später nachzuholen.

3.1.7 Anzahl der erzeugten Topologien

Eine Übersicht über die Anzahl der verschiedenen Einloop- und Zweiloop-Topologien bis zu vier äußeren Beinen gibt Tabelle 3.1. In Tabelle 3.2 ist aufgeführt, in wieviele verschiedene Haupttopologien diese gruppiert werden können. Das Programm zum Erzeugen der Topologien (Anhang B) kann die Topologien auch graphisch ausgeben. Ein kleiner Ausschnitt ist in Anhang C aufgeführt.

3.1.8 Erweiterung auf Dreiloop-Topologien

Eine Erweiterung der Topologie-Erzeugung auf Dreiloop-Niveau ist mit dieser Methode möglich, wenn man zu jeder der möglichen Dreiloop-Vakuumtopologien in Abb. 3.7 den Impulsfluß festlegt. Es gibt vier nicht-faktorisierende, drei teilweise faktorisierende und zwei vollständig faktorisierende Vakuumtopologien. Sie können durch Schließen der äußeren Beine aller Zweiloop-Zweipunkt-Topologien erzeugt werden, wenn man dabei doppelt entstehende ignoriert. Einige Vakuum-Topologien haben bereits innere Vertizes mit mehr als vier Beinen. Diese treten nicht direkt in Feynman-Graphen des Standardmodells auf, werden aber beispielsweise beim Kürzen von Propagatoren benötigt.

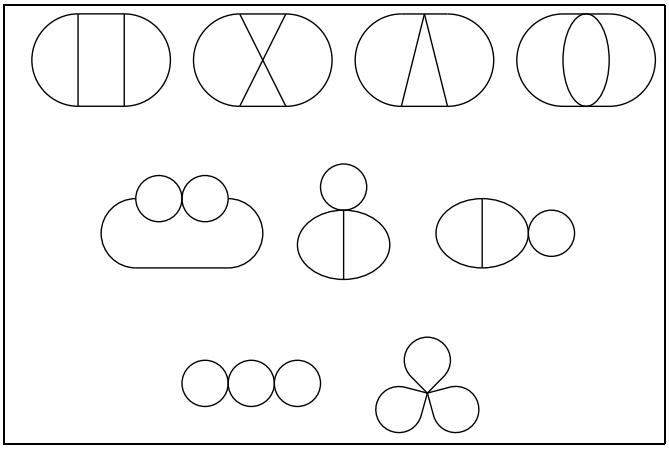


Abbildung 3.7: Dreiloop-Vakuum-Topologien

Beispielsweise kann man für die planare Dreiloop-Vakuumtopologie die Loopimpulse l_1, l_2 und l_3 wie in Abb. 3.8 fließen lassen. Der Graph wird durch die inneren Vertizes a, b, c und d in die Zweige l_1, $l_1 + l_2$, l_2 oben, l_2 unten, $l_2 + l_3$ und l_3 aufgeteilt, an denen man potentielle äußere Vertizes $e_1, \ldots, e_n$, $f_1, \ldots, f_n$, $g_1, \ldots, g_n$, $h_1, \ldots, h_n$, $i_1, \ldots, i_n$ und $j_1, \ldots, j_n$ anbringt.

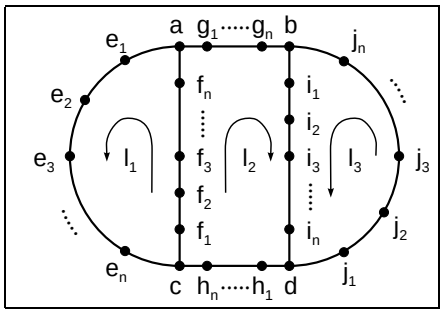


Abbildung 3.8: Benennung der Vertizes und Impulsfluß bei einer der Dreiloop-Vakuum-Topologien

3.2 Reduzierung der Parallelraum-Dimension

Bei bestimmten Impulskonfigurationen reduziert sich die effektive Anzahl der äußeren Beine eines Feynman-Graphen und damit seine Parallelraum-Dimension, z.B. wenn äußere Impulse linear abhängig sind oder der Graph nur eine Funktion einer Linearkombination dieser Impulse ist.

So läßt sich eine Einloop-Dreipunkt-Funktion, für die $q_2 = \alpha q_1$ (oder $q_1 = 0$) gilt — was in Parallel-/Orthogonalraumvariablen Gl. (1.2) $q_{21} = 0$ impliziert — durch Partialbruchzerlegung in eine Summe von drei Zweipunkt-Funktionen aufteilen:

$$\frac{1}{P_1(k+q_1)P_2(k-\alpha q_1)P_3(k)} = \frac{A}{P_1P_2} + \frac{B}{P_1P_3} + \frac{C}{P_1P_3} \tag{3.24}$$

mit

$$\begin{aligned} A &= -\frac{e_1+e_2}{(e_1+e_2)(m_3^2+e_1e_2)-e_1m_2^2-e_2m_1^2} \\ B &= \frac{e_1}{(e_1+e_2)(m_3^2+e_1e_2)-e_1m_2^2-e_2m_1^2} \\ C &= \frac{e_2}{(e_1+e_2)(m_3^2+e_1e_2)-e_1m_2^2-e_2m_1^2}\,. \end{aligned} \tag{3.25}$$

Auf Zweiloop-Niveau lassen sich bei der planaren Dreipunkt-Funktion (Abschnitt 1.5.6) die Propagatoren in k analog zu Gl. (3.24) zerlegen. Man erhält so Integrale der reduzierten planaren Dreipunkt-Funktion (Abschnitt 1.5.8) bzw. einer effektiven Zweipunkt-Funktion. Bei der gekreuzten Dreipunkt-Funktion (Abschnitt 1.5.7) zerlegt man mit $P_1(l+k-q_1)$, $P_2(l+k+q_2)$, $P_3(l-q_1)$, $P_4(k+q_2)$, $P_5(l)$ und $P_6(k)$:

$$\begin{aligned} \frac{1}{P_1P_2P_3P_4P_5P_6} &= \frac{A'}{P_2P_3P_4P_5P_6}+\frac{B'}{P_1P_3P_4P_5P_6}+\frac{C'}{P_1P_2P_4P_5P_6} \\ &\quad +\frac{D'}{P_1P_2P_3P_5P_6}+\frac{E'}{P_1P_2P_3P_4P_6}+\frac{F'}{P_1P_2P_3P_4P_5} \end{aligned} \tag{3.26}$$

mit

$$\begin{aligned} A' &= -\frac{e_1e_2}{(e_1+e_2)(e_2(m_5^2-m_3^2)+e_1(m_4^2-m_6^2))+e_1e_2(m_1^2-m_2^2)} \\ B' &= \frac{e_1e_2}{(e_1+e_2)(e_2(m_5^2-m_3^2)+e_1(m_4^2-m_6^2))+e_1e_2(m_1^2-m_2^2)} = -A' \\ C' &= \frac{(e_1+e_2)e_2}{(e_1+e_2)(e_2(m_5^2-m_3^2)+e_1(m_4^2-m_6^2))+e_1e_2(m_1^2-m_2^2)} \\ D' &= -\frac{(e_1+e_2)e_1}{(e_1+e_2)(e_2(m_5^2-m_3^2)+e_1(m_4^2-m_6^2))+e_1e_2(m_1^2-m_2^2)} \\ E' &= -\frac{(e_1+e_2)e_2}{(e_1+e_2)(e_2(m_5^2-m_3^2)+e_1(m_4^2-m_6^2))+e_1e_2(m_1^2-m_2^2)} = -C' \\ F' &= \frac{(e_1+e_2)e_1}{(e_1+e_2)(e_2(m_5^2-m_3^2)+e_1(m_4^2-m_6^2))+e_1e_2(m_1^2-m_2^2)} = -D'\,. \end{aligned} \tag{3.27}$$

Bei allen resultierenden Integralen handelt es sich um die reduzierte planare Dreipunkt-Funktion. Diese läßt sich für den Fall $q_2 = \alpha q_1$ durch minimale Abänderung der Parallel-/Orthogonalraum-Methode für Zweipunkt-Funktionen [54] berechnen.

Eine Dreipunkt-Funktion wie in Abb. 3.9 hängt effektiv nur von q_1 ab. Diese kann man durch eine Lorentztransformation auf eine Form bringen, in der der Parallelraum manifest eindimensional ist und nur von der 0-Komponente aufgespannt wird, vorausgesetzt, daß $q_1^2 > 0$ ist.

Dabei ergibt sich jedoch das Problem, daß in der Zählerstruktur noch Variablen des ursprünglichen Parallelraums stehen, die auf die niedrigere Dimension umgerechnet werden müssen. In einigen Fällen kann man dieses Problem verhindern, wenn vor dem Einsetzen der Feynmanregeln die effektive Parallelraum-Dimension bereits bekannt ist. Dies ist aber nicht mehr möglich, wenn Graphen wie in Abschnitt 1.5.4 durch Kürzen von Propagatoren im Verlauf der Rechnung entstehen.

Im folgenden soll zunächst die Umrechnung am Beispiel eines Einloop-Dreipunkt-Graphen verdeutlicht werden. Zweiloop-Graphen können behandelt werden, indem erst

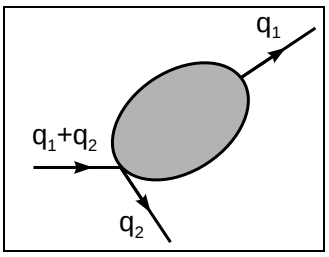


Abbildung 3.9: Eine Dreipunkt-Funktion, die effektiv nur eine Zweipunkt-Funktion ist und nur von q_1 abhängt

ein allgemeiner Ansatz für Orthogonalraum-Komponenten gelöst wird, mit dem dann beliebige Orthogonalraum-Komponenten ineinander umgerechnet werden können. Zuletzt wird gezeigt, daß die Ergebnisse dieses Abschnitts auch zum Eliminieren der freien Orthogonalraum-Lorentzindizes geeignet sind.

3.2.1 Einloop-Beispiel

Als Beispiel wird zunächst ein Einloop-Dreipunkt-Graph wie in Gl. (3.24) betrachtet. Nach einer eventuellen Lorentztransformation, Partialbruchzerlegung und Translation des Loopimpulses erhält man Terme wie

$$\begin{aligned} & \int d^D l \, \frac{l_1^2}{((l_0+q_0)^2 - l_1^2 - l_\perp^2 - m_1^2)(l_0^2 - l_1^2 - l_\perp^2 - m_2^2)} \\ = & \int d^D l \, \frac{l_1^2}{((l_0+q_0)^2 - \tilde{l}_\perp^2 - m_1^2)(l_0^2 - \tilde{l}_\perp^2 - m_2^2)} \,, \end{aligned} \tag{3.28}$$

wobei $l_\perp^2 = l_2^2 + l_3^2 + \ldots$ zu einem $(D-2)$-dimensionalen und $\tilde{l}_\perp^2 = l_1^2 + l_2^2 + l_3^2 + \cdots = l_1^2 + l_\perp^2$ zu einem $(D-1)$-dimensionalen Orthogonalraum gehört. Da das Integral in der Nennerstruktur symmetrisch bezüglich der einzelnen Orthogonalraumkomponenten l_1, l_2, $\ldots$ ist, sieht man, daß hier die Ersetzung

$$\begin{aligned} & \int d^D l \, \frac{l_1^2}{((l_0+q_0)^2 - \tilde{l}_\perp^2 - m_1^2)(l_0^2 - \tilde{l}_\perp^2 - m_2^2)} \\ = & \int d^D l \, \frac{1}{D-1} \frac{\tilde{l}_\perp^2}{((l_0+q_0)^2 - \tilde{l}_\perp^2 - m_1^2)(l_0^2 - \tilde{l}_\perp^2 - m_2^2)} \end{aligned} \tag{3.29}$$

lauten muß und damit das Integral vollständig in Variablen eines eindimensionalen Parallelraums ausgedrückt ist.

Höhere Tensorstrukturen, insbesondere im Zweiloop-Fall, lassen sich aber nicht mehr mit diesen einfachen Argumenten umrechnen. Daher soll im folgenden ein Verfahren vorgestellt werden, das den allgemeinen Fall, unter Berücksichtigung einer Umsetzung in GiNaC, behandelt. Ein geschlossener Ausdruck für die Umrechnung einer Einloop-Zweipunkt-Funktion für $q_0 = 0$ auf einen nulldimensionalen Parallelraum findet sich in [8].

3.2.2 Der Zweiloop-Orthogonalraum-Tensor

Um Orthogonalraum-Variablen bei Zweiloop-Funktionen ineinander umzurechnen, betrachtet man zunächst Integrale mit einem beliebigen Orthogonalraum-Tensor im Zähler

und versucht, diesen durch Kombinationen metrischer Orthogonalraum-Tensoren auszudrücken. Sei

$$I_f[k^{\mu_{\perp 1}}\cdots k^{\mu_{\perp m}} l^{\nu_{\perp 1}}\cdots l^{\nu_{\perp n}}] \equiv \int d^{D_\perp}k \int d^{D_\perp}l\, k^{\mu_{\perp 1}}\cdots k^{\mu_{\perp m}} l^{\nu_{\perp 1}}\cdots l^{\nu_{\perp n}} f(k^{\rho_\perp}k_{\rho_\perp}, l^{\rho_\perp}l_{\rho_\perp}, k^{\rho_\perp}l_{\rho_\perp}) \tag{3.30}$$

also ein Zweiloop-Integral, bei dem alle Integrationsvariablen im Orthogonalraum liegen. Dabei muß $m+n$ gerade sein, anderfalls verschwindet das Integral. Da es per Definition im Orthogonalraum keine äußeren Impulse gibt, muß das Ergebnis eine Linearkombination von Produkten metrischer Tensoren im Orthogonalraum $g^{\mu_\perp\nu_\perp}$ sein, unter Berücksichtigung der Symmetrie $\mu_{\perp 1} \leftrightarrow \mu_{\perp 2} \leftrightarrow \cdots \leftrightarrow \mu_{\perp m}$ und $\nu_{\perp 1} \leftrightarrow \nu_{\perp 2} \leftrightarrow \cdots \leftrightarrow \nu_{\perp n}$. Die Kennzeichnung als Orthogonalraumgröße mit $\perp$ wird im folgenden der besseren Lesbarkeit wegen weggelassen.

Der allgemeinste Ansatz für einen symmetrischen Tensor der Stufe $N = n+m$ (N gerade) enthält $N!$ Terme. Von diesen sind wegen $g^{\nu\mu} = g^{\mu\nu}$ und der Kommutativität der Multiplikation nur

$$\frac{N!}{2^{N/2}(N/2)!} = (N-1)!! \tag{3.31}$$

unabhängig. Diese $(N-1)!!$ Terme werden gruppiert nach der Anzahl der $g^{\mu\nu}$, die Indizes der l- und k-Variablen mischen. Wenn n (und damit auch m) gerade ist, treten nur Terme mit einer geraden Anzahl gemischter $g^{\mu\nu}$ auf, bei ungeraden n und m nur eine ungerade Anzahl. Es gibt höchstens $\min(n,m)$ gemischte $g^{\mu\nu}$.

Für n, m gerade (es sei o.B.d.A. $m \le n$ angenommen) erhält man damit folgenden Ansatz:

$$\begin{aligned} I_f[k^{\mu_1}\cdots k^{\mu_m} l^{\nu_1}\cdots l^{\nu_n}] = \\ & c_0\{g^{\mu_1\mu_2}\cdots g^{\mu_{m-1}\mu_m} g^{\nu_1\nu_2}\cdots g^{\nu_{n-1}\nu_n}\}_{(\mu_1\ldots\mu_m,\nu_1\ldots\nu_n)} \\ +& c_2\{g^{\mu_1\nu_1} g^{\mu_2\nu_2} g^{\mu_3\mu_4}\cdots g^{\mu_{m-1}\mu_m} g^{\nu_3\nu_4}\cdots g^{\nu_{n-1}\nu_n}\}_{(\mu_1\ldots\mu_m,\nu_1\ldots\nu_n)} \\ +& \cdots \\ +& c_m\{g^{\mu_1\nu_1} g^{\mu_2\nu_2}\cdots g^{\mu_m\nu_m} g^{\nu_{m+1}\nu_{m+2}}\cdots g^{\nu_{n-1}\nu_n}\}_{(\mu_1\ldots\mu_m,\nu_1\ldots\nu_n)}. \end{aligned} \tag{3.32}$$

Für n, m ungerade (wieder $m \le n$) lautet der Ansatz:

$$\begin{aligned} I_f[k^{\mu_1}\cdots k^{\mu_m} l^{\nu_1}\cdots l^{\nu_n}] = \\ & c_1\{g^{\mu_1\nu_1} g^{\mu_2\mu_3}\cdots g^{\mu_{m-1}\mu_m} g^{\nu_2\nu_3}\cdots g^{\nu_{n-1}\nu_n}\}_{(\mu_1\ldots\mu_m,\nu_1\ldots\nu_n)} \\ +& c_3\{g^{\mu_1\nu_1} g^{\mu_2\nu_2} g^{\mu_3\nu_3} g^{\mu_4\mu_5}\cdots g^{\mu_{m-1}\mu_m} g^{\nu_4\nu_5}\cdots g^{\nu_{n-1}\nu_n}\}_{(\mu_1\ldots\mu_m,\nu_1\ldots\nu_n)} \\ +& \cdots \\ +& c_m\{g^{\mu_1\nu_1} g^{\mu_2\nu_2}\cdots g^{\mu_m\nu_m} g^{\nu_{m+1}\nu_{m+2}}\cdots g^{\nu_{n-1}\nu_n}\}_{(\mu_1\ldots\mu_m,\nu_1\ldots\nu_n)}. \end{aligned} \tag{3.33}$$

$\{\ldots\}_{(\mu_1\ldots\mu_m,\nu_1\ldots\nu_n)}$ steht für die symmetrisierte Summe $\mu_{\perp 1} \leftrightarrow \mu_{\perp 2} \leftrightarrow \cdots \leftrightarrow \mu_{\perp m}$ und $\nu_{\perp 1} \leftrightarrow \nu_{\perp 2} \leftrightarrow \cdots \leftrightarrow \nu_{\perp n}$ unter Weglassung trivialer Kombinationen.

Die Koeffizienten $c_0,\ldots,c_m$ bzw. $c_1,\ldots,c_m$ lassen sich durch Kontrahieren mit jeweils einem beliebigen Repräsentanten aus $\{\ldots\}_{(\mu_1\ldots\mu_m,\nu_1\ldots\nu_n)}$ und anschließendem Lösen des linearen Gleichungssystems mit $m/2+1$ bzw. $(m+1)/2$ Unbekannten bestimmen. Diese

Koeffizienten müssen nur einmal für jede (m,n)-Kombination berechnet werden. Die dabei auftretenden Kontraktionen $g^{\mu}{}_{\mu} \equiv g^{\mu_\perp}{}_{\mu_\perp} = D_\perp$ sind für eine beliebige Orthogonalraumdimension $D_\perp$ gültig. Die Koeffizienten c_i sind Integrale $I_f[p(k^{\rho_\perp}k_{\rho_\perp}, l^{\rho_\perp}l_{\rho_\perp}, k^{\rho_\perp}l_{\rho_\perp})] \equiv I_f[p(k^2, l^2, k\cdot l)]$, abgeleitet von Gl. (3.30), bei denen $k^{\mu_{\perp 1}}\cdots k^{\mu_{\perp m}} l^{\nu_{\perp 1}}\cdots l^{\nu_{\perp n}}$ durch Polynome von k^2, l^2 und $k\cdot l$, deren Koeffizienten rationale Funktionen der Orthogonalraum-Dimension $D_\perp$ sind, ersetzt wurde. Einige Beispiele für Tensoren 2. und 4. Stufe findet man in Tabelle 3.3.

$I_f[k^{\mu_1}k^{\mu_2}]$	$\frac{1}{D} g^{\mu_1\mu_2} I_f[k^2]$
$I_f[k^{\mu_1}l^{\nu_1}]$	$\frac{1}{D} g^{\mu_1\nu_1} I_f[k\cdot l]$
$I_f[k^{\mu_1}k^{\mu_2}k^{\mu_3}k^{\mu_4}]$	$\frac{1}{D(D+2)}(g^{\mu_1\mu_2}g^{\mu_3\mu_4} + g^{\mu_1\mu_3}g^{\mu_2\mu_4} + g^{\mu_1\mu_4}g^{\mu_2\mu_3}) I_f[k^4]$
$I_f[k^{\mu_1}k^{\mu_2}k^{\mu_3}l^{\nu_1}]$	$\frac{1}{D(D+2)}(g^{\mu_1\mu_2}g^{\mu_3\nu_1} + g^{\mu_1\mu_3}g^{\mu_2\nu_1} + g^{\mu_2\mu_3}g^{\mu_1\nu_1}) I_f[k^2 k\cdot l]$
$I_f[k^{\mu_1}k^{\mu_2}l^{\nu_1}l^{\nu_2}]$	$\frac{1}{D^2(D^2+1)}(g^{\mu_1\mu_2}g^{\nu_1\nu_2}((D+1)I_f[k^2l^2] - 2I_f[(k\cdot l)^2])$ $+(g^{\mu_1\nu_1}g^{\mu_2\nu_2} + g^{\mu_1\nu_2}g^{\mu_2\nu_1})(I_f[(k\cdot l)^2] - I_f[k^2l^2]))$

Tabelle 3.3: Beispiele für Zweiloop-Orthogonalraum-Tensoren. Alle Indizes sind Orthogonalraumindizes, $D \equiv D_\perp$, $k^2 \equiv k^{\rho_\perp}k_{\rho_\perp}$ etc.

3.2.3 Transformation der Impulskomponenten

Integrale, bei denen die Parallelraum-Dimension reduziert werden muß, treten beispielsweise beim Kürzen von Propagatoren oder in Abzugstermen auf. Für die konkrete Transformation der Orthogonalraum-Variablen muß dabei wegen der Metrik des Minkowskiraums zwischen der Reduzierung auf einen nulldimensionalen Parallelraum und den übrigen Fällen unterschieden werden.

3.2.3.1 Nulldimensionaler Parallelraum

Eine Reduzierung auf einen nulldimensionalen Parallelraum wird notwendig, wenn beispielsweise bei einer Zweipunkt-Funktion, bei der nur ein einziger Propagator den äußeren Impuls q enthält, dieser gekürzt wird und man so effektiv ein Vakuumgraph entsteht. Als Zählerstruktur soll daher ein Ausdruck der Form

$$k_0^{m_0}(k_\perp^2)^{m_\perp} l_0^{n_0}(l_\perp^2)^{n_\perp}(k_\perp l_\perp z)^r \tag{3.34}$$

durch die Variablen k^2, l^2 und $k\cdot l$, also einen nulldimensionalen Parallelraum, ausgedrückt werden. Dazu ersetzt man

$$\begin{aligned} k_\perp^2 &\rightarrow k_0^2 - k^2 \\ l_\perp^2 &\rightarrow l_0^2 - l^2 \\ k_\perp l_\perp z &\rightarrow k_0 l_0 - k\cdot l\,, \end{aligned} \tag{3.35}$$

drückt alle Loopvariablen durch kontravariante Komponenten aus

$$\begin{aligned} k^2 &= g_{\mu\nu}k^\mu k^\nu \\ l^2 &= g_{\mu\nu}l^\mu l^\nu \\ k\cdot l &= g_{\mu\nu}k^\mu l^\nu \\ k_0 &= k^\mu\big|_{\mu=0} \\ l_0 &= l^\nu\big|_{\nu=0} \end{aligned} \tag{3.36}$$

und benutzt das Ergebnis aus dem vorigen Abschnitt, wobei noch einige $g_{\mu\nu}$ kontrahiert werden müssen.

3.2.3.2 Eindimensionaler Parallelraum

Bei Reduzierung auf einen eindimensionalen Parallelraum ($D_\perp = D-1$) hat man die Zählerstruktur

$$k_0^{m_0}k_1^{m_1}(k_\perp^2)^{m_\perp}l_0^{m_0}l_1^{n_1}(l_\perp^2)^{n_\perp}(k_\perp l_\perp z)^r\,. \tag{3.37}$$

k_0 und l_0 bleiben unverändert, die übrigen Variablen sollen durch $\tilde{k}_\perp^2$, $\tilde{l}_\perp^2$ und $\tilde{k}_\perp\tilde{l}_\perp\tilde{z}$ ausgedrückt werden. Die Ersetzung lautet

$$\begin{aligned} k_\perp^2 &\to \tilde{k}_\perp^2 - k_1^2 \\ l_\perp^2 &\to \tilde{l}_\perp^2 - l_1^2 \\ k_\perp l_\perp z &\to \tilde{k}_\perp\tilde{l}_\perp\tilde{z} - k_1 l_1\,. \end{aligned} \tag{3.38}$$

In kontravarianten Komponenten schreibt man

$$\begin{aligned} \tilde{k}_\perp^2 &= -g_{\mu_\perp\nu_\perp}k^{\mu_\perp}k^{\nu_\perp} \\ \tilde{l}_\perp^2 &= -g_{\mu_\perp\nu_\perp}l^{\mu_\perp}l^{\nu_\perp} \\ \tilde{k}_\perp\tilde{l}_\perp\tilde{z} &= -g_{\mu_\perp\nu_\perp}k^{\mu_\perp}l^{\nu_\perp} \\ k_1 &= k^\mu\big|_{\mu=1} \\ l_1 &= l^\nu\big|_{\nu=1} \end{aligned} \tag{3.39}$$

Dabei muß man das Problem in der Notation beachten, daß die Parallel-/Orthogonalraum-Variable k_1 für die *kontravariante* Komponente $k^\mu\big|_{\mu=1}$ steht und nicht für die *kovariante*, für die $k_1 = -k_\mu\big|_{\mu=1}$ gilt.

Beim Rückübersetzen der Ergebnisse muß man auch wieder beachten, daß $k^{\rho_\perp}k_{\rho_\perp} = -\tilde{k}_\perp^2$ etc. ist.

3.2.3.3 Zwei- und mehrdimensionaler Parallelraum

Den Fall eines zweidimensionalen Parallelraums kann man analog zum eindimensionalen behandeln, wenn man l_1 durch l_2 und k_1 durch k_2 ersetzt sowie die Orthogonalraum-Dimension entsprechend anpaßt ($D_\perp = D-2, \ldots$).

3.2.4 Clifford-Algebra und Orthogonalraum-Komponenten

Die Ergebnisse aus Abschnitt 3.2.2 werden noch an einer weiteren Stelle in *xloops* benötigt, nämlich bei der Behandlung von Orthogonalraum-Komponenten der Loopimpulse, die durch Einsetzen von Feynman-Regeln entstehen und mit Clifford-Algebra-Elementen kontrahiert werden, z.B. bei

$$\not{k} = k^0\gamma^0 - k^1\gamma^1 + g_{\mu_\perp\nu_\perp}k^{\mu_\perp}\gamma^{\mu_\perp} \tag{3.40}$$

aus einem Fermion-Propagator. Da die $\gamma_{\mu_\perp}$ Konstanten bei den Loop-Integrationen sind und daher aus dem Integral herausgezogen werden können, müssen die $k^{\mu_\perp}$ und $l^{\nu_\perp}$ vor dem Anwenden der Tensorberechnungsmethoden aus Abschnitt 1.5 erst in Kombinationen von $k_\perp^2$, $l_\perp^2$ und $k_\perp l_\perp z$ umgewandelt werden. Dies ist aber gerade die Lösung von Gl. (3.32) bzw. Gl. (3.33). Die $g^{\mu_\perp\nu_\perp}$ müssen dann noch in einem weiteren Schritt mit den $\gamma_{\mu_\perp}$ entsprechend

$$g^{\mu_\perp\nu_\perp}\gamma_{\mu_\perp}\gamma_{\nu_\perp} = D_\perp \mathbb{1} \tag{3.41}$$

kontrahiert werden.

3.3 Ein Konzept für einen Nachfolger von *xloops*

In den vorangegangenen Teilen dieser Arbeit wurden verschiedene Grundlagen vorgestellt, die Voraussetzung für eine Berechnung von Zweiloop-Feynman-Graphen mit drei äußeren Beinen und beliebigen Massen und Impulsen sind, wie dies vor allem im Standardmodell auftritt. Das Programmpaket *xloops*, das seit Anfang der 90er Jahre in Mainz zum automatisierten Berechnen von Feynman-Graphen entwickelt wurde und zur Zeit Diagramme bis zur Einloop-Dreipunkt- und Zweiloop-Zweipunkt-Ordnung rechnen kann, weist jedoch einige Schwächen auf, die es nicht sinnvoll erschienen ließen, *xloops* weiterzuentwickeln und die Zweiloop-Dreipunkt-Funktionalität in das bestehende System einzubauen:

- *xloops* ist ursprünglich als Paket zum Berechnen von Einloop-Graphen konzeptioniert worden, die Zweiloop-Methoden wurden nachträglich hinzugefügt.
- *xloops* ist in Maple geschrieben. Die Gründe, die gegen Maple, vor allem bei größeren Programmpaketen, sprechen, wurden bereits in Abschnitt 2.1 dargelegt.
- Die Topologie-Erzeugung ist unvollständig, da keine Subtopologien erzeugt und unterschieden werden können (bei Zweiloop-Zweipunkt gibt es nur eine Topologie, bei der Subtopologien auftreten, diese wird in *xloops* als Spezialfall behandelt). Insbesondere ist es nicht möglich, sich eine Liste mit den expliziten Impuls je Propagator ausgeben zu lassen, da diese nur beim Einsetzen der Feynman-Regeln implizit aus Annahmen über die erzeugten Topologien gewonnen werden, die bei Zweiloop-Dreipunkt-Topologien nicht mehr allgemein gültig sind. Von dieser Topologie-Information hängen aber große Teile des Paketes ab. Außerdem können keine Topologien mit mehr als vier Vertizes, wie sie beispielsweise in der chiralen Störungstheorie benötigt werden, erzeugt werden.
- Die starre Listenstruktur als Rückgabewert einer Einloop-Funktion $(A, B, C) \equiv A/\varepsilon + B + C\varepsilon + \mathcal{O}(\varepsilon^2)$ verhindert die Berechnung von kollinear divergenten Graphen, die einen $1/\varepsilon^2$-Term benötigen (Abschnitt 1.4).

- Der Quellcode von *xloops* ist unvollständig dokumentiert, die ursprünglichen Entwickler sind nicht mehr greifbar, aber es sind noch Fehler im Programm (Sunset Zweiloop-Zweipunkt-Topologie, Tensoren der Einloop-Dreipunkt-Funktion).

Daher soll im folgenden ein Konzept zur Neuprogrammierung von *xloops* in C++ auf Basis der Programmbibliothek GiNaC (Kapitel 2) vorgestellt werden, das offen genug ist für Erweiterungen, insbesondere bezüglich

- einer prinzipiell unbegrenzten Anzahl äußerer Beine, insbesondere auch Zweiloop-Vierpunkt-Funktionen,
- Dreiloop-Topologien,
- dem Standardmodell mit anderen Eichungen und
- beliebigen anderen Modelle, z.B. chirale Störungstheorie oder Supersymmetrie.

Eine graphische Benutzeroberfläche ist dabei von untergeordneter Bedeutung, kann aber noch später implementiert werden, da z.B. die tk-Bibliothek auch von C++ aus aufgerufen werden kann.

3.3.1 Topologien, Elementarteilchen und Modelle

Mit dem Verfahren aus Abschnitt 3.1 können alle benötigten Topologien erzeugt werden. Zu jeder Topologie erhält man u.a. Information über

- die grundlegende Vakuum-Topologie,
- die Anzahl der Vertizes mit einer definierten Numerierung, z.B. bei der nicht-faktorisierenden Vakuum-Topologie (Abb. 3.3) $a \to 0$, $b \to 1$, $c_1 \to 2$, ... ,
- die Anzahl der äußeren Beine, deren Positionen und die effektive Anzahl äußerer Beine (Abschnitt 3.1.5),
- die Anzahl der Propagatoren je Zweig (l, k, $l+k$) und eine Liste der Propagatoren je Zweig mit Anfangsvertex, Endvertex und Impuls, somit alle Information über zusammenhängende Linien und Loops,
- Hinweise auf verwandte Topologien (Subtopologien) und
- Hinweise auf Topologien, die entstehen, wenn man einen der Propagatoren kürzt, inklusive einer Umbenennung der Propagatoren und Impulse.

Da die Programmierumgebung C++ ist, bietet es sich an, zum Repräsentieren von Elementarteilchen und Modellen einen objektorientierten Ansatz zu wählen. Eine grundlegende Klasse für *Elementarteilchen* implementiert die allen Teilchen gemeinsamen Eigenschaften *Masse* und *Spin* und einen Zeiger auf sein Antiteilchen, das es auch selbst sein kann. Dadurch ergibt sich schon eine Unterscheidung zwischen Bosonen und Fermionen.

Kennzeichnende Eigenschaften von Teilchen sind auch ihre Wechselwirkung. Deshalb ist ein Teilchen eng mit einem Modell verbunden, in dem sie existieren. Zum einen besteht das Modell aus einer Liste von Feynman-Regeln für Propagatoren und Vertizes. Zum anderen gibt es zusätzliche Rechenregeln, die daraus eine Amplitude berechnen. Das grundlegende Modell, das nur die ebenso grundlegende Klasse Elementarteilchen kennt, kann

damit aber z.B. schon feststellen, in welcher Reihenfolge die Feynman-Regeln aneinander gesetzt werden müssen, nämlich entlang der Fermionlinien, ohne ihre konkreten Werte zu kennen, oder weiß, daß eine Clifford-Spur bei geschlossenen Fermionloops gebildet werden muß (Abschnitt 3.3.2).

Physikalische Modelle, wie z.B. QCD, werden von diesem grundlegenden Modell abgeleitet, füllen Teilchen und Vertizes aus und erweitern die Rechenregeln, z.B. daß eine Spur über die Farbalgebra-Elemente bei geschlossenen Quarkloops gebildet werden muß. Dazu wird es auch eine Klasse oder einen ganzen Baum von *Elementarteilchen* ableiten, um alle Quantenzahlen seiner Teilchen repräsentieren zu können. Eine mögliche Hierarchie zeigt Abb. 3.10.

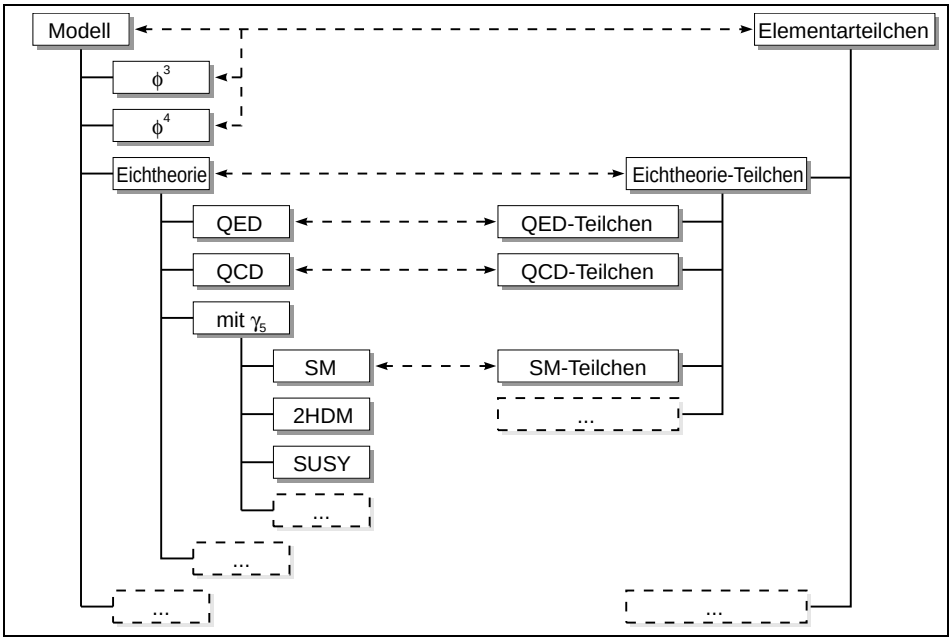


Abbildung 3.10: Eine mögliche Hierarchie für Teilchen und Modelle

3.3.2 Algorithmus zum Berechnen von Feynman-Graphen

Die Berechnung von einzelnen Feynman-Graphen mit Hilfe der Aufteilung der Raum-Zeit in einen Parallel- und einen Orthogonalraum kann zunächst ganz allgemein in einzelne Schritte zerlegt werden, die unabhängig vom betrachteten physikalischen Modell sind.

Das Ergebnis eines Graphen hängt von seiner Topologie, den Teilchen der inneren Propagatoren, den äußeren Teilchen, die nicht auf der Massenschale sein müssen und daher zur vollständigen Beschreibung zusätzlich durch Impulse parametrisiert werden müssen, und vom betrachteten Modell ab.

3.3.2.1 Transformation der äußeren Impulse

Die äußeren Impulse q_1, q_2 und $q_3 = -q_1 - q_2$ von Dreipunkt-Funktionen können auf verschiedene Arten parametrisiert werden, z.B. durch q_1^2, q_2^2 und q_3^2 oder durch Komponenten $q_1 = (q_{10}, q_{11}, q_{12}, q_{13})$, $q_2 = (q_{20}, q_{21}, q_{22}, q_{23})$. Zunächst muß festgelegt werden,

welche Impulse ein- und welche auslaufend definiert sind, da bei der Topologie-Erzeugung (Abschnitt 3.1) alle Impulse als einlaufend angenommen wurden.

Diese Impulse müssen nun durch eine Lorentztransformation in den minimalen Parallelraum in den ersten $D_{\|}$ Komponenten transformiert werden, die den Bedingungen aus Abschnitt 1.1.1.1 mit reellen Impulskomponenten genügen. Dies ist nicht immer möglich, z.B. kann der Fall $q_1^2 = q_2^2 = q_3^2 < 0$ nur in einem zweidimensionalen Parallelraum aus den 1- und 2-Komponenten verwirklicht werden, der Fall $q_1^2 = q_2^2 = q_3^2 > 0$ benötigt komplexe Komponenten e_1, e_2 oder q_z. Da diese Impulskonfigurationen keinem physikalischen Prozeß entsprechen, werden sie nicht betrachtet.

Eventuell muß auf eine gedrehte Variante (Subtopologie) ausgewichen werden, um eines der Teilchen in sein Ruhesystem setzen zu können. Nach diesen Vorarbeiten hat man die effektive Parallelraum-Dimension $D_{\|}$ festgestellt, die in der Regel $D_{\|} = n - 1$ für n-Punkt-Funktionen ist ($n \leq 5$), aber auch geringer sein kein.

3.3.2.2 Analyse des Graphen

Im nächsten Schritt wird die Struktur des Graphen analysiert. Dabei kann man Graphen mit nicht-existenten Vertizes, z.B. $H\gamma\gamma$, abweisen. Ob ein Vertex existiert, kann von Modellparametern abhängen, beispielsweise der Vertex $Hs\bar{s}$ von einer angenommenen Masse der leichten Quarks. Um konsistent zu bleiben, sollte die Funktion für die Vertizes (Abschnitt 3.3.2.3) Null bei nicht-existenten Vertizes zurückgeben, wodurch die Amplitude für den gesamten Graphen auch Null wird.

Weiterhin muß untersucht werden, bei welchen Linien im Graphen es sich um Fermionen handelt. Zusammenhängenden Fermionlinien wird ein gemeinsames Etikett für ihre Repräsentation der Clifford- (Abschnitt 2.6.2), Farbalgebra (Abschnitt 2.4.2.1) oder eventuelle weitere Algebren gegeben. Dabei kann man auch geschlossene Fermionloops (maximal einer bei Zweiloop-Graphen) feststellen, über die später eine Spur gebildet werden muß. So erhält man eine Reihenfolge, wie die Feynman-Regeln für die Propagatoren und Vertizes in Abschnitt 3.3.2.4 zusammengefügt werden müssen.

Modelle, die ein γ_5 benötigen (beispielsweise das Standardmodell), müssen diesen Schritt noch verfeinern, um den richtigen Anfangspunkt bei Fermionloops zu finden, da die Spur nicht mehr zyklisch ist [55]. Ebenso müßten Modelle mit Vier-Fermionen-Vertizes zusätzliche Logik über das Zusammengehören der Linien beisteuern.

3.3.2.3 Feynman-Regeln für Propagatoren und Vertizes

Vollständigkeitrelationen wie

$$(\not{p} + m)_{\alpha\beta} = 2m \sum_s u_\alpha(p,s)\bar{u}_\beta(p,s) \tag{3.42}$$

für den Zähler eines Fermion-Propagators legen nahe, daß man sich einen Propagator in zwei Hälften zerlegt denken kann, von denen jede mit einem Vertex verbunden ist und an ihren Enden freie Indizes beliebiger Art trägt. Äußere Teilchen kann man dann ebenfalls als „halbe“ Propagatoren auffassen.

Zu jedem „halben“ Teilchen wird zunächst ein Satz von Indizes angefordert (den Impuls des Teilchens kann man wie einen Index auffassen). Diese Indizes werden dann einmal in kontravarianter Form [1] zusammen mit denen der zweiten Hälfte eines Propagators benutzt,

[1] GiNaC kann auch für Nicht-Lorentzindizes zwischen kontra- und kovariant unterscheiden (Abschnitt 2.4.2.)

um dessen Feynman-Regel zu erhalten. Ein zweites mal werden sie in kovarianter Form mit den Indizes der anderen Teilchen für die Feynman-Regel des Vertizes verwendet. Über diese Indizes, die genau zweimal als ko-/kontravariantes Paar auftreten, wird entsprechend der Einsteinschen Summenkonvention summiert. Außer dem Teilchen selbst und seinen Indizes benötigen die Funktionen, die die Feynman-Regeln zurückgeben, noch den Impuls und ein eventuelles Repräsentationsetikett. Alle Teilchen an einem Vertex werden als einlaufend deklariert. Beim Umkehren der Orientierung eines Propagtors, die aus der Topologie-Erzeugung vorgegeben ist, ändert sich das Vorzeichen des Impulses, und das Teilchen muß durch sein Antiteilchen ersetzt werden (Abschnitt 3.3.1).

Als Beispiel findet man in Abb. 3.11 die Indizes eines Gluon- und eines Quark-Propagators sowie eines $gq\bar{q}$-Vertizes, wenn man die Clifford- und die Farbalgebra mit Matrixkomponenten rechnet.

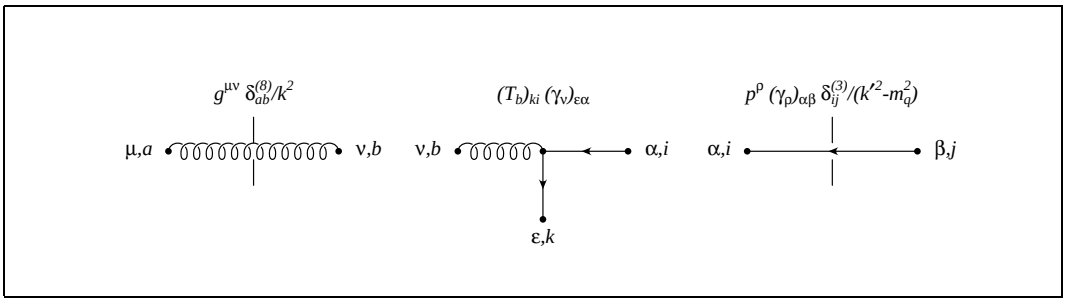


Abbildung 3.11: Indizes eines Gluon- und eines Quark-Propagators sowie eines $gq\bar{q}$-Vertizes

Die Feynman-Regeln der Vertizes werden als Tupel $(\mathcal{S}, \mathcal{L}, \dots)$ zurückgegeben, wobei $\mathcal{S}$ den skalaren Anteil (Massen, Kopplungskonstanten, numerische Faktoren, i) und $\mathcal{L}$ den Lorentz-Anteil (Impulse, Clifford-Algebra inkl. γ_5) bezeichnet. Weitere, modellabhängige Anteile wie z.B. Farbfaktoren von SU(3)-Darstellungen können, müssen aber nicht in separaten Elementen des Tupels angegeben werden (siehe Abschnitt 3.3.2.4).

Um die Vereinfachungen der Produkte von Elementen der Clifford-Algebra und der Impulskomponenten bei Aufteilung in Parallel- und Orthogonalraum auszunutzen, sollten alle Impulse und Clifford-Elemente durch kontravariante Parallel-/Orthogonalraum-Komponenten ausgedrückt werden — die in Abschnitt 1.1.1 eingeführten Komponenten k_0, k_1, … sind *kontravariant* — und alle D-dimensionalen Lorentzindizes nur noch in $g^{\mu\nu}$ gemäß

$$\begin{aligned}
\gamma^\mu &= g^\mu{}_\nu\gamma^\nu = g^\mu{}_0\gamma^0 + g^\mu{}_1\gamma^1 + g^\mu{}_{\nu_\perp}\gamma^{\nu_\perp} \\
\gamma_\mu &= g_{\mu\nu}\gamma^\nu = g_{\mu 0}\gamma^0 + g_{\mu 1}\gamma^1 + g_{\mu\nu_\perp}\gamma^{\nu_\perp} \\
q_i{}^\mu &= g^\mu{}_\nu q_i{}^\nu = g^\mu{}_0 q_i{}^0 + g^\mu{}_1 q_i{}^1 \\
k^\mu &= g^\mu{}_\nu k^\nu = g^\mu{}_0 k^0 + g^\mu{}_1 k^1 + g^\mu{}_{\nu_\perp} k^{\nu_\perp} \\
\not{q}_i &= g_{\mu\nu} q_i{}^\mu \gamma^\nu = q_i{}^0\gamma^0 - q_i{}^1\gamma^1 \\
\not{k} &= g_{\mu\nu} k^\mu \gamma^\nu = k^0\gamma^0 - k^1\gamma^1 + g_{\mu_\perp\nu_\perp} k^{\mu_\perp}\gamma^{\nu_\perp}
\end{aligned} \tag{3.43}$$

(am Beispiel eines zweidimensionalen Parallelraums, analog für $q_{i\mu}$, k_μ, l^μ l_μ und $\not{l}$) auftreten.

Läßt man Eichungen wie beispielswise axiale Eichungen außer Betracht, so haben Feynman-Regeln für Propagatoren die Form

$$\sum_{i=1}^{N} \frac{(\mathcal{S}_i, \mathcal{L}_i, \dots)}{(p^2 - m_i^2)^{n_i}}\,, \tag{3.44}$$

die aber nicht als rationale Funktion, sondern in einer Struktur mit separatem Zähler, Massen und Nennerpotenzen zurückgegeben wird. Der Fall $N = 2$ wird für den massiven Vektorboson-Propagator

$$\frac{g^{\mu\nu} - \frac{(1-\xi)p^\mu p^\nu}{p^2 - \xi m^2}}{p^2 - m^2}\,, \tag{3.45}$$

zusätzlich der Fall $n_1 = 2$ für den Photon-Propagator

$$\frac{g^{\mu\nu} - \frac{(1-\xi)p^\mu p^\nu}{p^2}}{p^2}\,, \tag{3.46}$$

jeweils bei Eichungen mit $\xi \neq 1$, benötigt. Höhere Nennerpotenzen benötigen bei Zweiloop-Dreipunkt-Funktionen wegen der in Abschnitt 1.5.2 beschriebenen Probleme noch methodische Vorarbeiten.

3.3.2.4 Einsetzen der Feynman-Regeln

Vor der weiteren Berechnung müssen die Summen aus Gl. (3.44), falls vorhanden, ausmultipliziert werden, da sie die Nennerstruktur vorgeben.

Jetzt können die Feynman-Regeln für Propagatoren und Vertizes entsprechend der in Abschnitt 3.3.2.2 ermittelten Reihenfolge nicht-kommutativ aneinander multipliziert werden. Die einzelnen Beiträge in $(\mathcal{S}, \mathcal{L}, \dots)$ können separat multipliziert werden. Wenn beispielsweise der Farbanteil zusammen mit $\mathcal{L}$ zurückgegeben wurde, so wird er von GiNaC in der Klasse `ncmul` separiert (Abschnitt 2.4.1).

Die so entstehenden nicht-kommutativen Produkte von Elementen aus Clifford- und anderen Algebren werden soweit wie möglich vereinfacht (vgl z.B. Abschnitt 2.6.2). $g^{\mu\nu}$ (nicht $g_{\mu_\perp\nu_\perp}$) werden kontrahiert, ebenso äquivalente Ausdrücke in anderen Algebren (beispielsweise $\delta^{(8)}_{ab}$, Abschnitt 2.4.2.1). Über die in Abschnitt 3.3.2.2 gefundenen Fermionloops muß die Spur über die Clifford- und die anderen Algebra-Elemente mit dem passenden Repräsentationsetikett gebildet werden. Für diese Vereinfachungsregeln bieten sich virtuelle Funktionen an, da so die Basisklasse *Modell* nichts von weiteren Algebren wissen muß.

3.3.2.5 Ausdrücken durch Tensor-Integrale

Um die zu berechnenden Integrale in der Form Gl. (1.90) ausdrücken zu können, müssen noch die freien Orthogonalraum-Indizes in $k^{\mu_\perp}$ und $l^{\nu_\perp}$ gemäß Gl. (3.32) bzw. Gl. (3.33) durch Kombinationen von $l_\perp^2$, $k_\perp^2$, $l_\perp k_\perp z$ und metrischen Tensoren im Orthogonalraum $g^{\mu_\perp\nu_\perp}$ ausgedrückt werden. Die $g^{\mu_\perp\nu_\perp}$ können mit den $\gamma_{\rho_\perp}$ kontrahiert werden.

Die verbleibenden Zähler sind nun Polynome in k_0, k_1, $k_\perp^2$, l_0, l_1, $l_\perp^2$ und $k_\perp l_\perp z$ (bei $D_\parallel = 2$) und damit sind die Integrale von der Form Gl. (1.90), deren Ergebnis eine Laurent-Reihe in ε ist.

Da die Integrationsgebiete der numerischen Zweifach-Integration nur von der Topologie und den äußeren Impulsen abhängen, sollte eine Möglichkeit bestehen, verschiedene Beiträge gemeinsam numerisch zu integrieren. Die erzielbare Genauigkeit kann so verbessert werden. Bei der Behandlung von Infrarot-Divergenzen ist dies sogar zwingend notwendig (Abschnitt 1.3.2).

3.3.3 Berechnung von Matrixelementen durch Zusammenarbeit mit anderen Paketen

Durch die Implementierung in C++ wird es möglich, relativ einfach andere Pakete einzubinden. Als Benutzeroberfläche kann die tk-, qt- oder gnome-Bibliothek benutzt werden. Zum Erzeugen aller Graphen, die zu einem Prozeß beitragen, kann auf das Paket QGRAF [68] zurückgegriffen werden. Die Ausgabe im `symbolic`-Modus enthält mit einer Liste aller Vertizes, einer Liste aller Propagatoren mit Impulsen und dem Symmetriefaktor inklusive einem Vorzeichen für Antisymmetrisierung bei identischen Fermionen im Endzustand alle Information, um mit einem Algorithmus ähnlich zu Abschnitt 3.1.1 jedem Graphen eindeutig eine Topologie zuordnen zu können.

Interessiert man sich nicht nur für differentielle Wirkungsquerschnitte oder Zerfallsbreiten, so muß man bei n-Punkt-Funktionen mit $n > 3$ nicht-triviale Phasenraumintegrationen ausführen, was in der Regel nur noch mit Monte-Carlo-Methoden möglich ist. Dafür existiert beispielsweise das Paket CompHEP [4], das als Schnittstelle eine Funktion erwartet, die das quadrierte Matrixelement in Abhängigkeit von äußeren Viererimpulse berechnet. Da aufgrund der Monte-Carlo-Methoden die Viererimpulse unter Berücksichtigung experimenteller Schnitte zufällig gewählt werden, ist es notwendig, diese durch eine Lorentztransformation in das Parallel-/Orthogonalraum-Bezugssystem (Abschnitt 1.1.1.1) umzurechnen.

Anhang A

Potenzregeln

GiNaC vereinfacht nur dann $(ax)^b$ zu $a^b x^b$ und $(x^a)^b$ zu x^{ab}, wenn die Variablen x, a und b, die prinzipiell beliebige komplexe Werte annehmen können, bestimmte Bedingungen erfüllen. Diese Bedingungen und der Beweis ihrer Richtigkeit [87] werden in diesem Anhang gezeigt.

A.1 Definitionen und allgemeine Regeln

Die Potenz zweier komplexer Zahlen ist in GiNaC über den komplexen Logarithmus

$$\ln x \equiv \ln |x| + i \arg(x) \tag{A.1}$$

mit

$$-\pi < \arg(x) \leq \pi \tag{A.2}$$

und seine Umkehrfunktion, die Exponentialfunktion

$$\exp(\ln x) = x\,, \tag{A.3}$$

(im allgemeinen gilt *nicht* $\ln(\exp x) = x$) definiert:

$$x^a \equiv e^{a \ln x} \quad \text{falls} \quad x \neq 0\,. \tag{A.4}$$

0^a ist definiert als Null, wenn $\mathrm{Re}(a) > 0$, ansonsten ist es undefiniert. Der Fall, daß die Basis Null ist, wird im folgenden nicht berücksichtigt, da er vor den hier diskutierten Regeln abgefangen wird. Dies gilt wegen Abschnitt 2.2.3 und 2.3.3.2 auch für den Fall, daß die Basis aus mehreren Faktoren besteht, von denen einer Null ist.

Aus der Multiplikationseigenschaft der Exponentialfunktion für beliebige komplexe Argumente x und y

$$e^x e^y = e^{x+y} \tag{A.5}$$

folgt sofort

$$x^{-a} = \frac{1}{x^a} \tag{A.6}$$

für beliebige komplexe x and a.

A.2 Potenzen von Produkten

Es gilt: $(ax)^b = a^b x^b$, wenn b eine ganze Zahl oder a reell und positiv ist. x kann dabei eine beliebige komplexe Zahl sein (bzw. ein Symbol oder zusammengesetzter Ausdruck, über den man keine Aussage machen kann).

$b \in \mathbb{Z}$, $x \in \mathbb{C}$ und $a \in \mathbb{C}$

Der Fall $b = 0$ ist trivial. Sei daher zunächst $b > 0$:

$$\begin{aligned}(ax)^b &= \underbrace{(ax)\cdots(ax)}_{b\times} \\ &= \underbrace{a\cdots a}_{b\times}\underbrace{x\cdots x}_{b\times} \\ &= a^b x^b\,. \end{aligned} \tag{A.7}$$

Damit kann man nun den Fall $b < 0$ (so daß $b = -|b|$) behandeln:

$$\begin{aligned}(ax)^b &= \frac{1}{(ax)^{|b|}} \\ &= \frac{1}{a^{|b|}x^{|b|}} \\ &= a^{-|b|}x^{-|b|} \\ &= a^b x^b\,. \end{aligned} \tag{A.8}$$

$a > 0$, $x \in \mathbb{C}$ und $b \in \mathbb{C}$

Bei reellem, positivem a wendet man zunachst Gl. (A.4) und Gl. (A.1) an:

$$\begin{aligned}(ax)^b &= e^{b\ln(ax)} \\ &= e^{b(\ln|ax| + i\arg(ax))}\,. \end{aligned} \tag{A.9}$$

Mit $a > 0$ gilt

$$\ln|ax| = \ln|a| + \ln|x| = \ln a + \ln|x| \tag{A.10}$$

und

$$\arg(ax) = \arg(x)\,. \tag{A.11}$$

Damit folgt

$$\begin{aligned}e^{b(\ln|ax| + i\arg(ax))} &= e^{b(\ln a + \ln|x| + i\arg(x))} \\ &= e^{b(\ln a + \ln x)} \\ &= e^{b\ln a}e^{b\ln x} \\ &= a^b x^b\,. \end{aligned} \tag{A.12}$$

Die Regel gilt analog, wenn die Basis aus mehr als zwei Faktoren besteht.

A.3 Potenzen von Potenzen

Es gilt: $(x^a)^b = x^{ab}$, wenn b eine ganze Zahl oder a reell und im Intervall $-1 < a \leq 1$ ist. x kann wieder eine beliebige komplexe Zahl sein.

$b \in \mathbb{Z}$, $x \in \mathbb{C}$ und $a \in \mathbb{C}$

Der Fall $b = 0$ ist auch hier trivial. Sei daher zunächst wieder $b > 0$:

$$\begin{aligned}(x^a)^b &= \underbrace{(x^a) \cdots (x^a)}_{b\times} \\ &= \underbrace{e^{a \ln x} \cdots e^{a \ln x}}_{b\times} \\ &= e^{\underbrace{a \ln x + \cdots + a \ln x}_{b\times}} \\ &= e^{ab \ln x} \\ &= x^{ab} \,. \end{aligned} \tag{A.13}$$

Für $b < 0$ (mit $b = -|b|$) gilt

$$\begin{aligned}(x^a)^b &= \frac{1}{(x^a)^{|b|}} \\ &= \frac{1}{x^{a|b|}} \\ &= x^{-a|b|} \\ &= x^{ab} \,. \end{aligned} \tag{A.14}$$

$-1 < a \leq 1$, $x \in \mathbb{C}$ und $b \in \mathbb{C}$

Für den Fall $-1 < a \leq 1$ muß man wieder Gl. (A.4) und Gl. (A.1) anwenden:

$$x^a = e^{a \ln |x| + ia \arg(x)} \,. \tag{A.15}$$

Da a reell ist, gilt

$$|x^a| = e^{a \ln |x|} \tag{A.16}$$

und

$$\arg(x^a) - a \arg(x) = 2k\pi \,. \tag{A.17}$$

Wegen Gl. (A.2) und $-1 < a \leq 1$ muß

$$-\pi < a \arg(x) \leq \pi \,, \tag{A.18}$$

also ist $k = 0$ und damit

$$\arg(x^a) = a \arg(x) \,. \tag{A.19}$$

Dieses Argument gilt z.B. nicht mehr für $a = -1$, da dann $-1 \arg(x) = -\pi$ sein kann. Berücksichtigt man, daß $a \ln|x| \in \mathbb{R}$ und für reelle y auch $\ln \exp y = y$ gilt, so folgt weiter

$$\begin{aligned} \ln(x^a) &= \ln|x^a| + i \arg(x^a) \\ &= \ln(e^{a \ln|x|}) + ia \arg(x) \\ &= a \ln|x| + ia \arg(x) \\ &= a \ln x \,. \end{aligned} \tag{A.20}$$

Damit kann man den Beweis vervollständigen:

$$\begin{aligned} (x^a)^b &= e^{b \ln x^a} \\ &= e^{ba \ln x} \\ &= x^{ab} \,. \end{aligned} \tag{A.21}$$

Anhang B

Programme

B.1 gentop

Das Programm `gentop` implementiert den in Abschnitt 3.1 beschriebenen Algorithmus zum Erzeugen aller Einloop- und Zweiloop-Basistopologien, die dann in Haupt- und Subtopologien gruppiert werden. Weiterhin werden die Topologien bestimmt, die man erhält, wenn man einen der Propagatoren kürzt, zusammen mit der notwendigen Propagatorpermutation (Abschnitt 3.1.4). Außerdem wird, falls zwei (oder mehr) äußere Beine am selben Vertex angreifen, die effektive $(n-1)$-Punkt-Topologie bestimmt (Abschnitt 3.1.5). Die erzeugten Topologien werden mit Hilfe des Pakets FeynDiagram [19] (vgl. auch Abschnitt B.8) graphisch in Form einer `eps`-Datei (Encapsulated PostScript) dargestellt und in ein automatisch erzeugtes LaTeX-Dokument eingebunden (vgl. Anhang C für einen Auszug). Alle erzeugte Topologie-Information wird in einer Datei gespeichert, so daß `gentop` nicht bei jedem Start des *xloops*-Nachfolgers ausgeführt werden muß.

B.2 coeffgen

Mit `coeffgen` können die Koeffizienten und beitragenden Gebiete der Vierfach-Integraldarstellung Gl. (1.12) von Zweiloop-Dreipunkt-Funktionen, deren Parallelraum von den 0- und 1-Komponenten aufgespannt wird, berechnet werden. Das Programm beschränkt sich nicht auf zeitartige Impulse wie in Gl. (1.3), wenn man ihm Hilfen über das Vorzeichen von Linearkombinationen der Impulskomponenten mitgibt (GiNaC kennt, im Gegensatz zu Maple, noch keinen `assume`-Befehl).

Die Dreipunkt-Funktionen können eine beliebige Anzahl von l- und k-Propagatoren (jeweils mindestens einen) und keinen, einen oder zwei gemischte $l+k$-Propagatoren haben. Zusätzlich kann eine Zählerstruktur, wie sie durch Tensor-Abzugsterme mit $j_l > 1$ oder $j_k > 1$ entstehen (Abschnitt 1.5.5), mit berücksichtigt werden. Die Linearisierungs-Transformation Gl. (1.7) kann für l und k getrennt gewählt werden. Damit lassen sich alle konvergenten und divergenten Topologien und deren Abzugsterme aus Kapitel 1 berechnen.

B.3 cancelprop

Das Kürzen von Propagatoren (Abschnitt 1.5.4) zum Reduzieren der Tensorstruktur auf Basisintegrale wird automatisch vom Modul `cancelprop` ausgeführt. Die Topologie wird

durch eine `twoloop_topology` Struktur (Abschnitt 2.4.4) repräsentiert. Zur Zeit werden nur Dreipunkt-Topologien gekürzt, da noch nicht endgültig festgelegt wurde, wie Zweipunkt-Topologien im Nachfolgepaket für *xloops* berechnet werden sollen, da auch eine direkte Berechnung der Tensor-Integrale möglich ist. Für Vierpunkt-Tensor-Integrale wurde noch keine methodische Vorarbeit geleistet. `cancelprop` kann aber nachträglich um diese Fälle erweitert werden.

B.4 subloop

Zweiloop-Zwei- und -Dreipunkt-Funktionen mit einem Zweipunkt-Untergraphen (skalare und Tensor-Integrale), die mit Hilfe von Dispersionsrelationen berechnet werden können (Abschnitt 1.5.9), sind im Modul `subloop` implementiert. Spezialfälle wie Infrarot-Divergenzen oder masselose Propagatoren im Zweipunkt-Untergraphen, deren Taylor-Entwicklung infrarot divergente Beiträge liefert, sind noch nicht implementiert.

B.5 funct

Das Modul `funct` stellt Schnittstellen zur numerischen Zweifach-Integration der verschiedenen Dreipunkt-Topologien aus Abb. 1.27 mit VEGAS [48, 62] oder einem Gauss-Verfahren, das Schnittstellen-kompatibel zu VEGAS sein muß, zur Verfügung. Da VEGAS keine Parameter außer den Koordinaten des Punktes, an dem der Integrand berechnet werden soll, übergibt, müssen alle zusätzlichen Parameter, von denen der Integrand abhängt (Tensorstufe, Massen und Impulse), vor dem Aufruf von VEGAS in Form von globalen Variablen gesetzt werden.

`funct` transformiert die VEGAS-Koordinaten $(x_1, x_2) \in]0,1[\times]0,1[$ auf die Dreiecke und Rechtecke in der (l_0, k_0)-Ebene, berechnet zu dann konkreten Werten von l_0 und k_0 die Koeffizienten aus der jeweiligen Vierfach-Integraldarstellung (entsprechend Gl. (1.12)) und bestimmt dann die Intervalle der t-Integration, die für die s-Integration verschiedene Klassen ergeben (vgl. Abschnitt 1.2 bzw. [30]). Damit kann die s-Integration analytisch ausgeführt werden, die verbleibende Integration über t wird an die `integ`-Bibliothek weitergereicht.

Für UV divergente Graphen werden automatisch entsprechende Abzugsterme berechnet. Infrarot-Divergenzen werden hier nicht automatisch erkannt, für sie muß eine spezielle Funktion aufgerufen werden. Dadurch können aber auch „nahezu“ infrarot divergente Graphen (m_6 klein, Abschnitt 1.3) regularisiert werden.

B.6 integ

Bei `integ` handelt es sich um eine Sammlung von Integrationsroutinen für die letzte analytisch ausführbare Integration (üblicherweise die t-Integration, Abschnitt 1.1.1.8) in beliebigen Grenzen. Die Integrationen müssen komplett in Real- und Imaginärteil zerlegt sein, d.h. beim Integranden wird immer nur der jeweilige Realteil betrachtet (Integrationen über Pole werden als Hauptwertintegrale aufgefaßt). Die Integrationsgrenzen sind beliebig. Es können auch uneigentliche Integrale mit oberen oder unteren Grenzen $\pm\infty$ berechnet werden, wenn `INFINITY` übergeben wird. Bei logarithmisch divergenten Integralen wird `INFINITY` als Abschneideparameter Λ interpretiert und alle Terme proportional zu $\ln\Lambda$ werden einfach weggelassen. Es liegt in der Verantwortung des `funct`-Moduls

sicherzustellen, daß sich diese Terme tatsächlich aufheben (mit GiNaC könnte diese Kontrolle durch Mitführen eines symbolischen $\ln\Lambda$ verbessert werden). Unter anderem werden folgende Integrale abgedeckt:

$$\begin{aligned}
&\text{P.V.}\int dt\,\frac{1}{t+t_0}\frac{1}{\sqrt{R_0(t)}}\ln\left|\frac{s_0(t)+(at+b)/c+\sqrt{R_0(t)}}{s_0(t)+(at+b)/c-\sqrt{R_0(t)}}\right| \\
&\text{P.V.}\int dt\,\frac{1}{t+t_0}\frac{1}{\sqrt{-R_0(t)}}\arctan\left(\frac{s_0(t)+(at+b)/c}{\sqrt{-R_0(t)}}\right) \\
&\text{P.V.}\int dt\,\frac{1}{t+t_0}\frac{1}{\sqrt{\pm R_0(t)}}
\end{aligned} \tag{B.1}$$

mit

$$\begin{aligned}
s_0(t) &= \frac{\tilde{a}t+\tilde{b}}{\tilde{c}} \\
R_0(t) &= \frac{(at+b-cs_0(t))^2+4s_0t}{c^2} = (At)^2+Bt+C\,,
\end{aligned} \tag{B.2}$$

die durch Eulersche Substitutionen [20] (mit $A>0$) und Aufspalten der Logarithmen bzw. Arkustangens auf Integrale vom Typ

$$\begin{aligned}
&\text{P.V.}\int dx\,\frac{\ln|px^2+qx+r|}{dx^2+ex+f} \\
&\text{P.V.}\int dx\,\frac{\arctan(qx+r)}{dx^2+ex+f}
\end{aligned} \tag{B.3}$$

zurückgeführt werden. Dabei werden alle relevanten Spezialfälle wie z.B. $px^2+qx+r = p(x-x_0)^2$ oder $d=0$ berücksichtigt.

B.7 qgrafconvert

`qgrafconvert` bildet die Schnittstelle zwischen dem Graphenerzeuger QGRAF [68] und *χloops*. Es analysiert die Ausgabe von QGRAF (`qlist.`) im `list = normal` Modus, um die Topologie festzustellen. Der Algorithmus ist äquivalent zu dem in Abschnitt 3.1.1 vorgestellten, mit dem Unterschied, daß die Vertexpermutation nicht zwischen inneren und äußeren Vertizes unterscheiden darf, da QGRAF innere Vertizes nicht auszeichnet. Jede Topologie muß dazu einmal im QGRAF-Format mit „anonymen“ numerierten Teilchen ausgegeben werden (`xlist.`). Die aktuelle Version von `qgrafconvert` erzeugt eine Eingabedatei für *χloops* (Aufrufe von `EvalGraph2()`), kann aber an GiNaC angepaßt werden.

B.8 qgraf2feyndiag

Mit `qgraf2feyndiag` kann die Ausgabe von QGRAF über den Umweg simulierter Aufrufe von `EvalGraph2()` (mit `qgrafconvert` erzeugt) graphisch dargestellt werden. Zum Zeichnen der Graphen wird die Bibliothek FeynDiagram [19] verwendet. Dazu muß zu jeder Topologie einmal die Lage der Vertizes und der sie verbindenden Propagatoren angegeben werden.

B.9 Maple-GiNaC-Konverter m2g

m2g konvertiert Maple-Programme nach C++. Die Konvertierung besteht im wesentlichen aus den beiden Schritten

- Analyse der syntaktischen Elemente eines Maple-Programms mit lex und yacc,
- Synthese eines C++-Programms aus den analysierten Daten.

Der Schritt der Analyse läßt sich vereinfachen, indem man auf die Standard-Unix-Hilfsprogramme `lex` und `yacc` (bzw. auf `flex` und `bison` auf GNU-Systemen) [63] zurückgreift. Im ersten Teilschritt zerlegt `flex` eine Maple-Eingabedatei in terminale Elemente der Maple-Syntax, sogenannte Tokens. Dies sind z.B. Schlüsselwörter wie `if`, `for` oder `while`, Operatoren wie `and` oder `<=`, ganze Zahlen (Folgen von Ziffern) oder Variablennamen (Folgen von Buchstaben, Ziffern und dem Unterstrich _, die nicht mit einer Ziffer beginnen und kein Schlüsselwort oder Operator sind).

Im zweiten Teilschritt werden diese Tokens von `yacc` bzw. `bison` entsprechend der Maple-Syntax analysiert. Diese Syntax muß in Form einer kontextfreien Grammatik, genau genommen einer LALR(1)[1] Grammatik [43], gegeben sein. `bison` erwartet eine Parameterdatei, die die Syntaxelemente in einer Backus-Naur-ähnlichen Form zusammen mit Anweisungen bei erfolgreicher Analyse enthält. Eine LALR(1)-Grammatik für Maple ist nicht publiziert, aber mit den Informationen in [11] kann man eine solche für die wesentlichen (und nach C++ konvertierbaren) Sprachelemente erstellen. Eine Besonderheit ist die Behandlung von Kommentaren. Kommentare werden normalerweise von `flex` gefiltert. Sollen sie jedoch in das konvertierte Programm übernommen werden, müssen sie als Quasi-Anweisung betrachtet werden. Dadurch können sie jedoch nicht mehr an beliebiger Stelle im Programm stehen.

Bei der Analyse wird die Struktur des Programms in Variablen von Type `node` festgehalten. `node` enthält einen Zeiger auf polymorphe Objekte, die von der Basisklasse `grammar` abgeleitet sind. Es gibt Klassen für binäre Operatoren (`binop`), Schleifen (`forstmt`, `whilestmt`), Bedingungen (`ifstmt`), Prozeduren (`procdecl`) etc. Nach erfolgreicher Analyse beschränkt sich die Synthese eines C++-Programms darauf, für jede Klasse eine geeignete virtuelle Ausgabefunktion zu implementieren.

Als Beispiel sei die Konvertierung einer Maple-Funktion zur Berechnung der Fakultät angegeben:

```
fact:=proc(n)
    # Prozedur zur Berechung der Fakultaet
    local i,r;
    r:=1;
    for i to n do
        r:=r*i;
    od;
    RETURN(r);
end:
```

Als Ausgabe erhält man die folgende C++-Funktion:

[1] **L**ook **A**head **L**eft **R**ecursive, 1 Symbol Look Ahead

```
ex fact(ex n)
{
    //  Prozedur zur Berechung der Fakultaet
    ex i,r;

    r=1;
    for (i=1; i<=n; ++i) {
        r=r*i;
    }
    return r;
}
```

Es gibt einige Einschränkungen, da nicht alle Maple-Sprachkonstrukte konvertiert werden können:

- Operatoren: `**`, `@`, `@@`, neutrale Operatoren `&x` (außer `&*` für `ncmul`).
- Kontrollstrukturen: Maple erlaubt eine sehr flexible Angabe der Optionen für `from`, `to` und `by` in `for`-Schleifen sowie Mischung mit dem `while`-Schlüsselwort. Davon werden nur die wichtigsten akzeptiert.
- Operationen auf Mengen: `union`, `minus` und `intersect` (da GiNaC keinen Mengentyp hat).
- Funktionaloperatoren `< | >`: Funktionen können in C++ nicht zur Laufzeit dynamisch erzeugt werden.
- Zugriff auf letzte Ergebnisse mit `"` und `""` (normalerweise nur in der interaktiven Oberfläche benutzt).
- Eingeschränkte Benutzung des Verbindungsoperators `.`: Variablen können in C++ nicht zur Laufzeit dynamisch erzeugt werden.
- Alle Variablen werden als vom Typ `ex` deklariert. Daher werden Operationen auf Zeichenketten nicht unterstützt.

B.10 GiNaC-cint

GiNaC-cint bietet die Möglichkeit, C++-Programme mit GiNaC-Erweiterungen interaktiv auszuführen. Dazu nutzt es eine spezielle Version des C++-Interpreters CINT [35]. Im folgenden ist eine kurze, erläuterte Beispielsitzung abgedruckt.

```
Welcome to GiNaC-cint (GiNaC V0.5.2)
This software is provided "as is" without any warranty.  Copyright of Cint is
owned by Agilent Technologies Japan and Masaharu Goto.  Registration is
  __,  _______  requested, at this moment, for commercial use.  Send e-mail to
 (__) *       | <MXJ02154@niftyserve.or.jp>.  The registration is free.
  ._) i N a C | The GiNaC framework is Copyright by Johannes Gutenberg Univ.,
<-------------' Germany and licensed under the terms and conditions of the GPL.
```

GiNaC-cint liest die Eingabe zeilenweise von der Tastatur, analysiert sie und übergibt sie zur Bearbeitung an CINT, sobald eine vollständige C++-, GiNaC-cint- oder Präprozessor-Anweisung eingegeben wurde. Solange die Anweisung nicht vollständig ist, wird ein anderer Prompt angezeigt.

```
GiNaC> int i
     > =5;
```

Die Header-Dateien `iostream` und `string` sind bereits automatisch eingebunden.

```
GiNaC> cout << i << endl;
5
```

Im Gegensatz zu anderen CAS müssen Symbole vor ihrer ersten Benutzung deklariert werden.

```
GiNaC> symbol x("x"),y("y");
```

GiNaC-cint unterscheidet sich vom normalen CINT vor allem dadurch, daß nach Anweisungen, die ein Objekt vom Typ `ex` zurückgeben, dieser Rückgabewert ausgegeben wird und zur weiteren Verwendung in einer Variable `Out`n zur Verfügung steht. Außerdem enthalten die Variablen `LAST`, `LLAST` und `LLLAST` die jeweils letzten drei Rückgabewerte. Rückgabewerte vom Typ `symbol`, `constant`, `function`, `power` oder `numeric` werden zwar erkannt, können aber wegen der internen Speicherverwaltung von CINT nicht ausgegeben werden.

```
GiNaC> x;

WARNING: The return value of the last expression you entered was a symbol,
constant, function, power or numeric, which cannot be safely displayed.
To force the output, cast it explicitly to type 'ex' or use 'cout',
for example (assume 'x' is a symbol):
GiNaC> ex(x);
OutX = x

GiNaC> cout << x << endl;
x

This warning will not be shown again.
```

Ableitungen, Vereinfachungen etc. können in Kapitel 2 beschrieben gebildet werden.

```
GiNaC> sin(x)*cos(y);
Out1 = sin(x)*cos(y)

GiNaC> diff(LAST,x);
Out2 = cos(x)*cos(y)

GiNaC> normal((Out1-Out2)/cos(y));
Out3 = sin(x)-cos(x)
```

Funktions-Definitionen müssen mit dem Schlüsselwort `.function` begonnen werden, da der Programmcode in einer temporären Datei bis zum Ende der GiNaC-cint-Sitzung zwischengespeichert werden muß.

```
GiNaC> .function
next expression can be a function definition
GiNaC> int fibonacci(int n) {
     >     if (n<=2) return 1;
     >     return fibonacci(n-1)+fibonacci(n-2);
     > }
creating file /tmp/ginac08803uaa
```

Auch Kontrollstrukturen wie z.B. Schleifen können benutzt werden. Die Ausführung beginnt, nachdem die letzte offene Klammer geschlossen wurde.

```
GiNaC> for (int n=1; n<10; ++n) {
     >     cout << "fibonacci(" << n << ")=" << fibonacci(n) << endl;
     > }
fibonacci(1)=1
fibonacci(2)=1
fibonacci(3)=2
fibonacci(4)=3
fibonacci(5)=5
fibonacci(6)=8
fibonacci(7)=13
fibonacci(8)=21
fibonacci(9)=34
```

GiNaC-cint wird durch Eingabe von `quit;` oder `.q` beendet. Dabei werden die temporären Dateien, in denen Funktionen gespeichert wurden, gelöscht.

```
GiNaC> quit;
removing file /tmp/ginac08803uaa
```

Anhang C

Auszug aus den erzeugten Topologien

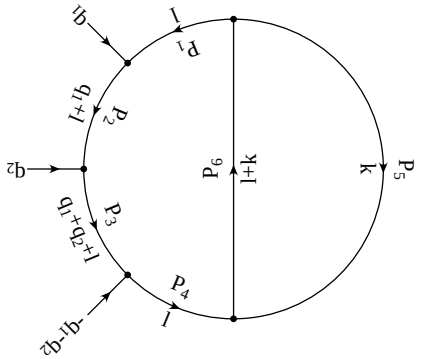
Die folgenden Seiten zeigen einen kleinen Auszug aus der graphischen Darstellung der mit dem Algorithmus aus Abschnitt 3.1 erzeugten Topologien. Alle Topologien bis Zweiloop-Vierpunkt füllen über 200 Seiten. Unter der graphischen Darstellung findet sich eine Tabelle, die angibt, welche Topologie man erhält, wenn man einen der Propagatoren kürzt, sowie die Impulstransformation und die permutierte Propagatorreihenfolge. Topologie 11 entspricht Abb. 1.28c, Topologie 12 Abb. 1.27b (planare Dreipunkt-Funktion) und Topologie 13 Abb. 1.27c (gekreuzte Dreipunkt-Funktion).

Two-Loop 3-Point (Non-Factorizing)

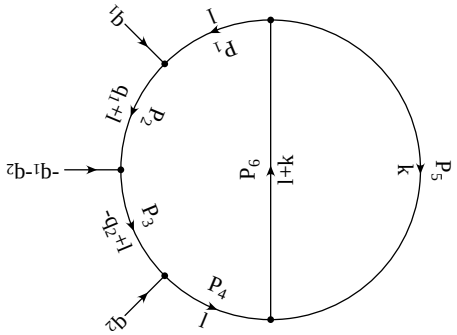
13 topologies with a total of 36 subtopologies

[...]

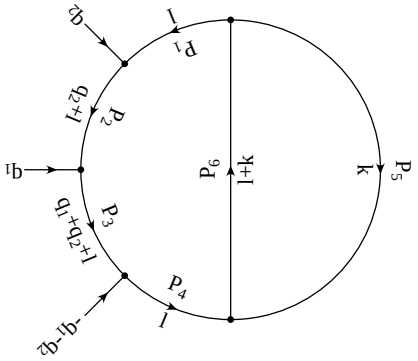
Topology 11 with 3 subtopologies



cp	type	top	sub	mom. trafo	P_1	P_2	P_3	P_4	P_5	P_6
1	2lnf3p	6	1	$l = l',\ k = k',$	—	1	2	3	4	5
2	2lnf3p	9	1	$l = l',\ k = k',$	1	—	2	3	4	5
3	2lnf3p	9	3	$l = l',\ k = k',$	1	2	—	3	4	5
4	2lnf3p	6	6	$l = -l',\ k = -k',$	3	2	1	—	4	5
5	2lf3p	9	1	$l = l',\ k = k' - l',$	1	2	3	4	—	5
6	2lf3p	9	1	$l = l',\ k = k',$	1	2	3	4	5	—

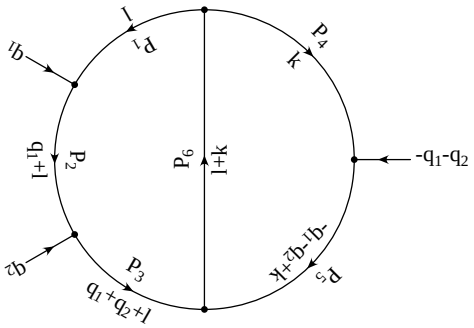


cp	type	top	sub	mom. trafo	P_1	P_2	P_3	P_4	P_5	P_6
1	2lnf3p	6	2	$l = l',\ k = k',$	—	1	2	3	4	5
2	2lnf3p	9	2	$l = l',\ k = k',$	1	—	2	3	4	5
3	2lnf3p	9	3	$l = l',\ k = k',$	1	2	—	3	4	5
4	2lnf3p	6	5	$l = -l',\ k = -k',$	3	2	1	—	4	5
5	2lf3p	9	2	$l = l',\ k = k' - l',$	1	2	3	4	—	5
6	2lf3p	9	2	$l = l',\ k = k',$	1	2	3	4	5	—

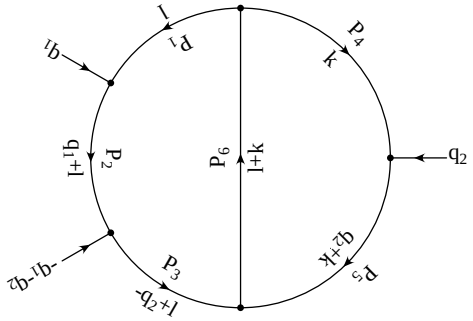


cp	type	top	sub	mom. trafo	P_1	P_2	P_3	P_4	P_5	P_6
1	2lnf3p	6	3	$l = l',\ k = k',$	—	1	2	3	4	5
2	2lnf3p	9	1	$l = l',\ k = k',$	1	—	2	3	4	5
3	2lnf3p	9	2	$l = -l',\ k = -k',$	3	2	—	1	4	5
4	2lnf3p	6	4	$l = -l',\ k = -k',$	3	2	1	—	4	5
5	2lf3p	9	3	$l = l',\ k = k' - l',$	1	2	3	4	—	5
6	2lf3p	9	3	$l = l',\ k = k',$	1	2	3	4	5	—

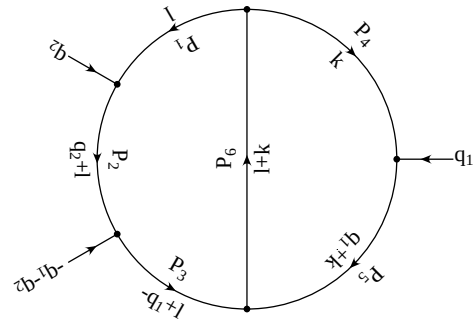
Topology 12 with 3 subtopologies



cp	type	top	sub	mom. trafo	P_1	P_2	P_3	P_4	P_5	P_6
1	2lnf3p	7	1	$l = l',\ k = k',$	—	1	2	3	4	5
2	2lnf3p	10	1	$l = l',\ k = k',$	1	—	2	3	4	5
3	2lnf3p	7	2	$l = -q_1 - q_2 - l',\ k = q_1 + q_2 - k',$	2	1	—	4	3	5
4	2lnf3p	6	4	$l = -q_1 - q_2 + l',\ k = q_1 + q_2 + k',$	1	2	3	—	4	5
5	2lnf3p	6	6	$l = -l',\ k = -k',$	3	2	1	4	—	5
6	2lf3p	10	1	$l = l',\ k = k',$	1	2	3	4	5	—

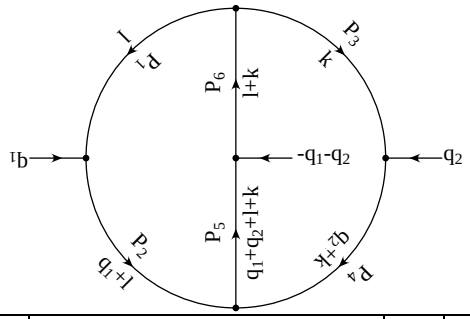


cp	type	top	sub	mom. trafo	P_1	P_2	P_3	P_4	P_5	P_6
1	2lnf3p	7	1	$l = -q_1 + k',\ k = q_1 + l',$	—	3	4	1	2	5
2	2lnf3p	10	2	$l = l',\ k = k',$	1	—	2	3	4	5
3	2lnf3p	7	3	$l = q_2 - l',\ k = -q_2 - k',$	2	1	—	4	3	5
4	2lnf3p	6	3	$l = q_2 + l',\ k = -q_2 + k',$	1	2	3	—	4	5
5	2lnf3p	6	5	$l = -l',\ k = -k',$	3	2	1	4	—	5
6	2lf3p	10	2	$l = l',\ k = k',$	1	2	3	4	5	—



cp	type	top	sub	mom. trafo	P_1	P_2	P_3	P_4	P_5	P_6
1	2lnf3p	7	2	$l = -q_2 + k',\ k = q_2 + l',$	—	3	4	1	2	5
2	2lnf3p	10	3	$l = l',\ k = k',$	1	—	2	3	4	5
3	2lnf3p	7	3	$l = -q_2 - k',\ k = q_2 - l',$	4	3	—	2	1	5
4	2lnf3p	6	1	$l = q_1 + l',\ k = -q_1 + k',$	1	2	3	—	4	5
5	2lnf3p	6	2	$l = -l',\ k = -k',$	3	2	1	4	—	5
6	2lf3p	10	3	$l = l',\ k = k',$	1	2	3	4	5	—

Topology 13 with 1 subtopology



cp	type	top	sub	mom. trafo	P_1	P_2	P_3	P_4	P_5	P_6
1	2lnf3p	7	1	$l = -q_1 - k' - l',\ k = q_1 + l',$	—	2	3	5	4	1
2	2lnf3p	7	1	$l = k' + l',\ k = -q_1 - q_2 - l',$	3	—	2	4	5	1
3	2lnf3p	7	2	$l = q_2 + l',\ k = -q_2 - k' - l',$	1	2	—	5	4	3
4	2lnf3p	7	2	$l = -q_1 - q_2 - l',\ k = k' + l',$	2	1	4	—	5	3
5	2lnf3p	7	3	$l = q_2 - l',\ k = -q_2 - k',$	2	1	4	3	—	5
6	2lnf3p	7	3	$l = -q_1 - q_2 + l',\ k = k',$	1	2	3	4	5	—

Literaturverzeichnis

[1] S. Bauberger, M. Böhm. *Simple one-dimensional integral representations for two-loop self-energies: the master diagram.* Nucl. Phys. **B445** (1995) 25.

[2] C. Bauer, A. Frink, R. Kreckel. *GiNaC — An open framework for symbolic computation within the C++ programming language.* `http://www.ginac.de`.

[3] C. Bauer, A. Frink, R. Kreckel. *GiNaC documentation.* `http://www.ginac.de/reference`.

[4] E.E. Boos, M.N. Dubinin, V.A. Ilin, A.E. Pukhov, V.I. Savrin. *CompHEP: Specialized package for automatic calculation of elementary particle decays and collisions.* Moskow State U. preprint SNUTP-94-116, hep-ph/9503280.

[5] V.I. Borodulin, R.N. Rogalyov, S.R. Slabospitsky. *CORE 2.1 (COmpendium of RElations, Version 2.1).* IHEP Protvino preprint IHEP-95-90, hep-ph/9507456.

[6] D.J. Broadhurst, J. Fleischer, O.V. Tarasov. *Two-loop two-point function with masses: asymptotic expansions and Taylor series in any dimension.* Z. Phys. **C60** (1993) 287.

[7] F.B. Brokken. *C++ annotations.* ICCE, University of Groningen. `http://www.icce.rug.nl/docs/cplusplus/cplusplus.html`.

[8] L. Brücher. *χloops-Formelsammlung.* persönliche Kommunikation, unveröffentlicht.

[9] L. Brücher, J. Franzkowski, D. Kreimer. *χloops: Automated Feynman diagram calculation.* Comput. Phys. Commun. **115** (1998) 140.

[10] R. Brun, F. Rademakers. *ROOT — An object oriented data analysis framework.* Nucl. Inst. & Meth. **A389** (1997) 81. `http://root.cern.ch`.

[11] B.W. Char et. al. *Maple V language reference manual.* Springer, New York, 1991.

[12] B.W. Char et. al. *Maple V library reference manual.* Springer, New York, 1991.

[13] M. Cline, G. Lomow, M. Girou. *C++ FAQs, 2nd Edition.* Addison-Wesley, Reading, 1998.

[14] J. Collins. *Renormalization.* Cambridge Monographs on Math. Phys., 1984.

[15] F. Constantinescu. *Distributionen und ihre Anwendung in der Physik.* Teubner, Stuttgart, 1974.

[16] A. Czarnecki, U. Kilian, D. Kreimer. *New representation of two-loop propagator and vertex functions.* Nucl.Phys. **B433** (1995) 259.

[17] A.I. Davydychev, B.J. Tausk. *Two-loop self-energy diagrams with different masses and the momentum expansion.* Nucl. Phys. **B397** (1993) 123.

[18] Debian GNU/Linux — The universal operating system. `http://www.debian.org`.

[19] W.C. Dimm. *FeynDiagram version 2.2 tutorial.* Cornell University, 1993.

[20] G.M. Fichtenholz. *Integral and differential calculus, Vol. I.* Fizmatgiz, Moskau, 1959.

[21] J. Fleischer. *Application of Padé approximants to the calculation of Feynman diagrams.* New Computing Techniques in Physics Research III (K.H. Becks und D. Perret-Gallix, Hrsg.), World Scientific, 1994.

[22] J. Fleischer. *Calculation of two-loop vertex functions from their small momentum expansion.* Int. J. Mod. Phys. **C6** (1995) 495.

[23] J. Fleischer, F. Jegerlehner, O.V. Tarasov, O.L. Veretin. *Two loop QCD corrections of the massive fermion propagator.* Nucl. Phys. **B539** (1999) 671.

[24] J. Fleischer, V.A. Smirnov, A. Frink, J. Körner, D. Kreimer, J.B. Tausk, K. Schilcher. *Calculation of infrared-divergent Feynman diagrams with zero mass threshold.* Eur. Phys. J. **C2** (1998) 747, hep-ph/9704353.

[25] J. Fleischer, V.A. Smirnov, O.V. Tarasov. *Calculation of Feynman diagrams with zero mass threshold from their small momentum expansion.* Z. Phys. **C74** (1997) 379.

[26] J. Fleischer, O.V. Tarasov. *Calculation of Feynman diagrams from their small momentum expansion.* Z. Phys. **C64** (1994) 413, hep-ph/9403230.

[27] J. Franzkowski. *Virtuelle Strahlungskorrekturen im Standardmodell der Elementarteilchenphysik.* Dissertation, Universität Mainz, 1997.

[28] Free Software Foundation, Inc. *GNU General Public License.* `http://www.fsf.org/copyleft/gpl.html`.

[29] Free Software Foundation, Inc. *GNU Lesser General Public License.* `http://www.fsf.org/copyleft/lesser.html`.

[30] A. Frink. *Massive Zwei-Loop Vertexfunktionen.* Diplomarbeit, Universität Mainz, 1996.

[31] A. Frink, U. Kilian, D. Kreimer. *New representation of the two loop crossed vertex function.* Nucl. Phys. **B488** (1997) 426, hep-ph/9610285.

[32] A. Frink, B.A. Kniehl, D. Kreimer, K. Riesselmann. *Heavy higgs lifetime at two loops.* Phys. Rev. **D54** (1996) 4548, hep-ph/9606310.

[33] J. Fujimoto, Y. Shimizu, K. Kato, T. Kaneko. *Numerical approach to two-loop three-point functions with masses.* Int. J. Mod. Phys. **C6** (1995) 525, hep-ph/9505270.

[34] K.O. Geddes, S.R. Czapor, G. Labahn. *Algorithms for computer algebra.* Kluwer, Norwell, 1995.

[35] M. Goto. *C++ interpreter — CINT.*
CQ Publishing. `http://root.cern.ch/root/Cint.html`.

[36] T. Granlund. *GMP — Arithmetic without limitations.* `http://www.swox.com/gmp`.

[37] B. Haible. *CLN: A class library for numbers.*
`http://clisp.cons.org/ haible/packages-cln.html`.

[38] R. Halin. *Graphentheorie.* Wiss. Buchges., Darmstadt, 1989.

[39] A.D. Hall jr. *The ALTRAN system for rational function manipulation - a survey.* Comm. ACM **14** (1971) 517.

[40] R. Harlander, M. Steinhauser. *Automatic computation of Feynman diagrams.* Prog. Part. Nucl. Phys. **43** (1999) 167.

[41] A.C. Hearn. *REDUCE user's manual Version 3.5.* RAND Publication, Santa Monica, 1993.

[42] G. 't Hooft, M. Veltman. *Scalar one-loop integrals.* Nucl. Phys. **B153** (1979) 365.

[43] J.E. Hopcroft, J.D. Ullman. *Introduction to automata theory, languages and computation.* Addison-Wesley, Reading, 1979.

[44] International Standard ISO/IEC 14882. *Programming languages — C++.* American National Standards Institute, New York, 1998.

[45] T. Ishikawa, T. Kaneko, K. Kato, S. Kawabata, Y. Shimizu, H. Tanaka. *GRACE manual.* KEK preprint KEK-92-19.

[46] T. Kinoshita. *Mass singularities of Feynman amplitudes.* J. Math. Phys. **3** (1962) 650.

[47] D.E. Knuth. *The art of computer programming, Vol. 3: Sorting and searching.* Addison-Wesley, Reading, 1973.

[48] R. Kreckel. *Parallelization of adaptive MC integrators.* Comp. Phys. Commun. **106** (1998) 258.

[49] R. Kreckel. Dissertation, Universität Mainz, in Vorbereitung.

[50] R. Kreckel, D. Kreimer, K. Schilcher. *First results with a new method for calculating two loop box function.* Eur. Phys. J. **C6** (1999) 693.

[51] D. Kreimer. *Dimensional regularization in the standard model.* Dissertation, Universität Mainz, 1992.

[52] D. Kreimer. *Tensor integrals for two loop standard model calculations.* Univ. of Tasmania preprint UTAS-PHYS-93-40, hep-ph/9312223.

[53] D. Kreimer. *Test der dimensionalen Reduktion in masseloser QED und QCD.* Diplomarbeit, Universität Mainz, 1989.

[54] D. Kreimer. *The master two-loop two-point function. The general case.* Phys. Lett. **B273** (1991) 277.

[55] D. Kreimer. *The role of gamma(5) in dimensional regularization.* Univ. of Tasmania preprint UTAS-PHYS-94-01, hep-ph/9401354.

[56] D. Kreimer. *The two-loop three-point functions: general massive cases.* Phys. Lett. **B292** (1992) 341.

[57] J. Küblbeck, M. Böhm, A. Denner. *Feyn Arts: Computer algebraic generation of Feynman graphs and amplitudes.* Comput. Phys. Commun. **60** (1990) 165.

[58] L.D. Landau. *On analytic properties of vertex parts in quantum field theory.* Nucl. Phys. **13** (1959) 181.

[59] B.W. Lee, C. Quigg, H.B. Thacker. *Weak interactions at very high energies: The role of the Higgs-boson mass.* Phys. Rev. **D16** (1977) 1519.

[60] T.D. Lee, M. Nauenberg. *Degenerate systems and mass singularities.* Phys. Rev. **133** (1964) B1549.

[61] G. Leibbrandt. *Introduction to the technique of dimensional regularization.* Rev. Mod. Phys. **47** (1975) 849.

[62] G.P. Lepage. *VEGAS - An adaptive multi-dimensional integration program.* Cornell University Preprint CLNS-80/447, Ithaca, 1980.

[63] J.R. Levine, T. Mason, D. Brown. *lex & yacc.* O'Reilly, 1992.

[64] L. Lewin. *Polylogarithms and associated functions.* North Holland, New York, 1981.

[65] K. Mehlhorn. *Data structures and algorithms 1: Sorting and searching.* Springer, Berlin, 1984.

[66] R. Mertig, M. Böhm, A. Denner. *Feyn Calc: Computer algebraic calculation of Feynman amplitudes.* Comput. Phys. Commun. **64** (1991) 345.

[67] U. Nierste, D. Müller, M. Böhm. *Two loop relevant parts of D-dimensional massive scalar one loop integrals.* Z. Phys. **C57** (1993) 605.

[68] P. Nogueira. *Automatic Feynman graph generation.* J. Comput. Phys. **105** (1993) 279.

[69] W. Oevel, F. Postel, G. Rüscher, S. Wehmeier. *Das MuPAD Tutorium.* Springer, Berlin, 1999.

[70] G.J. van Oldenborgh. *FF - a package to evaluate one-loop Feynman diagrams.* Rep. No. NIKHEP-H/90-15, Amsterdam, 1990.

[71] G. Passarino, M. Veltman. *One-loop corrections for e^+e^- annihilation into $\mu^+\mu^-$ in the Weinberg model.* Nucl. Phys. **B160** (1979) 151.

[72] W. Pauli, F. Villars. *On the invariant regularization in relativistic quantum theory.* Rev. Mod. Phys. **21** (1949) 434.

[73] M. Pepe Altarelli. *Higgs searches and prospects from LEP2.* ALEPH Kollaboration, hep-ex/9904006.

[74] A.J. Perlis et al. *An extension of ALGOL for manipulating formulae.* CACM **7(2)** (1964).

[75] P. Post. *Elektromagnetische und schwache Vektorformfaktoren des pseudoskalaren Mesonoktetts zur Ordnung p^6 der Chiralen Störungstheorie.* Dissertation, Universität Mainz, 1997.

[76] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery. *Numerical Recipes in C, 2nd Edition.* Cambridge University Press, 1992.

[77] A. Read. *Searches for Higgs bosons: Preliminary combined results from the four LEP experiments at $\sqrt{s} \approx 189$ GeV.* ALEPH 99-081 CONF 99-052, DELPHI 99-142 CONF 327, L3 Note 2242, OPAL Technical Note TN-614.

[78] E. Remiddi. *Differential equations for Feynman graph amplitudes.* Nuovo Cim. **A110** (1997) 1435.

[79] D. Rusin. persönliche Kommunikation in `news:sci.math.symbolic`.

[80] R. Scharf. Diplomarbeit, Universität Würzburg, 1991.

[81] A.V. Semenov. *LanHEP: A package for automatic generation of Feynman rules from the Lagrangian.* Comput. Phys. Commun. **115** (1998) 124.

[82] T.K. Shi, W.-H. Steeb. *SymbolicC++: An introduction to computer algebra using object-oriented programming.* Springer, Singapur, 1998.

[83] E.R. Speer. *Analytic renormalization.* J. Math. Phys. **9** (1968) 1404.

[84] M. Spira, A. Djouadi, D. Graudenz, P.M. Zerwas. *Higgs boson production at the LHC.* Nucl. Phys. **B453** (1995) 17.

[85] J. Stoer, R. Burlisch. *Einführung in die numerische Mathematik II.* Springer, Berlin, 1973.

[86] B. Stroustrup. *The C++ programming language.* Addison-Wesley, Reading, 1997.

[87] A. Strzebonski. persönliche Kommunikation in `news:sci.math.symbolic`.

[88] H. Veltman. *The equivalence theorem.* Phys. Rev. Lett. **D41** (1990) 2294.

[89] Waterloo Maple Inc. *Surpassing the Limits of Technical Computation.* `http://www.maplesoft.com/papers/papers_004.html`.

[90] G. Weiglein, R. Scharf, M. Böhm. *Reduction of general two loop selfenergies to standard scalar integrals.* Nucl. Phys. **B416** (1994) 606.

[91] M. Wester. *A review of CAS mathematical capabilities.* Applied Mechanics in the Americas, Vol. III. American Academy of Mechanics and Asociacion Argentina de Mecanica Computacional, Santa Fe, 1995.

[92] S. Wolfram. *Mathematica: a system for doing mathematics by computer, 2nd Edition.* Addison-Wesley, 1991.

[93] D.R. Yennie, S.C. Frautschi, H. Suura. *The infrared divergence phenomena and high-energy processes.* Ann. Phys. **13** (1961) 379.