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

Automatisierte Berechnung von Feynman-Graphen

Jens Vollinga
geb. in Mainz

Mainz, den 29. August 2005

Datum der mündlichen Prüfung: 3. November 2005
D77 (Diss. Universität Mainz)

Inhaltsverzeichnis

1 Vorwort

Die Erforschung der kleinsten Bausteine der Natur erfährt seit Jahrzehnten einen stetigen Fortschritt. Die theoretischen Modelle zur Beschreibung der Elementarteilchen gestatten uns immer genauere physikalische Vorhersagen. Wir besitzen inzwischen mit dem sogenannten Standardmodell eine äußerst erfolgreiche Theorie zur Beschreibung der Natur, deren Vorhersagen mit beindruckender Präzision durch Experimente bestätigt werden konnten. Ein wesentlicher Bestandteil des Standardmodells ist die Existenz des Higgs-Teilchens. Das Higgs-Teilchen aber wurde bisher noch nicht experimentell nachgewiesen. Das Ergebnis der Suche danach wird das Standardmodell entweder ein weiteres Mal bestätigen oder es widerlegen. Die dafür nötigen experimentellen Anstrengungen sind zurzeit mit dem Bau des *Large Hadron Colliders* LHC in vollem Gange.

Auch auf dem Gebiet der Theorie wird zu diesem Zweck erhebliche Arbeit geleistet. Das Berechnen von Standardmodell-Observablen ist sehr aufwändig und schwierig. Die Probleme sind teilweise so gewaltig, dass die Verbesserung einer Berechnung um eine Präzisionsordnung ähnlich lange Zeiträume benötigt, wie zwischen dem Bau bedeutender Beschleunigerexperimente verstreicht. Für die Analyse der LHC-Daten müssen noch eine Reihe von Rechnungen fertiggestellt werden. Ein Grund dafür ist, dass neue mathematische Methoden, die zur Lösung der Formeln erforderlich wären, noch nicht, oder nur teilweise entwickelt sind. Hinzu kommt, dass trotz einer einfachen Strukturierbarkeit der Berechnungsschritte durch sogenannte Feynman-Diagramme, der Umfang und die Detailfülle der Berechnungen so enorm groß wird, dass zur Beherrschung dieser Komplexität mehr und mehr Aufgaben an Computerprogramme abgegeben werden müssen. Die Automatisierung der Berechnungen wird zwingend. Im Hinblick auf den LHC, aber auch auf zukünftige Experimente sind daher zwei Punkte wichtig: zum einen die Entwicklung neuer mathematischer Methoden, und zum anderen die Automatisierung der Berechnungen. Beide Themen bilden den Schwerpunkt dieser Arbeit.

In Kapitel 2 skizzieren wir zunächst den Weg vom theoretischen Modell zur experimentell überprüfbaren Vorhersage von Observablen. Danach geben wir einen Überblick über den aktuellen Stand der Berechnungen und einen Einblick in deren Automatisierung. Die nachfolgenden Kapitel beschreiben dann Arbeiten zu einzelnen Aspekten dieser automatisierten Berechnung. In Kapitel 3 präsentieren wir ein Projekt, das ganz nah an der Anwendung für den LHC steht, und die automatisierte Berechnung noch fehlender Beiträge zu wichtigen Observablen zum Inhalt hat. Kapitel 4 beschäftigt sich anschließend mit neuen analytischen Methoden zur Berechnung von Feynman-Diagrammen. Damit eng verbunden ist das Thema von Kapitel 5. Es befasst sich mit der numerischen Auswertung bestimmter Funktionen, die für diese analytischen Methoden wichtig werden.

2 Automatisierte Berechnung von $|\mathcal{M}|^2$

In diesem Kapitel skizzieren wir zunächst die Verbindung zwischen den theoretischen Modellen und den Experimenten in der Elementarteilchenphysik. Wir beschränken uns dabei auf die störungstheoretische Beschreibung von Streuexperimenten. Die Darstellung ist stark vereinfacht und geht nicht auf die mathematische Begründung der feldtheoretischen Konzepte und der Störungstheorie ein. Im Rahmen der Störungstheorie ist das zentrale Element der Verbindung zwischen Theorie und Experiment das invariante Matrixelement $\mathcal{M}$. Nach einer Übersicht über den aktuellen Stand bei der Berechnung von $\mathcal{M}$ im Kontext verschiedener experimenteller Anwendungen, diskutieren wir die Automatisierung dieser Berechnungen und motivieren damit die Arbeiten in den folgenden Kapiteln.

2.1 Vom Modell zur theoretischen Vorhersage

Die theoretischen Modelle zur Beschreibung von Elementarteilchen werden als Quantenfeldtheorien formuliert. Quantenfeldtheorien vereinen das Konzept eines Feldes, durch das sich Teilchenerzeugung und -vernichtung beschreiben lässt, mit dem der Quantisierung. Jedes konkrete Modell wird vollständig durch eine Lagrange-Dichte $\mathcal{L}$ kodiert, aus der sich der physikalische Inhalt der Theorie ergibt. Wir betrachten im folgenden die Quantenelektrodynamik (QED) als Beispiel. Die QED wird durch die Lagrange-Dichte

$$\mathcal{L} = \bar{\psi}(i\not{\partial} - m\mathbb{1} - e\not{A})\psi - \tfrac{1}{4}(\partial_\mu A_\nu - \partial_\nu A_\mu)^2 \tag{2.1}$$

beschrieben. Dabei ist ψ der Feldoperator für das Elektron bzw. Positron[1] und A_μ beschreibt das Photon. Die durchgestrichenen Größen sind nach üblicher Konvention mit Dirac-Matrizen γ^μ kontrahiert. m und e sind Koeffizienten, die Masse und Ladung der Elektronen festlegen. Hier und im folgenden sind die physikalischen Einheiten zu $c = \hbar = 1$ gewählt. Die Lagrange-Dichte kann man in zwei Komponenten unterteilen:

$$\mathcal{L} = \mathcal{L}_0 + \mathcal{L}_I \,. \tag{2.2}$$

Der erste Term beschreibt die freien Felder, der zweite Term beschreibt die Wechselwirkung (*interaction*) zwischen den Felder. Für den Fall (2.1) hat man $\mathcal{L}_I = -e\bar{\psi}\not{A}\psi$.

Mit Hilfe der Euler-Lagrange-Gleichungen lassen sich aus $\mathcal{L}_0$ die bekannten freien Bewegungsgleichungen, wie z.B. die Dirac-Gleichung, ableiten. Lösungen für die Bewegungsgleichungen der vollständigen Lagrange-Dichte sind aber im allgemeinen nicht bekannt. Sofern die Eigenschaften von Elementarteilchen durch Streuexperimente untersucht werden, genügt die Berechnung von Übergangswahrscheinlichkeiten zwischen zwei Zuständen.

[1] Wir betrachten nur die „einfache“ QED mit Elektronen, Positronen und Photonen. Die anderen elektromagnetisch wechselwirkenden Teilchen ignorieren wir der Kürze wegen.

Die dazu entwickelte Streutheorie nimmt einen Zweiteilchen-Anfangszustand $|k_1 k_2\rangle_{\text{in}}$ und einen Mehrteilchen-Endzustand ${}_{\text{out}}\langle p_1 p_2 \cdots|$ an, die jeweils Teilchen beschreiben, die in einer unendlich weit entfernten Vergangenheit bzw. Zukunft definierte Impulse $k_1, k_2, p_1, \ldots$ besitzen. Es wird angenommen, dass der Zusammenhang zwischen diesen asymptotischen Zuständen und den Wechselwirkungszuständen, die zu einer bestimmten, endlichen Referenzzeit definiert sind, durch den Zeitentwicklungsoperator e^{-iHt} gegeben ist, z.B.

$$|k_1 k_2\rangle_{\text{in}} = \lim_{t\to-\infty} e^{iHt}|k_1 k_2\rangle \,. \tag{2.3}$$

Der Hamilton-Operator H lässt sich aus der Lagrange-Dichte bestimmen und enthält alle Wechselwirkungsterme, die bei der Streuung beitragen. Damit bekommt man

$${}_{\text{out}}\langle p_1 p_2 \cdots|k_1 k_2\rangle_{\text{in}} = \lim_{\substack{t_+\to\infty \\ t_-\to-\infty}} \langle p_1 p_2 \cdots|e^{-iH(t_+-t_-)}|k_1 k_2\rangle = \langle p_1 p_2 \cdots|\mathcal{S}|k_1 k_2\rangle \,. \tag{2.4}$$

Die für die Streutheorie wichtige $\mathcal{S}$-Matrix wird mit (2.4) als zeitliche Grenzwerte des Zeitentwicklungsoperators definiert. Alle Information über die Physik der Streuung sind in dieser Größe enthalten. Der Beitrag zu $\mathcal{S}$ für „Streuung“ ohne Wechselwirkung sowie die Impulserhaltung separiert man aus der $\mathcal{S}$-Matrix ab

$$\mathcal{S} = \mathbb{1} + i(2\pi)^4\delta^{(4)}\big(k_1+k_2-\textstyle\sum p_i\big)\,\mathcal{M} \tag{2.5}$$

und definiert so das invariante Matrixelement $\mathcal{M}$. Die Berechnung von $\mathcal{M}$ ermöglicht eine komplette theoretische Vorhersage des Streuprozesses. Wir werden später sehen, wie man aus $\mathcal{M}$ messbare Größen, wie zum Beispiel den Wirkungsquerschnitt, berechnet.

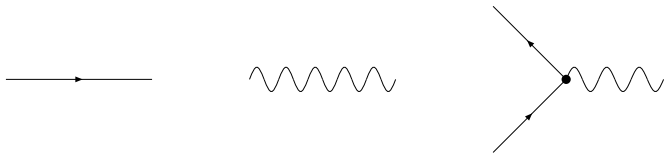
Im allgemeinen ist die Wechselwirkung schwach und $\mathcal{L}_I$ kann deshalb als Störung behandelt werden. Die Stärke der Wechselwirkung wird durch eine oder mehrere Kopplungskonstanten beschrieben; im Fall der QED ist das e. Die Kopplungskonstante e ist klein genug, um den Zeitentwicklungsoperator in eine Reihe in Potenzen von e entwickeln zu können. Die Entwicklung kann man aus dem formalen Zusammenhang der $\mathcal{S}$-Matrix mit der Wechselwirkungs-Lagrange-Dichte

$$\mathcal{S} = T\exp\Big[i\int \mathrm{d}^4x\,\mathcal{L}_I\Big] \tag{2.6}$$

herleiten. Der Operator T ordnet alle folgenden Operatoren der Zeit nach. Die Exponentialfunktion kann als Reihe aufgeschrieben und Term für Term ausgerechnet werden. Dabei werden im wesentlichen die Erzeugungs- und Vernichtungsoperatoren der Feldoperatoren auf alle mögliche Arten kombiniert. Die Rechnung so auszuführen ist in der Praxis kompliziert und unübersichtlich. Von Feynman wurde ein vereinfachtes Verfahren entwickelt, das diese Rechnung in eine anschauliche und systematische Form bringt. Die bei der Reihenentwicklung auftretenden Terme ordnet man dabei Diagrammen zu. Die explizite Rechnung kann dann durch einfache Regeln für den Umgang mit diesen Feynman-Diagrammen ersetzt werden.

Für die Berechnung von $\mathcal{M}$ mit Hilfe von Feynman-Diagrammen müssen nur alle möglichen Diagramme gezeichnet werden, die sich aus wenigen Grundbausteinen bilden lassen. Die Diagramme werden zusammengesetzt aus Linien (Propagatoren), die die wechselwirkungsfreie Ausbreitung der Teilchen in Raum und Zeit beschrieben und Vertizes, die elementare

Wechselwirkungsprozesse der Teilchen beschreiben. Für die QED hat man zum Beispiel die drei Elemente:



Das linke Element steht für die freie Bewegung von Elektronen bzw. Positronen, das mittlere für die Ausbreitung von Photonen und das letzte ist die elementare Wechselwirkung, mit der sich die Linien zu einem Graphen verbinden lassen. Je nach der Ordnung der Störungsreihe, die man berechnen möchte, müssen nur die Diagramme mit einer entsprechenden Anzahl von Wechselwirkungspunkten gezeichnet werden. Hat man zum Beispiel jeweils ein Elektron und Positron im Anfangs- und im Endzustand, dann kann man Diagramme wie in Abbildung 2.1 bilden. Bei der praktischen Aufstellung aller Diagramme sind noch weitere Details zu beachten, die wir hier aber nicht diskutieren.

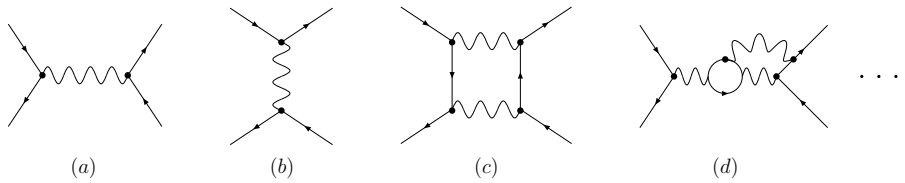


Abbildung 2.1: Mögliche Feynman-Diagramme für einen $2 \to 2$-Prozess, (a) und (b) beschreiben Beiträge der führenden Ordnung, (c) und (d) sind Beispiele für die nächstführenden Ordnungen in der Störungsreihen-Entwicklung.

Hat man alle Diagramme gezeichnet, kann man ihnen nach den soganannten Feynman-Regeln mathematische Formeln zuordnen. Diese Feynman-Formeln ergeben sich aus der Lagrange-Dichte und lassen sich für bekannte Theorien in vielen Lehrbüchern nachschlagen. Das Matrixelement $\mathcal{M}$ erhält man, indem man die Ausdrücke für die verschiedenen Diagramme addiert. Die Ausdrücke enthalten im wesentlichen Dirac-Matrizen γ^μ, Spinoren u, v für die ein- und auslaufenden Fermionen, Polarisationsvektoren ϵ^μ für die äußeren Photonen und Impulse. Jeder Linie wird ein Impuls zugeordnet, wobei an jedem Vertex Impulserhaltung gilt. Der Impuls von inneren Linien ist dann entweder durch die Impulse der ein- und auslaufenden Teilchen festgelegt, oder bleibt unbestimmt. Der letztere Fall tritt auf, wenn das Diagramm eine oder mehrere Schleifen besitzt. Dann ist über die inneren, unbestimmten Impulse zu integrieren.

Der nächste Schritt ist die Renormierung. Die Parameter in der Lagrange-Dichte entsprechen bei Verwendung der Störungsreihe nicht mehr unbedingt den physikalischen Größen, wie Masse oder Ladung. In Abbildung 2.1 kann man sich das anschaulich klarmachen. Wären nur die beiden Graphen (a) und (b) zu berechnen, könnte man die Ladung als elektromagnetische Kopplungskonstante mit dem Koeffizient e der Wechselwirkungspunkte identifizieren. Nimmt man weitere Diagramme wie (c) oder (d) für die störungstheoretische Entwicklung hinzu, dann ist diese Identifikation nicht mehr so einfach möglich, weil nun Diagramme mit einer unterschiedlichen Anzahl von Wechselwirkungspunkten kombiniert werden. Entsprechend findet man Diagramme, die den Zusammenhang zwischen dem

Parameter m der Lagrange-Dichte und der gemessenen Elektronmasse m modifizieren. Am Ende der Rechnung ist also nicht klar, welche Werte man für e oder m einzusetzen hat. Andererseits kennt man die Werte für Masse und Ladung durch das Experiment. Die Verknüpfung der Vorfaktoren in der Lagrange-Dichte mit physikalischen Größen trägt den Namen Renormierung. Details finden sich in Lehrbüchern wie [1]. Die Renormierung beseitigt auch Probleme mit Divergenzen, die bei der Berechnung der Schleifenintegrale entstehen können. Bei der Renormierung hat man gewisse Freiheiten, die Parameteridentifikation unterschiedlich zu wählen. Deshalb muss man sich bei einem Vergleich von berechneten Werten immer auf ein Renormierungsschema einigen. Renormierungsschemata sind z.B. das MS-Schema oder das *on-shell*-Schema.

Das Matrixelement $\mathcal{M}$ ist nun bestimmt und die nächste Teilaufgabe ist es, aus $\mathcal{M}$ Observable auszurechnen. Die Experimente, mit denen die Eigenschaften von Elementarteilchen untersucht werden, sind im wesentlichen Zählexperimente. Hat man Teilchen mit einer gewissen Wahrscheinlichkeit in einem Detektor identifiziert, rekonstruiert man das Ereignis und erhöht die Zählung des entsprechenden Ereignisses. Ist die Luminosität des Teilchenstrahls bekannt, weiss man also wieviele Teilchen pro Zeitintervall und Strahlquerschnitt zur Kollision gebracht werden können, dann kann man daraus den totalen Wirkungsquerschnitt σ bzw. den differenziellen Wirkungsquerschnitt bestimmen. Diesen gemessenen Wert möchte man nun mit einer theoretischen Vorhersage vergleichen. Die Formel für den differenziellen Wirkungsquerschnitt lautet:

$$\mathrm{d}\sigma = \frac{1}{4\sqrt{(p_1p_2)^2 - m_1^2m_2^2}}\,|\mathcal{M}|^2\,\delta^{(4)}(p_1+p_2-\textstyle\sum_{i=1}^{n} q_i)\,\frac{\mathrm{d}^3q_1}{(2\pi)^3 2E_{q_1}}\cdots\frac{\mathrm{d}^3q_n}{(2\pi)^3 2E_{q_n}}\,. \qquad (2.7)$$

Die einlaufenden Teilchen haben die Impulse $p_{1,2}$ und Massen $m_{1,2}$. Die auslaufenden Teilchen haben die Impulse $q_1, \ldots, q_n$ und die Energien $E_{q_1}, \ldots, E_{q_n}$. Zur Bestimmung des Wirkungsquerschnitts sind noch alle Integrationen über die auslaufenden Impulse auszuführen. Die Integration nennt man Phasenraum-Integration. Die Grenzen der Integration hängen von dem messtechnisch abdeckbaren Bereich um den Kollisionspunkt im Detektor ab. Der Weg von der Theorie zum experimentellen Vergleich ist damit nun beschrieben.

2.2 Status der störungstheoretischen Berechnungen von $|\mathcal{M}|^2$

Die Bestimmung des invarianten Matrixelements $\mathcal{M}$ ist die zentrale Aufgabe bei der Berechnung von Observablen. Die Genauigkeit der Berechnungen hängt dabei von der Ordnung ab, bis zu der die Störungsreihe entwickelt wird. Der Maßstab ist die Genauigkeit der experimentellen Messungen. Fortschritte bei der Messtechnik und bei der Erhöhung der Luminosität machen so die Berechnung von immer höheren Schleifenordnungen notwendig. Höhere Schwerpunktsenergien ermöglichen zudem die Erzeugung einer größeren Anzahl von Teilchen im Detektor. Daher wird auch die Berücksichtigung von Feynman-Diagrammen mit vielen äußeren Teilchen notwendig.

Das Problem bei der Berechnung von Feynman-Diagrammen mit Schleifen ist das Lösen der zugehörigen Integrale. Die Integrale sind sehr anspruchsvoll und man benötigt eine Vielzahl von speziellen Methoden. Jede zusätzliche Schleife in einem Diagramm macht eine weitere Impuls-Integration notwendig. Dabei vervielfacht sich die mathematische Heraus-

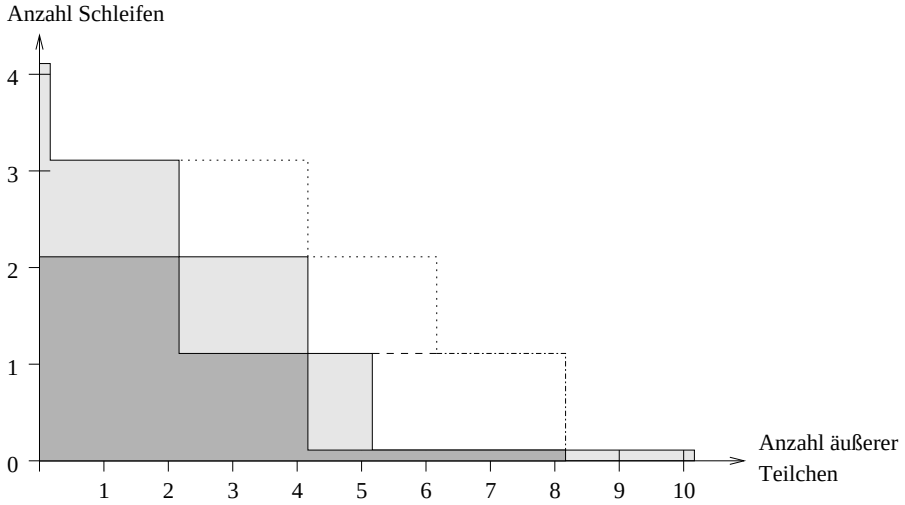


Abbildung 2.2: Status der Berechnungen von Feynman-Diagrammen [3]. Markiert sind die Fälle, die vollständig berechnet sind (dunkelgrau), die teilweise berechnet sind (hellgrau), oder für den LHC (gestrichelt) oder ILC (gepunktet) noch zu rechnen sind.

forderung jedesmal. Schon bei einer kleinen Zahl von Schleifen kommt man an die Grenze des praktisch Möglichen.

Die Berechnung von Prozessen mit vielen äußeren Teilchen ist auf eine andere Weise kompliziert. Die Anzahl der Diagramme, die für einen Prozess zu berücksichtigen sind, wächst ungefähr mit der Fakultät der Anzahl der äußeren Teilchen [2]. Gleichzeitig können die einzelnen Beiträge der Diagramme, z.B. im Fall von Gluon-Selbstwechselwirkungen in der QCD, große algebraische Ausdrücke ergeben. Insgesamt ist der Umfang der Berechnung durch die Anzahl der Terme und der nötigen mathematischen Umformungen so groß, dass selbst moderne Computer schnell überfordert werden. Die numerische Stabilität bei der Auswertung der Ergebnisse ist meist noch ein weiterer Problempunkt.

Eine Übersicht über den aktuellen Stand bei den Berechnungen gibt Abbildung 2.2 [3]. In der Abbildung ist die Anzahl der Schleifen eines Diagramms gegen die Anzahl der äußeren Beine aufgetragen. Die dunkelgrau unterlegten Bereiche kennzeichnen die Fälle, für die Rechnungen bekannt und etabliert sind. Für die hellgrau unterlegten Bereiche sind Teillösungen und Spezialfälle bekannt.

Die vorhandenen Rechnungen genügen für vergangene Experimente wie LEP2 oder aktuelle wie Tevatron. Die resultierende Präzision ist ausreichend. In Zukunft sind aber weitere Experimente geplant: in naher Zukunft wird der *Large Hadron Collider* LHC in Betrieb gehen und für ein paar Jahre später ist der internationale Linearbeschleuniger ILC geplant. Die vorhandenen Rechnungen genügen dann nicht mehr. Die nötigen neuen Rechnungen sind in Abbildung 2.2 als gestrichelte oder punktierte Linien eingezeichnet. Der LHC benötigt vor allem die Berechnung von Einschleifen-Integralen mit mehr als fünf äußeren Teilchen, denn als Proton-Proton-Beschleuniger erzeugt der LHC bei jeder Kollision sehr viele Teilchen. Gleichzeitig sind die QCD-Prozesse skalenabhängig und erst die Berechnung der nächsten Ordnung der Störungsreihe (*next-to-leading order* (NLO) Korrekturen) ergibt die nötige Präzision. Der ILC wird zwar weniger Teilchen in einem Detektor er-

zeugen, fordert aber durch die „sauberere“ Physik einer Elektron-Positron-Maschine eine noch viel höhere Präzision.

Abbildung 2.2 gibt ungefähr den aktuellen Stand wieder, ignoriert aber einige wichtige Details. Wichtig bei der Berechnung von Feynman-Integralen ist nämlich auch die Anzahl von Massen- und Energieparametern im Integral. Je mehr Massenskalen vorhanden sind, desto schwieriger ist die Auswertung. Für viele der in Abbildung 2.2 gezeigten Rechnungen muss die Einschränkung auf eine der zwei Massenskalen gemacht werden. Gerade bei der elektroschwachen Wechselwirkung sind aber unter Umständen mehr Massenskalen nötig.

Die Behandlung von Divergenzen mit Hilfe der dimensionalen Regularisierung liefert als Ergebnis für ein Diagramm eine Laurent-Reihe in Potenzen des Regularisierungsparameters ϵ, die je nach dem Grad der Divergenzen negative Potenzen von ϵ enthält. Je höher die Schleifenordnung ist, desto höher ist der Divergenzgrad in ϵ. Multipliziert man für die Berechnung von $|\mathcal{M}|^2$ die Reihen von zwei Diagrammen, dann muss je nach den negativen Exponenten von ϵ einer Reihe, die andere Reihe zu höheren Ordnungen von ϵ berechnet werden, um den endlichen Beitrag zu erhalten. Letztlich hat man also bei der Berechnung von Prozessen mit einer höheren Schleifenordnung auch das Problem, die Ergebnisse für die Diagramme in höhere Ordnungen von ϵ berechnen zu müssen. Dieses Problem ist hochgradig nicht-trivial und nur für Spezialfälle gelöst.

Zusammenfassend kann man sagen, dass zunächst die Entwicklung von NLO-Korrekturen für die beim LHC relevanten Prozesse wichtig ist. Es werden Lösungen benötigt, die bei der praktischen Analyse der LHC-Daten benutzt werden können. Im weiteren müssen höhere Schleifenordnungen berechnet werden. Dafür können bekannte Methoden verwendet werden, allerdings nimmt der Rechenaufwand erheblich zu und stellt ein eigenes, organisatorisches Problem dar. Für die Behandlung von Integralen mit mehreren Massenskalen sowie der Entwicklung der Lösungen nach höheren Ordnungen in ϵ müssen aber noch neue Ideen und Methoden entwickelt werden.

2.3 Automatisierung

Neben den prinzipiellen Schwierigkeiten Feynman-Integrale zu lösen, ist im vorigen Abschnitt das Problem der wachsenden Komplexität der Rechnungen angesprochen worden. Die große Anzahl von beitragenden Feynman-Diagrammen macht die Berechnungen zu umfangreich, um sie noch von Hand ausführen zu können. Die Hilfe von Computern wird notwendig.

Computer werden schon seit langem bei der Ausführung der Berechnungen verwendet. Anfangs erfolgte dies lediglich, um die Formeln numerisch auszuwerten, später wurden Programme geschrieben, die aus den Ergebnissen für die verschiedenen Diagramme automatisch $|\mathcal{M}|^2$ berechnen. Daneben werden bei der Lösung von Integralen Computer zunehmend als unterstützendes Hilfsmittel eingesetzt. Diese Arten der Berechnungs-Automatisierung haben inzwischen einen hohen Entwicklungsstand erreicht. Der im ersten Abschnitt beschriebene Weg von der Theorie zur experimentell messbaren Observablen ist in Abbildung 2.3 nochmal schematisch dargestellt. Die wesentlichen Rechenschritte sind in ihrer natürlichen Abfolge gezeigt. Im folgenden geben wir kurz einen Eindruck von der Automatisierung dieser einzelnen Schritte.

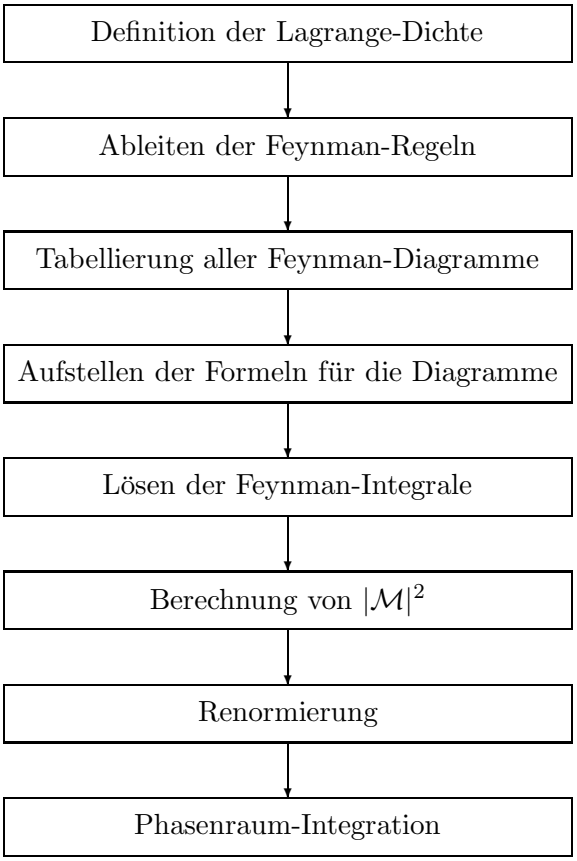


Abbildung 2.3: Ablauf der Berechnung einer Observablen zusammengefasst in den wesentlichen Rechenschritten.

Definition der Lagrange-Dichte. Dieser Schritt lässt sich nur schwer automatisieren, da hier die ganze menschliche Intelligenz und Kreativität gefragt ist. Computer werden hier nur als Werkzeuge genutzt, z.B. für wiederkehrende Operationen wie die Symmetrisierung einzelner Terme der Lagrange-Dichte.

Ableiten der Feynman-Regeln. Das Ableiten der Feynman-Regeln aus der jeweiligen Lagrange-Dichte kann automatisiert werden (siehe z.B. [4]). Benötigt man die Feynman-Regeln für viele verschiedene Modelle, oder besitzt die Lagrange-Dichte viele Terme, dann kann dies eine erhebliche Erleichterung sein.

Tabellierung aller Feynman-Diagramme. Im Gegensatz zum vorigen Punkt ist die Automatisierung der Feynman-Diagramm-Erzeugung nicht nur eine Erleichterung, sondern eine Notwendigkeit, um Prozesse mit vielen äußeren Teilchen oder mit höheren Schleifenordnungen berechnen zu können. Die Anzahl der Diagramme kann viele tausend übersteigen und ist deshalb nicht mehr per Hand tabellierbar. Die Erstellung der Diagramme ist ein weitgehend kombinatorisches Problem, das als gelöst gelten kann. Es existieren daher auch schon eine Reihe von gut funktionierenden Softwareprogrammen [5, 6, 7], die diese Aufgabe automatisiert erledigen.

Aufstellen der Formeln für die Diagramme. Mit Hilfe der Feynman-Regeln lässt sich

jedem Diagramm ein mathematischer Ausdruck zuordnen. Dieser Rechenschritt ist einfach, weil nur zwei Datenmengen, die Liste der Feynman-Formeln und die Liste der Diagramme, nach einfachen Regeln miteinander verknüpft werden müssen. Alle im vorigen Abschnitt zitierten Programme erledigen diese Aufgabe automatisch. Problematischere Fälle mit supersymmetrischen Teilchen [8] können ebenfalls behandelt werden.

Lösen der Feynman-Integrale. Wie schon angemerkt, ist der Computer als Werkzeug bei der Berechnung der Integrale hilfreich, meist als Computeralgebrasystem wie z.B. *Mathematica* oder *Maple*. Das automatisierte Lösen von Integralen ist aber bisher nur in Spezialfällen möglich. Trotzdem ist eine Teilautomatisierung der Integralberechnung möglich. Mit Hilfe von Reduktionsverfahren kann man unbekannte Integrale auf bekannte zurückführen. Man braucht dann nur noch die bekannten Integrale zu programmieren und kann trotzdem eine viel größere Anzahl von Integralen berechnen lassen. Die Reduktionsverfahren müssen sich dazu natürlich auch automatisieren lassen. Oft ist dies möglich. Am Ende des Kapitels gehen wir darauf noch etwas näher ein.

Berechnung von $|\mathcal{M}|^2$. Die Ausdrücke für die einzelnen Diagramme müssen addiert und quadriert werden. Dabei müssen unter anderem Spuren über Produkte von γ-Matrizen oder von Farb-Matrizen (Gell-Mann-Matrizen) berechnet werden. Lange Zeit waren Computersysteme mit der Berechnung dieser Spuren bei einer großen Anzahl von Matrizen überfordert. Inzwischen können Computer diese Operationen meist problemlos ausführen. Allerdings können die Ausdrücke insgesamt sehr groß werden und die Leistungsfähigkeit der Computer überfordern, so dass man die Berechnung nicht symbolisch sondern numerisch ausführen muss. Dann hat man aber keine Formel als Ergebnis, deren Variablen man für verschiedene Impulskonfigurationen wählen kann, sondern nur eine komplexe Zahl. Die Phasenraum-Integration muss entsprechend für jeden Integrationspunkt die Berechnung von $|\mathcal{M}|^2$ wiederholen.

Renormierung. Dieser Rechenschritt ist nur wenig automatisiert. Manche Renomierungsbedingungen lassen sich leicht algebraisch definieren und können von einem Computer ausgeführt werden. Im allgemeinen ist aber der manuelle Eingriff des Menschen nötig.

Phasenraum-Integration. Der letzte Schritt ist die Phasenraumintegration. Der Integrand ist fast immer zu kompliziert für eine analytische Rechnung, und oft liegt er auch nur numerisch vor. Die Integration wird daher mit Hilfe von Monte-Carlo-Verfahren numerisch ausgeführt. Der Integrand muss dabei divergenzfrei sein und eine stabile numerische Auswertung erlauben. Beides ist in der Praxis oft nicht gegeben. Zwar sind die physikalischen Divergenzen in der Regel entfernt, aber durch Reduktionsverfahren bei der Integralberechnung erhält man neue, künstliche Divergenzen in Teilbeiträgen. Die Phasenraum-Integration muss entsprechend angepasst werden, um die problematischen Phasenraum-Punkte in den Griff zu bekommen. Eine vollständige Automatisierung ist deshalb nicht ohne weiteres möglich. Hinzu kommt die Abhängigkeit der Integrationsgrenzen von dem experimentellen Aufbau. Die Grenzen sind meist nicht trivial. Deshalb muss das Integral auf einfacherere Gebiete transformiert werden. Dieser Schritt ist ebenfalls nur teilweise automatisiert.

Die beschriebene Teilautomatisierung einzelner Rechenschritte wird schon von einigen Computerprogrammen umgesetzt [6, 9]. Es wird versucht die Automatisierung noch voll-

ständiger zu machen. Die Hintergründe dafür sind, dass zwischen den einzelnen Schritten immer noch der Eingriff von Menschen nötig ist und die Größe und Komplexität der dabei zu betrachtenden Daten ein Problem darstellt. Es muss viel Zeit und Arbeit für die gleichen monotonen Arbeitsvorgänge investiert werden. Dabei ist die Gefahr von menschlichen Fehlern ein zunehmend großes Problem. Durch eine weitergehende Automatisierung wird diese Gefahr reduziert. Darüber hinaus kommen die verwendeten großen Computerprogramme nicht mehr ohne Verifizierung aus. Der Programmcode muss getestet und überprüft werden. Diese Prüfung ist strenggenommen bei jeder Änderung oder Erweiterung des Programms sinnvoll, so dass auch das Überprüfen zu einem automatisierten Prozess werden muss. Solch automatisierte Selbsttests sind in den Programmen teilweise schon vorhanden.

Die Automatisierung der einzelnen Rechenschritte ist ein eigenes wissenschaftliches Themengebiet, auf dem viel geforscht wird. Die Reduktionsverfahren zur Berechnung von Feynman-Integralen nehmen darin einen ganz zentralen Platz ein. Sowohl die Schwierigkeiten, als auch die Fortschritte auf diesem Gebiet sind bemerkenswert. Wir gehen im folgenden auf zwei repräsentative Beispiele ein.

Ein lange bekanntes und sehr nützliches Verfahren zur Reduktion von vielen verschiedenen Integralen auf ein paar wenige Basisintegrale ist das *integration-by-parts*-Verfahren (IBP). Viele Berechnungen auf Mehrschleifenniveau wurden erst mit Hilfe des IBP-Verfahrens möglich. Aus den Integraleigenschaften in dimensionaler Regularisierung kann man Beziehungen zwischen verschiedenen Integralen ableiten. Durch die Verknüpfung der unterschiedlichen Beziehungen kann man am Ende ein schwieriges Integral durch mehrere einfachere Integrale ausdrücken. Man muss nur noch wenige Basisintegrale lösen. Obwohl das Verfahren schon 1981 veröffentlicht wurde [10], dauerte es bis zum Jahr 2000, bis ein implementierbarer Algorithmus zur automatisierten Anwendung von IBP beschrieben wurde [11]. Eine Implementierung folgte vier Jahre später [12].

Eine andere Art von Schwierigkeit kann am Beispiel des Passarino-Veltman-Verfahrens [13] zur Tensorreduktion dargestellt werden. Das Passarino-Veltman-Verfahren drückt tensorielle Integrale durch skalare Integrale aus. Das Verfahren lässt sich ganz direkt und einfach als Computerprogramm umsetzen. Die Probleme werden erst später bei der Anwendung sichtbar. An einem typischen Einschleifen-Integral lassen sich diese illustrieren. Gegeben sei

$$I^{\mu_1\cdots\mu_p} = \int \mathrm{d}l \, \frac{l^{\mu_1}\ldots l^{\mu_p}}{D_1\cdots D_k} \,, \tag{2.8}$$

mit den k Propagatortermen D_i, die von dem Schleifenimpuls l, den Massen m_i und den äußeren Impulsen $p_1, \ldots, p_n$ abhängen. Um im Zähler den aus l^μ gebildeten Tensor auszuwerten, nutzt man aus, dass das Ergebnis des Integrals nur von Tensoren der äußeren Impulse und einfachen Invarianten abhängen kann, nicht aber von den Schleifenimpulsen. Also kann man den Ansatz machen:

$$I^{\mu_1\cdots\mu_p} = \sum_s \mathcal{T}_s^{\mu_1\cdots\mu_p} J_s \,. \tag{2.9}$$

Der tensorielle Anteil $\mathcal{T}$ wird aus dem metrischen Tensor $g^{\mu\nu}$ und den Vektoren $p_1^\mu, \ldots, p_n^\mu$ gebildet. Die Indizes $\mu_1, \cdots, \mu_p$ werden auf diese Tensoren verteilt. Die Summe läuft dabei über alle möglichen symmetrisierten Verteilungen s. Jeder dieser Tensorterme hat ein

skalares Integral J_s als Faktor, dessen genaue Form noch unbekannt ist. Um die J_s zu bestimmen, kann man beide Seiten der Gleichung (2.9) jeweils mit den Tensoren $g^{\mu\nu}$ und $p_1^\mu, \ldots, p_n^\mu$ kontrahieren, und erhält so mehrere Gleichungen. Es ergeben sich auf der linken Seite der Gleichungen Skalarprodukte zwischen den äußeren Impulsen und dem Schleifenimpuls, die sich als Propagatoren umschreiben lassen und reduzierte Integrale I_i ergeben. Letztlich hat man so ein Gleichungssystem mit den Unbekannten J_s, das die Form

$$G \cdot \begin{pmatrix} J_1 \\ \vdots \\ J_k \end{pmatrix} = \begin{pmatrix} I_1 \\ \vdots \\ I_k \end{pmatrix} \tag{2.10}$$

hat. Die Matrix G enthält Skalarprodukte zwischen allen äußeren Impulsen; bei geeigneter Normierung der $\mathcal{T}_s$ hat man $G_{ij} = 2p_i \cdot p_j$. Zur Lösung des Systems muss man G invertieren und erhält dabei die Determinante von G im Nenner der Lösungen. Diese inverse Determinante wird Gram-Determinante genannt und ist die Ursache von erheblichen Problemen bei der Anwendung des Verfahrens. Im Phasenraum gibt es Punkte, an denen die Skalarprodukte und damit die Gram-Determinante im Nenner Null wird. Die resultierenden Pole machen eine direkte Phasenraum-Integration unmöglich. Es sind schon viele Verbesserungen des Verfahrens entwickelt worden, aber erst kürzlich ist eine Methode publiziert worden [14], die eine vollständig automatisierte Behandlung der problematischen Pole möglich macht. Eine Demonstration der Machbarkeit dieses Verfahrens im Rahmen einer größeren bzw. vollständigen Rechnung steht aber noch aus.

Die Automatisierung der störungstheoretischen Berechnungen ist ein sehr aktuelles Thema. Die Hauptschwierigkeiten der Automatisierung liegen, wie in den Beispielen angesprochen, entweder in dem Fehlen eines programmierbaren Algorithmus, oder in der Vermeidung spezifischer, meist numerischer Probleme bei der Anwendung eines bekannten Verfahrens. In den folgenden Kapiteln werden Beiträge zu verschiedenen Aspekten der Automatisierung vorgestellt.

3 Numerische Berechnung von masselosen Einschleifen-N-Punkt-Funktionen

Luke: *Is the numerical side stronger?*
Yoda: *No, no. Quicker, easier, more seductive.*
Luke: *But how am I to know the analytical side from the numerical?*
Yoda: *You will know ...*

Star Wars V

In Kapitel 2 haben wir beschrieben, wie wichtig die NLO-Berechnung von N-Jet-Ereignissen im Hinblick vor allem auf den LHC ist. Es werden dafür Softwareprogramme notwendig, die diese Rechnungen – auf Grund ihres Umfangs und ihrer Komplexität – weitgehend automatisiert ausführen können. Das Verfahren und seine Implementierung als Computerprogramm, welches im folgenden beschrieben wird, hat genau diese automatisierte Berechnung zum Ziel.

Das Projekt zur Berechnung von NLO-Beiträgen besteht aus mehreren Teilen, von denen die Berechnung der Einschleifen-N-Punkt-Funktionen der zentrale ist. In dieser Arbeit beschreiben wir nur dieses, bereits fertiggestellte Kernmodul. Eine detaillierte Beschreibung der weiteren Teile, vor allem der Generierung der Baumgraphenstrukturen, wird hier ausgelassen, auch weil deren Implementierung noch aussteht.

Zunächst soll trotzdem kurz auf die schon bei Baumgraphen-Berechnungen benutzten Techniken eingegangen werden, da diese auch für die NLO-Rechnungen zum Einsatz kommen und direkte Auswirkungen auf die Art und Weise der Schleifenberechnung haben. Danach wird das Verfahren Schritt für Schritt beschrieben, zunächst die Tensorintegrale, dann die skalaren Integrale betreffend. Abschließend wird das eigentliche Computerprogramm, seine Implementierung und Optimierung, behandelt.

3.1 Helizitätsspinoren

Schon die Berechnung von Baumgraphen kann einen so erheblichen Umfang annehmen, dass man an die Grenzen der menschlichen oder maschinellen Leistungsfähigkeit stößt. Die Berechnung von virtuellen Korrekturen zu $2 \to 2$ – Prozessen war in den 80er Jahren solch eine Herausforderung. Damals fiel auf, dass der große Umfang der Rechnung in keinem Verhältnis zum kurzen, einfachen Ergebnis stand. Die Methode, die damals erdacht wurde, um zumindest einen Teil der offensichtlich vorhandenen, überflüssigen Zwischenterme zu vermeiden, war die Verwendung von Helizitätsspinoren[1].

[1] Neben einem Überblick und vielen Anwendungen dieser Methode findet sich in [15] auch ein interessanter Abriss der historischen Entwicklung.

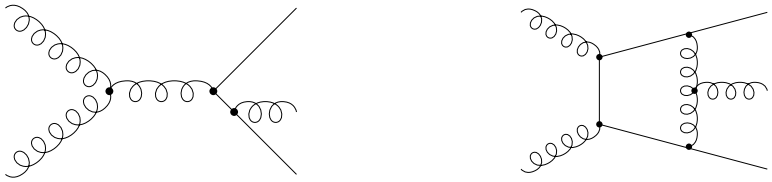


Abbildung 3.1: Beispiele für NLO-Graphen

Man berechnet dabei zunächst Amplituden für die Streuung von polarisierten Teilchen, quadriert diese und summiert bzw. mittelt erst anschließend über die Polarisationen. Dies führt zu einer Reduktion des Aufwands im Vergleich zum Standardverfahren. Um polarisierte Anfangs- und Endzustände beschreiben zu können, setzt man masselose Teilchen voraus[2], für die die Helizitätszustände mit Hilfe der Chiralitätsprojektoren definiert werden können:

$$|p\pm\rangle = \frac{1 \pm \gamma_5}{2}\, u(p)\ , \qquad \langle p\pm| = \overline{u(p)}\, \frac{1 \mp \gamma_5}{2}\ . \tag{3.1}$$

Entscheidend ist, dass man auch die Polarisationsvektoren von Photonen oder Gluonen mit Hilfe von Helizitätsspinoren schreiben kann:

$$\epsilon_\mu^\pm(p,l) = \pm\frac{\langle l \mp|\gamma_\mu|p\mp\rangle}{\sqrt{2}\langle l \mp|p\pm\rangle}\ , \tag{3.2}$$

wobei l ein beliebiger masseloser Impuls ist, den man frei wählen kann. p ist der Impuls des Photons bzw. Gluons. Erleichtert wird die Berechnung durch die Tatsache, dass einige Diagramme wegen nicht-passenden Spin-Kombinationen direkt Null sind, man diese also nicht explizit auszurechnen hat. Die geschickte Wahl von l kann die Rechnungen ebenfalls vereinfachen. Schließlich erhält man mittels Paritätstransformationen Identitäten zwischen verschiedenen Amplituden, so dass nur ein Teil der Amplituden tatsächlich berechnet werden muss. Die algebraischen Rechnungen mit Helizitätsspinoren sind sehr einfach und man hat viele Beziehungen zur Verfügung, mit denen man Ausdrücke leicht umformen kann. Letztlich können die Ergebnisse immer in Form von Skalarprodukten zwischen Helizitätsspinoren geschrieben werden. In Anhang A sind alle relevanten Formeln zusammengefasst.

3.1.1 Numerische Auswertung

Die einfache numerische Handhabung von Helizitätsspinoren ist der zweite entscheidende Vorteil dieser Methode. Die Berechnung eines Skalarprodukts (Details siehe Anhang A) ist einfach

$$\begin{aligned}\langle p-|q+\rangle &= \frac{1}{\sqrt{|p_+||q_+|}}(q_+ p_{\perp^*} - p_+ q_{\perp^*})\\ \langle p+|q-\rangle &= -\frac{1}{\sqrt{|p_+||q_+|}} e^{-i\phi_p} e^{-i\phi_q}(q_+ p_\perp - p_+ q_\perp)\ ,\end{aligned} \tag{3.3}$$

[2] In vielen Anwendungen, z.B. für Streuprozesse bei hohen Energien, ist die Annahme von masselosen Teilchen durchaus gerechtfertigt. Die Erweiterung des Verfahrens auf massive Teilchen ist in [16] beschrieben.

und besteht nur aus wenigen mathematischen Operationen, so dass eine Implementierung auf Computern sehr effizient ist. Statt große analytische Ausdrücke quadrieren und gegebenenfalls Spuren darüber bilden zu müssen, kann man einfach die Skalarprodukte und damit die Teilergebnisse numerisch ausrechnen, und dann die komplexen Zahlen quadrieren und aufsummieren. Die Berechnung ist numerisch stabiler, weil man viel weniger Terme hat, die sich mit verschiedenen Vorzeichen kompensieren könnten. Probleme mit Nennern, die Null werden können, gibt es auch nicht. Noch gravierender wirkt sich der Vorteil dieser schnellen, numerischen Herangehensweise aus, wenn man bei der Berechnung von Matrixelementen auf rekursive Verfahren angewiesen ist. Die Berechnung von Baumgraphen mit vielen äußeren Teilchen ist ein solcher Fall. Mit Standardverfahren müsste man wiederholt bei jedem Rekursionsschritt lange analytische Ergebnisse in die Reduktionsformeln einsetzen und würde so noch viel längere algebraische Ausdrücke erzeugen, die schnell jedes existierende Computeralgebrasystem an oder über die Grenzen seiner Leistungsfähigkeit bringen. Mit einem numerischen Ansatz verschwindet dieses Problem. Die Skalarprodukte und damit das gesamte Ergebnis können nach (3.3) in einfache komplexe Zahlen verwandelt werden, so dass bei jedem Rekursionsschritt immer der gleiche, geringe Aufwand entsteht, nämlich das Einsetzen weniger Zahlen in eine Reduktionsformel. In der praktischen Anwendung ist die Helizitätsspinoren-Methode daher immer ein numerisches Verfahren. Das ist der Grund, warum wir für unser Projekt zur Berechnung von NLO-Beiträgen numerische und nicht analytische Methoden verwenden.

3.1.2 Algebraische Umformungen

Als ein Beispiel für die Verwendung von Helizitätsspinoren und deren algebraischer Manipulation soll der tensorielle Zähler eines konkreten 6-Punkt-Integrals umgeformt werden.

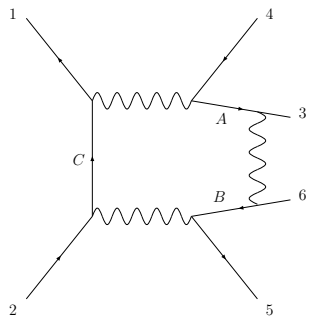


Abbildung 3.2: Beispiel für ein 6-Punkt-Diagramm.

Sei ein Diagramm wie in Abbildung 3.2 gegeben[3], das die paarweise Erzeugung von W-Bosonen und deren Vernichtung in Fermion-Antifermion-Paare beschreibt, wobei zwischen zwei Endteilchen noch ein Vektorboson ausgetauscht wird. Die Propagatoren, die nichttriviale Zählerfaktoren liefern, sind mit A, B und C gekennzeichnet. Die äußeren Teilchen sind durchnumeriert und haben die entsprechenden Impulse $p_1, \ldots, p_6$. In konventioneller Schreibweise lautet der Zähler:

$$\mathcal{Z} = \bar{v}(p_1)\gamma_\mu \not{p}_C \gamma_\nu u(p_2)\, \bar{u}(p_5)\gamma^\nu \not{p}_B \gamma_\beta v(p_6)\, \bar{u}(p_3)\gamma^\beta \not{p}_A \gamma^\mu v(p_4)\,, \tag{3.4}$$

[3] Das Beispiel ist [17] entnommen.

wobei

$$p_A = k \text{ (Schleifenimpuls)}, \quad p_B = k + p_3 + p_6, \quad p_C = k - p_1 - p_4 \,. \tag{3.5}$$

Wählt man eine der 64 möglichen Helizitätskonfigurationen aus, dann kann man den Zähler mit Hilfe von Helizitätsspinoren schreiben, zum Beispiel:

$$\mathcal{Z} = \langle 1+|\gamma_\mu \not{p}_C \gamma_\nu|2-\rangle\langle 5+|\gamma^\nu \not{p}_B \gamma_\beta|6-\rangle\langle 3+|\gamma^\beta \not{p}_A \gamma^\mu|4-\rangle \,. \tag{3.6}$$

Der Zähler lässt sich nun so umformen, dass er nur noch Spinor-Sandwiches mit je maximal einem Schleifenimpuls enthält:

$$\begin{aligned}
\mathcal{Z} &= \frac{1}{\langle p_2+|\not{p}|p_5-\rangle\langle p_6+|\not{p}|p_3-\rangle}\langle 1+|\gamma_\mu \not{p}_C \gamma_\nu \not{p}_2 \not{p} \not{p}_5 \gamma^\nu \not{p}_B \gamma_\beta \not{p}_6 \not{p} \not{p}_3 \gamma^\beta \not{p}_A \gamma^\mu|4-\rangle \\
&= -8\frac{1}{\langle p_2+|\not{p}|p_5-\rangle\langle p_6+|\not{p}|p_3-\rangle}\langle 1+|\not{p}_A \not{p}_6 \not{p} \not{p}_3 \not{p}_B \not{p}_2 \not{p} \not{p}_5 \not{p}_C|4-\rangle \\
&= -8\langle 1+|\not{p}_A|6-\rangle\langle 3+|\not{p}_B|2-\rangle\langle 5+|\not{p}_C|4-\rangle \\
&= 8\Big[\langle 1+|\not{k}|6-\rangle\langle 3+|\not{p}_6|2-\rangle\langle 5+|\not{p}_1|4-\rangle \\
&\quad - \langle 1+|\not{k}|6-\rangle\langle 3+|\not{p}_6|2-\rangle\langle 5+|\not{k}|4-\rangle \\
&\quad + \langle 1+|\not{k}|6-\rangle\langle 3+|\not{k}|2-\rangle\langle 5+|\not{p}_1|4-\rangle \\
&\quad - \langle 1+|\not{k}|6-\rangle\langle 3+|\not{k}|2-\rangle\langle 5+|\not{k}|4-\rangle\Big] \,.
\end{aligned} \tag{3.7}$$

Dabei wurden für die erste und dritte Zeile die Vollständigkeitsrelationen (A.7) benutzt. In der zweiten Zeile wurden mit Hilfe der bekannten Chisholm-Identitäten die Indizes β, μ und ν kontrahiert. An Ende wurde nur noch ausmultipliziert.

Damit sind die für die Schleifenberechnung wichtigen Details des Helizitätsverfahrens bekannt. Die Entwicklung neuer Techniken ist jedoch nicht stehengeblieben und soll noch kurz skizziert werden. Die Ausdrücke für Gluon-Vertizes machen durch ihre große Anzahl von Indizes Schwierigkeiten. Als Lösung wurde die Zerlegung der Amplituden in einen Farbanteil und einen „farblosen“, sogenannten Partialamplituden-Anteil erdacht (engl.: *color decomposition*). Für die Partialamplituden, die im wesentlichen den ursprünglichen Amplituden entsprechen, wurden weitere Verfahren entwickelt, bei denen die Baumgraphen nach und nach rekursiv durch Anhängen immer weiterer Linien konstruiert werden. Diese Techniken können dann mit den Schleifenrechnungen kombiniert werden. In [18] und [19] finden sich ausführliche Darstellungen dieser Verfahren. Ganz neue Techniken aus dem Umfeld der String-Theorie werden in [20] beschrieben.

Als Vorgriff auf den Abschnitt 3.2.4 soll aber erwähnt sein, dass das Helizitätsverfahren von 4 Dimensionen ausgeht. Die Berechnung der Einschleifen-Integrale im folgenden wird dimensionale Regularisierung benutzen, also $D = 4-2\epsilon$ Dimensionen annehmen. Der vielleicht auftretende Verdacht, dass hier eine Inkonsistenz vorhanden ist, wird im genannten Abschnitt über gemischte Regularisierungsschemata ausgeräumt werden.

3.2 Tensorreduktion

Zunächst sollen die Konventionen und Notationen für Einschleifen-Diagramme festgelegt werden. Wie in Abbildung 3.3 gezeigt, laufen alle Impulse p_i aus der Schleife aus. Die

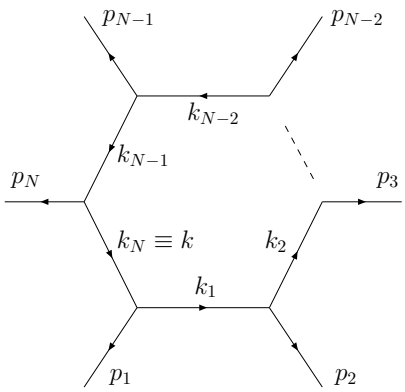


Abbildung 3.3: Konventionen für das Einschleifen-N-Punkt-Diagramm

Propagatoren

$$k_i = k - q_i \quad \text{mit} \quad q_i = \sum_{j=1}^{i} p_j \quad \text{und} \quad q_0 = 0 \tag{3.8}$$

mit dem Schleifenimpuls k sind entsprechend definiert. Um die Schreibweise elegant zu halten, soll $k_N = k_0 \equiv k$ definiert sein.

Aus der Problemstellung ist klar, dass nicht alle äußeren Impulse p_i *on-shell* sein müssen, da prinzipiell an jedes externe Bein des Graphen ein nicht-trivialer Baumgraph angehängt sein könnte. Bei allen Rechnungen wird die dimensionale Regularisierung verwendet, die eine bequeme, einheitliche Behandlung sowohl der UV- als auch der IR-Divergenzen ermöglicht. Wir setzen außerdem voraus, dass alle in der Schleife umlaufenden Teilchen masselos sind. Damit lässt sich in allgemeinster Form das zu lösende Schleifenintegral wie folgt schreiben:

$$\begin{aligned} I_N^{\mu_1 \cdots \mu_r} &= e^{\epsilon \gamma_E} \mu^{2\epsilon} \int \frac{\mathrm{d}^D k}{i\pi^{\frac{D}{2}}} \frac{k^{\mu_1} \cdots k^{\mu_r}}{k^2 (k-p_1)^2 \cdots (k-p_1-\cdots-p_{N-1})^2} \\ &= e^{\epsilon \gamma_E} \mu^{2\epsilon} \int \frac{\mathrm{d}^D k}{i\pi^{\frac{D}{2}}} \frac{k^{\mu_1} \cdots k^{\mu_r}}{k_1^2 \cdots k_N^2} \,. \end{aligned} \tag{3.9}$$

Die Dimension D ist gewählt zu $D = 4 - 2\epsilon$. Die Vorfaktoren des Integrals dienen nur der vereinfachten Schreibweise des Ergebnisses im Rahmen des $\overline{MS}$-Renormierungsschemas. r ist der Rang des Tensorintegrals. Ziel der Tensoreduktion ist nun die Rückführung des Integrals auf skalare Integrale mit $r = 0$.

3.2.1 Reduktionsidee

In den üblichen Verfahren macht man nun den Ansatz, dass man alle möglichen Kombinationen von erlaubten tensoriellen Größen mit unbekannten skalaren Vorfaktoren aufstellt. Zum Beispiel für ein 5-Punkt-Integral mit Rang 2 würde man ansetzen

$$\begin{aligned} I_5^{\mu_1 \mu_2} &= e^{\epsilon \gamma_E} \mu^{2\epsilon} \int \frac{\mathrm{d}^D k}{i\pi^{\frac{D}{2}}} \frac{k^{\mu_1} k^{\mu_2}}{k_1^2 k_2^2 k_3^2 k_4^2 k_5^2} \\ &= E_1 p_1^{\mu_1} p_1^{\mu_2} + E_2 p_2^{\mu_1} p_2^{\mu_2} + E_3 p_3^{\mu_1} p_3^{\mu_2} + \cdots + E_{11} g^{\mu_1 \mu_2} \,. \end{aligned} \tag{3.10}$$

Man findet insgesamt 11 linear unabhängige Terme mit noch unbekannten skalaren Vorfaktoren $E_1, \ldots, E_{11}$. Durch Multiplikation dieser Gleichung mit den externen kinematischen Größen bekommt man ein Gleichungssystem, durch dessen Lösung die skalaren Integrale konkret bestimmt und ausgerechnet werden können. Leider bekommt man so auch die unerwünschten Gram-Determinanten in den Nenner, die erhebliche Probleme bei der numerischen Auswertung verursachen.

Bevor nun das neue Verfahren vorgestellt wird, das diese Probleme nicht oder nur in abgeschwächter Form aufwirft, soll nochmal kurz das Charakteristische dieses Weges beschrieben werden: man betrachtet zunächst das tensorielle Integral, ohne die Eigenschaften der γ-Matrizen bei der Rechnung zu berücksichtigen. Erst nachdem man konkrete Lösungen für die skalaren Integrale gefunden und sie als Vorfaktoren in einem tensoriellen Ansatz eingesetzt hat, behandelt man die γ-Matrizen algebraisch in dem Ausdruck für das invariante Matrixelement $\mathcal{M}$.

Das hier verwendete Verfahren macht dies nun genau umgekehrt. Es nutzt die Algebra der γ-Matrizen direkt bei der Tensorreduktion aus. Wie das funktioniert und wie man auf die Idee zu diesem neuen Ansatz kommen kann, lässt sich an folgendem Beispiel demonstrieren.

Berechnet man

$$\gamma_{\mu_1}\gamma_{\mu_2}\, I_5^{\mu_1\mu_2} = e^{\epsilon\gamma_E}\mu^{2\epsilon}\int \frac{\mathrm{d}^D k}{i\pi^{\frac{D}{2}}} \frac{\not{k}\,\not{k}}{k_1^2 k_2^2 k_3^2 k_4^2 k_5^2}\,, \tag{3.11}$$

so kann man dazu natürlich das Verfahren von Passarino-Veltman anwenden. Weiß man allerdings, dass

$$\not{k}\,\not{k} = k^2 \equiv k_5^2 \tag{3.12}$$

ist und man somit eine Reduktion

$$\gamma_{\mu_1}\gamma_{\mu_2}\, I_5^{\mu_1\mu_2} = e^{\epsilon\gamma_E}\mu^{2\epsilon}\int \frac{\mathrm{d}^D k}{i\pi^{\frac{D}{2}}} \frac{1}{k_1^2 k_2^2 k_3^2 k_4^2} \tag{3.13}$$

sofort ausführen kann, sieht man, dass man so einfacher das gleiche Ergebnis bekommt.

Dabei ist die Dirac-Algebra das entscheidende Werkzeug zur Vereinfachung. Es ist nun die Frage, wie man diese Idee in systematischer Weise umsetzt. Es muss sichergestellt werden, dass nach geeigneten Umformungen im Zähler immer Terme entstehen, die entweder einem Propagatorterm entsprechen, oder von k unabhängig sind. Dabei sollten Gram-Determinanten im Nenner möglichst vermieden werden. Das konkrete Verfahren wurde in [17], [21] und [22] entwickelt und soll nun vorgestellt werden.

3.2.2 Basis-Formeln

Im Allgemeinen befinden sich neben den $\not{k}$ auch andere, mit γ-Matrizen kontrahierte Größen im Zähler. Die $\not{k}$ stehen also in der Regel nicht nebeneinander. Um die obige Reduktionsidee umsetzen zu können, muss man die $\not{k}$ nebeneinander bringen. Dies gelingt durch die Dirac-Algebra

$$\{\gamma^\mu, \gamma^\nu\} = 2g^{\mu\nu} \tag{3.14}$$

mit der man als Beispiel

$$\not{a}\not{b} = 2(a\cdot b) - \not{b}\not{a}$$

die γ-Matrizen vertauschen kann. Dabei treten auch Skalarprodukte auf. Nicht mehr offensichtlich ist nun, ob diese Skalarprodukte die passende Form haben, Propagatoren zu kürzen. Tun sie das nicht, dann erhält man wieder ein Gleichungssystem, das man zu lösen hat, bei dem die Gram-Determinanten im Nenner auftreten.

Das Problem kann man systematisch angehen, indem man mittels Dirac-Algebra eine Beziehung für $\not{k}$ findet, die $\not{k}$ in allgemeinster Form ausdrückt. Dieses ist zugleich eine der zentralen Formeln für die rekursive Tensorreduktion. Die Formel lautet

$$\not{k} = \frac{1}{(2l_1 \cdot l_2)} \left[(2k \cdot l_2) \not{l}_1 + (2k \cdot l_1) \not{l}_2 - \not{l}_1 \not{k} \not{l}_2 - \not{l}_2 \not{k} \not{l}_1 \right] \tag{3.15}$$

und ist ausschließlich mittels der Antikommutatorbeziehung der Dirac-Algebra hergeleitet. Die l_1 und l_2 sind zunächst beliebige Vierervektoren. Mit dieser Gleichung lassen sich nun die $\not{k}$ in den Spinor-Sandwiches wie die im Beispiel (3.7) ersetzen.

Die beiden ersten Terme reduzieren den Rang bereits explizit, wenn l_1 und l_2 gleich den p_i aus den Schleifenpropagatoren oder einer Linearkombination aus ihnen gewählt wurden. Dazu ist zu wissen, dass

$$2k_j \cdot p_i = k_{i-1}^2 - k_i^2 + q_i^2 - q_{i-1}^2 - 2p_i \cdot q_j \tag{3.16}$$

gilt und hier offensichtlich die beiden ersten Terme jeweils ein Integral liefern, bei dem der entsprechende Propagator gekürzt ist, und die letzten Terme ein um einen Rang reduziertes N-Punkt-Integral liefern. Mit unserer Konvention für den Zähler des Schleifenintegrals ist $j = 0$, und damit fällt der letzte Term in (3.16) weg.

Schwieriger sind die beiden letzten Terme in Gleichung (3.15). Hier wird nun gleichzeitig zur Dirac-Algebra die Spinor-Technik wichtig. In unserer Problemstellung kann man den Zähler des Integrals ohne Einschränkung der Allgemeinheit als

$$\langle a_1 - | \not{k} | b_1 - \rangle \cdots \langle a_r - | \not{k} | b_r - \rangle \tag{3.17}$$

annehmen (siehe dazu Beispiel (3.7)). Sollte ein anderes k als k_N in den Produkten stehen, kann man zuvor immer den Term mit $k_i = k - q_i$ passend aufspalten. Wenn nun l_1 und l_2 masselos sind, dann lassen sich die zwei letzten Terme in (3.15) wieder passend umwandeln. Dass man l_1 und l_2 immer als masselos annehmen darf, wird im nächsten Abschnitt geklärt. Die Einfügung von Projektionsoperatoren und Ausnutzung der Vollständigkeitsrelationen ergibt

$$\langle a - | \not{l}_1 \not{k} \not{l}_2 | b - \rangle = \langle a l_1 \rangle [l_2 b] \langle l_2 - | \not{k} | l_1 - \rangle \,. \tag{3.18}$$

Der Effekt der beiden letzten Terme in (3.15) ist also, ein $\langle a - | \not{k} | b - \rangle$ im Zähler durch ein $\langle l_{1,2} - | \not{k} | l_{2,1} - \rangle$ zu ersetzen.

Betrachtet man vorläufig nur Integrale mit Rang $r \geq 2$ und wendet (3.15) auf zwei $\not{k}$ im Zähler an, dann bleiben neben Termen mit der direkten Reduktion durch (3.16) noch solche übrig, welche die Formen

$$\langle l_1 - | \not{k} | l_2 - \rangle \langle l_1 - | \not{k} | l_2 - \rangle \tag{3.19}$$

oder

$$\langle l_1 - | \not{k} | l_2 - \rangle \langle l_2 - | \not{k} | l_1 - \rangle \tag{3.20}$$

haben. Zur weiteren Behandlung kann man folgende Formeln verwenden:

$$\langle l_1 - |\not{k}| l_2 - \rangle \langle l_2 - |\not{k}| l_1 - \rangle = (2k \cdot l_1)(2k \cdot l_2) - (2l_1 \cdot l_2)k^2 \tag{3.21}$$

und

$$\begin{aligned}\langle l_1 - |\not{k}| l_2 - \rangle \langle l_1 - |\not{k}| l_2 - \rangle = &- \frac{\langle l_1 - |\not{p}_3| l_2 - \rangle}{\langle l_2 - |\not{p}_3| l_1 - \rangle} \langle l_1 - |\not{k}| l_2 - \rangle \langle l_2 - |\not{k}| l_1 - \rangle \\ &+ \frac{\langle l_1 - |\not{k}| l_2 - \rangle}{\langle l_2 - |\not{p}_3| l_1 - \rangle} \left[(2l_1 p_3)\,(2l_2 k_l) + (2l_2 p_3)\,(2l_1 k_l) - (2l_1 l_2)\,(2p_3 k_l) \right], \end{aligned} \tag{3.22}$$

mit einem dritten unabhängigen, externen Impuls p_3. Mit (3.16) erhält man dann wieder Integrale mit gekürzten Propagatoren oder skalare Integrale. Für 2- und 3-Punkt-Integrale existiert ein solcher Impuls natürlich nicht und die obige Formel ist nicht anwendbar. Allerdings verschwinden alle 3-Punkt-Integrale, wenn sie Terme der Form

$$\langle l_1 - |\not{k}| l_2 - \rangle \cdots \langle l_1 - |\not{k}| l_2 - \rangle \tag{3.23}$$

im Zähler haben. Sie verschwinden aus Symmetriegründen, da die resultierenden

$$\langle l_1 - |\gamma_{\mu_1}| l_2 - \rangle \cdots \langle l_1 - |\gamma_{\mu_r}| l_2 - \rangle \tag{3.24}$$

mit p_1^μ, p_2^μ und $g^{\mu\nu}$ kontrahiert werden.

Die vollständige Tensorreduktion wird durch die Anwendung von (3.15) auf jedes $\not{k}$ im Zähler des Tensorintegrals erreicht. Mit (3.16) werden die entstehenden Terme auf skalare Integrale oder Integrale mit gekürzten Propagatoren zurückgeführt. Gegebenfalls muss noch (3.21) und (3.22) benutzt werden. Die wichtigen Reduktionsformeln sind also (3.15) und (3.16), sowie (3.21) und (3.22). Damit ist das Reduktionsverfahren weitgehend beschrieben. Das Verfahren funktioniert ganz allgemein für Integrale mit Rang $r \geq 2$. Rang-1-Integrale müssen jedoch noch berechnet werden. Außerdem fehlt noch die Behandlung von 2-Punkt-Integralen. Beides wird in den Abschnitten 3.2.5 und 3.2.6 nachgeholt. Zuvor ist aber noch zu zeigen, warum die l_1 und l_2 immer als masselos angenommen werden können und was die Auswirkungen eines gemischten Regularisierungsschemas auf die oben berechneten Reduktionsformeln sind.

Bis jetzt wurde die Fixierung der Eichung noch nicht erwähnt. Wir haben die Freiheit eine Eichung unserer Wahl zu benutzen und legen uns auf die Feynman-Eichung fest. Das hat zur Folge, dass der Rang r der Integrale immer kleiner oder gleich der Anzahl N äußerer Teilchen ist. Die Reduktionsformeln in diesem Abschnitt, aber auch die in den folgenden, erhalten die Beziehung $r \leq N$. Dies wird später für die Betrachtung der 2-Punkt-Integrale und der notwendigen Basisintegrale von Bedeutung sein.

3.2.3 Masselose Impulse

Besitzt das Integral zwei masselose äußere Impulse p_x und p_y, so kann man einfach

$$l_1 = p_x, \quad l_2 = p_y \tag{3.25}$$

wählen. Ist dies nicht der Fall, kann man zwei masselose Vektoren konstruieren. Der Preis dafür ist die teilweise Wiedereinführung der Gram-Determinate im Nenner.

Nehmen wir an, dass p_x lichtartig und p_y massiv ist, dann kann man konstruieren

$$l_1 = p_x, \quad l_2 = -\alpha_2 p_x + p_y \,, \tag{3.26}$$

wobei

$$\alpha_2 = \frac{p_y^2}{2p_x \cdot p_y} \,. \tag{3.27}$$

Die inverse Formel lautet

$$p_x = l_1, \quad p_y = \alpha_2 l_1 + l_2 \,. \tag{3.28}$$

Sind alle Impulse massiv, dann kann man

$$\begin{aligned} l_1 &= \frac{1}{1-\alpha_1\alpha_2}(p_x - \alpha_1 p_y) \\ l_2 &= \frac{1}{1-\alpha_1\alpha_2}(-\alpha_2 p_x + p_y) \end{aligned} \tag{3.29}$$

setzen. Aus (3.29) und der Bedindung $l_1^2 = l_2^2 = 0$ ergeben sich die α_i als Lösung der quadratischen Gleichungen

$$\begin{aligned} \alpha_1^2 p_y^2 - 2\alpha_1 p_x p_y + p_x^2 &= 0 \\ \alpha_2^2 p_x^2 - 2\alpha_2 p_x p_y + p_y^2 &= 0 \,. \end{aligned} \tag{3.30}$$

Für $2p_x \cdot p_y > 0$ gilt

$$\alpha_1 = \frac{2p_x p_y - \sqrt{\Delta}}{2p_y^2}, \quad \alpha_2 = \frac{2p_x p_y - \sqrt{\Delta}}{2p_x^2} \,. \tag{3.31}$$

Für $2p_x \cdot p_y < 0$ gilt

$$\alpha_1 = \frac{2p_x p_y + \sqrt{\Delta}}{2p_y^2}, \quad \alpha_2 = \frac{2p_x p_y + \sqrt{\Delta}}{2p_x^2} \,. \tag{3.32}$$

Dabei ist jeweils

$$\Delta = (2p_x p_y)^2 - 4p_x^2 p_y^2 \,. \tag{3.33}$$

Δ ist die zweidimensionale Gram-Determinante.

3.2.4 Regularisierungsschemata

Für die Berechnung der Schleifenintegrale haben wir bis jetzt dimensionale Regularisierung verwendet. Wie in Abschnitt 3.1 angesprochen, setzen die Helizitätsmethoden zur Erzeugung der Baumgraphen jedoch vierdimensionale und nicht D-dimensionale Größen voraus. Um die beiden Komponenten, Schleifenrechnung und Baumgraphen, trotzdem miteinander kombinieren zu können, muss das gewöhnliche dimensionale Regularisierungsschema

(CDR), bei dem alle Objekte in $D = 4-2\epsilon$ Dimensionen definiert sind, durch ein abgewandeltes Schema ersetzt werden. In der Literatur [23] findet man zwei passende Alternativen: das t'Hooft-Veltman-Schema (HV) und das vierdimensionale Helizitätsschema (FD). Im HV-Schema werden die Impulse und Helizitätsspinoren der beobachtbaren[4] Teilchen in 4 Dimensionen behandelt, alle anderen Objekte wie in CDR in D Dimensionen. Im FD-Schema werden zusätzlich auch die Helizitätsspinoren der nicht-beobachtbaren Teilchen in 4 Dimensionen behandelt.

Da wir die Spinoren-Sandwiches (3.17) effizient numerisch auswerten möchten, ist für uns das FD-Schema günstiger. An der Beschreibung der Helizitätsspinoren ändert sich gegenüber dem schon beschriebenen dadurch nichts. Die Zähler der Schleifenintegrale ändern sich aber, da nun die Impulse dort in 4 Dimensionen definiert werden. Entsprechend zerlegen wir den D-dimensionalen Schleifenimpuls $k_{(D)}$ in einen 4-dimensionalen Anteil $k_{(4)}$ und einen $(D-4)$-dimensionalen Anteil $k_{(-2\epsilon)}$, so dass

$$k_{(D)}^2 = k_{(4)}^2 + k_{(-2\epsilon)}^2 \tag{3.34}$$

gilt. Das Schleifenintegral schreibt sich dann als

$$I_N^{\mu_1\cdots\mu_r} = e^{\epsilon\gamma_E}\mu^{2\epsilon}\int \frac{\mathrm{d}^D k}{i\pi^{\frac{D}{2}}}\,\frac{k_{(4)}^{\mu_1}\cdots k_{(4)}^{\mu_r}}{k_{1,(D)}^2\cdots k_{N,(D)}^2}\,. \tag{3.35}$$

Der geklammerte Index an den k soll die Dimension klarstellen, in denen der Vektor definiert ist. In den meisten nachfolgenden Formeln werden wir diese Indizes aber zur kompakteren Notation wieder weglassen, wenn keine Verwechslung möglich ist.

Da bei der Berechnung physikalischer Amplituden die Tensorindizes im Schleifenintegral (3.35) immer mit beobachtbaren, also 4-dimensionalen Impulsen kontrahiert werden, werden zunächst nur die 4-dimensionalen Anteile $k_{(4)}$ im Zähler benötigt. Allerdings hat die Tatsache, dass nun nicht mehr ein D-dimensionales k im Zähler steht, Auswirkungen auf die Reduktionsformeln in Abschnitt 3.2.2. Die Hauptreduktionsformel (3.15) wurde ausschließlich mittels der Antikommutator-Relation hergeleitet und gilt daher in allen Dimensionen. Man muss sich hier also nur die k durch $k_{(4)}$ ersetzt denken. In der Formel für das Skalarprodukt (3.16) ändert sich auch nichts, da sich, wenn man auf der rechten Seite die $k_{(4)}$ durch $k_{(D)}$ ersetzt, die $k_{(-2\epsilon)}$-Komponenten jeweils wegheben. Änderungen bekommt man für die Beziehung (3.21) und damit indirekt für (3.22). Das $k_{(4)}^2$ des letzten Terms in (3.21) kürzt nun keinen Propagator mehr. Es muss nach (3.34) durch $k_{(D)}^2$ ersetzt werden, wodurch man einen weiteren Term mit $-k_{(-2\epsilon)}^2$ erhält. Dieser ergibt ein Rang-reduziertes Integral, bei dem $-k_{(-2\epsilon)}^2$ im Zähler steht.

Es ist deshalb notwendig, die Problemstellung zu erweitern, und Integrale der Form

$$I_N^{\mu_1\cdots\mu_r,s} = e^{\epsilon\gamma_E}\mu^{2\epsilon}\int \frac{\mathrm{d}^D k}{i\pi^{\frac{D}{2}}}\,\frac{k_{(4)}^{\mu_1}\cdots k_{(4)}^{\mu_r}\,(-k_{(-2\epsilon)}^2)^s}{k_{1,(D)}^2\cdots k_{N,(D)}^2}\,, \tag{3.36}$$

[4] Beobachtbare Teilchen sind alle externen Teilchen eines Prozesses, die nicht soft oder kollinear sind. Nicht-beobachtbare Teilchen sind die internen bzw. virtuellen Teilchen und die externen soften oder kollinearen Teilchen.

zu betrachten. Zunächst, am Anfang der Tensorreduktion, ist $s = 0$, d.h. der Faktor $(-k^2_{(-2\epsilon)})^s$ existiert nicht. Durch die Anwendung von Gleichung (3.21) können dann Integrale entstehen, bei denen s jeweils um Eins erhöht wird. Der Faktor $(-k^2_{(-2\epsilon)})^s$ wird bei den sukzessiven Tensorreduktionsschritten lediglich mitgeführt und hat dort keinen weiteren Einfluss. Er wird erst dann wichtig, wenn das Integral komplett auf ein skalares Integral reduziert ist. In Abschnitt 3.2.6 und 3.2.8 wird darauf eingegangen.

3.2.5 Rang-1-Integrale

Das Verfahren setzt bis jetzt immer mindestens zwei tensorielle Schleifenimpulse im Zähler voraus. Es bleibt also noch die Behandlung der Integrale vom Rang eins.

Wir nehmen an, dass der Zähler die Form

$$\langle l_1 - | \not{k} | l_2 - \rangle \tag{3.37}$$

hat. Falls die Spinoren nicht zu masselosen Impulsen gehören, muss zuvor noch durch Einfügung von Einsen umgeformt werden:

$$\langle a - | \not{k} | b - \rangle = \langle a - | l_2 + \rangle \langle l_2 + | \not{k} | l_1 + \rangle \langle l_1 + | b - \rangle = \langle a l_2 \rangle [l_1 b] \langle l_1 - | \not{k} | l_2 - \rangle \,, \tag{3.38}$$

wobei dann immer l_1 und l_2 masselos gewählt werden können.

Für die 4-Punkt-Funktion nun ist der Trick die Erweiterung mit einem zusätzlichen Spinor-Sandwich, das von einem unabhängigen dritten, äußeren Impuls abhängt:

$$\langle l_1 - | \not{k} | l_2 - \rangle = \langle l_1 - | \not{k} | l_2 - \rangle \frac{\langle l_2 - | \not{p}_3 | l_1 - \rangle}{\langle l_2 - | \not{p}_3 | l_1 - \rangle} = \frac{1}{\langle l_2 - | \not{p}_3 | l_1 - \rangle} \mathrm{Spur}\big(\Pi_- \not{l}_2 \not{p}_3 \Pi_- \not{l}_1 \not{k}\big) \,, \tag{3.39}$$

und der zweifachen Verwendung der Vollständigkeitsrelation (A.7). Der Projektionsoperator besteht aus den zwei Termen

$$\Pi_- = \frac{1 - \gamma_5}{2} = \frac{1}{2} - \frac{\gamma_5}{2} \,.$$

Die Spuren in (3.39), die γ_5 enthalten, sind proportional zum total antisymmetrischen Tensor und verschwinden deshalb unter der Integration. Die anderen Terme ergeben

$$\begin{aligned} \langle l_1 - | \not{k}^{(4)}_l | l_2 - \rangle = {} & \frac{1}{2 \langle l_2 - | \not{p}_3 | l_1 - \rangle} \left[(2 l_1 p_3)(2 l_2 k_l) + (2 l_2 p_3)(2 l_1 k_l) - (2 l_1 l_2)(2 p_3 k_l) \right] \\ & + \text{ Terme, die nach Integration verschwinden} \,. \end{aligned} \tag{3.40}$$

Im Fall von 3-Punkt-Funktionen hat man keinen unabhängigen dritten Impuls p_3 zur Verfügung. Allerdings gilt für diesen Fall die Aussage in Abschnitt 3.2.2, dass dieses Rang-1-Integral aus Symmetriegründen verschwindet.

N-Punkt-Funktionen mit $N \geq 5$ besitzen einen weiteren unabhängigen Impuls, den wir ohne Beschränkung der Allgemeinheit als p_4 annehmen können. Man kann daher mit einem

weiteren Spinor-Sandwich erweitern und erhält als Reduktionsformel

$$\begin{aligned}\langle l_{1,2}-|\not{k}_l^{(4)}|l_{2,1}-\rangle = -\frac{1}{\delta_{1,2}}\Big[&(2l_1p_4)\,(2l_2k_l)+(2l_2p_4)\,(2l_1k_l)\\ &-(2l_1l_2)\,(2p_4k_l)\Big]\langle l_{1,2}-|\not{p}_3|l_{2,1}-\rangle\\ +\frac{1}{\delta_{1,2}}\Big[&(2l_1p_3)\,(2l_2k_l)+(2l_2p_3)\,(2l_1k_l)\\ &-(2l_1l_2)\,(2p_3k_l)\Big]\langle l_{1,2}-|\not{p}_4|l_{2,1}-\rangle\,,\end{aligned} \tag{3.41}$$

mit

$$\begin{aligned}\delta_1 &= +\langle l_1-|\not{p}_4|\,l_2-\rangle\,\langle l_2-|\not{p}_3|\,l_1-\rangle - \langle l_1-|\not{p}_3|\,l_2-\rangle\,\langle l_2-|\not{p}_4|\,l_1-\rangle\\ \delta_2 &= -\langle l_1-|\not{p}_4|\,l_2-\rangle\,\langle l_2-|\not{p}_3|\,l_1-\rangle + \langle l_1-|\not{p}_3|\,l_2-\rangle\,\langle l_2-|\not{p}_4|\,l_1-\rangle\ .\end{aligned} \tag{3.42}$$

3.2.6 Zweipunkt-Funktion

Als letztes fehlendes Glied in der Tensorreduktion bleibt die Behandlung der Zweipunkt-Funktion. Diese kann mit Standardtechniken erledigt werden. Allgemein hat man nach Feyman-Parametrisierung

$$\begin{aligned}I_2^{\mu_1\dots\mu_r,s} &= e^{\epsilon\gamma_E}\mu^{2\epsilon}\int\frac{\mathrm{d}^Dk}{i\pi^{\frac{D}{2}}}\,\frac{k^{\mu_1}\cdots k^{\mu_r}(-k^2_{(-2\epsilon)})^s}{k^2(k-p)^2}\\ &= e^{\epsilon\gamma_E}\mu^{2\epsilon}\int_0^1 da\int\frac{\mathrm{d}^Dk}{i\pi^{\frac{D}{2}}}\,\frac{(k+ap)^{\mu_1}\cdots(k+ap)^{\mu_r}(-k^2_{(-2\epsilon)})^s}{[-k^2+a(1-a)(-p^2)]^2}\,,\end{aligned} \tag{3.43}$$

wobei p der äußere Impuls ist.

Es gilt $r \le N = 2$. Multipliziert man den Zähler aus, erhält man so Terme der Form

$$k^{\mu_{\sigma(1)}}k^{\mu_{\sigma(2)}},\quad ak^{\mu_{\sigma(1)}}p^{\mu_{\sigma(2)}},\quad a^2p^{\mu_{\sigma(1)}}p^{\mu_{\sigma(2)}}\,. \tag{3.44}$$

Die $\sigma(i)$ sind Permutationen über $\{1,2\}$. Aus Symmetriegründen verschwindet der mittlere Term mit ungerader Anzahl k. Man hat also nur das skalare Integral I_2 und das Integral mit $k^{\mu_1}k^{\mu_2}$ im Zähler auszurechnen.

Mit Hilfe der Beta-Funktion bekommt man für das skalare Integral

$$I_2 = e^{\epsilon\gamma_E}\left(\frac{-p^2}{\mu^2}\right)^{-2\epsilon}\frac{\Gamma(-\epsilon)\Gamma(1-\epsilon)^2}{\Gamma(2-2\epsilon)} = \frac{1}{\epsilon}+2-\ln\left(\frac{-p^2}{\mu^2}\right)+\mathcal{O}(\epsilon)\,. \tag{3.45}$$

Das tensorielle Integral

$$\int\frac{\mathrm{d}^Dk}{i\pi^{\frac{D}{2}}}\,\frac{k^{\mu_1}k^{\mu_2}}{[-k^2+a(1-a)(-p^2)]^2} = \frac{g^{\mu_1\mu_2}}{D}\int\frac{\mathrm{d}^Dk}{i\pi^{\frac{D}{2}}}\,\frac{k^2}{[-k^2+a(1-a)(-p^2)]^2} \tag{3.46}$$

kann ähnlich gelöst werden. Der Fall $s > 0$ wird in Abschnitt 3.2.8 besprochen und besitzt eine ebenso einfache Lösung, wenn vorausgesetzt werden kann, dass alle Indizes des Integrals mit vierdimensionalen Größen kontrahiert werden. Diese Voraussetzung ist bei uns immer erfüllt. Alle Formeln sind im Anhang B angegeben.

3.2.7 Problempunkte im Phasenraum

Wie in Kapitel 2 beschrieben, ist das Auftreten von Gram-Determinanten im Nenner eines der größten Probleme bei Tensorreduktionsverfahren, da an bestimmten Punkten im Phasenraum diese Determinanten sehr klein bzw. Null werden. Die problematischen Fälle werden im folgenden der Reihe nach diskutiert.

1. Sind bei einem Integral alle äußeren Impulse massiv, dann muss zur Bestimmung von $l_{1,2}$ die Größe Δ (siehe (3.33)) verwendet werden. Dieses Δ ist identisch mit der zweidimensionalen Gram-Determinante. Der Term

$$\beta := \frac{1}{1-\alpha_1\alpha_2}\,, \tag{3.47}$$

der bei der Konversion zwischen $l_{1,2}$ und $p_{x,y}$ als Vorfaktor auftritt, ist im Grenzfall linear abhängiger äußerer Impulse gleich

$$\beta = \pm\frac{p_x^2}{2\alpha_1\sqrt{\Delta}}\,. \tag{3.48}$$

Die Skalarprodukte $(2k\cdot l_{1,2})$ in (3.15) erhalten bei der Umschreibung in die reduzierenden Skalarprodukte $(2k\cdot p_i)$ einen Faktor β. Bei jedem dieser Reduktionsschritte wird also eine Wurzel der Gram-Determinante in den Nenner gebracht. Allerdings geschieht dies nur für den Fall, dass alle äußeren Impulse massiv sind.

2. In den Formeln (3.21) und (3.40) steht der Term $\langle l_{1,2}-|\not{p}_3|l_{2,1}-\rangle$ im Nenner. Wie man leicht nachrechnen kann ist

$$|\langle l_{1,2}-|\not{p}_3|l_{2,1}-\rangle|^2 = \frac{1}{2}\mathrm{Spur}\big(\not{l}_1\not{p}_3\not{l}_2\not{p}_3\big) = \frac{2}{(2l_1\cdot l_2)}\Delta_3\,, \tag{3.49}$$

wobei Δ_3 die dreidimensionale Gram-Determinante ist. Also sind die Terme $\langle l_{1,2}-|\not{p}_3|l_{2,1}-\rangle$ proportional zur Wurzel dieser Gram-Determinante.

3. Das Skalarprodukt $(2l_1\cdot l_2)$ läßt sich schreiben als

$$(2l_1\cdot l_2) = \frac{p_x^2 p_y^2}{(2p_x\cdot p_y)\mp\sqrt{\Delta}}\,. \tag{3.50}$$

Ist p_x^2 oder p_y^2 Null, dann wird auch das Skalarprodukt Null und es treten Probleme auf. Allerdings ist der Fall, dass beide p^2 Null sind, eine richtige physikalische Singularität (kollinear) und wird daher sowieso aus den Observablen entfernt. Die Fälle in denen ein p^2 Null ist, lassen sich so behandeln, dass man die $l_{1,2}$ durch andere p definiert. Hat man genügend äußere Impulse zur Auswahl, $N \geq 4$, kann man statt des massiven p_x bzw. p_y einen anderen, masselosen Impuls zur Definition von $l_{1,2}$ auswählen und hat dann wieder den obigen Fall einer physikalischen Singularität. Für $N = 3$ kann man den massiven Impuls umdefinieren. Nimmt man p_x als massiv an, lautet die Umdefinition $p_x \to p_x - p_y$. Die Singularität hat dann die Form des β-Faktors wie unter Punkt 1.

Das Problem der verschwindenden Gram-Determinanten ist also auch bei dieser Reduktionsmethode vorhanden. Allerdings tritt nur noch die Wurzel der Gram-Determinante auf und das numerische Problem wird entsprechend abgeschwächt. In der Praxis ist diese Abschwächung ausreichend, da die Singularitäten nun integrabel sind und numerisch stabil behandelt werden können. Die Phasenraumpunkte, an denen die Gram-Determinante genau Null ist, werden bei der Integration ausgelassen.

3.2.8 Skalare Basisintegrale

Mit dem bisher beschriebenen Verfahren lassen sich alle Integrale auf die Form

$$I_N^s = e^{\epsilon\gamma_E}\mu^{2\epsilon}\int \frac{\mathrm{d}^D k}{i\pi^{\frac{D}{2}}}\,\frac{(k^2_{(-2\epsilon)})^s}{k_1^2\cdots k_N^2} \tag{3.51}$$

bringen. Für $s = 0$ haben wir I_2 berechnet. Die 3- und 4-Punkt-Integrale I_3 und I_4 sind aufwändiger zu berechnen, die Lösungen können aber noch in geschlossener Form angegeben werden. Die höheren Integrale I_N mit $N \geq 5$ müssen rekursiv mit Verfahren berechnet werden, die im nächsten Abschnitt beschrieben werden. In Anhang B sind alle für diese Verfahren benötigten Basisintegrale mit ihren Lösungen notiert.

Es bleibt noch, die Fälle mit $s > 0$ zu betrachten. Wie zum Beispiel in [24] ausgeführt, kann das Integral mit $(k^2_{(-2\epsilon)})^s$ im Zähler als $(D + 2s)$-dimensionales Integral geschrieben werden:

$$\int \frac{\mathrm{d}^{4-2\epsilon}k}{i\pi^{2-\epsilon}}\,\frac{(k^2_{(-2\epsilon)})^s}{k_1^2\cdots k_N^2} = \frac{\Gamma(s-\epsilon)}{\Gamma(-\epsilon)}\int \frac{\mathrm{d}^{4+2s-2\epsilon}k}{i\pi^{2+s-\epsilon}}\,\frac{1}{k_1^2\cdots k_N^2}\,. \tag{3.52}$$

Da der Faktor

$$\frac{\Gamma(s-\epsilon)}{\Gamma(-\epsilon)}$$

bei der Taylor-Entwicklung einen Faktor ϵ liefert, müssen nur die Integrale berücksichtigt werden, die UV- oder IR-Divergenzen besitzen. Wie man sich überlegen kann, sind die höherdimensionalen Integrale alle IR-konvergent. Die N-Punkt-Integrale I_N^s mit $N \geq 5$ sind außerdem UV-konvergent, so dass sie für $s > 0$ immer Null ergeben. Da in der von uns gewählten Eichung $r \leq N$ gilt und die Tensorreduktion die Ungleichung $r + 2s \leq N$ respektiert, müssen nur die Integrale $I_2^{s\leq 2}$, $I_3^{s=1}$ und $I_4^{s=2}$ berechnet werden. I_2^s ist im Anhang B notiert, die anderen beiden Lösungen lauten:

$$I_3^{s=1} = \frac{1}{2} + \mathcal{O}(\epsilon) \tag{3.53}$$

und

$$I_4^{s=2} = -\frac{1}{6} + \mathcal{O}(\epsilon)\,. \tag{3.54}$$

3.3 Skalare Reduktion

Es bleibt, die skalaren Integrale auszuwerten. Die 2-Punkt-Funktion wurde schon behandelt. Ergebnisse für die 3- und 4-Punkt-Funktionen findet man in der Literatur. Die für uns

wichtigen Fälle mit internen masselosen Linien wurden vor kurzem in kompakter Form berechnet [25, 26]. Die Formeln sind meist sehr kurz und enthalten neben elementaren Funktionen nur Dilogarithmen. Sie werden in dieser Arbeit nicht weiter besprochen. Eine Übersicht über alle Formeln findet sich in [27] und in Anhang B.

Die Berechnung von N-Punkt-Funktionen mit $N > 4$ unterscheidet sich prinzipiell von der niedrigerer Funktionen. Auf Grund überbestimmter Kinematik können höhere Funktionen jeweils durch niedrigere Funktionen ausgedrückt werden. Die entsprechenden Reduktionsformeln sind für spezielle Fälle schon seit langem bekannt [28]. In jüngster Zeit wurden die Reduktionsformeln soweit entwickelt, dass sie praktisch für alle N-Punkt-Funktionen die komplette Reduktion auf Boxgraphen beschreiben. Probleme bei der Behandlung von IR-divergenten Diagrammen wurden dabei gelöst [29, 30, 31]. Im folgenden werden die Reduktionsformeln besprochen.

3.3.1 Grundlegende Definitionen

Um die Reduktionsformel in ihrer kompakten Form besprechen zu können, müssen zunächst noch ein paar Definitionen eingeführt werden. Die sogenannte *Streichung* von Propagatoren (engl.: *pinching*) ist eine zentrale Operation, die wie folgt zu verstehen ist. Hat man ein Integral der Form

$$I_N = e^{\epsilon\gamma_E}\mu^{2\epsilon}\int \frac{\mathrm{d}^D k}{i\pi^{\frac{D}{2}}}\frac{1}{k_1^2\cdots k_N^2}\,, \tag{3.55}$$

dann nennt man das durch Streichung des i-ten Propagators entstehende Integral

$$I_N[i] = e^{\epsilon\gamma_E}\mu^{2\epsilon}\int \frac{\mathrm{d}^D k}{i\pi^{\frac{D}{2}}}\frac{1}{k_1^2\cdots k_{i-1}^2 k_{i+1}^2\cdots k_N^2} \tag{3.56}$$

das i-reduzierte Integral. Die Nummer des gestrichenen Propagators wird als Argument in Klammern notiert. Graphisch veranschaulicht wird der englische Name *pinching* für die Operation klar, siehe Abbildung 3.4.

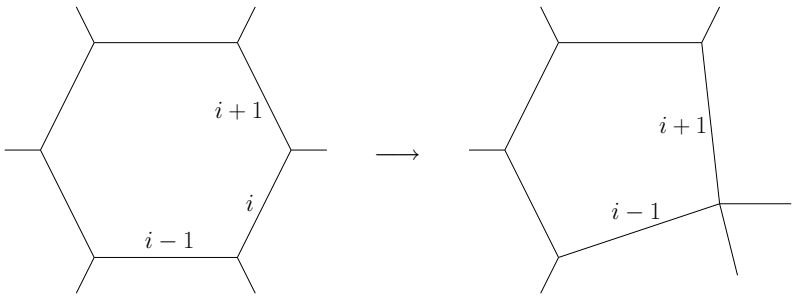


Abbildung 3.4: Streichung (*pinching*) des i-ten Propagators

Die beiden äußeren Beine werden praktisch zusammengeklemmt (*pinched*). Zu beachten ist dabei, dass das neue Integral von veränderten äußeren Impulsen abhängt. Aus p_{i-1} wird $p_{i-1} + p_i$.

Eine weitere wichtige Größe ist die kinematische Matrix, denn die folgenden Reduktionsformeln beruhen alle auf der Lösung von Gleichungsystemen, die durch die kinematische Matrix oder die Gram-Matrix definiert sind. Die kinematische Matrix ist definiert als

$$S_{ij} = (q_i - q_j)^2 \,. \tag{3.57}$$

3.3.2 5-Punkt-Funktionen

Die Reduktionformel für 5-Punkt-Funktionen wurde in [32] berechnet und lautet

$$I_5 = \sum_{i=1}^{5} b_i I_4[i] + \mathcal{O}(\epsilon) \,. \tag{3.58}$$

Der Term $\mathcal{O}(\epsilon)$ besteht im wesentlichen aus einer 5-Punkt-Funktion in $6-2\epsilon$ Dimensionen und einem Faktor ϵ. Da die 5-Punkt-Funktion in dieser Dimension keine Divergenzen mehr enthält, gehört der Term insgesamt zur Ordnung ϵ und braucht von uns nicht weiter betrachtet zu werden.

Die Größen b_i sind definiert als

$$b_i = \sum_{j=1}^{5} (S^{-1})_{ij} \,, \tag{3.59}$$

also als Zeilensumme der invertierten kinematischen Matrix S. Die Boxgraphen $I_4[i]$ gehen je durch Streichung des i-ten Propagators aus dem ursprünglichen Integral hervor.

3.3.3 6-Punkt-Funktionen

Die Formel für die 6-Punkt-Funktion wurde in [33] berechnet:

$$I_6 = \sum_{i=1}^{6} b_i \, I_5[i] \,. \tag{3.60}$$

Die Größen b_i sind definiert als

$$b_i = \sum_{j=1}^{6} (S^{-1})_{ij} \,. \tag{3.61}$$

3.3.4 N-Punkt-Funktionen

Für $N \geq 7$ ist die Determinante der kinematischen Matrix im Allgemeinen nicht mehr ungleich Null. Daraus folgt, dass man die Inverse nicht mehr bilden kann. Die Reduktionsformeln für $N = 5$ und $N = 6$ lassen sich deshalb zunächst nicht verallgemeinern.

In [30, 31] wurde dieses Problem gelöst. Die Reduktionsformel lautet

$$I_N = \sum_{i=1}^{n} r_i \, I_{N-1}[i] \tag{3.62}$$

und sieht den Formeln für 5- und 6-Punkt-Funktionen sehr ähnlich. Allerdings ist die Bestimmung der Koeffizienten r_i nun etwas aufwändiger. Da das zugrundeliegende Gleichungssystem streng genommen nicht mehr lösbar ist, also die Inversion der Matrix nicht mehr möglich ist, kommt die *Singulärwert-Zerlegung* zum Zuge. Eine genauere Beschreibung der Methode zusammen mit einer konkreten softwaretechnischen Implementierung findet sich in [34]. Die Gram-Matrix wird dabei wie folgt in drei Komponenten zerlegt:

$$G_{ij} = \sum_{k=1}^{4} U_{ik} w_k \left(V^T\right)_{kj} . \tag{3.63}$$

U und V sind orthogonale Matrizen. Der Vektor w_k enthält die Singulärwerte.

Danach lassen sich die r_i berechnen als

$$r_i = \frac{V_{i5}}{W_5} \quad \text{für} \quad 1 \leq i \leq N-1 \quad \text{und} \quad r_N = -\sum_{j=1}^{N-1} r_j . \tag{3.64}$$

Der Wert W_5 ist gegeben als

$$W_5 = \frac{1}{2} \sum_{j=1}^{N-1} G_{jj} V_{ji} . \tag{3.65}$$

Damit ist die skalare Reduktion vollständig.

3.4 Implementierung

In den vorigen Abschnitten wurden die Details eines Verfahrens zur Berechnung von masselosen Einschleifen-n-Punkt-Funktionen beschrieben und erklärt. Die praktische Umsetzung dieses Verfahrens in Form eines Computerprogramms wurde gemeinsam von A. van Hameren, S. Weinzierl und mir unternommen [27]. Ziel war es, ein Computerprogramm zu schreiben, das nach Angabe der äußeren Impulse, der Tensorstruktur und der Renormierungsskala das entsprechende Schleifenintegral berechnet. Sowohl die genannten Eingabeparameter als auch das Ergebnis sollen dabei rein numerischer Art sein. Das Programm soll sich später mit anderen Komponenten, wie z.B. der rekursiven Baumgraphengenerierung, kombinieren lassen und so ein Teil eines größeren Projekts zur Berechnung von NLO-Jet-Ereignissen werden.

Wir haben uns bei der Implementierung auch um die Einhaltung von Prinzipien des Software-Engineerings bemüht, wie der Wartbarkeit, Erweiterbarkeit oder Robustheit. Neben prinzipiellen Überlegungen war dafür auch konkret der Einsatz des Programms als Teilkomponente in einem komplexen System verantwortlich. Ein solches System soll hoffentlich von vielen verschiedenen Forschergruppen benutzt und gegebenenfalls angepasst werden können. Die verwendeten Datentypen einfach austauschen zu können, um eine höhere numerische Präzision zu erreichen, ist beispielsweise ein naheliegender Wunsch, der durch diesen Ansatz erleichtert wird.

Da die Software nur Werte für einen Teil des invarianten Matrixelements $|\mathcal{M}|^2$ liefert, also für die Berechnung zum Beispiel eines Wirkungsquerschnitts noch die Phasenraumintegration auszuführen bleibt, ist die Geschwindigkeit des Programms von entscheidender

Bedeutung. Die Phasenraumintegration wird in der Regel numerisch mittels Monte-Carlo-Verfahren gemacht. Das bedeutet, dass für die Berechnung eines konkreten Wirkungsquerschnitts unser Programm vielleicht mehrere hunderttausend Mal aufgerufen wird und jede Ineffizienz in der Programmierung entsprechend vielfach auf den Anwender zurückfällt. Die daher wichtige Optimierung ist im übernächsten Abschnitt beschrieben.

3.4.1 Verifizierung

Ganz zentral bei der Entwicklung ist jedoch, die Korrektheit des Programms zu gewährleisten. In unserem Fall wird dies erheblich durch die Tatsache erleichtert, dass am Ende eine einfache Zahl als Ergebnis geliefert wird, die man vergleichen kann. Neben dem Vergleich mit anderen etablierten Programmen für bestimmte Spezialfälle oder dem Test von pathologischen Eingabewerten, bei denen das Ergebnis leicht per Hand berechnet werden kann, ist der Vergleich verschiedener Implementierungen des gleichen Algorithmus sehr nützlich.

Wir haben nicht gemeinsam an einem Programm gearbeitet, sondern jeder hat für sich ein eigenes Programm geschrieben. Die Mehrfachprogrammierung lohnt sich, da der eigentliche Programmieraufwand gegenüber dem zu erwartenden Aufwand für die Fehlersuche kleiner ist. Die Fehlersuche wird erheblich durch den möglichen Vergleich zwischen den verschiedenen Implementierungen erleichtert. Auch eine Beurteilung der Optimierung fällt leichter. Für die verschiedenen Implementierungen haben wir auch verschiedene Programmiersprachen benutzt: C++ und FORTRAN. Im folgenden soll trotzdem manchmal nur von *einem* Programm gesprochen werden, wenn diese Tatsachen nicht von Bedeutung sind.

Der erste Schritt war die Programmierung der grundlegenden, mathematischen Funktionen und der Hilfsfunktionen für die Kinematik. Das sind z.B. der Dilogarithmus oder das Skalarprodukt zwischen zwei Spinoren. Danach folgte dann die Programmierung der Basisintegrale. Der Vergleich mit existierenden, gut getesteten Programmen wie *FF*[5] war teilweise möglich. Der Vergleich der unterschiedlichen Implementierungen, die zum Teil auch auf unterschiedlichen Formeln aus der Literatur beruhen, sichert hier gegen Fehler ab. Der nächste Schritt war die Implementierung aller skalaren Integrale, d.h. der skalaren Reduktion. Hier halfen zur Fehlerfindung nur der Vergleich zwischen unseren Implementierungen und kleinere Tests, wie z.B. die Verschiebung der Propagatornumerierung. Im letzten Schritt wurde dann die Tensorreduktion programmiert. Hier ist die Anzahl an Eingabeparametern schon recht hoch, aber weil die zugrunde liegenden skalaren Integrale gut getestet wurden, kann man die Tests noch übersichtlich halten. Neben dem gegenseitigen Vergleich konnten hier weitere Tests angewandt werden. Die Permutation der Tensorindizes darf z.B. keinen Einfluss auf das Ergebnis haben. Auch lassen sich niederrangige Reduktionen für Spezialfälle mit masselosen äußeren Spinoren per Hand durchführen und das automatische Ergebnis dagegen vergleichen.

Unsere Programme berechnen nicht nur den endlichen Teil der Beiträge, d.h. den Koeffizienten C_0 in der ϵ-Reihe, sondern auch die divergenten Teile, also die Koeffizienten C_{-2} und C_{-1} in der Reihe

$$C_{-2}\frac{1}{\epsilon^2} + C_{-1}\frac{1}{\epsilon} + C_0 + \mathcal{O}(\epsilon)\,.$$

[5] *FF* ist in [35] beschrieben.

Dies bietet weitere Möglichkeiten zum Vergleich.

Eine besondere Fehlerquelle bei numerischen Verfahren stellt die Unterscheidung von Impulsen in masselose und massive Impulse dar. Ein Impuls ist masselos, wenn sein Quadrat $p^2 = p_0^2 - p_1^2 - p_2^2 - p_3^2$ gleich Null ist. Die Kernfrage ist: wann ist eine Null eine Null? Die endliche Präzision der gewöhnlichen Datentypen führt meistens zu Werten, die nicht exakt Null sind, sondern nur unterhalb einer gewissen Schranke liegen. Werden mehrere Impulse addiert oder subtrahiert, oder wie l_1 und l_2 mittels einer Lösung einer quadratischen Gleichung bestimmt, dann können masselose Impulse ein Impulsquadrat besitzen, das weit von der Null entfernt ist. Ein Vergleich mit massiven Impulsen zeigt, dass es immer einen signifikanten Unterschied in der Größenordnung zwischen massiven und masselosen Impulsen gibt, den man ausnutzen kann, um masselose Impulse eindeutig zu identifizieren. Allerdings gibt es keine, von den Eingabeparametern unabhängige Schranke. Die Unterscheidung von masselos und massiv ist bei der Auswahl der richtigen Formeln für bestimmte Basisintegrale wichtig. Dort werden je nach Konfiguration der äußeren Impulse unterschiedliche Spezialformeln nötig. Eine falsche Entscheidung kann das Ergebnis stark verändern. Tückisch ist diese Fehlerquelle dadurch, dass nur unter bestimmten Umständen und für bestimmte Fälle die Fehler auftreten. Gelöst wird das Problem durch die Einführung einer weiteren Variable zusätzlich zu den vier Komponenten des Impulses. Diese Variable enthält die Information, ob der Impuls massiv oder masselos ist, und muss vom Benutzer zu Beginn richtig gesetzt werden. Bei den meisten Rechenschritten können diese Variablen durch das Programm korrekt verändert werden. In Zweifelsfällen greift das Programm auf die Überprüfung des Impulsquadrats mittels einer Schranke zurück. Die Schranke ist nicht fest eingestellt, sondern wird nach der Größenordnung der äußeren Impulse gewählt.

3.4.2 Optimierung

Als Computer kam für alle Tests und Geschwindigkeitsmessungen ein Athlon64 3400+ System mit 2 GB Speicher zum Einsatz. Compiliert wurde die Software für den 32bit-Modus mit GNU g++ 4.0 und der Optimierungseinstellung `-O2`.

Die erste Implementierung war überhaupt nicht optimiert. Zum Beispiel benötigte die Berechnung einer Einschleifen-5-Punkt-Funktion vom Tensorrang 5 31.68 Sekunden. Diese Zeit ist „spürbar“ und macht den Einsatz in einem Monte-Carlo-Integrator unpraktikabel, da dort eine solche Rechnung mehrere hunderttausend oder millionen Mal ausgeführt werden muss.

Analysiert man das Programm, dann findet man als Hauptgrund für die schlechte Leistung, dass im Ablauf der Rekursionsschritte sowohl für die Tensorreduktion als auch die skalare Reduktion viele Integrale mehrfach berechnet werden. Bei der sklaren Reduktion ist es zum Beispiel egal, ob man zuerst die Propagatoren $m, m+1$ und dann $n, n+1$ vereint, oder ob man umgekehrt erst $n, n+1$ und dann $m, m+1$ vereint. Man landet in beiden Fällen bei dem gleichem Integral. Sobald also mindestens eine zweifache skalare Reduktion, konkret für N-Punkt-Funktionen mit $N \geq 6$, nötig wird, treten Redundanzen auf. An den Rekursionsformeln für die Tensorreduktion kann man etwa sehen, dass bestimmte Reduktionsschritte nur den Rang vermindern, andere den Rang und die Anzahl Propagatoren gleichzeitig reduzieren. Es ist klar, dass dadurch auch wieder auf verschie-

	2-Punkt-Funktion		skalare 4-Punkt-F.		
	Anzahl der Aufrufe	Laufzeit in %	Anzahl der Aufrufe	Laufzeit in %	Gesamtlaufzeit in Sekunden
nicht-optimiert	2254496	22.9	1366692	14.3	31.68
optimiert	386	4.3	75	1.1	0.02

(a) Ergebnisse für die 5-Punkt-Rang-5-Funktion.

	skalare 4-Punkt-F.		skalare 5-Punkt-F.		
	Anzahl der Aufrufe	Laufzeit in %	Anzahl der Aufrufe	Laufzeit in %	Gesamtlaufzeit in Sekunden
nicht-optimiert	151200	53.0	30240	88.8	33.95
optimiert	210	5.5	252	16.0	0.77

(b) Ergebnisse für die skalare 10-Punkt-Funktion.

Tabelle 3.1: Analyse der Programmlaufzeit für zwei verschiedene Integral-Funktionen. Gezeigt ist die Anzahl der Aufrufe bestimmter, zugeordneter C++-Funktionen und deren prozentualen Anteils an der Gesamtlaufzeit. Je zwei repräsentative C++-Funktionen sind dargestellt. Alle Zahlen sind für den optimierten und den nicht-optimierten Fall angegeben.

denen Wegen gleiche Integrale berechnet werden müssen. Wie drastisch die Redundanz ist, kann man den Tabellen 3.1(a) und 3.1(b) entnehmen.

Diese Tabellen enthalten Zahlen über den internen Ablauf des Programms für zwei verschiedene Testfälle[6]. Einmal wurde das Programm gestartet, um eine 5-Punkt-Rang-5-Funktion zu berechnen, und einmal wurde es für die Berechnung einer skalaren 10-Punkt-Funktion gestartet. Es sind die Zahlen für jeweils das nicht-optimierte und das optimierte Programm gezeigt. Neben der Gesamtlaufzeit des Programms, findet man in beiden Beispielen für zwei häufig auftretende C++-Unterprogramme die Anzahl der Aufrufe und deren prozentualen Anteil an der Gesamtlaufzeit des Programms. Diese C++-Funktionen erledigen jeweils die komplette Berechnung, die ihnen zugedacht ist und liefern als Ergebnis eine Zahl. Die C++-Funktion für die 2-Punkt-Funktion in Tabelle 3.1(a) behandelt sowohl skalare als auch tensorielle Integrale. Die anderen zwei Funktionen berechnen nur skalare Funktionen. An der Berechnung der Beispiele sind viele C++-Funktionen beteiligt. Die drei C++-Funktionen in Tabelle 3.1(a) wurden ausgewählt, weil sich an ihnen die Redundanz am deutlichsten zeigt.

Die Optimierung besteht einzig und allein darin, dass sich das Programm für jeden Satz von Eingabeparametern, der bei dem Aufruf einer bestimmten C++-Funktion gegeben wird, das Ergebnis dieser Funktion merkt. Bevor die Funktion zu rechnen beginnt, prüft

[6] Die Daten wurden mit Hilfe des Programms `gprof` gewonnen. `gprof` ist ein weit verbreitetes und viel benutztes Hilfsprogramm, mit dem sich der Ablauf eines kompilierten Computerprogramms analysieren lässt.

sie immer zuerst, ob nicht schon einmal für diese Eingabeparameter eine Rechnung ausgeführt wurde. Falls ja, verwendet sie das gespeicherte Ergebnis. Diese Optimierung lässt sich einfach in die rekursive Struktur des Programms einbauen und besteht im wesentlichen nur in der Verwaltung einer Datenbank. In C++ kann eine solche Datenbank ganz einfach mit der vorhandenen Datenstruktur `map<>` programmiert werden, die einen assoziativen Speicher zur Verfügung stellt, bei dem man mittels eines Schlüssels (eine geeignete Codierung der Eingabeparameter) einen Wert (das Ergebnis als Zahl) suchen lassen kann. Die numerische Präzision bleibt bei dieser Optimierungmethode unberührt. Die Genauigkeit der Ergebnisse mit und ohne Optimierung ist gleich.

Die Tabellen zeigen, wie gewaltig der Effekt einer solchen einfachen Optimierung ist. Anstatt mehrere millionen Mal aufgerufen zu werden, wird die 2-Punkt-Funktion nur noch wenige hundert Mal zur Berechnung bemüht. Die Auswirkung auf die Laufzeit des Programms ist groß: das Programm läuft um mehr als einen Faktor Tausend schneller. Die beiden gezeigten Rechnungen demonstrieren in gewissem Maße Extremfälle, zum einen ein hoher Rang, zum anderen eine hohe Anzahl Teilchen. In praktisch relevanten Fällen wird man meist irgendwo zwischen diesen beiden Extremen liegen.

Bei den Zahlen fällt auf, dass die Gesamtlaufzeit nicht in dem selben Maße abnimmt, wie die Anzahl der Funktionsaufrufe. Das Gleiche kann man auch an den prozentualen Anteilen an der Gesamtlaufzeit der einzelnen Funktionen erkennen. Während im nicht-optimierten Fall der Großteil der Rechenzeit auf die eigentlichen Rechenfunktionen entfällt, ist das im optimierten Fall nicht mehr so. Eine Analyse zeigt, dass nun ein großer Teil der Zeit auf das Suchen in der Datenbank entfällt. Für die 5-Punkt-Rang-5-Funktion zum Beispiel verwendet das Programm knapp 47% seiner Zeit auf das Suchen in der Datenbank. Der andere große Teil der Zeit wird für das Erzeugen und Vernichten von Daten verbraucht. Dies sind die Operationen, bei denen z.B. aus der gegebenen Liste von Spinoren eine neue, verkürzte Liste erzeugt werden muss.

Betrachtet man die Leistung des Programms aus dem Blinkwinkel eines größeren Gesamtprojekts, so ist zwar jede noch so kleine Optimierung irgendwann einmal wünschenswert, die einfache, oben beschriebene Optimierung gegnügt aber schon, um mit dem Projekt vernünftig fortfahren zu können. D.h. man kann weitere Optimierungen später angehen und sich zuerst um die Vollendung des eigentlichen Projektziels kümmern. Die Analyse hat gezeigt, dass die Optimierung der mathematischen Operationen praktisch kaum relevant ist. Optimiert werden können vor allem die Datenbank und die Operationen auf den Datenstrukturen der Eingabeparameter. Anstatt in einem assoziativen Speicher, wie der von `map<>`, zu suchen kann man auch die Codierung der Eingabeparameter so geschickt wählen, dass ein Vergleich in konstanter Zeit möglich wird. Es gibt aber durchaus Argumente gegen solche Optimierungen an diesem Punkt der Softwareentwicklung. Die bisherige Implementierung zeichnet sich durch ein hohes Maß an Allgemeinheit aus, und der Code gibt die Struktur der mathematischen Methoden wieder, was ihn lesbar, wartbar und erweiterbar macht. Die meisten weitergehenden Optimierungen würden diese Strukturen zerstören oder vernebeln. Da das Projekt noch nicht seine Endstufe erreicht hat und noch viele Änderungen am Code notwendig werden könnten, scheint es daher sinnvoll, diese Optimierungen in die Zukunft zu verlagern.

3.4.3 Ergebnisse

Das Ergebnis besteht im Wesentlichen in der Fertigstellung von drei verschiedenen Programmen, die alle die Berechnung von masselosen Einschleifen-N-Punkt-Funktionen ausführen können und für gegebene Eingabeparameter die Koeffizienten der ϵ-Pole und des endlichen Anteils liefern. Die Programme wurden in [27] vorgestellt. In dieser Arbeit finden sich auch weitere Details zu Testdaten mit Ergebnissen, die hier nicht gezeigt wurden.

4 Analytische Berechnung von Schleifenintegralen

Nachdem im vorigen Kapitel ein Projekt zur numerischen Berechnung von Einschleifen-Integralen vorgestellt wurde, soll nun in diesem Kapitel auf analytische Verfahren eingegangen werden. Wir stellen das xloops-Projekt vor, mit dem man Feynman-Diagramme weitgehend analytisch berechnen lassen kann. Nach der Beschreibung der analytischen Methoden in xloops stellen wir neue Ideen und Möglichkeiten zu deren erweiterten Anwendung vor.

4.1 Das xloops-Projekt

Im Jahr 1991 wurde von Dirk Kreimer die Technik, Impulse in einen Orthogonal- und einen Parallelraumanteil zu zerlegen, auf die Berechnung von massiven Zweischleifen-Integralen angewandt [36, 37]. Damit ließen sich diese Integrale prinzipiell auf eine neue, effiziente Weise berechnen. Um das Verfahren praktisch zu erproben, wurde ein Software-Projekt begründet, mit dem Ziel eine funktionsfähige Implementierung zu schaffen. Daraus entwickelte sich bald die Idee, ein allgemeineres Programm zu schreiben, das automatisiert die störungstheoretischen Beiträge zu physikalischen Prozessen in Ein- und Zweischleifen-Ordnung berechnen kann: xloops.

xloops wurde mit dem Computeralgebrasystem *Maple* programmiert. Zusätzlich wurden eine Reihe weiterer Programmiersprachen für spezielle Aufgaben, wie z.B. die graphische Benutzerschnittstelle oder die numerische Integration, benutzt [38]. Bevor noch das hauptsächliche Ziel von xloops, nämlich die Berechnung von Zweischleifen-Integralen, erreicht werden konnte, zwang die softwaretechnische Komplexität der Aufgabe die Programmierer zu einer kompletten Neuimplementierung von xloops auf Basis der eigens dafür geschaffenen C++–Bibliothek für Computeralgebra GiNaC [39].

Die Struktur von xloops ist in Abbildung 4.1 gezeigt. Das komplette System ist nun in nur einer Programmiersprache geschrieben. Die Programmiersprache C++ bildet die Basistechnologie des ganzen Systems. Darauf aufbauend finden sich drei externe Bibliotheken: GiNaC wurde schon erwähnt. Es erlaubt, algebraische Umformungen beliebiger symbolischer Ausdrücke als Teil von C++ zu programmieren. nestedsums benutzt GiNaC und kann spezielle Funktionen der mathematischen Physik nach kleinen Argumenten Taylor-entwickeln. Die C++-Bibliothek nestedsums wurde von Stefan Weinzierl programmiert [40]. VEGAS ist die Implementierung eines bekannten Verfahrens [41] zur numerischen Integration. Das eigentliche xloops besteht aus den Komponenten zur Berechnung von Einschleifen- und Zweischleifen-Integralen, sowie einem Graphen-Generator, der zu einem gegebenen Prozess alle beitragenden Feynman-Diagramme erzeugt, und einem kleinen interaktiven Kom-

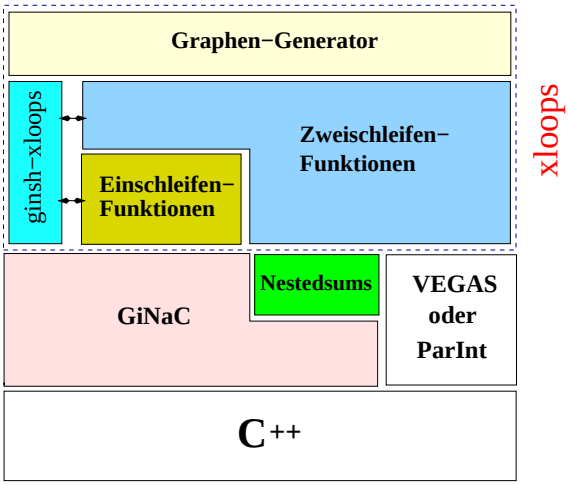


Abbildung 4.1: Struktur von xloops. Gezeigt ist der hierarchische Aufbau aus verschiedenen Einzelkomponenten. C++ bildet die Basistechnologie, auf der alle weiteren Teile aufbauen. GiNaC, nestedsums und VEGAS sind externe Bibliotheken, die von xloops genutzt werden. Darüber befinden sich die eigentlichen Komponenten von xloops.

mandozeileninterpreter *ginsh-xloops*.

Man benutzt xloops, indem man ein C++-Programm schreibt, dass die von xloops angebotenen C++-Funktionen aufruft. Es gibt zwar auch die Möglichkeit mit *ginsh-xloops* interaktiv Berechnungen auszuführen, dies ist aber nicht die primäre Anwendungsweise. xloops bietet dem Benutzer eine Reihe von C++-Funktionen an. Zur Berechnung der Einschleifen-Dreipunkt-Funktion kann der Benutzer zum Beispiel zwischen den folgenden drei Funktionen wählen

- `ex Scalar3Pt(...)`
- `ex OneLoop3Pt(...)`
- `ex OneLoopTens3Pt(...)`

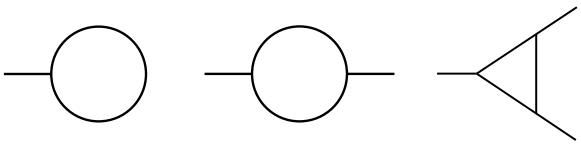
Die dazu jeweils entsprechenden Integrale

$$\int \mathrm{d}^D l \frac{1}{P_1^{t_1} P_2^{t_2} P_3^{t_3}}, \quad \int \mathrm{d}^D l \frac{l_0^{p_0} l_1^{p_1} l_\perp^{p_2}}{P_1^{t_1} P_2^{t_2} P_3^{t_3}}, \quad \int \mathrm{d}^D l \frac{l_{\mu_1} \dots l_{\mu_k}}{P_1^{t_1} P_2^{t_2} P_3^{t_3}},$$

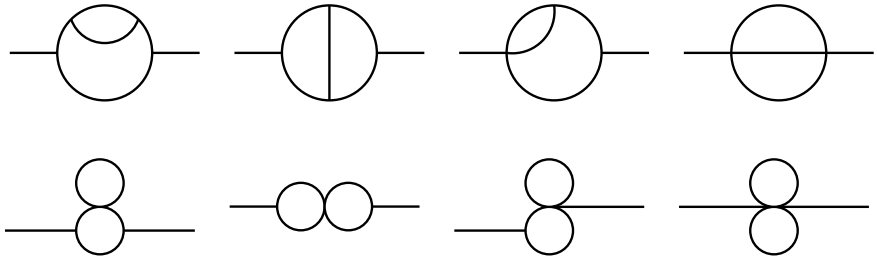
unterscheiden sich durch die Zählerstruktur. Die Funktionen erwarten eine Reihe von Parametern, wie zum Beispiel die Massen, äußeren Impulse oder Exponenten der Propagatoren, in Form von GiNaC-Datentypen `ex` (siehe Anhang C). Das berechnete Ergebnis wird genauso wieder als `ex` zurückgeliefert. `ex` kann beliebige symbolische Ausdrücke oder auch Zahlen enthalten. Die Konventionen für die Integrale werden im Abschnitt 4.2 besprochen. Es wird immer dimensionale Regularisierung verwendet. Genauere Details über die Parameter und die Integraldefinitionen werden hier nicht weiter beschrieben, können aber im Handbuch von xloops nachgelesen werden. Alle in xloops berechenbaren Integrale besitzen die drei gezeigten Typen von C++-Funktionen zum Aufrufen.

4.1.1 Status

In xloops sind zurzeit noch nicht alle denkbaren Ein- und Zweischleifen-Integrale implementiert. Die vorhandenen Integrale können mit beliebigen Massen, Propagatorexponenten und mit den drei im vorigen Abschnitt illustrierten Zählerstrukturen berechnet werden. Es ist daher ausreichend und übersichtlicher die vorhandenen Funktionen an Hand der zugrundeliegenden Topologien der Feynman-Graphen zu charakterisieren. Abbildung 4.2 zeigt die Topologien, die mit xloops berechnet werden können. Das sind zum einen die Ein- bis Dreipunkt-Einschleifen-Topologien. Zum anderen sind dies bei den Zweischleifen-Topologien die Zweipunkt-Graphen, deren Berechnung und Implementierung in [42] beschrieben ist.



(a) Einschleifen-Topologien



(b) Zweischleifen-Topologien

Abbildung 4.2: Implementierte Topologien in xloops.

Die in Abbildung 4.3 gezeigten Topologien sind zwar noch nicht in xloops berechenbar, allerdings wird momentan an deren Implementierung gearbeitet und Fortschritte sind diesbezüglich für die nähere Zukunft zu erwarten. Dazu gehören die Einschleifen-Vierpunkt-Topologie, auch Boxgraph genannt, sowie die Zweischleifen-Drei- [43] und Vier-Punkt-Topologien [44].

Die besprochenen Topologien bzw. Integrale werden dem Benutzer von xloops durch C^{++}-Funktionen zugänglich gemacht. Diese C^{++}-Funktionen bilden eine Ebene der Benutzerschnittstelle, d.h. des Programminterfaces, von xloops. Wie man in Abbildung 4.1 sehen kann, ist das aber nicht die oberste Komponenten-Ebene in xloops. Darüber existiert noch der Graphen-Generator, der eine vollkommen andere Steuerung von xloops erlaubt. Statt einzelne Integrale ausrechnen zu lassen, kann sich der Benutzer durch Vorgabe eines physikalischen Prozesses das entsprechende Matrixelement $|\mathcal{M}|^2$ berechnen lassen. Die beteiligten Feynman-Diagramme werden dabei automatisch erzeugt, berechnet und geeignet kombiniert. Der Graphen-Generator existiert bereits als eigenständiges Softwareprogramm [45], ist aber bis jetzt noch nicht in xloops integriert. In Abbildung 4.4 ist der Ablauf einer

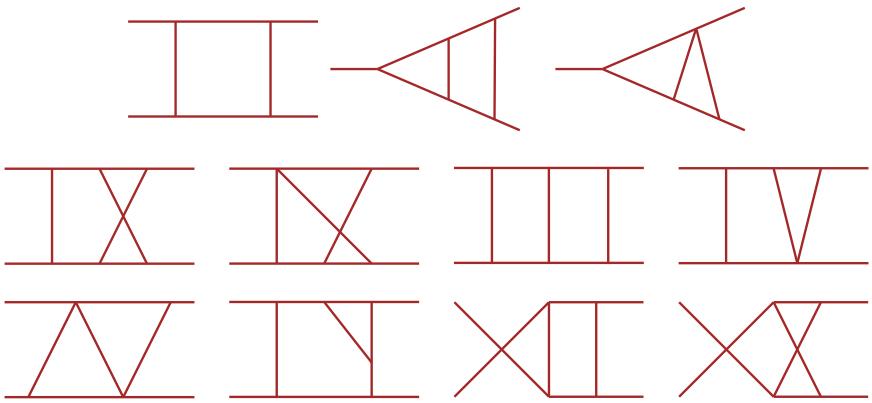


Abbildung 4.3: Topologien an denen gearbeitet wird und die noch nicht in xloops implementiert sind. Das sind der Boxgraph und verschiedene Drei-Punkt- und Vier-Punkt-Zweischleifen-Topologien.

automatisierten Berechnung eines Matrixelements gezeigt. Die Kästchen bezeichnen Programmkomponenten, die jeweils eine spezielle Aufgabe übernehmen. Beginnend mit der Definition eines Prozesses und des zugrunde liegenden Modells, d.h. der Benennung der äußeren Teilchen einer Reaktion und der erlaubten Wechselwirkungen mitsamt Feynman-Regeln, erzeugt der Graphen-Generator alle Feynman-Diagramme. Durch die Feynman-Regeln werden die passenden analytischen Ausdrücke erstellt und die nötigen Integralrechnungen identifiziert. Die Integrale werden dann, wenn nötig mit Hilfe numerischer Verfahren, berechnet. Abschließend werden die Ergebnisse kombiniert und als quadriertes Matrixelement zurückgeliefert.

Die in Abbildung 4.4 gestrichelt gezeichneten Komponenten sind noch nicht programmiert. Neben dem Einbau des Graphen-Generators sind also die Behandlung der Feynman-Regeln und des Matrixelements noch zu erledigende Aufgaben bei der Entwicklung von xloops.

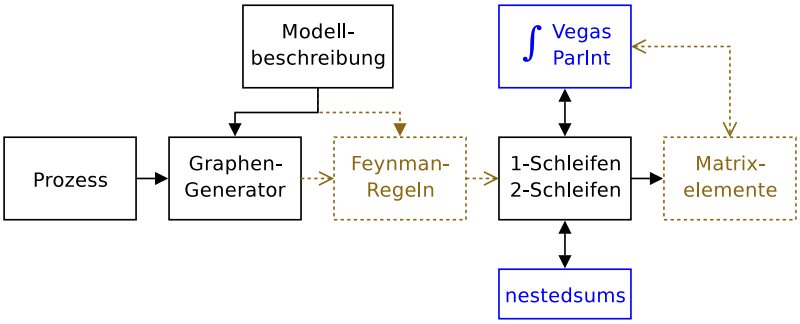


Abbildung 4.4: Ablaufdiagramm von xloops. Von links beginnend sind die einzelnen Arbeitsschritte dargestellt. Die gestrichelten Elemente sind noch nicht vorhanden.

4.2 Analytische Techniken in xloops

Wir beschreiben nun die Methoden zur Berechnung von Feynman-Integralen, wie sie in xloops angewendet werden. Die Darstellung wird viele Details auslassen und sich auf die Aspekte konzentrieren, die für das Verständnis des eigentlichen Ziels dieses Kapitels, nämlich die Vorstellung neuer Ideen und Ansätze für xloops, notwendig sind. Wir beginnen mit der grundlegenden Technik der Parallel-/Orthogonalraummethode, deren Einsatz bei Zweischleifen-Integralen, wie schon erwähnt, die hauptsächliche Motivation für xloops war.

4.2.1 Die Parallel-/Orthogonalraummethode

Die dimensionale Regularisierung macht die Dimension eines Integrals zu einer Variablen D, welche beliebige Werte annehmen kann. Mathematisch abstrakt wird ein dimensional regularisiertes Integral einfach als ein Funktional betrachtet, das einer Zahl D eine Funktion zuordnet. Diese Funktion soll die wesentlichen analytischen Eigenschaften eines gewöhnlichen Integrals besitzen [1]. Neben diesen Eigenschaften ist die wichtigste Forderung, dass die Funktion für $D \to 4$ das ursprüngliche Integral reproduziert. Diese Forderung mathematisch zu formulieren führt auf natürliche Weise zur Aufteilung sowohl des Integrals als auch der Impulsvektoren in zwei Teile, von denen der erste dem ursprünglichen Problem in ganzzahliger Dimension $J = 4$ entspricht, und der zweite Teil den abstrakten $(D-J)$–dimensionalen Rest umfasst.

Die Dimension J ist ganz bewusst als Variable geschrieben worden. Besitzt das Feynman-Integral zum Beispiel weniger als vier äußere Impulse, dann lässt sich die Kinematik vereinfacht beschreiben. Man kann durch eine geeignete Lorentz-Transformation die Impulse so transformieren, dass bestimmte Vektorkomponenten durchgehend Null werden, die effektiven Impulsvektoren also in einem Unterraum mit kleinerer Dimension leben. Wenn es dann für die Rechnung günstig ist, kann man ohne Probleme J gleich dieser kleineren Dimension wählen. Die Impulse werden allgemein wie folgt notiert:

$$l = (l_0, l_1, \ldots, l_{J-1}, l_\perp) \,. \tag{4.1}$$

Dabei bilden die l_0 bis l_{J-1} den Parallelraumanteil und $l_\perp$ beschreibt den restlichen Anteil im Orthogonalraum. Das Impulsquadrat ist wie üblich definiert als

$$l^2 = l_0^2 - l_1^2 - \cdots - l_{J-1}^2 - l_\perp^2 \,. \tag{4.2}$$

Bei äußeren Impulsen

$$q = (q_0, q_1, \ldots, q_{J-1}, 0) \tag{4.3}$$

ist der Orthogonalraumanteil immer Null. Die Namen „Parallel - Orthogonal“ für die beiden Impulsanteile leiten sich von der Eigenschaft des Skalarprodukts ab. Das Skalarprodukt zwischen einem D-dimensionalen Schleifenimpuls l und einem äußeren Impuls q ist:

$$l \cdot q = l_0 q_0 - l_1 q_1 - \cdots - l_{J-1} q_{J-1} \,. \tag{4.4}$$

Die Orthogonalraumanteile verschwinden dabei immer, nur der Anteil der parallel zu den äußeren Impulsen ist, bleibt erhalten.

Die Vorschrift für ein dimensional regularisiertes Integral lautet dann

$$\int \mathrm{d}^D l f(l) = \frac{2\pi^{\frac{D-J}{2}}}{\Gamma\left(\frac{D-J}{2}\right)} \int_{-\infty}^{\infty} \mathrm{d}l_0 \cdots \int_{-\infty}^{\infty} \mathrm{d}l_{J-1} \int_{0}^{\infty} \mathrm{d}l_\perp l_\perp^{D-J-1} f(l_0, \ldots, l_{J-1}, l_\perp) \,. \tag{4.5}$$

Hier sieht man die Aufteilung in Parallelraum-Integrale und ein Orthogonalraum-Integral. Die günstige Eigenschaft des Skalarprodukts zwischen Schleifenimpuls und äußeren Impulsen hat man sich schon zunutze gemacht, um das Orthogonalraumintegral zu einem Euklidischen Integral umzuschreiben. Wegen des Skalarprodukts kommt $l_\perp$ nur quadriert vor, so dass das Orthogonalraumintegral direkt ausgeführt werden kann.

Am Beispiel der skalaren Einschleifen-Drei-Punkt-Funktion werden wir diese und alle folgenden analytischen Techniken illustrieren. Das Drei-Punkt-Integral sieht so geschrieben wie folgt aus:

$$\begin{aligned} I_3 = \frac{2\pi^{\frac{D-2}{2}}}{\Gamma\left(\frac{D-2}{2}\right)} \int_{-\infty}^{\infty} \mathrm{d}l_0 \int_{-\infty}^{\infty} \mathrm{d}l_1 \int_{0}^{\infty} \mathrm{d}l_\perp l_\perp^{D-3} \frac{1}{(l_0+q_{10})^2 - l_1^2 - l_\perp^2 - m_1^2 + i\rho} \\ \frac{1}{(l_0+q_{20})^2 - (l_1+q_{21})^2 - l_\perp^2 - m_2^2 + i\rho} \frac{1}{l_0^2 - l_1^2 - l_\perp^2 - m_3^2 + i\rho} \,. \end{aligned} \tag{4.6}$$

Die Dimension J ist hier gleich Zwei. Die äußeren Impulse wurden so transformiert, dass nur drei Komponenten ungleich Null sind:

$$q_1 = (q_{10}, 0, 0, 0), \qquad q_2 = (q_{20}, q_{21}, 0, 0) \,. \tag{4.7}$$

Der Term $i\rho$ definiert die Integrale an den Polstellen der Integranden. Am Ende der Rechnung wird $\rho \to 0$ gesetzt.

Durch eine Partialbruchzerlegung erhält man drei Summanden, die $l_\perp^2$ jeweils nur noch in einem Nenner enthalten, so dass man mit der bekannten Formel

$$\int_0^\infty \mathrm{d}s \, \frac{s^\alpha}{(s^2+z)} = \tfrac{1}{2}\Gamma\left(\tfrac{\alpha+1}{2}\right)\Gamma\left(1-\tfrac{\alpha+1}{2}\right) z^{\frac{\alpha+1}{2}-1} \tag{4.8}$$

die Orthogonalraumintegration ausführen kann. Danach hat man:

$$\begin{aligned} I_3 = -\pi^{\frac{D-2}{2}}\Gamma\left(\tfrac{-D+2}{2}+1\right) \sum_{k=1}^{3} \\ \times \int_{-\infty}^{\infty} \mathrm{d}l_0 \int_{-\infty}^{\infty} \mathrm{d}l_1 \, \frac{[-l_0^2 + l_1^2 + m_k^2 - i\rho]^{\frac{D-2}{2}-1}}{(a_{1k}l_0 + b_{1k}l_1 + c_{1k})(a_{2k}l_0 + b_{2k}l_1 + c_{2k})} \,. \end{aligned} \tag{4.9}$$

Das bemerkenswerte an dieser Methode ist, dass unabhängig von der Anzahl der äußeren Impulse maximal vier Integrationen pro Schleife auszuführen bleiben.

4.2.2 Integrationen mittels Residuensatz

Der nächste Schritt in xloops ist das Lösen eines Teils der Parallelraumintegrale mittels Residuensatz. Der Nenner in (4.9) hat Nullstellen in l_0 und l_1, die entsprechend Pole auf der reellen Achse ergeben. Der Integrationsweg kann durch einen Halbkreis im Unendlichen geschlossen werden, so dass sich der Wert des Integrals als die Summe der Residuen an den jeweiligen Polstellen ergibt. Problematisch ist dabei der Term im Zähler, der Verzweigungsschnitte in jeder Halbebene erzeugt. Dieiesser Term lässt sich aber wegen der unterschiedlichen Vorzeichen von l_0^2 und l_1^2 in einer Variable linearisieren, z.B. durch $l_0 \to x + y$, $l_1 \to y$. Damit ist der Term

$$[-2xy - x^2 + m_k^2 - i\rho]^{\frac{D-2}{2}-1} \tag{4.10}$$

in y linearisiert. Der Schnitt liegt je nach Vorzeichen von x in der oberen oder unteren Halbebene und der Integrationsweg für das y-Integral muss entsprechend unterschiedlich gelegt werden. Die zwei möglichen Lösungen des y-Integrals werden durch Stufenfunktionen als Vorfaktoren $\theta(-2x)$ und $\theta(2x)$ unterschieden, und erhalten durch die verschiedene Umlaufrichtung des Integrationswegs ein relatives Vorzeichen. Die Pole auf der reellen Achse lassen sich durch das Lösen eines linearen Gleichungssystems bestimmen. In unserem Beispiel gibt es zwei Nullstellen. Die Lösung ergibt sich dann als

$$\begin{aligned} I_3 &= -\pi^{2-\epsilon}\,\Gamma(\epsilon) \sum_{k=1}^{3} \frac{1}{a_{1k}(a_{2k}+b_{2k}) - a_{2k}(a_{1k}+b_{1k})} \sum_{l=1}^{2} (-1)^l \\ &\qquad \times \int_{-\infty}^{\infty} \mathrm{d}x \left\{\theta(2x) - \theta(-2x)\right\} \frac{[g_{kl}x^2 + h_{kl}x + m_k^2 - i\rho]^{-\epsilon}}{x + f_{kl}} \\ &= -\pi^{2-\epsilon}\,\Gamma(\epsilon) \sum_{k=1}^{3} \frac{1}{a_{1k}(a_{2k}+b_{2k}) - a_{2k}(a_{1k}+b_{1k})} \sum_{l=1}^{2} (-1)^l \\ &\qquad \times \left\{\int_{0}^{\infty} \mathrm{d}x - \int_{-\infty}^{0} \mathrm{d}x\right\} \frac{[g_{kl}x^2 + h_{kl}x + m_k^2 - i\rho]^{-\epsilon}}{x + f_{kl}}\,. \end{aligned} \tag{4.11}$$

D wurde durch $D = 4 - 2\epsilon$ ersetzt. Die Faktoren f_{kl}, g_{kl} und h_{kl} bestimmen sich durch die Nullstellen.

Auf kompliziertere Feynman-Integrale kann man das beschriebene Verfahren genauso übertragen. Im Fall von Vier-Punkt-Funktionen kann man zwei von vier Parallelraumintegralen ausführen. Das zur Linearisierung notwendige unterschiedliche Vorzeichen zwischen zwei Quadraten von l kann man immer herstellen. Die Berechnung der Nullstellen kann erheblichen algebraischen Aufwand verursachen, aber mögliche Optimierungen wurden schon in [44] entwickelt. Am Ende der Residuenintegrationen steht immer ein Ergebnis ähnlich (4.11), bei dem noch eine oder mehrere Integrationen über einem rationalen Integranden auszuführen bleiben.

4.2.3 Carlsons R-Funktion

Es gibt eine Vielfalt von speziellen Funktionen in der Mathematik, die oft als Lösungen für partielle Differentialgleichungen unterschiedlichster Art definiert sind. Beispiele sind Bessel-, Hankel-, Legendre- oder Kummer-Funktionen, um nur ein paar Namen zu nennen. Das entsprechende mathematische Fachgebiet ist sehr groß und wenig übersichtlich. Dies liegt an einem fehlenden Ordungsprinzip für die vielen Funktionen. Es gibt nur wenig Systematik. Die R-Funktion wurde von B. C. Carlson in die mathematische Forschung eingeführt [46], um diesen Mangel zu verringern. Die R-Funktion umfasst als Spezialfälle zum Beispiel die Gauß'sche Hypergeometrische Funktion, die elliptischen Integrale, die erste Appell-Funktion, und noch viele andere. Obwohl die Motivation für die Einführung der R-Funktion rein ästhetisch-mathematischer Natur war, hat sie auch bei der Berechnung von Feynman-Integralen eine wichtige Anwendung gefunden.

Die R-Funktion sieht wie folgt aus:

$$\mathrm{R}_t\big(b_1,\ldots,b_k;z_1,\ldots,z_k\big)\,. \tag{4.12}$$

Bei den Parametern unterscheidet man zwischen dem Index der Funktion t, den Gewichten b_i, sowie den eigentlichen Argumenten z_i. Es ist nützlich für die Summe der Gewichte ein eigenes Symbol

$$\beta=\sum_{i=1}^{k} b_i \tag{4.13}$$

zu definieren. Die Parameter der R-Funktion können alle beliebige komplexe Zahlen sein, mit der Einschränkung $\beta\neq 0,-1,-2,\ldots$.

Für bestimmte Parameterkombinationen lassen sich die oben angesprochenen Funktionen als Spezialfälle der R-Funktion schreiben. Zwei Beispiele sollen das veranschaulichen:

$$\mathrm{R}_t\big(b;z\big)=z^t \tag{4.14}$$

$$\mathrm{R}_t\big(b_1,b_2-b_1;z_1,z_2\big)=z_2^t\ {}_2\mathrm{F}_1(-t,b_1;b_2;1-\tfrac{z_1}{z_2})\,. \tag{4.15}$$

Das ist zum einen die einfache Potenzfunktion, und zum anderen die Gauß'sche Hypergeometrische Funktion ${}_2\mathrm{F}_1$.

Die Verwendung der R-Funktion zur Lösung von Integralen ist die zweite Besonderheit von xloops. Die R-Funktion besitzt eine Darstellung als Integral, die dies ermöglicht. Die Integral-Darstellung ist den zu berechnenden Feynman-Integralen so ähnlich, dass durch wenige Umformungen das betreffende Integral direkt mit einer R-Funktion identifiziert werden kann. Die Lösung des Integrals ist dann die R-Funktion.

Die Integral-Darstellung der R-Funktion lässt sich ganz allgemein wie folgt schreiben [46]:

$$\begin{aligned}\int_r^\infty \mathrm{d}x\ (x-r)^{\alpha-1}\prod_{i=1}^{k}(z_i+w_i x)^{-b_i} &= \mathrm{B}(\beta-\alpha,\alpha)\ \prod_{i=1}^{k} w_i^{-b_i}\\ &\quad\times \mathrm{R}_{\alpha-\beta}\Big(b_1,\ldots,b_k;r+\frac{z_1}{w_1},\ldots,r+\frac{z_k}{w_k}\Big)\,.\end{aligned} \tag{4.16}$$

Auf der linken Seite der Gleichung steht das zu lösende Integral, das wir später mit dem Integral in (4.11) in Übereinstimmung bringen müssen. r ist eine beliebige reelle Zahl, die

z_i, w_i, b_i und α sind beliebige komplexe Zahlen (mit der Einschränkung $\beta \neq 0, -1, \ldots$). Das Integral kann man direkt in eine Beta-Funktion[1] B, einen rationalen Faktor und eine R-Funktion übersetzen. Die Integral-Darstellung der R-Funktion ist entsprechend umgekehrt, als Integral geteilt durch die Beta-Funktion und den rationalen Faktor, zu lesen.

Betrachtet man (4.11), so muss vor Anwendung von (4.16) nur noch der Zähler des Integranden linearisiert werden. Die resultierende R-Funktion hat dann drei Argumente, die sich aus den zwei linearisierten Faktoren im Zähler und dem Term im Nenner von (4.11) ergeben. Die Gewichte sind zweimal ϵ und einmal 1. Für r ist der Wert Null und für α der Wert Eins zu wählen. Damit hat man die Integrale in (4.11) gelöst und kennt im Prinzip das Endergebnis:

$$\begin{aligned} I_3 = -\pi^{2-\epsilon}\,\Gamma(\epsilon) \sum_{k=1}^{3} \frac{1}{a_{1k}(a_{2k}+b_{2k}) - a_{2k}(a_{1k}+b_{1k})} \sum_{l=1}^{2} (-1)^l \left(\frac{a_{kl}-b_{kl}}{a_{kl}+b_{kl}}\right)^{-\epsilon} \\ \times \mathrm{B}(2\epsilon, 1)\Big[\mathrm{R}_{-2\epsilon}(\epsilon,\epsilon,1; u_{1kl}, u_{2kl}, u_{3kl}) - \mathrm{R}_{-2\epsilon}(\epsilon,\epsilon,1; -u_{1kl}, -u_{2kl}, -u_{3kl})\Big] \,. \end{aligned} \tag{4.17}$$

Die u_{ikl} berechnen sich aus den f_{kl}, g_{kl}, h_{kl} und m_k.

Das Endergebnis ist exakt, aber noch nicht wirklich brauchbar, da man das Ergebnis für $D = 4$, also $\epsilon \to 0$, braucht. Deshalb muss (4.17) noch nach ϵ entwickelt werden.

4.2.4 Entwicklung nach ϵ

Der schwierige Teil der Aufgabe ist die Entwicklung der R-Funktionen, die Entwicklung der anderen Faktoren ist einfach. Dazu braucht man die Darstellung der R-Funktion als Reihe [46]:

$$\begin{aligned} &\mathrm{R}_t(b_1, \ldots, b_k; z_1, \ldots, z_k) = \\ &\qquad \sum_{n=0}^{\infty} \frac{(-t,n)}{(\beta,n)} \sum_{m_1} \frac{(b_1,m_1)}{m_1!}(1-z_1)^{m_1} \cdots \sum_{m_k} \frac{(b_k,m_k)}{m_k!}(1-z_k)^{m_k} \,. \end{aligned} \tag{4.18}$$

Dabei sind die Werte der Laufindizes m_i der k Untersummen nur durch die Bedingung

$$\sum_{i=1}^{k} m_i = n, \quad m_i \geq 0 \tag{4.19}$$

festgelegt und ansonsten nicht weiter spezifiziert. Die erste Aufgabe bei der praktischen Auswertung der Reihe (4.18) ist immer, eine konkrete Realisierung für die Laufindizes zu finden, wie es auch im gleich folgenden Beispiel getan wird. Die Reihe (4.18) ist mit Hilfe des Appell-Symbols[2] geschrieben, das definiert ist als

$$(a,b) = \frac{\Gamma(a+b)}{\Gamma(a)} \,. \tag{4.20}$$

[1] Die Beta-Funktion ist definiert als $\mathrm{B}(a,b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$.

[2] In der Literatur hat die Funktion auch den Namen Pochhammer-Symbol und wird dann meist als $(a)_b$ notiert.

An einem konkreten Beispiel zeigen wir nun die Entwicklung von R. Wir schreiben die Mehrfachsumme in (4.18) für den in (4.17) benötigten Fall aus:

$$\mathrm{R}_{-2\epsilon}(\epsilon,\epsilon,1;x,y,z) = \sum_{n=0}^{\infty} \frac{(2\epsilon,n)}{(2\epsilon+1,n)} \sum_{m_1=0}^{n} \sum_{m_2=0}^{n-m_1} \frac{(\epsilon,m_1)}{m_1!} \frac{(\epsilon,m_2)}{m_2!} \frac{(1,n-m_1-m_2)}{(n-m_1-m_2)!} \times (1-x)^{m_1}(1-y)^{m_2}(1-z)^{n-m_1-m_2} \,. \tag{4.21}$$

Die Laufindizes m_1, m_2 und $m_3 = n - m_1 - m_2$ erfüllen immer die Bedingung (4.19). Der dimensionale Regularisierungsparameter ϵ steht an vier Stellen in Appell-Symbolen. Die Appell-Symbole

$$(a\epsilon, m) = a\Gamma(m)\epsilon + \mathcal{O}(\epsilon^2) \tag{4.22}$$

sind von der Ordnung ϵ. Nach deren Entwicklung muss das Resultat mit Reihendarstellungen von bekannten Funktionen identifiziert werden. Im vorliegenden Fall benötigen wir nur den Logarithmus und den Dilogarithmus:

$$\sum_{i=1}^{\infty} \frac{x^i}{i} = -\ln(1-x), \qquad \sum_{i=1}^{\infty} \frac{x^i}{i^2} = \mathrm{Li}_2(x) \,. \tag{4.23}$$

Die Summen in (4.21) haben aber eine viel kompliziertere Struktur und man kann diese nur schrittweise für konkrete Werte von n, m_1 und m_2 umschreiben. Man nimmt zuerst $n = 0$ an und findet, dass die Summe Eins ergibt. Als nächstes nimmt man $n = 1$ an und hat die drei Fälle $m_1 = 1$, $m_2 = 1$ und $n - m_1 - m_2 = 1$. Wegen (4.22) sind diese Terme von der Ordnung ϵ bzw. ϵ^2. Identifikationen mit Logarithmen und Dilogarithmen sind noch möglich, weil nur jeweils eine Untersumme vorhanden ist. Terme mit $n > 1$, die von der Ordnung ϵ^3 sind, haben die vollständigen m_i-Untersummen mit komplizierten Index-Mustern nach (4.19). Diese können praktisch nicht mehr mit einfachen Funktionen identifiziert werden. Als Faustregel kann gelten, dass die Anzahl der ϵ in den Gewichten der R-Funktion die höchste Potenz der Taylor-Entwicklung angibt, für die eine Umformung in einfache Polylogarithmen angegeben werden kann.

Die Entwicklung einer R-Funktion ist aufwändig und müsste für alle benötigten Parameterkombinationen wiederholt werden. Durch die Existenz von Beziehungen zwischen R-Funktionen mit unterschiedlichen Parametern, kann man aber die vielen Fälle auf einige wenige reduzieren. Diese nennt man Basis-R-Funktionen. Eine dieser wichtigen Beziehungen ist

$$-(t+1)(z_i - z_j)\mathrm{R}_t(b;z) = (\beta-1)\left[\mathrm{R}_{t+1}(b-e_i;z) - \mathrm{R}_{t+1}(b-e_j;z)\right] , \tag{4.24}$$

wobei die Kurznotationen $b = (b_1, \ldots, b_k)$, $z = (z_1, \ldots, z_k)$ und $b - e_i = (\ldots, b_i - 1, \ldots)$ verwendet werden. Ist in einer R-Funktion das i-te Gewicht b_i Null, dann kann es zusammen mit dem Argument z_i gestrichen werden:

$$\mathrm{R}_t(b_1, \ldots, b_{k-1}, 0; z_1, \ldots, z_{k-1}, z_k) = \mathrm{R}_t(b_1, \ldots, b_{k-1}; z_1, \ldots, z_{k-1}) \,. \tag{4.25}$$

Diese Eigenschaft führt zusammen mit (4.24) oft zu Vereinfachungen. $\mathrm{R}_t(2-\epsilon, 1, 1; z_1, z_2, z_3)$ zum Beispiel kann in zwei R-Funktionen mit jeweils nur noch zwei statt drei Argumenten umgeschrieben werden.

4.2.5 Weitere Rechenschritte

Die drei wesentlichen analytischen Rechenschritte wurden nun vorgestellt und an Hand der skalaren Drei-Punkt-Funktion veranschaulicht. Für nicht-skalare Funktionen werden die einzelnen Formeln komplizierter, aber die Rechenschritte selbst bleiben gleich. Betrachtet man Funktionen mit mehr als drei äußeren Impulsen oder mit mehr als einer Schleife, dann müssen noch weitere Integrationen ausgeführt werden.

Im Fall der Einschleifen-Vier-Punkt-Funktion ist in Formel (4.17) noch eine zusätzliche Integration auszuführen. Dazu müssen die R-Funktionen wie beschrieben entwickelt werden. Anschließend kann man das Integral auf analytisch bekannte Integrale zurückführen. Bei Zweischleifen-Integralen bleiben mehr Integrale übrig, die dann nur noch numerisch integriert werden können. Die Integranden sind aber dafür gut geeignet und die numerische Integration kann von xloops automatisiert ausgeführt werden.

Für die Berechnung von tensoriellen Funktionen wird noch ein weiterer, bisher unerwähnter Rechenschritt notwendig: die Tensorreduktion. xloops führt diese am Anfang der Berechnung aus und verwendet dafür das Passarino-Veltman-Verfahren [13].

4.3 Neue analytische Ansätze

Die Parallel-/Orthogonalraum-Methode und die Verwendung von Carlsons R-Funktion sind die besonderen Techniken des xloops-Verfahrens. Diese Techniken besitzen noch Potenzial für die Entwicklung neuer analytischer Berechnungsmethoden. Konkrete Ideen sind die Berechnung von Feynman-Integralen in höheren Ordnungen von ϵ, und die methodisch einheitliche Berechnung aller Einschleifen-N-Punktfunktionen.

Wie im Beispiel der Drei-Punkt-Funktion gesehen, kann das Ergebnis als Summe über R-Funktionen angegeben werden. Das Ergebnis ist noch vollkommen allgemein in dem Sinne, dass man noch nicht nach ϵ bis zu einer bestimmten Ordnung entwickelt hat. Die einzige Schwierigkeit, die Drei-Punkt-Funktion in beliebigen höheren Ordungen von ϵ zu berechnen, liegt in der Entwicklung der R-Funktion. Wie gesehen, gibt es dabei praktische Hindernisse. Mit den bisherigen Methoden kommt man meist nicht weiter als bis zur Ordnung ϵ^2. Wenn man ein Verfahren entwickeln könnte, bei dem die Entwicklung der R-Funktion allgemein funktioniert und am besten noch von einem Computer übernommen werden kann, hätte man neue Berechnungsmethoden zur Verfügung, die auch praktische Relevanz besitzen. Die ersten Schritte dahin wurden schon unternommen und sind in den nächsten Abschnitten beschrieben.

Die Formeln für die Einschleifen-N-Punkt-Funktionen, bevor das letzte Integral ausgeführt wird, sind untereinander sehr ähnlich. Die einzigen Unterschiede liegen praktisch in der Anzahl der Argumente für die R-Funktion. Die zugehörigen Gewichte, die für $N > 4$ dazukommen, sind meist trivial, z.B. gleich Eins. Da die R-Funktion entwickelt werden muss, bevor das Integral ausgeführt werden kann, stellt sich die Frage, ob die Menge der resultierenden Entwicklungskoeffizienten, und damit der Integranden nicht endlich ist und einfach tabelliert werden könnte. Dies ist zu erwarten, da man N-Punkt-Funktionen auf Boxgraphen zurückführen kann (siehe [28] oder Kapitel 3). Die Art der zusätzlichen Gewichte ist so, dass man davon ausgehen kann, durch Rekursionsformeln wie beispielsweise

(4.24) und (4.25) am Ende immer die gleichen Funktionen zu finden. Die Berechnung von N-Punkt-Funktionen ist nicht neu. Allerdings ist der hier angedachte Weg zur Berechnung durch die einheitliche Behandlung aller Funktionen ästhetisch ansprechend und es ist nicht falsch, eine weitere, analytische Berechnungsmethode zur Hand zu haben, um Ergebnisse vergleichen zu können. Die praktische Implementierung des beschriebenen Ansatzes ist aber noch nicht sehr weit fortgeschritten, daher kann man über konkrete Vorzüge nur spekulieren. Es ist aber zu bemerken, dass das xloops-Verfahren keine Einschränkungen an die Massen der Teilchen hat und sich somit unter Umständen ganz neue Ergebnisse berechnen lassen. Für die fernere Zukunft kann man auch über die Kombination der beiden Ansätze nachdenken. Im weiteren werden wir nur die Arbeiten zum ersten Ansatz vorstellen, d.h. die automatisierte Taylor-Entwicklung von R-Funktionen.

4.4 Z-Summen und verallgemeinerte Polylogarithmen

Ein Verfahren zur automatisierten Reihenentwicklung von R-Funktionen wurde von uns inzwischen entwickelt und in xloops implementiert. Bevor wir das Verfahren im nächsten Abschnitt beschreiben, soll hier zunächst die notwendige Mathematik eingeführt werden.

Für die verschiedenen, immer wieder bei der Berechnung von Feynman-Integralen sowie der Reihenentwicklung von speziellen Funktionen auftretenden Strukturen und Objekte wurde in [47] eine vereinheitlichende mathematische Beschreibung entwickelt, die sich bei der Anwendung auf die genannten Probleme als äußerst nützlich erweist. Die zentralen mathematischen Objekte, die in [47] eingeführt werden, sind die Z-Summe und die S-Summe. Beides sind geschachtelte Mehrfachsummen, die sich nur durch unterschiedliche Obergrenzen der Teilsummen voneinander unterscheiden. Konkret ist eine Z-Summe definiert als

$$Z(N; m_1, \ldots, m_k; x_1, \ldots, x_k) = \sum_{i_1=1}^{N} \sum_{i_2=1}^{i_1-1} \cdots \sum_{i_k=1}^{i_{k-1}-1} \frac{x_1^{i_1}}{i_1^{m_1}} \frac{x_2^{i_2}}{i_2^{m_2}} \cdots \frac{x_k^{i_k}}{i_k^{m_k}} \tag{4.26}$$

und eine S-Summe als

$$S(N; m_1, \ldots, m_k; x_1, \ldots, x_k) = \sum_{i_1=1}^{N} \sum_{i_2=1}^{i_1} \cdots \sum_{i_k=1}^{i_{k-1}} \frac{x_1^{i_1}}{i_1^{m_1}} \frac{x_2^{i_2}}{i_2^{m_2}} \cdots \frac{x_k^{i_k}}{i_k^{m_k}} \,. \tag{4.27}$$

Beide Summen-Typen lassen sich ohne Schwierigkeiten ineinander umschreiben: eine Z-Summe kann als eine Summe von S-Summen geschrieben werden und umgekehrt. Obwohl daher im Prinzip ein Summen-Typ genügen würde, ist es praktisch, beide Typen zur Verfügung zu haben. Man kann die Summen auch rekursiv definieren, zum Beispiel die Z-Summe:

$$\begin{aligned} Z(N; m_1, \ldots, m_k; x_1, \ldots, x_k) &= \sum_{i=1}^{N} \frac{x_1^i}{i^{m_1}} Z(i-1; m_2, \ldots, m_k; x_2, \ldots, x_k) \\ Z(N;;) &= \begin{cases} 1, & N \geq 0 \\ 0, & N < 0 \end{cases} \,. \end{aligned} \tag{4.28}$$

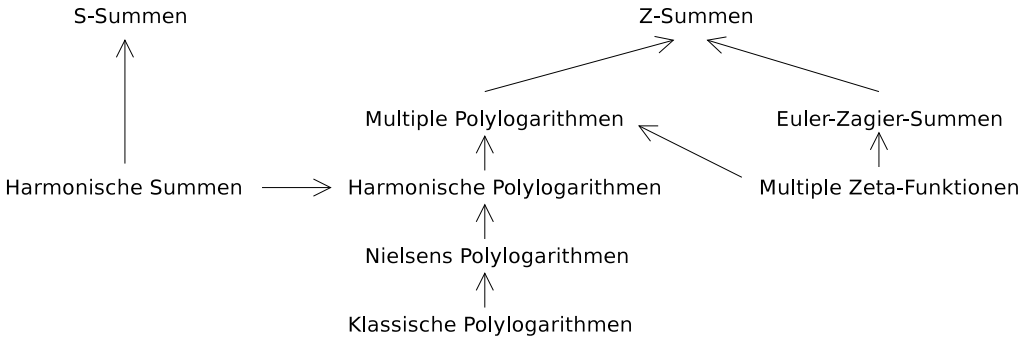


Abbildung 4.5: Hierarchie der verschiedenen Funktionen als Spezialfälle der S- bzw. Z-Summen. Die Pfeile zeigen in Richtung der allgemeineren Funktion.

Mit diesen Summen lassen sich nun eine Reihe von Funktionen zusammenfassend beschreiben. Dazu gehören vor allem die Polylogarithmen sowie die Euler-Zagier-Summen. Die Polylogarithmen bilden eine Hierarchie von den einfachen Klassischen Polylogarithmen über immer allgemeinere Varianten bis hin zum allgemeinsten Fall, den Multiplen Polylogarithmen. Die Hierarchie ist in Abbildung 4.5 gezeigt. Die folgende Übersicht zeigt die konkreten Zusammenhänge über die Parameter. Z-Summen mit der Obergrenze $N = \infty$ beschreiben Polylogarithmen. Der allgemeinste Polylogarithmus ist der Multiple Polylogarithmus von Goncharov[3] [48]:

$$\mathrm{Li}_{m_1,\ldots,m_k}(x_1,\ldots,x_k) = Z(\infty; m_1,\ldots,m_k; x_1,\ldots,x_k)\,. \tag{4.29}$$

Alle weiteren Polylogarithmen ergeben sich als Spezialfälle dieser Funktion. Der Harmonische Polylogarithmus [49] ist zum Beispiel:

$$\mathrm{H}_{m_1,\ldots,m_k}(x) = Z(\infty; m_1,\ldots,m_k; x, 1,\ldots,1)\,. \tag{4.30}$$

Nielsens Polylogarithmus [50] ist

$$\mathrm{S}_{n,p}(x) = Z(\infty; \underbrace{n+1, 1,\ldots,1}_{p}; x, 1,\ldots,1)\,, \tag{4.31}$$

und der Klassische Polylogarithmus schreibt sich als

$$\mathrm{Li}_n(x) = Z(\infty; n; x)\,. \tag{4.32}$$

Die Multiple Zeta-Funktion schließlich ist

$$\zeta(m_1,\ldots,m_k) = Z(\infty; m_1,\ldots,m_k; 1,\ldots,1)\,. \tag{4.33}$$

[3] Goncharov [48] definierte die Funktion ursprünglich mit Argumenten, die in umgekehrter Reihenfolge stehen. Durch die spätere systematische Behandlung aller Polylogarithmen mit Hilfe von Z-Summen ist es aber sinnvoll geworden, diese Konvention zu ändern.

Eine wichtige Funktion, die nicht zu den Polylogarithmen gehört, ist die Euler-Zagier-Summe:

$$\mathrm{Z}_{m_1,\ldots,m_k}(n) = Z(n; m_1, \ldots, m_k; 1, \ldots, 1)\,. \tag{4.34}$$

Von den Funktionen, die sich als S-Summe schreiben lassen, ist die Harmonische Summe

$$\mathrm{S}_{m_1,\ldots,m_k}(n) = S(n; m_1, \ldots, m_k; 1, \ldots, 1) \tag{4.35}$$

die wichtigste.

Die große Nützlichkeit der Z-Summen beruht auf der Tatsache, dass sie sowohl die Polylogarithmen als auch die Euler-Zagier-Summen umfassen. Die Euler-Zagier-Summen treten nämlich als Entwicklungskoeffizienten bei der Taylor-Entwicklung der Eulerschen Γ-Funktion nach einem kleinen Argument $\epsilon \to 0$ auf. Die Entwicklung ist

$$\begin{aligned}\Gamma(n+\epsilon) = \Gamma(1+\epsilon)\Gamma(n)\,\Big[1 &+ \epsilon \mathrm{Z}_1(n-1) \\ &+ \epsilon^2 \mathrm{Z}_{11}(n-1) + \epsilon^3 \mathrm{Z}_{111}(n-1) + \cdots + \epsilon^{n-1} \mathrm{Z}_{11\ldots1}(n-1)\Big]\,.\end{aligned} \tag{4.36}$$

Hier ist n als positive Ganzzahl angenommen. Für negative n existiert ebenfalls eine Formel, die wir aber im weiteren nicht benötigen.

Bis jetzt haben wir zwei wichtige Eigenschaften der Z-Summen kennengelernt. Zum einen enthalten sie die Polylogarithmen, die als Funktionsklasse in den Berechnungen von Feynman-Integralen praktisch immer auftreten. Zum anderen bilden sie die Koeffizienten der Taylor-Entwicklung der Γ-Funktion, und diese Entwicklung gehört bei allen dimensional regularisierten Feynman-Integralen zu den notwendigen Rechenschritten. Es bleibt noch, eine dritte wichtige Eigenschaft zu beschreiben.

4.4.1 Shuffle-Algebra und Quasi-Shuffle-Algebra

Z-Summen mit gleicher oberer Summengrenze kann man miteinander multiplizieren. Die Definition des Produkts lautet

$$\begin{aligned}&Z(n; m_1, \ldots, m_k; x_1, \ldots, x_k)\; Z(n; m'_1, \ldots, m'_l; x'_1, \ldots, x'_l) \\ &\quad= \sum_{i=1}^{n} \frac{x_1^i}{i^{m_1}} Z(i-1; m_2, \ldots, m_k; x_2, \ldots, x_k) Z(i-1; m'_1, \ldots, m'_l; x'_1, \ldots, x'_l) \\ &\quad+ \sum_{i=1}^{n} \frac{x_1'^i}{i^{m'_1}} Z(i-1; m_1, \ldots, m_k; x_1, \ldots, x_k) Z(i-1; m'_2, \ldots, m'_l; x'_2, \ldots, x'_l) \\ &\quad+ \sum_{i=1}^{n} \frac{(x_1 x'_1)^i}{i^{m_1+m'_1}} Z(i-1; m_2, \ldots, m_k; x_2, \ldots, x_k) Z(i-1; m'_2, \ldots, m'_l; x'_2, \ldots, x'_l)\,.\end{aligned} \tag{4.37}$$

Die Formel ist rekursiv. Jeder der drei Terme enthält wieder ein Produkt von Z-Summen, das wieder mit der gleichen Produktformel ausmultipliziert werden muss. Die Z-Summen haben bei jedem Schritt weniger Argumente, so dass der Vorgang irgendwann endet, wenn

eine der Z-Summen keine Argumente mehr hat. Das Ergebnis dieser letzten Multiplikation ist die jeweils andere Z-Summe, die noch Argumente besitzt.

Am Ende erhält man eine Summe von neuen Z-Summen als Ergebnis der Multiplikation. Die wichtige Eigenschaft dabei ist, dass das Ergebnis nur wieder Z-Summen enthält. Das ganze Verfahren zur Taylor-Entwicklung von speziellen Funktionen mit Hilfe von Z-Summen beruht auf dieser Eigenschaft.

Die Menge der Z-Summen ist unter dem Produkt (4.37) geschlossen und man kann diese Eigenschaft mathematisch so ausdrücken, dass die Z-Summen eine Algebra bilden. Diese Algebra hat in der Mathematik den Namen Quasi-Shuffle-Algebra. Für Z-Summen mit der Obergrenze $N = \infty$, also Polylogarithmen, kann man auch noch ein weiteres Produkt definieren. Die zugehörige Algebra heißt Shuffle-Algebra. Der Name stammt von der „Durchmischung“ der Parameter in den resultierenden Summen. In (4.37) wird dagegen nur quasi ge-„shuffelt“, weil der dritte Term die Parameter kombiniert, statt sie zu „durchmischen“.

Um im nächsten Kapitel die Produkte kompakt notieren zu können, führen wir die Algebren nochmal unabhängig vom Begriff der Z-Summen ein. Die Shuffle- und Quasi-Shuffle-Algebra-Produkte kann man an Hand von Operationen auf Listen abstrakt definieren [51]. Eine Liste

$$a = (a^{(1)}, a^{(2)}, \ldots, a^{(m)})$$

enthält beliebige Objekte $a^{(i)}$. Schreibt man Listen nebeneinander, so kann man sie zusammenfassen:

$$ab = (a^{(1)}, \ldots, a^{(k)})(b^{(1)}, \ldots, b^{(m)}) = (a^{(1)}, \ldots, a^{(k)}, b^{(1)}, \ldots, b^{(m)}) \,.$$

Genauso kann man auch eine Liste zerlegen, z.B. indem man das erste Element herausnimmt:

$$a' = (a^{(1)}, a^{(2)}, \ldots, a^{(k)}) = a^{(1)}(a^{(2)}, \ldots, a^{(k)}) = a^{(1)}a \,.$$

Die leere Liste ist das Eins-Element $\mathbb{1} = ()$ der Algebren.

Das Shuffle-Produkt $\sqcup\!\sqcup$ ist dann definiert durch

$$\mathbb{1} \sqcup\!\sqcup a = a \sqcup\!\sqcup \mathbb{1} = a \tag{4.38}$$

$$a'b' = a^{(1)}a \sqcup\!\sqcup b^{(1)}b = a^{(1)}(a \sqcup\!\sqcup b') + b^{(1)}(a' \sqcup\!\sqcup b) \,. \tag{4.39}$$

Die Definition ist rekursiv. In der zweiten Gleichung wird das Produkt jeweils auf zwei Produkte mit verkürzten Listen zurückgeführt, bis schließlich eine Liste leer wird und die erste Gleichung zur Anwendung kommt. Die Einzelelemente werden am Schluss zusammengefügt und man bekommt eine Summe über Listen.

Das Quasi-Shuffle-Produkt $*$ ist definiert durch

$$\mathbb{1} * a = a * \mathbb{1} = a \tag{4.40}$$

$$a'b' = a^{(1)}a * b^{(1)}b = a^{(1)}(a * b') + b^{(1)}(a' * b) + [a^{(1)}, b^{(1)}](a * b) \,. \tag{4.41}$$

$[,]$ ist eine neue Operation, die kommutativ und assoziativ sein muss und den Grad der Elemente addiert, sonst aber beliebig definiert sein kann. Im konkreten Fall der Z-Summen

stehen die Elemente der Listen für je einen Summanden $\frac{x^i}{i^m}$, und man definiert die Operation $[,]$ als

$$\left[\frac{x_a^i}{i^{m_a}}, \frac{x_b^i}{i^{m_b}}\right] = \frac{(x_a x_b)^i}{i^{m_a+m_b}} .$$

Der Grad eines Elements ist gleich dem Exponenten m_i definiert. Damit entspricht der dritte Term in (4.41) dem dritten Term in (4.37).

4.4.2 Die Algorithmen der C^{++}-Bibliothek nestedsums

Wie man mit Z-Summen spezielle Funktionen nach kleinen Parametern $\epsilon \to 0$ entwickelt, zeigen wir skizzenhaft am Beispiel der Gauß'schen Hypergeometrischen Funktion

$${}_2F_1(a,b;c;x) = \sum_{n=0}^{\infty} \frac{(a,n)(b,n)}{(c,n)} \frac{x^n}{n!} . \tag{4.42}$$

Eine besser geeignete Schreibweise ist

$${}_2F_1(a,b;c;x) = 1 + \sum_{n=1}^{\infty} \frac{\Gamma(a+n)\Gamma(b+n)\Gamma(c)}{\Gamma(a)\Gamma(b)\Gamma(c+n)\Gamma(n)} \frac{x^n}{n} , \tag{4.43}$$

bei der die Appell-Symbole als Γ-Funktionen ausgeschrieben sind und der Summand $n = 0$ aus der Summe gezogen wurde. Die Fakultät wurde zu

$$n! = \Gamma(n+1) = n\Gamma(n)$$

umgeschrieben.

Wir betrachten nun Hypergeometrische Funktionen, deren Argumente a, b und c von der Form $s + t\epsilon,\ s \in \mathbb{Z},\ t \in \mathbb{R}$ sind, wie es in praktischen Rechnungen meistens der Fall ist. Dann kann man nach (4.36), oder nach entsprechenden Formeln für negative s bzw. $t \neq 1$ die wir hier nicht besprochen haben, alle Γ-Funktionen, die ϵ enthalten, entwickeln. Die Ordnung der Entwicklung muss man zu Beginn festlegen. Die Faktoren $\Gamma(n)$ der Taylor-Reihen kürzen sich alle weg. Es bleiben Z-Summen als Entwicklungskoeffizienten, die man als nächstes ausmultipliziert. Weil die Z-Summen eine Quasi-Shuffle-Algebra bilden, ergibt sich wie im letzten Abschnitt dargestellt eine Summe von einzelnen Z-Summen. Nach Potenzen von ϵ geordnet erhält man aus dem zweiten Term in (4.43) Ausdrücke der Form

$$\begin{aligned} &\sum_i \alpha_i \epsilon^i \sum_{n=1}^{\infty} \big[Z(n-1;\ldots) + \ldots + Z(n-1;\ldots)\big] \frac{x^n}{n} \\ &= \sum_i \alpha_i \epsilon^i \sum_{\{\text{alle } Z\text{-Summen}\}} \sum_{n=1}^{\infty} \frac{x^n}{n} Z(n-1;\ldots) . \end{aligned} \tag{4.44}$$

Die α_i sind reelle Vorfaktoren aus der Reihenentwicklung. Die Summe über n mit einer Untersumme $Z(n-1;\ldots)$ kann man nach Definition (4.28) als eine Z-Summe schreiben. Das Endergebnis ist also tatsächlich eine Taylor-Reihe in ϵ mit Summen von Z-Summen als Entwicklungskoeffizienten.

Das Verfahren wurde in [47] allgemein formuliert. Es lassen sich Funktionen entwickeln, deren Reihendarstellung nicht nur, wie in unserem Beispiel, Terme der Form

$$\frac{x^i}{i^m}$$

und Brüche von Γ-Funktionen

$$\frac{\Gamma(i+s_1+t_1\epsilon)}{\Gamma(i+s_2+t_2\epsilon)}$$

enthalten, sondern es sind auch Binomialkoeffizienten

$$\binom{N}{i}$$

und Untersummen mit gegenläufigen Obergrenzen

$$Z(N-i;\ldots)$$

als Objekte zugelassen. Die praktisch relevanten Kombinationen wurden in vier Algorithmen zusammengefasst, die mit den Buchstaben A bis D bezeichnet sind. In [40] wurde die C++-Bibliothek nestedsums vorgestellt, die dem Programmierer den Zugriff auf diese vier Algorithmen in Form von C++-Funktionen ermöglicht. nestedsums benutzt GiNaC, um die nötigen algebraischen Operationen auszuführen. Entsprechend verwendet das Interface der nestedsums-Bibliothek die Datentypen von GiNaC.

Wir zeigen im folgenden die Summen, die durch die Algorithmen A und B entwickelt werden können, um einen Eindruck von den Möglichkeiten von nestedsums zu vermitteln, und auch weil die beiden Algorithmen im folgenden Abschnitt verwendet werden. Details zum Interface von nestedsums finden sich in [40]. Durch den Algorithmus A kann eine Summe der Form

$$\begin{aligned}\sum_{i=1}^{N} \frac{x^i}{(i+o_1)^m} \frac{\Gamma(i+a_1+b_1\epsilon)}{\Gamma(i+c_1+d_1\epsilon)} \cdots \\ \cdots \frac{\Gamma(i+a_k+b_k\epsilon)}{\Gamma(i+c_k+d_k\epsilon)} Z(i-1+o_2; m_1,\ldots,m_l; x_1,\ldots,x_l)\end{aligned} \tag{4.45}$$

nach ϵ entwickelt werden. Der Algorithmus B behandelt Summen der Form

$$\begin{aligned}\sum_{i=1}^{N-1} &\frac{x^i}{(i+o_1)^m} \frac{\Gamma(i+a_1+b_1\epsilon)}{\Gamma(i+c_1+d_1\epsilon)} \cdots \\ &\cdots \frac{\Gamma(i+a_k+b_k\epsilon)}{\Gamma(i+c_k+d_k\epsilon)} Z(i-1+o_2; m_1,\ldots,m_l; x_1,\ldots,x_l) \\ &\times \frac{y^{N-i}}{(N-i+o_3)^{m'}} \frac{\Gamma(N-i+a'_1+b'_1\epsilon)}{\Gamma(N-i+c'_1+d'_1\epsilon)} \cdots \\ &\cdots \frac{\Gamma(N-i+a'_{k'}+b'_{k'}\epsilon)}{\Gamma(N-i+c'_{k'}+d'_{k'}\epsilon)} Z(N-i-1+o_4; m'_1,\ldots,m'_{l'}; x'_1,\ldots,x'_{l'}) \,,\end{aligned} \tag{4.46}$$

die im Gegensatz zu A auch Untersummen mit gegenläufigem Index $N-i$ enthalten. Bei A kann $N=\infty$ sein, bei B muss N endlich sein. $o_{1,2,3,4}$ sind konkrete Ganzzahlen, mit denen die Formeln noch allgemeiner gefasst werden können. $o_{1,3}$ müssen positiv sein. Die a_i und c_i müssen ebenfalls bestimmte Ganzzahlen sein, die b_i und d_i sind beliebig.

4.5 Automatisierte Reihenentwicklung von R

Die in Abschnitt 4.3 gewünschte automatisierte Taylor-Entwicklung von R-Funktionen in beliebige Ordnungen von ϵ ist mit nestedsums nun möglich. Das Verfahren wird im folgenden beschrieben.

4.5.1 Der Algorithmus

Der erste Schritt ist die Umschreibung der Reihendarstellung der R-Funktion (4.18) in eine weniger kompakte aber nützlichere Form, indem man die Appell-Symbole ausschreibt. In Definition (4.18) sollte über alle m_i summiert werden, mit der Bedingung $\sum m_i = n$. Eine Möglichkeit, die Summationsindizes m_i gemäß dieser Bedingung zu wählen ist:

$$\begin{aligned}
&\mathrm{R}_t(b_1,...,b_k;z_1,...,z_k) = \\
&\quad \frac{1}{\Gamma(b_1)\cdots\Gamma(b_k)} \sum_{n=0}^{\infty} \frac{\Gamma(n-t)\Gamma(\beta)}{\Gamma(-t)\Gamma(n+\beta)} \sum_{m_1=0}^{n} \frac{\Gamma(m_1+b_1)}{\Gamma(m_1)} \frac{(1-z_1)^{m_1}}{m_1} \\
&\qquad \times \sum_{m_2=0}^{n-m_1} \frac{\Gamma(m_2+b_2)}{\Gamma(m_2)} \frac{(1-z_2)^{m_2}}{m_2} \\
&\qquad \times \cdots \\
&\qquad \cdots \times \sum_{m_{k-2}=0}^{n-m_1-\cdots-m_{k-3}} \frac{\Gamma(m_{k-2}+b_{k-2})}{\Gamma(m_{k-2})} \frac{(1-z_{k-2})^{m_{k-2}}}{m_{k-2}} \\
&\qquad \times \sum_{m_{k-1}=0}^{n-m_1-\cdots-m_{k-2}} \frac{\Gamma(m_{k-1}+b_{k-1})}{\Gamma(m_{k-1})} \frac{(1-z_{k-1})^{m_{k-1}}}{m_{k-1}} \\
&\qquad\quad \times \frac{\Gamma(n-m_1-\cdots m_{k-1}+b_k)}{\Gamma(n-m_1-\cdots-m_{k-1})} \frac{(1-z_k)^{n-m_1-\cdots-m_{k-1}}}{n-m_1-\cdots-m_{k-1}}\,.
\end{aligned} \tag{4.47}$$

Die Bedingung $\sum m_i = n$ ist hier dadurch gewährleistet, dass man die letzte Summe durch die Subsitution $m_k = n - m_1 - \cdots m_{k-1}$ implizit ausführt. Die Untersummen sind mit ihrer Indexstruktur besonders ausführlich dargestellt, weil man daran die Anwendbarkeit der nestedsums-Algorithmen erkennen kann. Die beiden letzten Zeilen von (4.47) haben die passende Form für den Algorithmus B. Die in (4.46) vorhandenen Z sind dabei gleich Eins. x entspricht $1 - z_{k-1}$ und y ist $1 - z_k$. Der Laufindex der äußeren Summe ist m_{k-1} und die letzte Zeile in (4.47) hat den gegenläufigen Index $n - m_1 - \cdots - m_{k-2} - m_{k-1}$. Man erkennt allerdings, dass die Unter- und Obergrenze der Summe nicht mit Formel (4.46) übereinstimmt. Auf dieses Problem kommen wir gleich zurück. Wendet man den Algorithmus B nun an, dann kann man die beiden letzten Zeilen von (4.47) durch eine Summe von Z-Summen ersetzen, die alle die Obergrenze $n-m_1-\cdots-m_{k-1}$ haben. Danach kann man den Algorithmus B iterativ anwenden, bis alle m_i-Summen umgewandelt sind. Die resultierenden Z-Summen können zusammen mit der Summe mit Index n in der ersten Zeile von (4.47) durch den Algorithmus A verarbeitet werden. Die R-Funktion ist dadurch komplett entwickelt.

Wie schon angemerkt, passen die Ober- und Untergrenzen der Summen nicht exakt zu den Algorithmen A und B. Bevor man also das oben beschriebene Verfahren durchführen kann, müssen noch die Summengrenzen behandelt werden. Der Fall $n = 0$ lässt sich ganz einfach aus der Summe herausnehmen und ergibt wie in (4.43) Eins[4].

Hat ein Summenindex den Wert der Obergrenze, dann müssen alle folgenden Indizes gleich Null sein. Die betroffenen Summen ergeben Eins. Es verschwinden also alle Terme, die rechts von der betrachteten Summe stehen. Jeder der $k-1$ möglichen Indizes begründet einen Spezialfall. Zusammen mit dem Fall, dass keiner der m_i-Indizes gleich der Summen-Obergrenze ist, hat man nun k verschiedene Summen, die jede noch bezüglich der Untergrenzen behandelt werden muss.

Ist ein m_i gleich Null, dann ist die zugehörige Summe gleich Eins. Das Resultat ist eine verkürzte Summe. Man erhält wieder eine Reihe von Spezialfällen, von denen nun aber jeder das passende „Indexbild“ hat und man die Algorithmen A und B, wie oben beschrieben, anwenden kann.

4.5.2 Implementierung und Anwendung

In xloops waren schon Taylor-Reihen von einigen speziellen R-Funktionen implementiert, die für die Integralberechnungen gebraucht werden. Die Entwicklungen wurden per Hand berechnet und umfassen daher nur wenige, niedrige Potenzen in ϵ. Die Formeln wurden direkt in C^{++}-Code übersetzt und werden von xloops auf Anfrage einfach zurückgeliefert.

Das oben beschriebene Verfahren wurde nun zusätzlich in xloops implementiert und berechnet alle Entwicklungen jeweils automatisiert neu. Nur wenn die nachgefragte Ordnung in ϵ zu hoch, oder die nachgefragte R-Funktion gar nicht fest vorprogrammiert ist, wird diese automatisierte Entwicklung gestartet, sonst wird auf die alte Implementierung zurückgegriffen.

Der Test der Implementierung ist schwierig. Ergebnisse der alten und neuen Implementierung wurden miteinander verglichen. Allerdings ist die Anzahl der Vergleiche auf die wenigen R-Funktionen beschränkt, die von Hand berechnet werden konnten. Beispiele sind

$$\mathrm{R}_{-\epsilon}(1,1;x,y) \tag{4.48}$$
$$\mathrm{R}_{-2\epsilon}(\epsilon,\epsilon,1;x,y,z) \tag{4.49}$$
$$\mathrm{R}_{-1-\epsilon}(1+\epsilon,1;x,y)\,. \tag{4.50}$$

(4.48) konnte in allen Ordnungen ϵ, (4.49) bis zur Ordnung ϵ^2 und (4.50) bis zur Ordnung ϵ^1 verglichen werden. Mit Reduktionsformeln wie (4.24) kann man über die Basis-R-Funktionen hinaus weitere Fälle mit der alten Implementierung berechnen, die sich dann vergleichen lassen.

Ein prinzipielles Problem bei den Vergleichen ist die Eigenschaft des neuen Verfahrens, als Entwicklungskoeffizienten möglichst allgemeine Polylogarithmen zu erzeugen. Hätte man

[4] Der Faktor $\frac{1}{\Gamma(b_1)\cdots\Gamma(b_k)}$ vor den Summen ist natürlich am Ende noch an das Gesamtergebnis zu multiplizieren.

bei einer konkreten Rechnung als Koeffizienten das Ergebnis

$$\ln(1-xy)\ln\Big(1-\frac{1-xy}{1-y}\Big)+\mathrm{Li}_2\Big(\frac{1-xy}{1-y}\Big)-\mathrm{Li}_2\Big(\frac{1}{1-y}\Big)$$

gefunden, dann würde mit nestedsums das Ergebnis

$$\mathrm{Li}_{1,1}(x,y)$$

lauten. Beide Ergebnisse sind identisch, aber statt der einfachen Logarithmen und Dilogarithmen hat man nun einen Multiplen Polylogarithmus. Sofern man nicht die Identität zwischen beiden Ergebnissen algebraisch bewiesen hat, bleibt nur der numerische Vergleich.

Möchte man die neuen Möglichkeiten zur Taylor-Entwicklung für die Integralberechnung in xloops nutzen, um wie in Abschnitt 4.3 beschrieben, die Drei-Punkt-Funktion in höheren Ordungen von ϵ auszuwerten, dann stößt man zur Zeit noch auf ungelöste Probleme. Unter den relevanten R-Funktionen, die in höheren Ordungen zu entwickeln wären, befinden sich die Funktionen[5]

$$\mathrm{R}_{1-2\epsilon}\big(\epsilon-\tfrac{1}{2},\epsilon-\tfrac{1}{2},1;x,y,z\big) \tag{4.51}$$

und

$$\mathrm{R}_{1+j-2\epsilon}\big(\epsilon-\tfrac{p_2}{2},\epsilon-\tfrac{p_2}{2};x,y\big)\,. \tag{4.52}$$

j und p_2 sind Ganzzahlen. Die Funktionen besitzen halbzahlige Parameter. Wie aus der Beschreibung des Entwicklungs-Algorithmus ersichtlich ist, können solche Argumente nicht behandelt werden. Eine Verallgemeinerung des Algorithmus auf halbzahlige Argumente existiert bereits [52], eine funktionsfähige Implementierung ist aber noch nicht programmiert.

Möchte man das Verfahren auch für andere Integral-Berechnungen einsetzen, dann hat man das Problem, höhere Polylogarithmen numerisch auswerten zu müssen. Software zur numerischen Auswertung dieser Polylogarithmen exisitierte aber nicht. Dieser Umstand hat die Entwicklung und Implementierung von numerischen Verfahren angestoßen, deren Beschreibung der Inhalt des nächsten Kapitels ist.

[5] In unserer Beispielrechnung kommen diese R-Funktionen nicht vor. Wir haben allerdings auch nur die skalare Drei-Punkt-Funktion berechnet. Der Zähler der tensoriellen Funktionen führt zu diesen weiteren R-Funktionen

5 Numerische Auswertung von Multiplen Polylogarithmen

In Kapitel 4 wurde eine analytische Methode zur Berechnung von Schleifenintegralen vorgestellt. Dabei traten in den Lösungen nicht mehr nur einfache Polylogarithmen auf, sondern eine ganze Reihe von verallgemeinerten Polylogarithmen, je nach Komplexität des Integrals oder der Anzahl der Massenskalen. Für die praktische Verwendung der berechneten Ergebnisse ist daher die numerische Auswertung dieser Funktionen sehr wichtig. Multiple Polylogarithmen konnten jedoch bis jetzt nicht allgemein numerisch ausgewertet werden. In diesem Kapitel soll nun das neu entwickelte Verfahren zur Berechnung von beliebigen Multiplen Polylogarithmen vorgestellt werden.

Auch wenn die numerische Auswertung von mathematischen Funktionen keine physikalische Problemstellung darstellt, so ist deren Bedeutung für die Physik nicht zu unterschätzen. In vielen Fällen werden physikalische Probleme erst lösbar und berechenbar, wenn man passende mathematische Objekte finden oder neu definieren kann, die ein (Zwischen-)Ergebnis in eine analytisch behandelbare Struktur bringen. Integrale zum Beispiel, von denen man annahm, dass sie noch gelöst oder weiter reduziert werden müssten, können sich als die Integraldarstellung von neuen speziellen Funktionen entpuppen und einfach als elementare, mathematische Objekte einer Lösung belassen werden. Die Bereitschaft der Forscher, solche mathematischen Objekte als Bestandteile von Lösungen zu akzeptieren, hängt aber oft von der Möglichkeit einer schnellen und stabilen numerischen Auswertung ab. Daraus folgt, dass die Entwicklung von Methoden zur numerischen Auswertung von Funktionen für Fortschritte bei physikalischen Berechnungen sehr wichtig ist und deshalb auch für Physiker durchaus eine fachbezogene Aufgabe darstellt.

Die Problemstellung der numerischen Auswertung von Funktionen ist wahrscheinlich vielen Physikern nicht geläufig. Daher soll zunächst am Beispiel des Dilogarithmus das grundlegende Handwerk der numerischen Auswertung dargestellt werden. Danach erst wird die Auswertung der Multiplen Polylogarithmen angegangen. Abschließend werden Details zur Implementierung des Verfahrens in die C++-Bibliothek GiNaC beschrieben.

5.1 Beispiel Dilogarithmus

Funktionen können auf verschiedene Weise definiert sein, und dementsprechend unterschiedliche Methoden zur numerischen Auswertung gibt es. Ziel ist es immer, eine Vorschrift zur Verfügung zu haben, die für gegebene Argumente angibt, wie man den Wert der Funktion durch die Ausführung von elementaren mathematischen Operationen wie Addition oder Multiplikation berechnen kann. Es kann verschiedene Vorschriften geben, die für verschiedene Argumente jeweils besser oder schlechter funktionieren. Kriterien für besser

oder schlechter sind die Geschwindigkeit der Auswertung oder die erreichbare Präzision des Ergebnisses. Insgesamt ist dieses mathematische Gebiet viel zu groß und komplex, um es hier in aller Allgemeinheit angemessen darstellen zu können. Eine Beschränkung auf das Notwendigste ist daher geboten und die Funktionsklasse, die uns hier eigentlich interessiert, sind die Polylogarithmen.

Die verschiedenen Polylogarithmen besitzen nun gemeinsame charakteristische Eigenschaften, welche ganz entscheidend die Methoden für die numerische Auswertung prägen. Der einfache Logarithmus „vererbt“ seine analytischen Eigenschaften auf die aus ihm erzeugten Polylogarithmen, so dass diese gleichfalls Pole und Verzweigungsschnitte (engl.: *branch cuts*) in der komplexen Ebene der Funktionsargumente besitzen. Neben einer Darstellung als Integral, die diese analytischen Eigenschaften widerspiegelt, besitzen alle Polylogarithmen außerdem eine Darstellung als Reihe. Integraldarstellung und Reihendarstellung haben jeweils eine charakteristische Struktur, die schon im vorigen Kapitel im Abschnitt über Z-Summen dargestellt wurde. Diese wesentlichen Eigenschaften reduzieren die riesige Menge an Techniken und Vorgehensweisen bei der numerischen Auswertung von Funktionen auf ein paar wenige, die man am Beispiel des Dilogarithmus als dem einfachsten Polylogarithmus am leichtesten einführen und erklären kann.

Die Integraldarstellung ist eine prinzipielle Möglichkeit zur numerischen Auswertung, die sich aber bei näherer Betrachtung als wenig geeignet herausstellt. Das liegt hauptsächlich an der Gegenwart von Verzweigungsschnitten und Polen. Außerdem ist die numerische Integration nicht immer das schnellste Verfahren, und die Kontrolle der Fehler, d.h. der Präzision, ist meist nicht optimal. Wenn es keine Alternative zur Benutzung der Integraldarstellung gäbe, würde man durchaus diesen Weg beschreiten. Allerdings zeigt sich, dass die Reihendarstellung alle notwendigen Möglichkeiten bietet und in der Praxis tatsächlich auch den besten Ansatz darstellt.

Der Dilogarithmus ist definiert[1] als

$$\mathrm{Li}_2(x) = -\int_0^x \mathrm{d}t\, \frac{\ln(1-t)}{t} = -\int_0^1 \mathrm{d}t\, \frac{\ln(1-xt)}{t}\,. \tag{5.1}$$

Das linke Integral ist die üblichere Schreibweise, obwohl das rechte Integral bei komplexen Argumenten Vorteile hat.

Wie schon geschrieben, ist diese Darstellung jedoch für die numerische Auswertung nicht gut geeignet. Man hat auf dem Integrationsweg eine Polstelle bei $x = 1$ und auch die Kontrolle der Präzision ist durch die Form des Integranden nicht trivial. Die Integraldarstellung ist allerdings für die Unterschuchung der analytischen Eigenschaften entscheidend. Die Lage des Verzweigungsschnitts des Dilogarithmus hängt von der Lage des Schnitts des Logarithmus ab. Gewöhnlich wird der Schnitt des Logarithmus so definiert, dass er auf

[1] Es gab früher viele Notationen und zum Teil leicht unterschiedliche Definitionen für diese Funktion. Praktisch erst mit Lewins Buch [53] hat sich die hier verwendete Schreibweise langsam durchgesetzt. Im Hinblick auf die weiter oben gemachte Bemerkung über die Wichtigkeit der numerischen Auswertbarkeit von Funktionen ist es interessant zu bemerken, dass dies wohl nur deswegen geschehen ist, weil sich Lewin als erster um eine systematische numerische Auswertung von Polylogarithmen bemüht hat, was sein Buch zu einem Standardwerk werden ließ.

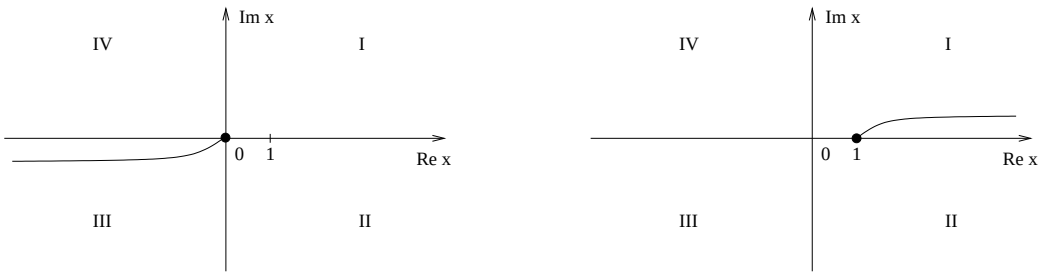


Abbildung 5.1: Schematische Darstellung der Verzweigungsschnitte des Logarithmus (links) und des Dilogarithmus (rechts). Die Singularitäten sind als dicke schwarze Punkte eingezeichnet. Die eingezeichneten Schnitte liegen etwas abseits der reellen Achse, um die Konvention für die Stetigkeit am Schnitt zu symbolisieren. Man hat sich diesen Abstand als infinitesimal zu denken. Liegt die reelle Achse wie beim Logarithmus über dem Schnitt, dann ist der Schnitt stetig zum darüber liegenden Gebiet. Die Argumente auf der negativen reellen Achse werden dann so behandelt, als ob sie einen positiven infinitesimalen Imaginärteil besäßen.

der negativen reellen Achse liegt mit der Singularität bei Null. Daraus ergibt sich die Lage des Schnitts beim Dilogarithmus als der Teil $(1, +\infty)$ der positiven reellen Achse.

Die Funktionswerte für Argumente auf dem Verzweigungsschnitt sind noch nicht eindeutig festgelegt. Der Schnitt selbst kann entweder zum darüber oder zum darunter liegenden Gebiet gehören, d.h. die Funktionswerte sind dann jeweils unstetig, wenn man den Schnitt entweder von unter- oder von oberhalb kommend betritt. Die Konvention für Argumente auf dem Schnitt wird für den Logarithmus nach der Formel

$$\ln(-x \mp i0) = \ln(|x|) \mp i\pi\theta(x) \tag{5.2}$$

üblicherweise so festgelegt, dass reelle Argumente x einen positiven infinitesimalen Imaginärteil zugeordnet bekommen. Die Infinitesimale wird symbolisch mit $0i$ notiert. Die umgekehrte Konvention müßte dem Argument einen negativen Imaginärteil zuordnen. $\theta(x)$ ist die bekannte Heavisidesche Stufenfunktion. Für den Dilogarithmus mit reellen Argumenten x gilt

$$\mathrm{Li}_2(x \pm i0) = \mathrm{Re}\big\{\mathrm{Li}_2(x)\big\} \pm i\pi \ln(x)\theta(x-1)\,. \tag{5.3}$$

Die Konvention ist, dass man den reellen Argumenten des Dilogarithmus einen negativen infinitesimalen Imaginärteil zuordnet. In Abbildung 5.1 ist die Konvention schematisch gezeigt. Daneben ist die sich daraus ergebende Konvention für den Dilogarithmus gezeigt. Der Logarithmus auf dem Schnitt ist zum Quadranten IV und der Dilogarithmus ist auf dem Schnitt zum Quadranten II kontinuierlich. Beim Weg vom Quadranten III bzw. I auf den Schnitt springt der Imaginärteil der jeweiligen Funktionen um 2π bzw. $2\pi\log(x)$.

Nachdem die analytischen Eigenschaften besprochen sind, gehen wir nun für die numerische Auswertung auf die Reihendarstellung ein. Die Reihendarstellung des Dilogarithmus lautet

$$\mathrm{Li}_2(x) = \sum_{n=1}^{\infty} \frac{x^n}{n^2}\,, \quad |x| \le 1 \tag{5.4}$$

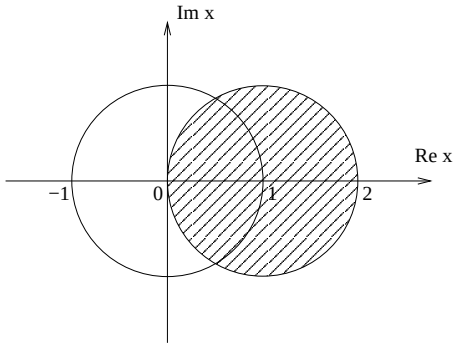


Abbildung 5.2: Wirkung der $1-x$-Transformation. Der schraffierte Bereich wird auf den Bereich des Einheitskreises abgebildet.

Das praktische Verfahren zur Berechnung eines Funktionswerts ist einfach. Man summiert für ein gegebenes x sukzessive die Terme $\frac{x^n}{n^2}$ auf und prüft jeweils, ob sich die Summe innerhalb der gewünschten Präzision nicht mehr ändert. Falls nicht, beendet man die Summierung und hat eine Näherung für den Wert des Dilogarithmus am Punkt x damit berechnet. So einfach und schnell die Methode ist, hat sie jedoch einen offensichtlichen Makel: die Reihe konvergiert nur für Werte $|x| \leq 1$. Allgemein anwenden kann man die Reihendarstellung also nur, wenn man einen Weg findet, Argumente von außerhalb des Konvergenzbereichs nach innerhalb zu transformieren.

Betrachtet man das Integral (5.1) für die Funktion $\mathrm{Li}_2(\frac{1}{x})$, so kann man durch ein paar elementare Umformungen das Integral in einfachere Integrale zerlegen. Man erhält so eine Beziehung zwischen $\mathrm{Li}_2(x)$ und $\mathrm{Li}_2(\frac{1}{x})$:

$$\mathrm{Li}_2(x) = -\mathrm{Li}_2\left(\frac{1}{x}\right) - \frac{\pi^2}{6} - \frac{1}{2}\left(\ln(-x)\right)^2 \ . \tag{5.5}$$

Diese Beziehung ist der Schlüssel zur numerischen Auswertung mittels Reihendarstellung, da sie die Transformation beliebiger Argumente x innerhalb des Konvergenzbereichs erlaubt. Hat man ein Argument $|x| > 1$ gegeben, dann rechnet man nicht $\mathrm{Li}_2(x)$ direkt aus, sondern die Terme auf der rechten Seite von (5.5). Der Dilogarithmus der dabei zu berechnen ist, hat das Argument $\left|\frac{1}{x}\right| < 1$ und stellt kein Problem mehr dar.

Die Konvergenz der Reihe wird aber schlecht, wenn man die Argumente nahe am Rand des Konvergenzbereiches wählt. Im besonderen sind alle Werte auf dem Rand des Konvergenzbereiches, hier $|x| = 1$, problematisch. Manche Argumente auf dem Rand lassen sich als Spezialfälle direkt berechnen, zum Beispiel hat man

$$\begin{aligned} \mathrm{Li}_2(1) &= \zeta(2) = \frac{\pi^2}{6} \\ \mathrm{Li}_2(-1) &= -\frac{\pi^2}{12} \\ \mathrm{Li}_2(\pm i) &= -\frac{\pi^2}{48} \pm iC \end{aligned} \tag{5.6}$$

mit Catalans Konstante $C \approx 0.916$. Für die anderen Fälle muss man nach weiteren Transformationsformeln suchen. Betrachtet man $\mathrm{Li}_2(1-x)$ so kann man wie in (5.5) eine Trans-

formationsformeln herleiten:

$$\mathrm{Li}_2(x) = -\mathrm{Li}_2(1-x) + \frac{\pi^2}{6} - \ln(x)\ln(1-x)\,. \tag{5.7}$$

Hier wird x durch $1-x$ ersetzt, so dass alle Werte nahe bei $x = 1$ in die Nähe der Null transformiert werden. Der Effekt dieser Transformation ist in der Abbildung 5.2 veranschaulicht. Alle Argumente im schraffierten Gebiet werden in den Konvergenzbereich transformiert, darunter auch weitere Teile des Konvergenzrandes. Die Transformation $x \to 1-x$ alleine genügt noch nicht. Man kann die beiden Transformationen (5.5) und (5.7) aber selbstverständlich hintereinander ausführen und erhält so weitere Transformationen. Führt man als Beispiel nach der $\frac{1}{x}$ die $1-x$ Transformation aus, so ist die resultierende Transformation durch $x \to 1-\frac{1}{x} = \frac{x-1}{x}$ beschrieben. Insgesamt kann man zusammen mit der Identität $x \to x$ die folgenden sechs Transformationen bilden:

$$x \quad \to \quad x, \quad 1-x, \quad \frac{1}{x}, \quad \frac{1}{1-x}, \quad \frac{x-1}{x}, \quad \frac{x}{x-1}\,. \tag{5.8}$$

Betrachtet man die Menge aller konformen Transformationen der Art

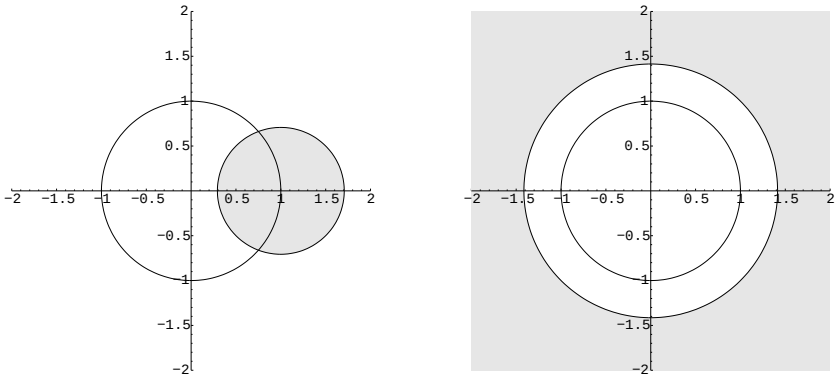
$$x \to \frac{\alpha x + \beta}{\gamma x + \delta} \qquad \text{mit} \qquad \alpha\delta - \beta\gamma \neq 0 \tag{5.9}$$

(Möbius-Transformation), dann bilden die sechs Transformationen (5.8) eine abgeschlossene Untergruppe davon. Es genügt also im Prinzip, nur die beiden Transformationen (5.5) und (5.7) für einen Polylogarithmus zu finden, um alle sechs Transformationen ausführen zu können. In Abbildung 5.3 sind die Effekte der verschiedenen Transformationen gezeigt. Die grau unterlegten Gebiete werden jeweils in den Konvergenzbereich transformiert. Allerdings wurde dabei der Konvergenzbereich $|x| < R$ nicht mit $R = 1$, sondern praxisgerechter mit $R < 1$ definiert. Es wurde $R = \frac{1}{\sqrt{2}} \approx 0.707$ gesetzt. Der Wert wurde nach einer realistischen Optimierung der Transformationsgrenzen gewählt. Die Optimierung wird gleich noch diskutiert werden. Zunächst aber kann man feststellen, dass für reelle Argumente die beschriebenen Transformationen ausreichen, um nur solche Dilogarithmen ausrechnen zu müssen, deren gegebenenfalls transformierte Argumente für eine schnelle Reihenkonvergenz nahe genug an der Null liegen. Untersucht man die Transformationen genauer, dann findet man zwei Punkte

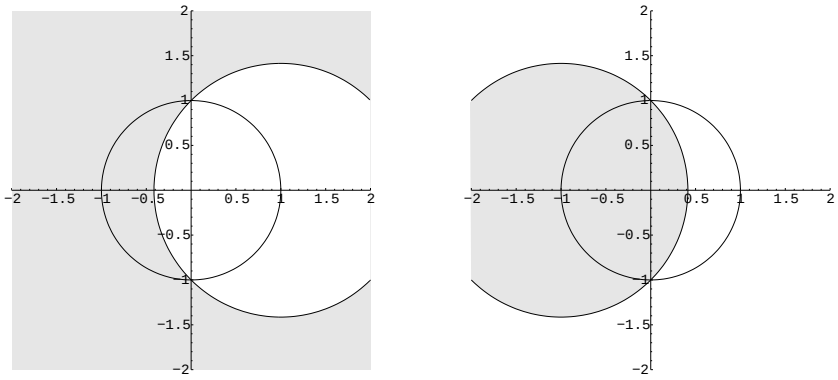
$$x_{1,2} = \frac{1}{2} \pm \sqrt{\frac{3}{4}}i \tag{5.10}$$

bei denen alle sechs Transformationen versagen. Die Punkte werden immer wieder auf sich selbst abgebildet. Zur Behandlung von komplexen Argumenten genügen diese sechs Transformationen also noch nicht. Eine Implementierung die auch allgemein für komplexe Zahlen funktionieren soll, benötigt immer noch mindestens eine weitere Transformation neben (5.5) und (5.7). In Abbildung 5.3 ist die Transformation $x \to \frac{x}{1+x}$ gezeigt, welche die Problempunkte beseitigt. Andere Transformationen, die möglich wären, sind $x \to -x$ oder $x \to \frac{1-x}{1+x}$.

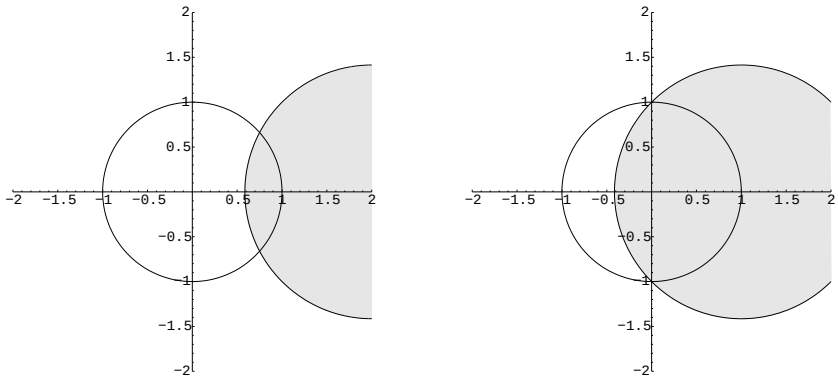
Der notwendige Formalismus zur numerischen Auswertung ist nun beschrieben. Es bleibt in der Praxis aber noch ein weiterer Punkt zu beachten. Die Reihe konvergiert unterschiedlich schnell, je nachdem wie weit das Argument von der Null entfernt ist. Für $|x| \approx 1$ ist



(a) $1-x$ (b) $\frac{1}{x}$



(c) $\frac{1}{1-x}$ (d) $\frac{x}{x-1}$



(e) $\frac{x-1}{x}$ (f) $\frac{x}{1+x}$

Abbildung 5.3: Verschiedene Transformationen auf der komplexen Ebene. Das jeweils grau unterlegte Gebiet wird durch die angebene Formel in das Gebiet $|x| < \frac{1}{\sqrt{2}}$ transformiert. Zusätzlich eingezeichnet ist der Konvergenzrand $|x| = 1$ der Dilogarithmus-Reihe.

die Konvergenz am schlechtesten. Dann kann unter Umständen die Aufsummierung von mehreren hundert oder tausend Termen notwendig sein. Es ist also sinnvoll, die Aufsummierung zu optimieren. Ein Hilfsmittel haben wir schon kennengelernt. Die Transformationsformeln bieten nicht nur die notwendige Möglichkeit zur Sicherung der prinzipiellen Konvergenz, sondern man kann sie auch benutzen, um Argumente möglichst nahe an die Null zu bringen. In der Praxis sind die Auswahl der angewandten Formeln und die Gebiete, in denen sie verwandt werden, eine Frage der Optimierung verschiedener Faktoren. Bei der Transformation treten neue mathematische Objekte auf, die ebenfalls ausgewertet werden müssen. Die einzelnen Terme müssen dann am Ende kombiniert werden. Unter Umständen ist der Aufwand dafür größer als der Gewinn der Verschiebung des Arguments von z.B. $x = 0.6$ zu $x = 0.4$ durch die $x \to 1 - x$ Transformation. Die Transformationsgrenzen müssen daher durch empirische Messung der Auswertungsgeschwindigkeiten bestimmt werden und sind deshalb sehr von der jeweiligen Implementierung abhängig. In Abschnitt 5.4 über die Implementierung werden Details besprochen.

Auch die Auswertung der Reihe selbst lässt sich noch beschleunigen. Es gibt dazu eine ganze Menge von Verfahren in der Literatur [34], z.B. Padé-Approximation, Chebychev-Ökonomisierung, etc. Auf diese und ähnliche Verfahren wird hier nicht weiter eingegangen, da sie nur bei einer bekannten, festliegenden numerischen Präzision angewendet werden können. Die C++-Bibliothek GiNaC, für die wir im weiteren die Implementierung beschreiben, arbeitet aber mit Numerik beliebiger Präzision, und so sind nur die Algorithmen für uns interessant, welche eine beliebige numerische Präzision erlauben.

Ein Verfahren, das bei Dilogarithmen zur Anwendung kommen kann, ist die Umschreibung der Reihe mittels Bernoulli-Zahlen. Die Substitution $t = 1 - e^{-u}$ im Integral des Dilogarithmus (5.1) ergibt

$$\mathrm{Li}_2(x) = -\int_0^x \mathrm{d}t\, \frac{\ln(1-t)}{t} = \int_0^{-\ln(1-x)} \mathrm{d}u\, \frac{u}{e^u - 1}\,. \tag{5.11}$$

Die Bernoulli-Zahlen B_n sind nun durch die Reihe

$$\frac{x}{e^x - 1} = \sum_{n=0}^{\infty} B_n \frac{x^n}{n!}, \qquad |x| < 2\pi \tag{5.12}$$

definiert, so dass man nach Einsetzen von (5.12) in (5.11) den Dilogarithmus umschreiben kann als

$$\mathrm{Li}_2(x) = \sum_{i=0}^{\infty} B_i \frac{z^{i+1}}{(i+1)!}\,, \tag{5.13}$$

mit $z = -\ln(1-x)$. Die Voraussetzung $|-\ln(1-x)| < 2\pi$ von Gleichung (5.12) ist nach den Transformationen immer erfüllt. Diese Reihe konvergiert nun in vielen Fällen schneller als die ursprüngliche Reihe. Man kann wie in [54] das asymptotische Verhalten der beiden Reihen vergleichen und finden, dass die Bernoulli-Reihe schneller konvergiert. Allerdings ist für die Praxis das asymptotische Verhalten nicht so entscheidend, da die Transformationen die Argumente so nah an die Null bringen, dass auch im ungünstigsten Fall meist weniger als hundert Reihenglieder aufsummiert werden müssen. Man muss die Reihen

n	0	1	2	4	6	8	10	12	14	16	18
B_n	1	$-\frac{1}{2}$	$\frac{1}{6}$	$-\frac{1}{30}$	$\frac{1}{42}$	$-\frac{1}{30}$	$\frac{5}{66}$	$-\frac{691}{2730}$	$\frac{7}{6}$	$-\frac{3617}{510}$	$\frac{43867}{798}$

Abbildung 5.4: Auflistung der Bernoulli-Zahlen B_n bis $n \leq 18$. Die Bernoulli-Zahlen für ungerade $n > 2$ sind alle Null und wurden weggelassen.

konkret vergleichen und das heißt man muss letztendlich an Hand der Implementierung Vergleichsmessungen anstellen.

Ein paar allgemeine Überlegungen lassen sich jedoch trotzdem anstellen. In Tabelle 5.1 sind die ersten Bernoulli-Zahlen dargestellt. Als Vorfaktor wirken sich die Bernoulli-Zahlen günstig aus, da sie zunächst kleiner als Eins sind. Außerdem sind ab $n > 2$ alle ungeraden Bernoulli-Zahlen Null. Dieser Vorteil schwindet mit wachsendem n, da die Bernoulli-Zahlen schnell viel größer als Eins werden. Je mehr Reihenglieder summiert werden müssen, weil entweder die Präzision sehr hoch oder das Argument weit von Null entfernt ist, desto ungünstiger wirkt sich diese Tatsache aus. In der Bernoulli-Reihe steht statt einer Potenz eine Fakultät im Nenner. Die Fakultät wächst viel schneller als die Potenz, was die Konvergenz beschleunigt. Allerdings ist für kleine Werte des Summationsindex noch kein großer Unterschied vorhanden. Je mehr Reihenglieder summiert werden müssen, desto deutlicher wird der Effekt. Der letzte Punkt, den man vergleichen kann ist die Größe von x und $z = -\ln(1-x)$. Betrachtet man $x \in [0, 1]$, so ist z immer größer als x. Der Abstand wächst je näher x an Eins kommt. Dies verlangsamt die Bernoulli-Reihe im Vergleich. Die drei Einflussfaktoren zusammengenommen ergeben folgendes Bild: Der Einsatz der Bernoulli-Reihe ganz nah an der Null lohnt sich nicht, weil zu wenige Reihenglieder aufsummiert werden müssen. Ansonsten ist die Bernoulli-Reihe der ursprünglichen Reihe vorzuziehen, wobei die Größe der Argumente durch die Transformationsgleichungen beschränkt ist.

Bevor nun die Multiplen Polylogarithmen besprochen werden, soll noch einmal zusammengefasst werden, was die notwendigen Elemente bei der numerischen Auswertung des Dilogarithmus waren. Zunächst benötigte man die Reihendarstellung. Als nächstes musste man Transformationsgleichungen finden, um alle Argumente in den Konvergenzbereich bringen zu können. Das dritte Element waren weitere Transformationsgleichungen oder andere Hilfsmittel, um die Reihenkonvergenz zu beschleunigen. Diese drei Elemente sind bei der Auswertung *aller* Polylogarithmen notwendig und sie werden daher im folgenden immer wiederkehren.

5.2 Der Multiple Polylogarithmus

5.2.1 Definition

Der Multiple Polylogarithmus $\mathrm{Li}_{m_1,\ldots,m_k}(x_1, \ldots, x_k)$ wurde in [48] eingeführt und stellt eine Verallgemeinerung aller anderen bekannten Polylogarithmen sowie der Multiplen Zeta-

Funktionen dar. Er ist definiert[2] durch eine mehrfach geschachtelte Reihe

$$\mathrm{Li}_{m_1,\ldots,m_k}(x_1,\ldots,x_k) = \sum_{i_1>i_2>\ldots>i_k>0} \frac{x_1^{i_1}}{{i_1}^{m_1}} \ldots \frac{x_k^{i_k}}{{i_k}^{m_k}} \,. \tag{5.14}$$

Die Argumente unterscheidet man in Gewichte m_i und Skalen x_i. Die Anzahl k der Summen nennt man die Tiefe des Polylogarithmus. Die Summe aller einzelnen Gewichte m_i nennt man das Gewicht des Polylogarithmus.

Die Integraldarstellung des Multiplen Polylogarithmus lässt sich am besten notieren, wenn man zuvor noch eine Kurznotation für iterierte Integrale einführt. Mehrfach iterierte Integrale sollen dabei mittels $\circ$-Operator wie folgt abgekürzt notiert werden:

$$\int_0^\Lambda \frac{\mathrm{d}t}{t-a_1} \circ \ldots \circ \frac{\mathrm{d}t}{t-a_n} = \int_0^\Lambda \frac{\mathrm{d}t_1}{t_1-a_1} \times \ldots \times \int_0^{t_{n-2}} \frac{\mathrm{d}t_{n-1}}{t_{n-1}-a_{n-1}} \int_0^{t_{n-1}} \frac{\mathrm{d}t_n}{t_n-a_n} \tag{5.15}$$

und

$$\int_0^\Lambda \left(\frac{\mathrm{d}t}{t}\circ\right)^m \frac{\mathrm{d}t}{t-a} = \int_0^\Lambda \underbrace{\frac{\mathrm{d}t}{t} \circ \ldots \frac{\mathrm{d}t}{t}}_{m-\text{fach}} \circ \frac{\mathrm{d}t}{t-a} \,. \tag{5.16}$$

Damit lautet die Integraldarstellung des Multiplen Polylogarithmus

$$\begin{aligned}\mathrm{Li}_{m_1,\ldots,m_k}(x_1,\ldots,x_k) = & \int_0^{x_1\cdots x_k} \left(\frac{\mathrm{d}t}{t}\circ\right)^{m_1-1} \frac{\mathrm{d}t}{x_1\cdots x_{k-1}-t} \circ \cdots \\ & \circ \left(\frac{\mathrm{d}t}{t}\circ\right)^{m_{k-2}-1} \frac{\mathrm{d}t}{x_1x_2-t} \circ \left(\frac{\mathrm{d}t}{t}\circ\right)^{m_{k-1}-1} \frac{\mathrm{d}t}{x_1-t} \circ \left(\frac{\mathrm{d}t}{t}\circ\right)^{m_k-1} \frac{\mathrm{d}t}{1-t} \,.\end{aligned} \tag{5.17}$$

An Hand der Integraldarstellung lassen sich die analytischen Eigenschaften untersuchen und man kann ggf. Transformationsgleichungen für die numerische Auswertung herleiten. Im Hinblick auf die numerische Auswertung ist die obige Definition nicht sehr geeignet, weil die Skalen x_i in den Nennern der Brüche als Produkte auftreten. Die Pole und Schnitte der Funktion hängen also nicht von den x_i selbst ab, sondern von einem Produkt dieser x_i. Es ist daher schwierig, sich über die Pol-Struktur klar zu werden. Außerdem ist eine konsistente Definition von Funktionswerten auf den Verzweigungsschnitten durch die Wahl eines infinitesimalen Imaginärteils für die Argumente nicht möglich. Wenn man für die Integranden eine Konvention für den Imaginärteil $i0$ wie zum Beispiel

$$\int \frac{\mathrm{d}t}{x_1\cdots x_k - t \pm i0}$$

festlegt, dann läßt sich der Imaginärteil $i0$ nicht eindeutig den einzelnen Argumente x_i zuordnen.

[2] Goncharov [48] definierte die Funktion ursprünglich mit Argumenten, die in umgekehrter Reihenfolge stehen. Durch die spätere systematische Behandlung aller Polylogarithmen mit Hilfe von Z-Summen ist es aber sinnvoll geworden, diese Konvention zu ändern.

Es ist daher sinnvoll für die Integraldarstellung zunächst einen „neuen“ Multiplen Polylogarithmus G einzuführen, bei dem die Produkte der x_i selber als Argumente verwendet werden. Führt man die Transformation $y \to x_1 \cdots x_k$ aus, so erhält man die folgende Definition:

$$\mathrm{G}(z_1, ..., z_k; y) = \int_0^y \frac{\mathrm{d}t}{t - z_1} \circ \frac{\mathrm{d}t}{t - z_2} \cdots \circ \frac{\mathrm{d}t}{t - z_k} \,. \tag{5.18}$$

Für den Fall, dass einige Argumente z_i Null sind, verwenden wir auch folgende kompakte Notation:

$$\mathrm{G}_{m_1,...,m_k}(z_1, ..., z_k; y) = \mathrm{G}(\underbrace{0, ..., 0, z_1}_{m_1}, ..., z_{k-1}, \underbrace{0, ..., 0, z_k}_{m_k}; y) \,, \tag{5.19}$$

bei der man wieder von Gewichten m_i und Skalen z_i sprechen kann. y ist ein freier, reeller Parameter.

G ist nicht wirklich ein neuer Multipler Polylogarithmus. Li und G beschreiben exakt die gleiche Funktion. Beide Darstellungen lassen sich einfach ineinander umwandeln:

$$\mathrm{Li}_{m_1,...,m_k}(x_1, ..., x_k) = (-1)^k \mathrm{G}_{m_1,...,m_k}\left(\frac{1}{x_1}, \frac{1}{x_1 x_2}, ..., \frac{1}{x_1 ... x_k}; 1\right) . \tag{5.20}$$

Der Vorfaktor $(-1)^k$ resultiert aus der Umkehrung der Reihenfolge von t und den x_i im Nenner. Wir haben also jetzt zwei Darstellungen für den Multiplen Polylogarithmus, die für verschiedene Zwecke jeweils besser geeignet sind: Li für alles was mit der Reihendarstellung zu tun hat, und G für alles was die Integraldarstellung betrifft. Shuffle-Identitäten lassen sich zum Beispiel eleganter mit Hilfe von G beschreiben, Quasi-Shuffle-Identitäten dagegen sind einfacher mit Hilfe von Li notiert.

5.2.2 Analytische Eigenschaften

Der Parameter y der Funktion G ist redundant, wie man sich leicht klarmachen kann. Durch die Skalierungseigenschaft der iterierten Integrale gilt:

$$\mathrm{G}(\sigma z_1, ..., \sigma z_k; \sigma y) = \mathrm{G}(z_1, ..., z_k; y) \,. \tag{5.21}$$

Man kann also immer durch die Wahl $\sigma = \frac{1}{y}$ zu $\mathrm{G}(z_1, \ldots, z_k; 1)$ transformieren. Es ist trotzdem sinnvoll, diesen Parameter beizubehalten. Bei den nötigen Integraltransformationen wird die Integral-Obergrenze meist mitverändert. Da sie nicht auf $y = 1$ festgelegt ist, sondern y unbestimmt bleibt, können alle entstehenden Integrale wieder mit dem Multiplen Polylogarithmus G identifiziert werden. Die rekursiven Identitäten im folgenden lassen sich so besser schreiben.

Die Pole und Schnitte des Multiplen Polylogarithmus sind an der Integraldarstellung ersichtlich. Die Funktion G hat Verzweigungsschnitte, wenn ein z_i eine positive reelle Zahl ist und die entsprechende Integrationsvariable t größer als z_i wird. Unter Umständen besitzt ein Multipler Polylogarithmus k Singularitäten auf der positiven reellen Achse, an denen die Verzweigungsschnitte in Richtung $+\infty$ beginnen. Man kann die Konventionen für die

Schnitte durch einen infinitesimalen Imaginärteil im Nenner der Integranden festlegen. Dazu ist die Schreibweise

$$z_+ = z + i0, \qquad z_- = z - i0 \tag{5.22}$$

nützlich. Die konkrete Festlegung der Konvention ist bei der Implementierung der Funktion notwendig. Für die Berechnung der Transformationsgleichungen wird die Unterscheidung der Imaginärteile aber auch schon notwendig.

5.2.3 Trailing zeros

Bei der Definition von G haben wir nicht ausgeschlossen, dass Argumente z_i am Ende der Argumentenliste Null sind, also z.B. $\mathrm{G}(z_1, \ldots, z_i, 0, ..., 0; y)$. Solche Null-Argumente nennt man *trailing zeros*. G-Funktionen mit *trailing zeros* lassen sich nicht direkt in Li-Funktionen umschreiben, sie stellen eine Verallgemeinerung gegenüber den Multiplen Polylogarithmen Li dar. Problematisch sind die *trailing zeros*, weil man mit ihnen mathematisch nicht definierte Ausdrücke aufschreiben kann. Das folgende, $\mathrm{G}(0; y)$ entsprechende Integral kann formal durch die Stammfunktion gelöst werden

$$\int_0^y \frac{\mathrm{d}t}{t} = \ln(y) - \ln(0)\,, \tag{5.23}$$

ergibt aber den undefinierten Term $\ln(0)$. G-Funktionen mit mehreren *trailing zeros* führen zu komplizierteren Polynomen, die ebenfalls $\ln(0)$ enthalten.

Trotzdem erlaubt man *trailing zeros*, weil die direkte Anwendung der Transformationsgleichungen, die sich für die Multiplen Polylogarithmen finden lassen, G-Funktionen mit *trailing zeros* erzeugen. Wie wir noch sehen werden, sind diese Transformationsgleichungen alle rekursiv definiert. Zur endgültigen Transformation einer Funktion müssen die Gleichungen wiederholt auf schon teiltransformierte Funktionen angewandt werden. Die divergenten Terme, d.h. die G-Funktionen mit *trailing zeros*, entstehen dabei in charakteristischer Weise als Zwischenlösungen. Am Ende der Transformationen jedoch heben sich diese alle wieder gegenseitig weg.

Der mathematisch saubere Weg mit diesen Funktionen umzugehen wäre, alle Untergrenzen der Integrale gleich einer Variablen Λ zu setzen, die man am Ende aller Rechenschritte und Umformungen gegen Null gehen lässt. Vor dem Grenzübergang hätten sich alle divergenten Terme schon weggehoben und das Ergebnis wäre deshalb definiert. In der Praxis spart man sich diese Art der Regularisierung, indem man einfach folgende Definition einführt:

$$\mathrm{G}(\underbrace{0, ..., 0}_{k}; y) = \frac{1}{k!} (\ln y)^k \ . \tag{5.24}$$

Letztlich ist diese genauso konsistent, aber eleganter anzuwenden. Die Definition von G muss dafür aber noch durch folgende rekursive Definition ergänzt bzw. ersetzt werden:

$$\mathrm{G}(z_1, ..., z_k; y) = \int_0^y \frac{\mathrm{d}t}{t - z_1} G(z_2, ..., z_k; t)\,. \tag{5.25}$$

Diese rekursive Definition ist identisch zur ursprünglichen Definition (5.18), erlaubt es aber (5.24) zu verwenden.

Damit sich am Ende aller Transformationen unterschiedliche G-Funktionen mit *trailing zeros* explizit gegenseitig wegheben, ist es sinnvoll die G-Funktionen auf eine einheitliche Form zu bringen, indem man die Null-Elemente separiert. Zu diesem Zweck wendet man die Shuffle-Algebra-Eigenschaften an. Betrachtet man die folgende Multiplikation

$$\mathrm{G}(0;y)\mathrm{G}(z_1,...,z_j,\underbrace{0,...,0}_{k-j-1};y) = (k-j)\mathrm{G}(z_1,...,z_j,\underbrace{0,...,0}_{k-j};y) + \sum_{(s_1,...,s_j)=(z_1,...,z_{j-1})\,\sqcup\!\sqcup\,(0)} \mathrm{G}(s_1,...,s_j,z_j,\underbrace{0,...,0}_{k-j-1};y)\,, \quad (5.26)$$

so kann man nach dem ersten Term auf der rechten Seite auflösen und erhält die Gleichung

$$\mathrm{G}(z_1,...,z_j,\underbrace{0,...,0}_{k-j};y) = \frac{1}{k-j}\left[\mathrm{G}(0;y)\mathrm{G}(z_1,...,z_j,\underbrace{0,...,0}_{k-j-1};y) - \sum_{(s_1,...,s_j)=(z_1,...,z_{j-1})\sqcup\!\sqcup(0)} \mathrm{G}(s_1,...,s_j,z_j,\underbrace{0,...,0}_{k-j-1};y)\right]. \quad (5.27)$$

$\sqcup\!\sqcup$ ist das Shuffle-Produkt wie in Abschnitt 4.4.1 definiert. Damit lassen sich rekursiv alle *trailing zeros* separieren.

5.3 Numerische Auswertung von G

5.3.1 Konvergenz der Reihe

Der Multiple Polylogarithmus Li ist konvergent, wenn

$$|x_1x_2...x_j| \leq 1 \;\text{ für alle }\; j \in \{1,...,k\} \quad \text{und} \quad m_1, x_1 \neq 1 \quad (5.28)$$

gilt, was aus der Reihendarstellung (5.14) folgt. Daraus lässt sich für die Parameter von G ableiten, dass

$$|y| \leq |z_j| \;\text{ für alle }\; j \quad (5.29)$$

zu gelten hat. Das bedeutet, dass $|y|$ das kleinste Element aus der Menge $\{|z_1|,\cdots,|z_k|,|y|\}$ sein muss. Das Ziel der gesuchten Transformationsgleichungen ist es also ein G, welches die Bedingung (5.29) nicht erfüllt, durch einfachere Funktionen oder durch G-Funktionen, die diese Bedingung erfüllen, auszudrücken.

5.3.2 Transformationsformeln

Die Transformation, um die Reihendarstellung des Multiplen Polylogarithmus konvergent zu machen, lässt sich nicht durch eine einzelne Formel beschreiben. Man muss für verschiedene Spezialfälle jeweils passende Formeln anwenden. Diese Formeln bauen zum Teil aufeinander auf und wir führen sie deshalb schrittweise, beginnend mit dem einfachsten Fall, ein.

Zunächst gehen wir von Multiplen Polylogarithmen der Tiefe Eins aus. Die Transformationsformel dafür ist schon bekannt, da diese Multiplen Polylogarithmen den Klassischen Polylogarithmen entsprechen. Allerdings schreiben wir die Formeln allgemeiner und rekursiv, um sie als Bausteine in den weiteren Transformationsformeln benutzen zu können. Sei im folgenden s immer das kleinste Element unter den Argumenten, welches am Ende der Transformation an die Stelle der Integralobergrenze y treten soll. Die Imaginärteile sind wichtig und wie in (5.22) mit Indizes $\pm$ unterschieden. Ist das Gewicht $m = 1$, dann hat man

$$\mathrm{G}_1(s_\pm; y) = \mathrm{G}_1(y_\mp; s) - \mathrm{G}_1(0; s) + \ln(-y_\mp)\,. \tag{5.30}$$

Für ein Gewicht $m \geq 2$ kann man die Formel

$$\mathrm{G}_m(s_\pm; y) = -\zeta(m) + \int_0^y \frac{\mathrm{d}t}{t}\,\mathrm{G}_{m-1}(t_\pm; y) - \int_0^s \frac{\mathrm{d}t}{t}\,\mathrm{G}_{m-1}(t_\pm; y) \tag{5.31}$$

herleiten. Diese Formel muss man nun rekursiv anwenden. Die G_{m-1} auf der rechten Seite müssen wieder mittels (5.31) und schliesslich (5.30) ersetzt werden. Ein kurzes Beispiel für $m = 3$ soll die rekursive Anwendung verdeutlichen: es wird zunächst zweimal (5.31) angewendet

$$\begin{aligned}
\mathrm{G}_3(s_\pm; y) &= -\zeta(3) + \left\{ \int_0^y \frac{\mathrm{d}t_1}{t_1} - \int_0^s \frac{\mathrm{d}t_1}{t_1} \right\} \mathrm{G}_2(t_{1,\pm}; y) \\
&= -\zeta(3) + \left\{ \int_0^y \frac{\mathrm{d}t_1}{t_1} - \int_0^s \frac{\mathrm{d}t_1}{t_1} \right\} \left(-\zeta(2) + \left\{ \int_0^y \frac{\mathrm{d}t_2}{t_2} - \int_0^{t_1} \frac{\mathrm{d}t_2}{t_2} \right\} \mathrm{G}_1(t_{2,\pm}; y) \right),
\end{aligned}$$

dann wird einmal (5.30) auf das verbliebene $\mathrm{G}_1(t_{2,\pm}; y)$ angewendet

$$\begin{aligned}
&= -\zeta(3) - \zeta(2)\Big[\mathrm{G}_1(0; y) - \mathrm{G}_1(0; s)\Big] \\
&\quad + \left\{ \int_0^y \frac{\mathrm{d}t_1}{t_1} \int_0^y \frac{\mathrm{d}t_2}{t_2} - \int_0^s \frac{\mathrm{d}t_1}{t_1} \int_0^y \frac{\mathrm{d}t_2}{t_2} - \int_0^y \frac{\mathrm{d}t_1}{t_1} \int_0^{t_1} \frac{\mathrm{d}t_2}{t_2} + \int_0^s \frac{\mathrm{d}t_1}{t_1} \int_0^{t_1} \frac{\mathrm{d}t_2}{t_2} \right\} \\
&\qquad \times \Big[\mathrm{G}_1(y_\mp; t_2) - \mathrm{G}_1(0; t_2) + \ln(-y_\mp)\Big]\,,
\end{aligned}$$

anschließend wird ausmultipliziert und die iterierten Integrale in G-Funktionen umgeschrieben

$$\begin{aligned}
&= -\zeta(3) - \zeta(2)\Big[\mathrm{G}_1(0;y) - \mathrm{G}_1(0;s)\Big] \\
&\quad + \mathrm{G}(0;y)\mathrm{G}(0,y_\mp;y) - \mathrm{G}(0;s)\mathrm{G}(0,y_\mp;y) - \mathrm{G}(0,0,y_\mp;y) + \mathrm{G}(0,0,y_\mp;s) \\
&\quad - \mathrm{G}(0;y)\mathrm{G}(0,0;y) + \mathrm{G}(0;s)\mathrm{G}(0,0;y) + \mathrm{G}(0,0,0;y) - \mathrm{G}(0,0,0;s) \\
&\quad + \ln(-y_\mp)\Big[\mathrm{G}(0;y)\mathrm{G}(0;y) - \mathrm{G}(0;s)\mathrm{G}(0;y) - \mathrm{G}(0,0;y) + \mathrm{G}(0,0;s)\Big] ,
\end{aligned}$$

und als letztes wird zusammengefasst

$$\begin{aligned}
&= \mathrm{G}_3(y_\mp;s) + 2\,\zeta(2)\Big[\ln(s) - \ln(y)\Big] - \frac{1}{3}\ln^3(y) - \frac{1}{6}\ln^3(s) + \frac{1}{2}\ln(s)\ln^2(y) \\
&\quad + \ln(-y_\mp)\Big[\frac{1}{2}\ln^2(y) - \ln(s)\ln(y) + \frac{1}{2}\ln^2(s)\Big] ,
\end{aligned}$$

wobei $\mathrm{G}_m(y_\mp;y)$ wegen der Skalierungseigenschaft einer Zeta-Funktion $-\zeta(m)$ entspricht. An diesem Beispiel kann und sollte man sich die Anwendung von Rekursionsformeln klarmachen, da alle folgenden Formeln analog benutzt werden müssen.

Der nächste Schritt ist nun die Behandlung von G mit einer größeren Tiefe. Die Transformation dafür muss zwischen drei Fällen unterscheiden: das kleinste Element ist entweder das erste Argument, oder das letzte Argument oder es steht in der Mitte der Argumentenliste. Berechnen wir zunächst den Fall, dass sich das Element irgendwo in der Mitte befindet. Dazu schreibt man G wie folgt um:

$$\begin{aligned}
\mathrm{G}(a_1,...,a_{i-1},s_r,a_{i+1},...,a_w;y) &= \mathrm{G}(a_1,...,a_{i-1},0,a_{i+1},...,a_w;y) \\
&\quad + \int_0^{s_r} \mathrm{d}s_{r+1}\,\frac{\partial}{\partial s_{r+1}}\,\mathrm{G}(a_1,...,a_{i-1},s_{r+1},a_{i+1},...,a_w;y) .
\end{aligned} \tag{5.32}$$

Dies ist nur eine einfache Identität für die Stammfunktion. Man hat nun zwei Terme, von denen der erste bezüglich der Tiefe reduziert ist und sofort weiter rekursiv behandelt werden kann. Im Fall des zweiten Terms ziehen wir die partielle Ableitung in das Integral hinein, direkt vor den s_{r+1} enthaltenden Bruch. Wir verwenden dann die Identität

$$\frac{\partial}{\partial s}\frac{1}{t-s} = -\frac{\partial}{\partial t}\frac{1}{t-s} . \tag{5.33}$$

Damit bekommt man einen Ausdruck der Form

$$\begin{aligned}
&\cdots \int_0^{t_{i-2}} \frac{\mathrm{d}t_{i-1}}{t_{i-1}-a_{i-1}} \int_0^{t_{i-1}} \frac{\partial}{\partial s}\,\frac{\mathrm{d}t_i}{t_i-s} \int_0^{t_i} \frac{\mathrm{d}t_{i+1}}{t_{i+1}-a_{i+1}} \cdots \\
= \quad &\cdots \int_0^{t_{i-2}} \frac{\mathrm{d}t_{i-1}}{t_{i-1}-a_{i-1}} \int_0^{t_{i-1}} \left(-\frac{\partial}{\partial t_i}\,\frac{\mathrm{d}t_i}{t_i-s}\right) \int_0^{t_i} \frac{\mathrm{d}t_{i+1}}{t_{i+1}-a_{i+1}} \cdots \\
= \quad &\cdots (-1) \int_0^{t_{i-2}} \frac{\mathrm{d}t_{i-1}}{t_{i-1}-a_{i-1}} \int_0^{t_{i-1}} \frac{\partial}{\partial t_i} \left(\frac{\mathrm{d}t_i}{t_i-s} \int_0^{t_i} \frac{\mathrm{d}t_{i+1}}{t_{i+1}-a_{i+1}} \cdots \right) \\
&\qquad + \quad \cdots \int_0^{t_{i-2}} \frac{\mathrm{d}t_{i-1}}{t_{i-1}-a_{i-1}} \int_0^{t_{i-1}} \frac{\mathrm{d}t_i}{t_i-s} \left(\frac{\partial}{\partial t_i} \int_0^{t_i} \frac{\mathrm{d}t_{i+1}}{t_{i+1}-a_{i+1}} \cdots \right) \\
= \quad &\cdots (-1) \int_0^{t_{i-2}} \frac{\mathrm{d}t_{i-1}}{t_{i-1}-a_{i-1}} \frac{1}{t_{i-1}-s} \int_0^{t_i} \frac{\mathrm{d}t_{i+1}}{t_{i+1}-a_{i+1}} \cdots \\
&\qquad + \quad \cdots \int_0^{t_{i-2}} \frac{\mathrm{d}t_{i-1}}{t_{i-1}-a_{i-1}} \int_0^{t_{i-1}} \frac{\mathrm{d}t_i}{t_i-s} \frac{1}{t_i-a_{i+1}} \cdots .
\end{aligned} \tag{5.34}$$

Im zweiten Schritt wurde die partielle Ableitung $(\partial f)g = \partial(fg) - f(\partial g)$ nach der Produktregel ersetzt, anschließend wurden Identitäten für die Stammfunktion benutzt. Als nächstes führt man Partialbruchzerlegungen nach

$$\frac{1}{t-\alpha}\frac{1}{t-s} = \frac{1}{s-\alpha}\left(\frac{1}{t-s} - \frac{1}{t-\alpha}\right) \tag{5.35}$$

durch und erhält dann vier Terme, die als rekursive Transformationsformel wie folgt notiert werden können:

$$\begin{aligned}
&\int_0^{s_r} \mathrm{d}s_{r+1} \frac{\partial}{\partial s_{r+1}} \mathrm{G}\left(a_1, ..., a_{i-1}, s_{r+1}, a_{i+1}, ..., a_w; y\right) \\
&\qquad = -\int_0^{s_r} \frac{\mathrm{d}s_{r+1}}{s_{r+1}-a_{i-1}} \mathrm{G}\left(a_1, ..., a_{i-2}, s_{r+1}, a_{i+1}, ..., a_w; y\right) \\
&\qquad\quad + \int_0^{s_r} \frac{\mathrm{d}s_{r+1}}{s_{r+1}-a_{i-1}} \mathrm{G}\left(a_1, ..., a_{i-2}, a_{i-1}, a_{i+1}, ..., a_w; y\right) \\
&\qquad\quad + \int_0^{s_r} \frac{\mathrm{d}s_{r+1}}{s_{r+1}-a_{i+1}} \mathrm{G}\left(a_1, ..., a_{i-1}, s_{r+1}, a_{i+2}, ..., a_w; y\right) \\
&\qquad\quad - \int_0^{s_r} \frac{\mathrm{d}s_{r+1}}{s_{r+1}-a_{i+1}} \mathrm{G}\left(a_1, ..., a_{i-1}, a_{i+1}, a_{i+2}, ..., a_w; y\right) .
\end{aligned} \tag{5.36}$$

Jeder der Terme auf der rechten Seite enthält wieder ein G, das gegebenfalls weiterbehandelt werden muss. Man schaut dazu jeweils nach dem neuen kleinsten Argument unter den verbliebenden a_i und transformiert entsprechend.

Die Gleichungen (5.32) und (5.36) zusammengenommen sind unsere Transformationformel für Multiple Polylogarithmen mit Tiefe ≥ 2, wenn das kleinste Element in der Mitte der Argumentenliste steht. Falls nun das kleinste Element an erster Stelle steht, kann man die Schritte (5.34) noch genauso ausführen. Die Partialbruchzerlegung aber muss nur einmal angewendet werden und so bekommt man für diesen Fall die Transformationsgleichung

$$\begin{aligned}\int_0^{s_r} \mathrm{d}s_{r+1} \frac{\partial}{\partial s_{r+1}} \mathrm{G}(s_{r+1}, a_{i+1}, ..., a_w; y) \\ = \int_0^{s_r} \frac{\mathrm{d}s_{r+1}}{s_{r+1} - y} \mathrm{G}\left(a_{i+1}, ..., a_w; y\right) \\ + \int_0^{s_r} \frac{\mathrm{d}s_{r+1}}{s_{r+1} - a_{i+1}} \mathrm{G}\left(s_{r+1}, a_{i+2}, ..., a_w; y\right) \\ - \int_0^{s_r} \frac{\mathrm{d}s_{r+1}}{s_{r+1} - a_{i+1}} \mathrm{G}\left(a_{i+1}, a_{i+2}, ..., a_w; y\right) \ .\end{aligned} \tag{5.37}$$

Steht das kleinste Element an der letzten Stelle, dann ist die Ausführung der beschriebenen Rechenschritte nicht mehr möglich, weil die Produktregel in (5.34) nicht mehr angewendet werden kann. Würde man hier direkt die Stammfunktion auswerten, dann bekäme man Probleme mit der konsistenten Definition der Imaginärteile bei der Partialbruchzerlegung. Als Ausweg bieten sich Shuffle-Identitäten an, mit denen man auch diesen letzten Fall behandeln kann. Dazu schreibt man die betreffende G-Funktion wie folgt um:

$$\begin{aligned}\mathrm{G}(a_1, ..., a_k, \underbrace{0, ..., 0}_{m-1}, s_r; y) = \mathrm{G}(a_1, ..., a_k; y)\, \mathrm{G}(\underbrace{0, ..., 0}_{m-1}, s_r; y) \\ - \sum_{\substack{(b_1,...,b_{k+m})=(a_1,...,a_k) \sqcup\!\sqcup (0,...,0,s_r) \\ \setminus\ (a_1,...,a_k,0,...,0,s_r)}} \mathrm{G}(b_1, ..., b_{k+m}; y) \ .\end{aligned} \tag{5.38}$$

Die Summe geht über alle Parameterkombinationen, die durch das Shuffle-Produkt gegeben sind, mit Ausnahme von $(a_1, ..., a_k, 0, ..., 0, s_r)$. Man erhält nach Anwendung von (5.38) nur G-Funktionen, bei denen der kleinste Parameter nicht mehr am Ende steht, so dass wieder die beiden anderen Transformationsgleichungen benutzt werden können.

5.3.3 Beschleunigung der Konvergenz

Mit den beschriebenen Transformationsformeln sind nun alle Multiplen Polylogarithmen im Prinzip berechenbar. Als nächstes ist die Beschleunigung der Konvergenzgeschwindigkeit wichtig. Beispielsweise konvergiert die Reihe der Funktion

$$\mathrm{G}(1.01, 1.02, 1.03; 1) \tag{5.39}$$

nur relativ langsam. Die problematischen Parameterpunkte sind alle die, die nahe bei der Eins liegen. Wie im Abschnitt 5.1 über Dilogarithmen erläutert, ist die Anwendung weiterer Transformationen, wie z.B. $x \to 1-x$, ein Standardverfahren zur Beschleunigung. Alle bisher in der Literatur betrachteten Polylogarithmen haben jedoch nur maximal eine Skala, der Multiple Polylogarithmus besitzt im allgemeinen viele verschiedene Skalen. Transformationen lassen sich zwar finden, zum Beispiel

$$\mathrm{G}\left(z_{1\pm},...,z_k;1-y\right) = \mathrm{G}\left(z_{1\pm},...,z_k;1\right) + \int_0^y \frac{\mathrm{d}t}{t-(1-z_1)_\mp} \mathrm{G}\left(z_2,...,z_k;1-t\right) \ , \quad (5.40)$$

aber bei der systematischen Anwendung treten dann Schwierigkeiten auf. Es werden in (5.40) durch den zweiten Term G-Funktionen produziert, bei denen die Transformation schrittweise auf alle Argumente angewandt wird. Im allgemeinen können dadurch „gute“ Argumente, d.h. solche die für die schnelle Konvergenz günstig liegen, in solche verwandelt werden, die eine schlechte Konvergenz verursachen. Es ist nicht ersichtlich, wie man nur bestimmte Argumente transformiert und andere unbeeinflußt lassen kann. Die Transformationen wirken sich immer auf keine oder auf alle Argumente aus.

Die „Rettung“ kommt in Form der sogenannten Hölder-Faltung, die in [55] hergeleitet und ausführlich beschrieben wird. Die Hölder-Faltung lautet für G-Funktionen

$$\mathrm{G}\left(z_1,...,z_w;1\right) = \sum_{j=0}^{w} (-1)^j \, \mathrm{G}\left(1-z_j,1-z_{j-1},...,1-z_1;1-\frac{1}{p}\right) \mathrm{G}\left(z_{j+1},...,z_w;\frac{1}{p}\right) \ , \quad (5.41)$$

für $z_1 \neq 1, \ z_w \neq 0$.

Zum einen werden hier die Skalen teilweise von den problematischen Einsen weggeschoben, zum anderen und viel entscheidender kann man beliebig $p > 1$ wählen. Mit Hilfe der Skalierungstransformation (5.21) kann man dann auf G-Funktionen reduzieren, in denen alle Parameter mit p multipliziert auftreten. Oft ist $p = 2$ die günstigste Wahl. Die skalierten Parameter liegen dann alle weit von den problematischen Parameterpunkten entfernt. Die Anwendung der Hölder-Faltung erzeugt natürlich mehr Terme als ursprünglich vorhanden, was die ganze Auswertung teilweise wieder verlangsamt. In der konkreten Implementierung wird man daher schauen müssen, in welchen Gebieten man die Hölder-Faltung anwendet.

Die Ersetzung der ursprünglichen Reihe durch eine Bernoulli-Reihe ist bei Multiplen Polylogarithmen nicht ohne weiteres möglich. Lässt sich ein G mit einem einfacheren Polylogarithmus identifizieren, dann lohnt es sich, die entsprechenden numerischen Auswertungsverfahren des einfacheren Polylogarithmus anzuwenden. Diese sind immer schneller.

5.4 Implementierung in GiNaC

Ausgangspunkt für die Beschäftigung mit numerischer Auswertung war der Wunsch die Polylogarithmen, die in Lösungen von Feynman-Integralen auftreten, praktisch verwenden zu können. Für alle einfachen Polylogarithmen einschließlich der Harmonischen Polylogarithmen waren die Verfahren zur Auswertung schon entwickelt. Das Verfahren für Multiple Polylogarithmen wurde von uns entwickelt [54] und ist im vorigen Abschnitt beschrieben

worden. Es ist nun sehr wünschenswert für alle diese Polylogarithmen eine benutzbare numerische Auswertung zur Verfügung zu haben. GiNaC bietet sich hierfür als Basis an. Bis zum Zeitpunkt dieser Arbeit waren nur die Auswertung von Dilogarithmen, Trilogarithmen und Riemannschen ζ-Funktionen in GiNaC implementiert. Es mussten also praktisch alle Polylogarithmen erst noch programmiert werden. Der natürlich Weg ist, bei den einfachen Polylogarithmen anzufangen und nach und nach die allgemeineren Polylogarithmen hinzuzufügen. Die folgende Beschreibung der Implementierung richtet sich genau nach dieser Reihenfolge und behandelt jeweils einen bestimmten Typ von Polylogarithmus. Neben dem Verfahren und den Optimierungen werden auch die jeweiligen Testmöglichkeiten beschrieben.

5.4.1 Klassischer Polylogarithmus

Die notwendigen Verfahren sind schon seit langem bekannt. In [50] und darauf aufbauend in [56, 57] wurden alle notwendigen Formeln erarbeitet. Eine populäre Darstellung findet sich in [53, 58]. Neben der Reihendarstellung

$$\mathrm{Li}_n(x) = \sum_{i=1}^{\infty} \frac{x^i}{i^n} \tag{5.42}$$

sind dies vor allem die Transformationsformeln

$$\begin{aligned}\mathrm{Li}_n(x) = (-1)^{n+1}\,\mathrm{Li}_n\Big(\frac{1}{x}\Big) - \frac{1}{n!}\ln^n(-x) \\ - \sum_{j=0}^{n-2} \frac{1}{j!}(1+(-1)^{n-j})(1-2^{1-n+j})\zeta(n-j)\ln^j(-x)\end{aligned} \tag{5.43}$$

und

$$\mathrm{Li}_n(x) = \sum_{j=0}^{n-2} \frac{\ln^j(x)}{j!}\Big[\zeta(n-j) - \mathrm{S}_{1,n-j-1}(1-x)\Big] - \frac{1}{(n-1)!}\ln^{n-1}(x)\ln(1-x)\,. \tag{5.44}$$

Prinzipiell genügt es, diese Formeln für reelle Argumente zu kennen, da man alle anderen notwendigen Transformationen durch die mehrfache Hintereinanderausführung dieser beiden erzeugen kann. Sie dienen hier vor allem der Illustration der Komplexität im Vergleich zu den folgenden Polylogarithmen. Außerdem lässt sich an Gleichung (5.44) auch ein neues Phänomen erkennen. Die Transformationsformeln enthalten nunmehr höhere Polylogarithmen als Faktoren, d.h. die Transformation ist in gewissem Sinne keine geschlossene Operation mehr. Bestimmte Transformationen für Nielsens Polylogarithmus S haben ebenso diese Eigenschaft, sie benötigen Harmonische Polylogarithmen.

Für die Implementierung wurden von uns durch Zeitmessung die Gebiete der komplexen Ebene bestimmt, in denen die verschiedenen Transformationen die numerische Auswertung beschleunigen. Die Grenzen der Gebiete wurden so angepasst, dass sie möglichst einfach sind, weil die Abfrage, in welchem Gebiet ein Argument liegt, Zeit kostet und bei einfachen Gebieten erheblich schneller ist. Einfache Gebiete sind in der Regel Rechtecke in der komplexen Ebene, so dass man mittels der Frage *kleiner als* oder *größer als* den

Real- oder Imaginärteil prüfen kann. Die konkreten Werte, die für die Implementierung gewählt wurden, können sich im Verlauf von Optimierungen verschiedener Programmteile noch verschieben.

Die Bernoulli-Reihe lautet für Klassische Polylogarithmen

$$\mathrm{Li}_n(x) = \sum_{j=0}^{\infty} \frac{C_n(j)}{(j+1)!}(-\ln(1-x))^{j+1} \tag{5.45}$$

mit

$$C_1(j) = \delta_{j,0}, \qquad C_{n+1}(j) = \sum_{k=0}^{j} \binom{j}{k} \frac{B_{j-k}}{k+1} C_n(k)\,. \tag{5.46}$$

Für $n = 2$ sind die Vorfaktoren $C_n(j)$ einfach die Bernoulli-Zahlen selbst. Bei größeren n werden die Vorfaktoren rekursiv aus Bernoulli-Zahlen berechnet. Es ist sinnvoll, die Vorfaktoren einmalig zu berechnen und in einer Tabelle zu speichern. Prinzipiell ist die Bernoulli-Reihe immer dann zu verwenden, wenn das Argument weit von Null entfernt ist. Der genaue Umschaltpunkt zwischen normaler Reihe und Bernoulli-Reihe muss aber durch Zeitmessungen experimentell bestimmt werden. Bei uns wurde er zu $\mathrm{Re}\,x < 0.25$ für $n = 2$ und $\mathrm{Re}\,x < 0.3$ für $n \geq 3$ gewählt. Außerdem wird für $n \geq 12$ immer die normale Reihe benutzt, da bei solch großen Exponenten die normale Reihe schneller konvergiert.

Die Tabellen für die Vorfaktoren C_n hängen von der eingestellten Präzision ab. Wird die Präzision erhöht, dann werden mehr Reihenglieder und damit auch mehr C_n benötigt. Die Tabelle muß dann gegenebenfalls erweitert werden. In GiNaC ist es jederzeit möglich die Präzision zu ändern. Die Implementierung muß also darauf achten, dass die Tabelle der momentanen Präzision entspricht.

In Abbildung 5.4.1 ist der Programmcode gezeigt, der die Reihe eines Klassischen Polylogarithmus numerisch aufsummiert. Der abgedruckte Programmcode entspricht weitgehend der aktuell existierenden Implementierung in GiNaC. Ohne auf Details einzugehen, sollen kurz ein paar wesentliche Punkte hervorgehoben werden. In Zeile 5 wird die Transformation für die Bernoulli-Reihe einmalig berechnet. In jedem Durchlauf der C^{++}-Schleife in den Zeilen 10 bis 20 wird ein weiteres Reihenglied der Bernoulli-Reihe aufsummiert. Am Ende der Schleife wird jedesmal geprüft, ob die numerische Präzision schon erfüllt ist, indem der neue Summationswert mit dem vorigen verglichen wird. Die Bernoulli-Zahlen liegen vorberechnet in einem Vektor `X`. Sollte in Zeile 15 erkannt werden, dass der vorhande `X`-Vektor zu klein ist, wird er vergrößert.

Sehr wichtig ist der Test der Implementierung, d.h. man muss sicherstellen, dass die Funktionen auch richtig berechnet werden. Dabei wurden von uns mehrere Strategien angewandt. Zunächst ist da der Vergleich mit existierenden Programmen. Für eine Menge von komplexen Zahlen kann man die Ergebnisse für verschiedene Präzisionen vergleichen. Dies ist der wichtigste Test für die eigentliche Reihenauswertung. Als nächsten Schritt muss man die Transformationen überprüfen. Bei der Transformation $x \to 1 - x$ nutzt man die Eigenschaft, dass es Werte für x gibt, bei denen x und $1 - x$ beide im Konvergenzbereich liegen. Also kann man den gleichen Punkt auf zwei verschiedene Arten berechnen: einmal ohne Transformation, z.B. $x = 0.4$, und einmal mit Transformation, dann $x = 0.6$. Es muss absolute Übereinstimmung vorliegen. Die Polylogarithmen haben spezielle Werte für die

```
 1    cl_N Lin(int n, const cl_N& x)
 2    {
 3        vector<cl_N>::const_iterator it = X[n].begin();
 4        vector<cl_N>::const_iterator xend = X[n].end();
 5        cl_N u = -log(1-x);
 6        cl_N factor = u * cl_float(1, float_format(Digits));
 7        cl_N res = u;
 8        cl_N resbuf;
 9        unsigned i = 2;
10        do {
11            resbuf = res;
12            factor = factor * u / i;
13            res = res + (*it) * factor;
14            i++;
15            if (++it == xend) {
16                make_X_bigger();
17                it = X[n].begin() + (i-2);
18                xend = X[n].end();
19            }
20        } while (res != resbuf);
21        return res;
22    }
```

Abbildung 5.5: Beispielcode der Implementierung der Reihensummierung in GiNaC.

Argumente $x = 1$ und $x = -1$. Man kann diese vergleichen, indem man die Argumente diesen Punkten nähert und die Werte der Polylogarithmen auf ihre stetige Annäherung an die speziellen Werte untersucht. Die einfachste und auch effektivste Überprüfung von Werten außerhalb des Konvergenzbereichs, d.h. letztlich der Korrektheit der $\frac{1}{x}$-Transformation, ist der Vergleich mit anderen Programmen. Wir haben *Mathematica* benutzt, da man auch dort die Präzision beliebig einstellen kann und die Auswertung an vielen repräsentativ gewählten Punkten durch ein *Mathematica*-Programm automatisieren kann. Mit diesen Tests ist die Korrektheit unserer Implementierung für die Klassischen Polylogarithmen sichergestellt. Die Tests wurden im übrigen so gestaltet, dass sie als automatisierte Tests in der Selbstüberprüfung von GiNaC enthalten sind. Man kann also jederzeit diese Tests starten und sich überzeugen, dass sich bei der Erweiterung und Optimierung von GiNaC an dieser Stelle keine unbeabsichtigten Fehler eingeschlichen haben.

Der Klassische Polylogarithmus ist nun in GiNaC enthalten und kann zum Beispiel wie folgt

```
...
ex result = sqrt( Li(5, p0 - p1) );
...
```

verwendet werden. Die Funktion `Li(n,x)` wird, wenn möglich, zu speziellen Werten evaluiert, also z.B. wenn `x` die Zahl Eins ist. Für die numerische Auswertung erwartet GiNaC

für `n` eine positive Ganzzahl und für `x` eine beliebige komplexe Zahl.

5.4.2 Nielsens Polylogarithmus

Die Formeln für Nielsens Polylogarithmus finden sich in [50, 56, 57], wobei die letzte Arbeit nicht nur in der Darstellung die beste Quelle ist, sondern auch die Fehler der Vorgänger nicht mehr enthält.

Die Reihendarstellung lautet

$$\mathrm{S}_{n,p}(x) = \sum_{i=1}^{\infty} \frac{x^i}{i^{n+1}} \sum_{j_1=1}^{i-1} \frac{1}{j_1} \cdots \sum_{j_{p-1}=1}^{j_{p-2}-1} \frac{1}{j_{p-1}} \equiv \sum_{i=1}^{\infty} \frac{x^i}{i^{n+1}} Z_{p-1}(i-1)\,. \tag{5.47}$$

$Z_{p-1}(i-1)$ ist eine Euler-Zagier-Summe, bei der alle Gewichte $m_i = 1$ sind:

$$Z_k(N) = \sum_{i_1=1}^{N} \sum_{i_2=1}^{i_1-1} \cdots \sum_{i_k=1}^{i_{k-1}-1} \frac{1}{i_1}\frac{1}{i_2}\cdots\frac{1}{i_k}\,. \tag{5.48}$$

Die zwei wichtigen Transformationen lauten

$$\begin{aligned}\mathrm{S}_{n,p}(x) = {} & \sum_{j=0}^{n-1} \frac{\ln^j x}{j!}\Big[\mathrm{S}_{n-j,p}(1) - \sum_{k=0}^{p-1} \frac{(-1)^k \ln^k(1-x)}{k!}\mathrm{S}_{p-k,n-j}(1-x)\Big] \\ & + \frac{(-1)^p}{n!p!}\ln^n x \ln^p(1-x)\end{aligned} \tag{5.49}$$

und

$$\begin{aligned}\mathrm{S}_{n,p}(x) = {} & (-1)^n \sum_{k=0}^{p-1}(-1)^k \sum_{m=0}^{k} \frac{\ln^m(-x)}{m!}\binom{n+k-m-1}{k-m}\mathrm{S}_{n+k-m,p-k}\Big(\frac{1}{x}\Big) \\ & + (-1)^p\left[\sum_{j=0}^{n-1}\frac{\ln^j(-x)}{j!}P_{n-j,p} + \frac{1}{(n+p)!}\ln^{n+p}(-x)\right],\end{aligned} \tag{5.50}$$

wobei

$$\begin{aligned}P_{n,p} = {} & (-1)^{n+p-1}\sum_{k=0}^{p-1}(-1)^k(1-(-1)^n\delta_{k,0})\binom{n+k-1}{k}\sum_{j=0}^{[(n+k-1)/2]}(-1)^j\frac{\pi^{2j}}{(2j)!} \\ & \times \mathrm{S}_{n+k-2j,p-k}(1) + (-1)^{[(n+p)/2]+n}\epsilon_{n+p}\frac{\pi^{n+p}}{(n+p)(n-1)!p!}\end{aligned} \tag{5.51}$$

mit $\epsilon_m = 1$ wenn m gerade und $\epsilon_m = 0$ wenn m ungerade. $[x]$ bedeutet die größte Ganzzahl, die kleiner als x ist.

Die beiden Transformationen benötigen keine höheren Polylogarithmen. Man kann sehen, dass die Formeln immer komplizierter werden. Mit steigender Komplexität verschiebt sich auch der Anwendungsbereich der Formeln zur Beschleunigung der Konvergenz zu größeren Argumenten hin.

Bei der Reihendarstellung fällt auf, dass die Untersummen eine Menge von rationalen Zahlen erzeugen, die für unterschiedliche Argumente x gleich bleiben. Hier bietet sich eine Tabellierung der Untersummen an, um die Geschwindigkeit zu verbessern. Unsere Implementierung tut dies und speichert die Werte der Untersummen als rationale Zahlen. Dadurch sind sie von der eingestellten numerischen Präzision unabhängig und müssen nicht neu berechnet werden, wenn sich diese ändert.

Die Bernoulli-Reihe lautet

$$\mathrm{S}_{n,p}(x) = \sum_{j=0}^{\infty} \frac{C_{n,p}(j)}{(j+1)!}(-\ln(1-x))^{j+1} \tag{5.52}$$

mit

$$C_{1,p}(j) = \frac{B_j}{j!}, \qquad C_{n+1,p}(j) = \sum_{k=0}^{j} \frac{B_{j-k}}{(j-k)!}\frac{C_{n,p}(k)}{p+k} \tag{5.53}$$

Die Tabellierung der Vorfaktoren ist nun aufwändiger. Man hat eine zweidimensionale Tabelle zu verwalten. Hier muss natürlich eine praktische Grenze gesetzt werden, bis zu welchen Werten von n und p die Tabelle erzeugt wird. Zurzeit ist die Bernoulli-Reihe noch nicht implementiert, da das Hauptaugenmerk der Implementierung auf die Harmonischen und Multiplen Polylogarithmen gerichtet war.

In GiNaC kann die Funktion als `S(n,p,x)` benutzt werden. Ist $x = 1$ wird die Funktion automatisch zu ihrem Spezialwert evaluiert. Ist $p = 1$ wird die Funktion in einen Klassischen Polylogarithmus umgewandelt. Sind n und p positive Ganzzahlen und x eine beliebige komplexe Zahl, dann wird die Funktion numerisch ausgewertet.

5.4.3 Harmonischer Polylogarithmus

Der Harmonische Polylogarithmus wurde in [49] eingeführt und die numerische Auswertung dort und in [59] beschrieben. Die Autoren stellen in [59] auch eine Implementierung vor, die für ein maximales Gewicht $m = 4$ den Harmonischen Polylogarithmus in doppelter FORTRAN-Präzision berechnen kann. Allgemein, ohne Einschränkung des Gewichts, der Tiefe oder der Skala können diese Funktionen nun erstmals mit der hier beschriebenen Implementierung in GiNaC berechnet werden[3].

Der Harmonische Polylogarithmus wird den Überlegungen in Abschnitt 5.2.3 über *trailing zeros* folgend rekursiv definiert (siehe Gleichung (5.25)):

$$\mathrm{H}(a, \vec{m}; x) = \int_0^x \mathrm{d}t\; f(a;t)\, \mathrm{H}(\vec{m};t)\,, \tag{5.54}$$

[3] Erst vor kurzem wurde in [60] eine Implementierung für *Mathematica* vorgestellt, die auch allgemeine Harmonische Polylogarithmen berechnen kann.

wobei statt einem allgemeinen $\frac{1}{t-a}$ nur die drei folgenden Formen

$$\begin{aligned} f(1;t) &= \frac{1}{1-t} \\ f(0;t) &= \frac{1}{t} \\ f(-1;t) &= \frac{1}{1+t} \end{aligned} \tag{5.55}$$

als Integranden zugelassen werden. $\vec{m} = (m_1, \ldots, m_k)$ steht für eine beliebige Liste von Argumenten.

Die Transformationen können ebenfalls nur rekursiv notiert werden. Als Beispiel zeigen wir nur die $\frac{1}{x}$-Transformation. Die Basistransformationen lauten dafür

$$\begin{aligned} H\Big(1;\frac{1}{x}\Big) &= H(1;x) + H(0;x) + i\pi \\ H\Big(0;\frac{1}{x}\Big) &= -H(0;x) \\ H\Big(-1;\frac{1}{x}\Big) &= H(-1;x) - H(0;x)\,. \end{aligned} \tag{5.56}$$

Darauf aufbauend kann man mit

$$\mathrm{H}\Big(a,\vec{m};\frac{1}{x}\Big) = \mathrm{H}(a,\vec{m};1) - \int_1^x \mathrm{d}t\ \frac{1}{t^2}\ f\Big(a,\frac{1}{t}\Big)\ \mathrm{H}\Big(\vec{m};\frac{1}{t}\Big) \tag{5.57}$$

für alle Harmonischen Polylogarithmen konvergente Reihendarstellungen finden. Die Transformationsformel (5.57) ist je nach dem Wert von a noch auf unterschiedliche Weise umzuformen, zum Beispiel mit einer Partialbruchzerlegung. Die Details kann man in [49, 59] nachlesen.

Man kann für den Harmonischen Polylogarithmus Formeln für die Transformation

$$x \to \frac{1-x}{1+x} \tag{5.58}$$

finden. Man benötigt deshalb die $1-x$ - Transformation nicht. Mit (5.58) lassen sich alle problematischen Argumente ebenso behandeln. Zusätzlich kann man weitere Fälle, wie die in 5.1 erwähnten Punkte

$$x_{1,2} = \frac{1}{2} \pm \sqrt{\frac{3}{4}}i \tag{5.59}$$

behandeln.

Für die numerische Auswertung muss man noch eine Konvention für den Imaginärteil der Schnitte setzen. Die Konvention für alle bisherigen Funktionen wurde so gewählt, dass sie dem am weitesten verbreiteten Standard [61] entsprechen. Die „Väter“ des Harmonischen Polylogarithmus haben nun leider die entgegengesetzte Konvention gewählt und es war nun die Frage, ob man lieber alle Funktionen einem einheitlichen Standard unterwirft oder beim Harmonischen Polylogarithmus einen Sonderweg beschreitet. Wir haben uns

für den konsequenten Weg entschieden, auch weil sonst die automatische Evaluierung von höheren Polylogarithmen in niedrigere Polylogarithmen erschwert würde. Man kann als Benutzer selbst die ursprüngliche Konvention [59] erzwingen, indem man dem Argument einen positiven kleinen Imaginärteil gibt. Dieser muss so klein gewählt werden, dass er das Ergebnis innerhalb der vorgegebenen Präzision nicht merklich beeinflusst.

Die Bernoulli-Reihe für den Harmonischen Polylogarithmus lautet nun[4]

$$H_{m_1,\ldots,m_k}(x) = \sum_{j=0}^{\infty} \frac{C_{m_1,\ldots,m_k}(j)}{(j+1)!} \left(-\ln(1-x)\right)^{j+1}, \tag{5.60}$$

wobei

$$C_{1,m_2,\ldots,m_k}(j) = \begin{cases} 0, & j = 0, \\ C_{m_2,\ldots,m_k}(j-1), & j > 0, \end{cases} \tag{5.61}$$

und

$$C_{m_1+1,m_2,\ldots,m_k}(j) = \sum_{k=0}^{j} \left(\begin{array}{c} j \\ k \end{array} \right) \frac{B_{j-k}}{k+1} C_{m_1,m_2,\ldots,m_k}(k) \,. \tag{5.62}$$

Die Liste der hier benötigten Vorfaktoren ist nun so umfangreich, dass sie nicht mehr effizient tabelliert werden kann. In der jetzigen Implementierung wird die Bernoulli-Reihe nicht benutzt. Die Auswertungsgeschwindigkeit ist aber auch weitgehend nicht mehr durch die Reihensummierung bestimmt, sondern durch die aufwändigen rekursiven Transformationsschritte, so dass dieser Arbeitsschritt nun stattdessen entscheidend wird. Die einzige Optimierung die bleibt ist, einmal berechnete Transformationen zu speichern und wieder zu verwenden. Da die Parameteranzahl jedoch beliebig groß sein kann, ist die Speicherung der Transformationen ein Balanceakt zwischen effizienter numerischer Auswertung und geringem Speicher- und Datenverwaltungsaufwand. Zurzeit wird die Zwischenspeicherung noch nicht durchgeführt, sie ist aber für die Zukunft geplant.

Die Implementierung kann man auf mehrere Arten testen. Harmonische Polylogarithmen von der Tiefe Eins entsprechen Nielsens Polylogarithmen. Diese kann man also einfach durch den direkten Vergleich der beiden Polylogarithmen für gleiche Argumente überprüfen. Weiterhin kann man mit der Implementierung von [59] vergleichen. Grenzübergänge an bestimmten Punkten zu betrachten ist wichtig. Der Harmonische Polylogarithmus hat die Transformation $\frac{1-x}{1+x}$ implementiert. Diese hat Überschneidungen mit x, $1-x$ und $\frac{1}{x}$. Der Vergleich der verschiedenen Transformationen für diese Fixpunkte schafft weitere Sicherheit.

5.4.4 Multiple Zeta-Funktion

In den Transformationen des Harmonischen und des Multiplen Polylogarithmus werden alle möglichen Multiplen Zeta-Funktionen erzeugt. Diese müssen daher auch berechnet werden können.

Es werden keine Transformationen für nicht-konvergente Argumente benötigt. Die Multiplen Zeta-Funktionen $\zeta(m_1,\ldots,m_k)$ sind immer konvergent, außer für $m_1 = 1$. Allerdings

[4] Hier wird die kompakte Schreibweise wie bei den G benutzt, bei der die jeweiligen Gewichte m_i als Indizes notiert werden.

ist die Konvergenzgeschwindigkeit immer die schlechteste. Die Beschleunigung der Reihenauswertung ist also sehr wichtig. In der Literatur findet man dazu zwei Verfahren. Das eine ist die Hölder-Faltung [55] und wurde schon beschrieben. Das andere ist etwas älter und stammt noch aus der Zeit, als die Shuffle-Algebrastrukturen nicht bekannt waren. Damals hatte man zwar eine Reihe von Beziehungen zwischen verschiedenen ζ-Funktionen mit gleichem Gewicht, allerdings konnten einige nicht bewiesen werden und man interessierte sich für neue, noch unbekannte Beziehungen. In diesem Rahmen wurde von einer Gruppe von Forschern versucht, die Beziehungen zwischen den ζ-Funktionen quasi experimentell zu bestimmen, indem man die ζ-Funktionen numerisch ausrechnet [62]. Dies bedurfte einer sehr hohen Präzision von mehreren hundert Nachkommastellen. Das dazu von Crandall [63] erdachte Verfahren war in der Lage beliebige Multiple Zeta-Funktionen mit extrem hoher Präzision zu berechnen.

Beide Verfahren wurden implementiert, optimiert und miteinander verglichen. Das Ergebnis war, dass das Verfahren von Crandall nur bei sehr großer Präzision schneller ist als die Hölder-Faltung. Solch hohe Präzisionen sind nicht der Normalfall bei der Benutzung von GiNaC. Die Hölder-Faltung wird daher intern in den meisten Fällen benutzt. GiNaC prüft vor der numerischen Auswertung, welches Verfahren auf Grund der eingestellten Präzision als am schnellsten erkannt wurde, und benutzt dieses.

Eine noch fehlende Optimierung ist die Zwischenspeicherung von oft abgefragten ζ-Funktionen, die viel leichter als bei den anderen Polylogarithmen möglich ist, weil keine Skalen vorhanden sind, sondern nur die ganzzahligen Gewichte. Ein weiterer Ansatz, der noch verfolgt werden kann, ist es ζ-Funktionen mit Hilfe der (Quasi-)Shuffle-Algebra in eine Menge von leichter berechenbaren Basis-ζ-Funktionen zu zerlegen. Für ein paar einfache ζ-Funktionen, wie z.B. $\zeta(2,1) = \zeta(3)$ sind die Reduktionen schon implementiert. Eine systematische und allgemeine Behandlung fehlt aber noch.

Die Überprüfung der Implementierung kann durch das numerische Nachrechnen von Shuffle- und Quasi-Shuffle-Identitäten sichergestellt werden.

5.4.5 Multipler Polylogarithmus

Das neue Verfahren wurde schon in 5.3 beschrieben. Die Implementierung besteht in der direkten Übersetzung dieser Formeln in rekursive C++-Funktionen, die die jeweiligen mathematischen Operationen umsetzen. Auch die Optimierung wurde schon beschrieben. Es kommt nur die Hölder-Faltung zum Einsatz. Die Auswertung der eigentlichen Reihen ist dadurch schon sehr effektiv. Dominiert wird der Zeitaufwand für die ganze Auswertung von der Ausführung der rekursiven Transformationen.

Die Tests sind im wesentlichen die schon mehrfach erwähnten. Zum einen der Vergleich mit den anderen, schon getesteten Polylogarithmen für Parameter, bei denen die Multiplen Polylogarithmen diesen Funktionen entsprechen. Zum anderen der Vergleich der Werte mit und ohne benutzte Hölder-Faltung für die gleichen Argumente. Außerdem die qualitative Betrachtung der Stetigkeit an den Grenzen des Konvergenzbereichs. Als letztes und sehr wichtiges Instrument kamen die (Quasi-)Shuffle-Identitäten zum Einsatz. Die numerische Überprüfung der Identitäten ist ein strenger Test der Implementierung.

GiNaC bietet beide Varianten des Multiplen Polylogarithmus an: G und Li. Der Harmo-

nische Polylogarithmus lässt in seiner ursprünglichen Definition auch negative Gewichte zu. GiNaC bieten eine Funktion `convert_H_to_Li()` an, um eine fehleranfällige Konversion von Hand zu vermeiden.

5.4.6 Polylogarithmen in GiNaC

In GiNaC können alle Polylogarithmen für beliebige Argumente ausgewertet werden. Damit ist der Wunsch erfüllt, die Reihenentwicklung der R-Funktion praktisch verwenden zu können. GiNaC wird stetig weiterentwickelt und verbessert. Das betrifft auch die numerische Auswertung von Funktionen. In Zukunft werden noch die beschriebenen Optimierungsmöglichkeiten erprobt und gegebenfalls eingesetzt. Zur Zeit ist GiNaC die einzige Software, die Multiple Polylogarithmen auswerten kann.

6 Zusammenfassung

Die Berechnung von experimentell überprüfbaren Vorhersagen aus dem Standardmodell mit Hilfe störungstheoretischer Methoden ist schwierig. Die Herausforderungen liegen in der Berechnung immer komplizierterer Feynman-Integrale und dem zunehmenden Umfang der Rechnungen für Streuprozesse mit vielen Teilchen. Neue mathematische Methoden müssen daher entwickelt und die zunehmende Komplexität durch eine Automatisierung der Berechnungen gezähmt werden. In Kapitel 2 wurde eine kurze Einführung in diese Thematik gegeben. Die nachfolgenden Kapitel waren dann einzelnen Beiträgen zur Lösung dieser Probleme gewidmet.

In Kapitel 3 haben wir ein Projekt vorgestellt, das für die Analysen der LHC-Daten wichtig sein wird. Ziel des Projekts ist die Berechnung von Einschleifen-Korrekturen zu Prozessen mit vielen Teilchen im Endzustand. Das numerische Verfahren wurde dargestellt und erklärt. Es verwendet Helizitätsspinoren und darauf aufbauend eine neue Tensorreduktionsmethode, die Probleme mit inversen Gram-Determinanten weitgehend vermeidet. Es wurde ein Computerprogramm entwickelt, das die Berechnungen automatisiert ausführen kann. Die Implementierung wurde beschrieben und Details über die Optimierung und Verifizierung präsentiert.

Mit analytischen Methoden beschäftigte sich das vierte Kapitel. Darin wurde das xloops-Projekt vorgestellt, das verschiedene Feynman-Integrale mit beliebigen Massen und Impulskonfigurationen analytisch berechnen kann. Die wesentlichen mathematischen Methoden, die xloops zur Lösung der Integrale verwendet, wurden erklärt. Zwei Ideen für neue Berechnungsverfahren wurden präsentiert, die sich mit diesen Methoden realisieren lassen. Das war zum einen die einheitliche Berechnung von Einschleifen-N-Punkt-Integralen, und zum anderen die automatisierte Reihenentwicklung von Integrallösungen in höhere Potenzen des dimensionalen Regularisierungsparameters ϵ. Zum letzteren Verfahren wurden erste Ergebnisse vorgestellt.

Die Nützlichkeit der automatisierten Reihenentwicklung aus Kapitel 4 hängt von der numerischen Auswertbarkeit der Entwicklungskoeffizienten ab. Die Koeffizienten sind im allgemeinen Multiple Polylogarithmen. In Kapitel 5 wurde ein Verfahren für deren numerische Auswertung vorgestellt. Dieses neue Verfahren für Multiple Polylogarithmen wurde zusammen mit bekannten Verfahren für andere Polylogarithmus-Funktionen als Bestandteil der C^{++}-Bibliothek GiNaC implementiert.

A Helizitätsspinoren

Für einen vierdimensionalen, masselosen Dirac-Spinor $u(p)$, d.h. $\not{p}u(p) = 0,\ p^2 = 0$, lassen sich Helizitätsspinoren wie folgt definieren:

$$|p\pm\rangle = \frac{1 \pm \gamma_5}{2}\, u(p)\ , \qquad \langle p\pm| = \overline{u(p)}\ \frac{1 \mp \gamma_5}{2}\ .$$

Skalarprodukte zwischen Helizitätsspinoren werden oft benötigt. Man führt deshalb eine kompaktere Schreibweise für Skalarprodukte ein:

$$\langle pq\rangle \equiv \langle p-|q+\rangle, \qquad [pq] \equiv \langle p+|q-\rangle\ . \tag{A.1}$$

Skalarprodukte mit anderen Helizitätskombinationen bzw. mit gleichem, masselosem Impuls verschwinden:

$$\langle p+|q+\rangle = \langle p-|q-\rangle = [pp] = \langle pp\rangle = 0 \tag{A.2}$$

Es existieren nun eine Reihe nützlicher Beziehungen für Helizitätsspinoren:

Antisymmetrie

$$\langle pq\rangle = -\langle qp\rangle, \quad [pq] = -[qp] \tag{A.3}$$

Fierz-Identität

$$\langle p+|\gamma^\mu|q+\rangle\langle r+|\gamma_\mu|s+\rangle = 2[pr]\langle sq\rangle \tag{A.4}$$

Ladungskonjugation

$$\langle p+|\gamma^\mu|q+\rangle = \langle q-|\gamma^\mu|p-\rangle \tag{A.5}$$

Gordon-Identität

$$\langle p\pm|\gamma^\mu|p\pm\rangle = 2p^\mu \tag{A.6}$$

Vollständigkeitsrelationen

$$\begin{aligned} |p\pm\rangle\langle p\pm| &= \frac{1 \pm \gamma_5}{2}\ \not{p} \\ |p\mp\rangle\langle q\pm| &= \frac{\not{p}\not{b}\not{q}}{\langle p\pm|\not{b}|q\mp\rangle}\frac{1 \pm \gamma_5}{2}\ , \quad \text{mit einem beliebigen Vierervektor } b \end{aligned} \tag{A.7}$$

Schouten-Identität

$$\langle pq\rangle\langle rs\rangle = \langle pr\rangle\langle qs\rangle + \langle ps\rangle\langle rq\rangle \tag{A.8}$$

Um die Berechnung von Matrixelementen vollständig mit Helizitätsspinoren ausführen zu können, müssen noch die Polarisationsvektoren durch Helizitätsspinoren ausgedrückt werden:

$$\epsilon_\mu^\pm(p,l) = \pm\frac{\langle l\mp|\gamma_\mu|p\mp\rangle}{\sqrt{2}\langle l\mp|p\pm\rangle}\,. \tag{A.9}$$

l ist hier ein beliebiger, masseloser Impuls, auch Referenzimpuls genannt.

Für die numerische Darstellung der Spinoren wählt man am einfachsten Lichtkegelkoordinaten:

$$\begin{aligned} p_+ &= p_0 + p_3\,, \qquad & p_\perp &= p_1 + ip_2\,, \\ p_- &= p_0 - p_3\,, \qquad & p_{\perp^*} &= p_1 - ip_2\,. \end{aligned} \tag{A.10}$$

Die Helizitätsspinoren lassen sich dann schreiben als

$$\begin{aligned} |p+\rangle &= \frac{1}{\sqrt{|p_+|}}\begin{pmatrix} -p_{\perp^*} \\ p_+ \\ 0 \\ 0 \end{pmatrix}, \qquad \langle p+| = \frac{e^{-i\phi}}{\sqrt{|p_+|}}\begin{pmatrix} 0 & 0 & -p_\perp & p_+ \end{pmatrix}, \\ |p-\rangle &= \frac{e^{-i\phi}}{\sqrt{|p_+|}}\begin{pmatrix} 0 \\ 0 \\ p_+ \\ p_\perp \end{pmatrix}, \qquad \langle p-| = \frac{1}{\sqrt{|p_+|}}\begin{pmatrix} p_+ & p_{\perp^*} & 0 & 0 \end{pmatrix}. \end{aligned} \tag{A.11}$$

Die Phase ϕ ist gegeben durch

$$p_+ = |p_+|e^{i\phi}\,. \tag{A.12}$$

Effektiv lassen sich diese Dirac-Spinoren wegen der 0-Komponenten wie zweikomponentige Weyl-Spinoren behandeln. In der praktischen Berechnung und Implementierung benutzt man daher auch nur zwei Komponenten und verwendet statt der γ-Matrizen direkt die Pauli-Spinmatrizen. Um die Notation konsistent zu halten, wurde hier jedoch durchgehend die vierdimensionale Schreibweise benutzt.

Skalarprodukte lassen sich einfach berechnen:

$$\begin{aligned} \langle pq\rangle &= \frac{1}{\sqrt{|p_+||q_+|}}(q_+p_{\perp^*} - p_+q_{\perp^*}) \\ [qp] &= -\frac{1}{\sqrt{|p_+||q_+|}}e^{-i\phi_p}e^{-i\phi_q}(q_+p_\perp - p_+q_\perp)\,. \end{aligned} \tag{A.13}$$

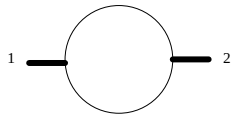
Weitergehende Informationen zu Helizitätsspinoren und dem dazugehörigen Formalismus zur Berechnung von Matrixelementen findet man z.B. in [15, 18, 19].

B Basisintegrale

Hier sind alle für das in Kapitel 3 beschriebene Verfahren notwendigen Basisintegrale mit ihren Lösungen aufgelistet. Die verwendeten Konventionen sind dort zu finden. Die jeweiligen Parameter der Integrale sind nicht explizit angegeben, können aber ganz einfach aus den Lösungen abgelesen werden. Neben der Renormierungsskala μ sind das in der Regel die Impulsquadrate der massiven äußeren Teilchen. Die Graphen zeigen außerdem durch verdickte Linien, welche äußeren Impulse als massiv angenommen sind.

2-Punkt-Integrale

Formeln sowohl für skalare als auch tensorielle Integrale sind hier aufgeführt. s ist der Exponent von $-k^2_{(-2\epsilon)}$ (siehe Abschnitt 3.2.6).



$$I_2 = \frac{1}{\epsilon} + 2 - \ln\left(\frac{-p_1^2}{\mu^2}\right) + \mathcal{O}(\epsilon) \tag{B.1}$$

$$I_2^{s=1} = -\frac{p_1^2}{6} + \mathcal{O}(\epsilon) \tag{B.2}$$

$$I_2^{\mu_1,s} = p_1^{\mu_1} A_{1,0}^s \tag{B.3}$$

$$I_2^{\mu_1\mu_2,s} = p_1^{\mu_1} p_1^{\mu_2} A_{2,0}^s + g^{\mu_1\mu_2} A_{2,1}^s \tag{B.4}$$

Dabei ist

$$A_{r,t}^{s=0} = \left(-\frac{p_1^2}{2}\right)^t \frac{(r-t)!}{(r+1)!} \left\{1 + \epsilon\left[2Z_1(r+1) - Z_1(r-t) - 2\right] + \mathcal{O}(\epsilon^2)\right\} I_2 \tag{B.5}$$

und

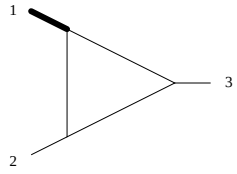
$$\begin{aligned} A_{r,t}^{s>0} = &-\left(p_1^2\right)^s \left(-\frac{p_1^2}{2}\right)^t \epsilon\, \frac{(s-1)!(r+s-t)!}{(r+2s+1)!} I_2 + \mathcal{O}(\epsilon) \\ &+ \text{Terme, die bei Kontraktion mit vierdimensionalen Größen verschwinden} \end{aligned} \tag{B.6}$$

mit der Harmonischen Summe

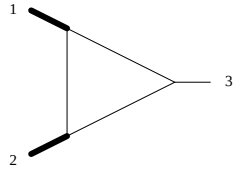
$$Z_1(n) = \sum_{j=1}^{n} \frac{1}{j}\,. \tag{B.7}$$

Skalare 3-Punkt-Integrale

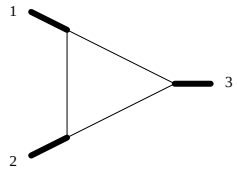
Das Integral, in dem alle drei äußeren Teilchen masselos sind, ist aus kinematischen Gründen Null.



$$I_3^{1m} = \frac{1}{p_1^2}\left\{\frac{1}{\epsilon^2} - \frac{1}{\epsilon}\ln\left(\frac{-p_1^2}{\mu^2}\right) + \frac{1}{2}\ln^2\left(\frac{-p_1^2}{\mu^2}\right) - \frac{\zeta(2)}{2}\right\} + \mathcal{O}(\epsilon) \tag{B.8}$$



$$I_3^{2m} = \frac{1}{(p_1^2 - p_2^2)}\left\{\frac{1}{\epsilon}\left[-\ln\left(\frac{-p_1^2}{\mu^2}\right) + \ln\left(\frac{-p_2^2}{\mu^2}\right)\right] + \ln^2\left(\frac{-p_1^2}{\mu^2}\right) - \ln^2\left(\frac{-p_2^2}{\mu^2}\right)\right\} + \mathcal{O}(\epsilon) \tag{B.9}$$



$$I_3^{3m} = \frac{1}{\nu_3(x_1 - x_2)}\left\{2\,\mathrm{Li}_2\left(\frac{1}{x_2}\right) - 2\,\mathrm{Li}_2\left(\frac{1}{x_1}\right) + \ln\big(x_1 x_2 + \mathrm{sign}(\nu_3)i0\big)\left[\ln\left(\frac{1-x_1}{-x_1}\right) - \ln\left(\frac{1-x_2}{-x_2}\right)\right]\right\} \tag{B.10}$$

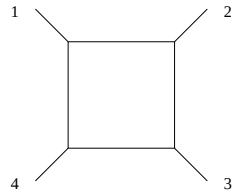
wobei x_1 und x_2 Lösungen der Gleichung

$$x\nu_1 + (1-x)\nu_2 - x(1-x)\nu_3 = 0$$

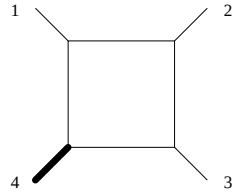
sind. Die ν_i sind eine Permutation der äußeren Massen $\{\nu_1, \nu_2, \nu_3\} = \mathcal{P}\{p_1^2, p_2^2, p_3^2\}$, und zwar so gewählt, dass die Wurzeln x_1 und x_2 im Intervall $[0, 1]$ liegen. Falls das Vorzeichen aller Massen gleich ist, dann muss ν_3 gleich der kleinsten Masse gewählt werden. Ansonsten wird es gleich der Masse gewählt, die das umgekehrte Vorzeichen der anderen beiden hat.

Skalare 4-Punkt-Integrale

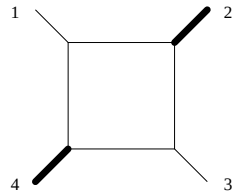
Die Mandelstam-Variablen sind definiert als $s = (p_1+p_2)^2,\ t = (p_1+p_3)^2$. Die Berechnung der Formeln findet sich in [25, 26].



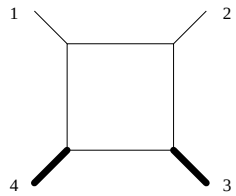
$$
\begin{aligned}
I_4^{0m} = & \frac{4}{\epsilon^2 st} - \frac{2}{\epsilon st}\left[\ln\left(\frac{-s}{\mu^2}\right) + \ln\left(\frac{-t}{\mu^2}\right)\right] \\
& + \frac{1}{st}\left[\ln^2\left(\frac{-s}{\mu^2}\right) + \ln^2\left(\frac{-t}{\mu^2}\right) - \ln^2\left(\frac{-s}{-t}\right) - 8\zeta(2)\right] + \mathcal{O}(\epsilon)
\end{aligned}
\tag{B.11}
$$



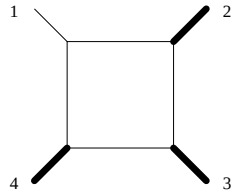
$$
\begin{aligned}
I_4^{1m} = & \frac{2}{\epsilon^2 st} - \frac{2}{\epsilon st}\left[\ln\left(\frac{-s}{\mu^2}\right) + \ln\left(\frac{-t}{\mu^2}\right) - \ln\left(\frac{-p_4^2}{\mu^2}\right)\right] + \frac{1}{st}\left[\ln^2\left(\frac{-s}{\mu^2}\right)\right. \\
& + \ln^2\left(\frac{-t}{\mu^2}\right) - \ln^2\left(\frac{-p_4^2}{\mu^2}\right) - \ln^2\left(\frac{-s}{-t}\right) - 2\,\mathrm{Li}_2\left(1 - \frac{(-p_4^2)}{(-s)}\right) \\
& \left. - 2\,\mathrm{Li}_2\left(1 - \frac{(-p_4^2)}{(-t)}\right) - 3\zeta(2)\right] + \mathcal{O}(\epsilon)
\end{aligned}
\tag{B.12}
$$



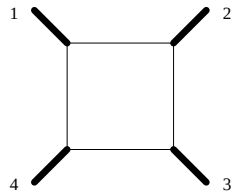
$$
\begin{aligned}
I_4^{2me} = & -\frac{2}{\epsilon\left(st - p_2^2 p_4^2\right)}\left[\ln\left(\frac{-s}{\mu^2}\right) + \ln\left(\frac{-t}{\mu^2}\right) - \ln\left(\frac{-p_2^2}{\mu^2}\right) - \ln\left(\frac{-p_4^2}{\mu^2}\right)\right] \\
& + \frac{1}{st - p_2^2 p_4^2}\left[\ln^2\left(\frac{-s}{\mu^2}\right) + \ln^2\left(\frac{-t}{\mu^2}\right) - \ln^2\left(\frac{-p_2^2}{\mu^2}\right) - \ln^2\left(\frac{-p_4^2}{\mu^2}\right)\right. \\
& - \ln^2\left(\frac{-s}{-t}\right) - 2\,\mathrm{Li}_2\left(1 - \frac{(-p_2^2)}{(-s)}\right) - 2\,\mathrm{Li}_2\left(1 - \frac{(-p_2^2)}{(-t)}\right) \\
& - 2\,\mathrm{Li}_2\left(1 - \frac{(-p_4^2)}{(-s)}\right) - 2\,\mathrm{Li}_2\left(1 - \frac{(-p_4^2)}{(-t)}\right) + 2\,\mathrm{Li}_2\left(1 - \frac{(-p_2^2)}{(-s)}\frac{(-p_4^2)}{(-t)}\right)\Big] \\
& + \mathcal{O}(\epsilon)
\end{aligned}
\tag{B.13}
$$



$$
\begin{aligned}
I_4^{2mh} = & \frac{1}{\epsilon^2 st} - \frac{1}{\epsilon st}\left[\ln\left(\frac{-s}{\mu^2}\right) + 2\ln\left(\frac{-t}{\mu^2}\right) - \ln\left(\frac{-p_3^2}{\mu^2}\right) - \ln\left(\frac{-p_4^2}{\mu^2}\right)\right] \\
& + \frac{1}{st}\left[\frac{3}{2}\ln^2\left(\frac{-s}{\mu^2}\right) + \ln^2\left(\frac{-t}{\mu^2}\right) - \frac{1}{2}\ln^2\left(\frac{-p_3^2}{\mu^2}\right) - \frac{1}{2}\ln^2\left(\frac{-p_4^2}{\mu^2}\right) - \ln^2\left(\frac{-s}{-t}\right)\right. \\
& - \ln\left(\frac{-s}{\mu^2}\right)\ln\left(\frac{-p_3^2}{\mu^2}\right) - \ln\left(\frac{-s}{\mu^2}\right)\ln\left(\frac{-p_4^2}{\mu^2}\right) + \ln\left(\frac{-p_3^2}{\mu^2}\right)\ln\left(\frac{-p_4^2}{\mu^2}\right) \\
& \left. - 2\,\mathrm{Li}_2\left(1 - \frac{(-p_3^2)}{(-t)}\right) - 2\,\mathrm{Li}_2\left(1 - \frac{(-p_4^2)}{(-t)}\right) - \frac{1}{2}\zeta(2)\right] + \mathcal{O}(\epsilon)
\end{aligned}
\tag{B.14}
$$



$$
\begin{aligned}
I_4^{3m} = & -\frac{1}{\epsilon\left(st - p_2^2 p_4^2\right)}\left[\ln\left(\frac{-s}{\mu^2}\right) + \ln\left(\frac{-t}{\mu^2}\right) - \ln\left(\frac{-p_2^2}{\mu^2}\right) - \ln\left(\frac{-p_4^2}{\mu^2}\right)\right] \\
& + \frac{1}{st - p_2^2 p_4^2}\left[\frac{3}{2}\ln^2\left(\frac{-s}{\mu^2}\right) + \frac{3}{2}\ln^2\left(\frac{-t}{\mu^2}\right) - \frac{1}{2}\ln^2\left(\frac{-p_2^2}{\mu^2}\right) - \frac{1}{2}\ln^2\left(\frac{-p_4^2}{\mu^2}\right)\right. \\
& - \ln^2\left(\frac{-s}{-t}\right) - \ln\left(\frac{-s}{\mu^2}\right)\ln\left(\frac{-p_3^2}{\mu^2}\right) - \ln\left(\frac{-s}{\mu^2}\right)\ln\left(\frac{-p_4^2}{\mu^2}\right) \\
& + \ln\left(\frac{-p_3^2}{\mu^2}\right)\ln\left(\frac{-p_4^2}{\mu^2}\right) - \ln\left(\frac{-t}{\mu^2}\right)\ln\left(\frac{-p_2^2}{\mu^2}\right) - \ln\left(\frac{-t}{\mu^2}\right)\ln\left(\frac{-p_3^2}{\mu^2}\right) \\
& + \ln\left(\frac{-p_2^2}{\mu^2}\right)\ln\left(\frac{-p_3^2}{\mu^2}\right) - 2\,\mathrm{Li}_2\left(1 - \frac{(-p_2^2)}{(-s)}\right) - 2\,\mathrm{Li}_2\left(1 - \frac{(-p_4^2)}{(-t)}\right) \\
& \left. + 2\,\mathrm{Li}_2\left(1 - \frac{(-p_2^2)}{(-s)}\frac{(-p_4^2)}{(-t)}\right)\right] + \mathcal{O}(\epsilon)
\end{aligned}
\tag{B.15}
$$



$$
I_4^{4m}\left(s,t,p_2^2,p_3^2,p_4^2,\mu^2\right) = I_3^{3m}(st,p_1^2p_3^2,p_2^2p_4^2,\mu^2) + K(s,t,p_1^2,p_3^2,p_2^2,p_4^2) \tag{B.16}
$$

mit

$$K(\alpha_1,\beta_1,\alpha_2,\beta_2,\alpha_3,\beta_3) = \frac{-2i\pi}{\lambda}\sum_{i=1}^{3}\theta(-\alpha_i)\theta(-\beta_i) \times\Big[\ln\Big(\sum_{j\neq i}\alpha_j\beta_j-(\alpha_i\beta_i-\lambda)(1+i0)\Big) -\ln\Big(\sum_{j\neq i}\alpha_j\beta_j-(\alpha_i\beta_i+\lambda)(1-i0)\Big)\Big] \tag{B.17}$$

C GiNaC

GiNaC ist eine C++-Bibliothek für Computeralgebra [39]. GiNaC wurde in Mainz entwickelt, um *Maple* im Rahmen des xloops-Projekts zu ersetzen. Die Version 1.0 wurde im Jahr 2000 fertiggestellt, momentan ist die Version 1.3.2 mit inzwischen erheblich erweiterten Fähigkeiten aktuell.

GiNaC ist eine (selbstzitierende) Abkürzung und steht für *GiNaC Is Not A CAS*. CAS ist die Abkürzung für *Computer Algebra System* und wird für Systeme wie zum Beispiel *Mathematica*, *Maple* oder *Reduce* verwendet. Im Gegensatz zu diesen vollintegrierten Systemen mit hochentwickelten Benutzerschnittstellen ist GiNaC eine C++-Bibliothek, deren Fähigkeiten man nur durch das Schreiben eigener C++-Programme nutzen kann.

C++ bietet die Möglichkeit durch *Klassen* neue Datentypen, zusätzlich zu den eingebauten wie zum Beispiel `int` oder `double`, zu definieren. Ein wesentlicher Beitrag von GiNaC ist die Bereitstellung von vielen neuen Datentypen, vor allem solcher zur Speicherung beliebiger algebraischer Ausdrücke. Die beiden wichtigsten sind `symbol` und `ex`. Mit `symbol` lassen sich unbestimmte mathematische Variablen definieren:

```
symbol x("x");
symbol epsilon("\\epsilon");
```

Ein zusätzlicher Name für die Bildschirmausgabe kann als Parameter bei der Definition angegeben werden. Im zweiten Fall wird das *TeX*-Format benutzt. Mit diesen Unbekannten lassen sich nun beliebige symbolische Ausdrücke formulieren. Diese können in Variablen vom Typ `ex` gespeichert werden:

```
ex result = 3.12 * x / sin(x) + epsilon;
```

C++ erlaubt die Erweiterung der vorhandenen Operationen wie '+', '−', '∗', ... auf eigene Datentypen („Überladen von Operatoren“), so dass sich Formeln sehr natürlich schreiben lassen. GiNaC nutzt diese Möglichkeit.

GiNaC kann mit komplexen Zahlen und exakten rationalen Zahlen umgehen. Bei Fließkommazahlen verwendet GiNaC Arithmetik mit beliebiger Präzision[1]. Man kann die Präzision durch die Angabe der Anzahl von signifikanten Dezimalstellen jederzeit ändern. Die meisten grundlegenden mathematischen Funktionen sind in GiNaC vordefiniert und können sowohl algebraisch manipuliert als auch numerisch ausgewertet werden. Zusätzlich existieren eine Reihe von speziellen Funktionen, die in der Physik oft benötigt werden, z.B. die Polylogarithmen. Darüber hinaus bietet GiNaC eine Vielzahl von algebraischen Operationen an, wie z.B. das Expandieren oder Zusammenfassen von Ausdrücken, die Ableitung und Reihenentwicklung, oder das Suchen und Ersetzen von Teilausdrücken, etc.

[1] GiNaC verwendet für die Numerik die C++-Bibliothek *cln* [64].

Ein kleines Beispielprogramm soll die Verwendung von GiNaC illustrieren. Es rechnet die Hermite-Polynome rekursiv aus und gibt das Ergebnis am Bildschirm aus:

```
 1  #include <iostream>
 2  using namespace std;
 3  #include <ginac/ginac.h>
 4  using namespace GiNaC;
 5
 6  ex HermitePoly(const symbol& x, int n)
 7  {
 8      ex HKer = exp(-pow(x, 2));
 9      return normal(pow(-1, n) * diff(HKer, x, n) / HKer);
10  }
11
12  int main(int argc, char* argv[])
13  {
14      if (argc != 2) return 1;
15      int i = atoi(argv[1]);
16
17      symbol z("z");
18      cout << "H_" << i << "(z) == " << HermitePoly(z, i) << endl;
19
20      return 0;
21  }
```

In den Zeilen 3 und 4 wird GiNaC eingebunden. In den Zeilen 6, 8 und 17 sieht man die Verwendung der Datentypen `ex` und `symbol`. In den Zeilen 8-9 werden eine Reihe von Operationen ausgeführt, deren Bedeutung sich vielleicht auch ohne genaue Kenntnis von GiNaC erraten lässt.

GiNaC wird intensiv weiterentwickelt, inzwischen von mehreren Programmierern aus der ganzen Welt. Meistens sind die Anwender von GiNaC auch gleichzeitig Entwickler. GiNaC ist Open-Source und gut dokumentiert, so dass nicht vorhandene Fähigkeiten oder Funktionen leicht in GiNaC eingebaut werden können. Auch abseits der Hochenergiephysik wird GiNaC vermehrt eingesetzt. Weitere Informationen zu GiNaC finden sich auf der Homepage

http://www.ginac.de

Abbildungsverzeichnis

Literaturverzeichnis

[1] J. C. Collins, *Renormalization*, Cambridge University Press, 1984.

[2] P. Cvitanovic, B. Lautrup, R. B. Pearson, *The Number And Weights Of Feynman Diagrams*, Phys. Rev. D **18** (1978) 1939.

[3] S. Heinemeyer, *Loop calculations: Summary*, [hep-ph/0408269].

[4] A. V. Semenov, *LanHEP: A package for automatic generation of Feynman rules in gauge models*, [hep-ph/9608488].

[5] T. Kaneko, S. Kawabata, Y. Shimizu, *Automatic Generation Of Feynman Graphs And Amplitudes In QED*, Comput. Phys. Commun. **43** (1987) 279.

[6] J. Küblbeck, M. Böhm, A. Denner, *Feyn Arts: Computer Algebraic Generation Of Feynman Graphs And Amplitudes*, Comput. Phys. Commun. **60** (1990) 165.

[7] P. Nogueira, *Automatic Feynman graph generation*, J. Comput. Phys. **105** (1993) 279.

[8] A. Denner, H. Eck, O. Hahn, J. Küblbeck, *Compact Feynman rules for Majorana fermions*, Phys. Lett. B **291** (1992) 278.
A. Denner, H. Eck, O. Hahn, J. Küblbeck, *Feynman rules for fermion number violating interactions*, Nucl. Phys. B **387** (1992) 467.

[9] T. Ishikawa, T. Kaneko, K. Kato, S. Kawabata, Y. Shimizu, H. Tanaka *GRACE manual: Automatic generation of tree amplitudes in Standard Models: Version 1.0*, KEK-92-19.

[10] K. G. Chetyrkin, F. V. Tkachov, *Integration By Parts: The Algorithm To Calculate Beta Functions In 4 Loops*, Nucl. Phys. B **192** (1981) 159.

[11] S. Laporta, *High-precision calculation of multi-loop Feynman integrals by difference equations*, Int. J. Mod. Phys. A **15** (2000) 5087 [hep-ph/0102033].

[12] C. Anastasiou, A. Lazopoulos, *Automatic integral reduction for higher order perturbative calculations*, JHEP **0407** (2004) 046 [hep-ph/0404258].

[13] G. Passarino, M. J. G. Veltman, *One Loop Corrections For E+ E- Annihilation Into Mu+ Mu- In The Weinberg Model*, Nucl. Phys. B **160** (1979) 151.

[14] T. Binoth, J. P. Guillet, G. Heinrich, E. Pilon, C. Schubert, *An algebraic / numerical formalism for one-loop multi-leg amplitudes*, [hep-ph/0504267].

[15] R. Gastmans, T. T. Wu, *The Ubiquitous Photon - Helicity Method for QED and QCD*, Oxford University Press, 1990.

[16] S. Dittmaier, *Weyl-van-der-Waerden formalism for helicity amplitudes of massive particles*, Phys. Rev. D **59** (1999) 016007 [hep-ph/9805445].

[17] R. Pittau, *A simple method for multi-leg loop calculations*, Comput. Phys. Commun. **104** (1997) 23 [arXiv:hep-ph/9607309].

[18] M. L. Mangano, S. J. Parke, *Multiparton Amplitudes In Gauge Theories*, Phys. Rept. **200** (1991) 301.

[19] L. J. Dixon, *Calculating scattering amplitudes efficiently*, [arXiv:hep-ph/9601359].

[20] Z. Bern, L. J. Dixon, D. A. Kosower, *Bootstrapping Multi-Parton Loop Amplitudes in QCD*, [arXiv:hep-ph/0507005].

[21] R. Pittau, *A simple method for multi-leg loop calculations. II: A general algorithm*, Comput. Phys. Commun. **111** (1998) 48 [arXiv:hep-ph/9712418].

[22] S. Weinzierl, *Reduction of multi-leg loop integrals*, Phys. Lett. B **450** (1999) 234 [arXiv:hep-ph/9811365].

[23] Z. Bern, D. A. Kosower, *The Computation of loop amplitudes in gauge theories*, Nucl. Phys. B **379** (1992) 451.

[24] Z. Bern, A. G. Morgan, *Massive Loop Amplitudes from Unitarity*, Nucl. Phys. B **467** (1996) 479 [arXiv:hep-ph/9511336].

[25] G. Duplancic, B. Nizic, *Dimensionally regulated one-loop box scalar integrals with massless internal lines*, Eur. Phys. J. C **20** (2001) 357 [arXiv:hep-ph/0006249].

[26] G. Duplancic, B. Nizic, *IR finite one-loop box scalar integral with massless internal lines*, Eur. Phys. J. C **24** (2002) 385 [arXiv:hep-ph/0201306].

[27] A. van Hameren, J. Vollinga, S. Weinzierl, *Automated computation of one-loop integrals in massless theories*, Eur. Phys. J. C **41** (2005) 361 [arXiv:hep-ph/0502165].

[28] D. B. Melrose, *Reduction Of Feynman Diagrams*, Nuovo Cim. **40** (1965) 181.

[29] S. Dittmaier, *Separation of soft and collinear singularities from one-loop N-point integrals*, Nucl. Phys. B **675** (2003) 447 [hep-ph/0308246].

[30] G. Duplancic, B. Nizic, *Reduction method for dimensionally regulated one-loop N-point Feynman integrals*, Eur. Phys. J. C **35** (2004) 105 [arXiv:hep-ph/0303184].

[31] W. T. Giele, E. W. N. Glover, *A calculational formalism for one-loop integrals*, JHEP **0404** (2004) 29 [arXiv:hep-ph/0402152].

[32] Z. Bern, L. J. Dixon, D. A. Kosower, *Dimensionally regulated pentagon integrals*, Nucl. Phys. B **412** (1994) 751 [arXiv:hep-ph/9306240].

[33] T. Binoth, J. P. Guillet, G. Heinrich, *Reduction formalism for dimensionally regulated one-loop N-point integrals*, Nucl. Phys. B **572** (2000) 361 [arXiv:hep-ph/9911342].

[34] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, *Numerical Recipes in C*, Cambridge University Press, Cambridge, 1992.

[35] G. J. van Oldenborgh, *FF: A Package To Evaluate One Loop Feynman Diagrams*, Comput. Phys. Commun. **66** (1991) 1.

[36] D. Kreimer, *The master two-loop two-point function – The general case*, Phys. Lett. B **273** (1991) 277.

[37] D. Kreimer, *Dimensional Regularization in the Standard Model*, Dissertation, Universität Mainz, 1992.

[38] L. Brücher, J. Franzkowski, A. Frink, D. Kreimer, *Introduction to XLOOPS*, Nucl. Instrum. Meth. A **389** (1997) 323 [hep-ph/9611378].

[39] C. Bauer, A. Frink, R. Kreckel, J. Vollinga, *GiNaC – An open framework for symbolic computation within the C++ programming language*, `http://www.ginac.de/tutorial`, 2005.

[40] S. Weinzierl, *Symbolic expansion of transcendental functions*, Comput. Phys. Commun. **145** (2002) 357 [math-ph/0201011].

[41] G. P. Lepage, *A New Algorithm for Adaptive Multidimensional Integration*, J. Comput. Phys. **27** (1978) 192.

[42] H. S. Do, *Feynman loop integrals and their automatic computer-aided evaluation*, Dissertation, Universität Mainz, 2003.

[43] M. Knodel, *Zweischleifenkorrekturen zum leptonischen Zerfall des Z-Bosons*, Dissertation, Universität Mainz, 2005.

[44] R. Kreckel, *Algorithmische Methoden zur Berechnung von Vierbeinfunktionen*, Dissertation, Universität Mainz, 2002.

[45] U. Seul, *Erstellung eines Computerprogramms zur automatischen Erzeugung von Feynman-Graphen*, Diplomarbeit, Universität Mainz, 2003.

[46] B. C. Carlson, *Special functions of applied Mathematics*, Academic Press, New York, 1977.

[47] S. Moch, P. Uwer, S. Weinzierl, *Nested sums, expansion of transcendental functions and multi-scale multi-loop integrals*, J. Math. Phys. **43** (2002) 3363 [hep-ph/0110083].

[48] A. B. Goncharov, *Multiple polylogarithms, cyclotomy and modular complexes*, Math. Res. Lett. **5** (1998) 497, (available at http://www.math.uiuc.edu/K-theory/0297).

[49] E. Remiddi, J. A. M. Vermaseren, *Harmonic Polylogarithms*, Int. J. Mod. Phys. **A15** (2000) 725 [hep-ph/9905237].

[50] N. Nielsen, *Der Eulersche Dilogarithmus und seine Verallgemeinerungen*, Nova Acta Leopoldina (Halle) **90** (1909) 121.

[51] H. N. Minh, M. Petitot, *Lyndon words, polylogarithms and the Riemann ζ function*, Discrete Math. **217** (2000) 273

[52] S. Weinzierl, *Expansion around half-integer values, binomial sums and inverse binomial sums*, J. Math. Phys. **45** (2004) 2656 [hep-ph/0402131].

[53] L. Lewin, *Dilogarithms and Associated Functions*, MacMillan, London 1958.

[54] J. Vollinga, S. Weinzierl, *Numerical evaluation of multiple polylogarithms*, Comput. Phys. Comm. **167** (2005) 177 [arXiv:hep-ph/0410259].

[55] J. M. Borwein, D. M. Bradley, D. J. Broadhurst, P. Lisonek, *Special Values of Multiple Polylogarithms*, Trans. Amer. Math. Soc. **353:3** (2001) 907 [math.CA/9910045].

[56] K. S. Kölbig, J. A. Mignaco, E. Remiddi, *On Nielsen's Generalized Polylogarithms and Their Numerical Calculation*, BIT **10** (1970) 38.

[57] K. S. Kölbig, *Nielsen's Generalized Polylogarithms*, SIAM J. Math. Anal. **17** (1986) 1232.

[58] L. Lewin, *Polylogarithms and Associated Functions*, North Holland, Amsterdam, 1981.

[59] T. Gehrmann, E. Remiddi, *Numerical Evaluation of Harmonic Polylogarithms*, Comput. Phys. Commun. **141** (2001) 296 [hep-ph/0107173].

[60] D. Maitre, *HPL, a Mathematica implementation of the harmonic polylogarithm*, [hep-ph/0507152].

[61] G. L. Steele, *Common Lisp the Language*, Digital Press, Woburn, Massachusetts, 1990.

[62] J. M. Borwein, D. M. Bradley, D. J. Broadhurst, *Evaluations of k-fold Euler/Zagier sums: a compendium of results for arbitrary k*, Electron. J. Combin. **4** (1997) R5 [hep-th/9611004].

[63] R. E. Crandall, *Fast evaluation of multiple zeta sums*, Math. Comp. **67** (1998) 1163.

[64] B. Haible, R. Kreckel, *CLN, a Class Library for Numbers* (Version 1.1.10), URL: http://www.ginac.de/CLN/.