Linear polynomial reduction for Feynman integrals

MASTERTHESIS

for attainment of the academic degree of
Master of Science
(M. Sc.)

submitted at
Mathematisch-Naturwissenschaftliche Fakultät I
Institut für Physik
Humboldt-Universität zu Berlin

by
Martin Lüders
born on 02.08.1988 in Potsdam

Advisor:

1. *Prof. Dr. Dirk Kreimer*
2. *Dr. Christian Bogner*

submitted on: *9th September 2013*

Contents

Abstract

This thesis deals with an integration algorithm for Feynman integrals using hyperlogarithms and it focus on classifying the class of Feynman graphs where we can use this algorithm. Therefor we prove a graph theoretical property which can be used to classify this class. We also determine the class explicitly for a subset of the Feynman graphs.

Chapter 1

Introduction

Computations of processes in pertubative quantum field theory involve three steps. The first one is to determine the corresponding Feynman diagrams, the second step is the regularisation and renormalisation such that we obtain convergent integrals and the last step is the integration of the integrals. For the last step there are many approaches but none is suitable for all cases thus one always has to decide which approach to use. In 2008 an algorithm for the integration of Feynman integrals using hyperlogarithms was presented by Francis Brown. It has the advantage that the calculations can be done by a computer which enables us to handle more complicated integrals. Like the other algorithms, it can be only used for a share of the Feynman integrals, they are called linearly reducible, and we want to know which ones are linearly reducible. To determine if we can use the algorithm, we have to apply one of the three reduction algorithms by Francis Brown. They act on the level of polynomials appearing during the integration, such that the test ist much faster than simply trying to integrate. The three algorithms differ in the method of defining an upper bound of the appearing polynomials. This thesis will show that the set of linearly reducible graphs is minor closed which enables us to classify the set of linearly reducible graphs by a smaller set of graphs and we will also calculate these graphs for an example. The second chapter is about the fundamentals of calculating an Feynman integral in parametric form and the third one is a short introduction to the part of the graph theory needed. The next chapter contains the idea of the integration algorithm and three reduction algorithms to test whether the integration algorithm is useable in a given case. In the fifth chapter we prove a theorem which enables us to simplify the classification of the set of Feynman graphs where the algorithm is suitable. The sixth chapter will deal with the results of the examination of a class of Feynman graphs using the implemented reduction algorithms and the last chapter is a short presentation of my own

implementation with the aim to enable the reader to run their own tests.

Chapter 2

Fundamentals

2.1 Feynman integrals

From the free Lagrangian and the interaction Lagrangian we can derive the Feynman propagator and the coupling terms. In scalar theories the coupling terms are scalar factors with a delta function for momentum conservation. In other theories the coupling terms can involve tensor structures. Using the Feynman propagator we can derive the Feynman integral for a given Feynman graph.
For a scalar theory like ϕ^3- or ϕ^4-theory the Feynman integrand in the momentum space is a product of propagators and with a delta function for every vertex. The propagators depend on the momentum k and the mass m of the line and have the form $\frac{1}{k^2-m^2}$. Thus the integral reads as

$$\int d^4k_1 \int d^4k_2 \ldots \int d^4k_n \frac{1}{k_1^2 - m_1^2} \cdots \frac{1}{k_n^2 - m_n^2} \prod_v \delta\left(\sum k_{v_i}\right),$$

with the notation that k_{v_i} are the momentum entering vertex v respectively $-k_{v_i}$ if the momentum is leaving the vertex v.
The next step is the integration of some of the k_i such that the δ functions disappear. Without loss of generality we can assume that only $k_1, \ldots k_l$ are left. The other momenta are replaced by a linear combination q_i of these l momenta, thus the new Feynman integral reads as

$$\int d^4k_1 \int d^4k_2 \ldots \int d^4k_l \frac{1}{k_1^2 - m_1^2} \cdots \frac{1}{k_l^2 - m_l^2} \frac{1}{q_{l+1}^2 - m_{l+1}^2} \frac{1}{q_n^2 - m_n^2}.$$

This integration is over the momentum space but it can be useful to tranfer the calculation into Feynman parameters. This can be done by using the

Feynman trick based on the identity:

$$\frac{1}{A} = \int_0^\infty dx \exp(-Ax).$$

It holds if the integral is well defined and the real part of A is positive.
It can be extended to obtain the generalized Feynman identity which involves products $\frac{1}{\prod_{j=1}^n A_j^{\nu_j}}$. Usally we only need the simple case $\nu_j = 1$ for all j:

$$\frac{1}{\prod_{j=1}^n A_j} = (n-1)! \prod_{j=1}^n \left(\int_0^\infty d\alpha_j \right) \frac{\delta\left(1 - \sum_{j=1}^n \alpha_j\right)}{\left(\sum_{j=1}^n \alpha_j A_j\right)^n}. \tag{2.1}$$

This identity is useful because we transfer the product of Feynman propagators into a sum where the momentum integration can be done easily. The integration domain with respect to the Feynman parameters α_i is compact because the δ function in 2.1 enables us to exchange the upper bound to 1.
Using the generalized Feynman identity and integrating out the momenta and perform a few algebraic transformations (see [9, 8]) leads to the well known parametric Feynman integral:

$$I_G = \frac{\Gamma(\nu - LD/2)}{\prod_{j=1}^n \Gamma(\nu_j)} \int_{\alpha_j \geq 0} \delta\left(1 - \sum_{i=1}^n \alpha_i\right) \left(\prod_{j=1}^n d\alpha_j \alpha_j^{\nu_j - 1} \right) \frac{\varphi^{\nu-(L+1)D/2}}{\Psi^{\nu - LD/2}}. \tag{2.2}$$

Theories with tensor structure like QED also lead to integrals of the same form but with additional prefactors which do not influence the integration process. D is the dimension in the theory used. L is the loop number of the graph and ν_i is the exponent of the Feynman propagator of edge e_i. As stated before it is usally equal to one because for a physical graph we have $\nu_i = 1$ for all i, but there are identities between Feynman integrals which involve higher/lower exponents. Hence it will be useful to look at the more general case. The α_i are the Feynman parameters which will be our new integration parameters. The functions φ and Ψ are the first and second Symanzik polynomial and depend on the Feynman parameters and Ψ also depends on particle masses and kinematic invariants. The next few pages will deal with the calculations of them.
The convergence of integral 2.2 depends on the graph and the dimension of the theory used.
There are many approaches to obtain a convergent integral, for example dimensional regularization (see [2]) where we assume that the dimension is

decreased by ϵ such we obtain a convergent integral for $\epsilon > 0$ and look at the laurent expansion with respect to ϵ. Another approach is descriped by Dirk Kreimer and Francis Brown in [3] using the Hopf-algebra-structure of QFT. Another problem are the infrared divergences which should cancel out but the divergences may lead to problems during the calculation. The resulting integral will depend on the approach choosen but in general we obtain a new integrand which depends on powers of the graph polynomials and on the logarithm of graph polynomials.

2.2 Graph polynomials

First of all we have to define the terms spanning tree and spanning 2-tree:

Definition 1: For a given graph G we define the set of spanning trees $\mathcal{T}_1$ to be the set consisting of all subgraphs of G which are trees and contain all vertices of G. Hence if G is not connected, there is no spanning tree.

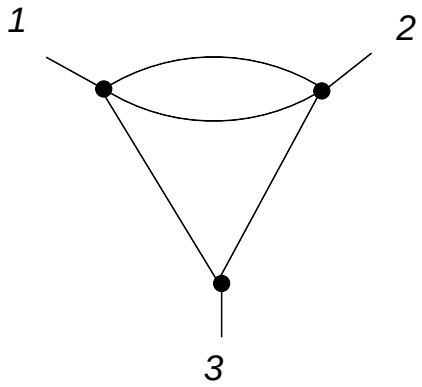


Figure 2.1: Graph G

Example. *In this example we choose G to be the graph in figure 2.1. From a combinatorical point of view it is clear that we need to choose two edges to get a spanning tree, because from Euler's Formula we know that the loop number is $L = E - V + 1$ and a tree has $L = 0$. This leads to $\binom{4}{2} = 6$ possible ways to select two out of four edges. But choosing the two parallel arcs will not lead to a spanning tree, because it does not contain vertex 3 and it is not even a tree because it contains a loop. Thus we obtain five spanning trees shown in figure 2.2 as thick lines.*

Remark. *The number of edges of a graph G we have to delete to obtain a spanning tree is always equal to the loop number of the graph G.*

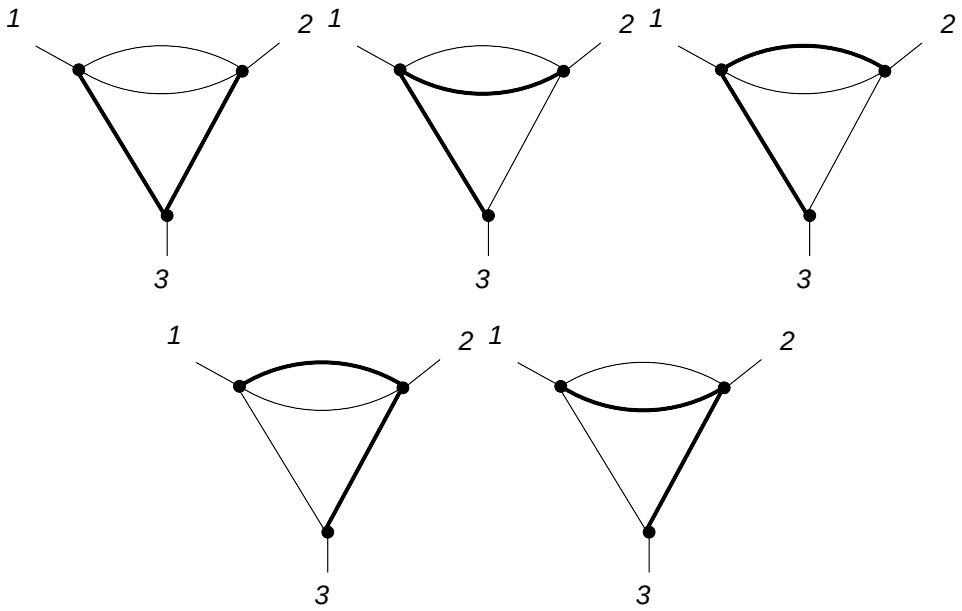


Figure 2.2: Spanning trees of G

Definition 2: For a given graph G we define the set of spanning 2-trees $\mathcal{T}_2$ to be the set consisting of all unordered pairs (T_1, T_2) of subgraphs of G which fulfill the conditions: T_1 and T_2 are trees, $T_1 \cap T_2 = \emptyset$ and $T_1 \cup T_2$ contains all vertices of G.

Example. *Using the graph G as in the example above, our 2-trees consist of a tree T_1 consisting of a single edge and another tree T_2 only consisting of a single vertex. All four edges of G are possible as T_1 thus we obtain four 2-trees for our graph G*

Remark. *As in the case of spanning trees, the 2-trees always consist of a fixed number of edges but we have to delete one more edge than in the case of spanning trees.*

Using these two definitions we define the Symanzik polynomials:

Definition 3: Let G be a graph with the spanning trees $\mathcal{T}_1$ and the spanning 2-trees $\mathcal{T}_2$ and P_T be the set of external momenta attached to T. Then

$$\varphi = \sum_{T \in \mathcal{T}_1} \prod_{e_i \notin T} \alpha_i,$$

$$\Psi = \sum_{(T_1,T_2) \in \mathcal{T}_2} \left(\prod_{e_i \notin T_1 \cup T_2} \alpha_i \right) \left(\sum_{p_j \in P_{T_1}} \sum_{p_k \in P_{T_2}} \frac{p_j \cdot p_k}{\mu^2} \right) + \varphi \sum_{i=1}^{n} \alpha_i \frac{m_i^2}{\mu^2}.$$

Hence if G has loop number L then φ is homogeneous of degree L and φ homogeneous of degree $L+1$ in the α-variables.

Remark. *If the graph G is not connected, then there are no spanning trees and the first Symanzik polynomial φ is zero.*

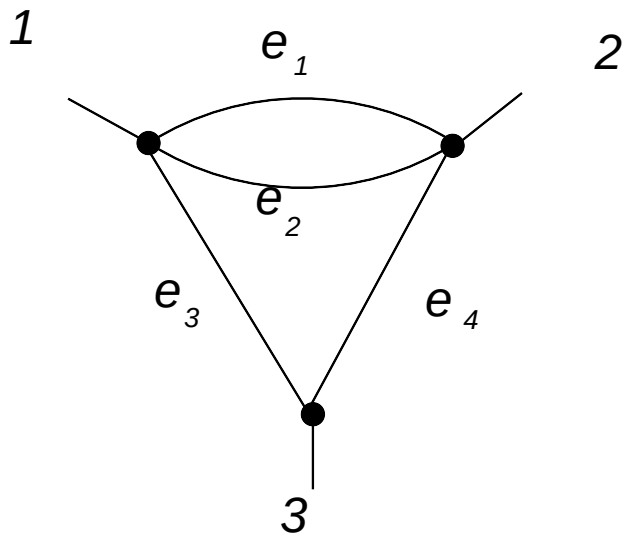


Figure 2.3: Graph G

Example. *Looking again at our graph G in figure 2.1 and choosing an ordering of the edges, for example the one in figure 2.3, we obtain:*

$$\varphi = \alpha_1\alpha_2 + \alpha_1\alpha_4 + \alpha_2\alpha_4 + \alpha_2\alpha_3 + \alpha_1\alpha_3.$$

Assume all internal edges have mass zero, then the second Symanzik polynomial is:

$$\Psi = \alpha_2\alpha_3\alpha_4(p_1+p_2)p_3+\alpha_1\alpha_3\alpha_4(p_1+p_2)p_3+\alpha_1\alpha_2\alpha_4(p_1+p_3)p_2+\alpha_1\alpha_2\alpha_3(p_2+p_3)p_1.$$

Because of momentum conservation we get $p_1+p_2+p_3=0$, hence we can rewrite Ψ to be:

$$\Psi = \alpha_2\alpha_3\alpha_4(-p_3^2) + \alpha_1\alpha_3\alpha_4(-p_3^2) + \alpha_1\alpha_2\alpha_4(-p_2^2) + \alpha_1\alpha_2\alpha_3(-p_1^2).$$

There are many other ways to calculate the Symanzik polynomials, most useful for implementation are matrix based approaches. The one used in my program is based on [4].
The calculation of the first Symanzik polynomial of a graph G using the approch in [4] is divided into four steps:

- Choose an arbitary but fixed ordering of the n vertices and the k edges in G and label the edges and vertices with the corresponding numbers.
- Create a quadratic $n \times n$-matrix M. For $1 \leq i \leq n$ set $M_{ii} = \sum_{j \in N_i} x_j$ where N_i is the set of indizes of edges in G which contain the vertex i. For $i \neq j$ set $M_{ij} = -\sum_{l \in N_i \cap N_j} x_l$. Note that $N_i \cap N_j$ is the set of edges between vertex i and j, hence M is a symmetric matrix.
- Delete row n and column n of M to obtain the new Matrix $\tilde{M}$ and calculate the determinant $f := \left|\tilde{M}\right|$.
- Substitute $x_i = \frac{1}{\alpha_i}$ in f to obtain $\tilde{f}$ and define

$$\varphi = \left(\prod_{i=1}^{k} \alpha_i\right) \tilde{f}.$$

Remark. *The determinant of $\tilde{M}$ does not change if we delete row i and column i for $1 \leq i \leq n$ instead of deleting row n and column n.*

A similar calculation can be used for the second Symanzik polynomial:

- Choose an arbitary but fixed ordering of the n vertices and the k edges in G and label the edges and vertices with the corresponding numbers.
- create a quadratic $n \times n$-matrix M. For $1 \leq i \leq n$ set $M_{ii} = \sum_{j \in N_i} x_j + \sum_{p_j \in P_i} p_j$ neglecting the vectorical character of p_i for a moment. Here N_i is the set of indizes of edges in G which contain the vertex i and P_i is the set of external momenta at vertex i. For $i \neq j$ set $M_{ij} = -\sum_{l \in N_i \cap N_j} x_l$. Note that $N_i \cap N_j$ is the set of edges between vertex i and j, hence M is a symmetric matrix.
- Calculate the determinant $f := |M|$ and define $p_i p_j$ to be the dot product of p_i and p_j, any other products of momenta do not matter because they will be removed in the next step.
- Define f' to be the polynomial f after removing all monomials which do not have a total degree of exactly two in the external momenta p_i.
- Substitute $x_i = \frac{1}{\alpha_i}$ in f' to obtain $\tilde{f}$ and define

$$\Psi_0 = \left(\prod_{i=1}^{k} \alpha_i\right) \tilde{f}.$$

- Let m_i be the internal mass of edge e_i. Define:

$$\Psi = \Psi_0 + \varphi \sum_{i=1}^{k} \alpha_i m_i^2.$$

Remark. *Usally it is useful to rewrite Ψ in Mandelstam variables to get rid of the dot product. If there are more than four external momenta you have to use a more general approach (see [7]) to define the maximal set of independend scalar values of the dot products.*

Example. *Let us again look at the graph G of figure 2.3 with the same ordering of edges. It has 3 vertices and 4 edges.*

- *We already chose the ordering of figure 2.3.*
- *The matrix M reads:*

$$M = \begin{bmatrix} x_1 + x_2 + x_3 & -x_1 - x_2 & -x_3 \\ -x_1 - x_2 & x_1 + x_2 + x_4 & -x_4 \\ -x_3 & -x_4 & x_3 + x_4 \end{bmatrix}.$$

- *The calculation of the minor $\tilde{M}$ and the function f yields:*

$$\begin{aligned} \tilde{M} &= \begin{bmatrix} x_1 + x_2 + x_3 & -x_1 - x_2 \\ -x_1 - x_2 & x_1 + x_2 + x_4 \end{bmatrix}, \\ f &= (x_1 + x_2 + x_3)(x_1 + x_2 + x_4) - (x_1 + x_2)^2 \\ &= (x_1 + x_2)(x_3 + x_4) + x_3 x_4. \end{aligned}$$

- *Substituting the x_i gives:*

$$\begin{aligned} \varphi &= \alpha_1\alpha_2\alpha_3\alpha_4 \left(\left(\frac{1}{\alpha_1} + \frac{1}{\alpha_2} \right) \left(\frac{1}{\alpha_3} + \frac{1}{\alpha_4} \right) + \frac{1}{\alpha_3\alpha_3} \right) \\ \varphi &= (\alpha_2 + \alpha_1)(\alpha_4 + \alpha_3) + \alpha_1\alpha_2 \\ &= \alpha_1\alpha_2 + \alpha_1\alpha_3 + \alpha_1\alpha_4 + \alpha_2\alpha_3 + \alpha_2\alpha_4. \end{aligned}$$

This is the same result as in the calculation using spanning trees. The way to obtain Ψ is similar:

- *We choose again the ordering of figure 2.3.*

- *The resulting matrix M is:*

$$M = \begin{bmatrix} x_1+x_2+x_3+p_1 & -x_1-x_2 & -x_3 \\ -x_1-x_2 & x_1+x_2+x_4+p_2 & -x_4 \\ -x_3 & -x_4 & x_3+x_4+p_3 \end{bmatrix}.$$

- *We do not need the summands linear in a single p_i or linear in $p_1p_2p_3$, hence we go straight to f':*

$$f' = p_1p_2(x_3+x_4) + p_1p_3(x_1+x_2+x_4) + p_2p_3(x_1+x_2+x_3).$$

- *Substituting the x_i lead to:*

$$\begin{aligned}
\Psi_0 &= \alpha_1\alpha_2\alpha_3\alpha_4 \left(\frac{p_1p_2}{\alpha_3} + \frac{p_1p_2}{\alpha_4} + \frac{p_1p_3}{\alpha_1} + \frac{p_1p_3}{\alpha_2} + \frac{p_1p_3}{\alpha_4} + \frac{p_2p_3}{\alpha_1} + \frac{p_2p_3}{\alpha_2} + \frac{p_2p_3}{\alpha_3} \right) \\
&= \alpha_1\alpha_2\alpha_3\alpha_4 \left(\frac{p_1p_3+p_2p_3}{\alpha_1} + \frac{p_1p_3+p_2p_3}{\alpha_2} + \frac{p_1p_2+p_2p_3}{\alpha_3} + \frac{p_1p_2+p_1p_3}{\alpha_4} \right) \\
&= \alpha_1\alpha_2\alpha_3\alpha_4 \left(\frac{-p_3^2}{\alpha_1} + \frac{-p_3^2}{\alpha_2} + \frac{-p_2^2}{\alpha_3} + \frac{-p_1^2}{\alpha_4} \right) \\
&= \alpha_2\alpha_3\alpha_4(-p_3^2) + \alpha_1\alpha_3\alpha_4(-p_3^2) + \alpha_1\alpha_2\alpha_4(-p_3^2) + \alpha_1\alpha_2\alpha_3(-p_3^2).
\end{aligned}$$

- *We use again $m_i = 0$ for the internal edges such that*

$$\Psi = \Psi_0.$$

There is also one identity between Symanzik polynomials of a graph G and the Symanzik polynomials of the graphs $G//e_k$ and $G\backslash e_k$ obtained by deleting or contracting one edge. For explizit definitions of $G//e$ and $G\backslash e$ see the next chapter.

Lemma 4. *For massless internal edges the following well known identities holds:*

$$\begin{aligned}
\varphi_G &= \varphi_{G\backslash e}\alpha_k + \varphi_{G//e} \\
\Psi_G &= \Psi_{G\backslash e_k}\alpha_k + \Psi_{G//e_k}.
\end{aligned}$$

Proof: Consider the spanning trees of G. Each tree T either contains the edge e_k or not. All spanning trees T not containing e_k are also spanning trees of $G\backslash e_k$ and all spanning trees of $G\backslash e_k$ are spanning trees of G and are not containing e_k. This introduces a bijection between the spanning trees of G not containing e_k and the spanning trees of $G\backslash e_k$. The similar holds for the

spanning trees of $G//e_k$ and the spanning trees of G containing e_k. $G//e_k$ is obtained by identifying the two vertices of the edge e_k. If we have a spanning tree T of G which contains e_k, we obtain a spanning tree of $G//e_k$ by the contraction of the edge e_k in T. For the other direction we have to add the edge e_k to the spanning trees of $G//e_k$ to connect the two vertices of e_k which are identified in $G//e_k$. Hence we get a bijection between the spanning trees on $G//e_k$ and the spanning trees on G which contain e_k. Using the defintion of the first Symanzik polnomial we obtain:

$$\begin{aligned}
\varphi_G &= \sum_{T\in\mathcal{T}_1} \prod_{e_i\notin T} \alpha_i \\
\varphi_G &= \sum_{T\in\mathcal{T}_1, e_k\in T} \prod_{e_i\notin T} \alpha_i + \sum_{T\in\mathcal{T}_1, e_k\notin T} \prod_{e_i\notin T} \alpha_i \\
\varphi_G &= \varphi_{G//e_k} + \alpha_k \sum_{T\in\mathcal{T}_1, e_k\notin T} \prod_{e_i\notin T, e_i\neq e_k} \alpha_i \\
\varphi_G &= \varphi_{G//e_k} + \alpha_k \varphi_{G\backslash e_k}.
\end{aligned}$$

The same can be done for the second Symanzik polynomials because we get the same correspondence between 2-trees of G and the 2-trees of $G//e$ and $G\backslash e$.

Remark. *Note that we need to be able to define $G//e_k$ in a proper way. If e_k is an edge which connects one vertex with itself a contraction in the sense introduced in the next chapter is not possible.*
Deletion of an edge such that the graph G is no longer connected leads to a trivial first Symanzik polynomial, but the formula is still valid.

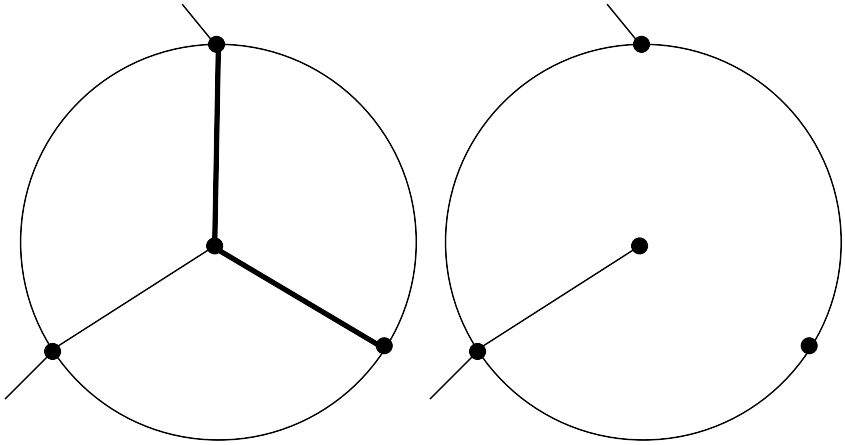
Chapter 3

Graph Theory

Feynman graphs usally consist of different kinds of edges which correspond to different particles such as photons, electrons or protons. Hence different edges lead to different Feynman integrals but after dealing with the tensor structure and the regularization only integrands which depend on the two Symanzik polynomials are left. Hence the algorithm is independent of the kind of edges thus we can simply choose a single edge kind, making complete tests of all n-loop graphs much easier because it decreases the amount of possible graphs. But this only enable us to decide if a graph is linearly reducible and not. For the integration process the result will depend on the edge kind used because the integrands will get different prefactors.
However, standard graphs from graph theory are not enough for our purpose because we have external momenta which are an additional information usally not considered in graph theory. Thus we are using a standard graph together with a list of vertices where external momenta enter the graph and we will adopt the formalism of minors of graphs to our extended class of graphs.

Definition 5: A deletion of an edge maps a graph onto another graph with the same vertex count and with edge count decreased by one and simply removes the specified edge.

Remark. *The list of vertices with external momenta does not change. If the deleted edge was not a bridge, the loop number decreases by 1. The new graph will be denoted by $G\backslash e_i$ where G is the original graph and e_i the edge which is removed. Therefore e_i must be contained in G. The result of removing several different edges does not depend on the order such that we do not have to specify this order, thus $(G\backslash e_i)\backslash e_j$ can be written as $G\backslash\{e_i, e_j\}$.*



(a) Graph before deleting thick edges

(b) The same graph after deleting thick edges

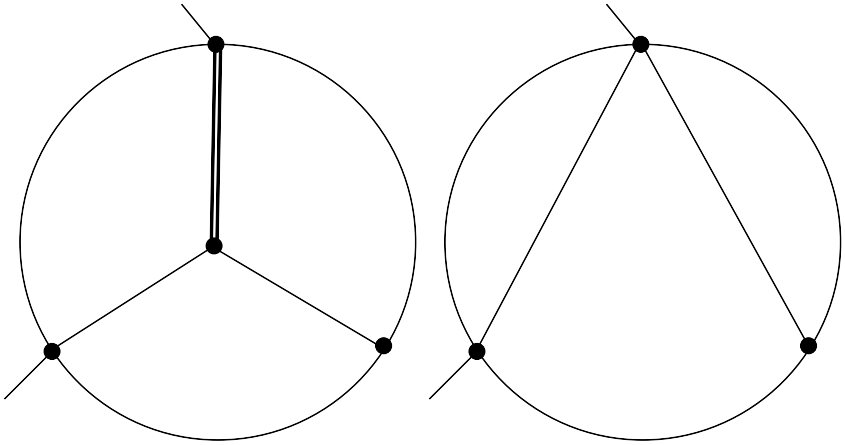
Figure 3.1: Example for deleting edges

Definition 6: A contraction of an edge e_i between vertex A and B maps a graph onto another graph with vertex count and edge count both decreased by one. It identifies the vertices A and B and removes the edge e_i.

Remark. *Hence the loop number does not change. The new graph will be denoted by $G//e_i$ where G is the original graph and e_i the edge which is contracted. Therefore e_i must be contained in G and furthermore we only allow edges where at most one of the corressponding vertices have an external momentum, thus we do not change the number vertices where external momenta enter. The result of removing several different edges does not depend on the order such that we do not have to specify this order, thus $(G//e_i)//e_j$ can be written as $G//\{e_i, e_j\}$. For an example see figure 3.2. Contraction of an edge connecting a vertex with itself would be the same as the deletion, because it decreases the loop number. We will always treat it as a deletion and define the contraction to be not valid.*

Remark. *Contracting and deleteting edges commute, such that we do not have to specify which one we do first.*

Definition 7: A graph G' is called a minor of a graph G if there are two distinct subsets A and B of edges of G such that $G' = G\backslash A//B$.



(a) Graph before contracting double lined edges

(b) The same graph after contracting edges

Figure 3.2: Example for contracting edges

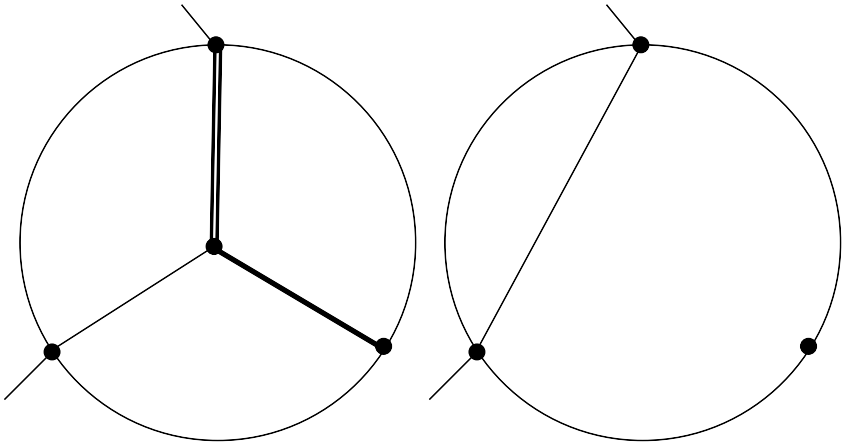


Figure 3.3: A graph and one of his minors

Example. *The right graph in figure 3.3 is a minor of the left one because it is obtained by contracting the double lined edge and deleting the thick edge.*

Remark. *Being a minor introduces a partial order on the set of graphs. Hence in a few cases we can say that graph G_1 is smaller than G_2 because G_1 is contained in G_2 as a minor. And we can conclude that if G_1 is smaller than G_2 and G_2 is smaller than G_3 then also G_1 is smaller than G_3 holds. Later we will see that if we can do the integration for a graph G, we can transfer the set of appearing polynomials to all graphs smaller than G.*

Definition 8: A set A of graphs is called minor closed if for every graph $G \in A$ every minor G' of G is an element of the set A.

Definition 9: Let A be a minor closed set. A graph G is called a critical minor if $G \notin A$ but every minor $G' \neq G$ of G is an element of A.

Example. *The set of planar graphs is minor closed because deleting edges of a planar graph always results in a planar graph. The same holds for contracting edges of a planar graph. The famous graph theoretical Robertson-Seymour theorem [12] tells us, that every minor closed set has a finite number of critical minors. In case of planar graphs it is known as Wagner's conjecture [13]. The critical minors are only the K_5 and the $K_{3,3}$ shown in figure 3.4. Hence every non-planar graph contains one of them as a minor. The Robertson-Seymour theorem holds for standard graphs and it is uncertain if it also holds for our extended graphs with external momenta.*

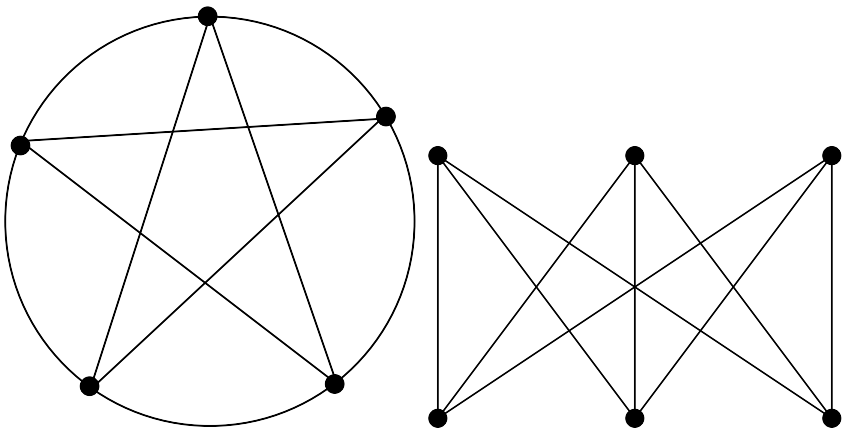


Figure 3.4: Critical minors of the planar graphs

Chapter 4

Reduction Algorithms

In this chapter we will introduce parts of the integration algorithm by Francis Brown, published in his article [5]. We only give an incomplete review of the integration procedure because we want to focus on the criteria which tells us if the integration will succeed. But the knowledge of the integration algorithm is useful to understand the reduction algorithm of testing if an expression is integrable.

4.1 Hyperlogarithms

The integration is based on the use of hyperlogarithms [14], but I will only give the definition from the paper [5] without describing the useful hints how to calculate with them. For a more detailed view about these functions I want to refer to [10].

Definition 10: Let $\Sigma = \{\sigma_0, \sigma_1, \ldots, \sigma_N\}$ be a set of distinct points of $\mathbb{C}$ and let us assume $\sigma_0 = 0$. Create an alphabet $A = \{a_0, \ldots a_N\}$ where each symbol a_i corresponds to the point σ_i. Denote A^x the set of all words in A including the empty word e. Let $\mathbb{Q}\langle A\rangle$ be the vector space generated by all words w in A^x. To each word w we assign a hyperlogarithm function:

$$L_w(z) : \mathbb{C}\backslash\Sigma \to \mathbb{C},$$

which is multivalued and let $log(z)$ denote the principal branch of the logarithm.

The hyperlogarithm is uniquely determined by the following three properties:

- $L_e(z) = 1$ and $L_{a_0^n}(z) = \frac{1}{n!}\log^n(z)$ for all $n \geq 1$.

- For all words $w \in A^x$ and $0 \leq i \leq N$:

$$\frac{\partial}{\partial z} L_{a_i w}(z) = \frac{1}{z - \sigma_i} L_w(z) \text{ for } z \in \mathbb{C} \backslash \Sigma.$$

- For all words $w \in A^x$ not of the form $w = a_0^n$:

$$\lim_{z \to 0} L_w(z) = 0.$$

Remark. *The three properties define $L_w(z)$ inductively over the weight of $L_w(z)$ which is the number of letters in w. $L_{a_i w}(z)$ is uniquely defined by $L_w(z)$ and the known constant of integration from the last property.*

4.2 The integration procedure

In our case the roots σ_i are rational functions of the Feynman parameters with arbitary prefactors in the Mandelstam variables and internal and external masses.
For the first cases we assume that the roots for the letters a_i do not depend on the next integration variable.
From the definition of the hyperlogarithm function we obtain

$$\int \frac{dz}{z - \sigma_i} L_w(z) = L_{a_i w}.$$

Using partial integration we can also integrate terms like:

$$\frac{1}{(z - \sigma_i)^n} L_w(z).$$

Each step of the partial integration reduces the weight of the hyperlogarithm function by one, thus the integration will succeed in at most k steps where k is the weight of $L_w(z)$.
We can also handle terms like $\frac{1}{f(z)} L_w(z)$ where $f(z)$ can be decomposed into linear factors. For two factors it can be done as follows:

$$\begin{aligned} \int \frac{dz}{(z - \sigma_i)(z - \sigma_j)} L_w(z) &= \int \left(\frac{1}{z - \sigma_i} - \frac{1}{z - \sigma_j} \right) \frac{1}{\sigma_i - \sigma_j} L_w(z) dz \\ &= \frac{1}{\sigma_i - \sigma_j} \left(L_{a_i w} - L_{a_j w} \right). \end{aligned}$$

The partial fraction decomposition leads to the new fraction $\frac{1}{\sigma_i - \sigma_j}$. In the case of more than two factors, we know that we still get a sum of fractions

with denominators linear in z or a power of a linear polynomial. But how does the additional fractions in the partial decomposition look? The answer is that we only get fractions of the form $\frac{1}{\sigma_i - \sigma_j}$ where σ_i and σ_j are roots of f. This can be proven via induction over the number of roots. Let $\{\sigma_k\}_{k=1}^n$ be the roots of f with $\sigma_1 \neq \sigma_n$ (otherwise we are in the previous case) and the polynomial has a leading coefficent of 1 then:

$$\begin{aligned}\frac{1}{f} &= \frac{1}{\prod_{k=1}^n (z - \sigma_k)} \\ &= \frac{1}{\sigma_n - \sigma_1} \left(\frac{1}{\prod_{k=2}^n (z - \sigma_k)} - \frac{1}{\prod_{k=1}^{n-1} (z - \sigma_k)} \right).\end{aligned}$$

Hence after this step we obtain two new denominators with degree decreased by one and new prefactors of the form $\frac{1}{\sigma_i - \sigma_j}$.
We omitted the part which deals with the problem of regularisation of the hyperlogs. It is a technical feature which is not needed for the understanding of the further thesis.
The hard part is that we obtain a hyperlogarithm where the σ_k may depend on the next integration variable. To handle this problem we have to replace the σ_k such that they no longer depend on the next integration parameter. If the σ_k depend in a well behaved manner on the next integration variable the replacement is possible. The condition is that the numerator and the denominator of σ_k factors into linear terms in the next integration variable and also for $\sigma_k - \sigma_i$ and for differences of roots where one appears as an argument of the hyperlogarithm and the other is contained in the rational prefactor of the hyperlogarithm. Then we can transform the implicit dependence into an explicit dependence.

This sketch gives us an idea that we have to focus on the σ_i in each step to decide if the integration can be done. It does not matter whether we look at the roots σ_i of a polynomial or the corresponding polynomial. At the beginning we have a given set of polynomials appearing in the integrand, which would be φ and Ψ in case of Feynman integrals. From this we want to calculate the appearing polynomials in the next steps. In each step we know that if all polynomials factor into linear terms in the next integration variable we can do the next integration. Let S_k be the set of all factors of the polynomials after integrating over $\alpha_1 \dots \alpha_k$. Our condition tells us that all elements have to be linear in the next integration variable α_{k+1}, but we need to know how the set will change in this step. Lets write each polyonmial $f_i \in S_k$ as $f_i = g_i \alpha_{k+1} + h_i$. It is obvious that we obtain the linear part g_i of

each polynomial and the absolute part h_i because they appear as prefactors in the next step. Furthermore we investigated the fact that we get additional prefactors from polyonmial decomposition. The roots σ_i correspond to f_i via $\sigma_i = \frac{h_i}{g_i}$ hence

$$\sigma_i - \sigma_j = \frac{h_i g_j - h_j g_i}{g_i g_j}.$$

Hence we also get polynomials of the form $h_i g_j - h_j g_i$ in our next step. Keeping this in mind we will recall three algorithms of polyonmial calculations to decide if the integration algorithm will succeed or not.

4.3 Simple reduction algorithm

The Simple reduction algorithm by Francis Brown was first presented in [5] and simply does the polynomial analysis for a given ordering of integration variables.
Let us assume a set $A = \{f_1, \ldots, f_N\}$ where all polynomials f_i are linear in α_r. Then they can be written as $f_i = g_i \alpha_r + h_i$. Note that g_i and h_i are independend of α_r. The algorithm constists of two steps:

- We define an intermediate set $\tilde{A}_{(r)}$ via:

$$\tilde{A}_{(r)} = \{(g_i)_{1 \leq i \leq N}, (h_i)_{1 \leq i \leq N}, (g_i h_j - g_j h_i)_{1 \leq i < j \leq N}\}.$$

- The new set $A_{(r)}$ is the set of all irreducible factors of the polynomials in $\tilde{A}_{(r)}$.

The new set $A_{(r)}$ is independent of α_r and describes an upper bound of the appearing polynomials after integrating α_r. This has to be done for each step of the integration. Assume we have choosen an order of integration $r_1, \ldots r_n$ we use the step above to calculate $S_{(r_1)}$ from the starting set S. This set we use to calculate $S_{(r_1)(r_2)} =: S_{(r_1,r_2)}$ and so on leading to a sequence of sets $S, S_{(r_1)}, S_{(r_1,r_2)}, \ldots S_{(r_1,\ldots r_n)}$. To obtain these sets we have to choose an order of integration such that every polynomial in $S_{(r_1,\ldots r_i)}$ is linear in $\alpha_{r_{i+1}}$ for all i from 0 to $n-1$. If such an ordering exists, we call the set S to be simply reducible.

Example. *We will demonstrate the Simple reduction algorithm on the graph shown in figure 4.1. Let us assume that edge* 1 *und* 2 *are massless and edge* 3 *has mass* M. *Let us further assume that* $p_3^2 = 0$ *and* $p_1^2 = m_1^2 \neq 0$, $p_2^2 = m_2^2 \neq 0$. *Hence we get:*

$$\varphi = \alpha_1 + \alpha_2 + \alpha_3 \tag{4.1}$$

$$\Psi = -m_2^2 \alpha_1 \alpha_2 - m_1^2 \alpha_1 \alpha_3 - M^2 \alpha_3 (\alpha_1 + \alpha_2 + \alpha_3). \tag{4.2}$$

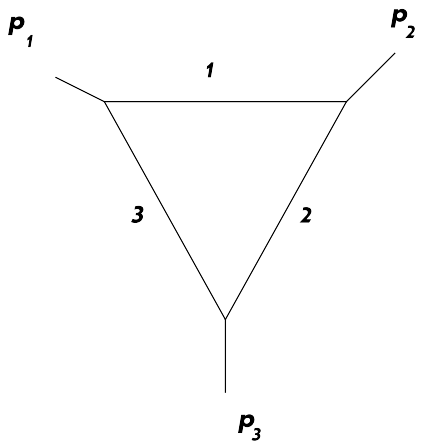


Figure 4.1: Example for the reduction algorithms

Our starting set is $S = \{\varphi, \Psi\}$. Ψ *is not linear in* α_3 *hence we can not start the reduction with* α_3.
Reduction for calculation of $S_{(1)}$*:*

- *The first set we have to calculate is* $\tilde{S}_{(1)}$*:*

$$\begin{aligned}\tilde{S}_{(1)} = \{&1, -m_2^2\alpha_2 - m_1^2\alpha_3 - M^2\alpha_3, \alpha_2 + \alpha_3, -M^2\alpha_3\left(\alpha_2 + \alpha_3\right), \\ &- M^2\alpha_3\left(\alpha_2 + \alpha_3\right) - \left(-m_2^2\alpha_2 - m_1^2\alpha_3 - M^2\alpha_3\right)\left(\alpha_2 + \alpha_3\right)\}.\end{aligned}$$

- *We can only factor the three polynomials*

$$-m_2^2\alpha_2 - m_1^2\alpha_3 - M^2\alpha_3 = (-1)\left(m_2^2\alpha_2 + m_1^2\alpha_3 + M^2\alpha_3\right)$$

$$-M^2\alpha_3\left(\alpha_2 + \alpha_3\right) = (-1)(M)^2(\alpha_3)(\alpha_2 + \alpha_3)$$

and

$$\begin{aligned}&-M^2\alpha_3\left(\alpha_2 + \alpha_3\right) + \left(m_2^2\alpha_2 + m_1^2\alpha_3 + M^2\alpha_3\right)\left(\alpha_2 + \alpha_3\right) \\ &\qquad = \left(m_2^2\alpha_2 + m_1^2\alpha_3\right)\left(\alpha_2 + \alpha_3\right).\end{aligned}$$

Hence we obtain:

$$S_{(1)} = \{1, -1, \alpha_2 + \alpha_3, M, \alpha_3, m_1^2\alpha_2 + m_2^2\alpha_3 + M^2\alpha_3, m_2^2\alpha_2 + m_1^2\alpha_3\}.$$

The same can be done to obtain $S_{(2)}$.

- *First step is the calculation of* $\tilde{S}_{(2)}$*:*

$$\begin{aligned}\tilde{S}_{(2)} = \{&1, -m_2^2\alpha_1 - M^2\alpha_3, \alpha_1 + \alpha_3, -m_1^2\alpha_1\alpha_3 - M^2\alpha_3\left(\alpha_1 + \alpha_3\right), \\ &- m_1^2\alpha_1\alpha_3 - M^2\alpha_3\left(\alpha_1 + \alpha_3\right) - \left(-m_2^2\alpha_1 - M^2\alpha_3\right)\left(\alpha_1 + \alpha_3\right)\}.\end{aligned}$$

- *We can only factor the two polynomials*

$$-m_2^2\alpha_1 - M^2\alpha_3 = (-1)\left(m_2^2\alpha_1 + M^2\alpha_3\right)$$

and

$$-m_1^2\alpha_1\alpha_3 - M^2\alpha_3\left(\alpha_1+\alpha_3\right) + \left(m_2^2\alpha_1 + M^2\alpha_3\right)\left(\alpha_1+\alpha_3\right) = \alpha_1\left(-m_1^2\alpha_3 + m_2^2\alpha_1 + m_2^2\alpha_3\right).$$

Hence we obtain:

$$S_{(2)} = \{1, -1, \alpha_1+\alpha_3, \alpha_1, m_2^2\alpha_1 + M^2\alpha_3, -m_1^2\alpha_3 + m_2^2\alpha_1 + m_2^2\alpha_3\}.$$

As already stated, $S_{(3)}$ cannot be calculated, but $S_{(1)}$ and $S_{(2)}$ are linear in every Feynman paramenter thus we can derive $S_{(1,2)}, S_{(1,3)}, S_{(2,1)}, S_{(2,3)}$. The trivial terms like $1, -1, m_1, m_2, M, \alpha_1, \alpha_3$ do not create new critical polynomials because their linear and absolut part are $0, 1$ or the polynomial itself and one part is equal to zero thus $g_ih_j - g_jh_i$ degenerates to g_ih_j which will factor into the factors of g_i and h_j. Using this observation we only have to consider the nontrivial cases of $g_ih_j - g_jh_i$.

- *For this reduction step we have to calculate $\tilde{S}_{(1,2)}$:*

$$\tilde{S}_{(1,2)} = \{1, -1, 0, M, \alpha_3, m_1^2\alpha_3, m_2^2, m_1^2\alpha_3 + M^2\alpha_3, \left(\alpha_3 m_2^2\right) - \left(m_1^2\alpha_3 + M^2\alpha_3\right)\}$$

- *We get 4 polynomials which can be factorized:*

$$m_2^2 = (m_2)^2$$

$$m_1^2\alpha_3 = (m_1)^2\alpha_3$$

$$m_1^2\alpha_3 + M^2\alpha_3 = (\alpha_3)\left(m_1^2 + M^2\right)$$

$$\left(\alpha_3 m_2^2\right) - \left(m_1^2\alpha_3 + M^2\alpha_3\right) = (\alpha_3)\left(m_2^2 - m_1^2 + M^2\right).$$

Hence we obtain:

$$S_{(1,2)} = \{1, -1, 0, M, \alpha_3, m_1, m_2, m_1^2 + M^2, m_2^2 - m_1^2 + M^2\}.$$

Hence we can now calculate $S_{(1,2,3)}$ and our graph is simply reducible with the ordering $1, 2, 3$. But we can also calculate $S_{(2,1)}$.

- *Calculation of $\tilde{S}_{(2,1)}$ yields:*

$$\tilde{S}_{(2,1)} = \{1, -1, 0, \alpha_3, m_2^2, M^2\alpha_3, -m_1^2\alpha_3 + m_2^2\alpha_3, \alpha_3 m_2^2 - M^2\alpha_3, \alpha_3 m_2^2 - \left(-m_1^2\alpha_3 + m_2^2\alpha_3\right), M^2\alpha_3 m_2^2 - m_2^2\left(-m_1^2\alpha_3 + m_2^2\alpha_3\right)\}$$

- *We get a few polynomials which can be factorized:*

$$m_2^2 = (m_2)^2$$

$$M^2\alpha_3 = (M)^2\alpha_3$$

$$-m_1^2\alpha_3 + m_2^2\alpha_3 = \alpha_3\,(m_1 + m_2)\,(m_2 - m_1)$$

$$\alpha_3 m_2^2 - M^2\alpha_3 = \alpha_3\,(m_2 + M)\,(m_2 - M)$$

$$\alpha_3 m_2^2 - \left(-m_1^2\alpha_3 + m_2^2\alpha_3\right) = \alpha_3(m_1)^2$$

$$M^2\alpha_3 m_2^2 - m_2^2\left(-m_1^2\alpha_3 + m_2^2\alpha_3\right) = \alpha_3(m_2)^2\left(M^2 + m_1^2 - m_2^2\right).$$

Hence we obtain:

$$S_{(2,1)} = \{1, -1, 0, \alpha_3, m_2, M, m_1^2{+}m_2^2, m_2{-}M, m_2{+}M, m_1, M^2{+}m_1^2{-}m_2^2\}.$$

Thus the graph is also simply reducible with the ordering $2, 1, 3$. *The orderings* $1, 3, 2$ *and* $2, 3, 1$ *succeed too.*

4.4 Fubini reduction algorithm

Francis Brown also described an enhanced version of the previous algorithm which leads to a better bound of the singularities. It takes the Fubini theorem into account, telling us that the result of an integration does not depend on the order of integration. Hence the singularities after integrating over α_{r_1} and α_{r_2} are contained in $S_{(r_1,r_2)}$ but also in $S_{(r_2,r_1)}$. Thus they must be contained in $S_{(r_1,r_2)} \cap S_{(r_2,r_1)}$. This new set is called $S_{[r_1,r_2]}$ and can be used to calculate $S_{[r_1,r_2](r_3)}$ with the same two steps as in the simple reduction algorithm. Fubini's theorem holds for more variables, hence we define:

$$S_{[r_1,\ldots,r_k]} = \bigcap_{i=1}^{k} S_{[r_1,\ldots,\hat{r_i},\ldots,r_k](r_i)}.$$

If some of the $S_{[r_1,\ldots,\hat{r_i},\ldots,r_k](r_i)}$ cannot be calculated because $S_{[r_1,\ldots,\hat{r_i},\ldots,r_k]}$ is not linear in α_{r_i}, then this set is ommited from the intersection. If this holds for all sets, then $S_{[r_1,\ldots,r_k]}$ cannot be calculated and the algorithm fails with the choosen ordering.
Similiar to the simple reduction algorithm we call S to be linearly reducible if there exists an ordering $r_1, \ldots r_n$ such that $S_{[r_1,\ldots r_i]}$ is linear in $\alpha_{r_{i+1}}$ for all i from 0 to $n-1$.

Example. *Let us again look at our graph 4.1 and the two corresponding Symanzik polynomials in 4.1 and 4.2. The Fubini reduction algorithm is the same for the calculation of $S_{(1)}$ and $S_{(2)}$, hence the results will be the same too. But in the next step we can see the advantage of the Fubini reduction algorithm, because if we want to obtain $S_{[1,2]}$ we have to calculate $S_{(1,2)}$ and $S_{(2,1)}$ and take the intersection. From the previous section we know:*

$$S_{(1,2)} = \{1, -1, 0, M, \alpha_3, m_1, m_2, m_1^2 + M^2, m_2^2 - m_1^2 + M^2\}$$

$$S_{(2,1)} = \{1, -1, 0, \alpha_3, m_2, M, m_1^2 + m_2^2, m_2 - M, m_2 + M, m_1, M^2 + m_1^2 - m_2^2\}.$$

Thus we obtain:

$$S_{[1,2]} = \{1, -1, 0, m_1, m_2, M, \alpha_3\}.$$

The reduction succeeds too but we have less critical polynomials. It can also happen for other graphs, that they are not simply reducible but from taking the Fubini theorem into account we can discard a few polynomials which might enable us to continue the reduction thus they can be linearly reducible.

4.5 Improved Fubini reduction algorithm

Further analysis by Francis Brown showed that not all polynomials from the previous algorithm will appear, he detected that a few elements of $(g_i h_j - g_j h_i)_{1 \leq i < j \leq N}$ will cancel out each other. Hence we only have to consider a smaller subset of pairs (i, j). This improvement is descriped in [6] and tells us, that we only have to consider compatible pairs of polynomials. Which polynomials are compatible will be defined in the reduction algorithm. We change the Fubini algorithm slightly by assigning to every set of polynomials a graph with a vertex per polynomial and an edge between two vertices if the polynomials are compatible.
We start again with the same set S and the compatibility graph is defined to be the complete graph. As before we calculate $S_{[r_1,\ldots,r_k](r_{k+1})}$ out of $S_{[r_1,\ldots,r_k]}$ if all polynomials are linear in $\alpha_{r_{k+1}}$. Every calculation consists of 4 steps. Let us say the starting set of this calculation is A which contains N polynomials and therefore a compatiblity graph with N vertices defining which polynomials are compatible.

- Create a new set $\tilde{A}_{(r)}$ via:

  $$\tilde{A}_{(r)} = \{(g_i)_{1 \leq i \leq N}, (h_i)_{1 \leq i \leq N}, (g_i h_j - g_j h_i)_{(i,j)\text{compatible}}\}.$$

  A pair (i, j) is compatible if the polynomials f_i and f_j are compatible by definition, that is if the compatibility graph contains an edge between the vertices corresponding to f_i and f_j.

- Assign to every polynomial in $\tilde{A}$ a pair of indizes (i,j) which characterizes how it was created. These are their parents. If it was created as the linear part g_i of f_i the indizes are (i,∞), for the absolute part h_i of f_i the indizes are $(i,0)$ and for $g_i h_j - g_j h_i$ the pair is (i,j). If a polynomial in $\tilde{A}$ is created multiple times, assign all pairs of indizes to it. A last special case is if f_i was independet of α_r, then $h_i = f_i$ and therefore we assign $(i,0)$ but additionally we assign the pair (i,∞).

- Define $A_{(r)}$ to be the set of all irreducible factors of the polynomials in $\tilde{A}_{(r)}$, keeping the pair of indizes of the polynomials from $\tilde{A}_{(r)}$. If we get the same factor from two different polynomials keep both pairs.

- Define a new compatibility graph by defining $a, b \in A_{(r)}$ to be compatible if their pairs of indizies contain at least one common index

As in the Fubini algorithm we take the intersection of different ways. On the level of the polynomials it is the same:

$$S_{[r_1,\ldots,r_k]} = \bigcap_{i=1}^{k} S_{[r_1,\ldots,\hat{r_i},\ldots,r_k](r_i)}.$$

On the level of compatibility graphs we define two polynomials in $S_{[r_1,\ldots,r_k]}$ to be compatible if they are compatible in the sets $S_{[r_1,\ldots,\hat{r_i},\ldots,r_k](r_i)}$ for all $1 \leq i \leq k$. We call a starting set S weakly linearly reducible if we find an ordering of edges such that $S_{[r_1,\ldots,r_k]}$ is linear in $\alpha_{r_{k+1}}$ for all $0 \leq k \leq n-1$.

Remark. *This algorithm enables us to get an even better bound for the critical polynomials. In our example from the graph in 4.1 nothing will change because the first step will be done with the complete graph. Hence the first reduction step of the improved Fubini algorithm is the same as in the Fubini algorithm. The second step could be already different but in our case this does not happen.*

Example. *To obtain a difference between the improved Fubini algorithm and the Fubini algorithm, we have to change our graph. An easy example is the box with 4 external momenta which fulfills the on-shell-condition and one massive internal line shown in figure 4.2.*
The two Symanzik polynomials are:

$$\begin{aligned}\varphi &= \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 \\ \Psi &= \varphi M^2 \alpha_1 + s\alpha_1\alpha_4 + t\alpha_1\alpha_4 - s\alpha_2\alpha_3.\end{aligned}$$

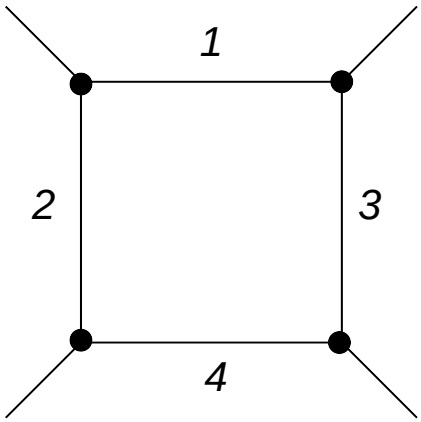


Figure 4.2: Example for the improved Fubini reduction algorithm

We want to calculate $S_{[1,2]}$. *The first observation is that* Ψ *is not linear in* α_1. *Hence we have to start with the calculation of* $S_{(2)}$. *Thus we obtain the set of polynomials:*

$$\tilde{S}_{(2)} = \{1, \alpha_1+\alpha_3+\alpha_4, M^2\alpha_1(\alpha_1+\alpha_3+\alpha_4)+s\alpha_1\alpha_4+t\alpha_1\alpha_4, M^2\alpha_1-s\alpha_3,$$
$$(\alpha_1+\alpha_3+\alpha_4)\left(M^2\alpha_1-s\alpha_3\right)-\left(M^2\alpha_1(\alpha_1+\alpha_3+\alpha_4)+s\alpha_1\alpha_4+t\alpha_1\alpha_4\right)\}.$$

Note that compatibilities between polynomials independent of all α_i *and any other polynomial does not matter, because one obtains* $f_i = 0$ *for one of the polynomials hence the term* $f_ih_j - f_jh_i$ *becomes* f_jh_i *which factors into* f_j *and* h_i *and these factors are always contained in the new set. We also lose no compatibilities in the next step, because* $f_i = 0$ *always leads to the linear part of the other polynomial. Hence if two polnomials are calculated from the i-th polynomial, both are the linear part of a polynomial and therefore already compatible.*
Let φ *be the first polynomial and* Ψ *the second polynomial, then the parents of the critical polynomials are:*

$$\begin{aligned} \alpha_1+\alpha_3+\alpha_4 &\quad (0,1)\\ M^2\alpha_1(\alpha_1+\alpha_3+\alpha_4)+s\alpha_1\alpha_4+t\alpha_1\alpha_4 &\quad (0,2)\\ M^2\alpha_1-s\alpha_3 &\quad (\infty,2)\\ (\alpha_1+\alpha_3+\alpha_4)(-s\alpha_3)-(s\alpha_1\alpha_4+t\alpha_1\alpha_4) &\quad (1,2). \end{aligned}$$

We can only factor the second polynomial and obtain α_1 and $M^2(\alpha_1+\alpha_3+\alpha_4)+s\alpha_4+t\alpha_4$ which still have the same parents $(0,2)$. Now we have to define the new compatibilities between the five new polynomials. Because we only have 4 possible parents $(0,\infty,1,2)$ the only ways that two

polynomials have no common parent is, that one is from $(0,1)$ and the other from $(\infty,2)$ or one from $(0,2)$ and the other from $(\infty,1)$. The third case $(0,\infty)$ and $(1,2)$ is not possible because $(0,\infty)$ cannot appear. In our case we conclude that all pairs of polynomials except for $(\alpha_1+\alpha_3+\alpha_4, M^2\alpha_1-s\alpha_3)$ are compatible. Using the Fubini reduction we would have gotten the same set $S_{(2)}$ but in the next step we would get one more polynomial in the Fubini algorithm from the pair $(\alpha_1+\alpha_3+\alpha_4, M^2\alpha_1 - s\alpha_3)$.
The next step is to calculate $S_{(2,1)}$. First we have to prove that all polynomials are linear in α_1, which is true.

$$\begin{aligned}\tilde{S}_{(2,1)} = \{&1, 0, \alpha_3+\alpha_4, M^2, M^2(\alpha_3+\alpha_4)+s\alpha_4+t\alpha_4, -s\alpha_3, (-s\alpha_3)(\alpha_3+\alpha_4),\\ &-s\alpha_3-s\alpha_4-t\alpha_4, (\alpha_3+\alpha_4)(M^2)-(M^2(\alpha_3+\alpha_4)+s\alpha_4+t\alpha_4),\\ &M^2(M^2(\alpha_3+\alpha_4)+s\alpha_4+t\alpha_4)+(s\alpha_3)(M^2),\\ &(\alpha_3+\alpha_4)(-s\alpha_3-s\alpha_4-t\alpha_4)-(-s\alpha_3)(\alpha_3+\alpha_4),\\ &M^2((-s\alpha_3)(\alpha_3+\alpha_4))-(M^2(\alpha_3+\alpha_4)+s\alpha_4+t\alpha_4)(-s\alpha_3),\\ &M^2((-s\alpha_3)(\alpha_3+\alpha_4))-(-s\alpha_3)(-s\alpha_3-s\alpha_4-t\alpha_4)\}.\end{aligned}$$

Factorisation leads to

$$\begin{aligned}S_{(2,1)} = \{&1, 0, \alpha_3+\alpha_4, M^2, M^2(\alpha_3+\alpha_4)+s\alpha_4+t\alpha_4, \alpha_3, s, -1,\\ &-s\alpha_3-s\alpha_4-t\alpha_4, (s+t), \alpha_4, (M^2(\alpha_3+\alpha_4)+s\alpha_4+t\alpha_4)+(s\alpha_3),\\ &M^2((\alpha_3+\alpha_4))-(-s\alpha_3-s\alpha_4-t\alpha_4)\}\\ &= \{1, 0, -1, M^2, s+t, \alpha_3, \alpha_4, \alpha_3+\alpha_4, M^2(\alpha_3+\alpha_4)+s\alpha_4+t\alpha_4,\\ &-s\alpha_3-s\alpha_4-t\alpha_4, M^2(\alpha_3+\alpha_4)+s\alpha_4+t\alpha_4+s\alpha_3\}.\end{aligned}$$

The polynomials appeared several times from different parents such that it is very hard to get the compatibilities for the next step, but we do not need them, because we can already see, that the improved Fubini algorithm is better than the Fubini algorithm, because the Fubini algorithm also contains the factors of $(\alpha_3+\alpha_4)M^2+s\alpha_3$ which is not contained in $S_{(2,1)}$. $S_{(1,2)}$ cannot be calculated thus we obtain $S_{[1,2]} = S_{(2,1)}$. Further calculation of these polynomials show that the graph is weakly linearly reducible but it is also linearly reducible even if intermediate steps contain more critical polynomials. But there are also cases with G weakly linearly reducible but not linearly reducible. But one cannot deal with them without a computer.

Remark. *It is obvious that every simply reducible set S is linearly reducible and every linearly reducible set is weakly linearly reducible.*

Chapter 5

Theoretical analysis

The previous chapter showed us a tool for deciding if we can integrate the Feynman integral with the method of [5]. But it has to be done for every graph. In this section we want to make an investigation how we can transfer our knowlegde about a graph to other graphs. This will enable us in many cases to decide if a graph is linearly reducible without doing the reduction process. Therefore we will prove the following theorem 11.

Theorem 11. *The set of linearly reducible graphs with a fixed number of external edges and without internal masses is minor-closed.*

Remark. *That means if we have found a graph G which is not linearly reducible, all graphs which contain G as a minor are not linearly reducible too. On the other hand if we have found a graph G which is linearly reducible, all minors of G are linearly reducible too.*

For the proof we will need a few definitions and lemmas first.

Definition 12: To each polynomial f in the n variables α_i, $i = 1 \dots n$ we assign a corresponding projective polynomial $\bar{f}$ in the $2n$ variables x_i, y_i, $i = 1 \dots n$ by:

$$\bar{f}(x_1, y_1, \dots, x_n, y_n) = \prod_i y_i^{n_i} \cdot f\left(\frac{x_1}{y_1}, \dots, \frac{x_n}{y_n}\right),$$

where n_i is the degree of f in α_i. Hence $y_i^{n_i}$ will cancel out all y_i in the denominator of $f\left(\frac{x_1}{y_1}, \dots, \frac{x_n}{y_n}\right)$.

Remark. *Choosing a greater n_i does not change the result because we will factorize the polynomials in each step such that the additional factor y_i does not matter.*

Remark. *Using the above definition we introduce the projective graph polynomials $\bar{\Psi}, \bar{\varphi}$. Without internal masses Ψ and φ are at most linear in all Feynman parameters α_i such that $\bar{\Psi}, \bar{\varphi}$ is at most linear in the x_i and y_i. There are only two cases where Ψ is independent of a α_k. First case is a graph which contains a brigde with edge e_k, hence e_k is contained in every spanning tree. And the other case is a graph which is not connected such that $\Psi = 0$.*

Lemma 13. *To eliminate special cases we will use the polynomials $\tilde{\Psi}, \tilde{\varphi}$. They are defined in the same way as the projective polynomials but with fixed $n_i = 1$ for all i. Thus we do not have to take care if a deletion or a contraction changes the powers n_i by creation of a brigde. Note that we obtain $\bar{\Psi}, \bar{\varphi}$ by factorizing $\tilde{\Psi}, \tilde{\varphi}$ because they can only contain additional factors y_i*
Let $\tilde{\Psi}_G$,$\tilde{\Psi}_{G//e_i}$ and $\tilde{\Psi}_{G\backslash e_i}$ be the polynomials for $G, G//e_i$ and $G\backslash e_i$. Then

$$\tilde{\Psi}_{G//e_i} = \tilde{\Psi}_G\Big|_{x_i=0,y_i=1}, \tag{5.1}$$

$$\tilde{\Psi}_{G\backslash e_i} = \tilde{\Psi}_G\Big|_{x_i=1,y_i=0}. \tag{5.2}$$

Proof: Remember the contraction-deletion formula in lemma 4:

$$\Psi_G = \Psi_{G\backslash e_k}\alpha_k + \Psi_{G//e_k}.$$

In the case of these polynomials it will transform to

$$\tilde{\Psi}_G = \tilde{\Psi}_{G\backslash e_k}x_k + \tilde{\Psi}_{G//e_k}y_k$$

because $\Psi_{G\backslash e_k}\alpha_k$ transforms into

$$\prod_{i=1}^{n} y_i \cdot \Psi_{G\backslash e_k}(\frac{x_1}{y_1}, \ldots, \frac{\hat{x_k}}{y_k}, \ldots, \frac{x_1}{y_1}) \cdot \frac{x_k}{y_k} = y_k\bar{\Psi}_{G\backslash e_k} \cdot \frac{x_k}{y_k}.$$

Similar for $\Psi_{G//e_k}$:

$$\prod_{i=1}^{n} y_i \cdot \Psi_{G//e_k}(\frac{x_1}{y_1}, \ldots, \frac{\hat{x_k}}{y_k}, \ldots, \frac{x_n}{y_n}) = y_k\bar{\Psi}_{G//e_k}$$

Setting $x_k = 0, y_k = 1$ immediately proves 5.1.
Setting $x_k = 1, y_k = 0$ proves 5.2.

Corollary 14. *The lemma also holds for the second graph polynomial because it fulfills the same contraction-deletion formula if we have no internal masses.*

Lemma 15. *If the Fubini reduction algorithm succeeds for the graph polynomials Ψ, φ of a graph G, it also succeeds for the corresponding projective graph polyonmial $\bar{\Psi}, \bar{\varphi}$ if we treat the x_i like the α_i and set $y_i = 1$ after reducing the corresponding x_i.*

Proof: We will prove that the sets of irreducible factors for every step of the reduction of the projective graph polynomials $\bar{\Psi}, \bar{\varphi}$ can be obtained via creation of the corresponding projective polynomials in the reduction of the graph polynomials Ψ, φ.
In the case of the starting set $S_0 = \{\bar{\Psi}, \bar{\varphi}\}$ it holds by assumptions. This is the starting point of our induction, hence it is sufficent to show it for one reduction step.
By assumption the set A in the reduction of the projective graph polynomials consists of the polynomials $\{\bar{f}_1, \dots \bar{f}_N\}$ where $f_1 \dots f_N$ are the polynomials in the reduction with the graph polynomials. From the definition of projective polynomials we can conclude, that $\bar{f}$ is linear in x_i if and only if f is linear in α_i.
The elements of the intermediate sets $\tilde{A}$ in the projective case correspond to the ones in the nonprojective case because from $\bar{f}$ corresponds to f we can conclude that $\frac{\partial}{\partial x_i}\bar{f}\Big|_{y_i=1}$ corresponds to $\frac{\partial}{\partial \alpha_i} f$. The same holds for $\bar{f}\big|_{x_i=0,y_i=1}$ and $f|_{\alpha_i=0}$. The property also tranfers to products because the highest degree in each variable simply adds up. For sums it is slightly more difficult because we can decrease the degree be adding a polynomials with the opposite leading coefficent. But this only leads to the case that $\bar{f}_1 + \bar{f}_2$ might be a multiple of $\overline{f_1 + f_2}$ by the factor of some y_i not causing any problems because they disappear in the factorization.
Hence we can say that the intermediate set $\tilde{A}$ in both cases corresponds to each other.
We still need to show that we can transfer the factorisations from the level of polynomials in Feynman parameter α_i to factorisations of projective polynomials in x_i, y_i. Assume that f factors into $g \cdot h$. We need to show that:

$$\bar{f} = \bar{g} \cdot \bar{h}$$

$$\prod_i y_i^{n_i(f)} \cdot f(\frac{x_1}{y_1}, \dots, \frac{x_n}{y_n}) = \prod_i y_i^{n_i(g)} \cdot g(\frac{x_1}{y_1}, \dots, \frac{x_n}{y_n}) \cdot \prod_i y_i^{n_i(h)} \cdot h(\frac{x_1}{y_1}, \dots, \frac{x_n}{y_n}).$$

This is true if and only if

$$\prod_i y_i^{n_i(f)} = \prod_i y_i^{n_i(g)} \cdot \prod_i y_i^{n_i(h)}$$

and this is true if and only if $n_i(f) = n_i(g) + n_i(h)$ for all i. But this is always true if $f = g \cdot h$ because if g is of degree $n_i(g)$ in α_i and h is of degree $n_i(h)$

in α_i then $g \cdot h$ is of degree $n_i(h) + n_i(g)$ in α_i.
This proves the fact that for every graph G which is linearly reducible the algorithm succeeds with the projective graph polynomials too, because in every step of the reduction we only have to exchange the polynomials by their corresponding ones.

Lemma 16. *The previous lemma also holds for the other direction and the proof is trivial, because*

$$\bar{\Psi}\big|_{y_1=y_2=\ldots=y_n=0} = \Psi(x_1, \ldots, x_n).$$

The same holds for φ.

Lemma 17. *If* G *is linearly reducible this also holds for* $G//e_k$ *and* $G\backslash e_k$.

Proof: By lemma 13 we know that we obtain $\bar{\Psi}_{G//e_k}, \bar{\varphi}_{G//e_k}$ by setting $x_k = 0, y_k = 1$ in $\bar{\Psi}_G, \bar{\varphi}_G$ and factor out additional factors y_i.
As long as we do not reduce edge k in the Fubini reduction algorithm of $G//e_k$, we can always choose the same factorisations as in the reduction for G and set $x_k = 0, y_k = 1$, because all functions are polynomials and setting variables to 1 or 0 will not change that fact. If we would reduce edge k in the reduction for G, we cannot do the same for $G//e_k$ because this edge is not contained. But we know that all polynomials are affine in x_k in this step of the Fubini reduction and therefore are affine in y_k too, because the power n_k of y_k was choosen such that n_k is the highest power of x_k.
Let $S = \{f_1, f_2, \ldots, f_n\}$ be the set before reducing e_k in the reduction of G. Then the new set contains irreducible factors of all polynomials of the form $f_i|_{x_k=0,y_k=1}$ and $\frac{\partial}{\partial x_k} f_i\Big|_{y_k=1}$.
But the set of $f_i|_{x_k=0,y_k=1}$ is the set of polynomials of the reduction of $G//e_k$ at the same step. Therefore we can continue the steps after this in the same way as before. Hence we can conclude $G//e_k$ is linearly reducible if G is linearly reducible.
In the case of $G\backslash e_k$ the proof is similar, only the step of reducing e_k we have to change slightly.
Using the fact that every summand either is containing x_k or y_k and f_i is affine in both variable we can rewrite:

$$\frac{\partial}{\partial x_k} f_i\Big|_{y_k=1} = f_i|_{x_k=1,y_k=0}\,.$$

Using the right side of this formula we obtain that $G\backslash e_k$ is linearly reducible if G is linearly reducible.

Corollary 18. *By induction: If G is linearly reducible then every minor of G is linearly reducible. This corollary proves theorem 11.*

Theorem 19. *The set of linearly reducible graphs with a fixed number of external edges and where each edge has attached either mass 0 or an independend value is minor-closed.*

The proof of this theorem is similar to the one before, but we only have the contraction-deletion-formula for the first graph polynomial. The second graph polynomial can be written as $\varphi = \varphi_0 + \varphi_1$, where φ_0 is the part independent of the masses, thus it fullfills the contraction-deletion-formula and $\varphi_1 = (\sum m_i^2 \alpha_i) \cdot \Psi$.
The lemma 13 is still valid for the first graph polynomial but for the second graph polynomial we have to change the proof.

Lemma 20. *As in lemma 13 we define $\tilde{\varphi}$ to eliminate special cases, but this time we set $n_i = 1$ if $m_i = 0$ and $n_i = 2$ otherwise.*
Choose an edge e_i where $m_i = 0$.
Let $\tilde{\varphi}_G$,$\tilde{\varphi}_{G//e_i}$ and $\tilde{\varphi}_{G\backslash e_i}$ be the polynomials for $G, G//e_i$ and $G\backslash e_i$. Then the same formula still holds:

$$\tilde{\varphi}_{G//e_i} = \tilde{\varphi}_G|_{x_i=0,y_i=1} \tag{5.3}$$
$$\tilde{\varphi}_{G\backslash e_i} = \tilde{\varphi}_G|_{x_i=1,y_i=0}\,. \tag{5.4}$$

Proof: From lemma 13 we already know that $\bar{\varphi}_0$ fullfills 5.3 and 5.4 because it does not depend on the mass setting.
For the nonprojective graph polynomial part φ_1 we obtain:

$$\varphi_{G//e_{i_1}} = \left(\sum_{j\neq i} m_j^2 \alpha_j\right) \cdot \Psi_{G//e_i} \tag{5.5}$$
$$= \left(\sum_{j} m_j^2 \alpha_j\right)\Bigg|_{m_i=0} \cdot \Psi_{G//e_i} \tag{5.6}$$
$$= \left(\sum_{j} m_j^2 \alpha_j\right) \cdot \Psi_{G//e_i}. \tag{5.7}$$

Note that φ_G is linear in α_i because φ_{G_0} is linear and from $m_i = 0$ we conclude that $\sum_j m_j^2 \alpha_j$ is independent of α_i and Ψ_G is linear in α_i. Thus if we look at the graph polynomial $\tilde{\varphi}$, the power of y_i is equal to 1. Setting $x_i = 0, y_1 = 1$ does not change the factor $\sum_j m_j^2 \alpha_j$ and we already know that

$\bar{\Psi}_G\big|_{x_i=0,y_i=1} = \bar{\Psi}_{G//e_i}$. Hence:

$$\tilde{\varphi}_{G//e_{i_1}} = \left(\sum_j m_j^2\alpha_j\right) \cdot \tilde{\Psi}_G\Big|_{x_i=0,y_i=1} \tag{5.8}$$

$$= \left(\left(\sum_j m_j^2\alpha_j\right) \cdot \tilde{\Psi}_G\right)\Bigg|_{x_i=0,y_i=1} \tag{5.9}$$

$$= \tilde{\varphi}_{G_1}|_{x_i=0,y_i=1}\,. \tag{5.10}$$

Similar formula holds for $\varphi_{G\backslash e_{i_1}}$ because $\tilde{\Psi}_G\Big|_{x_i=1,y_i=0} = \tilde{\Psi}_{G\backslash e_i}$. Hence

$$\tilde{\varphi}_{G//e_{i_1}} = \tilde{\varphi}_{G_1}|_{x_i=1,y_i=0}\,.$$

Together with the formula for φ_0 we obtain 5.3 and 5.4.

Lemma 15 also holds for the massive case because the proof can be transfered one to one. We will use it to show lemma 17 for the massive case:

Lemma 21. *If G is linearly reducible this also holds for $G//e_k$ and $G\backslash e_k$.*

Proof: First note, that if G is reducible and the mass m_k of edge e_k is arbitary, then G is linearly reducible too if we set $m_k = 0$. Hence it is sufficent to prove the lemma for edges with $m_k = 0$. Using lemma 20 creates the polynomials for the cases of the two minors. The following steps of the proof we can transfer one to one from lemma 17, because lemma 20 states the same property as lemma 13.

Corollary 22. *By induction: If G is linearly reducible then every minor of G is linearly reducible.*
This corollary proves theorem 19.

Remark. *This proof can be transfered from the fubini algorithm to the improved fubini algorithm without any problems because if two polynomials in the reduction of the minor are compatible the corresponding ones of the original graph are compatible too, hence we will keep our subset property.*

Chapter 6

Computional analysis

The theoretical part in the previous chapter gives us an idea how to classify the linearly reducible graphs but it does not give us any clue on the question for how many graphs this algorithm will succeed. Therefore I created a Maple implementation of the Fubini algorithm and the improved Fubini algorithm which enables us to do a case study for a large class of graphs.

6.1 Implementation

- **Fubini reduction**
  One of the implementations uses the Fubini algorithm (see section 4.4) to check if a given graph is linearly reducible. First of all it computes the two graph polynomials φ and Ψ using the matrix approach in section 2.2 and substitutes the external momenta by the Mandelstam variables. These two polynomials are used as starting set for the reduction.
  One of the problems in the implementation is the factorization because Maple does not have a built-in factorization algorithm which is useable for our case, because we only want the factors to be polynomials in the α_i but they may depend on arbitary algebraic functions in the Mandelstam variables. Hence it was nessesary to write an own algorithm which uses the roots of a polynomial to create the factorization. One case in the reduction, where we need the improved factorization is the polynomial $s\alpha_1^2 - t\alpha_2^2$ because every factorization involves roots of the Mandelstam variables s und t.
  The main problem resulting from the manualy implemented factorisation is that we cannot expect a unique factorization, because factors only depending on the Mandelstam variables can be arbitarily dis-

tributed among the factors of the polynomial. In case of our polynomial $s\alpha_1^2 - t\alpha_2^2$ we might get:

$$(\sqrt{s}\alpha_1 - \sqrt{t}\alpha_2)(\sqrt{s}\alpha_1 + \sqrt{t}\alpha_2) = s\alpha_1^2 - t\alpha_2^2 = s(\alpha_1 - \sqrt{\frac{t}{s}}\alpha_2)(\alpha_1 + \sqrt{\frac{t}{s}}\alpha_2).$$

Simply taking the intersection would lead to wrong results, because one set could contain $\sqrt{s}\alpha_1 - \sqrt{t}\alpha_2$ and the other one $\alpha_1 - \sqrt{\frac{t}{s}}\alpha_2$ resulting in removing both polynomials from the set of critical polynomials. This problem can be avoided via comparing the roots of the polynomials, for this polynomial we get $\alpha_1 = \sqrt{\frac{t}{s}}\alpha_2$ in both cases, the roots with respect to α_2 are the same too, hence the two polynomials should be treated as the same polynomial. Doing this comparision algebracily is very slow but nummerically it might happen that we get a false result. This may erroneously lead to linearly reducible sets because it can reduce the amount of critical polynomials. I tested the slow algebracily comparision on a few examples and got the same result as in the nummerical attempt. Thus I decided to use the much faster nummerical comparision.
Another source of errors may appear during factorization because it may happen that the program is unable to find a factorization of a polynomial even if one exists. Hence we get polynomials with higher degrees in our set of critical minors which may lead to the case that the reduction stops because we cannot find any variable for which all polynomials are linear thus we get a false negativ result. Maple is able to find all roots of polynomials up to degree four, thus the error can only appear if polynomials have degree five or higher in an α_k. This problem cannot be completly solved, because there is no general solution formula for polynomials of degree five which was already proved 1823 by Niels Henrik Abel.
A last source of errors is the problem of detecting whether a polynomial is linear, because Maple does not always simplify everything which leads to terms seeming to be nonlinear but in fact they are linear. For example $\alpha_1 + \sqrt{\alpha_2^2}$ is nonlinear for Maple. If we are using the `assuming` command to define α_2 to be a positiv real number Maple frequently crashes. At least the `assuming` command seems to work with the function which calculates the roots of a polynomial. There it is used to reduce the number of erroneosly nonlinear polynomials, but it is not assured that the same problem cannot appear on another position.

- **Improved Fubini reduction** For the implementation of the Improved Fubini algorithm (see section 4.5) I extended the previous implementation by adding the compatibility check which is stored in a set of compatible pairs of polynomials. It uses the same factorisation code as before, leading to the same problems.

6.2 Choice of test candidates

First of all I want to remark, that the analysis with the program does not have the aim of trying all possible simple graphs. It should show that there are many nontrivial graphs for which the reduction succeeds. Therefore I made a few assumptions to obtain a small class of graphs for testing. These constraints will consist of five assumptions.

- **Number and kind of external momenta**
  Graphs without external momenta are already analysed in [5] and Francis Brown also showed that these results can be transfered to graphs with two external legs. To obtain a small set of graphs we should take at most four external legs. For many applications it is sufficant to consider only external momenta which fulfill the on-shell condition $p_i^2 = 0$. In this case the second Symanzik polynomial φ will be zero if we only have three external legs hence we would be back to the case of vaccum graphs. Thus my sample should have four external legs. This also has the advantage that we are able to transform the products of the external momenta into Mandelstam variables, which is not possible for five or more external legs. If we want to extend the program to higher numbers of external momenta we are forced to use a more general way to obtain independent scalar values for the dot products of the momenta (e.g. see [7]).

- **Internal edges**
  We can decide if we allow massive internal edges but [11] states that every graph with a cut through three massive lines cannot be expressed by less general class of polylogarithms hence our reduction is likely to fail. Thus we should stay in the case without massive edges. This also greatly reduces the number of cases we have to test.

- **Loop number**
  Most of the one- and two-loop calculations can be calculated without the hyperlogarithm approach. Hence we should go to loop numbers as high as possible. High loop numbers lead to a problem because the

number of possible graphs with n loops grows faster than exponential. Thus we have to determine how many loops are possible, but we should test at least all 3-loop graphs.

Hence we restrict ourselves to all graphs with four external legs with on-shell condition $p_i^2 = 0$ and massless internal edges and start with up to three loops.

6.3 Refining the sample

To deal with the 3-loop-graphs we have to find a representative set of graphs. Randomly choosing edges would lead to many isomorphic graphs which do not give any new information. Hence we consider the following:

- Nonisomorphic graphs:

  Maple has a built-in command to create all nonisomorphic graphs with a given vertex and edge count. But it is unable to test if choosen vertices for external momenta are equivalent. For only four external legs we can still test all possiblities.

- Edge and external momentum permutations

  The permutation of internal edge labels only leads to a permutation of the indices in the Symanzik polynomials and does not change if the graph is reducible or not. Permutation of external momenta only permutate the Mandelstam variables. Hence it is sufficent to test only one way of labeling the edges.

- 1PI-graphs:

  It is well-known that we do not have to test graphs which are only 1-edge-connected because they can be written as a product of two graphs created by deleting the edge which links both parts.

- 2-valent vertices:

  A vertex with only two internal edges and without external edges does not appear in any theory. Hence these cases can be neglected too. Furthermore it would produce an infinte amount of graphs for a given loop number because splitting one edge into two edges does not change the loop number but creates a new graph. One more feature of our algorithm was that in 2.2 the reduction algorithms do not depend on the powers v_i and splitting one edge only doubles the v_i.

Furthermore one can neglect graphs which contain an already found critical minor. But these graphs can be used to test the implementation, because the theorem in the previous chapter tells us, that the set of linearly reducible graphs is minor closed.

Chapter 7

Results

7.1 Fubini algorithm

7.1.1 Some statistical results

For up to 2-loop-diagrams every graph with four external momenta fullfilling the on-shell condition $p_i^2 = 0$ and with massless internal lines is linearly reducible. Using the server of the math institute enabled me to run 4-loop-cases on about 200 cores for about four weeks. During this time about 600 cases were tested. The server was nessesary because even a few single cases reached about 300GB RAM usage and therefore cannot be handled on a standard computer. In comparision none of the 3-loop-cases reached more than 8GB RAM usage. The testing would have been more efficent if we could run a single case on several cores but the current Maple version is unable to execute the `factor` command in a multithreaded environment.
For the 3-loop-case we obtain 109 cases in our studied class after decreasing the number of these cases using the ideas of the previous section. Only 39 of these cases are not linearly reducible. Hence in about 64% of the cases it is possible to use the integration via hyperlogarithms. This number would change if we would change one of the ways we reduced the number of cases, but it shows that the algorithm succeeds in many cases. 29 cases out of the 39 cases which are not linearly reducible failed because they contain the critical minor K_4.
The 600 cases tested are only a small part of all 4-loop-cases, but they show, that the algorithm can also be used for graphs with higher loop numbers. From the about 600 cases tested, about 100 failed to be linearly reducible because they contained the critical minor K_4 and another 100 cases other failed too. An investigation of these latter graphs did not exhibit new critical minors which do not already appear on the 3-loop-level. But this does not

mean that there are no new critical minors with four loops, because it could be similar to the 3-loop-case where we only found a single small critical minor and all other critical minors have at least nine edges.
It seems that the fraction of linearly reducible graphs with four loops is higher, but this is a false impression. It depends on the fact, that only small graphs were tested, which contain less critical minors.
Due to experience about the memory usage I not even tried to compute some 5-loop-cases, but it is clear that the share of linearly reducible graphs will decrease. Even if we would assume that there are no more critical minors, the chance that a given graph contains an already known critical minor increases with the size of the given graph.

7.1.2 Critical minors

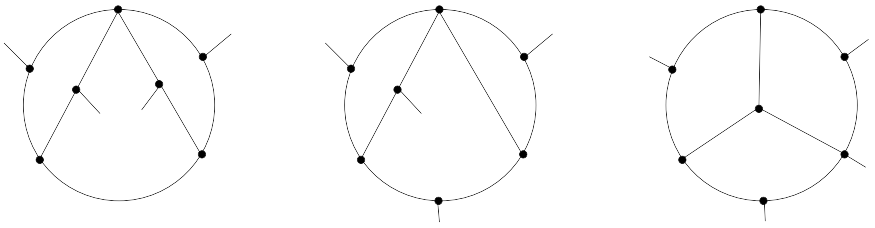
As already stated, the first critical minor which appeared was the K_4 on the 3-loop-level. It is the only case where we can prove by hand that it is a critical minor. This will be done in the next subsection.
On the 3-loop-level there are 10 cases not containing the K_4 which are not linearly reducible, but one graph appeared three times with different isomorphic external momenta. Hence we only have to consider the 8 graphs shown in figure 7.1.

The first three of them contain only 9 edges thus they must be critical minors because all graphs with at most 8 edges which do not contain the K_4 are linearly reducible. From the other 5 graphs only three are critcal minors because the other two contain a critcal minor with 9 edges. The graph (d) contains the graph (b) as minor and the graph (c) is a minor of graph (g).
Hence we get a set of 7 critical minors for 3 loops. These are the graphs $(a), (b), (c), (e), (f), (h)$ in figure 7.1 and the already known K_4.

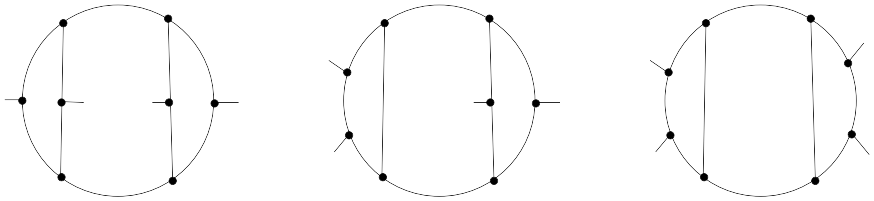
7.1.3 The critcal minor K_4

For the attempt to reduce K_4 we will use the same ordering of edges shown in figure 7.2 and assumptions as in the program. This leads to the graph-polynomials:

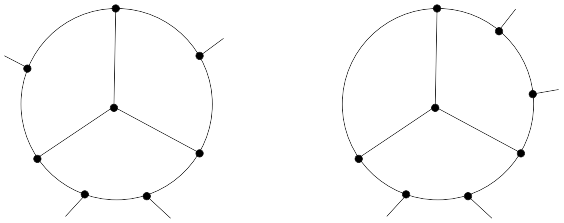
$$\begin{aligned}\varphi =&\alpha_4\alpha_5\alpha_6 + \alpha_3\alpha_4\alpha_6 + \alpha_3\alpha_4\alpha_5 + \alpha_2\alpha_5\alpha_6 + \alpha_2\alpha_4\alpha_5 + \alpha_2\alpha_3\alpha_6 + \alpha_2\alpha_3\alpha_5\\ &+ \alpha_2\alpha_3\alpha_4 + \alpha_1\alpha_5\alpha_6 + \alpha_1\alpha_4\alpha_6 + \alpha_1\alpha_3\alpha_6 + \alpha_1\alpha_3\alpha_5 + \alpha_1\alpha_3\alpha_4 + \alpha_1\alpha_2\alpha_6\\ &+ \alpha_1\alpha_2\alpha_5 + \alpha_1\alpha_2\alpha_4,\\ \Psi =&- s\alpha_2\alpha_3\alpha_4\alpha_5 - t\alpha_1\alpha_3\alpha_4\alpha_6 + (s+t)\alpha_1\alpha_2\alpha_5\alpha_6.\end{aligned}$$



(a) (b) (c)



(d) (e) (f)



(g) (h)

Figure 7.1: Not linearly reducible graphs

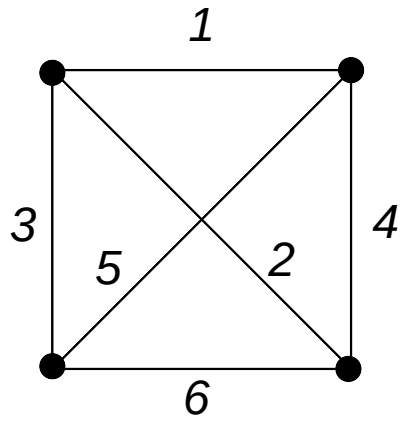


Figure 7.2: Non linearly reducible graph K_4

Due to the high symmetry of the graph K_4 we only have to consider a few subsets of edges for the reduction. For the first step we can simply choose edge 1 because choosing any other edge will only lead to a permutation of the indices of the α_i. For the second step there are two possiblities. We can choose two connected edges, without loss of generality 1 and 2, or two edges without common vertex, without loss of generality 1 and 6.
We need to calculate the set $S_{(1)}$ first.

$$\begin{aligned}\tilde{S}_{(1)} = \{&\alpha_4\alpha_5\alpha_6 + \alpha_3\alpha_5\alpha_6 + \alpha_3\alpha_4\alpha_5 + \alpha_2\alpha_5\alpha_6 + \alpha_2\alpha_4\alpha_5 + \alpha_2\alpha_3\alpha_6 + \alpha_2\alpha_3\alpha_5 \\ &+ \alpha_2\alpha_3\alpha_4, \alpha_5\alpha_6 + \alpha_4\alpha_6 + \alpha_3\alpha_6 + \alpha_3\alpha_5 + \alpha_3\alpha_4 + \alpha_2\alpha_6 + \alpha_2\alpha_5 + \alpha_2\alpha_4, \\ &- s\alpha_2\alpha_3\alpha_4\alpha_5, -t\alpha_3\alpha_4\alpha_6 + (s+t)\alpha_2\alpha_5\alpha_6, \\ &(\alpha_4\alpha_5\alpha_6 + \alpha_3\alpha_5\alpha_6 + \alpha_3\alpha_4\alpha_5 + \alpha_2\alpha_5\alpha_6 + \alpha_2\alpha_4\alpha_5 + \alpha_2\alpha_3\alpha_6 + \\ &\alpha_2\alpha_3\alpha_5 + \alpha_2\alpha_3\alpha_4)\,(-t\alpha_3\alpha_4\alpha_6 + (s+t)\alpha_2\alpha_5\alpha_6) + \\ &(\alpha_5\alpha_6 + \alpha_4\alpha_6 + \alpha_3\alpha_6 + \alpha_3\alpha_5 + \alpha_3\alpha_4 + \alpha_2\alpha_6 + \alpha_2\alpha_5 + \alpha_2\alpha_4)\,(s\alpha_2\alpha_3\alpha_4\alpha_5)\}\end{aligned}$$

The first four polynomials in $\tilde{S}_{(1)}$ are linear in all α_i. We only need to try to factorize the last polynomial which is of degree two in all α_i, thus we need to find a factorisation to start the next step. Let us call the polynomial f. Assume f can be written as a product of two polynomials f_1, f_2 both of degree 1 in α_2. Hence the roots with respect to α_2 of f_1 and f_2 have the simple form $\frac{g}{h}$ with polynomials g and h. Thus the roots of f should have the same form. Because f is of degree 2 in α_2 we can simply calculate the roots $x_{1/2}$.

$$
\begin{aligned}
f =&\alpha_2^2 A + \alpha_2 B + C\\
A =&t\alpha_6^2\alpha_5^2 + s\alpha_5^2\alpha_4\alpha_3 + s\alpha_5^2\alpha_3\alpha_6 + s\alpha_5^2\alpha_4\alpha_6 + t\alpha_6\alpha_5^2\alpha_3 + s\alpha_6^2\alpha_5^2 + t\alpha_6\alpha_5\alpha_4\alpha_3\\
&+ t\alpha_6\alpha_5^2\alpha_4 + s\alpha_6^2\alpha_5\alpha_3 + t\alpha_6^2\alpha_5\alpha_3 + s\alpha_5\alpha_4^2\alpha_3 + 2s\alpha_6\alpha_5\alpha_4\alpha_3\\
B =&t\alpha_6\alpha_5^2\alpha_4\alpha_3 + s\alpha_5\alpha_4^2\alpha_3^2 + t\alpha_6^2\alpha_5^2\alpha_4 + s\alpha_6^2\alpha_5\alpha_4\alpha_3 + s\alpha_5\alpha_4^2\alpha_3\alpha_6\\
&+ s\alpha_6^2\alpha_5^2\alpha_4 - t\alpha_6\alpha_4^2\alpha_3^2 + s\alpha_5\alpha_4\alpha_3^2\alpha_6 - t\alpha_6^2\alpha_4\alpha_3^2 + 2s\alpha_6\alpha_5^2\alpha_4\alpha_3\\
&+ s\alpha_5^2\alpha_4\alpha_3^2 - t\alpha_6\alpha_4\alpha_3^2\alpha_5 - t\alpha_6\alpha_4^2\alpha_3\alpha_5\\
C =&- t\alpha_6^2\alpha_4^2\alpha_3^2 - t\alpha_6^2\alpha_4^2\alpha_3\alpha_5 - t\alpha_6\alpha_4^2\alpha_3^2\alpha_5
\end{aligned}
$$

The roots are $x_{1/2} = \frac{-B\pm\sqrt{B^2-4AC}}{2A}$, hence we need $\sqrt{B^2-4AC}$ to be a polynomial. But in fact, it is not a polynomial. We can prove this via showing, that it has no double roots, but it is much easier to set most of the variables to 1 and look at the new polynomial. $\alpha_3 = \alpha_4 = \alpha_5 = s = t = 1$ gives $B^2 - 4AC = 36\alpha_6^4 + 80\alpha_6^3 + 56\alpha_6^2 + 16\alpha_6 + 4 = 4(\alpha_6+1)^2(9\alpha_6^2 + 2\alpha_6 + 1)$. This is not a square, hence it is impossible to factor f into two polynomials of degree one in α_2.
The high symmetry of the K_4 also proves that it cannot be factored into two polynomials of degree one in α_3, α_4 or α_5. The only missing part is to prove if it factors into two polynomials of degree one in α_6. This can be done in the same way as before, leading to:

$$
\begin{aligned}
f =&\alpha_6^2 A + \alpha_6 B + C\\
A =&t\alpha_5^2\alpha_2^2 + t\alpha_5^2\alpha_4\alpha_2 + s\alpha_5\alpha_4\alpha_3\alpha_2 - t\alpha_4\alpha_3^2\alpha_2 + s\alpha_5^2\alpha_4\alpha_2 + s\alpha_5^2\alpha_2^2 + s\alpha_5\alpha_3\alpha_2^2\\
&+ t\alpha_5\alpha_3\alpha_2^2 - t\alpha_4^2\alpha_3^2 - t\alpha_4^2\alpha_3\alpha_5\\
B =&t\alpha_5\alpha_4\alpha_3\alpha_2^2 + s\alpha_5^2\alpha_4\alpha_2^2 - t\alpha_4\alpha_3^2\alpha_5\alpha_2 + 2s\alpha_5\alpha_4\alpha_3\alpha_2^2 - t\alpha_4^2\alpha_3^2\alpha_5 + s\alpha_5^2\alpha_3\alpha_2^2\\
&+ s\alpha_5\alpha_4^2\alpha_3\alpha_2 + t\alpha_5^2\alpha_3\alpha_2^2 - t\alpha_4^2\alpha_3^2\alpha_2 + s\alpha_5\alpha_4\alpha_3^2\alpha_2 + t\alpha_5^2\alpha_4\alpha_2^2 + 2s\alpha_5^2\alpha_4\alpha_3\alpha_2\\
&- t\alpha_4^2\alpha_3\alpha_5\alpha_2 + t\alpha_5^2\alpha_4\alpha_3\alpha_2\\
C =&s\alpha_5^2\alpha_4\alpha_3^2\alpha_2 + s\alpha_5\alpha_4^2\alpha_3\alpha_2^2 + s\alpha_5\alpha_4^2\alpha_3^2\alpha_2 + s\alpha_5^2\alpha_4\alpha_3\alpha_2^2.
\end{aligned}
$$

Using the same setting $\alpha_3 = \alpha_4 = \alpha_5 = s = t = 1$ gives $B^2 - 4AC = 17\alpha_2^4 - 20\alpha_2^3 - 10\alpha_2^2 + 12\alpha_2 + 1 = (\alpha_2-1)^2(17\alpha_2^2 + 14\alpha_2 + 1)$ which is again no square. This leads to the fact, that $S_{[1](2)}$ cannot be computed, due to symmetry the same holds for $S_{[2](1)}$ hence $S_{[1,2]}$ is unobtainable. The same holds for $S_{[1,6]}$.
Hence the reduction of K_4 already fails for the second variable and therefore it is not linearly reducible.

7.2 Improved Fubini algorithm

7.2.1 Some statistical results

For comparison we tested the same cases as in the section before also with the Improved Fubini algorithm. We already know, that every graph which is linearly reducible is also weakly linearly reducible. Hence we expect at most the same number of graphs where the algorithm fails.
Indeed, the number of cases where the algorithm fails decreased from 39 to 34. The K_4 is still a critcal minor appearing in 29 cases which are not weakly linearly reducible.
Due to the experience with 4 loops with the Fubini algorithm I decided not to calculate all graphs on 4 loop level. I only tried a few examples to test the performance difference between the algorithms. On small graphs there is no difference in time and memory usage but for bigger graphs the Improved Fubini algorithm in general is slightly faster and uses only a fraction of the memory, but there are also a few graphs where the Fubini algorithm is faster.

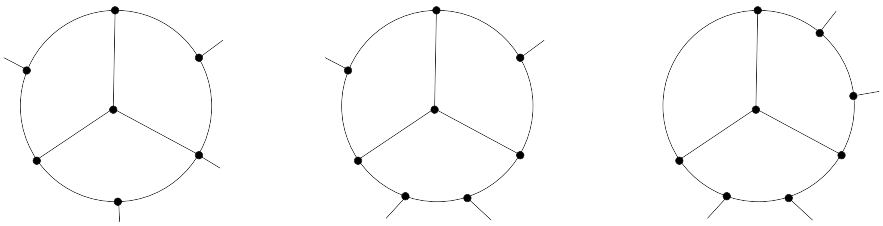
7.2.2 Critical minors

As already said, we still keep the K_4 as a critcal minor, because we saw in the last section that it already fails in the second step because there is no α_i in which all polynomials are linear. This cannot change because the first step in the improved Fubini algorithm and the Fubini algorithm are the same because in the first step all polynomials are compatible. Hence $S_{(1)}$ is the same in both algorithms.
The other five cases not involving the K_4 only consist of 3 different graphs shown in figure 7.3. They already appeared as non linearly reducible graphs in the previous section and we know that graph (b) has the graph (a) as a minor. Hence we only get a total of 3 critical minors for the improved Fubini algorithm.

7.3 Comparision between the algorithms

7.3.1 Reducibility

The decrease in non-reducible cases shows that the improved Fubini algorithm is an improvement. I expect that also in higher loop orders there are more critical minors if we only use the Fubini algorithm. Only viewing the number of non reducible cases, which decreased from 39 to 34 is a small benefit, but on the level of critical minors we achived a decrease from seven



(a) (b) (c)

Figure 7.3: Not weakly linearly reducible graphs

down to only three critical minors. The great impact of K_4 on the number of non reducible cases follows from the fact, that K_4 is much smaller than the other critical minors, hence it is a minor for many more graphs.

7.3.2 Size of the compatibility graph

As already stated in the description of the improved Fubini algorithm, we would get back the Fubini algorithm if we compute at each step with the complete graph as compatibility graph. Hence our benefit depends on how many edges the compatibility graph contains. This number of edges is hard to estimate because it depends on polynomials which appear multiple times, the number of irreducible factors of each polynomial and the intersection of compatiblity graphs.
To get an impression of the numbers, I studied how many polynomials in each step have been found and how many compatible pairs. Running through all 3 loop cases and average over each polynomial count leads to the diagram in figure 7.4. The fluctuations for higher polynomial numbers appear because of the fact that these high counts only appears one or two times. The error bars are calculated from the variance of edge counts for a specific count of polynomials. If we only have a given polynomial count once in our test set, we cannot calculate the variance and this point gets no error bar.
The complete graph contains $\binom{n}{2}$ edges if there are n polynomials. For comparision this number is drawn into the graph as individual points too.
I also considered comparing total numbers of all polynomials in a reduction but this leads to a misleading result, because in the improved Fubini reduction one might do more steps leading to more polynomials in the reduction. Hence it will depend on the chosen cases. Another problem is that the factorisation is slightly differently implemented which may lead to different

numbers of polynomials.

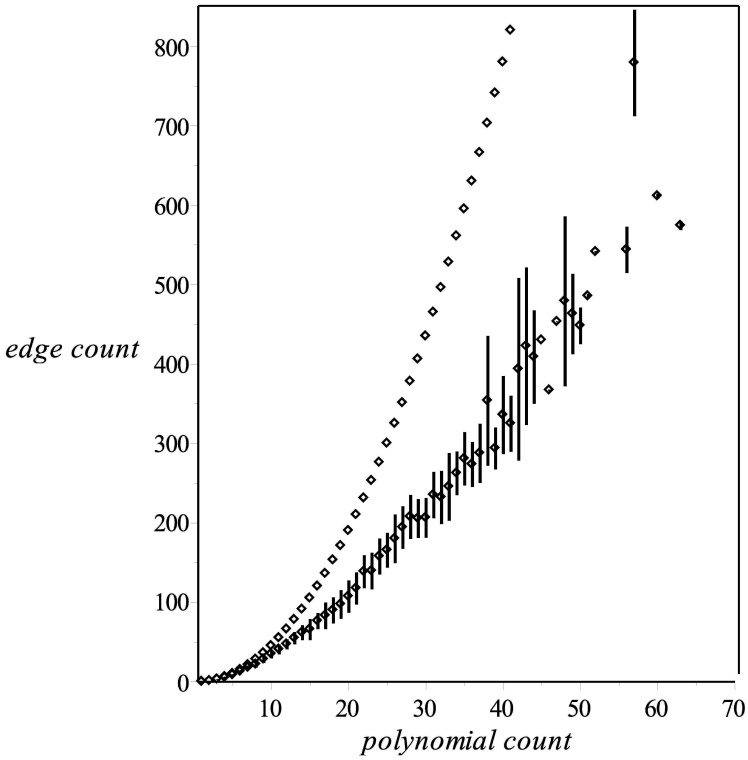


Figure 7.4: Edge count over polynomial count

Chapter 8

Introduction to the computer program

This chapter will explain how you can use the program for your own calculations.
First you have to download the source code [1] from the homepage of Prof. Dirk Kreimer. The Fubini algorithm and the improved Fubini algorithm are contained in the two files "fubini.txt" and "improvedfubini.txt". To use them in Maple, copy the files to the Maple working folder or start Maple from the folder the files are contained in. To load them into a Maple file use

```
read("fubini.txt"):

read("improvedfubini.txt"):
```

8.1 Implemented commands

8.1.1 The command poly

The command `poly` computes the two Symanzik polynomials for a given graph with 4 external edges and transforms the products of external momenta into Mandelstam variables. There is an on-shell-condition $p_i^2 = mein_i^2$ assumed thus the second Symanzik polynomial also depends on the rest masses of the four external particles $mein_1, mein_2, mein_3, mein_4$. If we want to obtain $p_i^2 = 0$, we have to set the four masses to 0. One can also set an individual mass for every internal line.
For the input the `GraphTheory` package of Maple is used. For instructions

[1] `www.mathematik.hu-berlin.de/~kreimer/fubinireduction.zip`

how to create a graph see the documentation of Maple. I would suggest to create the graph with the help of the weight matrix.
Maple's `GraphTheory` package does not support multiple edges, therefore I use the built-in weights of edges. An edge with weight 1 stands for a single edge between the two vertices, weight 2 stands for two edges between the vertices and so on. To set the masses for the internal edges we use a list as a second argument. The first entry is the mass of the first internal edge and so on. The ordering of the internal edges depends on the ordering of the vertices in a way that we first sort the edges by the lower vertex number and if two edges have the same lower vertex number they are ordered by the higher vertex number. In the case of two edges connecting the same vertices their ordering does not matter. If the graph was created by the use of a weight matrix, the ordering can be obtained by counting the entries in the upper triangle line by line.
In the main program the masses have to be numbers or variables from the set $\{M, M_1, M_2, \ldots M_{15}\}$.

The third input argument is a list with four integer entries to define the position of the four external momenta.
The output consists of the two Symanzik polynomials and the number of internal edges. Therefore the output can be directly used for the main program.

Example. *This example shows the calculation of the Symanzik polynomials for the example graph in figure 4.2 with the same assumptions. These are: one line is massive and all external momenta fulfill* $p_i^2 = 0$.

```
graph:=MakeWeighted(Graph(Matrix(4, 4, [0, 1, 0, 1, 1, 0, 1, 0, 0,
1, 0, 1, 1, 0, 1, 0]))):
S0:=poly(graph,[M,0,0,0],[1,2,3,4]);
     [{a[2]+a[4]+a[3]+a[1],a[1]^2*M^2+a[1]*M^2*a[2]+a[1]*M^2*a[4]+
     a[1]*M^2*a[3]+s*a[4]*a[1]-mein[1]^2*a[4]*a[1]-
     mein[1]^2*a[2]*a[1]-mein[2]^2*a[3]*a[1]-mein[2]^2*a[4]*a[1]+
     t*a[4]*a[1]-mein[3]^2*a[4]*a[1]-mein[3]^2*a[4]*a[3]-
     s*a[3]*a[2]-mein[4]^2*a[4]*a[2]-mein[4]^2*a[4]*a[1]}, 4]
#introducing the p[i]^2=0 condition
Sonshell0:=subs({mein[1]=0,mein[2]=0,mein[3]=0,mein[4]=0},S0);
     [{a[2]+a[4]+a[3]+a[1],a[1]^2*M^2+a[1]*M^2*a[2]+a[1]*M^2*a[4]+
     a[1]*M^2*a[3]+s*a[4]*a[1]+ t*a[4]*a[1]- s*a[3]*a[2]}, 4]
```

8.1.2 The command main

The command `main` starts the Fubini algorithm or the improved Fubini algorithm and will save the results in a global variable *sset*. It needs two arguments. The first one is the set of critical polynomials we have at the beginning. In case of Feynman integrals these are the two Symanzik polynomials. The second argument is the number n of integration parameters. The integration parameters are $a_1, \dots a_n$, hence the starting set must be written using the a_i as integration parameters.
Most of the limitations of this command are based on the hardware used because especially the Fubini algorithm consumes large amounts of memory. It is also possible to reach Maple limitations which result in errors like stack-overflow. The only build-in limitation is given by the numerical comparision of two polynomials which involves only random values for up to 15 integration variables. This can be extended without problems but even general graphs with 11 edges are not possible on a standard computer. The random values are fixed thus the result is deterministic and will not change if we run the program again. But as already stated in the previous chapter, this might lead to false positive results, but this risk in nessessary if we want to run the program in a reasonable amount of time.
There is a small difference between the implementation and the described algorithm in the third chapter. To reach a higher performance, all critical polynomials not depending on the Feynman parameters will be removed after each step. This improves the performance but will not change whether a set is (weakly) linearly reducible.

Example.

```
Sonshell0:=[{a[2]+a[4]+a[3]+a[1],a[1]^2*M^2+a[1]*M^2*a[2]+
a[1]*M^2*a[4]+a[1]*M^2*a[3]+ s*a[4]*a[1]+t*a[4]*a[1]-
s*a[3]*a[2]}, 4]:
main(op(Sonshell0));
```

8.1.3 The command fubinishow

The command `fubinishow` is used to show the results of the algorithm after running the `main` command. It will process the global variable containing many informations of the algorithm process. It only needs an integer as input to define which information should be presented. The possibilities are:

- 0:
  Only shows if the set was linearly reducible or not.

- 1:
  If the given set is (weakly) linearly reducible it shows the best integration order. Therefore the weight function `gewicht` is used to define the effort to calculate a given step. There is already a predefined function of the form k^i with k the number of critical polynomials in the step and i the number of the step. This function is choosen because a hyperlogarithm in this step could have a weight of i and each of the i paramenters can be chosen out of the k polynomials. Thus we could have k^i terms in this step. This function can be replaced by a function defined by the user.

- 2:
  It shows the best integration order and the appearing critical polynomials for each step.

- 3:
  It shows the sets of critical polynomials for all subsets of edges.

- 4:
  This case is only available for the improved Fubini reduction and it shows in addition the pairs of compatible polynomials.

Note that option 3 and 4 create a large output which could freeze the Maple instance for several minutes.
To get a specific set of critical polynomials there is also the command `fubinishowsingle` which needs a list of Feynman parameter indices as input and shows the specific set of critical polynomials together with the pairs of compatible polynomials.

Example. *After running the main function with the arguments from the previous example one can get an integration order via*

```
fubinishow(1);
     [4,3,2,1]
```

A single intermediate result like $S_{(2)}$ can be obtained via:

```
fubinishowsingle([2]);
     [[s*a[4]*a[1]+t*a[4]*a[1]+s*a[4]*a[3]+s*a[3]^2+s*a[3]*a[1],
     a[1]*M^2+t*a[4]+M^2*a[4]+M^2*a[3]+s*a[4],a[1],a[4]+a[3]+
     a[1],a[1]*M^2-s*a[3]],{{a[1],a[1]*M^2-s*a[3]},{a[1],
     a[4]+a[3]+a[1]},{a[1],a[1]*M^2+t*a[4]+M^2*a[4]+M^2*a[3]+
     s*a[4]},{a[1],s*a[4]*a[1]+t*a[4]*a[1]+s*a[4]*a[3]+s*a[3]^2+
```

```
s*a[3]*a[1]},{a[1]*M^2-s*a[3],a[1]*M^2+t*a[4]+M^2*a[4]+
M^2*a[3]+s*a[4]},{a[1]*M^2-s*a[3],s*a[4]*a[1]+t*a[4]*a[1]+
s*a[4]*a[3]+s*a[3]^2+s*a[3]*a[1]},{a[4]+a[3]+a[1],a[1]*M^2+
t*a[4]+M^2*a[4]+M^2*a[3]+s*a[4]},{a[4]+a[3]+a[1],
s*a[4]*a[1]+t*a[4]*a[1]+s*a[4]*a[3]+s*a[3]^2+s*a[3]*a[1]},
{a[1]*M^2+t*a[4]+M^2*a[4]+M^2*a[3]+s*a[4],s*a[4]*a[1]+
t*a[4]*a[1]+ s*a[4]*a[3]+ s*a[3]^2+ s*a[3]*a[1]}}]
```

8.1.4 The commands hasminork4, istminimalg, istminimalp

The command `hasminork4` tests if a given graph with four external momenta contains the K_4 as a minor where the external momenta are at the four vertices of the graph. The input is a graph and a list of integers for the external momenta. The output is 1 if it has the K_4 as a minor and 0 otherwise.
The command `istminimalg` proves if a graph is 1PI. The input is an unweighted graph and the output is true for 1PI and false otherwise. `istminimalp` tests if a graph given by its weight matrix with given momenta setting has no 2-valent vertices or if the two edges of the 2-valent vertex are connecting two vertices with a line between them. This can be used to obtain all graphs of given loop number via nonisomophic graphs, because graphs in Maple's `GraphTheory` package cannot contain double edges and the described setting corresponds to a double edge.

Example. *Assume we still have the variables defined during the last few examples.*

```
istminimalg(Graph(Matrix(4, 4, [0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0,
1, 1, 0, 1, 0])));
     true
istminimalp(Matrix(4, 4, [0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1,
0, 1, 0],[1,2,3,4]);
     1
hasminork4(graph,[1,2,3,4]);
     0
```

8.2 Hints for testing

8.2.1 The command NonIsomorphicGraphs

Maple has a built-in command to derive all non-isomorphic graphs of given edge and vertex number E and V. This can be used to produce all graphs

with a given loop number L because $L = E - V + 1$ holds for all connected graphs. Assuming that one can neglect graphs with 2-valent vertices, the number of vertices a graph with fixed loop number is limited.
This set can be refined by the functions `hasminork4, istminimalg, istminimalp` and other constraints.

Example. *To examine all 1PI graphs with two loops and four or five vertices you can use*

```
setgraph:=[NonIsomorphicGraphs(5,4,output=graphs,outputform=
adjacency,restrictto=connected),NonIsomorphicGraphs(6, 5,
output=graphs,outputform=adjacency,restrictto=connected)]:
for A in setgraph do
if istminimalg(A)=1 then
yourmainfunction(A);
fi;
od;
```

Remark. *It is usally faster and less memory consuming to deal with the adjacency matrices than with the graphs.*

8.2.2 Multithreaded computing

The current program runs on a single core only because the multithreaded approach crashes. The reason might be that the `factor` command from Maple is not threadsafe. This might be solved in future versions of Maple. To enable the use of multiple cores in the program for the improved Fubini reduction, the `seq` command in line 291 has to be exchanged by the parallel command `Seq`. If I recognize a change in Maple, I will also upload a new multithreaded version. In Maple 15 and 16 it does not work.
But even if it is impossible to use multiple cores for one problem, one can run multiple instances of this programm to test many different cases at the same time.

Chapter 9

Conclusion

The advantages of the reduction algorithms presented in chapter 4 are that they are much faster than the integration and can be used to determine a suitable integration order. Thus they are an important step when deriving a Feynman integral since the algorithms already show whether the integration algorithm can be applied in this case. Unfortunately, the three reduction algorithms only give an upper bound for the polynomials appearing during the integration. Even the improved Fubini reduction algorithm cannot give a strict bound, because the appearing polynomials not only depend the two Symanzik polynomials by itself but also on the form of the integrand.
Hence a perfect reduction algorithm for testing whether a graph is integrable by hyperlogarithms will depend on the theory used. This makes the results not universally valid. The results in chapter 7 also show that there is only a small set which might be further refined by a better reduction algorithm. Chapter 7 also shows that the improved Fubini algorithm is a useful improvement of the reduction algorithm thus it is worthwhile to find an algorithm which delivers an even better upper bound to further reduce the amount of non reducible graphs.

The minor closedness of the (weakly) linearly reducible graphs which has been proved in chapter 5 can be viewed as a first step in characterizing the set of reducible graphs. We introduced minors as an abstract graph theoretical object, but it also appears in the calculation of pertubative quantum field theory. That is because the minors of the higher loop corrections are the corrections of lower loop order or not connected graphs, which can be neglected. Thus we obtain two useful implications:
1. If the graphs corresponding to the n loop corrections are not linearly reducible, then this also holds for the higher loop corrections.
2. If all graphs corresponding to the n loop corrections are linearly reducible

then this also holds for all lower loop corrections.

Chapter 8 is written to engage other people to use this algorithm or further extend the program such that it is capable of doing the integration or the determination of the Feynman diagrams for the corrections and their renormalisations. This automation would enable us to calculate higher loop corrections compared to the integration by hand. The integration algorithm already has been implemented by Erik Panzer (see [15]) and an implementation using multiple polylogarithms is in progress (see [16]).

I would like to thank Dirk Kreimer for the opportunity to write this thesis in his research group, as well as for suggesting the topic. I am very grateful to my advisor Christian Bogner for his support in all phases of the work, especially for his hints and corrections during the writing process. My thanks also go to Erik Panzer and Francis Brown for the discussions concerning the integration algorithm, and finally I want to thank everyone in the research group for the great atmosphere.

Bibliography

[1] J. C. Collins, "Renormalization. An Introduction To Renormalization, The Renormalization Group, And The Operator Product Expansion", Cambridge, Uk: Univ. Pr. (1984) 380p

[2] G. 't. Hooft and M. Veltman, "Regularization and renormalization of gauge fields", Nuclear Physics B 44 (1972)

[3] F. Brown and D. Kreimer, "Decomposing Feynman rules", PoS LL **2012** (2012) 049 [arXiv:1212.3923 [hep-th]].

[4] C. Bogner and S. Weinzierl, "Feynman graph polynomials", Int. J. Mod. Phys. A **25** (2010) 2585 [arXiv:1002.3458 [hep-ph]].

[5] F. Brown, "The Massless higher-loop two-point function", Commun. Math. Phys. **287** (2009) 925 [arXiv:0804.1660 [math.AG]].

[6] F. Brown, "On the periods of some Feynman integrals", arXiv:0910.0114 [math.AG].

[7] F. Rohrlich, "Scalar Variables for the S-Matrix", Il nuovo cimento, Vol. XXXVIII, N.1 (1965)

[8] R. J. Eden, P. V. Landshoff, D. I. Olive and J. C. Polkinghorne, "The analytic S-Matrix", Cambridge, Uk: Univ. Pr. (1966)

[9] C. Itzykson and J. B. Zuber, "Quantum Field Theory", McGraw-Hill Inc (1980)

[10] L. Lewin, "Polylogarithms and associated functions", North Holland (1981)

[11] R. Scharf, "Zur Berechnung von skalaren Ein- und Zweischleifen-Integralen", Diploma thesis at Universität Würzburg (1991)

[12] N. Robertson, P. D. Seymour, "Graph minors. I." to "Graph minors. XX.", Journal of Combinatorial Theory (1983-2004)

[13] N. Robertson, P. D. Seymour, "Graph minors. XX., Wagner's conjecture", Journal of Combinatorial Theory (2004)

[14] J. A. Lappo-Danilewski, "Mémoires sur la théorie des systèmes des équations différentielles linéaires", Chelsea, New York (1953)

[15] E. Panzer, "On the analytic computation of massless propagators in dimensional regularization", Nuclear Physics, Section B 874 (2013)

[16] C. Bogner,F. Brown, "Symbolic integration and multiple polylogarithms", arXiv:1209.6524 [hep-ph]

Selbständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Berlin, den 9.09.2013

Martin Lüders