Next to Leading Order Calculations for Higgs Boson + Jets

ARMSTRONG, SIMON,THOMAS

Next to Leading Order Calculations for Higgs Boson + Jets

Simon Armstrong

A Thesis presented for the degree of
Doctor of Philosophy

Durham University

Institute of Particle Physics Phenomenology
Department of Physics
University of Durham
England

August 22, 2017

Next to Leading Order Calculations for Higgs Boson + Jets

Simon Armstrong

Submitted for the degree of Doctor of Philosophy

August 22, 2017

Abstract

Due to the recent Higgs boson discovery, an important target for particle physics is to investigate its properties to determine if it is the standard model Higgs boson or some other variety. The Large Hadron Collider is now in Run Two, collecting even more data at higher precisions, which requires predictions at next to leading order or higher orders. Therefore it is important to have an efficient and automatic calculation of the next to leading order amplitudes for the Higgs boson. This thesis discusses the methods needed to perform these calculations.

These calculations are specifically developed in order to add them to the BlackHat library, which already provides these types of calculations for amplitudes involving quarks, gluons and W and Z bosons. Both this thesis and BlackHat use recursive methods, as these are more efficient than using the Feynman rules directly. Specifically the BCFW recursion relation is used to calculate tree amplitudes and generalised unitarity is used to calculate one loop amplitudes. These methods are first used in 4 dimensions to calculate the cut constructable parts of the amplitudes, then the extension of these techniques to higher numbers of dimensions is discussed, allowing the rational terms to be extracted using D dimensional generalised unitarity. In general, two different even integer dimensions higher than 4 are required for a numeric implementation of D dimensional generalised unitarity with spinors, which therefore requires working in both 6 and 8 dimensions. To enable an efficient implementation, a technique is introduced that allows only 6 dimensional calculations to be used rather than both 6 and 8 dimensional calculations and a reduction of 6 dimensional calculations to be in terms of only 4 dimensional objects is developed. The methods presented in this thesis provide a solid groundwork for Higgs boson amplitudes to be implemented into BlackHat.

Declaration

The work in this thesis is based on research carried out at the Institute of Particle Physics Phenomenology, the Department of Physics, University of Durham, England. No part of this thesis has been submitted elsewhere for any other degree or qualification and it is all my own work unless referenced to the contrary in the text.

Acknowledgements

I would like to thank my supervisor Daniel Maitre for supporting and guiding me through this project. I would also like to thank my friends and family who were a huge help in proof reading my thesis, correcting my English and providing many helpful suggestions. I would especially like to thank my wife Beth for putting up with and supporting me throughout my PhD, particularly over the last year where we have both been working full time while I finished writing up my thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Motivation

It is an exciting time for particle physics, with the Large Hadron Collider (LHC) now collecting data for Run Two at higher than ever energies. At these high energies there is much hope that some new physics, beyond the standard model, will be detected which will help with understanding the problems with the standard model, such as the lack of candidates for dark matter and the hierarchy problem. Supersymmetry, in which each standard model has a heavier partner particle, is one example of a beyond the standard model theory that could be detected. These types of theory are one of the main targets for Run Two.

Another of the main targets for Run Two is characterisation of the recently detected Higgs boson. This would show whether it is the standard model Higgs boson or some more exotic version. There is also hope that the Higgs boson could provide hints of new, beyond the standard model, theories through its interactions with them. Both high precision measurements and high precision predictions are needed to enable accurate conclusions to be drawn. The LHC in its second run is starting to collect high precision data for the relevant channels and analyses, so the situation is being approached where the limitation is the precision of the model predictions. Many of the channels at the LHC produce large numbers of jets. To produce predictions a hard process is combined with parton shower and hadronisation algorithms in a Monte Carlo simulation, which converts quarks and gluons produced in the hard process into large showers of particles, through soft and collinear radiation and then into the observable hadrons which can be detected as part of jets. A group of hadronised particles, produced by soft and collinear radiation, travelling in roughly the same direction and from the same location, will be detected as a jet. Each of these

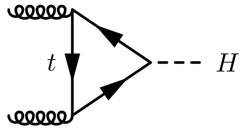


Figure 1.1: The leading contribution to the Higgs-gluon coupling which is mediated by a top quark loop.

jets is typically due to the radiation of a single particle produced in the hard process. Therefore, the leading contribution to processes with many jets will be from hard processes that produce a high multiplicity of particles. It is also important that the calculations are in a form that is efficient for use in a Monte Carlo simulation, so that they can be combined with the other steps to produce physically observable predictions.

The Higgs boson coupling is proportional to the mass of the particle involved and as such in quantum chromodynamics the strongest interaction is with the top and bottom quarks. Top and bottom quark masses are larger than the energy scale of the LHC and so can be treated as approaching infinite mass. All other quark masses are much lower than this scale and so can be approximated as massless. The exact leading order calculation for this type of amplitude is already a one loop amplitude, as the top quark runs in a loop. In this heavy top quark limit, as the mass is a large scale, the leading term in terms of the top quark mass can be approximated out and all other terms can be neglected. This only leaves the contribution from a top quark loop connecting two gluons and one Higgs boson, of the form in Figure 1.1, from which the loop quark's degrees of freedom can be integrated out to give a new effective vertex between a Higgs boson and two gluons. The effective Lagrangian term this introduces is given by

$$\mathcal{L}_H^{\mathrm{int}} = \frac{C}{2} H \,\mathrm{tr}\, G_{\mu\nu} G^{\mu\nu} \ , \tag{1.1}$$

where H is the Higgs field, G is the gluon field strength tensor and C is the dimensionful coupling constant which can be calculated order by order in α_s. As this limit is being used, all particles other than the Higgs boson will be assumed to be massless from here on. Currently, amplitudes with a Higgs boson and jets at tree level in the high top mass limit are automated for any multiplicity.

As the Next to Leading Order (NLO) correction and using the true masses of the top and bottom quarks could be large contributions, both are needed to increase the precision of our predictions. The accuracy of working in the infinite top mass limit at NLO has been directly investigated at low multiplicity by Harlander et al.[1] and Grazzini et al.[2] and has been found to give a relatively flat correction of the order of a few % for large areas of the phase space. This thesis examines the techniques and methods needed to calculate the NLO amplitude for a Higgs boson with many jets in the high top mass limit. The case of NLO without taking the high top mass limit is a two loop calculation and as such is beyond our current ability to compute in an efficient automated way for high multiplicities.

These NLO amplitudes are divergent in 4 dimensions and as such must be regulated. One of the most common methods, and the one used in this thesis, is to working in $d = 4-2\epsilon$ dimensions which causes the amplitude to have poles in ϵ which control and contain the divergences and will be cancelled with the soft and collinear contributions to calculate the finite cross section. There are many ways to perform this calculation. One scheme often used is the 't Hooft-Veltman scheme which allows all elements of the loop to extend into to the extra dimensions. Another scheme is the four dimensional helicity scheme[3] which keeps all spinor and polarisation vector states in 4 dimensions and allows only the internal loop momenta to enter the extra dimensions. This scheme has the advantage that the Ward identities are preserved which allows checks and tools to relate different amplitudes. This scheme is the scheme used in BlackHat and which will be used in this project. The different schemes are often related in ways that don't require extra loop calculations for example the 't Hooft-Veltman and four dimensional helicity scheme amplitudes for a pure gluon amplitude differ by a factor of

$$A^{HV} = A^{FDH} - \frac{\Gamma(1+\epsilon)\Gamma(1-\epsilon)}{3(4\pi)^{2-\epsilon}}\mu^{2\epsilon}A^{\text{tree}} \tag{1.2}$$

where A^{HV} is the amplitude in the 't Hooft-Veltman scheme, A^{FDH} is the amplitude in the four dimensional helicity scheme, A^{tree} is the tree amplitude and μ is a renormalisation scheme used to preserve the dimension of the amplitude.

It has already been shown that the amplitudes for any number of gluons in the Maximally Helicity Violating configuration are the very simple Parke-Taylor amplitudes[4]. In the Feynman diagram method, even for four gluons, the calculation contains three different diagrams. When the number of particles in an amplitude is increased, the number of terms increases factorially. This greatly limits the number of particles that can be included before the calculation of the amplitude becomes too large to effectively implement. At 1-loop these problems become worse and even simple four particle amplitudes can become difficult to perform. Therefore, the Feynman diagram method and explicit loop integrations are not efficient for these types of amplitudes and more efficient methods are needed.

The BlackHat library[5, 6, 7] already includes calculations for amplitudes with quarks, gluons and optionally one of the vector bosons, $W^{\pm}$ or Z, and other processes but does not currently include amplitudes for the Higgs boson. This thesis describes the methods needed to efficiently implement the calculation of the NLO 1-loop amplitude for Higgs boson with jets, in a generic way, for any number of quarks and gluons of any helicities. Once the calculations have been implemented into the BlackHat library it will be a very useful tool for adding NLO calculations into existing Monte Carlo Event Generators such as Herwig++ or Sherpa.

Following this introduction, Chapter 2 gives a short review of the techniques of colour ordered amplitudes and the spinor helicity formalism. These form the basis of the techniques used in the remaining chapters. The notations used in this thesis are also introduced in this chapter. Chapter 3 explores the BCFW recursion relation which is used to compute tree level amplitudes and discusses how it extends to amplitudes with a Higgs boson.

The next two chapters discuss the techniques used to calculate the loop amplitude. Firstly, Chapter 4 demonstrates how to calculate the cut constructable parts using Generalised Unitarity. Explicit numerical formulas are derived to enable a systematic numeric calculation to be implemented and this is extended to support amplitudes with a Higgs boson. Chapter 5 explores how to calculate the rational terms of amplitudes including a Higgs boson, using 6 dimensional spinors. After deriving a 6 dimensional extension of the spinor helicity formalism, tree amplitudes are calculated using it and simplified to allow an efficient calculation of the contributions needed for 6 dimensional Generalised Unitarity. Chapter 6 discusses how the calculation has been implemented, how to use the implementations and where possible, how BlackHat has been tested against the implementations developed in this project. Finally, Chapter 7 concludes the project and discusses the remaining steps needed to fully implement these calculations into BlackHat.

Chapter 2

An Introduction to Efficient Calculation Techniques

There are several techniques that can be used to efficiently calculate amplitudes which separate the amplitudes into simpler parts. The first method used is colour ordered amplitudes which separates the amplitudes' kinematics from the colour factors and splits the amplitudes into simpler "colour ordered" amplitudes. The second is to work in the spinor helicity formalism which separates the different helicity states and produces simpler formulas for different cases within the amplitude. Both of these techniques are discussed based on the versions used by Dixon[8]. A decomposition of the Higgs boson into a complex scalar is also introduced here which separates amplitudes into MHV like amplitudes. Finally, recursive calculations using unitarity techniques are introduced, which form the bedrock for the techniques discussed in the following two chapters.

2.1 Colour Ordered Amplitudes

Inclusion of colour factors in numerical calculations can greatly increase their complexity. If colour factors are included an amplitude is increased from a single complex number to a large tensor of complex numbers in colour space. Many of the elements in the matrix are zero and many of the rest are related, so it should be possible to extract and then combine elements to give a simpler form which has less redundancy and therefore leads to a more efficient calculation. One way this recombination can be done is by expanding an amplitude as the coefficients of different colour terms. Each of these coefficients would be a single complex number. The modulus squared of any specific type of amplitude, as used in the cross section, can be calculated from the coefficients, the number of colours and the number of quarks.

To reduce an amplitude to the smallest possible set of colour terms requires converting all colour factors to chains of the fundamental generators with no internal colour or gluon indices. This reduction is possible as the gluon colour factor, f^{abc}, which is the structure constant for the fundamental generators, can be written as[8]

$$f^{abc} = -\frac{i}{\sqrt{2}}\left(\mathrm{tr}[T^aT^bT^c] - \mathrm{tr}[T^aT^cT^b]\right) \ , \tag{2.1}$$

where T^a is the fundamental generator which is normalised as[8]

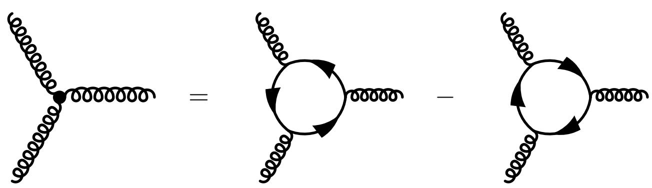
$$\mathrm{tr}[T^aT^b] = \delta^{ab} \ . \tag{2.2}$$

To remove internal gluon indices the Fierz identity for the fundamental generators,[8]

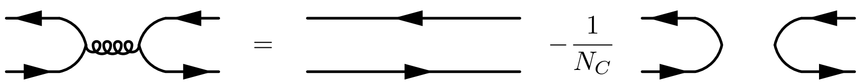
$$T^a{}_i{}^j T^a{}_{\bar{i}}{}^{\bar{j}} = \delta_i{}^{\bar{j}}\delta_{\bar{i}}{}^j - \frac{1}{N_c}\delta_i{}^j\delta_{\bar{i}}{}^{\bar{j}} \ , \tag{2.3}$$

is used. These relations can be used to convert the colour factor for any pure gluon amplitude into a sum of terms that are each a single trace of fundamental generators, one generator for each gluon. For amplitudes with quarks, the colour factor can be converted into a sum of terms that are each the product of a single (possibly empty) trace of fundamental generators and a chain of fundamental generators for each quark pair, where again each gluon's generator only appears once in each term.

The relations given in Equations 2.1 and 2.3 can be represented graphically in terms of replacements of quarks and gluons as[8]



(2.4)



(2.5)

where the rules only apply for the colour factors not kinematics.

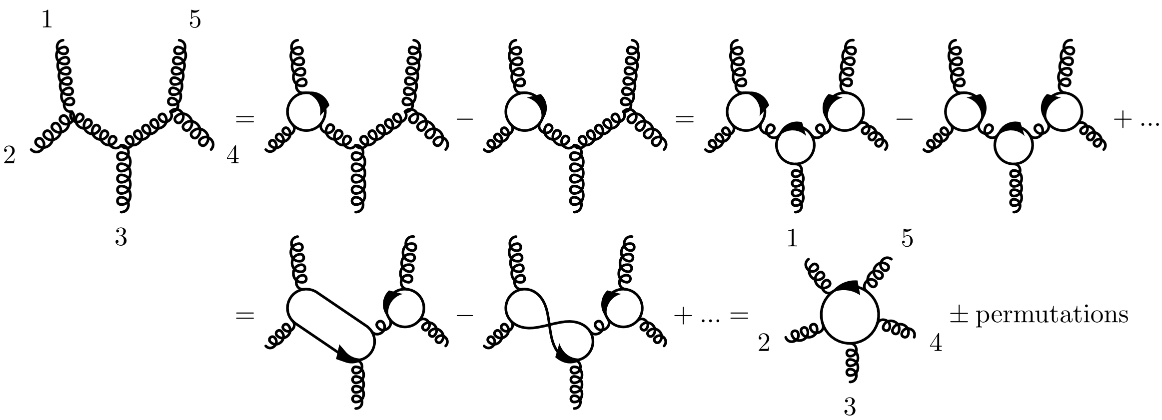


Figure 2.1: A tree level five gluon diagram along with its colour decomposition. The numbers label the gluons.

For example, the relations can be applied to the colour factor for the Feynman diagram shown in Figure 2.1, which is one of the diagrams contributing to the 5 gluon amplitude, which can be rearranged as

$$\begin{aligned}
\mathcal{D}_{5\ \text{gluon},tree} &= f^{a_1 a_2 b} f^{b a_3 c} f^{c a_4 a_5} D(1,2,3,4,5) \\
&= \left(-\frac{i}{\sqrt{2}}\right)^3 \left(\text{tr}[T^{a_1}T^{a_2}T^b] - \text{tr}[T^{a_1}T^bT^{a_2}]\right) \\
&\quad \left(\text{tr}[T^bT^{a_3}T^c] - \text{tr}[T^bT^cT^{a_3}]\right) \left(\text{tr}[T^cT^{a_4}T^{a_5}] - \text{tr}[T^cT^{a_5}T^{a_4}]\right) D(1,2,3,4,5) \\
&= \frac{i}{\sqrt{2}^3} \text{tr}[T^{a_1}T^{a_2}T^b]\,\text{tr}[T^bT^{a_3}T^c]\,\text{tr}[T^cT^{a_4}T^{a_5}] D(1,2,3,4,5) \pm \ldots \\
&= \frac{i}{\sqrt{2}^3} [T^{a_1}{}_i{}^j T^{a_2}{}_j{}^k] \delta_k{}^l \delta_n{}^i [T^{a_3}{}_l{}^m T^c{}_m{}^n]\,\text{tr}[T^cT^{a_4}T^{a_5}] D(1,2,3,4,5) \pm \ldots \\
&= \frac{i}{\sqrt{2}^3} [T^{a_1}{}_i{}^j T^{a_2}{}_j{}^k][T^{a_3}{}_k{}^m T^c{}_m{}^i]\,\text{tr}[T^cT^{a_4}T^{a_5}] D(1,2,3,4,5) \pm \ldots \\
&= \frac{i}{\sqrt{2}^3} \text{tr}[T^{a_1}T^{a_2}T^{a_3}T^c]\,\text{tr}[T^cT^{a_4}T^{a_5}] D(1,2,3,4,5) \pm \ldots \\
&= \frac{i}{\sqrt{2}^3} \text{tr}[T^{a_1}T^{a_2}T^{a_3}T^{a_4}T^{a_5}] D(1,2,3,4,5) \pm \ldots \\
&= \frac{i}{\sqrt{2}^3} \sum_{\sigma \in S_5/Z_5} (-1)^\sigma\, \text{tr}[T^{a_{\sigma(1)}}T^{a_{\sigma(2)}}T^{a_{\sigma(3)}}T^{a_{\sigma(4)}}T^{a_{\sigma(5)}}] D(1,2,3,4,5)\ ,
\end{aligned} \tag{2.6}$$

where $D(1,2,3,4,5)$ contains the kinematic parts of the diagram, $\pm \ldots$ represents more terms that are not shown, the $\frac{1}{N_C}$ terms cancel between different permutations and therefore vanish, S_5/Z_5 is the set of all permutations of 5 items that are not the same under cycles and $(-1)^\sigma$ is the sign of the permutations which is $+1$ for an even number of exchanges and -1 for an odd number of exchanges. The diagrammatic

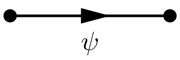
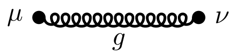
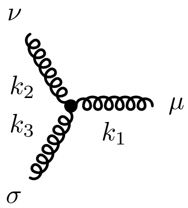
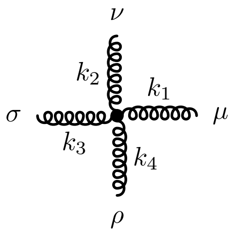
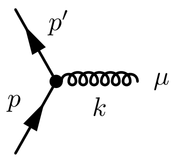
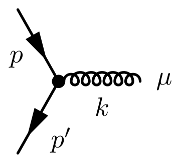
	$\frac{i}{\not{p}-m_\psi+i0}$		$\frac{i}{k^2+i0}\left(g^{\mu\nu}+(\xi-1)\frac{k^\mu k^\nu}{k^2}\right)$
	$-\frac{i}{\sqrt{2}}\left(g^{\mu\nu}(k_1-k_2)^\sigma+g^{\nu\sigma}(k_2-k_3)^\mu+g^{\sigma\mu}(k_3-k_1)^\nu\right)$		
	$ig^{\mu\sigma}g^{\nu\rho}-\frac{i}{2}(g^{\mu\nu}g^{\rho\sigma}+g^{\mu\rho}g^{\sigma\nu})$		
	$\frac{i}{\sqrt{2}}\gamma^\mu$		$-\frac{i}{\sqrt{2}}\gamma^\mu$

Table 2.1: Colour ordered Feynman rules in Faddeev-Popov gauge[8]. All momenta are inbound.

representation of the rearrangement is shown in Figure 2.1.

As all diagrams can be replaced with terms of the form shown in the last line of Equation 2.6, the entire tree amplitude can be changed to terms of that form. The coefficients of each trace structure can be extracted, which gives the colour ordered partial amplitudes. By combining them, the full amplitude can be constructed from the colour ordered partial amplitudes as

$$\mathcal{A}_{5\ \mathrm{gluon},tree}=g^3\sum_{\sigma\in S_5/Z_5}\mathrm{tr}[T^{a_{\sigma(1)}}T^{a_{\sigma(2)}}T^{a_{\sigma(3)}}T^{a_{\sigma(4)}}T^{a_{\sigma(5)}}]A_{5\ \mathrm{gluon},tree}(\sigma(1),\sigma(2),\sigma(3),\sigma(4),\sigma(5))\,. \tag{2.7}$$

Rather than calculating the full amplitude directly and then extracting the coefficient for each colour structure, it is possible to extract the coefficients for the colour structures from the Feynman rules and then evaluate the colour ordered partial amplitudes. The modified Feynman rules, given in Table 2.1, enable the colour ordered amplitudes to be evaluated by summing the expressions for all possible planar diagrams with a given fixed order of the external particles. There are two vertices for

$q\bar{q}g$ which differ in the direction of the quark line and are no longer equivalent, as the graphs are planar and the external particles have a fixed order.

For any number of gluons the formula extends to[8]

$$\mathcal{A}_{n \text{ gluon},tree} = g^{n-2} \sum_{\sigma \in S_n/Z_n} \mathrm{tr}[T^{a_{\sigma(1)}}...T^{a_{\sigma(n)}}] A_{n \text{ gluon},tree}(\sigma(1),...,\sigma(n)) \ , \tag{2.8}$$

where S_n/Z_n is the set of permutations of n items that are not equivalent under cycles and $A_{n \text{ gluon},tree}$ is a colour ordered amplitude. If there are external fermions then there will be chains of fundamental generators which connect with the quark colour indices. For example, with one pair of external fermions the amplitude takes the form[8]

$$\mathcal{A}_{n-2 \text{ gluon},2 \text{ quark},tree} = \\ g^{n-2} \sum_{\sigma \in S_{n-2}} [T^{a_{\sigma(3)}}...T^{a_{\sigma(n)}}]_i^{\ j} A_{n-2 \text{ gluon},2 \text{ quarks},tree}(1_p, 2_{\bar{p}}, \sigma(3),...,\sigma(n)) \ . \tag{2.9}$$

The same method applies for 1 loop calculations where, for example, the n gluon amplitude is given by[8]

$$\mathcal{A}_{n \text{ gluon},1-loop} = g^n \left(\sum_{\sigma \in S_n/Z_n} N_c \, \mathrm{tr}[T^{a_{\sigma(1)}}...T^{a_{\sigma(n)}}] A_{n;1}(\sigma(1),...,\sigma(n)) \right. \\ \left. \sum_{c=2}^{\lfloor \frac{n}{2} \rfloor +1} \sum_{\sigma \in S_n/S_{n;c}} \mathrm{tr}[T^{a_{\sigma(1)}}...T^{a_{\sigma(c-1)}}] \, \mathrm{tr}[T^{a_{\sigma(c)}}...T^{a_{\sigma(n)}}] A_{n;c}(\sigma(1),...,\sigma(n)) \right) , \tag{2.10}$$

where $S_n/S_{n;c}$ is the set of all permutations of n items that do not give the same trace terms, $\lfloor n \rfloor$ is the largest integer less than or equal to n and the $A_{n;c}$ are partial amplitudes. The $A_{n;1}$ are colour ordered partial amplitudes and are called primary amplitudes. The other partial amplitudes are not colour ordered. If there are only external gluons, these other partial amplitudes can be written as a sum of permutations of $A_{n;1}$. If there are external quarks, the $A_{n;c>1}$ can no longer be written in terms of only the $A_{n;1}$, but will need new colour ordered partial amplitudes which are defined by specifying the direction the quark travels around the loop. Again this decomposition can be performed diagrammatically and is shown for a two gluon and two fermion one loop diagram in Fig. 2.2.

The Higgs boson is a completely colourless particle, as are the W and Z bosons

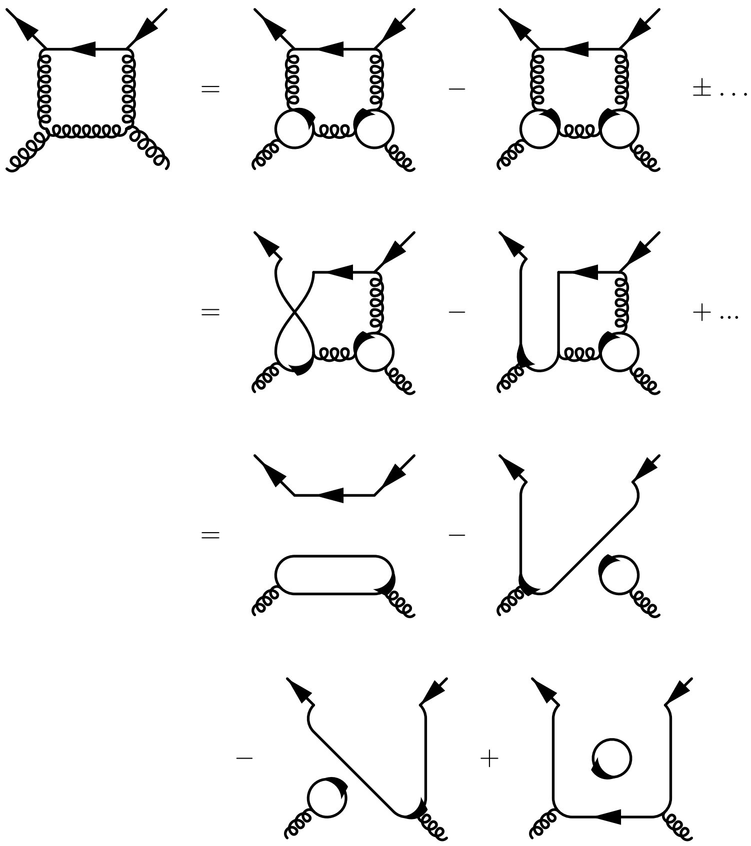


Figure 2.2: The colour decomposition of a one loop, two fermion and two gluon diagram. All $\frac{1}{N_c}$ contributions cancel at all stages.

which are already implemented into BlackHat. These particles do not have any colour factors attached and as such are not involved in any colour rearrangements. The Higgs boson can therefore be introduced into amplitudes without changing the colour ordering techniques and its ordering in partial amplitudes is not fixed, meaning that any Higgs boson must be included in all possible positions. The Higgs boson can therefore be included into colour ordered amplitudes without any extra complications.

2.2 Spinor Helicity Formalism

Spinors are defined as the solution of the equation

$$\not{p} u_p = 0 \ , \tag{2.11}$$

where $\not{p} = \gamma_\mu p^\mu$ and γ_μ are the gamma matrices. The solutions are normalised to

$$u \otimes \overline{u} = \not{p} \ , \tag{2.12}$$

where $\overline{u}$ is the conjugate spinor for u and $\otimes$ is an inner product summing over the different spinor states. The conjugate spinors are solutions to

$$\overline{u}_p \not{p} = 0 \ , \tag{2.13}$$

and for real momenta are related to the normal spinor by

$$\overline{u}_p = u_p^\dagger \gamma_0 \ . \tag{2.14}$$

In order to avoid problems with extending the definition of conjugate spinors to complex momenta, conjugate spinors defined without needing conjugation will be used for this project.

For 4 dimensions there is one degree of freedom left over after the spinors are normalised so there are two basis states in the spinor space. It was discovered that for massless QCD amplitudes are very simple when the basis states are helicity eigenstates. For massless spinors the helicity operator is defined by

$$\gamma^5 = i\gamma^0\gamma^1\gamma^2\gamma^3 \ . \tag{2.15}$$

The spinor eigen states are defined as

$$u_\pm(k) = P^\pm u(k) = \frac{1 \pm \gamma_5}{2} u(k) \qquad v_\mp(k) = P^\pm v(k) = \frac{1 \pm \gamma_5}{2} v(k) \ , \tag{2.16}$$

where $P^\pm$ is the projection operator for γ_5 and is defined as

$$P^\pm = \frac{1 \pm \gamma_5}{2} \ . \tag{2.17}$$

From the completeness relation for massless spinors

$$u(k) \otimes \bar{u}(k) = v(k) \otimes \bar{v}(k) = \not{k} \ , \tag{2.18}$$

it can be seen that the spinors for v and u are proportional to each other and as an overall phase is arbitrary the spinors can be defined such that $u_{\pm}(k) = v_{\mp}(k)$.

A more compact bra-ket notation can defined which is given by[8]

$$\begin{aligned} u^{+}(k_i) = v^{-}(k_i) \equiv \left|k_i^{+}\right\rangle \equiv |i\rangle \quad u^{-}(k_i) = v^{+}(k_i) \equiv \left|k_i^{-}\right\rangle \equiv |i] \\ \bar{u}^{+}(k_i) = \bar{v}^{-}(k_i) \equiv \left\langle k_i^{+}\right| \equiv [i| \quad \bar{u}^{-}(k_i) = \bar{v}^{+}(k_i) \equiv \left\langle k_i^{-}\right| \equiv \langle i| \ , \end{aligned} \tag{2.19}$$

where the sign of the bra-ket type spinor is defined by the eigen value under the helicity operator. Some combinations of these spinors have products that are zero due to being opposite helicity eigen states, these products are $[1|2\rangle$ and $\langle 1|2]$. As opposite sign projection operators annihilate and the two spinors in the product are projected with opposite sign projection operators, these products vanish. As with all massless spinors, a product of any two of these spinors for the same momentum vanishes. The remaining spinor products can be combined to form the relation[8]

$$\langle ij\rangle\,[ji] = \mathrm{Tr}[\frac{1-\gamma^5}{2}\not{k}_i\not{k}_j] = 2k_i.k_j = (k_i+k_j)^2 = s_{ij} \ , \tag{2.20}$$

which is derived using the completion relation for spinors, the anti-commutation relations for gamma matrices and, as the momenta are massless, $k_i^2 = k_j^2 = 0$. These spinor products are anti-symmetric so obey $\langle ij\rangle = -\langle ji\rangle$ and $[ij] = -[ji]$.

Gluon polarisation vectors can be written in terms of spinors as[8]

$$\epsilon_\mu^{\pm}(k;q) = \pm\frac{\langle k^{\pm}|\gamma_\mu|q^{\pm}\rangle}{\sqrt{2}\,\langle k^{\mp}q^{\pm}\rangle} \ , \tag{2.21}$$

or explicitly for each state

$$\epsilon_\mu^{+}(k;q) = \frac{[k|\gamma_\mu|q\rangle}{\sqrt{2}\,\langle kq\rangle} \ , \qquad \epsilon_\mu^{-}(k;q) = \frac{\langle k|\gamma_\mu|q]}{\sqrt{2}\,[qk]} \ , \tag{2.22}$$

where the vector, q, can be arbitrarily chosen independently for each gluon but must not be changed between uses of the same gluon and corresponds to a gauge choice. It can be shown to be a gauge by checking that a change in $q \to q'$ is proportional to the vector k,

$$\begin{aligned}
\epsilon^+_\mu(k;q') - \epsilon^+_\mu(k;q) &= \frac{[k|\gamma_\mu|q'\rangle}{\sqrt{2}\,\langle kq'\rangle} - \frac{[k|\gamma_\mu|q\rangle}{\sqrt{2}\,\langle kq\rangle} \\
&= \frac{(-\langle qk\rangle)\,[k|\gamma_\mu|q'\rangle - (-\langle q'k\rangle)\,[k|\gamma_\mu|q\rangle}{2\,\langle kq'\rangle\,\langle kq\rangle} \\
&= \frac{-\langle q|k|\gamma_\mu|q'\rangle + \langle q'|k|\gamma_\mu|q\rangle}{2\,\langle kq'\rangle\,\langle kq\rangle} \\
&= \frac{\langle q'|\gamma_\mu|k|q\rangle + \langle q'|k|\gamma_\mu|q\rangle}{2\,\langle kq'\rangle\,\langle kq\rangle} \\
&= \frac{\langle q'\,(\gamma_\mu k + k\gamma_\mu)\,q\rangle}{2\,\langle kq'\rangle\,\langle kq\rangle} \\
&= \frac{\langle q'q\rangle}{2\,\langle kq'\rangle\,\langle kq\rangle} 2k^\mu \\
&= \frac{\langle q'q\rangle}{\langle kq'\rangle\,\langle kq\rangle} k^\mu \ ,
\end{aligned} \tag{2.23}$$

where in the first few lines the properties of spinor products have been used, in the later lines the anti-commutation relations for slashed matrices have been used and the same method applies for the transformation of the other helicity polarisation vector. It is also possible to check that these expressions obey all the normal relations for polarisation vectors such as

$$\epsilon^\pm_\mu(k;q)k^\mu = \pm\frac{\langle k^\pm|k|q^\pm\rangle}{\sqrt{2}\,\langle k^\mp q^\pm\rangle} = 0 \ , \tag{2.24}$$

and

$$\begin{aligned}
\epsilon^+_\mu(k;q)\epsilon^{-\mu}(k;q) &= \frac{[k|\gamma_\mu|q\rangle}{\sqrt{2}\,\langle kq\rangle}\frac{\langle k|\gamma^\mu|q]}{\sqrt{2}\,[qk]} \\
&= \frac{2\,[kq]\,\langle kq\rangle}{2\,\langle kq\rangle\,[qk]} \\
&= -1 \ ,
\end{aligned} \tag{2.25}$$

where identities from [8] have been used, which are not shown or proved here. Due to the ability to choose the reference vector freely for each gluon and the many spinor products that are 0, it is often possible with a clever choice of reference vectors to vastly simplify the calculation of amplitudes and to cause many of the diagrams contributing to an amplitude to vanish.

Due to the simple relations between spinor products, many of which are 0, very simple expressions for the amplitudes can be found and as shown by Parke and Taylor the pure gluon amplitudes with all helicities the same or all but one the same are

0. They found the first non-zero amplitude to be the so called Maximally Helicity Violating amplitude which has all but two gluons having the same helicity. The mostly positive MHV amplitude for any number of gluons is given by[8]

$$A^{\text{tree}}_{n\ \text{gluon}}(1^-, 2^+, \ldots, i-1^+, i^-, i+1^+, \ldots, n^+) = i\frac{\langle 1i\rangle^4}{\langle 12\rangle\langle 23\rangle\ldots\langle n1\rangle}\ , \tag{2.26}$$

where one of the negative helicity gluons can be chosen to be labelled as 1 due to cyclic symmetry of colour ordered amplitudes. This simple form is the same for any number of gluons as long as they are in the MHV form, in contrast with amplitudes in other systems where the amplitude increases enormously in complexity as the number of particles increases. There is also an equivalent formula for the mostly negative case.

The amplitudes with one quark line are also found to be very simple in the MHV cases, which for these amplitudes is when all but one gluon have the same helicity. The mostly positive amplitude for this type of amplitude is given by[8]

$$A^{\text{tree}}_{n-2\ \text{gluon},\ q\bar{q}}(1^-, \ldots, i_q^+, \ldots, j_{\bar{q}}^-, \ldots, n^+) = i\frac{\langle 1i\rangle^3\langle 1j\rangle}{\langle 12\rangle\langle 23\rangle\ldots\langle n1\rangle}\ , \tag{2.27}$$

where again the choice has been made to label the negative helicity gluon as 1. The quark anti-quark pair must have opposite helicity as QCD conserves helicity along quarks lines and all momenta and helicities are taken as for outgoing particles.

For other cases the amplitudes are often not as simple, but they are still significantly simpler than in other forms and many cases have a value of zero.

2.3 Higgs Boson as part of a Complex Scalar Field

A further simplification of calculations for amplitudes involving a Higgs boson can be performed by defining a complex scalar field[9]

$$\phi = \frac{1}{2}\left(H + iA\right)\ , \tag{2.28}$$

where A is a real pseudo-scalar field which interacts with an effective vertex of

$$\mathcal{L}^{\text{int}}_A = \frac{C}{2} iA\,\text{tr}\, G_{\mu\nu}{}^{*}G^{\mu\nu}\ , \tag{2.29}$$

where ${}^*G^{\mu\nu}$ is the dual of the gluon field strength tensor and is given by

$$ {}^*G^{\mu\nu} = \frac{i}{2}\epsilon^{\mu\nu\sigma\rho}G_{\sigma\rho} \ . \tag{2.30} $$

The interaction terms for the pseudo-scalar, A, and for the Higgs boson, H, combine to produce an interaction term for the complex field ϕ of[9]

$$ \mathcal{L}_{\phi}^{\text{int}} = C\left(\phi \,\text{tr}\, G_{\text{SD}\mu\nu}G_{\text{SD}}^{\mu\nu} + \phi^{\dagger}\,\text{tr}\, G_{\text{ASD}\mu\nu}G_{\text{ASD}}^{\mu\nu}\right) \ , \tag{2.31} $$

where G_{SD} and G_{ASD} are the self dual and anti self dual gluon field strength tensors and are given by[9]

$$ G_{\text{SD}}^{\mu\nu} = \frac{1}{2}\left(G^{\mu\nu} + {}^*G^{\mu\nu}\right) \qquad\qquad G_{\text{ASD}}^{\mu\nu} = \frac{1}{2}\left(G^{\mu\nu} - {}^*G^{\mu\nu}\right) \ . \tag{2.32} $$

Working with the complex field ϕ rather than directly with the Higgs field H is particularly useful when working with helicity amplitudes and the spinor helicity formalism as the helicity structures do not mix and the amplitudes are closely related to their pure quark and gluon equivalents. For example the mostly positive MHV amplitude with a ϕ is given by[9]

$$ A^{\text{tree}}_{\phi,n\text{ gluon}}(\phi, 1^-, 2^+, \ldots, i-1^+, i^-, i+1^+, \ldots, n^+) = i\frac{\langle 1i\rangle^4}{\langle 12\rangle\langle 23\rangle\ldots\langle n1\rangle} \ , \tag{2.33} $$

where the form of this amplitude is identical to the form of the amplitude without the ϕ, which is given in Equation 2.26. The presence of the ϕ particle does impact the mostly negative amplitudes, as it causes them not to vanish. The all negative amplitude is given by[9]

$$ A^{\text{tree}}_{\phi,n\text{ gluon}}(\phi, 1^-, 2^-, \ldots, n^-) = (-1)^n i\frac{m_H^4}{[12]\,[23]\ldots[n1]} \ , \tag{2.34} $$

where m_H is the mass of the Higgs boson.

Amplitudes involving the conjugate field $\phi^{\dagger}$ do not need to be calculated, as amplitudes involving it are related to amplitudes involving ϕ by a parity transformation, which gives the relation[9]

$$ A_n(\phi^{\dagger}, 1^{-h}, \ldots, n^{-l}) = (-1)^{n_{q\bar{q}}}\, A_n(\phi, 1^h, \ldots, n^l)\big|_{\langle ij\rangle\leftrightarrow[ji]} \ , \tag{2.35} $$

where $n_{q\bar{q}}$ is the number of $q\bar{q}$ pairs in the amplitude and $\langle ij\rangle \leftrightarrow [ji]$ corresponds to complex conjugation of the amplitude if all momenta are real. Therefore when working in 4 dimensions, as is being done in Chapters 3 and 4, all Higgs boson amplitudes will be calculated as amplitudes in terms of ϕ and will then be combined afterwards.

2.4 Recursive Construction of Amplitudes

With colour ordered amplitudes and the spinor helicity formalism, amplitudes can often be written in simple forms but for an automated system a method is required to build amplitudes, systematically, from simpler building blocks. The traditional method is the Feynman diagram method where amplitudes are built by summing Feynman rules over different diagrams. This method includes all possible information about a process, including handling off-shell particles in a fully general way, but for producing amplitudes this information is unnecessary. The information about off-shell particles, along with the gauge redundancy that is in the Feynman rules but not in any final amplitude, increases the complexity of this method. An improvement would be to build amplitudes from lower multiplicity, already on-shell and numeric amplitudes and this is exactly what is done in the Blackhat library and in this project. The BCFW recursion relation is used to build tree amplitudes from lower multiplicity tree amplitudes and then Generalised Unitarity is used to build loop amplitudes from tree amplitudes. The next few chapters will explain and derive these methods and extend for the Higgs boson.

Both the BCFW recursion relation and Generalised Unitarity methods rely on unitarity. This is the statement that the residue of poles in amplitudes are the product of two lower multiplicity amplitudes up to a sum over internal particles. To derive this, conservation of probability needs to be applied to the S matrix. The S matrix is the matrix that contains information about the probability of an interaction and is defined in terms of the amplitude matrix as

$$S = 1 + iA \;, \tag{2.36}$$

where A is the matrix of amplitudes. If probability is conserved then $SS^\dagger = 1$.

Substituting Equation 2.36 into this condition results in

$$\begin{aligned}
1 &= (1+iA)(1-iA^\dagger) \\
&= 1 + i(A - A^\dagger) - AA^\dagger \\
AA^\dagger &= i(A - A^\dagger)\ ,
\end{aligned} \tag{2.37}$$

which tells us that the imaginary part of an amplitude is related to the product of lower multiplicity amplitudes. From the definition of amplitudes it can be seen that the only imaginary part will come from the $i\epsilon$ terms in propagators and will only contribute when the propagator goes on-shell. From this it is clear that the residue of an amplitude at a pole is proportional to a product of tree amplitudes.

Chapter 3

BCFW Recursion Relation

In 2005, Britto, Cachazo, Feng and Witten[10] discovered a recursion relation that relates an on-shell amplitude to lower multiplicity, on-shell amplitudes, which is called the BCFW recursion relation. This recursion relation can be applied to any amplitude, from any theory, but is particularly useful for calculations using the Spinor Helicity Formalism, where the amplitudes being calculated have a helicity structure, for reasons that will become apparent through this chapter. The BCFW recursion relation works by selecting two momenta from the external momenta in the amplitude and shifting them by an arbitrary complex amount, z, of a fixed vector n^μ, such that the two shifted momenta stay on-shell and that momentum conservation is preserved. The shift is given by

$$p_i^\mu \to \hat{p}_i^\mu(z) = p_i^\mu + zn^\mu \qquad\qquad p_j^\mu \to \hat{p}_j^\mu(z) = p_j^\mu - zn^\mu \ , \tag{3.1}$$

where $p_{i/j}$ are the two selected original momenta and $\hat{p}_{i/j}$ are the corresponding shifted momenta. To ensure the shifted momenta stay on-shell it is necessary to choose the shift vector n^μ such that it satisfies the conditions,

$$2n_\mu p_i^\mu + zn^2 = -2n_\mu p_j^\mu + zn^2 = 0 \tag{3.2}$$

and as n and $p_{i/j}$ are independent of z, this relation must be true at all orders in z and the conditions simplify to

$$n_\mu p_i^\mu = n_\mu p_j^\mu = n^2 = 0 \ , \tag{3.3}$$

The solution to these conditions used for the BCFW relation is given by

$$n^\mu = \langle i\gamma^\mu j] \ , \tag{3.4}$$

which satisfies the conditions explicitly. This shift is especially useful for calculations using the Spinor Helicity Formalism as it corresponds to a shift of the spinors given by[10]

$$|i\rangle \to |\hat{i(z)}\rangle = |i\rangle + z\,|j\rangle \qquad\qquad |j] \to |\hat{j(z)}] = |j] - z\,|i] \ , \tag{3.5}$$

where all the other spinors are unchanged by the shift.

When the shift is applied the resulting amplitude, $\hat{A}(z)$, is a function of the complex parameter z. The recurrence relation is derived by performing a contour integration of $\hat{A}(z)/z$, around a circle of radius r, and taking the limit of $r \to \infty$. In this limit the Cauchy residue theorem shows that the integral is given by

$$\lim_{a\to\infty} \oint_{|z|=a} \frac{\hat{A}(z)}{z} dz = 2\pi i \left(\hat{A}(0) + \sum_{z_0} \frac{\mathrm{Res}_{z=z_0}\, \hat{A}(z)}{z_0} \right) \ , \tag{3.6}$$

where the sum is over every pole, z_0, in the shifted amplitude. By the analyticity properties of amplitudes, the only poles in the shifted amplitude will be due to internal propagators going on-shell and will be single poles whose residues, by the optical theorem, are shown to be proportional to the product of two lower multiplicity, on-shell amplitudes. The form of the pole in the amplitude expanded around the location of the pole is given by

$$\begin{aligned}
\hat{A}(z) &= \frac{\hat{A}_l(z)\hat{A}_r(z)}{\hat{P}(z)^2} + \tilde{A}(z) \\
&= \frac{\hat{A}_l(z)\hat{A}_r(z)}{(P+zn)^2} + \tilde{A}(z) \\
&= \frac{\hat{A}_l(z)\hat{A}_r(z)}{P^2 + 2zP\cdot n} + \tilde{A}(z) \\
&= \frac{\hat{A}_l(z)\hat{A}_r(z)}{2P\cdot n} \frac{1}{z - \frac{-P^2}{2P\cdot n}} + \tilde{A}(z) \ ,
\end{aligned} \tag{3.7}$$

where $\hat{A}_{l/r}(z)$ are the two amplitudes either side of the selected propagator, $\hat{P}(z)$ is the shifted momentum in the propagator, n is the shift vector, $P = \sum p$ is the unshifted momentum of the selected propagator made up of a sum of external momenta including p_i but not p_j and $\tilde{A}(z)$ is the rest of the amplitude that does not have any poles at $z \to z_0$. Without loss of generality, the left amplitude and the propagator momentum can be taken to include p_i and not p_j. This is because exchanging the left and right sides will give the same terms again and including both would be a double counting. The remaining cases are when both p_i and p_j are on the same side of the propagator, which causes the z dependence to cancel and hence such terms will not contribute. Taking the residue of this pole and substituting it into the equation for the integral produces the result

$$\begin{aligned} A = \hat{A}(0) &= -\sum_m \frac{A_{l,m}A_{r,m}}{2P_i \cdot n z_{0,m}} - \frac{1}{2\pi i}\lim_{a\to\infty}\oint_{|z|=a}\frac{\hat{A}(z)}{z}dz \\ &= -\sum_m \frac{A_{l,m}A_{r,m}}{2P_m \cdot n \frac{-P_m^2}{2P_m\cdot n}} - \frac{1}{2\pi i}\lim_{a\to\infty}\oint_{|z|=a}\frac{\hat{A}(z)}{z}dz \\ &= \sum_m \frac{A_{l,m}A_{r,m}}{P_m^2} - \frac{1}{2\pi i}\lim_{a\to\infty}\oint_{|z|=a}\frac{\hat{A}(z)}{z}dz \ , \end{aligned} \tag{3.8}$$

where the sum is over the poles labelled by m, and $A_{l/r,m} = \hat{A}_{l/r}\left(\frac{-P_m^2}{2P_m\cdot n}\right)$ are the on-shell amplitudes resulting from splitting the diagram at the propagator labelled by m and evaluating for the shift parameter $z = \frac{-P_m^2}{2P_m\cdot n}$. If the integral can be shown to be zero then the BCFW recursion relation results. To do this without explicit integration or using the residue theorem, power counting is used. In some cases power counting can show us that the integral is zero, for all other cases the integral's value cannot be determined in this way. For pure gluon amplitudes this gives the condition that[10]

$$h_i = + \text{ or } h_j = - \text{ or both} \ , \tag{3.9}$$

where $h_{i/j}$ are the helicities of the gluons i and j respectively. This is equivalent to saying that the helicities for the shifted gluons (h_i, h_j) must be either $(+,+)$, $(-,-)$ or $(-,+)$ but not $(+,-)$. If a shift needs to be performed on a pair of particles with helicities $(+,-)$ then the shift can be performed by swapping the roles of the gluons in the shift or equivalently by swapping $|\cdot] \to |\cdot\rangle$ in the definition of the shift.

For MHV amplitudes it is easy to see that this condition is the correct condition. The amplitude only depends on $|\cdot\rangle$ spinors so the shift only has an effect through the spinor $|i\rangle$ being replaced with $|i\rangle + z|j\rangle$. If the gluon i has helicity $+$, then this spinor is only present in the amplitude in the denominator and as such, in the limit $|z| \to \infty$, the shifted amplitude will be proportional to $1/z^2$ or, if i and j are neighbouring particles, $1/z$ and the amplitude will vanish. For the case of a negative helicity for gluon i, if the other gluon shifted is the other negative helicity gluon, then the numerator has the form $\langle ij\rangle^4 \to (\langle ij\rangle + z\langle jj\rangle)^4 = \langle ij\rangle^4$ which is unchanged, so again the only contribution from the shift is in the denominator and the amplitude will vanish in the limit $|z| \to \infty$. The remaining case, which is the amplitude that should not vanish, is that gluon i has negative helicity and j has positive helicity. In this case the numerator has the form $\langle ik\rangle^4 \to (\langle ik\rangle + z\langle jk\rangle)^4$, where gluon k is the other negative helicity gluon. As $|z| \to \infty$ this form tends to ∞ and the one or two powers of z in the numerator can not save the amplitude from tending to ∞.

To work out the trend for a general amplitude is more complex and requires balancing the powers of z from the numerator and denominator of an amplitude. For a general pure gluon amplitude, from the Feynman rules, it can be seen that the contributions dependant on z in any diagram can only be from the single route through the diagram, from one of the shifted particles to the other one. The contributions along this route will be from propagators and from three particle vertices. The form of a gluon propagator from the internal momenta, P, has the form

$$\frac{1}{\hat{P}^2} = \frac{1}{P^2 + zP\cdot n} \xrightarrow[|z|\to\infty]{} \frac{1}{P\cdot n}\frac{1}{z} = 0 \ , \tag{3.10}$$

where P was chosen to contain i and therefore not j, and the form of a quark propagator for the internal momenta, P, has the form

$$\frac{\hat{\not{P}}}{\hat{P}^2} = \frac{\not{P} + z\left(|j\rangle\,[i| + |i]\,\langle j|\right)}{P^2 + zP\cdot n} \xrightarrow[|z|\to\infty]{} \frac{\not{P}}{P\cdot n}\frac{1}{z} + \frac{|j\rangle\,[i| + |i]\,\langle j|}{P\cdot n} = 0 \ . \tag{3.11}$$

The form of a three gluon vertex is

$$\begin{aligned} V_{3\ \mathrm{gluon}\ k_1 k_2 k_3}^{\ \mu\ \nu\ \sigma} &= \frac{1}{\sqrt{2}}\left(z\left(2g^{\mu\nu}n^\sigma - g^{\nu\sigma}n^\mu - g^{\sigma\mu}n^\nu\right) + \right. \\ &\left. g^{\mu\nu}\left(k_1 - k_2\right)^\sigma + g^{\mu\nu}\left(k_2 - k_3\right)^\mu + g^{\mu\nu}\left(k_3 - k_1\right)^\nu\right) \ , \end{aligned} \tag{3.12}$$

where k_1 contains i and k_2 contains j and k_3 therefore contains no z dependence. From this it can be seen that for any diagram the propagator and vertex combination will depend on z at the worst as $\sim z$ and possibly as constant or lower powers of z if some of the numerator terms vanish. The polarisation vectors for i and j can each either contribute $1/z$ or z, depending on their helicities, as they take the forms

$$\hat{\epsilon}^{+}_{\mu}(i;q) = \frac{[i|\gamma_\mu|q\rangle}{\sqrt{2}\left(\langle iq\rangle + z\,\langle jq\rangle\right)} \sim \frac{1}{z}\frac{[i|\gamma_\mu|q\rangle}{\sqrt{2}\,\langle jq\rangle} \tag{3.13}$$

$$\hat{\epsilon}^{-}_{\mu}(i;q) = -\frac{\langle i|\gamma_\mu|q] + z\,\langle j|\gamma_\mu|q]}{\sqrt{2}\,[iq]} \sim -z\frac{\langle j|\gamma_\mu|q]}{\sqrt{2}\,[iq]} \tag{3.14}$$

$$\hat{\epsilon}^{-}_{\mu}(j;q') = -\frac{\langle j|\gamma_\mu|q']}{\sqrt{2}\left([jq] - z\,[iq']\right)} \sim \frac{1}{z}\frac{\langle j|\gamma_\mu|q']}{\sqrt{2}\,[iq']} \tag{3.15}$$

$$\hat{\epsilon}^{+}_{\mu}(j;q') = \frac{[j|\gamma_\mu|q'\rangle - z\,[i|\gamma_\mu|q'\rangle}{\sqrt{2}\,\langle jq'\rangle} \sim -z\frac{[i|\gamma_\mu|q'\rangle}{\sqrt{2}\,\langle jq'\rangle}\ . \tag{3.16}$$

From this it can be seen that if gluon i has helicity $+$ and gluon j has helicity $-$, then the amplitude will at most go as $\frac{1}{z}$ for $|z| \to \infty$ and the integral at infinity will vanish. For the other valid helicity cases, the leading terms seem to go as z, but these leading terms can be shown to vanish, leaving a correct leading term of $\frac{1}{z}$.

This logic can be extended to amplitudes of quarks using the same methods. For this project it also needs to be extended to amplitudes with a complex scalar, ϕ, in them, which requires showing that these amplitudes vanish in the limit of $|z| \to \infty$. It is easy to see that if the amplitudes without the ϕ vanish, the ones with it will also vanish, as the amplitudes are related and the amplitudes that completely vanish without the ϕ will all vanish for any choice of shift particles.

Chapter 4

Generalised Unitarity Method

The Generalised Unitarity method has been used by many people to derive analytical forms for various one loop amplitudes[11, 12, 13]. It is a method that allows one loop amplitudes to be built up from tree level amplitudes without directly performing any integrations.

Any one loop amplitude can be expanded in terms of a basis of the scalar one loop amplitudes. These amplitudes are those containing only scalar particles. It is only necessary to use the subset of these amplitudes with all particles in the loop being massless and only the external particles entering the loop having mass, as in this project all particles are massless, other than the Higgs boson or complex scalar, ϕ, neither of which can enter the loop as that would introduce higher powers of the effective coupling and not be NLO. These scalar one loop integrals are of the form[14]

$$I_n(P_1, \ldots, P_n) = \int \frac{d^D l}{i\pi^{\frac{D}{2}}} \frac{1}{(l)^2 \ldots (l - P_1 - \ldots - P_{n-1})^2} , \tag{4.1}$$

where n is the number of propagators in the loop which is the number of external scalars entering the loop, $P_1, \ldots, P_n$ are the momenta of the external scalars entering the loop which must sum to zero and l is the loop momenta. Any one loop integral with massless propagators can be written as a sum of these integrals multiplied by rational functions of spinor products and Minkowski products of the external momenta, as contributions that are divergent in the limit $d \to 4$ are all due to integrating propagators over the loop momentum. Therefore, when using this basis any contributions that depend on the nature of the particles in the amplitude will be included in coefficients of these integrals. This basis is chosen as it is one of the simplest possible sets and any amplitude will have a unique expansion in terms of this basis. Any given integral can be simplified and numerator terms removed using the Passarino-Veltman reduction to write them as combinations of metric tensors and loop momenta squared, which reduces the integral to a combination of the basis integrals. Then any amplitudes with more than four propagators can then be reduced to amplitudes with at most four propagators. In four dimensions only loop integrals with at most four propagators are needed. For more than five propagators a simple partial fractioning of the amplitude will reduce it to a sum of terms with at most five propagators multiplied by rational functions of the Minkowski products of external momenta, as the set of external momenta is linearly dependent. The remaining terms with five propagators can again be reduced to give a combination of terms with at most four propagators along with terms that contain enough explicit powers of the dimension to cancel any poles from the integration and as such do not contribute to the cut producible parts of the amplitude. After this reduction an amplitude will be written in the form of sums of scalar one loop integrals with at most four propagators multiplied by rational functions of spinors and Minkowski products of external momenta plus a term with no poles in ϵ where $d = 4 - 2\epsilon$ is the dimension of space time. This expansion is shown in Figure 4.1. The full set of these integrals have been evaluated explicitly as functions of the external momenta flowing into each vertex of the loop.

To evaluate any one loop amplitude, all that remains to be calculated are the coefficients of each of these scalar loop functions. These can be calculated by taking any amplitude and making use of various methods to reduce the amplitude to an explicit expansion in terms of these functions. However, this becomes intractable when the number of particles increases and is complex to implement numerically. For a more efficient method generalised unitarity can be used. This extends the unitarity method to multiple cuts for loop amplitudes. A cut is defined as the process of replacing a propagator by a delta function or more precisely of introducing the factor

$$2\pi \left(l - P\right)^2 \delta\left(\left(l - P\right)^2\right) \ , \tag{4.2}$$

where l is the loop momenta and $\frac{1}{(l-P)^2}$ is the propagator that is being cut. This factor will integrate to zero unless multiplied by a function containing exactly the propagator to which it corresponds. If cuts are performed on a loop amplitude it will split it into a sum of products of tree amplitudes due to the factorisation properties

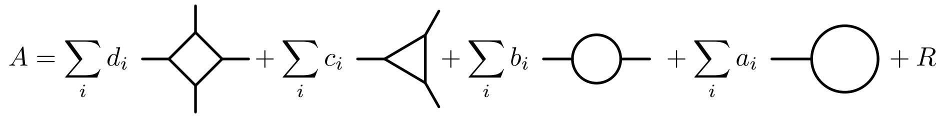


Figure 4.1: The expansion of an amplitude in terms of the scalar basis functions. The sums are over each possible scalar loop function of that form, defined by the momenta at each corner and R contains all contributions that do not contain poles in ϵ.

of amplitudes. The same cuts, when performed on the expansion, explicitly isolate coefficients of different loop functions. Four cuts will isolate the coefficient of a single box integral, as no integrals have more than four propagators and no other scalar loop functions will be extracted by this set of cuts because any other scalar loop integral will not contain all four of the propagators. Therefore, at least one of the cuts will not be matched by a propagator and will vanish when integrated. By performing each possible set of four cuts, all box coefficients can be extracted. For three cuts, in the expansion in terms of scalar loop functions, a single triangle function will be isolated, but there will also be contributions from box functions. However, triangle and lower cuts are not required for boxes to be extracted, so their contributions can be removed before attempting to calculate triangle coefficients. This same process applies for one or two cuts and allows us to isolate each coefficient in turn.

After applying three cuts to the raw amplitude, three of the four degrees of freedom in the loop momenta are removed. If the product of the three tree amplitudes is performed leaving the remaining degree of freedom as a free parameter, after subtracting the poles due to boxes, the result will be a power series in the free parameter. As will be shown later in this chapter, for the parametrisation used in this project, the coefficient to extract is the constant term in this expansion, as all other terms will vanish when integrated over the contour corresponding to the remaining degree of freedom from the momentum integration. For analytical methods this coefficient can be extracted by explicit rearrangement of the expression to isolate the needed term. For numerical methods this is not possible, instead, a discrete complex Fourier projection can be used to extract the coefficient, as long as the range of possible powers that can appear in the expansion is known. The continuous version of the projection for the term in the expansion of $f(z)$ of the form z^n is given by

$$P_n[f(z)] = \frac{1}{2\pi} \int_{\theta_0}^{\theta_0+2\pi} f\left(z_0 e^{it}\right) \left(z_0 e^{it}\right)^{-n} \mathrm{d}\, t \,, \tag{4.3}$$

where z_0 is an arbitrary complex value and θ_0 is an arbitrary real angle. For any power series, each term can be treated independently, as integration is linear and therefore, to prove the projection works, an expression of the form z^m can be substituted and then it can be shown that the result is 1 if $m = n$ and 0 otherwise. For $m = n$ the relation is given by

$$\begin{aligned} P_n[n^n] &= \frac{1}{2\pi}\int_{\theta_0}^{\theta_0+2\pi} \left(z_0 e^{it}\right)^n \left(z_0 e^{it}\right)^{-n} \mathrm{d}\, t \\ &= \frac{1}{2\pi}\int_{\theta_0}^{\theta_0+2\pi} \left(z_0 e^{it}\right)^{n-n} \mathrm{d}\, t \\ &= \frac{1}{2\pi}\int_{\theta_0}^{\theta_0+2\pi} 1 \, \mathrm{d}\, t \\ &= \frac{1}{2\pi}\left[t\right]_{\theta_0}^{\theta_0+2\pi} \\ &= 1 \ , \end{aligned} \tag{4.4}$$

and for $m \neq n$ it is

$$\begin{aligned} P_n[n^m] &= \frac{1}{2\pi}\int_{\theta_0}^{\theta_0+2\pi} \left(z_0 e^{it}\right)^m \left(z_0 e^{it}\right)^{-n} \mathrm{d}\, t \\ &= \frac{1}{2\pi}\int_{\theta_0}^{\theta_0+2\pi} \left(z_0 e^{it}\right)^{m-n} \mathrm{d}\, t \\ &= \frac{1}{2\pi}\left[\frac{-i}{m-n}\left(z_0 e^{it}\right)^{m-n}\right]_{\theta_0}^{\theta_0+2\pi} \\ &= \frac{1}{2\pi}\left(\left[\frac{-i}{m-n}\left(z_0 e^{i\theta_0}\right)^{m-n}\right] - \left[\frac{-i}{m-n}\left(z_0 e^{i(\theta_0+2\pi)}\right)^{m-n}\right]\right) \\ &= \frac{1}{2\pi}\frac{-i}{m-n}\left[\left(z_0 e^{i\theta_0}\right)^{m-n} - \left(z_0 e^{i\theta_0}\right)^{m-n}\right] \\ &= 0 \ . \end{aligned} \tag{4.5}$$

The discrete version to extract the coefficient of z^n using N evaluations is given by

$$D_{n,N}[f(z)] = \frac{1}{N}\sum_{j=0}^{N-1} f\left(z_0 e^{i2\pi \frac{j}{N}}\right)\left(z_0 e^{i2\pi\frac{j}{N}}\right)^{-n} \ , \tag{4.6}$$

The proof proceeds similar to before but now, if there are N terms in the projection, the result is 1 for $n = m \mod N$ and 0 for $n \neq m \mod N$. $m = n + kN$, with any integer k, results in

$$D_{n,N}[z^{n+kN}] = \frac{1}{N}\sum_{j=0}^{N-1}\left(z_0 e^{i2\pi\frac{j}{N}}\right)^{n+kN}\left(z_0 e^{i2\pi\frac{j}{N}}\right)^{-n}$$

$$
\begin{aligned}
&= \frac{1}{N}\sum_{j=0}^{N-1}\left(z_0 e^{i2\pi\frac{j}{N}}\right)^{(n+kN)-n} \\
&= \frac{1}{N}\sum_{j=0}^{N-1}\left(z_0 e^{i2\pi\frac{j}{N}}\right)^{kN} \\
&= \frac{1}{N} z_0^{kN}\sum_{j=0}^{N-1} e^{i2\pi kj\frac{N}{N}} \\
&= \frac{1}{N} z_0^{kN}\sum_{j=0}^{N-1} e^{i2\pi kj} \\
&= \frac{1}{N} z_0^{kN}\sum_{j=0}^{N-1} 1^{kj} \\
&= \frac{1}{N} z_0^{kN}\sum_{j=0}^{N-1} 1 \\
&= \frac{1}{N} z_0^{kN} N \\
&= z_0^{kN} \\
&= 1 \text{ if } k = 0 \;,
\end{aligned} \tag{4.7}
$$

and for $m \neq n \mod N$ it is given by

$$
\begin{aligned}
D_{n,N}[z^k] &= \frac{1}{N}\sum_{j=0}^{N-1}\left(z_0 e^{i2\pi\frac{j}{N}}\right)^{m}\left(z_0 e^{i2\pi\frac{j}{N}}\right)^{-n} \\
&= \frac{1}{N}\sum_{j=0}^{N-1}\left(z_0 e^{i2\pi\frac{j}{N}}\right)^{m-n} \\
&= \frac{1}{N} z_0^{m-n}\sum_{j=0}^{N-1} e^{i2\pi j\frac{m-n}{N}} \\
&= \frac{1}{N} z_0^{m-n}\sum_{j=0}^{N-1} \left(e^{i2\pi\frac{m-n}{N}}\right)^{j} \\
&= \frac{1}{N} z_0^{m-n} \times 0 \\
&= 0 \;.
\end{aligned} \tag{4.8}
$$

For $m - n \neq 0 \mod N$, $e^{i2\pi\frac{m-n}{N}}$ is an N^{th} root of unity, and when raised to N successive integers, each root is given exactly once. Combining these with the fact that the sum of all the N^{th} roots of unity is exactly 0, allows the fifth line of Equation 4.8 to be derived. From Equations 4.7 and 4.8 it is clear that if the expansion contains terms from $n_{\min}$ to $n_{\max}$ then a projection with at least $n_{\max} - n_{\min} + 1$ terms is needed.

The powers of these remaining parameters will come from numerator terms left after the diagrams have been reduced by cancelling with propagators. For boxes, as there are four propagators and external momenta are four dimensional, any combination that contains loop momenta squared or higher powers will be cancellable against a propagator and therefore should be contained in triangles or bubbles instead. Therefore, the highest power of the loop momenta that can remain is a term linear in the loop momenta. For triangles there is now the possibility of combinations that no longer fully cancel against propagators and are in directions orthogonal to components that could be cancelled against loop momenta. The highest power can be calculated using power counting to derive the highest power that could come from any diagram and as any extra powers in the denominator must cancel to reduce the diagram to a combination of our basis, powers of the loop momenta appearing in the denominator can be cancelled against those from the numerator.

For a normal renormalisable theory, for example the case of pure gluons amplitudes, each propagator provides two powers of the loop momenta in the denominator and each propagator must be connected by either a three or four point vertex. As four point vertices do not contain any powers of loop momenta, they will not contribute to the highest power of the loop momenta. The highest, therefore, must come from terms containing three point vertexes which each contribute a power of the loop momenta. Therefore, the highest power would come from a term with the minimum number of propagators and a three point vertex on each corner which for triangles is three powers of the loop momenta. For the case of amplitudes with a complex scalar, ϕ, they contain one power higher of momenta and their Feynman rules include a two gluon vertex with two powers of the particle momenta. This term would contribute one higher power of the loop momenta and therefore would have a numerator with up to four powers of the loop momenta.

For bubbles there is one less propagator and therefore one less vertex in the cases that produce the highest powers of the loop momenta in the numerator and their highest power possible is exactly one less than is possible for the triangles in the same theory. Bubbles, therefore, have up to two powers of the loop momenta in the numerator in renormalisable theories and for amplitudes with a ϕ up to three powers of the loop momenta.

This, along with the ability to calculate the tree amplitudes, is the only change needed to extend this for amplitudes with ϕ, as all the properties used apply to

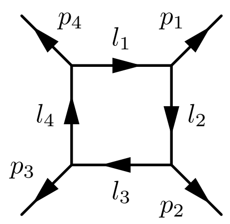


Figure 4.2: A box digram showing the labelling of the momenta and the direction each momenta is taken to be in.

any amplitude. To implement this efficiently and numerically, explicit formulas are required for the loop momenta solutions for each set of cuts. Firstly, for boxes, the relations that need to be solved are

$$\begin{aligned} l_1^2 &= \\ (l_1 - p_1)^2 &= \\ (l_1 - p_1 - p_2)^2 &= \\ (l_1 - p_1 - p_2 - p_3)^2 &= 0 \ , \end{aligned} \tag{4.9}$$

where $p_{1,2,3,4}$ are the external momenta outbound from each corner and l_1 is the loop momentum heading into corner 1 from corner 2 as shown in Figure 4.2. If the momenta flowing out of at least one of the corners is massless, then the solutions take a particularly simple form given by[6]

$$\begin{aligned} l_1^\mu &= \frac{\langle 1|2|3|4|\gamma^\mu|1\rangle}{2\,\langle 1|2|4|1\rangle} & {l'_1}^\mu &= \frac{[1|2|3|4|\gamma^\mu|1]}{2\,[1|2|4|1]} \\ l_2^\mu &= -\frac{\langle 1|\gamma^\mu|2|3|4|1\rangle}{2\,\langle 1|2|4|1\rangle} & {l'_2}^\mu &= -\frac{[1|\gamma^\mu|2|3|4|1]}{2\,[1|2|4|1]} \\ l_3^\mu &= \frac{\langle 1|2|\gamma^\mu|3|4|1\rangle}{2\,\langle 1|2|4|1\rangle} & {l'_3}^\mu &= \frac{[1|2|\gamma^\mu|3|4|1]}{2\,[1|2|4|1]} \\ l_4^\mu &= -\frac{\langle 1|2|3|\gamma^\mu|4|1\rangle}{2\,\langle 1|2|4|1\rangle} & {l'_4}^\mu &= -\frac{[1|2|3|\gamma^\mu|4|1]}{2\,[1|2|4|1]} \ , \end{aligned} \tag{4.10}$$

where without loss of generality corner 1 has been chosen to be a massless corner and $l_{1,2,3,4}$ and $l'_{1,2,3,4}$ are the two different solutions to the relations. These momenta satisfy the relations shown in Equation 4.9 if it can be shown that all these vectors are massless and that the difference between neighbouring loop momenta is the appropriate corners momenta. That they are all massless is easy to see from the form of the momenta and the relations for spinors. For corner 1 the momentum difference relation is given by

$$\begin{aligned}
l_1^\mu - l_2^\mu &= \frac{\langle 1|2|3|4|\gamma^\mu|1\rangle}{2\langle 1|2|4|1\rangle} - -\frac{\langle 1|\gamma^\mu|2|3|4|1\rangle}{2\langle 1|2|4|1\rangle} \\
&= \frac{\langle 1|2|3|4|\gamma^\mu|1\rangle + \langle 1|\gamma^\mu|2|3|4|1\rangle}{2\langle 1|2|4|1\rangle} \\
&= \frac{p_4^\mu \langle 1|2|3|1\rangle - \langle 1|2|3|\gamma^\mu|4|1\rangle + \langle 1|\gamma^\mu|2|3|4|1\rangle}{2\langle 1|2|4|1\rangle} \\
&= \frac{2p_4^\mu \langle 1|2|3|1\rangle - 2p_3^\mu \langle 1|2|4|1\rangle + \langle 1|2|\gamma^\mu|3|4|1\rangle + \langle 1|\gamma^\mu|2|3|4|1\rangle}{2\langle 1|2|4|1\rangle} \\
&= \frac{2p_4^\mu \langle 1|2|3|1\rangle - 2p_3^\mu \langle 1|2|4|1\rangle + 2p_2^\mu \langle 1|2|3|4|1\rangle - \langle 1|\gamma^\mu|2|3|4|1\rangle + \langle 1|\gamma^\mu|2|3|4|1\rangle}{2\langle 1|2|4|1\rangle} \\
&= -p_3^\mu + \frac{p_4^\mu \langle 1|2|3|1\rangle + p_2^\mu \langle 1|3|4|1\rangle}{\langle 1|2|4|1\rangle} \\
&= -p_3^\mu + \frac{-p_4^\mu \langle 1|2|4|1\rangle - p_2^\mu \langle 1|2|4|1\rangle}{\langle 1|2|4|1\rangle} \\
&= -p_3^\mu - p_4^\mu - p_2^\mu \\
&= p_1^\mu \ ,
\end{aligned} \tag{4.11}$$

where on the middle three lines commutation relations for slashed matrices have been used and on the last four lines momentum conservation between the external momenta has been used. The proofs for each of the other relations follow a similar method but are easier to show.

These relations only work if there is at least one momenta that is massless, which is true for pure gluon and quark amplitudes with seven or less external particles. For amplitudes with Higgs bosons, complex scalars, ϕ, or similarly any other external massive particles, these solutions are no longer enough for seven external particles, so a more general solution is required. To derive the general case, which is also the form of solution used for triangles and bubbles, a general basis of four massless independent vectors is needed. As there are in general no massless corners, the first step is to take two of the corners, which will be called corners 1 and 2 and their outbound momentum vectors K_1 and K_2 and project a pair of massless vectors by projecting them on each other. There are various definitions for how to perform the projection, but the one used here is a simplified version of that used by Berger and Forde[11] that leads to simpler formulas when working algebraically. This solution defines the two massless projections, $\tilde{K}_1$ and $\tilde{K}_2$ as

$$\begin{aligned}\tilde{K}_1 &= K_1 - \frac{K_1^2}{\gamma}K_2 \\ \tilde{K}_2 &= K_2 - \frac{K_2^2}{\gamma}K_1\ ,\end{aligned}\tag{4.12}$$

where γ is fixed such that the two projected vectors are massless and is given by

$$\begin{aligned}\gamma &= K_1\cdot K_2\left(1 \pm \sqrt{\frac{\Delta}{K_1\cdot K_2{}^2}}\right) \\ \Delta &= K_1\cdot K_2{}^2 - K_1^2K_2^2 = \det\begin{pmatrix} K_1^2 & K_1\cdot K_2 \\ K_1\cdot K_2 & K_2^2\end{pmatrix}\ ,\end{aligned}\tag{4.13}$$

where Δ is the determinant of the matrix of the products of any two of the three momentum conserving momenta, and is therefore independent of the labelling of the momenta. The two solutions for the momenta labeled by the choice of sign in γ are not independent. The solution with one sign is related to the solution with the opposite sign by the relations

$$\begin{aligned}\tilde{K}_1^{\mp} &= \frac{-\gamma^{\pm}}{K_2^2}\tilde{K}_2^{\pm} \\ \tilde{K}_2^{\mp} &= \frac{-\gamma^{\pm}}{K_1^2}\tilde{K}_1^{\pm}\ ,\end{aligned}\tag{4.14}$$

where the superscripts label which of the signs is chosen. From this it is clear that it is possible to choose either solution and ignore the other, as the results will be the same and using both would be a double counting. The solution with a positive sign is chosen so that the projected vectors are well behaved in the limit when K_1^2 and K_2^2 vanish, as for the negative sign solution, γ will vanish in this limit. The other two vectors used are built from the spinors for the two massless projected vectors and are given by

$$\begin{aligned}n_1 &= \frac{\left\langle \tilde{K}_1\middle|\gamma\middle|\tilde{K}_2\right]}{2} \\ n_2 &= \frac{\left\langle \tilde{K}_2\middle|\gamma\middle|\tilde{K}_1\right]}{2}\ .\end{aligned}\tag{4.15}$$

From these four vectors, any vector in 4 dimensional space can be built, as long as the four vectors are linearly independent, which they will be, if the two original vectors are linearly independent. These four vectors are particularly convenient to use as the products of various combinations vanish or are very simple. The spinor products that are simple are given by

$$\begin{aligned} n_1 \cdot n_2 &= -\tilde{K}_1 \cdot \tilde{K}_2 \\ 0 &= \tilde{K}_1 \cdot n_1 = \tilde{K}_1 \cdot n_2 \\ &= \tilde{K}_2 \cdot n_1 = \tilde{K}_2 \cdot n_2 \\ &= K_1 \cdot n_1 = K_1 \cdot n_2 \\ &= K_2 \cdot n_1 = K_2 \cdot n_2 \ . \end{aligned} \tag{4.16}$$

From this basis a general loop momenta can be written as

$$l_2 = \alpha \tilde{K}_1 + \beta \tilde{K}_2 + c_1 n_1 + c_2 n_2 \ , \tag{4.17}$$

where α, β, n_1 and n_2 are arbitrary complex constants. The first condition to apply will be to ensure that the loop momentum for the propagator between corners 1 and 2, l_2, is on-shell; this will be needed for all cuts. This corresponds to the condition

$$\begin{aligned} 0 &= l_2^2 \\ &= \left(\alpha \tilde{K}_1 + \beta \tilde{K}_2 + c_1 n_1 + c_2 n_2\right)^2 \\ &= 2\left(\alpha\beta \tilde{K}_1 \cdot \tilde{K}_2 + c_1 \alpha \tilde{K}_1 \cdot n_1 + c_2 \alpha \tilde{K}_1 \cdot n_2 + c_1 \beta \tilde{K}_2 \cdot n_1 + c_2 \beta \tilde{K}_2 \cdot n_2 + c_1 c_2 n_1 \cdot n_2\right) \\ &= 2\left(\alpha\beta \tilde{K}_1 \cdot \tilde{K}_2 + c_1 c_2 n_1 \cdot n_2\right) \\ &= 2\left(\alpha\beta - c_1 c_2\right) \tilde{K}_1 \cdot \tilde{K}_2 \ , \end{aligned} \tag{4.18}$$

which is used to fix $c_1 c_2 = \alpha\beta$. Which of c_1 and c_2 to fix in terms of the other variables is not specified. This is because, as will be shown below, α and β will be fixed by the next two cuts performed and one or both of them can be zero, in which case the solution for c_1 and c_2 will split into two branches, each with one of the constants vanishing and the other having a non zero value, ranging over all possible values depending on whether a fourth cut is applied. This form for the solution is particularly useful as the expression for the momenta, in terms of spinors, factorises and is given by

$$\begin{aligned} l_2^\mu &= \left(\left\langle \tilde{K}_1\right| + \frac{\beta}{c_1}\left\langle \tilde{K}_2\right|\right)\gamma^\mu\left(\alpha\left|\tilde{K}_1\right] + c_1\left|\tilde{K}_2\right]\right) \\ &= \left(\alpha\left\langle \tilde{K}_1\right| + c_2\left\langle \tilde{K}_2\right|\right)\gamma^\mu\left(\left|\tilde{K}_1\right] + \frac{\beta}{c_2}\left|\tilde{K}_2\right]\right) \ , \end{aligned} \tag{4.19}$$

where the two solutions are equivalent, unless one of K_1^2 and K_2^2 vanishes, in which case they are each valid in one branch of the solution space, as in the other they evaluate to $\frac{0}{0}$.

The next two cuts to be applied, which are needed for both triangles and boxes, are for the propagators either side of the loop momenta, l_2. These are given by the conditions

$$0 = (l_2 + K_1)^2 \qquad\qquad 0 = (l_2 - K_2)^2 \ , \tag{4.20}$$

for which solutions are given by

$$\begin{aligned} \alpha &= \frac{K_1\cdot\tilde{K}_2 K_2\cdot K_2 + K_1\cdot K_1 K_2\cdot\tilde{K}_2}{2(K_1\cdot\tilde{K}_2\tilde{K}_1\cdot K_2 - K_1\cdot\tilde{K}_1 K_2\cdot\tilde{K}_2)} \\ &= \frac{K_2^2(K_1^2+\gamma)}{4\Delta} \end{aligned} \tag{4.21}$$

$$\begin{aligned} \beta &= -\frac{K_1\cdot K_1\tilde{K}_1\cdot K_2 + K_1\cdot\tilde{K}_1 K_2\cdot K_2}{2(K_1\cdot\tilde{K}_2\tilde{K}_1\cdot K_2 - K_1\cdot\tilde{K}_1 K_2\cdot\tilde{K}_2)} \\ &= -\frac{K_1^2(K_2^2+\gamma)}{4\Delta} \ . \end{aligned} \tag{4.22}$$

Using these two solutions produces simple spinor representations for the loop momenta, l_1 and l_3, which are given by

$$\begin{aligned} l_1^\mu &= \left(\left\langle \tilde{K}_1\right| + \frac{\beta(1-C_{K_1})}{c_1}\left\langle \tilde{K}_2\right|\right)\gamma^\mu\left(\frac{\alpha}{1-C_{K_1}}\left|\tilde{K}_1\right] + c_1\left|\tilde{K}_2\right]\right) \\ &= \left(\frac{\alpha}{1-C_{K_1}}\left\langle \tilde{K}_1\right| + c_2\left\langle \tilde{K}_2\right|\right)\gamma^\mu\left(\left|\tilde{K}_1\right] + \frac{\beta(1-C_{K_1})}{c_2}\left|\tilde{K}_2\right]\right) \end{aligned} \tag{4.23}$$

$$\begin{aligned} l_3^\mu &= \left(\left\langle \tilde{K}_1\right| + \frac{\beta}{c_1(1-C_{K_3})}\left\langle \tilde{K}_2\right|\right)\gamma^\mu\left(\alpha(1-C_{K_3})\left|\tilde{K}_1\right] + c_1\left|\tilde{K}_2\right]\right) \\ &= \left(\alpha(1-C_{K_3})\left\langle \tilde{K}_1\right| + c_2\left\langle \tilde{K}_2\right|\right)\gamma^\mu\left(\left|\tilde{K}_1\right] + \frac{\beta}{c_2(1-C_{K_3})}\left|\tilde{K}_2\right]\right) \ , \end{aligned} \tag{4.24}$$

where C_{K_1} and C_{K_2} are constants given by

$$C_{K_1} = \frac{2\left(\gamma - K_1\cdot K_2\right)}{\gamma + K_2^2} \tag{4.25}$$

$$C_{K_2} = \frac{2\left(\gamma - K_1 \cdot K_2\right)}{\gamma + K_1^2} \ . \tag{4.26}$$

As for l_2, the two solutions are equivalent unless K_1^2 or $K_2^2 \to 0$, in which case they correspond to the two different branches of the solution.

The final cut is given by

$$0 = \left(l_2 - K_2 - K_3\right)^2 \ , \tag{4.27}$$

and is solved to give

$$\begin{aligned} c_1 &= \frac{X\left(1 \pm \sqrt{1 - \frac{16\alpha\beta K_3 \cdot n_1 K_3 \cdot n_2}{X^2}}\right)}{4K_3 \cdot n_1} \\ &= \frac{X\left(1 \pm \sqrt{1 + \frac{K_1^2 K_2^2 \Delta_4 (K_1^2 + K_2^2 + 2K_1 \cdot K_2)}{X^2 \Delta^2}}\right)}{4K_3 \cdot n_1} \end{aligned} \tag{4.28}$$

$$\begin{aligned} c_2 &= \frac{X\left(1 \mp \sqrt{1 - \frac{16\alpha\beta K_3 \cdot n_1 K_3 \cdot n_2}{X^2}}\right)}{4K_3 \cdot n_2} \\ &= \frac{X\left(1 \mp \sqrt{1 + \frac{K_1^2 K_2^2 \Delta_4 (K_1^2 + K_2^2 + 2K_1 \cdot K_2)}{X^2 \Delta^2}}\right)}{4K_3 \cdot n_2} \ , \end{aligned} \tag{4.29}$$

where these forms are again chosen to simplify choices of signs in the limit of K_1^2 or $K_2^2 \to 0$ and the terms X and Δ_4 are given by

$$\begin{aligned} X &= \left(K_2 + K_3\right)^2 - 2\alpha \tilde{K}_1 \cdot \left(K_2 + K_3\right) - 2\beta \tilde{K}_2 \cdot \left(K_2 + K_3\right) \\ &= \frac{\left(\left(K_1 + K_2\right)^2 + \left(K_1 - K_2\right) \cdot \left(K_4 - K_3\right)\right) K_1^2 K_2^2 + 2K_1 \cdot K_2 \left(K_1^2 K_2 \cdot K_3 + K_2^2 K_1 \cdot K_4\right)}{2\Delta} \\ &\quad + \left(K_2 + K_3\right)^2 \end{aligned} \tag{4.30}$$

$$\begin{aligned} \Delta_4 &= 2K_1.K_2 K_1.K_3 K_2.K_3 - \left(K_2.K_3\right)^2 K_1^2 - \left(K_1.K_3\right)^2 K_2^2 - \left(K_1.K_2\right)^2 K_3^2 + K_1^2 K_2^2 K_3^2 \\ &= \det \begin{pmatrix} K_1^2 & K_1 \cdot K_2 & K_1 \cdot K_3 \\ K_1 \cdot K_2 & K_2^2 & K_2 \cdot K_3 \\ K_1 \cdot K_3 & K_2 \cdot K_3 & K_3^2 \end{pmatrix} \ . \end{aligned} \tag{4.31}$$

These forms are not directly applicable to bubble diagrams as they only have one independent momentum vector and a basis must be built from at least two vectors. A solution of the same form would be very useful as it would again have a simple form in terms of spinors and would allow for efficient calculations using the same structures.

Therefore, an arbitrary on-shell reference vector, χ, is used as the second vector from which to build the basis. The exact value of this vector is irrelevant and will cancel in all calculations, but care is needed to ensure the vector will not have badly behaved products with any of the external momenta and as such should be chosen to be a complex vector that is not just a complex multiple of a real vector. The main example used in this project for numerical evaluation is $\chi = (1+2i, 1-2i, 1+4i, \sqrt{15})$. Using this vector and the method used above for triangle and boxes with the two cut conditions, equations are arrived at for the loop momenta of[12]

$$\begin{aligned} l_1 &= (1-y)\tilde{K}_1 + \frac{yK_1^2}{2K_1\cdot\chi}\chi - tn_1 - \frac{y(1-y)K_1^2}{2tK_1\cdot\chi}n_2 \\ l_2 &= -y\tilde{K}_1 - \frac{(1-y)K_1^2}{2K_1\cdot\chi}\chi - tn_1 - \frac{y(1-y)K_1^2}{2tK_1\cdot\chi}n_2 \ , \end{aligned} \tag{4.32}$$

where t and y are the two degrees of freedom left after the two cuts. These momenta can again be written in a compact spinor form as

$$\begin{aligned} l_1^\mu &= \left(-t\left\langle\tilde{K}_1\right| + \frac{yK_1^2}{2K_1\cdot\chi}\left\langle\chi\right|\right)\gamma^\mu\left(-\frac{1-y}{t}\left|\tilde{K}_1\right] + \left|\chi\right]\right) \\ l_2^\mu &= \left(-t\left\langle\tilde{K}_1\right| - \frac{(1-y)K_1^2}{2K_1\cdot\chi}\left\langle\chi\right|\right)\gamma^\mu\left(\frac{y}{t}\left|\tilde{K}_1\right] + \left|\chi\right]\right) \ . \end{aligned} \tag{4.33}$$

To perform the integrals, the Jacobian factors, J, and the integration contours for these choices of loop momenta representation are needed. The Jacobian factors result from both the change in integration variables from the loop momenta itself to the coefficients of our momenta definition and from performing the integrations over the delta functions. These can be given by the formula

$$J = \left|\frac{\sqrt{\det\left(\frac{\partial l}{\partial x_i}\cdot\frac{\partial l}{\partial x_j}\right)}}{\det\left(\frac{\partial d_k}{\partial x_l}\right)}\right| \ , \tag{4.34}$$

where x_i and x_j run over the set of new integration variables, d_k runs over the set of functions inside the delta functions and x_l runs over the variables integrated out with the delta functions. For example, for triangles this is due to changing from the loop momenta to the four variables α, β, c_1 and c_2 and then integrating out α, β and either c_1 or c_2 using the three delta functions. For this case x_i and x_j would run over all four of the variables α, β, c_1 and c_2 and x_l would run over α, β and whichever of c_1 and c_2 is integrated out.

For the case of the general box cut this gives a Jacobian factor, J_4, of

$$\begin{aligned} J_4 &= \left| \frac{\tilde{K}_1 \cdot \tilde{K}_2}{16 \left(K_1 \cdot \tilde{K}_1 K_2 \cdot \tilde{K}_2 - K_1 \cdot \tilde{K}_2 \tilde{K}_1 \cdot K_2 \right) \left(K_3 \cdot n_1 c_1 - K_3 \cdot n_2 c_2 \right)} \right| \\ &= \frac{1}{8} \left| \sqrt{\frac{\Delta}{X^2 \Delta^2 + \Delta_4 K_1^2 K_2^2 \left(K_1 + K_2 \right)^2}} \right| . \end{aligned} \tag{4.35}$$

For the case of triangle solutions the Jacobian factor is

$$\begin{aligned} J_3 &= \left| \frac{\tilde{K}_1 \cdot \tilde{K}_2}{8 \left(K_1 \cdot \tilde{K}_1 K_2 \cdot \tilde{K}_2 - K_1 \cdot \tilde{K}_2 \tilde{K}_1 \cdot K_2 \right) c} \right| \\ &= \frac{1}{8} \left| \frac{1}{c \sqrt{\Delta}} \right| , \end{aligned} \tag{4.36}$$

where c is whichever of c_1 and c_2 is the remaining free parameter. For the case of bubble solutions the Jacobian factor is

$$\begin{aligned} J_2 &= \left| \frac{\sqrt{(n_1.n_2)^2 \left(\tilde{K}_1.\chi \right)^2}}{4 t K_1.\chi n_1.n_2} \right| \\ &= \frac{1}{4} \left| \frac{1}{t} \right| . \end{aligned} \tag{4.37}$$

To calculate the conditions for the integration contour, it should be noted that the full integration space of the loop momenta is over all real momenta, so the contour will cover the set of parameters in the loop momenta that cause the loop momenta to be real. To evaluate these conditions the spinor representations are used along with the condition that for real vectors $\langle l|^\dagger \propto |l]$. For triangles, this relation simplifies to

$$c^* c = \text{const} , \tag{4.38}$$

where c is whichever of c_1 and c_2 is the free parameter and const is a positive function of the external momenta that, as all poles other than those at $c = 0$ are subtracted leaving only a power series in c, is irrelevant by Cauchy residue theorem. For bubbles, as there are two parameters, the solution space is a surface rather than a line, but again a simple relation can be found which is

$$t^* t = \text{const } y(1-y) , \tag{4.39}$$

where again const is a positive function of external momenta and therefore, the solution space is that $0 \leq y \leq 1$ and t is integrated round a circle whose radius is a function of y but, as all poles other than those at $t = 0$ are removed, this simplifies to an integration round any circle centred on the origin.

The exact values of the Jacobian expressions are not relevant, as the same factor would appear when integrating both the scalar loop and the true amplitude terms. The relevant parts are the forms of these in terms of the free parameters, as the Jacobian factors must be well behaved at all relevant points and therefore not contribute poles and, when combined with the forms of the contours, will determine which terms in the integrand will be extracted by performing the remaining integrals. For triangles, as the Jacobian is of the form $1/c$ and the contour is a circle around 0, it is the constant terms in the expansion of the triangles in terms of c that will contribute when integrated and all others will vanish. They will still be needed for subtracting from bubbles, as the other coefficients in terms of c can contribute to the poles that need subtracting. For bubbles, in terms of t, it is the constant term that is needed, again using the same logic as for triangles. To numerically evaluate the integrals, it is necessary to know what ranges of powers there can be in each variable. For this momentum parametrisation the integrand must be a polynomial of order $p - 1$ of the monomials y, t and $\frac{y(1-y)}{t}$ and therefore, for the t^0 terms, the range of powers of y is from 0 to $p - 1$. From direct evaluation of the y integral it is clear that the relevant contribution is given by

$$B = \sum_n \frac{b_{0,n}}{n+1} \ , \tag{4.40}$$

where $b_{i,j}$ is the coefficient of $t^i y^j$ in the expansion of the bubble after removing poles due to boxes and triangles. To extract all these coefficients requires $n \times m$ evaluations, where n is one more than the difference between the lowest and highest power of t and m is one more than the highest power of m, as the lowest power of y is 0. However, this is not the most efficient that can be achieved as instead of extracting each coefficient and combining, a simpler expression can be produced that will directly produce this combination. If the highest power of y is 3, as it is for standard QCD amplitudes, then the formula

$$B = \frac{1}{5}\left(b_0(0) + 3b_0\left(\frac{2}{3}\right)\right) \ , \tag{4.41}$$

will extract the relevant contribution, where $b_i(y_0)$ is the coefficient of t^i evaluated for $y = y_0$. This relation will not work for amplitudes with a complex scalar, ϕ, as a

y^4 term is present, therefore a more general relation is needed that is correct for one more power in y. The formula needed is

$$B = \frac{1}{2}\left(b_0\left(\frac{1}{2}\left(1+\frac{1}{\sqrt{3}}\right)\right) + b_0\left(\frac{1}{2}\left(1-\frac{1}{\sqrt{3}}\right)\right)\right) \ . \tag{4.42}$$

If even higher powers of y occurred then a relation with more evaluations would be needed but they are not required for this project.

With these terms the boxes can now be evaluated directly, but for the triangles and bubbles one more contribution is needed, the explicit forms of the subtractions. These poles are of the form of a numerator, which is the unintegrated numerator for the box or triangle diagram, divided by a denominator, which is the product of the propagators corresponding to the extra cuts in that diagram. For example, for the contribution of the box to the second triangle from the left in Figure 4.3, the triangle has cuts for $(l_2+K_1)^2 = 0$, $l_2^2 = 0$ and $(l_2-K_2)^2 = 0$ and the box has an extra cut at $(l_2-K_2-K_3)^2 = 0$, so the pole contribution will be from "box numerator form"$/(l_2-K_2-K_3)^2$. For this case the two reference vectors for the box are the same as the two reference vectors for the triangle, but the calculations are equivalent even if the choice of reference vectors is different, as long as care is made to use the correct signs on the box contributions and to use the vector for the extra pole evaluated relative to the appropriate loop momenta of the triangle.

The numerators of boxes can at most be linear in the free parameter in the triangle loop momentum parametrisation, as anything with higher powers would be cancellable with propagators in the box and therefore will be accounted for in triangle and bubble contributions. As its value at the location of the two box contributions are known, it must be given by

$$\text{“box numerator form”} = \frac{c\,(b^+-b^-)+c^+b^- - c^-b^+}{c^+-c^-} \ , \tag{4.43}$$

where c is whichever of c_1 and c_2 is the free parameter in the triangle parametrisation, $c^{+/-}$ are the two solutions to the extra box cut as given in Equation 4.28 or Equation 4.29 and $b^{+/-}$ are the two box coefficients corresponding to each of the solutions. Combining these relations, along with the $1/c$ from the Jacobian and simplifying results in expressions for the poles of

$$\text{“pole”} = -\frac{1}{2K_3 \cdot n\,(c^+ - c^-)} \left(\frac{b^+}{c - c^+} - \frac{b^-}{c - c^-} \right) \;, \tag{4.44}$$

where n is n_1 if c is c_1 and n is n_2 if c is c_2.

Applying a triangle as a pole to a bubble, requires finding both the numerator structure and the simplified pole structure. The pole structure comes from evaluating the propagator corresponding to the extra cut at $l_2 - K_2$, using the momentum representation for l_2 given in Equation 4.32 which gives

$$\frac{1}{(l_2 - K_2)^2} = -\frac{tK_1 \cdot \chi}{K_2 \cdot n_2 K_1^2\,(y^+ - y^-)} \left(\frac{1}{y - y^+} - \frac{1}{y - y^-} \right) \;, \tag{4.45}$$

where $y^{+/-}$ are the two solutions of the propagator going on-shell for y which are given by

$$y^\pm = \left(\frac{1}{2} + t\frac{K_1 \cdot \chi K_1 \cdot K_2 - K_1^2 K_2 \cdot \chi}{K_2 \cdot n_2 K_1^2} \right) \left(1 \pm \sqrt{1 + \frac{t\frac{2tK_1\cdot\chi K_2\cdot n_1 + K_1^2 K_2\cdot\chi + K_2^2 K_1\cdot\chi}{K_1^2 K_2\cdot n_2}}{\left(\frac{1}{2} + \frac{t\left(K_1\cdot\chi K_1\cdot K_2 - K_1^2 K_2\cdot\chi\right)}{K_1^2 K_2\cdot n_2} \right)^2}} \right) . \tag{4.46}$$

Each c in the triangle form must come from a loop momentum multiplied with some combination of external momenta. Therefore, to expand out in terms of the space of the bubble momentum solutions, the values of the coefficients must be projected out of the full momentum. As the basis elements in the triangle momentum parametrisation have simple products with each other, an obvious choice for how to project out the coefficients, is to project using the basis vectors. The coefficients are found to be extracted by

$$c_1 = -\gamma\frac{l_2 \cdot n_2}{2\Delta} \qquad\qquad c_2 = -\gamma\frac{l_2 \cdot n_1}{2\Delta} \;, \tag{4.47}$$

which if applied to the bubble momentum form will give

$$\begin{aligned} c_1(t,y) &= \frac{\gamma}{4\Delta} \left(\left\langle \tilde{K}_{t,2} \middle| \tilde{K}_1 \right\rangle + \frac{(1-y)\left\langle \tilde{K}_{t,2} \middle| \chi \right\rangle K_1^2}{2tK_1\cdot\chi} \right) \left(t \left[\chi \middle| \tilde{K}_{t,1} \right] + y \left[\tilde{K}_1 \middle| \tilde{K}_{t,1} \right] \right) \\ c_2(t,y) &= \frac{\gamma}{4\Delta} \left(\left\langle \tilde{K}_{t,1} \middle| \tilde{K}_1 \right\rangle + \frac{(1-y)\left\langle \tilde{K}_{t,1} \middle| \chi \right\rangle K_1^2}{2tK_1\cdot\chi} \right) \left(t \left[\chi \middle| \tilde{K}_{t,2} \right] + y \left[\tilde{K}_1 \middle| \tilde{K}_{t,2} \right] \right) \;, \end{aligned} \tag{4.48}$$

where $\tilde{K}_{t,1/2}$ are the $\tilde{K}_{1/2}$ for the triangle by which this pole is caused. For triangles with a massless corner, whichever of these solutions corresponds to the variable that gives contributions for that triangle will be used. For the case where all corners are massive, both solutions will be used, as the solution which was not used to evaluate the triangle can be used to give an expression for $1/c$ that has no poles in terms of y. With these terms the all massive case can be evaluated, but in the massless triangle case an extra complication arises, as there are terms that are proportional to $\sqrt{1+at+bt^2}$ where a and b are constants in terms of t and y. These terms integrate to zero on our contour, but would need an infinite number of terms in the Fourier projection to cause them to cancel correctly. For the massive case, due to symmetries, these terms cancel and so can be ignored, but for the massless case cancellations do not always occur and must be added in manually. These contributions come from partial fractioning of the pole form, which takes our numerator function $\mathrm{num}(c(t,y))$ divided by the pole at $y = y^{\pm}$, and rearranges it to the form

$$\frac{C}{y^+ - y^-}\frac{\mathrm{num}(c(t,y))}{y - y^{\pm}} = \frac{C}{y^+ - y^-}\frac{\mathrm{num}(c(t,y^{\pm}))}{y - y^{\pm}} + \frac{Cf(t,y,y^{\pm})}{y^+ - y^-} \ , \tag{4.49}$$

where C contains all the terms in the coefficient part of the expanded pole form that are free from y, y^+ and y^- and f is a rational function of its parameters, with only positive powers of y and $y^{\pm}$ and both positive and negative powers of t. This function has terms that should vanish due to integrations over terms of the form $\sqrt{1+at+bt^2}$, but which do not vanish in our numerical implementations and therefore must be subtracted. This expression will only give contributions that contain the dangerous term, when there are terms with an odd power of the square root. The simplest expression that would fully cancel this term is one which has exactly the same form, but with the opposite sign on the square root which corresponds to using the other solution for y. To directly evaluate f in terms of the other solution would require analytically calculating the form of f, which is a non trivial task, as such it is much easier to use the directly evaluated pole form to calculate the cancellation needed. This is a function of the form

$$\frac{C}{y^+ - y^-}\left(\frac{\mathrm{num}(c(t,y))}{y - y^{\pm}} - \frac{\mathrm{num}(c(t,y))}{y - y^{\mp}} + \frac{\mathrm{num}(c(t,y^{\mp}))}{y - y^{\mp}}\right) \ . \tag{4.50}$$

For poles from boxes contributing to bubbles, the product of the two extra propagators and the known numerator form found above can be simplified using partial fractions to get a form for the poles that combines the box on triangle and triangle on bubble forms found above, giving

$$\frac{\text{“box numerator form”}}{\left(l_2-K_2\right)^2\left(l_2-K_3\right)^2}=-\frac{tK_1\cdot\chi}{K_1^2}\Bigg(
+\frac{1}{K_2\cdot n_2\left(y_{2+}-y_{2-}\right)}\left(\frac{\text{“pole”}\left(K_3,c\left(t,y_{2+}\right)\right)}{y-y_{2+}}-\frac{\text{“pole”}\left(K_3,c\left(t,y_{2-}\right)\right)}{y-y_{2-}}\right)
+\frac{1}{K_3\cdot n_2\left(y_{3+}-y_{3-}\right)}\left(\frac{\text{“pole”}\left(K_2,c\left(t,y_{3+}\right)\right)}{y-y_{3+}}-\frac{\text{“pole”}\left(K_2,c\left(t,y_{3-}\right)\right)}{y-y_{3-}}\right)\Bigg)\ , \tag{4.51}$$

where $y_{2/3\pm}$ are the two solutions for y for the poles with K_2 or K_3 respectively and “pole”(K,c) is the box on triangle pole from Equation 4.44 for the extra cut at K evaluated for the value of c given. From this it is clear that the box on bubble pole gives a pair of contributions that each look like the contribution of a triangle and as such for evaluation they can be combined with the triangle pole that has the same pole structure.

For efficiency reasons it is useful to skip the calculation of terms that are known to not contribute. The more terms that can be skipped, the less work there is to do and the less terms there are at higher numbers of loop propagators, the less poles there are that need subtracting from lower loop propagator calculations. The simplest set of terms to ignore is any for which one of the corners is an amplitude with helicity choices that vanish, which applies irrespective of the particles involved, but has more cases to consider, as which helicity combinations vanish depends on the particles involved. The next few simplifications depend on properties of the loop momenta when corners are on-shell. Firstly, when a single corner goes on-shell, the loop momenta on each side of it can be simplified to

$$\begin{aligned}
l_1^\mu &= \langle K_1|\gamma^\mu\left(\left(1+\frac{K_2^2}{2K_1\cdot K_2}\right)|K_1]+c_1\left|\tilde{K}_2\right]\right)\\
&=\left(\left(1+\frac{K_2^2}{2K_1\cdot K_2}\right)\langle K_1|+c_2\left\langle\tilde{K}_2\right|\right)\gamma^\mu|K_1]\\
l_2^\mu &= \langle K_1|\gamma^\mu\left(\frac{K_2^2}{2K_1\cdot K_2}|K_1]+c_1\left|\tilde{K}_2\right]\right)\\
&=\left(\frac{K_2^2}{2K_1\cdot K_2}\langle K_1|+c_2\left\langle\tilde{K}_2\right|\right)\gamma^\mu|K_1]\ ,
\end{aligned}\tag{4.52}$$

where the massless corner has been labelled as 1. From these expressions it is clear that whether c_1 or c_2 is non-zero, determines which type of spinor is proportional for all the momenta in that corner. For amplitudes with three on-shell particles, only two cases are non vanishing and each of these must have two particles with one helicity and one with the opposite helicity and will be labelled by the helicity of which they have two. These amplitudes will each depend on only one type of spinor and therefore will vanish when the spinors they depend on are proportional. As a result, if a corner is massless, only one branch of the momentum solutions need be considered. Therefore, for numerical calculations, if there is at least one massless corner then one of them will be chosen to be corner 1 and only evaluated with the correct branch of the solution for the helicities in that corner.

If two neighbouring corners are on-shell then the loop momenta between them and either side of them are related in spinor forms and are given by

$$\begin{aligned} l_1^\mu &= \langle K_1| \gamma^\mu \left(|K_1] + c_1 \, |K_2]\right) \\ &= \left(\langle K_1| + c_2 \, \langle K_2|\right) \gamma^\mu \, |K_1] \\ l_2^\mu &= c_1 \, \langle K_1|\gamma^\mu|K_2] \\ &= c_2 \, \langle K_2|\gamma^\mu|K_1] \\ l_3^\mu &= \left(c_1 \, \langle K_1| - \langle K_2|\right) \gamma^\mu \, |K_2] \\ &= \langle K_2| \gamma^\mu \left(c_2 \, |K_1] - |K_2]\right) \ , \end{aligned} \tag{4.53}$$

where spinors can now be written for K_1 and K_2, as they are on-shell and so the $\tilde{K}_1$ and $\tilde{K}_2$ projected vectors are just the same as the unprojected vectors and so are no longer needed. If two adjacent corners are massless, then they must have opposite helicities, as otherwise they each vanish on opposite solutions and so neither solution will contribute for that diagram. For boxes, if two opposite corners are both on-shell, the solutions will have the same type of spinor proportional in each of the two corners and therefore must in both corners depend only on the other type of spinor and must be of the same helicity. These relations serve to greatly reduce the numbers of diagrams that are relevant and which need to be computed. For bubbles, an even stricter condition can be found as scalar loop diagrams vanish if there are no scales in the diagram and as there are only two corners, their momentum vectors must be equal and opposite, so the only scale available is the mass of this momentum. If it vanishes then the scalar loop function vanishes, so diagrams with only one massless particle on one side of the bubble will vanish and do not need to be computed. All these relations apply equally well to amplitudes with a complex scalar, ϕ, as to pure QCD amplitudes, although care must be taken as a corner with contains only a ϕ is not a massless corner.

Contributions that are not relevant have been removed from the set of terms shown in Figure 4.3, Figure 4.4 and Figure 4.5 using all these relations.

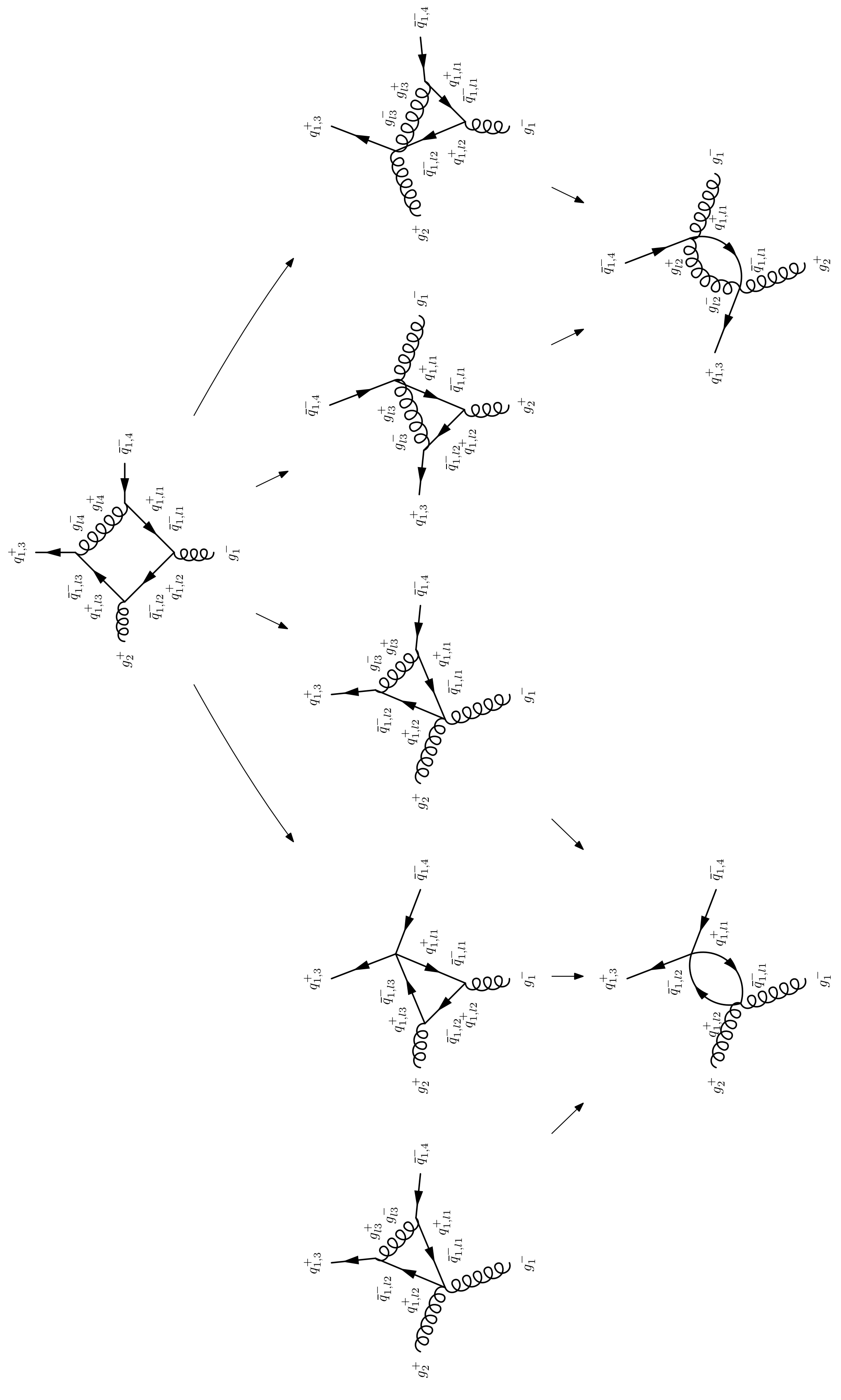
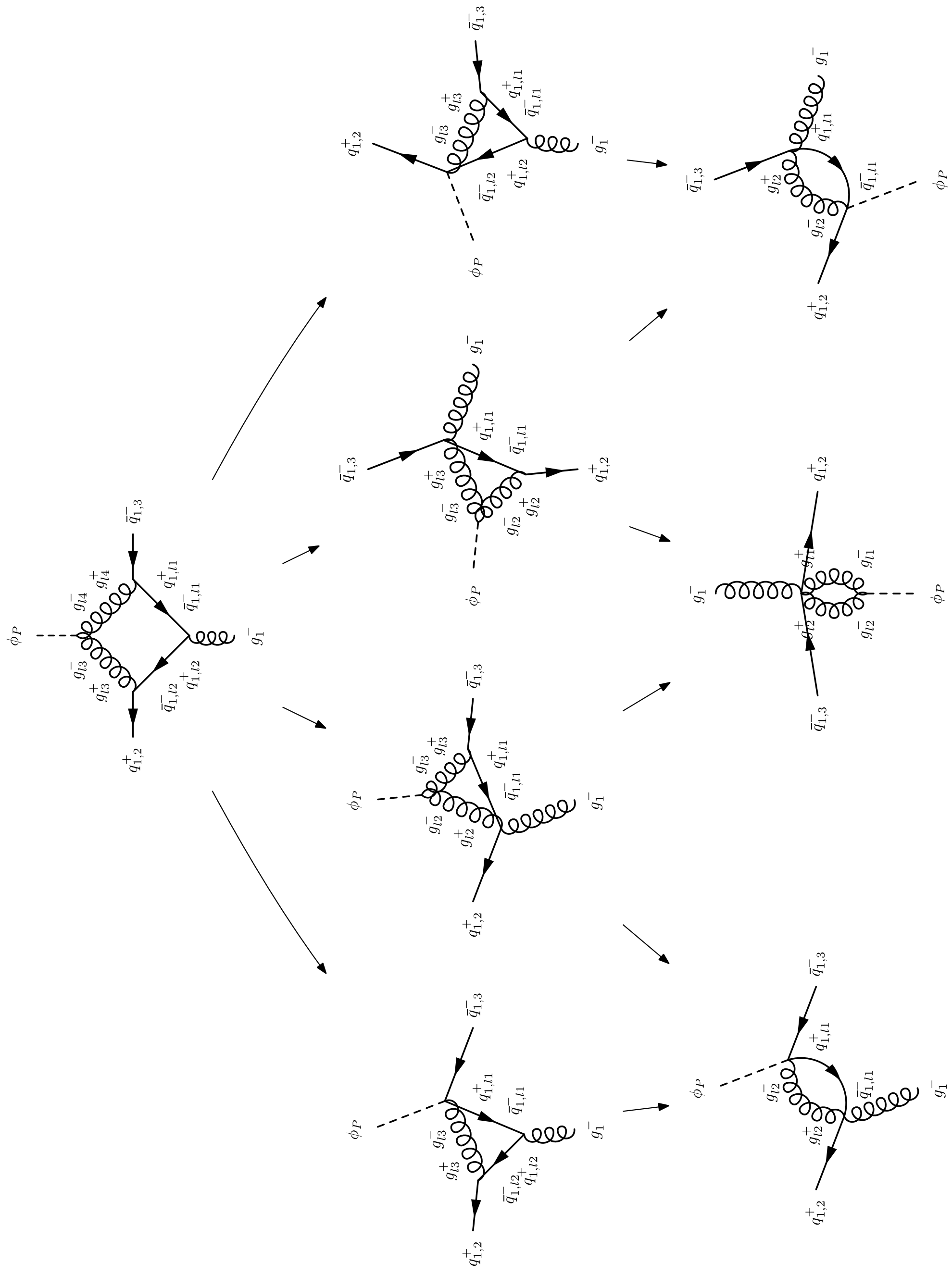
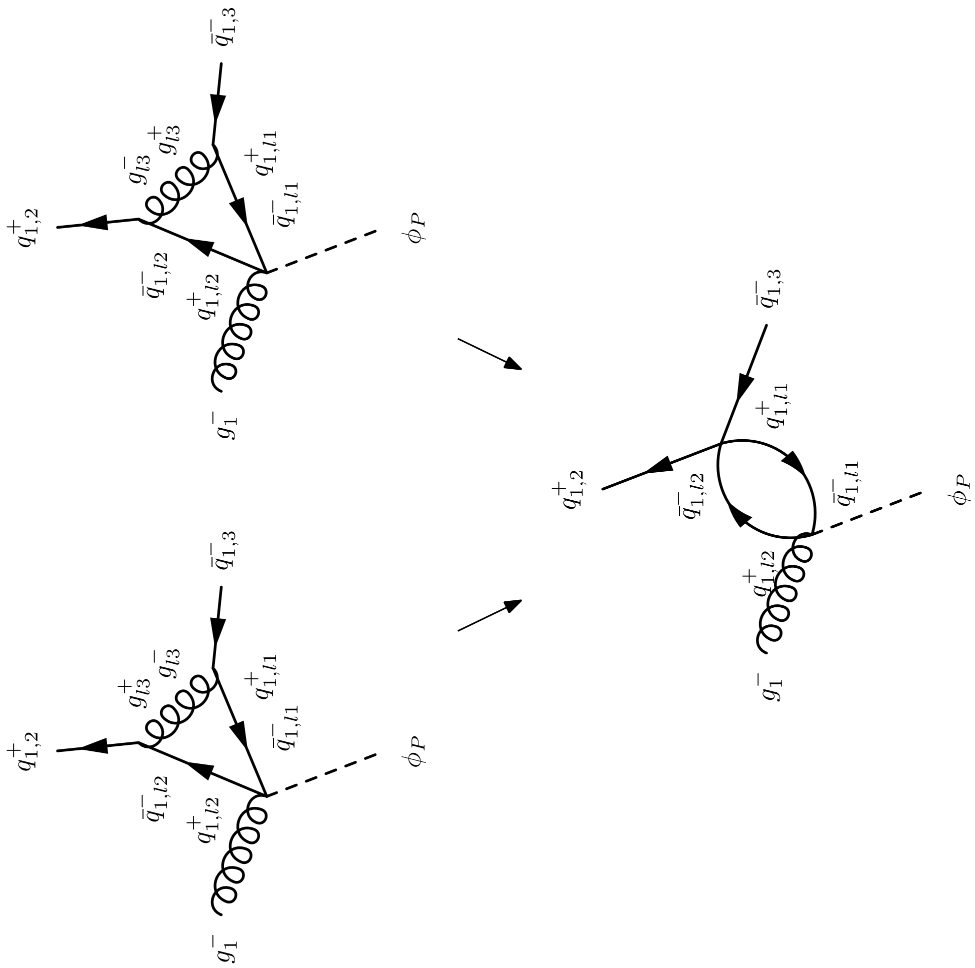


Figure 4.3: A graph showing the different terms that contribute to the amplitude for $g_1^- g_2^+ q_{1,3}^+ \bar{q}_{1,4}^-$ with the quark travelling left. The arrows indicate other diagrams to which a diagram contributes a pole.



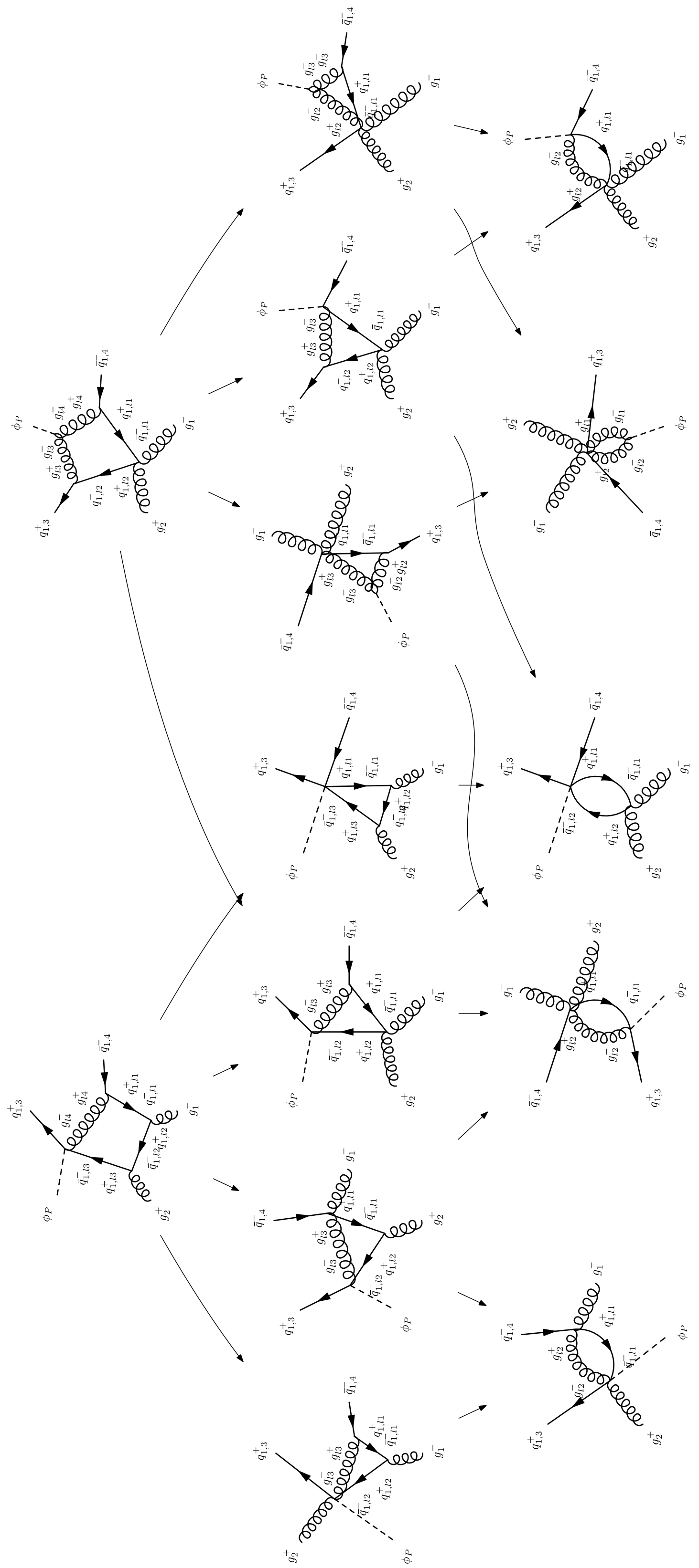
(4.4.a) One of the two fragments of the diagram of terms. The other is in Figure 4.4.b.

Figure 4.4: A graph showing the different terms that contribute to the amplitude for $g_1^- q_{1,2}^+ \bar{q}_{1,3}^- \Phi_P$ with the quark travelling left. The arrows indicate other diagrams to which a diagram contributes a pole.



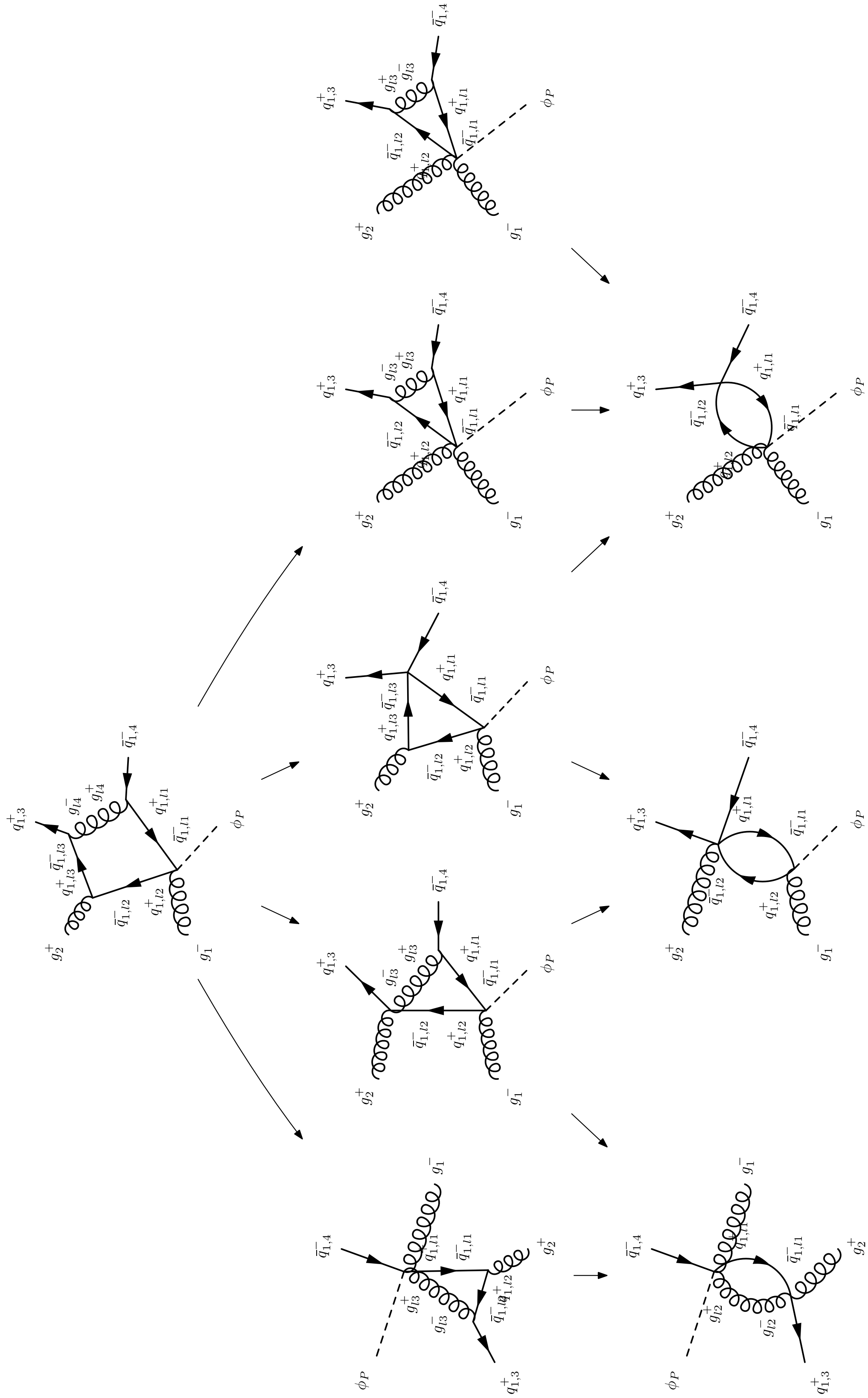
(4.4.b) One of the two fragments of the diagram of terms. The other is in Figure 4.4.a.

Figure 4.4: A graph showing the different terms that contribute to the amplitude for $g_1^- q_{1,2}^+ \bar{q}_{1,3}^- \Phi_P$ with the quark travelling left. The arrows indicate other diagrams to which a diagram contributes a pole.



(4.5.a) One of the two fragments of the diagram of terms. The other is in Figure 4.5.b.

Figure 4.5: A graph showing the different terms that contribute to the amplitude for $g_1^- g_2^+ q_{1,3}^+ \bar{q}_{1,4}^- \Phi_P$ with the quark travelling left. The arrows indicate other diagrams to which a diagram contributes a pole.



(4.5.b) One of the two fragments of the diagram of terms. The other is in Figure 4.5.a.

Figure 4.5: A graph showing the different terms that contribute to the amplitude for $g_1^- g_2^+ q_{1,3}^+ \bar{q}_{1,4}^- \Phi_P$ with the quark travelling left. The arrows indicate other diagrams to which a diagram contributes a pole.

Chapter 5

Rational Terms

In the previous chapters, methods have been developed that will construct the majority of each amplitude for processes with a Higgs boson and many jets at one loop, however there are terms that cannot be extracted using just these techniques. These terms are the terms that do not have any poles or cuts in 4 dimensions and therefore cannot be constructed by techniques that require poles to build the amplitudes. These terms are called rational terms. One of the common ways to calculate rational terms is to use relations involving supersymmetric amplitudes[13]. These amplitudes contain combinations of the physical amplitudes and extra amplitudes. The most common combination to use is a combination of $\mathcal{N}=1$ chiral and $\mathcal{N}=4$ supersymmetric theories, where $\mathcal{N}$ counts how many supersymmetries there are in the theory. For external gluon amplitudes these are given by[13]

$$\begin{aligned} A_n^{\mathcal{N}=4} &\equiv A_n^g + 4A_n^f + 3A_n^S \\ A_n^{\mathcal{N}=1 \text{ chiral}} &\equiv A_n^f + A_n^S \ , \end{aligned} \tag{5.1}$$

where A^g, A^f and A^S are the different versions of an amplitude, with gluons, Weyl fermions and complex scalars respectively in the loop. Using these building blocks pure gluon amplitudes and quark amplitudes can be built up as[13]

$$\begin{aligned} A_n^g &= A_n^{\mathcal{N}=4} - 4A_n^{\mathcal{N}=1 \text{ chiral}} + A_n^S \\ A_n^f &= A_n^{\mathcal{N}=1 \text{ chiral}} - A_n^S \ . \end{aligned} \tag{5.2}$$

As all supersymmetric amplitudes are cut constructable in the four dimensional helicity scheme, they have no rational terms. Therefore, the rational terms of their component amplitudes are related to each other by[13]

$$A_n^g\big|_{\text{rational}} = A_n^S\big|_{\text{rational}} \qquad\qquad A_n^f\big|_{\text{rational}} = -\,A_n^S\big|_{\text{rational}}\ , \tag{5.3}$$

where $|_{\text{rational}}$ means take only the rational part of the amplitude. From these relations it can be seen that the rational part of quark and gluon amplitudes is the same as the rational part of their related scalar amplitudes, but with a minus sign for quark amplitudes. These scalar amplitudes are much simpler and so can be extracted using generalised unitarity in 6 dimensions or recursion relations. Unfortunately this is not possible with Higgs boson amplitudes, as forming a supersymmetric amplitude would require using the supersymmetric extension for the Higgs boson, rather than the standard model Higgs boson.

Another type of method that has been used to extract these terms is recursion methods using ideas similar to BCFW as done by Berger, Del Duca and Dixon[15]. These methods use properties of the loop amplitudes under shifts of the external momenta to relate the rational terms to the cut terms and therefore rely on the pole structure of the cut terms when shifts are applied. These methods are therefore not as easy to apply numerically as they rely on complete knowledge of the pole structure of the amplitudes.

The method used here is to use generalised unitarity applied directly in more than 4 dimensions, as in more than 4 dimensions the rational terms have cuts and poles. If calculating in any specific dimensions greater than four, then the dependence on the dimensionality of space time can be extracted by using the dependence on the magnitude of the momenta pointing into the extra dimensions, as long as there is no explicit dependence on the direction or complex argument of any components. In the four dimensional helicity scheme (FDH)[3], which is being used here, all internal spinor and gluon states are kept in 4 dimensions. Therefore, the dependence of amplitudes on the dimension of space time, due to loop momenta and due to extra spinor and gluon states, must be separated. To derive the relations used to separate these two dependencies, it is useful to separate the dimension of the loop momenta, which will be called D_l, from the dimension of the spinor and gluon polarisation states, which will be called D_s. Using this split, a loop amplitude is given by[14]

$$A_{D_l,D_s}(\{p_i\}) = \int \frac{d^{D_l} l}{i(\pi)^{\frac{D_l}{2}}} \frac{\mathcal{N}_{D_s}(\{p_i\},l)}{d_1 d_2 \dots d_N} \ . \tag{5.4}$$

where D_l must be less than or equal to D_s.

If there is a closed quark loop in an amplitude, the extra contributions due to an increase of two in dimensions will be factors of two, due to there being twice as many states and the gamma matrices being twice as big. If chiral quarks are used, the quark space will split into two. If it can be shown that using either of the chiral quark subspaces is a valid representation of the 4 dimensional states and that amplitudes do not mix the two spaces and are identical in either case, then the state reduction can be performed by using only one of the two spaces in the calculations. For the case of pure gluon amplitudes, it can be shown that the dependence of the loop amplitude on D_s must be at most linear, as it arises purely from terms that form a closed loop of metric tensors. Therefore the amplitude in any dimensionality can be given by[14]

$$A_{D_l,D_s}(\{p_i\}) = A^0_{D_l}(\{p_i\}) + (D_s - 4)\, A^1_{D_l}(\{p_i\}) \ , \tag{5.5}$$

where $A^0_{D_l}(\{p_i\})$ and $A^1_{D_l}(\{p_i\})$ is independent of D_s, but will still depend on D_l. Combining two evaluations at two different values for D_s, the values of the two different components can be extracted which will allow a continuation to different values of D_s. Therefore, the value in the four dimensional helicity scheme is given by[14, 16]

$$A_{FDH} = \frac{(D_2 - 4)A_{D_l,D_s=D_1} - (D_1 - 4)A_{D_l,D_s=D_2}}{D_2 - D_1} \ . \tag{5.6}$$

To evaluate these expressions explicitly requires calculating in two different, even, integer dimensions, both of which must be greater than or equal to D_l, which is itself greater than 4. This would require working in both 6 and 8 dimensions although with the restriction that the loop momenta remains in either 5 or 6 dimensions. In both of these sets of dimensions a generalised unitarity calculation would need to be performed to extract the coefficients of the integrals and then once combined they can be limited back down to $d = 4 - 2\epsilon$ dimensions. Again the basis of loop integrals is needed and now the basis is extended to include pentagon scalar loop integrals. The coefficients now contain extra tensor components dependant on the loop momenta and there are now multiple elements that do not vanish when integrated. Some of these

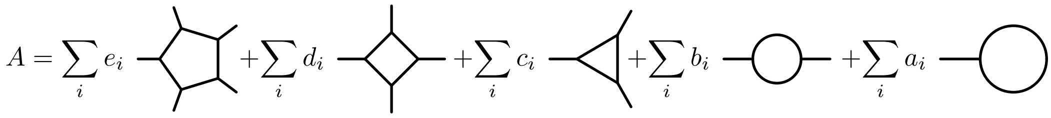


Figure 5.1: The expansion of an amplitude in terms of the scalar basis functions in D dimensions. The sums are over each possible scalar loop function of that form, defined by the momenta at each corner. The coefficients contain loop momentum tensor structures and are therefore inside the loop momentum integral.

tensor terms give rise to D_l dependence which when combined with poles in the loop integrals results in the rational terms. In terms of this basis the amplitude is expanded as shown in Figure 5.1. The forms that can appear in the coefficients are limited as any term containing external momenta can be replaced by a combination of inverse propagators, loop momenta squared and rational functions of external momenta as

$$l \cdot P = \frac{1}{2}\left(l^2 + P^2 - (l-P)^2\right) \ . \tag{5.7}$$

A further restriction is that there is no dependence on the direction in the extra $D-4$ dimensions, as there are no external vectors in this subspace, only on the total magnitude of these components which will be labelled by μ^2 which is defined by

$$l^2 = \bar{l}^2 - \mu^2 \ , \tag{5.8}$$

where $\bar{l}$ contains the 4 dimensional components of l. Therefore, the terms that can appear are all possible terms built from μ^2 and $t_i = l \cdot n_i$, where n_i are unit vectors that form an orthonormal basis of the subspace of 4 dimensional space that is transverse to all external momenta. A final restriction is on the highest power of loop momenta that can occur in the integrals, which for amplitudes involving a Higgs boson is one higher than the number of propagators in the loop momenta. Therefore, the general forms of the coefficients are

$$e_i = \mu^2 \tilde{e}_i \tag{5.9}$$

$$d_i = \bar{d}_i^0 + t_1 \bar{d}_i^1 + \mu^2\left(\tilde{d}_i^0 + t_1 \tilde{d}_i^1\right) + \mu^4\left(\tilde{d}_i^2 + t_1 \tilde{d}_i^3\right) \tag{5.10}$$

$$
\begin{aligned}
c_i &= \bar{c}_i^0 + t_1\bar{c}_i^1 + t_2\bar{c}_i^2 + (t_1^2 - t_2^2)\bar{c}_i^3 + t_1t_2\bar{c}_i^4 + (t_1^2 - 3t_2^2)t_1\bar{c}_i^5 + (t_2^2 - 3t_1^2)t_2\bar{c}_i^6 \\
&+ (t_1^4 + t_2^4 - 6t_1^2t_2^2)\bar{c}_i^7 + (t_1^2 - t_2^2)t_1t_2\bar{c}_i^8 \\
&+ \mu^2\left(\tilde{c}_i^0 + t_1\tilde{c}_i^1 + t_2\tilde{c}_i^2 + (t_1^2 - t_2^2)\tilde{c}_i^3 + t_1t_2\tilde{c}_i^4\right) + \mu^4\tilde{c}_i^5 \qquad (5.11) \\
b_i &= \bar{b}_i^0 + t_1\bar{b}_i^1 + t_2\bar{b}_i^2 + t_3\bar{b}_i^3 + (t_1^2 - t_3^2)\bar{b}_i^4 + (t_2^2 - t_3^2)\bar{b}_i^5 + t_1t_2\bar{b}_i^6 + t_1t_3\bar{b}_i^7 + t_2t_3\bar{b}_i^8 \\
&+ t_1(t_1^2 - 3t_2^2)\bar{b}_i^9 + t_2(t_2^2 - 3t_3^2)\bar{b}_i^{10} + t_3(t_3^2 - 3t_1^2)\bar{b}_i^{11} + t_1(t_2^2 - t_3^2)\bar{b}_i^{12} \\
&+ t_2(t_3^2 - t_1^2)\bar{b}_i^{13} + t_3(t_1^2 - t_2^2)\bar{b}_i^{14} + t_1t_2t_3\bar{b}_i^{15} + \mu^2\left(\tilde{b}_i^0 + t_1\tilde{b}_i^1 + t_2\tilde{b}_i^2 + t_3\tilde{b}_i^3\right) \ , \qquad (5.12)
\end{aligned}
$$

where the dependence on t_i encodes all the 4 dimensional loop momentum dependence, coefficients with a bar over them appear in 4 dimensional generalised unitarity calculations, coefficients with a ˜ over them are new coefficients for D dimensional calculations and all coefficients are independent of the dimension of the loop momenta, D_l. It is now possible to expand each of these coefficients out into their own integrals. Most of the integrals vanish and as such only a small subset are needed directly, but for numerical calculations, in theory, all coefficients are needed as subtractions. The only integrals, other than the normal 4 dimensional basis, that do not vanish in the limit of $D = 4 - 2\epsilon \to 4$ are integrals whose numerators can only be powers of μ^2. However the box and pentagon with μ^2 inserted do vanish in this limit. In the limit of $D = 4 - 2\epsilon \to 4$, dropping terms of order ϵ, the extra non-vanishing integrals are given by[17]

$$I_4[\mu^4] = \int \frac{d^Dl}{i\pi^{\frac{D}{2}}} \frac{\mu^4}{D_1D_2D_3D_4} = -\frac{1}{6} \qquad (5.13)$$

$$I_3[\mu^4] = \int \frac{d^Dl}{i\pi^{\frac{D}{2}}} \frac{\mu^4}{D_1D_2D_3} = \frac{1}{24}(P_1^2 + P_2^2 + P_3^2) \qquad (5.14)$$

$$I_3[\mu^2] = \int \frac{d^Dl}{i\pi^{\frac{D}{2}}} \frac{\mu^2}{D_1D_2D_3} = \frac{1}{2} \qquad (5.15)$$

$$I_2[\mu^2] = \int \frac{d^Dl}{i\pi^{\frac{D}{2}}} \frac{\mu^2}{D_1D_2} = -\frac{1}{6}P_1^2 = -\frac{1}{6}P_2^2 \ , \qquad (5.16)$$

where the $I_n[X]$ is the scalar integral with n propagators and a numerator term of X, D_i is the i^{th} propagator and P_i is the momenta flowing out of the i^{th} corner. Combining these with the coefficients above, the rational term is given by

$$R = -\frac{1}{6}\sum_i \tilde{d}_i^2 + \frac{1}{2}\sum_i\left(\frac{P_{i,1}^2 + P_{i,2}^2 + P_{i,3}^2}{12}\tilde{c}_i^5 + \tilde{c}_i^0\right) - \frac{1}{6}\sum_i P_{i[1]}^2\tilde{b}_i^0 \ , \qquad (5.17)$$

where $P_{i,j}$ is the momenta in the j^{th} corner of the cut i.

To calculate the amplitudes in the four dimensional helicity scheme, the relevant coefficients must be extracted using the methods from the previous chapter and combined to form the rational terms in both 6 and 8 dimensions. The rational terms in each dimension are then combined using Equation 5.6 to form the rational term in the four dimensional helicity scheme. Alternatively, as the relations are linear, the extrapolation of the spinor and polarisation state dimension dependence to 4 dimensions can be done at the coefficient level and then only combined to give the rational term once the FDH versions of each coefficient are found.

Working in higher numbers of dimensions increases the complexity of the calculations, as such it is always simpler to work in fewer dimensions and a lower number of different dimensions. It is possible to avoid working in 8 dimensions by simplifying the calculation so only a single dimension greater than 4 is needed by making use of the form of the Feynman rules, as is done by Giele, Kunszt and Melnikov[14] and Davies[16]. The single dimension now needed will be taken to be 6 dimensions. This is done by noticing that if D_l is taken as being 5, then D_1 and D_2 can be taken to be 5 and 6 respectively. Furthermore, if the amplitude is purely gluons then the difference between working in D_1 and D_2, will be the contribution due to gluons with their polarisation vectors pointing in the 6th dimension, which, if the Feynman rules can be extracted consistently, is equivalent to a complex scalar particle[14, 16]. The new Feynman rules needed for pure gluon amplitudes are just the normal three and four gluon vertex factors, but with one of the gluons forced to be in the 6th dimension, along with the propagator factor for gluons in the 6th dimension and gluon polarisation vectors in the 6th dimension. The polarisation vector must be given by

$$\epsilon_6^\mu = n_6^\mu \ , \tag{5.18}$$

where n_6^μ is the unit vector in the 6th dimension. On contracting this with a three gluon vertex it is clear that the combination will vanish, unless exactly one of the other gluons is also polarised in the direction of the 6th dimension. Therefore, taking two of the gluons, with momenta k_1 and k_2, as being polarised in the 6th dimension, the vertex factor is given by

$$-\frac{1}{\sqrt{2}}(k_1 - k_2)^\sigma n_6^\mu n_6^\nu \ , \tag{5.19}$$

where the σ index is contracted with the remaining gluon line and the indices μ and ν are contracted with gluons pointing in the 6th dimension. The same logic applies to the four gluon vertex, which requires that if any gluons are polarised in the 6th dimension, either two or four are. If two neighbouring gluons are polarised in the 6th dimension the four gluon vertex is given by

$$-\frac{i}{2}g^{\sigma\rho}n_6^{\mu}n_6^{\nu}\ , \tag{5.20}$$

where the indices μ and ν are contracted with the gluons polarised in the 6th dimension and the indices σ and ρ are contracted with the two remaining gluons which are next to each other. It is clear from the form of the gluon propagator that if the gluon at one end is polarised in the 6th dimension, the other end will also be, which allows it to be simplified to

$$\frac{in_6^{\mu}n_6^{\nu}}{k^2+i0}\ . \tag{5.21}$$

For the Higgs boson calculations the same extraction must be applied to the Higgs boson vertex rules. These two vertex rules obey the same relations as above and therefore will be given for the case of two gluons being in the direction of the 6th dimension. The vertex rule from the 2 gluon vertex is given by

$$-2ik_1\cdot k_2 n_6^{\mu}n_6^{\nu}\ , \tag{5.22}$$

where k_1 and k_2 are the momenta of the gluons polarised in the 6th dimension which are contracted with the indices μ and ν. The vertex rule derived from the 3 gluon vertex is given by

$$-i\sqrt{2}(k_1-k_2)^{\sigma}n_6^{\mu}n_6^{\nu}\ , \tag{5.23}$$

where k_1 and k_2 are the momenta of the gluons polarised in the 6th dimension which are contracted with the indices μ and ν and the remaining index, σ, is contracted with the remaining gluon.

From these relations it is clear that for the tree amplitudes in any corner of a generalised unitarity calculation, as there will be exactly two external gluons polarised in the 6th dimension, there must be a single line through the diagrams of gluons polarised in this direction and these gluons can all be treated as scalars using the new Feynman rules calculated above with the factors of n_6^{μ} stripped. The Feynman rules

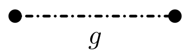
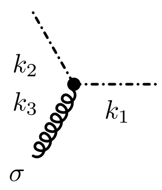
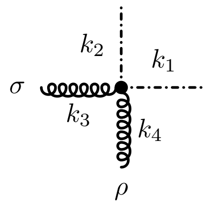
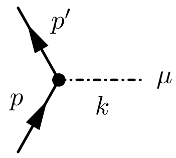
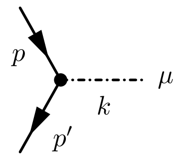
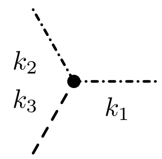
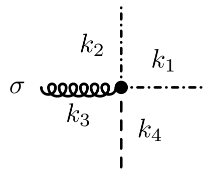
	$\frac{i}{k^2+i0}$		
	$-\frac{i}{\sqrt{2}}(k_1-k_2)^\sigma$		$-\frac{i}{2}g^{\rho\sigma}$
	$\frac{i}{\sqrt{2}}\not{\eta}_6$		$-\frac{i}{\sqrt{2}}\not{\eta}_6$
	$-2ik_1\cdot k_2$		$-i\sqrt{2}(k_1-k_2)^\sigma$

Table 5.1: Colour ordered Feynman rules in Faddeev-Popov gauge for the scalar particle equivalent to a gluon polarised in the 6th dimension. The dash-dotted lines are the new scalar and all momenta are inbound.

for the new complex scalar are given in Table 5.1.

The last case to handle is amplitudes with external quarks which enter the loop, which can have a mixture of quarks and gluons as the cut particles. This case can be handled by working with only one of the two copies of 4 dimensional spinors, to reduce the quark state dimension dependence, and then using scalar 6th dimensional gluons to subtract the extra gluon polarisation states. This is possible, as after reducing to only the 4 dimensional set of spinors, the dependence on D_s is due to terms with a closed loop of metric tensors and pairs of gamma matrices in a quark line, which again can result in at most linear dependence on D_s. The extra Feynman rules needed for this case are also shown in Table 5.1.

Therefore, if the calculations for the full 6 dimensional amplitudes and, where there are gluons in the loop, the subtraction terms using the scalar equivalent for the gluon, can be performed, the coefficients of scalar loop integrals in the amplitude can be extracted and then the rational terms can be calculated using only 6 dimensions.

To perform these calculations, an explicit numerical implementation must be derived and must be checked to ensure the properties used above are true. These aspects will be investigated in the next few sections.

5.1 6 Dimensional Spinor Helicity Formalism

The 6 dimensional extension of the spinor helicity formalism will enable efficient calculation of amplitudes in 6 dimensions by making use of the many cancellations and vanishing products, especially as many of the vectors are still purely 4 dimensional. To derive 6 dimensional spinors, a 6 dimensional generalisation of the gamma matrices is needed. The conditions these have to obey, as for the 4 dimensional case, are

$$\{\gamma^\mu, \gamma^\nu\} = g^{\mu\nu} \ , \tag{5.24}$$

where $g^{\mu\nu} = \mathrm{diag}(1, -1, -1, \ldots)$ is the metric of space time. There are many solutions to these equations but some are more convenient to use than others. This project uses a recursive definition, so that the different dimensions can be compared directly. In addition, a basis is used where as many elements as possible are zero, so that as many elements as possible in both spinors and products are zero and different elements mix as little as possible in products. The basis is also chosen to be based on the Weyl (chiral) basis as helicity spinors will be used and therefore a diagonal chirality operator, the γ_5 equivalent, will ensure different helicities do not mix. The chirality operator from now on will be referred to as γ_C, as γ_5 is the name of a normal gamma matrix in 6 or more dimensions. The basis chosen is[18]

$$\begin{aligned} \gamma_d^\mu &= \gamma_{d-2}^\mu \otimes \sigma_3 \qquad & \gamma_d^{d-2} &= iI_{d-2} \otimes \sigma_1 \qquad & \gamma_d^{d-1} &= -iI_{d-2} \otimes \sigma_2 \\ & & \gamma_2^0 &= \sigma_1 & \gamma_2^1 &= -i\sigma_2 \ , \end{aligned} \tag{5.25}$$

where I_d is the identity matrix in d dimensions which is given by

$$I_d = I_2 \otimes I_{d-2} \qquad\qquad I_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \ , \tag{5.26}$$

and the Pauli matrices, σ_i, are given by

$$\sigma_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad \sigma_2 = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \qquad \sigma_3 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} . \tag{5.27}$$

Using these definitions the 6 dimensional gamma matrices are given by

$$\gamma_6^0 = \begin{pmatrix} \begin{matrix} \begin{matrix} 0 & 1 \\ 1 & 0 \end{matrix} & 0 \\ 0 & \begin{matrix} 0 & -1 \\ -1 & 0 \end{matrix} \end{matrix} & 0 \\ 0 & \begin{matrix} \begin{matrix} 0 & -1 \\ -1 & 0 \end{matrix} & 0 \\ 0 & \begin{matrix} 0 & 1 \\ 1 & 0 \end{matrix} \end{matrix} \end{pmatrix} \quad \gamma_6^1 = \begin{pmatrix} \begin{matrix} \begin{matrix} 0 & -1 \\ 1 & 0 \end{matrix} & 0 \\ 0 & \begin{matrix} 0 & 1 \\ -1 & 0 \end{matrix} \end{matrix} & 0 \\ 0 & \begin{matrix} \begin{matrix} 0 & 1 \\ -1 & 0 \end{matrix} & 0 \\ 0 & \begin{matrix} 0 & -1 \\ 1 & 0 \end{matrix} \end{matrix} \end{pmatrix}$$

$$\gamma_6^2 = \begin{pmatrix} \begin{matrix} 0 & i \\ i & 0 \end{matrix} & 0 \\ 0 & \begin{matrix} 0 & -i \\ -i & 0 \end{matrix} \end{pmatrix} \quad \gamma_6^3 = \begin{pmatrix} \begin{matrix} 0 & 1 \\ -1 & 0 \end{matrix} & 0 \\ 0 & \begin{matrix} 0 & -1 \\ 1 & 0 \end{matrix} \end{pmatrix} \quad \gamma_6^4 = \begin{pmatrix} 0 & i \\ i & 0 \end{pmatrix} \quad \gamma_6^5 = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} , \tag{5.28}$$

where the matrices are given in block form and are all 8×8 matrices. Using this basis the γ_C is given by

$$\gamma_C = \begin{pmatrix} \begin{matrix} \begin{matrix} 1 & 0 \\ 0 & -1 \end{matrix} & 0 \\ 0 & \begin{matrix} -1 & 0 \\ 0 & 1 \end{matrix} \end{matrix} & 0 \\ 0 & \begin{matrix} \begin{matrix} -1 & 0 \\ 0 & 1 \end{matrix} & 0 \\ 0 & \begin{matrix} 1 & 0 \\ 0 & -1 \end{matrix} \end{matrix} \end{pmatrix} . \tag{5.29}$$

In hindsight, a basis that produced a chirality operator that was diagonal, with the top four values being one and the bottom four values being minus one, would have made the calculations simpler, as splitting the space into two using the helicity projector would have split spinors and matrices in half, top from bottom, whereas with this basis the space is split in a more complex pattern of rows, $1, 4, 6, 7$ and $2, 3, 5, 8$. However, as this only makes a difference to readability and simplicity of algebraic implementations, the basis defined in Equation 5.25 is used from here on.

Using this basis, spinors can be derived by using the defining equation

$$\not{p}u(p) = 0 \ , \tag{5.30}$$

where $u(p)$ is a spinor corresponding to the momentum p. After using this definition there are still four degrees of freedom left. One is removed by the normalisation condition of

$$\sum u(p)\bar{u}(p) = \not{p} \ , \tag{5.31}$$

where $\bar{u}$ is the conjugate spinor for u and for real momenta is defined by $\bar{u} = u^{\dagger}\gamma_0$. The others are used to split the general spinor into multiple spinor states. The first split is into the eigen spaces of the chirality operator, γ_C, which splits the spinor space into two subspaces, each with one remaining degree of freedom. So that the external particles, which are still 4 dimensional, have simple spinors, the last split should reduce to the eigen states of the 4 dimensional chirality operator, $\gamma_{6,4C}$, when the momenta is 4 dimensional. Unfortunately it is not possible to split the remaining spinor space by the 4 dimensional helicity operator for full 6 dimensional momenta. For this project, one of the simplest extensions away from the 4 dimensional limit was chosen, which is to keep the components that are already non-zero the same and only allow one extra component to become non-zero. Any choice for both gamma matrices and spinors should produce the same answers, other than an overall phase change on amplitudes, but this has not been tested directly as it would require reproducing the complete calculations for multiple choices of spinor basis.

Using these choices the spinors are given by

$$u_{++} = \begin{pmatrix} \sqrt{p_0+p_1} \\ 0 \\ 0 \\ \frac{p_3-ip_2}{\sqrt{p_0+p_1}} \\ 0 \\ \frac{p_5-ip_4}{\sqrt{p_0+p_1}} \\ 0 \\ 0 \end{pmatrix} \quad u_{-+} = \begin{pmatrix} 0 \\ \frac{p_5+ip_4}{\sqrt{p_0+p_1}} \\ 0 \\ 0 \\ \sqrt{p_0+p_1} \\ 0 \\ 0 \\ \frac{p_3-ip_2}{\sqrt{p_0+p_1}} \end{pmatrix} \quad u_{+-} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ -\frac{p_5+ip_4}{\sqrt{p_0+p_1}} \\ 0 \\ \frac{p_3+ip_2}{\sqrt{p_0+p_1}} \\ \sqrt{p_0+p_1} \\ 0 \end{pmatrix} \quad u_{--} = \begin{pmatrix} 0 \\ \frac{p_3+ip_2}{\sqrt{p_0+p_1}} \\ \sqrt{p_0+p_1} \\ 0 \\ 0 \\ 0 \\ 0 \\ -\frac{p_5-ip_4}{\sqrt{p_0+p_1}} \end{pmatrix} . \tag{5.32}$$

As in 4 dimensions, it is possible to write the conjugate spinors in terms of a combination of the transposed spinors. For this choice of spinors, the conjugates are given by

$$\bar{u}_{h-l} = ilu^T_{hl}\sigma_1 \otimes \sigma_2 \otimes \sigma_1 \equiv \underline{u}_{hl} \ , \tag{5.33}$$

where the underlined spinor, $\underline{u}_{hl}$, is the conjugate spinor, but is labelled by the helicity of the spinor it can be written in terms of and $i\sigma_1 \otimes \sigma_2 \otimes \sigma_1$ is a real anti-diagonal matrix. For example $\bar{u}_{++}$ is given by

$$\begin{aligned} \bar{u}_{++} &= u^\dagger_{++}\gamma_0 \\ &= \begin{pmatrix} \sqrt{p_0+p_1} & 0 & 0 & \frac{p_3+ip_2}{\sqrt{p_0+p_1}} & 0 & \frac{p_5+ip_4}{\sqrt{p_0+p_1}} & 0 & 0 \end{pmatrix}\gamma_0 \\ &= \begin{pmatrix} 0 & \sqrt{p_0+p_1} & -\frac{p_3+ip_2}{\sqrt{p_0+p_1}} & 0 & -\frac{p_5+ip_4}{\sqrt{p_0+p_1}} & 0 & 0 & 0 \end{pmatrix} , \end{aligned} \tag{5.34}$$

and $\underline{u}_{+-}$ is given by

$$\begin{aligned} \underline{u}_{+-} &= -iu^T_{+-}\sigma_1 \otimes \sigma_2 \otimes \sigma_1 \\ &= -i\begin{pmatrix} 0 & 0 & 0 & -\frac{p_5+ip_4}{\sqrt{p_0+p_1}} & 0 & \frac{p_3+ip_2}{\sqrt{p_0+p_1}} & \sqrt{p_0+p_1} & 0 \end{pmatrix}\sigma_1 \otimes \sigma_2 \otimes \sigma_1 \\ &= \begin{pmatrix} 0 & \sqrt{p_0+p_1} & -\frac{p_3+ip_2}{\sqrt{p_0+p_1}} & 0 & -\frac{p_5+ip_4}{\sqrt{p_0+p_1}} & 0 & 0 & 0 \end{pmatrix} . \end{aligned} \tag{5.35}$$

This shows that $\underline{u}_{+-} = \bar{u}_{++}$ as expected.

The spinor products for each helicity combination can be worked out, both in general from the relations and in the specific case of these spinors. The spinor products that are exactly zero are $\underline{u}_{+h_1}(p)u_{+h_2}(q)$ and $\underline{u}_{-h_1}(p)u_{-h_2}(q)$, where p and q are two general on-shell 6 dimensional momenta and h_1 and h_2 are arbitrary signs, which can be shown using the definition of the spinors. As the spinors are eigenstates of the 6 dimensional helicity operator, the appropriate sign projection operator,

$$P_{\pm} = \frac{1 \pm \gamma_C}{2} \ , \tag{5.36}$$

can be applied to each spinor. When applied to spinors and conjugate spinors the projection operators satisfy the relations

$$\begin{aligned} &P_{\pm}u_{\pm,h} = u_{\pm,h} && P_{\pm}u_{\mp,h} = 0 \\ &\underline{u}_{\mp,h}P_{\pm} = u_{\mp,h} && \underline{u}_{\pm,h}P_{\pm} = 0 \ . \end{aligned} \tag{5.37}$$

If the projection operators introduced for the two spinors in a spinor product have opposite signs then they will cancel and the product will vanish.

In general, the other spinor products are non-zero, however a similar logic to that used above, but using the 4 dimensional chirality operator, shows that in the limit of both momenta being 4 dimensional $\underline{u}_{h_1h_2}(p)u_{h_1h_2}(q)$ vanishes. The remaining spinor products, in this limit, must be given by the standard 4 dimensional spinor products, apart from possible changes in sign. For the choice of spinors used in this project, various spinor products can be related to each other. The full set of relations and their limits in the case of 4 dimensional momenta are given by,

$$\begin{aligned} &\underline{u}_{++}(p)u_{--}(q) = -\underline{u}_{--}(q)u_{++}(p) = \underline{u}_{--}(p)u_{++}(q) = -\underline{u}_{++}(q)u_{--}(p) \xrightarrow{p,q_{4,5}\to 0} 0 \\ &\underline{u}_{-+}(p)u_{+-}(q) = -\underline{u}_{+-}(q)u_{-+}(p) = \underline{u}_{+-}(p)u_{-+}(q) = -\underline{u}_{-+}(q)u_{+-}(p) \xrightarrow{p,q_{4,5}\to 0} 0 \\ &\underline{u}_{++}(p)u_{-+}(q) = \underline{u}_{-+}(q)u_{++}(p) = -\underline{u}_{-+}(p)u_{++}(q) = -\underline{u}_{++}(q)u_{-+}(p) = -\underline{u}_{+}(p)u_{+}(q) \\ &\underline{u}_{--}(p)u_{+-}(q) = \underline{u}_{+-}(q)u_{--}(p) = -\underline{u}_{+-}(p)u_{--}(q) = -\underline{u}_{--}(q)u_{+-}(p) = -\underline{u}_{-}(p)u_{-}(q) \ , \end{aligned} \tag{5.38}$$

where the spinors with only one sign are 4 dimensional spinors as given in Equation 2.19. In Section 5.2 even simpler explicit expressions for these products in terms of massless 4 dimensional projections will be derived.

The other main contribution needed for calculating amplitudes in 6 dimensions is a representation for polarisation vectors of gluons. As for the spinors, a representation that reduces to their 4 dimensional versions would be helpful as it would allow easy comparison to 4 dimensional amplitudes and calculations. Unfortunately, the simplest option of using the same expression as is used in 4 dimensions, as shown in Equation 2.21, does not obey the relations expected for polarisation vectors when used in 6 dimensions. It is possible to find a simple extension to the 4 dimensional case which does obey all the required relations, which is given by

$$\epsilon^{\mu}_{h,l,-h,m}(p;k) = \frac{\underline{u}_{hl}(p)\gamma^{\mu}\not{k}u_{-hm}}{2\sqrt{2}p\cdot k} , \tag{5.39}$$

where k is again a arbitrary on-shell vector that fixes the gauge for the gluon. This representation though does have the issue of there being eight different states where as physically there are known to be four gluon states. This issue is resolved by noticing that the states form pairs that are equivalent, up to a choice of sign, related by

$$\epsilon_{+,h,-,l}(p;k) = -hl\epsilon_{-,l,+,h}(p;k) . \tag{5.40}$$

Therefore, where convenient, the polarisation vector form will be simplified to

$$\epsilon_{h,l}(p;k) = \epsilon_{+,h,-,l}(p;k) . \tag{5.41}$$

The explicit mention of the arbitrary reference vector will also be dropped where there is only one polarisation vector and will be implicitly labelled as k.

The other expression needed when calculating amplitudes and expressions involving gluons is a formula for the contraction of a polarisation vector with a gamma matrix, either in another polarisation vector or in a spinor chain. This can be shown to be given by

$$\not{\epsilon}_{hl}(p) = \frac{\not{k}u_{-,l}(p)\underline{u}_{+,h}(p) - u_{-,l}(p)\underline{u}_{+,h}(p)\not{k} + hl\left(\not{k}u_{+,h}(p)\underline{u}_{-,l}(p) - u_{+,h}(p)\underline{u}_{-,l}(p)\not{k}\right)}{\sqrt{2}p\cdot k} , \tag{5.42}$$

where, when combined with any spinor chain or polarisation vector, at most two of the four terms will contribute.

5.2 Reducing 6 Dimensional Spinors to 4 Dimensions

A 6 dimensional vector can be decomposed into a 4 dimensional vector and a vector purely in the 5th and 6th dimensions,

$$p^\mu = p^\mu_{4D,M} + p^\mu_{6D} \,, \tag{5.43}$$

where p is the full 6 dimensional vector, $p_{4D,M}$ is the 4 dimensional part and p_{6D} is the part purely in the 5th and 6th dimensions. In components the vectors are given by

$$p = \begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{pmatrix} \qquad p_{4D,M} = \begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \\ 0 \\ 0 \end{pmatrix} \qquad p_{6D} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ p_4 \\ p_5 \end{pmatrix} \,. \tag{5.44}$$

Both $p_{4D,M}$ and p_{6D} are in general massive vectors and their dot product is zero as they have no components in common directions. Using an extra arbitrary massless 4 dimensional vector, a, it is possible to write the 4 dimensional part in terms of a massless 4 dimensional vector as

$$p^\mu_{4D} = p^\mu_{4D,M} - a^\mu \frac{p^2_{4D,M}}{2p_{4D,M} \cdot a} \,. \tag{5.45}$$

The vector p_{6D} can be written in terms of any pair of 4 dimensional vectors and the values of the 5th and 6th dimensional components. If the two vectors are chosen to be the vectors a and p_{4D}, p_{6D} can written as

$$p^\nu_{6D} = hl \frac{\mu^{hl}_p \underline{u}_{hl}(p_{4D})\gamma^\nu \not{a} u_{-h-l}(p_{4D}) - \mu^{-hl}_p \underline{u}_{h-l}(p_{4D})\gamma^\nu \not{a} u_{hl}(p_{4D})}{4p_{4D} \cdot a} \,, \tag{5.46}$$

where h and l are arbitrary signs and $\mu^\pm_p = p_5 \pm ip_4$ contains the values of the 5th and 6th dimensional components of the vector p.

The spinors for the full 6 dimensional vector, p, can also be written in terms of the spinors for their massless 4 dimensional projection and the arbitrary vector used in the massless projection by inserting the fraction $\frac{\not{a}\not{p}_{4D}+\not{p}_{4D}\not{a}}{2p_{4D}\cdot a}$, which due to the commutation properties of gamma matrices is equivalent to the identity, and gives

$$\begin{aligned} u_{h,l}(p) &= \frac{\not{a}\not{p}_{4D}+\not{p}_{4D}\not{a}}{2p_{4D}\cdot a}u_{hl}(p) \\ &= u_{h-l}(a)\frac{\underline{u}_{hl}(a)\not{p}_{4D}u_{hl}(p)}{2p_{4D}\cdot a}+u_{hl}(a)\frac{\underline{u}_{h-l}(a)\not{p}_{4D}u_{hl}(p)}{2p_{4D}\cdot a}+ \\ & u_{h-l}\left(p_{4D}\right)\frac{\underline{u}_{hl}\left(p_{4D}\right)\not{a}u_{hl}(p)}{2p_{4D}\cdot a}+u_{hl}\left(p_{4D}\right)\frac{\underline{u}_{h-l}\left(p_{4D}\right)\not{a}u_{hl}(p)}{2p_{4D}\cdot a}\ . \end{aligned} \tag{5.47}$$

The remaining dependence on the vector p is in the coefficients of each spinor, which can be evaluated in terms of p_a, a and $\mu_p^{\pm}$, once a choice is given for a.

For the rest of this project a is chosen to be

$$a^{\mu}=\frac{1}{2}\begin{pmatrix}1\\-1\\0\\0\\0\\0\end{pmatrix}, \tag{5.48}$$

as this gives one of the simplest forms for all the expressions for the spinor definition used. With this choice of reference vector the spinor can be written as

$$u_{h,l}(p)\to u_{h,l}(p_{4D})-hl\frac{\not{a}u_{-h,l}(p_{4D})}{2p_{4D}\cdot a}\mu_p^{-hl}\ . \tag{5.49}$$

The reason for this choice is to ensure that only two spinors appear in the replacement for the 6 dimensional spinor, rather than the four that can appear in general, and that they have different helicities. If this is ensured then in an expansion of the product of a spinor for a 6 dimensional vector and a spinor for any other 4 dimensional vector, at most one term will exist and in the expansion of a spinor product for two 6 dimensional vectors, at most two terms will exist. To ensure that only two spinors appear in the replacement for the 6 dimensional spinor and that they have different helicities requires that the products $\underline{u}_{hl}\left(p_{4D}\right)\not{a}u_{hl}(p)$ and $\underline{u}_{h-l}(a)\not{p}_{4D}u_{hl}(p)$ vanish. This choice of vector is the unique real vector that causes both of these products to vanish. It is also the vector that when contracted with a momenta gives the expression that will appear in the square roots within its spinors.

By applying this replacement to the definition of the conjugate spinor as given in Equation 5.33, the replacement for the conjugate spinor can be derived, which gives

$$\underline{u}_{h,l}(p) \to \underline{u}_{h,l}(p_{4D}) + hl\frac{\underline{u}_{-h,l}(p_{4D})\not{a}}{2p_{4D}\cdot a}\mu_p^{-hl} \ . \tag{5.50}$$

Combining this replacement with the replacement for spinors, a replacement for a slashed matrix can also be derived which is given by

$$\not{p} = \not{p}_{4D} + \frac{\not{a}}{2p_{4D}\cdot a}\mu_p^+\mu_p^- + \sum_{h,l=\pm}\frac{hl\left(\not{a}u_{-h,-l}(p_{4D})\underline{u}_{h,l}(p_{4D}) - u_{h,l}(p_{4D})\underline{u}_{-h,-l}(p_{4D})\not{a}\right)\mu_p^{hl}}{2p_{4D}\cdot a} \ . \tag{5.51}$$

Using these replacements on spinor products gives the relations

$$\underline{u}_{h,l}(p)u_{-h,l}(q) = \underline{u}_{h,l}(p_{4D})u_{-h,l}(q_{4D}) = -hl\underline{u}_l(p_{4D})u_l(q_{4D}) \tag{5.52}$$

$$\underline{u}_{h,l}(p)u_{-h,-l}(q) = hl\frac{1}{2}\left(\frac{\underline{u}_{-h,l}(p_{4D})\not{a}u_{-h,-l}(q_{4D})}{p_{4D}\cdot a}\mu_p^{-hl} + \frac{\underline{u}_{h,l}(p_{4D})\not{a}u_{h,-l}(q_{4D})}{q_{4D}\cdot a}\mu_q^{-hl}\right) \ , \tag{5.53}$$

where the right hand sides depend only on 4 dimensional vectors and can therefore be written as 4 dimensional spinor products.

5.3 Calculating 6 Dimensional Amplitudes

The amplitudes calculated in this project will always have physical external particles in 4 dimensions. This limits where the extra dimensions can be introduced into the tree and loop amplitudes being calculated. Therefore many of the amplitudes can be greatly simplified. It is also necessary to check the validity of the properties needed for 6 dimensional generalised unitarity calculations and for the state sum reduction using scalar particles.

The corners of the cut loops require tree amplitudes. These tree amplitudes will have external momenta for all but the two legs that are cut loop propagators. Therefore these are the only momenta that can have 6 dimensional components. As there are only two particles that have 6 dimensional components, by conservation of momentum, the 5th and 6th components of their momenta must be equal and opposite to the values in the other particles' momenta. A simple logic would also imply that as there are no other 6 dimensional vectors, the direction in the 5th and 6th components should be irrelevant and a rotation between the components should not change anything, but this needs to be proven explicitly and it must be shown precisely which types of amplitudes are guaranteed to show this behaviour. There are also restrictions on the helicity states that can appear for external and internal particles. In general, quarks and gluons both have twice as many states as they do in 4 dimensions, however the Higgs boson has one state, which is the same as in 4 dimensions.

To show that loop amplitudes do not depend on the direction of the 5th and 6th components of the momenta, the dependence of tree amplitudes on them must be shown. It must then be shown that any dependence cancels when used in generalised unitarity calculations. The simplest case is an n gluon amplitude which will therefore be handled first. Without loss of generality the first two particles are chosen to be the ones from the cut loop propagators and will be called p and q respectively. Using Feynman rules, the possible terms that can appear in any given amplitude can be calculated without evaluating the amplitude in full. For a pure gluon amplitude the Feynman rules are given in Table 2.1. To prove exactly what terms can be contributed for each diagram, one vertex can be chosen and starting with its vertex factor, new vertex factors and propagators connecting them can be added. These can then be split into separate terms. If this process is repeated, at each stage any term will have the form

$$C\frac{\prod_{i=1}^{m} P_i \cdot Q_i \prod_{j=1}^{N_g-2n} R_j^{\mu_j} \prod_{k=1}^{n} g^{\nu_k\sigma_k}}{\prod_{l=1}^{N_g+m-n-2} S_l^2}\ , \tag{5.54}$$

where C is a complex number, P, Q, R and S are momentum vectors containing some combination of neighbouring momentum from the total amplitude, m and n count how many of their respective types of factors are in the term and obey the relations

$$0 \le m \le \left\lfloor \frac{N_g}{2} - 1 \right\rfloor \qquad\qquad m+1 \le n \le \left\lfloor \frac{N_g}{2} \right\rfloor\ , \tag{5.55}$$

and N_g is the number of external indices or equivalently the number of gluon polarisation vectors the term needs to be contracted with to form an amplitude. The number of vertex factors of each type can be calculated as

$$\begin{aligned} N_3 &= N_g + 2m - 2n \\ N_4 &= n - m - 1\ , \end{aligned} \tag{5.56}$$

where N_3 is the number of three gluon vertices and N_4 is the number of four gluon vertices. This form will be proved in Appendix A.

When these terms are contracted with the gluon polarisation vectors, each polarisation vector will either contract with a free momenta, R, or with a metric, g, that will contract a pair of polarisation vectors together. From this, every term in an amplitude is of the form

$$C\frac{\prod_{i=1}^{m} P_i \cdot Q_i \prod_{j=1}^{N_g-2n} R_j \cdot \epsilon_j \prod_{k=1}^{n} \epsilon_k \cdot \epsilon_k'}{\prod_{l=1}^{N_g+m-n-2} S_l^2} , \tag{5.57}$$

where $\epsilon_{j/k}^{(\prime)}$ are gluon polarisation vectors.

Extending the general form of a term to an amplitude with a quark line, can be done by tracing along the quark line, contracting each gamma matrix from a vertex factor with a gluon term of the form in Equation 5.54. Each time a gluon term is added, either a free vector, R, or a metric, g, must be contracted with the gamma matrix in the vertex term. If a vector is contracted then it will introduce a slashed matrix into the quark line. If a metric is contracted then a free gamma matrix is left that will be contracted with a gluon polarisation vector and so introduce a slashed gluon polarisation vector to the quark line. In general therefore, the terms in an amplitude with one quark pair and any number of gluons are of the form

$$C\frac{\prod_{i=1}^{m} P_i \cdot Q_i \prod_{j=1}^{N_g-N_{qg}-2n} R_j \cdot \epsilon_j \prod_{k=1}^{n} \epsilon_k \cdot \epsilon_k'}{\prod_{l=1}^{N_g+m-n-N_{qR}-1} S_l^2} \bar{u}\left(\prod_{a}^{N_{qR}+N_{qg}-1} \not{U}_a \not{T}_a\right) \not{U}_{(N_{qR}+N_{qg})} u , \tag{5.58}$$

where N_{qg} of the U_as are gluon polarisation vectors and the remaining N_{qR} of the U_as, along with T_as, are momentum vectors containing some combination of neighbouring momenta from the total amplitude and the summation limits now obey the conditions

$$\begin{aligned} 0 \leq m &\leq \left\lfloor \frac{N_g + N_{qR} + N_{qg}}{2} \right\rfloor - N_{qR} - N_{qg} \\ m + N_{qR} \leq n &\leq \left\lfloor \frac{N_g + N_{qg} + N_{qR}}{2} \right\rfloor - N_{qg} \ . \end{aligned} \tag{5.59}$$

To show how the amplitudes depend on the direction of the extra components of vectors, it is required to show how the various types of term depend on the extra dimensions. Three of the types of term needed are products of two momentum vectors, products of two polarisation vectors and products of polarisation vectors and momentum vectors. The final type of term is spinor chains with an odd number of gamma matrices, each contracted with either a momenta or a polarisation vector. Firstly, products of two momenta will be considered. They can only depend on the 4th and 5th dimensional components if one or both of them contains one of the loop propagator momenta, p or q. Without loss of generality, both of the momenta in a product can be expanded as the sum of a possibly massive momentum vector, which contains all contributions from any other momenta and one, both or none of the loop momenta. It is then possible to expand the products into multiple terms. The terms that this can produce and their expansions in terms of their 4 dimensional equivalent vectors are

$$R \cdot S = R \cdot S \tag{5.60}$$

$$R \cdot p = \frac{\mu_p^- \mu_p^+ R \cdot a}{2p_{4D} \cdot a} + R \cdot p_{4D} = p_{4D,M} \cdot R \tag{5.61}$$

$$R \cdot q = \frac{\mu_q^- \mu_q^+ R \cdot a}{2q_{4D} \cdot a} + R \cdot q_{4D} = q_{4D,M} \cdot R \tag{5.62}$$

$$p \cdot p = q \cdot q = 0 \tag{5.63}$$

$$\begin{aligned} p \cdot q &= q_{4D} \cdot p_{4D} + \frac{\left(\mu_q^+ p_{4D} \cdot a - \mu_p^+ q_{4D} \cdot a\right)\left(\mu_q^- p_{4D} \cdot a - \mu_p^- q_{4D} \cdot a\right)}{p_{4D} \cdot a q_{4D} \cdot a} \\ &= q_{4D,M} \cdot p_{4D,M} - \frac{1}{2}\left(\mu_q^- \mu_p^+ + \mu_p^- \mu_q^+\right) \ , \end{aligned} \tag{5.64}$$

where R and S are the massive vectors containing only 4 dimensional momenta. All of these terms only depend on $\mu^{\pm}_{p/q}$ through the combinations $\mu_p^+\mu_p^- = \mu_q^+\mu_q^-$ and $\mu_p^+\mu_q^- = \mu_q^+\mu_p^- = -\mu_p^+\mu_p^-$. These forms are the only ways that $\mu^{\pm}_{p/q}$ can occur in Minkowski products of momenta.

The next form that occurs is $R_j \cdot \epsilon_j$. As for Minkowski products of momenta, the momenta, R, can be expanded as a 4 dimensional massive vector and possibly one or both of the 6 dimensional vectors. The resulting terms can be expanded in terms of 4 dimensional equivalent vectors as

$$\epsilon_{hh}(r).S = \frac{\underline{u}_{+h}(r)\$\not{k}\underline{u}_{-h}(r)}{2\sqrt{2}r \cdot k} \tag{5.65}$$

$$
\begin{aligned}
\epsilon_{hh}(r).p &= \frac{\underline{u}_{+h}(r)\not{p}\not{k}\underline{u}_{-h}(r)}{2\sqrt{2}r\cdot k} \\
&= \frac{\underline{u}_{+h}(r)\not{a}\not{k}\underline{u}_{-h}(r)\mu_p^+\mu_p^-}{4\sqrt{2}r\cdot k p_{4D}\cdot a} + \frac{\underline{u}_{+h}(r)\not{p}_{4D}\not{k}\underline{u}_{-h}(r)}{2\sqrt{2}r\cdot k} \\
&= \frac{\underline{u}_{+h}(r)\not{p}_{4D,M}\not{k}\underline{u}_{-h}(r)}{2\sqrt{2}r\cdot k} && (5.66) \\
\epsilon_{hh}(p).S &= \frac{\underline{u}_{+h}(p)\not{S}\not{k}\underline{u}_{-h}(p)}{2\sqrt{2}p\cdot k} \\
&= \frac{\underline{u}_{+h}(p_{4D})\not{a}\not{S}\not{k}\not{a}\underline{u}_{-h}(p_{4D})\mu_p^+\mu_p^-}{8\sqrt{2}\left(p_{4D}\cdot k+\frac{k\cdot a}{2p_{4D}\cdot a}\mu_p^+\mu_p^-\right)(p_{4D}\cdot a)^2} + \frac{\underline{u}_{+h}(p_{4D})\not{S}\not{k}\underline{u}_{-h}(p_{4D})}{2\sqrt{2}\left(p_{4D}\cdot k+\frac{k\cdot a}{2p_{4D}\cdot a}\mu_p^+\mu_p^-\right)} \\
&= \frac{\underline{u}_{+h}(p_{4D})\not{a}\not{S}\not{k}\not{a}\underline{u}_{-h}(p_{4D})\mu_p^+\mu_p^-}{8\sqrt{2}p_{4D,M}\cdot k\,(p_M\cdot a)^2} + \frac{\underline{u}_{+h}(p_{4D})\not{S}\not{k}\underline{u}_{-h}(p_{4D})}{2\sqrt{2}p_{4D,M}\cdot k} && (5.67) \\
\epsilon_{h-h}(p).S &= \frac{\underline{u}_{+h}(p)\not{S}\not{k}\underline{u}_{--h}(p)}{2\sqrt{2}p\cdot k} \\
&= h\frac{S\cdot ap_{4D}\cdot k-k\cdot ap_{4D}\cdot S}{\sqrt{2}p_{4D}\cdot a\left(p_{4D}\cdot k+\frac{k\cdot a}{2p_{4D}\cdot a}\mu_p^+\mu_p^-\right)}\mu_p^{-h} \\
&= h\frac{S\cdot ap_{4D,M}\cdot k-k\cdot ap_{4D,M}\cdot S}{\sqrt{2}p_{4D,M}\cdot ap_{4D,M}\cdot k}\mu_p^{-h}\ , && (5.68)
\end{aligned}
$$

where only terms involving p are shown, as it is easy to produce the equivalent terms involving q from the versions shown. For these cases it can be seen that factors of $\mu_p^+\mu_p^-$ appear in many places. However, lone factors of μ_p^+ or μ_p^- only appear in terms from gluons with opposite helicities and always have the sign of the μ as the second sign of the helicity. The other case, involving contracting the polarisation vector for one of the 6 dimensional momenta with the other 6 dimensional momenta, show the same pattern as for the ones shown above but are too complex to show here.

The same can be done for the contraction of two polarisation vectors and again the same relations apply. Now, as there are two polarisation vectors, it is possible for both gluons to have the same pair of helicities and for the two helicities on each gluon to be opposite, i.e. $\epsilon_{h-h}\cdot\epsilon_{h-h}$. In this situation the terms will have an overall factor of ${\mu_p^{\pm}}^2$. If the two gluons have exactly opposite polarisations and both have their two signs opposite, i.e. $\epsilon_{h-h}\cdot\epsilon_{-hh}$, then there will be no overall factor of $\mu_p^+\mu_p^-$ as factors of $\mu_p^+\mu_p^-$ can be converted to Minkowski products of vectors. Overall, this means that each term in a pure gluon amplitude can contain many factors of $\mu_p^+\mu_p^-$, but any lone factors of $\mu_p^{\pm}$ always come from polarisation vectors with their two helicities opposite. The sign of the factors depends on the helicities of the most common type of gluon with opposite signs. If the most common type of gluon is ϵ_{h-h} then the amplitude will have overall factors of μ_p^{-h} and the number of factors will be $\#\epsilon_{h-h}-\#\epsilon_{-hh}$, where $\#\epsilon_{hl}$ means the number of polarisation vectors of type ϵ_{hl}.

For all pure gluon amplitudes needed to calculate the rational parts, there can only be two gluons that have these helicities, so at most there will be two overall factors of μ and that will only be the case if both the gluons have the same helicities. As the signs of the gluons either side of a cut are related, the overall factors of $\mu^{\pm}$ for the product of the amplitudes either side of the cut must be the same as for a single amplitude without the two gluons from the cut. Therefore, the full combination of tree amplitudes used to build a cut contribution must have no overall dependence on $\mu^{\pm}$ and must only depend on $\mu^{\pm}$ via the combination $\mu^{+}\mu^{-}$.

To extend this to amplitudes with quarks, first requires proving the forms of tree amplitudes with quarks in and then again combining them to show that any overall factors of $\mu^{\pm}$ cancel. The only extra contribution needed for quark amplitudes is knowing the dependence of one or more quark lines on $\mu^{\pm}$. Firstly the cases of only a single spinor chain will be considered. It is clear from the vanishing products and the expansion of slashed matrices in terms of spinors that any quark line will vanish if the quarks at either end have opposite values for their first helicity signs. This separates the sets of amplitudes into two spaces as is required. It still remains to show that the two spaces are both equivalent but this just requires the amplitudes to not depend on the value of the first helicity sign. To simplify the relations all slashed momenta will be expanded as above, but this time also replacing any instances of the loop momenta q with minus the sum of all the other momenta using conservation of momenta. This leaves spinor chains with an odd number of elements which are either $\not{p}$, slashed polarisation vectors for p or q, a possibly massive 4 dimensional slashed momenta or slashed polarisation vectors for 4 dimensional external gluons. To further simplify the cases that need to be handled, any $\not{p}$ can be commuted to the start of the spinor line, followed by the polarisation vector for p if present, then the polarisation vector for q and finally followed by all the purely 4 dimensional slashed vectors and polarisation vectors. Each of these swaps will also introduce an extra term of the form of a spinor chain with the two swapped elements removed, multiplied by the Minkowski product of the two items removed. These extra terms will also always have an odd number of slashed matrices and be of the same form as the original spinor chain but with less elements in them. The simplest case to handle is an amplitude with the two cut particles being the quarks and no other quarks in the amplitude. Then the form of the spinor lines can always be converted to the spinor for the quark p followed by a chain of an odd number of slashed momenta or polarisation vectors which are all purely 4 dimensional. Any spinor chain terms that have a $\not{p}$ in them will result in terms with no $\not{p}$ in them once the commutations are applied as any $\not{p}$ will be commuted to the start and then cause the term to vanish as $\underline{u}(p)\not{p} = 0$. There now remain two forms to calculate for this case which depend on the helicities of the quarks and are given by

$$\underline{u}_{hl}(p) \overbrace{\not{X}}^{n\times} u_{h-l}(q) = \underline{u}_{hl}(p_{4D}) \overbrace{\not{X}}^{n\times} u_{h-l}(q_{4D}) + \frac{\underline{u}_{-hl}(p_{4D})\not{a} \overbrace{\not{X}}^{n\times} \not{a} u_{-h-l}(q_{4D})}{4p_{4D} \cdot a q_{4D} \cdot a} \mu^+ \mu^- \tag{5.69}$$

$$\underline{u}_{hl}(p) \overbrace{\not{X}}^{n\times} u_{hl}(q) = \frac{hl}{2} \left(\frac{\underline{u}_{hl}(p_{4D})\not{a} \overbrace{\not{X}}^{n\times} u_{-hl}(q_{4D})}{p_{4D} \cdot a} + \frac{\underline{u}_{-hl}(p_{4D}) \overbrace{\not{X}}^{n\times} \not{a} u_{hl}(q_{4D})}{q_{4D} \cdot a} \right) \mu^{-hl} , \tag{5.70}$$

where $\not{X}$ represents the n slashed matrices in the chain and n is odd. From these forms it can be seen that amplitudes with a single quark line, where the two quarks are the cut loop propagators, have an overall factor of μ^{-hl} if the quarks are both of helicity hl and have no overall factors of $\mu^{\pm}$ otherwise. All other dependence on $\mu^{\pm}$ is via the combination $\mu^+\mu^-$.

The next simplest case is an amplitude with one of the cuts being a quark and the other being a gluon. Without loss of generality the cut quark will be chosen to be p and the cut gluon will be chosen to be q. After applying all the simplifications and relations as above, two types of spinor chain remain. Firstly are spinor chains with only 4 dimensional slashed matrices and secondly are chains with the slashed polarisation vector, $\not{\epsilon}(q)$, at the start. The different cases for these terms are given by

$$\underline{u}_{hl}(p) \overbrace{\not{X}}^{n\times} u_{h-l}(r) = \underline{u}_{hl}(p_{4D}) \overbrace{\not{X}}^{n\times} u_{h-l}(r) \tag{5.71}$$

$$\underline{u}_{hl}(p) \overbrace{\not{X}}^{n\times} u_{hl}(r) = hl \frac{\underline{u}_{-hl}(p_{4D})\not{a} \overbrace{\not{X}}^{n\times} u_{hl}(r)}{2p_{4D} \cdot a} \mu^{-hl} \tag{5.72}$$

$$\underline{u}_{hl}(p)\not{\epsilon}_{m,m}(q) \overbrace{\not{X}}^{n-1\times} u_{h-l}(r) \propto 1 \tag{5.73}$$

$$\underline{u}_{hl}(p)\not{\epsilon}_{m,-m}(q) \overbrace{\not{X}}^{n-1\times} u_{h-l}(r) \propto \mu^{-m} \tag{5.74}$$

$$\underline{u}_{hl}(p)\not\epsilon_{m,m}(q)\overbrace{\not X}^{n-1\times}u_{hl}(r)\propto\mu^{-hl} \tag{5.75}$$

$$\underline{u}_{hl}(p)\not\epsilon_{h,-l,-h,l}(q)\overbrace{\not X}^{n-1\times}u_{h-l}(r)\propto 1 \tag{5.76}$$

$$\underline{u}_{hl}(p)\not\epsilon_{h,l,-h,-l}(q)\overbrace{\not X}^{n-1\times}u_{h-l}(r)\propto\left(\mu^{-hl}\right)^2\ , \tag{5.77}$$

where in the later terms, factors that do not depend on $\mu^{\pm}$, or only depend on $\mu^{\pm}$ via the combination $\mu^+\mu^-$, have been removed as the expressions are too complex to show in full here.

The last type of term that can contribute to an amplitude with a single quark line is the case where the cut particles are both gluons. If the connection between the two 6 dimensional gluons does not include the quark line, then the quark line is independent of p and q and the dependence on $\mu^{\pm}$ must come from pure gluon factors. The remaining cases are too complex to show here but have been calculated and are found to obey the same relations as above, namely that each gluon of helicity $\epsilon_{h,-h}$ and each quark line where the quarks have helicities $\underline{u}_{hl}$ and u_{hl}, contributes a factor of μ^{-hl} or removes a factor of μ^{hl}. These relations can also be shown to work for multiple quark lines by examining the new cases contributed. For pure gluon amplitudes it has already been shown above that these factors of $\mu^{\pm}$ cancel between amplitudes when combined for a generalised unitarity contribution. Using the same logic, combined with the formulas derived above for the dependence of quark lines, it is clear that the overall factors of $\mu^{\pm}$ for any product of tree amplitudes is again the same as a single amplitude without the cut quarks. Therefore, for amplitudes with quark lines in, there again must be no overall dependence on $\mu^{\pm}$ for the combination of tree amplitudes needed for a generalised unitarity term. In addition, for any generalised unitarity contribution for an amplitude which includes a combination of quarks, gluons and optionally a Higgs boson, the only dependence on $\mu^{\pm}$ is via the combination $\mu^+\mu^-$.

Therefore, it has been shown that the exact value of the components p_4 and p_5 of the loop momenta are irrelevant and can be chosen freely without affecting the results of the generalised unitarity calculations and that μ^+ and μ^- can be chosen to have the same value and to always have a positive real part or be pure imaginary with a positive imaginary part. This reduces the two extra complex components of the momenta to one extra complex value for $\mu=\mu^+=\mu^-$.

To show that the quark state reduction works, requires showing that amplitudes do not depend on which of the two quark spaces they are from, other than an overall sign shared by all terms. The effect from the quark helicities is only able to affect the quark lines in each term. For the cases with no cut quarks, the two spaces give exactly the same value as they must have 4 dimensional type helicities. If the quarks are cut then there are more options available and the tree amplitudes, in general, will depend on which quark space is used, due to the factors of hl and μ^{-hl} that appear, but these always cancel to terms of the form $\mu^+\mu^-$, while also cancelling the dependence on which space they are from. Therefore, both spaces will give equal contributions and the state sum reduction works.

It now remains to show that the extra amplitudes needed for the gluon state reduction also simplify and obey the relations expected. For scalar subtraction amplitudes for pure gluon amplitudes, the terms that can contribute are related to those for the pure gluon amplitude, by the requirement that for any term of the gluon amplitude where the two cut gluons would have been contracted with each other using a metric tensor, there is a term of the same form but with that metric tensor removed. Each corner in these amplitudes must depend on μ^+ and μ^- only via the combination $\mu^+\mu^-$, as all contributions that give overall factors of $\mu^\pm$, were shown earlier to be contractions of the polarisation vectors for the cut gluons with either each other, other polarisation vectors or momentum vectors and these factors can not appear in these terms. Therefore, these subtraction terms are again independent of the direction of the extra components and could even be calculated with the loop momenta taken in a direction that includes the 6th dimension, which would not otherwise have been permitted.

For amplitudes with external quarks, as well as some gluons in the loop, if any corners do not contain the quarks, then by the same logic as above, those corners must only depend on μ^+ and μ^- via the combination $\mu^+\mu^-$. If all the cut particles in a contribution are gluons and each quark pair is isolated to a single corner, then the only case to consider is amplitudes like those shown on the left of Figure 5.2. Many diagrams in these amplitudes will not have the scalars enter the quark line and will therefore have the two lines connected by a gluon. For this case, the same logic as for pure gluon amplitudes can be used to show that again the term must have no overall factors of μ^+ or μ^- and can only depend on $\mu^\pm$ via the combination $\mu^+\mu^-$. The remaining type of diagram for this type of corner is where the scalar lines both enter the quark line and is shown in the rightmost diagram in Figure 5.2. For this

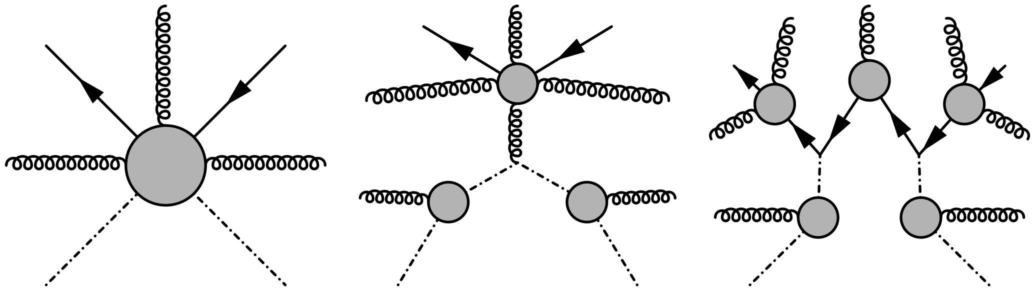


Figure 5.2: The general form of the scalar subtraction amplitude for a corner with a quark pair that does not enter the loop, along with the two types of term that contribute to it. The dash-dotted lines represent the scalar corresponding to gluons in the 6th dimension. An external gluon shown in the diagrams can represent any number of external gluons, including none. The cut particles are always the bottom two particles in each diagram and shaded circles represent all possible colour ordered diagrams that contribute to the amplitude or type of term.

case the term that is new and could potentially break the assumptions, is the quark line with the scalar lines entering it, as these introduce explicit factors of $\not{n}_6$, which is not a massless physical momentum vector. Again these terms will be handled by commuting these factors to the very end of the quark chain, which will leave one term with both of these slashed matrices at the end and many terms with one or both of these vectors removed from the chain and contracted with another contribution from the line. The term which still has both factors of $\not{n}_6$ present and has them adjacent to each other can then be simplified by removing these two slashed matrices, as

$$\not{n}_6\not{n}_6 = n_6^2 I = -I \ , \tag{5.78}$$

where I is the identity matrix for the space of gamma matrices. This term therefore gives a contribution without overall factors of μ^+ or μ^- and only depends on $\mu^\pm$ via the contribution $\mu^+\mu^-$. The other terms produced by this commutation will contain chains with one or zero factors of $\not{n}_6$, multiplied by one or two factors, each of the form of n_6 contracted with either a momentum vector or an external polarisation vector. These extra factors will all vanish as the loop momenta is restricted to only be in 5 dimensions and the polarisation vectors are for external gluons with 4 dimensional polarisations and momenta and either of these will vanish when contracted with n_6.

The remaining type of corner that needs investigation is those that have a quark for one of the cut particles and a gluon for the other as shown in Figure 5.3. Again each

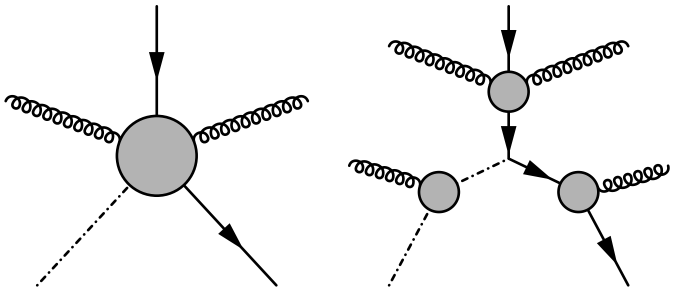


Figure 5.3: The general form of the scalar subtraction amplitude for a corner with a single quark that enters the loop, along with the form of the only type of term that contributes to it. The dash-dotted lines represent the scalar corresponding to gluons in the 6th dimension. An external gluon shown in the diagrams can represent any number of external gluons, including none. The cut particles are always the bottom two particles in each diagram and shaded circles represent all possible colour ordered diagrams that contribute to the amplitude or type of term.

term will be commuted so that a fixed order of the slashed matrices is achieved. Here, as for the other quark corner above, any extra terms produced by the commutation of $\not{n}_6$ must vanish as they will introduce factors of n_6 contracted with either a loop momenta or an external polarisation vector. The quark line will contain exactly one factor of $\not{n}_6$ and a combination of slashed external polarisation vectors and slashed momenta, which will contain a combination of the loop momenta, l, and external momenta. As for the normal amplitudes, if there are multiple factors of the loop momenta, they can be commuted to be next to each other and will then vanish. Therefore, there can be at most one loop momenta left in the amplitude after this. The remaining factor of the loop momenta will also vanish if it is commuted to the end of the quark line that is the cut quark, leaving just terms with a single factor of $\not{n}_6$ next to the spinor for the cut quark. This quark line is potentially problematic as it may introduce factors of $\mu^{\pm}$ that do not combine with a factor of the opposite sign. Using the properties of quark lines derived above, the dependence of these terms on μ can be shown to be

$$\underline{u}_{hl}(p)\not{n}_6 \overbrace{\not{X}}^{n-1\times} u_{hl}(r) = l\underline{u}_{hl}(p_{4D})\left(u_{-hl}(n_{6,4D})\underline{u}_{-hl}(a) - u_{-hl}(a)\underline{u}_{-hl}(n_{6,4D})\right) \overbrace{\not{X}}^{n-1\times} u_{hl}(r) \tag{5.79}$$

$$\underline{u}_{hl}(p)\not{n}_6 \overbrace{\not{X}}^{n-1\times} u_{h-l}(r) = -h\underline{u}_{hl}(p_{4D})\not{a}u_{-h,-l}(n_{6,4D})\underline{u}_{-h-l}(a) \overbrace{\not{X}}^{n-1\times} u_{h-l}(r)\frac{\mu_p^{-hl}}{2p_{4D}\cdot a}\ , \tag{5.80}$$

where $n_{6,a}$ is a 4 dimensional vector introduced as the 4 dimensional equivalent to n_6. $n_{6,a}$ is derived by splitting $2n_6$ into the difference between two massless 6 dimensional vectors, n_6^+ and n_6^-, which are given by

$$n_6^+ = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \qquad n_6^- = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ -1 \end{pmatrix}, \tag{5.81}$$

and then applying the normal reduction to 4 dimensional vectors. Both of these vectors, when reduced, result in the same 4 dimensional massless vector, $n_{6,a}$ which is given by

$$n_{6,4D} = \frac{1}{2} \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}. \tag{5.82}$$

As all terms in these amplitudes must have a factor of this form, the overall dependence on $\mu^\pm$ is consistent, but it is no longer as it was for normal amplitudes. This is potentially problematic as now factors of $\mu^\pm$ will no longer cancel and there could be overall factors of $\mu^\pm$ that depend on the helicity of the cut terms. If the two corners of this form are neighbours, then there are two cases for the internal dependencies, which are given in Table 5.2. If there are other corners between these two corners, then there are four cases, as all that matters is the helicites of the external and cut quarks next to the corners where the quarks exit the diagram because, as found above, multiple neighbouring corners with both cut particles being quarks have the same overall factors of $\mu^\pm$ as one corner without the internal cuts. These four cases are also shown in Table 5.2. These terms, as shown in the table, are also well behaved and depend on $\mu^\pm$ only via the combination $\mu^+\mu^-$ and as such all of the calculations are independent of the direction of the extra components, other than the requirement that $l \cdot n_6 = 0$ when calculating these extra cut terms.

As all the amplitudes are independent of the value of the components of the loop momenta in the extra dimensions, it would greatly simplify implementing calculations

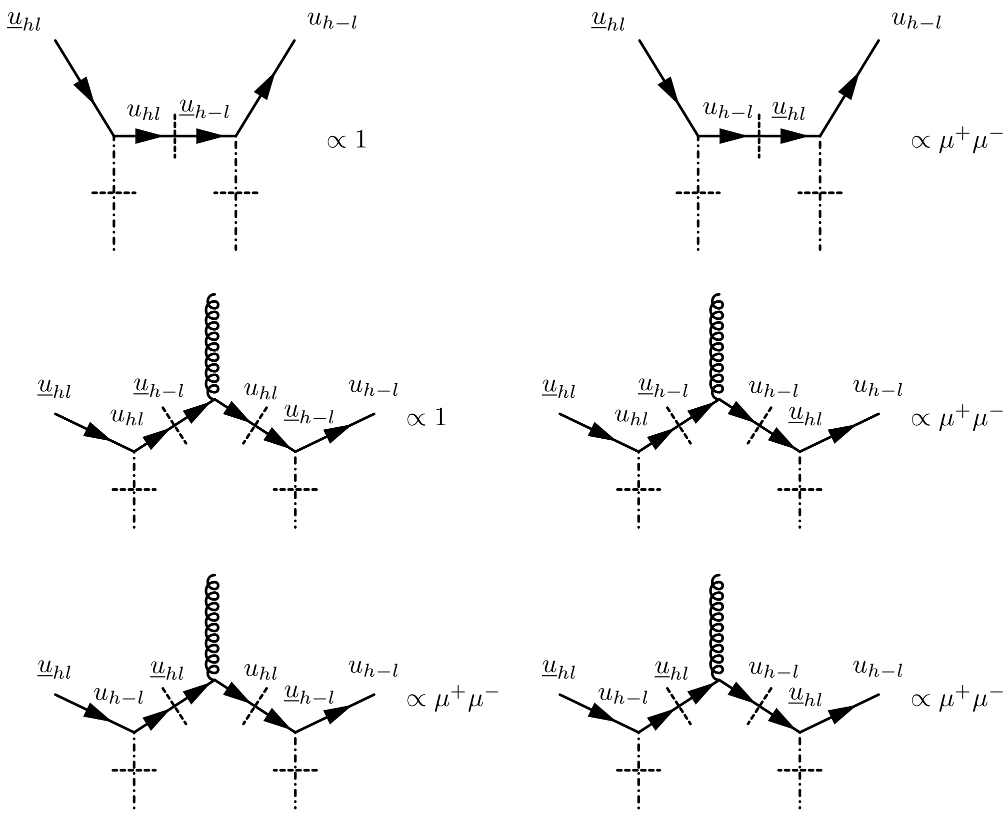


Table 5.2: The different types of quark line dependence in generalised unitarity terms for the scalar gluon subtraction terms. The top row are the contributions if the corners where the quark line exits the diagram are next to each other and the other rows are for if there are any number of extra corners between them. Only one extra corner is shown, as the contribution from a series of neighbouring corners with both cut particles being quarks, combines to give the same contribution as only a single corner with the same edge cut quarks. Any extra external gluons in any corner are not shown as the contribution depends only on the quark line and scalar gluons.

if all amplitudes were calculated in terms of the massless 4 dimensional projected momenta and the value of μ^2. This will require new effective particle types that label the particles that are 6 dimensional. For these particles, their true momenta must be calculated by combining the massless projection with the given value for μ^2. To actually calculate these amplitudes will still require using the full 6 dimensional Feynman rules, but all external momenta will be purely 4 dimensional and all factors and terms in the amplitude will be written in terms of purely 4 dimensional spinor products, Minkowski products, constants and the value of μ^2. It is important to note that conservation of momenta in amplitudes involving external particles that are 6 dimensional is more complex, as it is the total 6 dimensional momenta that is conserved not the reduced momenta. The exact relation between the projected on-shell momenta of two cut on-shell 6 dimensional particles is

$$l^{\mu}_{i+1,4D} = l^{\mu}_{i,4D} - P^{\mu} - a^{\mu} \frac{P \cdot a}{2 l_{i,4D} \cdot a \left(l_{i,4D} \cdot a - P \cdot a \right)} \mu^{+} \mu^{-} \ , \tag{5.83}$$

where the full 6 dimensional momenta satisfy the relation $l^{\mu}_{i+1} = l^{\mu}_{i} - P^{\mu}$ and $l_{i+1,4D}$ will be massless if and only if the full 6 dimensional vector would have been massless.

After a complete set of the lowest multiplicity and simplest amplitudes is calculated explicitly from the Feynman rules, the BCFW recursion relation can be used to calculate the higher multiplicity amplitudes, by applying it to the purely 4 dimensional momenta. However, care must be taken to use the modified conservation of momenta rule given above and to sum over all possible states including the extra 6 dimensional ones, if the two 6 dimensional momenta are on opposite sides of a cut. The exact formula that combines the two amplitudes on either side of the cut to give the contribution will also need to be calculated. It is also important to check which choices of shifted particles will be valid. These conditions are not yet known and are likely to be complex if one of the 6 dimensional particles is chosen.

5.4 Calculating the Rational Terms using 6 Dimensional Spinors

Once tree amplitudes can be calculated, the next step is to combine them to produce the coefficients of the scalar basis integrals and then combine the coefficients with the values of the basis integrals to give the rational terms. For numeric calculations an explicit numerical expression for the loop momenta is again required. As before, it would be convenient if the loop momenta could be written in terms of spinors and vectors for the external momenta. If the cut conditions are evaluated in terms of the massive 4 dimensional loop momenta, then the conditions extend in a relatively simple way and give the loop momentum for three or more cuts of

$$l^{2}_{2,4D,M} = c_1 n^{\mu}_{1} + c_2 n^{\mu}_{2} + \tilde{K}^{\mu}_{1} \frac{K_2^2 (K_1^2 + \gamma)}{4\Delta} - \tilde{K}^{\mu}_{2} \frac{K_1^2 (K_2^2 + \gamma)}{4\Delta} \ , \tag{5.84}$$

where c_1 and c_2 now satisfy the conditions

$$c_1 c_2 = - \frac{\gamma \left(\mu^2 + \frac{K_1^2 K_2^2 (K_1^2 + K_2^2 + 2 K_1 \cdot K_2)}{4\Delta} \right)}{4\Delta} \ , \tag{5.85}$$

and if a fourth cut is performed they are given by

$$c_1 = \frac{X\left(1 \pm \sqrt{1 + \frac{\Delta_4}{X^2\Delta}\left(\frac{K_1^2 K_2^2 (K_1^2 + K_2^2 + 2K_1 \cdot K_2)}{\Delta} + 4\mu^2\right)}\right)}{4K_3 \cdot n_1} \tag{5.86}$$

$$c_2 = \frac{X\left(1 \mp \sqrt{1 + \frac{\Delta_4}{X^2\Delta}\left(\frac{K_1^2 K_2^2 (K_1^2 + K_2^2 + 2K_1 \cdot K_2)}{\Delta} + 4\mu^2\right)}\right)}{4K_3 \cdot n_2} \,, \tag{5.87}$$

where the conventions for the directions of momenta and definitions for $\tilde{K}_1$, $\tilde{K}_2$, n_1, n_2, γ, Δ, Δ_4 and X are the same as for Equation 4.28. As this definition is massive it does not have a spinor representation, but its massless projection does. To project down into a spinor form, the vector a must be expanded in terms of the basis of $\tilde{K}_1$, $\tilde{K}_2$, n_1 and n_2, so that extra linearly dependant vectors and spinors are not introduced. In terms of this basis, a is given by

$$a^\mu = \frac{\tilde{K}_1^\mu \left(K_2^2 K_1 \cdot a - \gamma K_2 \cdot a\right) + \tilde{K}_2^\mu \left(K_1^2 K_2 \cdot a - \gamma K_1 \cdot a\right) + n_1^\mu \gamma n_2 \cdot a + n_2^\mu \gamma n_1 \cdot a}{2K_1^2 K_2^2 - 2\left(K_1 \cdot K_2\right)^2} \,, \tag{5.88}$$

where $n_1 \cdot a$ and $n_2 \cdot a$ are related by

$$n_1 \cdot a \; n_2 \cdot a = \frac{(\gamma - 2K_1 \cdot K_2)\left(K_2^2 \left(K_1 \cdot a\right)^2 + K_1^2 \left(K_2 \cdot a\right)^2 - 2K_1 \cdot a K_2 \cdot a K_1 \cdot K_2\right)}{K_1^2 K_2^2} \,. \tag{5.89}$$

Using this representation for a, the spinor form for the massless 4 dimensional loop momentum is given by

$$\begin{aligned} l_{2,4D}^\mu =& \left(\left\langle \tilde{K}_1 \right| + \frac{\beta + \mu^2 \frac{K_1^2 K_2 \cdot a - \gamma K_1 \cdot a}{C(c_1,c_2)}}{c_1 + \mu^2 \frac{\gamma n_2 \cdot a}{C(c_1,c_2)}} \left\langle \tilde{K}_2 \right|\right) \gamma^\mu \\ & \left(\left(\alpha + \mu^2 \frac{K_2^2 K_1 \cdot a - \gamma K_2 \cdot a}{C(c_1,c_2)}\right) \left|\tilde{K}_1\right] + \left(c_1 + \mu^2 \frac{\gamma n_2 \cdot a}{C(c_1,c_2)}\right) \left|\tilde{K}_2\right]\right) \\ =& \left(\left(\alpha + \mu^2 \frac{K_2^2 K_1 \cdot a - \gamma K_2 \cdot a}{C(c_1,c_2)}\right) \left\langle \tilde{K}_1 \right| + \left(c_2 + \mu^2 \frac{\gamma n_1 \cdot a}{C(c_1,c_2)}\right) \left\langle \tilde{K}_2 \right|\right) \\ & \gamma^\mu \left(\left|\tilde{K}_1\right] + \frac{\beta + \mu^2 \frac{K_1^2 K_2 \cdot a - \gamma K_1 \cdot a}{C(c_1,c_2)}}{c_2 + \mu^2 \frac{\gamma n_1 \cdot a}{C(c_1,c_2)}} \left|\tilde{K}_2\right]\right) \,, \end{aligned} \tag{5.90}$$

where the common function $C(c_1, c_2)$ is extracted to simplify the representation and is given by

$$
\begin{aligned}
C(c_1,c_2) &= 2(K_2^2 K_1\cdot a(K_1^2+K_1\cdot K_2) - K_1^2 K_2\cdot a(K_2^2+K_1\cdot K_2) + 2\Delta(c_1 n_1\cdot a + c_2 n_2\cdot a)) \\
&= 4c_1\Delta n_1\cdot a + 2(K_2^2 K_1\cdot a(K_1^2+K_1\cdot K_2) - K_1^2 K_2\cdot a(K_2^2+K_1\cdot K_2)) \\
&+ \frac{\left(K_2^2 K_1\cdot a^2 + K_1^2 K_2\cdot a^2 - 2K_1\cdot a K_2\cdot a K_1\cdot K_2\right)\left(\mu^2 + \frac{K_1^2K_2^2(K_1^2+K_2^2+2K_1\cdot K_2)}{4\Delta}\right)}{c_1 n_1\cdot a} \\
&= 4c_2\Delta n_2\cdot a + 2(K_2^2 K_1\cdot a(K_1^2+K_1\cdot K_2) - K_1^2 K_2\cdot a(K_2^2+K_1\cdot K_2)) \\
&+ \frac{\left(K_2^2 K_1\cdot a^2 + K_1^2 K_2\cdot a^2 - 2K_1\cdot a K_2\cdot a K_1\cdot K_2\right)\left(\mu^2 + \frac{K_1^2K_2^2(K_1^2+K_2^2+2K_1\cdot K_2)}{4\Delta}\right)}{c_2 n_2\cdot a}\ .
\end{aligned}
\tag{5.91}
$$

As in 4 dimensions the bubble can not directly use the above form as it only has one independent external momenta so an extra arbitrary vector is required to parametrise the set of momenta. By the same method as above, the loop momenta cut conditions can be solved and the loop momentum given by

$$
\begin{aligned}
l_{1,4D,M} &= (1-y)\tilde{K}_1 + \frac{yK_1^2}{2K_1\cdot\chi}\chi - tn_1 - \frac{y(1-y)K_1^2-\mu^2}{2tK_1\cdot\chi}n_2 \\
l_{2,4D,M} &= -y\tilde{K}_1 - \frac{(1-y)K_1^2}{2K_1\cdot\chi}\chi - tn_1 - \frac{y(1-y)K_1^2-\mu^2}{2tK_1\cdot\chi}n_2\ ,
\end{aligned}
\tag{5.92}
$$

where again this solution gives a massive vector. As for the loop momentum parametrisation for pentagons, boxes and triangles, the reduced 4 dimensional loop momentum will have a spinor representation which will factorise. As in 4 dimensions no extra variants of the loop momentum expression are needed to handle the case where $K_1^2 \to 0$ as this will not contribute due to the presence of K_1^2 in the scalar integral's value.

It is necessary to know exactly what form the combinations of tree amplitudes will have in terms of the remaining degrees of freedom and which of these coefficients are needed to form the rational terms for each cut in order to evaluate the coefficients. This is calculated by taking the form of the loop momentum solution, which for three or more cuts is given above, and substituting it into the form of the integral numerators as given in Equations 5.9 to 5.12. The directions for the vectors n_i can be chosen, subject to the conditions that defined them, for each box, triangle and bubble diagram, so that the resulting forms are as simple as possible, without changing the form that results because of the symmetry of the numerator forms. The result will be a power series in the remaining degrees of freedom, with a relatively complex shape in the space of the different degrees of freedom.

The reduced momenta forms given above include extra poles, but these must cancel when the different amplitudes and coefficients are combined to produce the full generalised unitarity contribution. Care must be taken if the vectors become 4 dimensional, as then $\mu^2 \to 0$ and the loop momenta should collapse back to the standard 4 dimensional form, but may numerically diverge from this solution. It is also important to ensure that the product of the loop momenta with a does not vanish, as then the projection would fail. This should be unlikely to happen as long as the products of external momenta with a do not vanish and complex values for the extra components are used. If the product does vanish then there is likely to be a factorisation of the badly behaved components that allows them to cancel between the coefficients. No extra cases will be needed to handle the situation where the momenta of one or both of the external corners used to construct the momenta representation is massless, unlike in 4 dimensions, as long as μ^2 is not zero.

The final component needed to calculate the rational terms, are the subtraction terms resulting from contributions that have extra cuts. These again will be functions of the coefficients for the higher terms, though there are now significantly more terms to handle. It is hoped that some of the coefficients can be ignored if it can be shown that both their contribution to the terms that are actually needed and their contribution to the subtractions for lower diagrams will always vanish when the relevant terms are projected out. As for the pure 4 dimensional calculation, each coefficient can be projected out using complex Fourier projections. Combining all these steps, the coefficients for each amplitude and any subtraction amplitude can be calculated and then combined using Equation 5.17 and Equation 5.6, to give the rational terms in the four dimensional helicity scheme. The full details of how to implement this calculation have not been worked out in this project due to time constraints.

Chapter 6

Implementing the Calculations

Throughout this project Mathematica 8[19] has been used to implement the calculations and derive the formulas given in this thesis. For the 4 dimensional calculations in Chapters 3 and 4, the Mathematica package Spinors@Mathematica (S@M)[20] was used, which provides an implementation of the spinor helicity formalism in 4 dimensions for both analytical and numeric calculations. This Mathematica package uses the same definition for spinors as is used in BlackHat. This enables direct numerical comparisons of the values produced by BlackHat and those produced by the Mathematica implementation, as no overall external momentum dependent phases need to be combined with the values before comparison. The automatic numerical calculations for tree amplitudes and the cut coefficients are implemented in terms of the basic spinor and 4-vector objects provided by this package, as a collection of Mathematica source files. This implementation can calculate the tree amplitudes and coefficients of the one loop scalar basis integrals for any amplitude containing one or zero Higgs bosons with any number of quark pairs and gluons. The Mathematica code for the 4 dimensional calculations is divided into three files: common shared code in Common.txt; the tree level, colour ordered, helicity amplitudes in HelAmplN.txt and the one loop cut part calculations in Loop-Cuts.txt. Types of particles are labelled by objects of the form `Type[Properties,...]`, where `Type` is the type of particle, either `Higgs`, `Phi`, `Gluon` or `Quark`, and `Properties,...` includes the helicity of the particle and, for quarks, its direction (`1` for a quark and `-1` for an anti-quark which is the same as a quark travelling in the opposite direction) and flavour. For example, a positive helicity gluon is labelled as `Gluon[1]`, a negative helicity quark of flavour `2` is labelled as `Quark[1,-1,2]` and a negative helicity anti quark of flavour `3` is labelled as `Quark[-1,-1,3]`. Various properties of the types of particles that are used later in the calculations are defined in Common.txt. This file also loads S@M, defines a few extra methods for it and improves the functions that calculate spinors from momenta to handle cases where elements are infinite or indeterminate (of the form 0/0).

The tree level amplitudes are calculated using the function `HelAmplN`, which is called using expressions of the form `HelAmplN[{Momenta,...},{ParticleTypes,...}]`, where the `Momenta` are either lists containing the elements of the momenta, or names that have been defined in S@M as four vectors or spinors and have their momenta defined. This function's arguments are separated based on whether or not a particle is colour ordered and then the particles are sorted into a canonical order so that when declaring amplitudes, the minimum number of combinations needs to be checked for. In HelAmplN.txt, explicit formulas for the MHV and anti-MHV amplitudes are declared, along with some amplitudes that vanish. These amplitudes are used by the BCFW implementation to build up higher multiplicity amplitudes. It was discovered that some theoretically valid shifts had terms of the form 0/0 which caused numerical problems and were not handled well in this implementation, therefore multiple pairs of shift particles are tried until one is found that provides a valid amplitude. Many cases where this could happen were handled by explicitly checking for certain forms of terms and declaring that they would vanish, even though often they are badly behaved numerically. Care was needed, while combining the two amplitudes to form a BCFW term, to ensure that the correct factors were introduced by the contribution for the particles either side of the cut, so that they form a valid propagator, given that they were calculated with momenta in the opposite directions which are not guaranteed to have a simple relationship. A simpler method was used in the cut part calculations, where the spinors for the reverse particle are explicitly defined as i times the non-reversed particle's spinors, as this does not require as much care on these factors.

The coefficients of the scalar loop integrals were also calculated in Mathematica using the code in Loop-Cuts.txt. For this section of calculations the processes and set of momenta must be declared using the functions `DeclareProcess` and `DeclareMomConf`. Both of these functions take as their first argument a list containing a pair of lists, where the first list is the colour ordered particles and the second is the non-colour ordered particles. `DeclareProcess` also has a second argument which is the quarks that turn left after entering the loop. For each quark pair either the quark or the anti-quark is left turning. It is also possible to have a closed quark loop which is represented by the quark flavour -1. It is not important whether the closed loop quark is a quark or an anti-quark, as the differences always cancel in closed quark loops. The helicity property of left turning quarks is also not specified as it is implicit from the helicity of the external quark. Both these declaration functions return a new unique symbol that labels the process or momentum configuration.

The Mathematica code generates the set of all possible non-vanishing cuts and the sets of cut propagators for each cut. Cuts are described by a pair of lists, the first of which lists particles which are the first particle not in the previous corner, which is the same as the first particle in that corner unless the corner is empty. The second list shows which corner contains each non-colour ordered particle. For example, `{{2,4,4,5},{2}}` in an amplitude with five ordered particles and one unordered particle, means the first corner has particles 2 and 3, the second corner contains the first and only unordered particle, the third corner contains just particle 4 and the last corner contains particles 5 and 1. This split is shown in Figure 6.1. The list of propagator particles is given by the particles heading out of each corner, towards the next corner, in corner number order. This labelling is also shown in Figure 6.1. The list of possible cuts is generated using `SplitOptions[process,n]`, where `process` is the symbol returned by `DeclareProcess` and `n` is the number of cuts wanted and is 2 for bubbles, 3 for triangles and 4 for boxes. The list of possible sets of propagators for a given cut is calculated using `SplitValidPropOptions[process,split]`, where `split` is a split as returned by `SplitOptions`. It is possible for a split to be returned by `SplitOptions`, but for for it to have no valid sets of propagators and therefore for `SplitValidPropOptions` to return an empty list.

The numeric value of the coefficient of each scalar function is calculated using the functions `CalculateBoxContribution`, `CalculateTriContribution` and `CalculateBubbleContribution`, all of which take the process, momentum configuration, cut and cut propagators as arguments, in that order.

For tree level amplitudes the values were tested against the values produced by BlackHat and found to agree to within the expected error, which is of the order of the square root of the error in the input, due to square roots in the spinor definitions. This

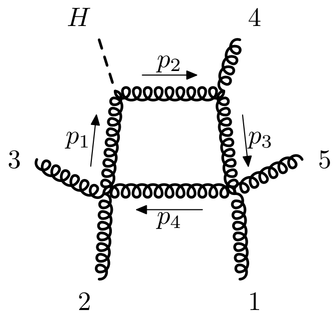


Figure 6.1: The cut labelled {{2,4,4,5},{2}} for a five gluon, one Higgs amplitude. The propagators are labelled by p_i, where i is the index in the list of propagators for the cut. The direction that the propagator particle is viewed as travelling in is shown with arrows. All external particles are taken as outbound. This shape of diagram and cut label is equally valid for any set of five ordered and one non-ordered external particles.

is performed with the python script TreeTests.py which links to BlackHat, generates momenta and processes and calculates them, then generates Mathematica code to calculate each amplitude and compare it to the value calculated with BlackHat. The cut part integral coefficients were also tested to ensure that both implementations produced the same set of non vanishing amplitudes and the same numeric values for all cuts for all types of amplitude and again the values were found to agree. This is tested using the python script OneLoopTests.py. The only issue was with amplitudes with a closed quark loop which were out by a factor of -1, which is a factor that, in BlackHat, has been absorbed into the coefficients rather than the integral. There were also a few cuts that one version produced but the other did not produce, but these always had a value of zero as is required. This inconsistency is not necessarily an issue as time spent calculating coefficients that can be shown to be zero is traded against the effort needed to work out which terms are going to vanish. The main class of amplitudes that vanish but are still produced by BlackHat, was cuts for Higgs boson amplitudes where the corner containing the Higgs boson had helicities that vanish. It would improve the efficiency of BlackHat's calculation to not calculate these contributions that are known to always vanish. The methods of implementation for the cut part calculations for amplitudes with a Higgs boson and for pure quark and gluon amplitudes were compared in BlackHat and showed the same differences as were found in this project. This is another confirmation that the existing implementation in BlackHat is correct for calculating the cut parts of one loop amplitudes with a Higgs boson.

Care was required when comparing cuts between BlackHat and the Mathematica implementation as different labelling conventions are used. If the amplitude contains only colour ordered particles, then there is only a difference if the first particle in the first corner is not particle 1, in which case the last element from the label is moved to the front to form the BlackHat label. If there are non-colour ordered particles then it is even more complex to convert the split labels, as BlackHat inserts the non-colour ordered particles into the colour ordered particles in different positions for each cut, so it can label the cut as if all the particles were ordered. This conversion along with the many other conversions needed to convert between BlackHat labels and Mathematica labels are in BHMathLink.py.

There is also Mathematica code in DrawCuts.txt to draw graphs showing the possible cuts for an amplitude along with which diagrams contribute poles to which other diagrams. This was used to produce Figures 4.3–4.5. These diagrams are produced using pdfLaTeX[21], feynmp[22], pdfcrop[23] and dot[24] to draw the individual Feynman diagrams, process them and then combine them to produce the overall figure. Similar functionality is also available in BlackHat and can be accessed using the `--plotgraph` option of OneLoopTests.py or its short version.

For the rational terms, a Mathematica implementation of spinors in 6 dimensions has been developed. It has been used to derive and check the relations between 6 dimensional and 4 dimensional spinors and between 6 dimensional momenta and spinor expressions. It is implemented in 6DSpinorHelicity.txt and uses the spinors as defined in Section 5.1. Both algebraic and numeric calculations can be performed. There are many rearrangements and simplifications available. Spinors and conjugate spinors are represented by the objects `spinor[name,helicity,...]` and `spinorbar[name,helicity,...]`, where `name` is the label for the particle and `helicity,...` represents the one or two helicity signs for the spinor depending on if the spinor is 4 or 6 dimensional respectively. Slashed matrices are represented by `Sm[name,helicity]` for massless momenta and `SmM[name,helicity]` for massive momenta, where `helicity` is the helicity of the spinor space that the slashed matrix is in. These are combined in products using the function `Sp[elements,...]` where each element is a spinor, conjugate spinor or slashed matrix. It is also possible to insert linear functions of these elements as elements in larger spinor products. All products are expanded as fully as possible and completely closed spinor products are extracted and separated into individual factors. Closed spinor products will also reverse direction to produce a canonical form. As was shown in Section 5.1 many spinor products vanish

```
Sp[spinorbar[p, 1, 1], Sm[r], Sm[s], Sm[t], spinor[q, -1, -1]]
Sp[spinorbar[p, 1, 1], Sm[r], Sm[s], Sm[t], spinor[q, 1, -1]]
Sp[spinorbar[q, 1, -1], Sm[t], Sm[s], Sm[r], spinor[p, 1, 1]]
% == %%
%% // TraditionalForm
Sp[spinorbar[q, 1, -1] + spinorbar[p, 1, -1],
  (Sp[Sm[r], Sm[s]] + Sp[Sm[s], Sm[r]]), spinor[p, -1, -1]]
0
Sp[u̲_{1,1}[p], Sm[r, 1], Sm[s, -1], Sm[t, 1], u_{1,-1}[q]]
Sp[u̲_{1,1}[p], Sm[r, 1], Sm[s, -1], Sm[t, 1], u_{1,-1}[q]]
True
```

$\langle \underline{u}_{1,1}(p),\ \mathrm{Sm}(r,1),\ \mathrm{Sm}(s,-1),\ \mathrm{Sm}(t,1),\ u_{1,-1}(q)\rangle$

```
Sp[u̲_{1,-1}[p], Sm[r, 1], Sm[s, -1], u_{-1,-1}[p]] + Sp[u̲_{1,-1}[p], Sm[s, 1], Sm[r, -1], u_{-1,-1}[p]] +
 Sp[u̲_{1,-1}[q], Sm[r, 1], Sm[s, -1], u_{-1,-1}[p]] + Sp[u̲_{1,-1}[q], Sm[s, 1], Sm[r, -1], u_{-1,-1}[p]]
2 Mp[Mom[s], Mom[r]] Sp[u̲_{1,-1}[q], u_{-1,-1}[p]]
DeclareVectorDimension[r, 4]
DeclareVectorDimension[s, 4]
Sp[spinorbar[p, 1, 1], spinor[q, -1, 1]]
Sp[spinorbar[p, 1, 1], spinor[q, -1, -1]]
Sp[spinorbar[r, 1, 1], spinor[s, -1, 1]]
Sp[spinorbar[r, 1, 1], spinor[s, -1, -1]]
4
4
Sp[u̲_{1,1}[p], u_{-1,1}[q]]
Sp[u̲_{1,1}[p], u_{-1,-1}[q]]
Sp[u̲_{1,1}[r], u_{-1,1}[s]]
0
```

Figure 6.2: Examples of the spinor and slashed matrix objects and their products using the implementation in 6DSpinorHelicity.txt. The first block of equations show various of the simplifications that are performed automatically by the implementation for any momenta while the second block shows simplifications that only happen if the momenta are 4 dimensional.

and these conditions are implemented here and cause the relevant terms to simplify to zero. Using the function `DeclareVectorDimension[momenta, dimensions]`, it is possible to declare that vectors are in a lower number of dimensions, for example, a 4 dimensional vector while working in 6 dimensions, where `momenta` is the name of the momenta and `dimensions` is the number of dimensions that the vector is in. Declaring that vectors are 4 dimensional while working with 6 dimensional spinors and momenta allows the extra properties given in Section 5.1 to be used and many more products to be simplified away. These spinor and slashed matrix objects and some of the automatic simplifications performed are shown in Figure 6.2.

Momenta are represented by `Mom[name]` for massless momenta and `MomM[name]` for massive momenta, where `name` is the name of the vector. These are combined to give their Minkowski product using the function `Mp[mom1,mom2]`, where `mom1` and `mom2` are both momenta and examples are shown in Figure 6.3. Again, as for spinor products,

```
Mp[Mom[p], Mom[q]]
% // TraditionalForm
Mp[Mom[q], Mom[p]]
% == %%
Mp[Mom[p], Mom[p]]
Mp[MomM[p], MomM[p]]
% // TraditionalForm
```

```
Mp[Mom[q], Mom[p]]
```

$p_q \cdot p_p$

```
Mp[Mom[q], Mom[p]]

True

0

Mp[MomM[p], MomM[p]]
```

$P_p \cdot P_p$

Figure 6.3: Examples of momenta objects and their Minkowski products using the implementation in 6DSpinorHelicity.txt.

```
Mp[Sp[spinorbar[p, 1, 1], γ, Sm[q], spinor[r, -1, -1]], Mom[s]]
% // TraditionalForm
% // DoMpToSm // TraditionalForm
```

$\text{Mp}[\text{Sp}[\underline{\text{u}}_{1,1}[\text{p}], \sigma[1], \text{Sm}[\text{q}, -1], \text{u}_{-1,-1}[\text{r}]], \text{Mom}[\text{s}]]$

$\langle \underline{u}_{1,1}(p), \sigma^{\mu}, \text{Sm}(q, -1), u_{-1,-1}(r) \rangle \cdot p_{s\ \mu}$

$\langle \underline{u}_{1,1}(p), \text{Sm}(s, 1), \text{Sm}(q, -1), u_{-1,-1}(r) \rangle$

```
Mp[Sp[spinorbar[p, 1, 1], γ, Sm[q], spinor[r, -1, -1]],
 Sp[spinorbar[s, 1, -1], γ, spinor[t, 1, 1]]]
% // TraditionalForm
% // DoMpToSpinorChains // Simplify // TraditionalForm
```

$\text{Mp}[\text{Sp}[\underline{\text{u}}_{1,1}[\text{p}], \sigma[1], \text{Sm}[\text{q}, -1], \text{u}_{-1,-1}[\text{r}]], \text{Sp}[\underline{\text{u}}_{1,1}[\text{t}], \sigma[1], \text{u}_{1,-1}[\text{s}]]]$

$\langle \underline{u}_{1,1}(p), \sigma^{\mu}, \text{Sm}(q, -1), u_{-1,-1}(r) \rangle \cdot \langle \underline{u}_{1,1}(t), \sigma_{\mu}, u_{1,-1}(s) \rangle$

$$-2 \langle \underline{u}_{1,1}(t), u_{-1,-1}(r) \rangle \langle \underline{u}_{1,1}(p), \text{Sm}(q, 1), u_{1,-1}(s) \rangle +$$
$$2 \langle \underline{u}_{1,-1}(s), u_{-1,-1}(r) \rangle \langle \underline{u}_{1,1}(t), \text{Sm}(q, 1), u_{1,1}(p) \rangle +$$
$$2 \langle \underline{u}_{1,1}(p), u_{-1,-1}(r) \rangle \langle \underline{u}_{1,1}(t), \text{Sm}(q, 1), u_{1,-1}(s) \rangle$$

Figure 6.4: Examples of spinor chains containing gamma matrices and the simplification of the products with momenta and each other using the implementation in 6DSpinorHelicity.txt.

products are fully expanded and if needed, reversed, to give a canonical representation. This canonicalisation is very important as otherwise it is possible to have a complex expression that should cancel, but does not, as the different terms are using different forms of their spinor chains or Minkowski products. It is not easy to find these manually in large expressions. Unfortunately there are still equivalent expressions that cannot be automatically canonicalised, as they would involve commuting matrices around in spinor chains, which if it does not cause a cancellation could drastically increase the number of terms in an expression, causing it to be too complex to handle and work with.

Complex, momenta like, expressions can also be written involving spinor chains with gamma matrices, `\[Gamma]` or `\[Sigma][h]`, inserted, which can be combined using `Mp`; these and simplification of their products are shown in Figure 6.4. If the other vector in the Minkowski product is a normal momenta rather than a spinor chain, then the product can be simplified by applying `DoMpToSm` to it. Simplifications for the Minkowski product of two spinor chains containing gamma matrices are harder to perform in general, but for most cases can be performed by applying `DoMpToSpinorChains`. The main case this function cannot replace is expressions of the form $\underline{u}_{\pm,h}(p)\gamma^{\mu}u_{\pm,l}(q)\ \underline{u}_{\pm,m}(r)\gamma_{\mu}u_{\pm,n}(s)$, as it is not possible to write them purely in terms of the spinors they contain. If this type of term is encountered, the simplest solution is to insert the fraction $\frac{\not{a}\not{b}+\not{b}\not{a}}{2a\cdot b}$ into one of the spinor chains, where a and b are two vectors which should be chosen so that the expression simplifies. As this requires knowledge of what the other vectors are, it cannot be automated and as such is not provided as a function.

Expressions involving chains of slashed matrices can often be simplified by commuting matrices so that objects for the same momenta are neighbouring, or so that different terms are rearranged to have the same form. The function `CommuteMatricies[element1,element2]`, where `element1` and `element2` are both either explicit gamma matrix objects of the form `\[Sigma][h]` or the name of a momenta, can be applied to an expression to commute `element1` and `element2` whenever they occur in that order in the expression. Basic automatic commutations are also possible where it is simple to see that the commutation will produce simpler expressions, as it will cause there to be two neighbouring slashed matrices for the same momenta, or cause a momenta to be next to its corresponding spinor, and as such these terms will vanish. This is done using the functions `CommuteMatriciesAway` and `CommuteSigmaMatriciesAway`, depending on whether it is a gamma matrix or only slashed matrices that are in between the matching items that will be commuted away. Examples of these functions for commuting slashed matrices are shown in Figure 6.5.

These spinor and Minkowski products can also be evaluated numerically using functions defined in 6DSpinorHelicity.txt. There can be multiple sets of values defined for any momenta as the definitions are attached to a `tag`. To define a value for a momentum for a specific `tag` the function `DeclareVectorMomentum` is used, which takes arguments of the form `DeclareVectorMomentum[tag,momentum,{p0,p1,...}]` or `DeclareVectorMomentum[tag,momentum,{{helicities}->spinor,...}]`, where `helicities` is the helicities of the spinor and all possible helicities must be included in

```
Sp[spinorbar[p, 1, 1], Sm[r], Sm[s], Sm[t], spinor[q, 1, -1]] // CommuteMatricies[r, s]
Sp[spinorbar[p, 1, 1], Sm[r], Sm[q], Sm[r], spinor[q, 1, -1]] // CommuteMatriciesAway
Sp[spinorbar[q, 1, -1] + spinorbar[p, 1, -1],
 (Sp[Sm[r], Sm[s]] + Sp[Sm[s], Sm[r]]), spinor[p, -1, -1]]
% // CommuteMatricies[r, s]
```

$$2\,\mathrm{Mp}[\mathrm{Mom}[s],\,\mathrm{Mom}[r]]\,\mathrm{Sp}\left[\underline{u}_{1,1}[p],\,\mathrm{Sm}[t,1],\,u_{1,-1}[q]\right]-\mathrm{Sp}\left[\underline{u}_{1,1}[p],\,\mathrm{Sm}[s,1],\,\mathrm{Sm}[r,-1],\,\mathrm{Sm}[t,1],\,u_{1,-1}[q]\right]$$

$$2\,\mathrm{Mp}[\mathrm{Mom}[r],\,\mathrm{Mom}[q]]\,\mathrm{Sp}\left[\underline{u}_{1,1}[p],\,\mathrm{Sm}[r,1],\,u_{1,-1}[q]\right]$$

$$\mathrm{Sp}\left[\underline{u}_{1,-1}[p],\,\mathrm{Sm}[r,1],\,\mathrm{Sm}[s,-1],\,u_{-1,-1}[p]\right]+\mathrm{Sp}\left[\underline{u}_{1,-1}[p],\,\mathrm{Sm}[s,1],\,\mathrm{Sm}[r,-1],\,u_{-1,-1}[p]\right]+\mathrm{Sp}\left[\underline{u}_{1,-1}[q],\,\mathrm{Sm}[r,1],\,\mathrm{Sm}[s,-1],\,u_{-1,-1}[p]\right]+\mathrm{Sp}\left[\underline{u}_{1,-1}[q],\,\mathrm{Sm}[s,1],\,\mathrm{Sm}[r,-1],\,u_{-1,-1}[p]\right]$$

$$2\,\mathrm{Mp}[\mathrm{Mom}[s],\,\mathrm{Mom}[r]]\,\mathrm{Sp}\left[\underline{u}_{1,-1}[q],\,u_{-1,-1}[p]\right]$$

Figure 6.5: Examples of commuting spinor chains using the implementation in 6DSpinorHelicity.txt.

the list, otherwise missing options will not be usable for evaluating expressions. Once momenta are declared, then expressions using them can be evaluated by applying `Ev[dimensions,tag]`, where `dimensions` is the number of dimensions the expression should be evaluated for, which is required as some expressions can be used algebraically in any number of dimensions, but numerical evaluation requires the number of dimensions to be stated. Numerically evaluating various expressions along with the different ways of declaring momenta values are shown in Figure 6.6.

The 4 dimensional spinors used in 6DSpinorHelicity.txt as part of the recursive definition unfortunately do not have the same form as those used in BlackHat or S@M, but differ by a rotation of vectors in space. The rotation swaps the 2nd and 4th components and negates the 3rd component and is given by the Lorentz transformation

$$\Lambda^{\mu}{}_{\nu} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} . \tag{6.1}$$

This is especially important for the reduction of 6 dimensional momenta to massless 4 dimensional momenta, as the vector a used in the reduction was chosen for that particular spinor representation and needs to be converted if a different representation is used. For the BlackHat and S@M definition of spinors it is clear that a, as given

```
DeclareVectorMomentum[mom1, p, {√474, 3, 12, -4, -4, 17}];
DeclareVectorMomentum[mom1, q, {-√583, 11, -13, 2, 15, 8}];
DeclareVectorMomentum[mom2, p, {1 - 5 i, 3 + 2 i, 11 - 3 i, -7 i, 2 + 11 i, 5}];
DeclareVectorMomentum[mom2, q, {{1, 1} → {{4 - 3 i}, {-3 - 18 i}, {16 - 2 i}, {0}},
   {1, -1} → {{0}, {6/5 + 2 i/5}, {i}, {1}}, {-1, 1} → {{-6/5 - 2 i/5}, {0}, {1}, {42/25 - 81 i/25}},
   {-1, -1} → {{3 + 4 i}, {4 - 3 i}, {0}, {-16 + 2 i}}}];
DeclareVectorMomentum[mom2, r, {5, 0, 3, 0, 0, 0, 0, 4}];
```

```
Mom[p] // Ev[6, mom2]
Mom[q] // Ev[6, mom2]
% == %%
```

$\{1-5\,i,\ 3+2\,i,\ 11-3\,i,\ -7\,i,\ 2+11\,i,\ 5\}$

$\{1-5\,i,\ 3+2\,i,\ 11-3\,i,\ -7\,i,\ 2+11\,i,\ 5\}$

True

```
{Sp[spinorbar[p, 1, 1], spinor[r, -1, 1]],
   Sp[spinorbar[p, 1, -1], spinor[r, -1, 1]]} // Ev[6, mom2] // Simplify
{Sp[spinorbar[q, 1, 1], spinor[r, -1, 1]], Sp[spinorbar[q, 1, -1], spinor[r, -1, 1]]} //
  Ev[6, mom2] // Simplify
% == %%
%%%[[1]] %%%[[2]] == %%[[1]] %%[[2]]
```

$$\left\{\left(-\frac{42}{5}+\frac{66\,i}{5}\right)\sqrt{\frac{4}{5}-\frac{3\,i}{5}},\ (6+2\,i)\sqrt{\frac{4}{5}-\frac{3\,i}{5}}\right\}$$

$$\left\{\frac{6+78\,i}{\sqrt{5}},\ \frac{6+2\,i}{\sqrt{5}}\right\}$$

False

True

```
Sm[p, h] // Ev[6, mom1] // TraditionalForm
Sm[p, h] // Ev[6, mom2] // TraditionalForm
```

$$\begin{pmatrix} \sqrt{474}-3\,h & 4-12\,i & -17+4\,i & 0 \\ -4-12\,i & -3\,h-\sqrt{474} & 0 & -17+4\,i \\ 17+4\,i & 0 & -3\,h-\sqrt{474} & -4+12\,i \\ 0 & 17+4\,i & 4+12\,i & \sqrt{474}-3\,h \end{pmatrix}$$

$$\begin{pmatrix} (1-5\,i)-(3+2\,i)\,h & -3-4\,i & 6-2\,i & 0 \\ -3-18\,i & (-3-2\,i)\,h-(1-5\,i) & 0 & 6-2\,i \\ 16-2\,i & 0 & (-3-2\,i)\,h-(1-5\,i) & 3+4\,i \\ 0 & 16-2\,i & 3+18\,i & (1-5\,i)-(3+2\,i)\,h \end{pmatrix}$$

Figure 6.6: Examples of declaring numeric values for momenta and evaluating spinors, momenta and slashed matrix objects in terms of them using the implementation in 6DSpinorHelicity.txt.

in Equation 5.48, will be given by

$$a^{\mu}_{\mathrm{BH}} = \frac{1}{2}\begin{pmatrix} 1 \\ 0 \\ 0 \\ -1 \end{pmatrix} . \tag{6.2}$$

These replacements for 6 dimensional spinors in terms of massless 4 dimensional spinors are provided in the variables `rep` and `reph[h]`, where in the latter the replacement uses `h` and its negative for the summed over helicities needed in the replacement, which can greatly help with producing a simple result. The reverse replacement is also provided in the variable `unrep`. These leave the expression defined in terms of the explicitly named extra component of the momenta p_4 and p_5, which can then be replaced with the μ expressions defined in Equation 5.46. This is achieved by using the replacements stored in the variables `tomu`, `tomuh[h]` and `tomuhp[hfn]`, where the first replacement uses + and `-` as the signs in the μ, the second uses `+h` and `-h` as the signs and the third version's argument is a function that takes the name of the momenta and returns the expression to base the signs for the μs on. Again the reverse replacement is provided and is stored in the variable `frommu`. To avoid replacing all momenta in an expression these replacements will currently only replace an explicitly specified list of vectors; the replacements will need to be manually updated if extra vectors need to be replaced, by changing the pattern named as `pp` in the replacement rules.

Due to time constraints the full calculation for 6 dimensional tree and one loop amplitudes has not been performed, but could be developed using the objects and tools defined in 6DSpinorHelicity.txt along with the Feynman rules defined in Tables 2.1 and 5.1. Unfortunately the conversion to 4 dimensional spinors cannot be performed using replacement rules, as slashed matrices have the same form in any number of dimensions, but when converting from 6 to 4 dimensions should convert to the sum of both helicities, for which it is impossible to declare a rule. There are also issues with simplifications for spinor products applying during the replacement causing incorrect results and terms to vanish that should not have, since while in the middle of applying the replacement some objects are 6 dimensional and some are already 4 dimensional. As such the final conversion to 4 dimensional expressions will have to be done manually while writing the 4 dimensional implementation.

Chapter 7

Conclusions

In this thesis progress has been made towards adding Higgs boson amplitudes to BlackHat. This requires calculating tree amplitudes with any number of quarks and gluons both with and without a Higgs boson. These tree amplitudes have been calculated and are shown to agree with BlackHat, apart from a few differences in sign conventions for quarks. The cut parts have also been calculated and again agree with BlackHat, after taking account of the differences in quark signs, apart from a factor of -1 for closed quark loop contributions. BlackHat was seen to produce cuts that were known to vanish. The main case where this happened was amplitudes containing a Higgs boson where the corner with the Higgs boson causes the coefficient to vanish unless it is restricted to have at least two negative helicities and this is not currently done in BlackHat. Adding this restriction would save calculating a large class of cuts that can easily be seen to vanish. Apart from this possible improvement in speed the code was found to reliably and efficiently calculate the cut parts. The various restrictions and assumptions used in deriving the calculations and therefore the limitations of the method as currently implemented in this project are discussed. The main restriction on the method as discussed in this thesis is that only massless particles are allowed in loops. This restriction could be lifted and will have to be lifted when it is used in calculating the rational terms.

The method to calculate the rational terms was unfortunately not completed in this project due to time constraints so could not be implemented into BlackHat. The basic method is discussed in Chapter 5 and a Mathematica implementation of 6 dimensional spinor helicity formalism has been developed, along with a method to reduce the calculations back down to 4 dimensions by introducing effective massive particles which is discussed in Chapter 6. This implementation is included in Appendix C. The general form of the dependence of tree amplitudes needed for generalised unitarity calculations on the extra dimensions has been investigated and there is no dependence found on the direction of the extra components, apart from in the amplitudes needed to subtract the extra gluon polarisation states. For those amplitudes it is required that the product of the loop momenta with the polarisation vector in the 6^{th} dimension is taken to be zero. If this is ensured, even if the direction of the polarisation vector has to be effectively chosen based on the direction of the loop momenta, then again the amplitudes are independent of the direction of the extra components. With these restrictions the rational terms can be calculated in terms of only 4 dimensional momenta and amplitudes with new effective massive particles, however the internal degrees of freedom in the Feynman rules still require full 6 dimensional momenta and states.

Once a set of the simplest amplitudes has been calculated, higher multiplicity amplitudes can again be calculated with the BCFW recursion relations, but this time using massive 4 dimensional momenta. The general form has been investigated, but the details of the implementation, including which particles are valid to use as the shifted particles and what other factors need to be introduced into each term other than the pair of tree amplitudes, has not been investigated. Once all tree amplitudes can be calculated the scalar loop integral coefficients needed for the rational terms can be extracted using generalised unitarity. The forms of the numerator terms in terms of the degrees of freedom in the loop momenta definitions needs calculating, along with the subtraction terms needed to subtract the higher order terms from the lower order terms, which must be formulated in a way that can be evaluated numerically. Once these elements of the calculation are completed, the rational terms can be calculated. Finally, these calculations will need to be implemented into BlackHat, which will also require converting the calculation to use the conventions used in BlackHat and implementing it in terms of BlackHat's choice of spinor conventions. These combined with the already implemented cut part calculations will provide an efficient and automatic calculation for the loop amplitude in a way that will make BlackHat an even more useful addition to the set of tools available for NLO calculations at the LHC. This thesis provides an important set of building blocks towards this aim.

Bibliography

[1] R. V. Harlander, T. Neumann, K. J. Ozeren, and M. Wiesemann, “Top-mass effects in differential Higgs production through gluon fusion at order α_s^4,” *JHEP* **08** (2012) 139, arXiv:1206.0157 [hep-ph].

[2] M. Grazzini and H. Sargsyan, “Heavy-quark mass effects in Higgs boson production at the LHC,” *JHEP* **09** (2013) 129, arXiv:1306.4581 [hep-ph].

[3] Z. Bern, A. De Freitas, L. J. Dixon, and H. L. Wong, “Supersymmetric regularization, two loop QCD amplitudes and coupling shifts,” *Phys. Rev.* **D66** (2002) 085002, arXiv:hep-ph/0202271 [hep-ph].

[4] S. J. Parke and T. R. Taylor, “Amplitude for n-gluon scattering,” *Phys. Rev. Lett.* **56** (Jun, 1986) 2459–2460. https://link.aps.org/doi/10.1103/PhysRevLett.56.2459.

[5] “Blackhat.” http://blackhat.hepforge.org/.

[6] C. F. Berger, Z. Bern, L. J. Dixon, F. Febres Cordero, D. Forde, H. Ita, D. A. Kosower, and D. Maitre, “An Automated Implementation of On-Shell Methods for One-Loop Amplitudes,” *Phys. Rev.* **D78** (2008) 036003, arXiv:0803.4180 [hep-ph].

[7] Z. Bern, L. J. Dixon, F. Febres Cordero, S. Höche, H. Ita, D. A. Kosower, D. Maître, and K. J. Ozeren, “The BlackHat Library for One-Loop Amplitudes,” *J. Phys. Conf. Ser.* **523** (2014) 012051, arXiv:1310.2808 [hep-ph].

[8] L. J. Dixon, “Calculating scattering amplitudes efficiently,” arXiv:hep-ph/9601359 [hep-ph].

[9] L. J. Dixon, E. N. Glover, and V. V. Khoze, "MHV rules for Higgs plus multi-gluon amplitudes," *JHEP* **0412** (2004) 015, arXiv:hep-th/0411092 [hep-th].

[10] R. Britto, F. Cachazo, B. Feng, and E. Witten, "Direct proof of tree-level recursion relation in Yang-Mills theory," *Phys.Rev.Lett.* **94** (2005) 181602, arXiv:hep-th/0501052 [hep-th].

[11] C. F. Berger and D. Forde, "Multi-Parton Scattering Amplitudes via On-Shell Methods," *Ann. Rev. Nucl. Part. Sci.* **60** (2010) 181–205, arXiv:0912.3534 [hep-ph].

[12] D. Forde, "Direct extraction of one-loop integral coefficients," *Phys.Rev.* **D75** (2007) 125019, arXiv:0704.1835 [hep-ph].

[13] H. Ita, "Susy Theories and QCD: Numerical Approaches," *J.Phys.* **A44** (2011) 454005, arXiv:1109.6527 [hep-th].

[14] W. T. Giele, Z. Kunszt, and K. Melnikov, "Full one-loop amplitudes from tree amplitudes," *JHEP* **04** (2008) 049, arXiv:0801.2237 [hep-ph].

[15] C. F. Berger, V. Del Duca, and L. J. Dixon, "Recursive Construction of Higgs-Plus-Multiparton Loop Amplitudes: The Last of the Phi-nite Loop Amplitudes," *Phys.Rev.* **D74** (2006) 094021, arXiv:hep-ph/0608180 [hep-ph].

[16] S. Davies, "One-Loop QCD and Higgs to Partons Processes Using Six-Dimensional Helicity and Generalized Unitarity," *Phys.Rev.* **D84** (2011) 094016, arXiv:1108.0398 [hep-ph].

[17] J. Wenzler, "Towards an unitarity-based implementation of higgs production in association with jets including quantum corrections," Master's thesis, Albert-Ludwigs-Universität, Freiburg, 2016.

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

[19] "Mathematica 8." https://www.wolfram.com/mathematica/.

[20] D. Maitre and P. Mastrolia, "S@M, a Mathematica Implementation of the Spinor-Helicity Formalism," *Comput.Phys.Commun.* **179** (2008) 501–574, arXiv:0710.5559 [hep-ph].

[21] “pdflatex.” http://www.tug.org/applications/pdftex/.

[22] “feynmp.” https://www.ctan.org/pkg/feynmf.

[23] “pdfcrop.” https://www.ctan.org/pkg/pdfcrop.

[24] “dot.” http://www.graphviz.org/.

Appendix A

The Form of a Generic Term in an Amplitude.

The first case to prove is the generic term in a pure gluon amplitude which is given by

$$C\frac{\prod_{i=1}^{m} P_i \cdot Q_i \prod_{j=1}^{N_g-2n} R_j^{\mu_j} \prod_{k=1}^{n} g^{\nu_k \sigma_k}}{\prod_{l=1}^{N_g+m-n-2} S_l^2} \ . \tag{A.1}$$

This can be proven to be the correct form using proof by induction. Firstly the base cases are a single three or four gluon vertex on its own which corresponds to the cases where $m = 0$, $n = 1$ and $N_g = 3$ or $m = 0$, $n = 2$ and $N_g = 4$ respectively. To prove the recursion this form must be combined with each possible form of propagator and extra vertex combination and show that all terms in the result are also of this form. Firstly the case of combining a momentum index with the metric tensor from a three gluon vertex is considered. The relation is then given by

$$
\begin{aligned}
&C\frac{\prod_{i=1}^{m} P_i\cdot Q_i \prod_{j=1}^{N_g-2n} R_j^{\mu_j} \prod_{k=1}^{n} g^{\nu_k\sigma_k}}{\prod_{l=1}^{N_g+m-n-2} S_l^2}\frac{i}{k^2+i0}\frac{-i}{\sqrt{2}}g_{\mu_1}{}^{\mu_1'}(k_1-k_2)^{\mu_{(N_g-2n+1)}} \\
&=\frac{C}{\sqrt{2}}\frac{\prod_{i=1}^{m} P_i\cdot Q_i\left(\prod_{j=2}^{N_g-2n} R_j^{\mu_j}\right)R_1^{\mu_1'}(k_1-k_2)^{\mu_{(N_g-2n+1)}}\prod_{k=1}^{n} g^{\nu_k\sigma_k}}{k^2\prod_{l=1}^{N_g+m-n-2} S_l^2} \\
&=C'\frac{\prod_{i=1}^{m} P_i\cdot Q_i \prod_{j=1}^{N_g'-2n} R_j'^{\mu_j'} \prod_{k=1}^{n} g^{\nu_k\sigma_k}}{\prod_{l=1}^{N_g'+m-n-2} S_l'^2}\ ,
\end{aligned}
\tag{A.2}
$$

where in the last line the new factors have been relabelled to match the terms they combine with and N_g has been replaced with $N_g' = N_g+1$ which is the correct increase in external gluons. The next case is when a metric tensor in the term is contracted with a metric tensor of the extra three gluon vertex and is given by

$$
\begin{aligned}
&C\frac{\prod_{i=1}^{m} P_i\cdot Q_i \prod_{j=1}^{N_g-2n} R_j^{\mu_j} \prod_{k=1}^{n} g^{\nu_k\sigma_k}}{\prod_{l=1}^{N_g+m-n-2} S_l^2}\frac{i}{k^2+i0}\frac{-i}{\sqrt{2}}g_{\sigma_1}{}^{\sigma_1'}(k_1-k_2)^{\mu_{(N_g-2n+1)}} \\
&=\frac{C}{\sqrt{2}}\frac{\prod_{i=1}^{m} P_i\cdot Q_i\left(\prod_{j=1}^{N_g-2n} R_j^{\mu_j}\right)(k_1-k_2)^{\mu_{(N_g-2n+1)}}\left(\prod_{k=2}^{n} g^{\nu_k\sigma_k}\right)g^{\nu_1\sigma_1'}}{k^2\prod_{l=1}^{N_g+m-n-2} S_l^2} \\
&=C'\frac{\prod_{i=1}^{m} P_i\cdot Q_i \prod_{j=1}^{N_g'-2n} R_j'^{\mu_j'} \prod_{k=1}^{n} g^{\nu_k\sigma_k'}}{\prod_{l=1}^{N_g'+m-n-2} S_l'^2}\ .
\end{aligned}
\tag{A.3}
$$

The next case is contracting the momenta from a three gluon vertex with a metric tensor from the term and is given by

$$
\begin{aligned}
& C\frac{\prod_{i=1}^{m} P_i\cdot Q_i \prod_{j=1}^{N_g-2n} R_j^{\mu_j} \prod_{k=1}^{n} g^{\nu_k\sigma_k}}{\prod_{l=1}^{N_g+m-n-2} S_l^2}\frac{i}{k^2+i0}\frac{-i}{\sqrt{2}}g^{\nu_1'\sigma_1'}(k_1-k_2)_{\sigma_1} \\
&= \frac{C}{\sqrt{2}}\frac{\prod_{i=1}^{m} P_i\cdot Q_i \left(\prod_{j=1}^{N_g-2n} R_j^{\mu_j}\right)(k_1-k_2)^{\nu_1}\left(\prod_{k=2}^{n} g^{\nu_k\sigma_k}\right)g^{\nu_1'\sigma_1'}}{k^2\prod_{l=1}^{N_g+m-n-2} S_l^2} \\
&= C'\frac{\prod_{i=1}^{m} P_i\cdot Q_i \prod_{j=1}^{N_g'-2n} R_j'^{\mu_j'} \prod_{k=1}^{n} g^{\nu_k'\sigma_k'}}{\prod_{l=1}^{N_g'+m-n-2} S_l'^2}\ .
\end{aligned} \tag{A.4}
$$

The last case for adding a three gluon vertex is contracting the momenta from the vertex with a momenta in the term which results in

$$
\begin{aligned}
& C\frac{\prod_{i=1}^{m} P_i\cdot Q_i \prod_{j=1}^{N_g-2n} R_j^{\mu_j} \prod_{k=1}^{n} g^{\nu_k\sigma_k}}{\prod_{l=1}^{N_g+m-n-2} S_l^2}\frac{i}{k^2+i0}\frac{-i}{\sqrt{2}}g^{\nu_{n+1}'\sigma_{n+1}'}(k_1-k_2)_{\mu_1} \\
&= \frac{C}{\sqrt{2}}\frac{\left(\prod_{i=1}^{m} P_i\cdot Q_i\right) R_1\cdot(k_1-k_2)\prod_{j=2}^{N_g-2n} R_j^{\mu_j}\left(\prod_{k=1}^{n} g^{\nu_k\sigma_k}\right)g^{\nu_{n+1}'\sigma_{n+1}'}}{k^2\prod_{l=1}^{N_g+m-n-2} S_l^2} \\
&= C'\frac{\prod_{i=1}^{m'} P_i'\cdot Q_i' \prod_{j=1}^{N_g'-2n'} R_j'^{\mu_j'} \prod_{k=1}^{n'} g^{\nu_k'\sigma_k'}}{\prod_{l=1}^{N_g'+m'-n'-2} S_l'^2}\ ,
\end{aligned} \tag{A.5}
$$

where in the last line as well as replacing N_g with $N_g' = N_g + 1$, n has been replaced with $n' = n + 1$ and m has been replaced with $m' = m + 1$ which all agree with the expected limits on the different factors.

Lastly for the pure gluon amplitude the contraction of a four gluon vertex with a term must be shown to produce correctly formatted terms. Again each case will be handled separately. Firstly, if the vertex is contracted with a metric tensor from the term then the combination is given by

$$
\begin{aligned}
&C\frac{\prod_{i=1}^{m} P_i\cdot Q_i \prod_{j=1}^{N_g-2n} R_j^{\mu_j} \prod_{k=1}^{n} g^{\nu_k\sigma_k}}{\prod_{l=1}^{N_g+m-n-2} S_l^2}\frac{i}{k^2+i0}cg_{\sigma_1}{}^{\sigma'_1}g^{\nu'_{n+1}\sigma'_{n+1}} \\
&= icC\frac{\prod_{i=1}^{m} P_i\cdot Q_i \prod_{j=1}^{N_g-2n} R_j^{\mu_j}\left(\prod_{k=2}^{n} g^{\nu_k\sigma_k}\right) g^{\nu_1\sigma'_1}g^{\nu'_{n+1}\sigma'_{n+1}}}{k^2\prod_{l=1}^{N_g+m-n-2} S_l^2} \\
&= C'\frac{\prod_{i=1}^{m} P_i\cdot Q_i \prod_{j=1}^{N'_g-2n'} R_j{}^{\mu_j} \prod_{k=1}^{n'} g^{\nu'_k\sigma'_k}}{\prod_{l=1}^{N'_g+m-n'-2} S_l'^2}\ ,
\end{aligned}
\tag{A.6}
$$

where now N_g is replaced by $N'_g = N_g + 2$ and n is replaced by $n' = n + 1$. The last case for pure gluon amplitudes is contracting the four gluon vertex with a momenta in the term which gives

$$
\begin{aligned}
&C\frac{\prod_{i=1}^{m} P_i\cdot Q_i \prod_{j=1}^{N_g-2n} R_j^{\mu_j} \prod_{k=1}^{n} g^{\nu_k\sigma_k}}{\prod_{l=1}^{N_g+m-n-2} S_l^2}\frac{i}{k^2+i0}cg_{\mu_1}{}^{\mu'_1}g^{\nu'_{n+1}\sigma'_{n+1}} \\
&= icC\frac{\prod_{i=1}^{m} P_i\cdot Q_i \left(\prod_{j=2}^{N_g-2n} R_j^{\mu_j}\right) R_1^{\mu'_1}\left(\prod_{k=1}^{n} g^{\nu_k\sigma_k}\right) g^{\nu'_{n+1}\sigma'_{n+1}}}{k^2\prod_{l=1}^{N_g+m-n-2} S_l^2} \\
&= C'\frac{\prod_{i=1}^{m} P_i\cdot Q_i \prod_{j=1}^{N'_g-2n'} R_j{}^{\mu_j} \prod_{k=1}^{n'} g^{\nu'_k\sigma'_k}}{\prod_{l=1}^{N'_g+m-n'-2} S_l'^2}\ .
\end{aligned}
\tag{A.7}
$$

The changes in number of factors of each type that can be introduced by adding each type of vertex are summarised in Table A.1.

Now the form for amplitudes with a single quark line will be derived which is given

Terms of the form	Formula	Three Gluon Vertex		Four Gluon Vertex
$P \cdot Q$	m		$+1$	
R^μ	$N_g - 2n$	$+1$	-1	
$g^{\mu\nu}$	n		$+1$	$+1$
$\frac{1}{S^2}$	$N_g + m - n - 2$	$+1$	$+1$	$+1$
Number of gluons	N_g	$+1$	$+1$	$+2$

Table A.1: The different forms of factors that can appear in pure gluon amplitudes along with how the number of each factor present is changed by adding extra propagator and vertex combinations of each type.

by

$$C \frac{\prod_{i=1}^{m} P_i \cdot Q_i \prod_{j=1}^{N_g - N_{qg} - 2n} R_j \cdot \epsilon_j \prod_{k=1}^{n} \epsilon_k \cdot \epsilon'_k}{\prod_{l=1}^{N_g + m - n - N_{qR} - 1} S_l^2} \bar{u} \left(\prod_{a}^{N_{qR} + N_{qg} - 1} \not{U}_a \not{T}_a \right) \not{U}_{(N_{qR} + N_{qg})} u \ , \quad \text{(A.8)}$$

where N_{qg} of the U_as are gluon polarisation vectors and the rest (N_{qR} of them) along with T_a are momentum vectors containing some combination of neighbouring momenta. To derive this formula, the form is built by tracing along the quark line from the spinor towards the conjugate spinor and then at the end adding the conjugate spinor for the other quark to the end of the quark line. The form used to build this up is therefore the equation given above with the conjugate spinor, $\underline{u}$, removed which is given by

$$C \frac{\prod_{i=1}^{m} P_i \cdot Q_i \prod_{j=1}^{N_g - N_{qg} - 2n} R_j \cdot \epsilon_j \prod_{k=1}^{n} \epsilon_k \cdot \epsilon'_k}{\prod_{l=1}^{N_g + m - n - N_{qR} - 1} S_l^2} \left(\prod_{a}^{N_{qR} + N_{qg} - 1} \not{U}_a \not{T}_a \right) \not{U}_{(N_{qR} + N_{qg})} u \ . \quad \text{(A.9)}$$

The base cases for the proof by induction are therefore the quark combined with a single quark vertex which is itself combined with a gluon propagator and then a term for a combination of gluons, which will be of the form found above for the pure gluon amplitude. There are two different cases depending on whether the gamma matrix introduced by the quark vertex is contracted with a momentum from the gluon term or a metric tensor and therefore eventually with a gluon. If the gamma matrix is contracted with a momentum vector then the terms will have the form of

$$
\begin{aligned}
& C\frac{\prod_{i=1}^{m} P_i\cdot Q_i \prod_{j=1}^{(N_g+1)-2n-1} R_j^{\mu_j}\prod_{k=1}^{n} g^{\nu_k\sigma_k}}{\prod_{l=1}^{(N_g+1)+m-n-2} S_l^2}\frac{i}{k^2+i0}\not{R}_{((N_g+1)-2n)}u \\
&= C'\frac{\prod_{i=1}^{m} P_i\cdot Q_i \prod_{j=1}^{N_g-2n} R_j^{\mu_j}\prod_{k=1}^{n} g^{\nu_k\sigma_k}}{k^2\prod_{l=1}^{N_g+m-n-1} S_l^2}\not{R}_{(N_g-2n+1)}u \\
&= C'\frac{\prod_{i=1}^{m} P_i\cdot Q_i \prod_{j=1}^{N_g-2n} R_j^{\mu_j}\prod_{k=1}^{n} g^{\nu_k\sigma_k}}{\prod_{l=1}^{N_g+m-n} {S'_l}^2}\not{V}_1 u\ ,
\end{aligned}
\tag{A.10}
$$

where N_g+1 is used where N_g would have been used in the pure gluon amplitude as one gluon from the factor is replaced with the quark line. This matches the expected form for when $N_{qR}=1$ and $N_{qg}=0$.

$$
\begin{aligned}
& C\frac{\prod_{i=1}^{m} P_i\cdot Q_i \prod_{j=1}^{(N_g+1)-2n} R_j^{\mu_j}\prod_{k=1}^{n-1} g^{\nu_k\sigma_k}}{\prod_{l=1}^{(N_g+1)+m-n-2} S_l^2}\frac{i}{k^2+i0}\gamma^{\sigma_n}u \\
&= C'\frac{\prod_{i=1}^{m} P_i\cdot Q_i \prod_{j=1}^{N_g-2n} R_j^{\mu_j}\prod_{k=1}^{n-1} g^{\nu_k\sigma_k}}{k^2\prod_{l=1}^{N_g+m-n-1} S_l^2}\gamma^{\sigma_n}u \\
&= C'\frac{\prod_{i=1}^{m} P_i\cdot Q_i \prod_{j=1}^{N_g-2n'-2} R_j^{\mu_j}\prod_{k=1}^{n'} g^{\nu_k\sigma_k}}{\prod_{l=1}^{N_g+m-n'-1} {S'_l}^2}\gamma^{\sigma_n}u\ ,
\end{aligned}
\tag{A.11}
$$

where in the last line n has been replaced with $n'=n-1$. Again this matches with the expected form, but now for when $N_{qR}=0$ and $N_{qg}=1$. The recursion relations are derived by adding an extra quark propagator and quark vertex with the quark vertex contracted with a gluon propagator and a pure gluon term. The formulas are too complex to show here but the changes induced by each combination are shown in Table A.2.

Terms of the Form	Formula	Type of Vector that is Contracted into the Quark Line: Polarisation Vector	Type of Vector that is Contracted into the Quark Line: Momenta
$P \cdot Q$	m	$+m'$	$+m'$
R^{μ}	$N_g - 2n - N_{qg}$	$+N'_g - 2n' - 1$	$+N'_g - 2n'$
$g^{\mu\nu}$	n	$+n'$	$+n'$
$\frac{1}{S^2}$	$N_g + m - n + N_{qR} - 1$	$+N'_g + m' - n'$	$+N'_g + m' - n' + 1$
Number of gluons	N_g	$+N'_g$	$+N'_g$
Number of Polarisation Vectors in the Quark Line	N_{qg}	$+1$	
Number of Momenta in the Quark Line	$2N_{qR} + N_{qg}$	$+1$	$+2$

Table A.2: The different forms of factors that can appear in amplitudes with one quark line along with how the number of each factor present is changed by adding extra propagator and vertex combinations of each type.

Appendix B

Source Code for 4 Dimensional Calculations and Comparing to BlackHat

The code created as part of this project can also be downloaded from http://bit.ly/2oXSBSu.

B.1 Mathematica Implementation of Tree and One Loop Cut Part Calculations

Listing B.1: Common.txt

```
(* Functions to handle a list with wrapping. Handles the modular arithmetic with the range of indices being 1 to length rather than the more common 0 to length - 1 *)
LLMod[l_List, i_Integer] := 1 + Mod[i - 1, Length[l]]
LL[l_List, i_Integer] := l[[LLMod[l, i]]]
LL[l_List, i_List] := (LL[l, #1] & ) /@ i
LLRange[l_List, i_Integer, j_Integer] :=
 (LLMod[l, #1] & ) /@ Range[i, i + Mod[j - i, Length[l]]]

(* Load S@M *)
<< "Spinors`"

(* Silence the many messages that S@M prints *)
```

```
Spinors`Private`PRINT[x__] := Null

(* Add an undeclare method *)
Spinors`UndeclareSpinorMomentum[s_] := (Spinors`Private`NumLa[s] =.;
  Spinors`Private`NumCLa[s] =.; Spinors`Private`NumLat[s] =.;
  Spinors`Private`NumCLat[s] =.; Unprotect[Num4V]; Num4V[s] =.;
  Protect[Num4V];
  Spinors`Private`NumSpinorList :=
   Evaluate[DeleteCases[Spinors`Private`NumSpinorList, s]];
  Spinors`Private`NumDefs[])
Spinors`FullyUndeclareSpinor[s_] := (UndeclareSpinorMomentum[s];
   UndeclareSpinor[s]);

(* Declare an improved version of the spinor definition formula that
    handles Indeterminates and Complex Infinities *)
Spinors`Private`Sol2Dim1[k_List] :=
 Module[{sqk1p = Sqrt[k[[1]] + k[[4]]], ptplus = k[[2]] + I k[[3]]},
  If[MemberQ[k, Indeterminate | ComplexInfinity], {Indeterminate,
    Indeterminate},
   If[sqk1p == 0,
    If[k[[1]] - k[[4]] ==
      0, {(k[[2]] - I k[[3]])/Sqrt[2 k[[2]]], (k[[2]] + I k[[3]])/
       Sqrt[2 k[[2]]]}, {(k[[2]] - I k[[3]])/Sqrt[k[[1]] - k[[4]]],
      Sqrt[k[[1]] - k[[4]]]}], {sqk1p, ptplus/sqk1p}]]]
Spinors`Private`Sol2Dim2[k_List] :=
 Module[{factor = 1, sqk1p = Sqrt[k[[1]] + k[[4]]],
   ptminus = k[[2]] - I k[[3]]},
  If[MemberQ[k, Indeterminate | ComplexInfinity], {Indeterminate,
    Indeterminate},
   If[sqk1p == 0,
    If[k[[1]] - k[[4]] ==
      0, {(k[[2]] + I k[[3]])/Sqrt[2 k[[2]]], (k[[2]] - I k[[3]])/
       Sqrt[2 k[[2]]]}, {(k[[2]] + I k[[3]])/Sqrt[k[[1]] - k[[4]]],
      Sqrt[k[[1]] - k[[4]]]}], {sqk1p, ptminus/sqk1p}]]]

DeclareSpinor[Epsilon[___]]

(* Use a different sign convention. Can be set to true after this is
    loaded and before calculation is performed *)
$UseBHSigns=False

(* Set the precision if it is not already set *)
```

```
If[Hold[Evaluate[$Precision]] === Hold[$Precision], $Precision =
  MachinePrecision]

(* Allows increased precision in internal parts of the calculation *)
NN[e_] := N[e, $Precision]
NN[e_, f : (_Integer | _Real)] := N[e, f $Precision]
NN[e_, Max] := N[e, 10 MachinePrecision]

(* Set the symbol s so that it evaluates to the numeric value of the
    expression v at f times the current precision but only actually
    evaluates once per precision at which it is used *)
NNCache[s_Symbol, v_, f_: 1] :=
 s := (Module[{vv = Evaluate[NN[v, f]], p = $Precision},
    s := vv /; p == $Precision]; s)


(* Only declare the spinor momentum if the elements of the momenta or
    spinor are numeric *)
MyDeclareSpinorMomentum[a_,
  xx : {_?NumericQ, _?NumericQ, _?NumericQ, _?NumericQ}] :=
 DeclareSpinorMomentum[a, xx]
MyDeclareSpinorMomentum[
  a_, {la : {{_?NumericQ}, {_?NumericQ}},
   lat : {{_?NumericQ, _?NumericQ}}}] :=
 DeclareSpinorMomentum[a, la, lat]

(* Evaluate an expression with the specified sets of definitions for
    momenta. Will remain unevaluated if the spinor elements are not
    numeric *)
WithSpinors[expr_,
   spinors : {_, ({_?NumericQ, _?NumericQ, _?NumericQ, _?
          NumericQ} | {{{_?NumericQ}, {_?NumericQ}}, {{_?NumericQ, _?
            NumericQ}}})} ...] :=
  Module[{tmp}, MyDeclareSpinorMomentum @@@ {spinors}; tmp = expr;
   FullyUndeclareSpinor[#[[1]]] & /@ {spinors}; tmp];
SetAttributes[WithSpinors, HoldFirst]
SyntaxInformation[WithSpinors] = {"ArgumentsPattern" -> {_, ___},
  "LocalVariables" -> {"Table", {2, \[Infinity]}}}

(* Declare various properties for the different types of particle *)
ParticleName[Gluon[h_]]:="Gluon helicity:"<>ToString[h]
ParticleName[Phi[1]]:="Phi"
```

```
ParticleName[Phi[-1]]:="Phi dagger"
ParticleName[Higgs[]]:="Higgs boson"
ParticleName[Quark[1,h_,f_]]:="Quark of flavour:"<>ToString[h]<>" and helicity:"<>ToString[h]
ParticleName[Quark[-1,h_,f_]]:="Anti Quark of flavour:"<>ToString[h]<>" and helicity:"<>ToString[h]

ReverseParticle[phi_Phi]:=phi
ReverseParticle[higgs_Higgs]:=higgs
ReverseParticle[Gluon[h_]]:=Gluon[-h]
ReverseParticle[Quark[p_,h_,f_]]:=Quark[-p,-h,f]

IsOrderedQ[_Phi|_Higgs]:=False
IsOrderedQ[_Gluon|_Quark]:=True
IsUnorderedQ[p_]:=!IsOrderedQ[p]

IsMassiveQ[_Phi|_Higgs]:=True
IsMassiveQ[_Gluon|_Quark]:=False

(* Skip evaluating if the condition is false *)
SetAttributes[SkipIf, HoldFirst]
SkipIf[val_, False] := val
SkipIf[_, True] := Sequence[]

(* Provide a sort key that uniquely sorts particle types into a format that gives convenient orderings *)
SortKey[Gluon[h_]]:={0,h,0,0,0}
SortKey[Quark[p_,h_,f_]]:={-1,0,0,0,0}
SortKey[Phi[p_]]:={100,-p,0,0,0}
SortKey[Higgs[]]:={100,0,0,0,0}
```

Listing B.2: HelAmplN.txt

```
(* Split the particles into separate groups for the colour ordered and non-colour ordered particles *)
HelAmplN[p : {___}, types : {Except[_List] ...}] :=
 HelAmplN[{Extract[p, Position[types, _?IsUnorderedQ]],
   Extract[p, Position[types, _?IsOrderedQ]]}, {Cases[
    types, _?IsUnorderedQ], Cases[types, _?IsOrderedQ]}]

(* Put the particles into a canonical order *)
HelAmplN[{pNonOrd_List, p_List}, {typesNonOrd_List, types_List}] :=
 Module[{bestI =
```

```
      Sort[Table[{RotateLeft[SortKey /@ types, i], i}, {i, 0,
          Length[types] - 1}]][[1, 2]]},
  HelAmplN[{pNonOrd, RotateLeft[p, bestI]}, {typesNonOrd,
     RotateLeft[types, bestI]}] /; bestI != 0]

(* Allow the momenta for the particles to be passed in directly rather
   than requiring previously declared momenta names *)
HelAmplN[{pVNonOrd : {_List ...},
   pV : {_List ...}}, {typesNonOrd_List, types_List}] :=
 Module[{isMassive = IsMassiveQ /@ Join[typesNonOrd, types], n, p, P,
    allp}, n = Plus @@ isMassive /. {False -> {0, 1}, True -> {1, 0}};
  p = Range[n[[2]]];
  P = Table[Symbol["PP" <> ToString[i]], {i, n[[1]]}];
  allp = isMassive[[ ;; ]];
  allp[[Position[isMassive, True] // Flatten]] = P;
  allp[[Position[isMassive, False] // Flatten]] = p;
  MapThread[
   If[#1, DeclareSpinorMomentum,
      DeclareLVectorMomentum][#2, #3] &, {isMassive, allp,
     Join[pVNonOrd, pV]}, 1];
  HelAmplN[{allp[[;; Length[pVNonOrd]]],
     allp[[Length[pVNonOrd] + 1 ;;]]}, {typesNonOrd, types}]]

(* Declare an error message format for use in amplitudes *)
HelAmplN::badamp =
  "Bad amplitude evaluated for particles of types `` called `` has \
bad value ``";

(* Declare the vanishing, MHV and anti-MHV pure gluon amplitudes *)
HelAmplN[{{},
    p_List}, {{}, {Gluon[-1] | PatternSequence[], Gluon[1], Gluon[1],
     Gluon[1] ..}}] := 0
HelAmplN[{{},
    p_List}, {{}, {Gluon[-1], Gluon[-1], Gluon[-1] ..,
     Gluon[1] | PatternSequence[]}}] := 0

HelAmplN[{{}, p_List}, {{},
    types : {Gluon[-1], Gluon[-1], Gluon[1]}}] :=
 I*Module[{num = Spaa[p[[1]], p[[2]]] // NN,
     denom = Product[Spaa[LL[p, i], LL[p, i + 1]], {i, Length[p]}] //
        NN}, If[num == 0 && denom == 0,
     Message[HelAmplN::badamp, types, p,
```

```
      HoldForm[Num^4/Denom] /. {Num -> num, Denom -> denom}]; 0,
     num^4/denom]]
HelAmplN[{{}, p_List}, {{},
    types : {Gluon[-1], Gluon[1], Gluon[1]}}] :=
 I*(-1)^Length[p]*
  Module[{num = Spbb[p[[2]], p[[-1]]] // NN,
     denom = Product[Spbb[LL[p, i], LL[p, i + 1]], {i, Length[p]}] //
       NN}, If[num == 0 && denom == 0,
     Message[HelAmplN::badamp, types, p,
      HoldForm[Num^4/Denom] /. {Num -> num, Denom -> denom}]; 0,
     num^4/denom]]

HelAmplN[{{},p_List}, {{},{Gluon[-1],gp1:(Gluon[1]...),Gluon[-1],Gluon[1]...}}] :=
 I*Spaa[p[[1]], p[[Length[{gp1}]+2]]]^4/
  Product[Spaa[LL[p, i], LL[p, i + 1]], {i, Length[p]}]//NN
HelAmplN[{{},p_List}, {{},{gm1:Gluon[-1]...,Gluon[1],gm2:Gluon[-1]...,Gluon[1]}}] :=
 I*(-1)^Length[p]*Spbb[p[[Length[{gm1}]+1]], p[[-1]]]^4/
  Product[Spbb[LL[p, i], LL[p, i + 1]], {i, Length[p]}]//NN

(* Declare the vanishing, MHV and anti-MHV amplitudes with one quark pair *)
HelAmpl[{{}, _List}, {{}, {Quark[_, _, _], Quark[_, _, _],
    Gluon[hg_], Gluon[hg_] ..}}, _ : Null] := 0
HelAmpl[{_List, _List}, {_List, {Quark[pq1_, hq1_, f1_],
     Quark[pq2_, hq2_, f2_], Gluon[_] ...}} _ : Null] :=
 0 /; pq1 == pq2 || hq1 == hq2 || f1 != f2

HelAmplN[{_, _}, {_List, {_Quark, _Quark}}] := 0

HelAmplN[{{}, p : {pq1_, pq2_, pg_}},
   types : {{}, {Quark[Pq1_, hq1_, f_], Quark[Pq2_, hq2_, f_],
     Gluon[hg_]}}, epsilon_ : Epsilon[]] :=
 If[$UseBHSigns, 1, -Pq1 hq1] If[hq1 == 1, -(-1)^Length[p], 1] I*
   Module[{num =
       If[hg == hq1, If[hg == -1, Spaa[pg, pq1], Spbb[pg, pq1]],
         If[hg == -1, Spaa[pg, pq2], Spbb[pg, pq2]]] // NN,
     denom = If[hg == -1, Spaa[pq2, pq1], Spbb[pq2, pq1]] // NN},
    If[num == 0 && denom == 0,
     Message[HelAmplN::badamp, types, p,
      HoldForm[Num^2/Denom] /. {Num -> num, Denom -> denom}]; 0,
```

```
      num^2/denom]] /; Pq1 == -Pq2 && hq1 == -hq2

HelAmplN2QuarksH1H3[Sptype_, h1_, h3_, {p1_, p2_, p3_}] :=
 NN[Sptype[p1, p3]]^2 /; h1 == h3
HelAmplN2QuarksH1H3[Sptype_, h1_, h3_, {p1_, p2_, p3_}] :=
 NN[Sptype[p2, p3]]^2 /; h1 == -h3

HelAmplN2Quarks[Sptype_, h1_, h3_, {p1_, p2_, p3_}, p_List] :=
 I*HelAmplN2QuarksH1H3[Sptype, h1, h3, {p1, p2, p3}]*
  NN[Sptype[p1, p3]*
     Sptype[p2, p3]/
      Product[Sptype[LL[p, i], LL[p, i + 1]], {i, Length[p]}]]

HelAmplN[{{},
    p : {pq1_, pq2_, pgs__}}, {{}, {Quark[Pq1_, hq1_, f_],
     Quark[Pq2_, hq2_, f_], go1 : Gluon[hog_] ..., Gluon[hg_],
     Gluon[hog_] ...}}, epsilon_: Epsilon[], hg_|PatternSequence[]] :=
  If[$UseBHSigns,1,-Pq1*hq1]*If[hg == 1, -(-1)^Length[p], 1]*
   HelAmplN2Quarks[If[hg == -1, Spaa, Spbb], hq1,
    hg, {pq1, pq2, {pgs}[[Length[{go1}] + 1]]}, p] /; Pq1==-Pq2 && hq1==-hq2 && hog==-hg

(* Declare the vanishing, MHV and anti-MHV amplitudes with a phi *)
HelAmplN[{{_},p_List}, {{Phi[1]},{Gluon[_],Gluon[1]...}}] := 0
HelAmplN[{{_},p_List}, {{Phi[1]},{Gluon[-1],gp1:(Gluon[1]...),Gluon[-1],Gluon[1]...}}] :=
 I*Spaa[p[[1]], p[[Length[{gp1}]+2]]]^4/
  Product[Spaa[LL[p, i], LL[p, i + 1]], {i, Length[p]}]//NN
HelAmplN[{{_},p_List}, {{Phi[1]},{Gluon[-1]..}}] :=
 I*((-1)^Length[p]*s @@ p^2)/Product[Spbb[LL[p, i], LL[p, i + 1]],
     {i, Length[p]}]//NN

(* Declare the 3 negative, Next-to-MHV amplitude. This uses an arbitrary reference vector which is generated at random *)
HelAmplPhiRatioSum[p_, l1_, l2_, qs_, qe_, r_] :=
 HelAmplPhiRatioSum[p, LLMod[p, l1], LLMod[p, l2], LLMod[p, qs],
   LLMod[p, qe], r] /; l1 > Length[p] || l2 > Length[p] ||
   qs > Length[p] || qe > Length[p]
HelAmplPhiRatioSum[p_, e_, e_, e_, e_, r_] := 1
HelAmplPhiRatioSum[p_, l1_, l2_, qs_, qe_, r_] :=
 Sum[Spab[LL[p, l1], LL[p, qi], r], {qi, LLRange[p, qs, qe]}]/
  Sum[Spab[LL[p, l2], LL[p, qi], r], {qi, LLRange[p, qs, qe]}]
```

```
HelAmplPhi3NegTerm[p_, {m1_, m2_, m3_}, i_, j_, qs_, qe_,epsilon_] :=
 (Spaa[LL[p, m2], LL[p, m3]]^4*Spaa[LL[p, i], LL[p, i + 1]]*
   Spaa[LL[p, j], LL[p, j + 1]]*HelAmplPhiRatioSum[p, m1, i, qs, qe,
    epsilon]*HelAmplPhiRatioSum[p, m1, j, qs, qe, epsilon]*
   HelAmplPhiRatioSum[p, m1, i + 1, qs, qe, epsilon]*
   HelAmplPhiRatioSum[p, m1, j + 1, qs, qe, epsilon])/
  s @@ LL[p, LLRange[p, qs, qe]]
HelAmplPhi3Neg[p_, {m1_, m2_, m3_},epsilon_] :=
 Sum[HelAmplPhi3NegTerm[p, {m1, m2, m3}, i, j, i + 1, j,epsilon] +
   HelAmplPhi3NegTerm[p, {m1, m2, m3}, i, j, j + 1, i,epsilon],
  {i, LLRange[p, m1, m2 - 1]}, {j, LLRange[p, m3, m1 - 1]}]

RandomNSphere[n_] :=
 Module[{tmp = RandomVariate[MultinormalDistribution[
      ConstantArray[0, n], IdentityMatrix[n]]]},
  tmp/Sqrt[Plus @@ (tmp*tmp)]]
RandomNBall[n_] := Random[Real]^(1/n)*RandomNSphere[n]
Random4Vector[Real, m_:0, scale_:1] :=
 Module[{tmp = scale*RandomNBall[3]},
  Prepend[tmp, (2*Random[Integer] - 1)*Sqrt[m^2 + Plus @@ (tmp*tmp)]]]

NWithEpsilons[expr_] :=
 Module[{es = DeleteDuplicates[Cases[expr, Epsilon[___],
      100]]}, PRINT["Generating Momenta for", es];
   (DeclareSpinorMomentum[#1, Random4Vector[Real]] & ) /@ es; N[expr]]

HelAmplN[{{_},p_List}, {{Phi[1]},{Gluon[-1],gp1:Gluon[1]...,Gluon[-1],
      gp2:Gluon[1]...,Gluon[-1],Gluon[1]...}}] :=
 NWithEpsilons[I*Sum[HelAmplPhi3Neg[p, RotateLeft[{1,Length[{gp1}]+2,
      Length[{gp1,gp2}]+3}, i],Epsilon[]], {i, 3}]/
  Product[Spaa[LL[p, i], LL[p, i + 1]], {i, Length[p]}]]

(* Will return all ways of splitting a list of unique elements into two
     parts *)
AllSplits[l_List] := {#, Complement[l, #]} & /@ Subsets[l]

(* Will return all possible propagators for a BCFW cut *)
PropsForCut[{Gluon[_] ...}, {a_, b_}] := {Gluon[1], Gluon[-1]}
PropsForCut[types_List, {a_, b_}] :=
 PropsForCut[RotateLeft[types, b - 1][[;; LLMod[types, a - b]]]]
PropsForCut[{left___, Gluon[_] .., right___}] :=
 PropsForCut[{left, right}]
```

```
PropsForCut[{left___, Quark[p1_, h1_, f_], Quark[p2_, h2_, f_],
   right___}] := PropsForCut[{left, right}] /; p1 == -p2 && h1 == -h2
PropsForCut[{}] := {Gluon[1], Gluon[-1]}
PropsForCut[{q : Quark[_, _, _]}] := {q}
PropsForCut[{_, __}] := {}

(* Check if a pair of particles are valid for the BCFW shift particles i
     and j *)
InvalidShiftsQ[_List, {Gluon[1] | Quark[_, 1, _],
   Gluon[-1] | Quark[_, -1, _]}, {_Integer, _Integer}] := True
InvalidShiftsQ[
  types_List, {Quark[pa_, _, f_], Quark[pb_, _, f_]}, {a_Integer,
   b_Integer}] :=
 True /; pa == -pb && (LLMod[types, a + 1] == b ||
     LLMod[types, b + 1] == a)
InvalidShiftsQ[types_List, {Quark[_, -1, _], Gluon[-1]} | {Gluon[1],
    Quark[_, 1, _]}, {a_Integer, b_Integer}] := True /;
      (LLMod[types, a + 1] == b || LLMod[types, b + 1] == a)
InvalidShiftsQ[_List, {_, _}, {_Integer, _Integer}] := False

(* Check for the factor introduced in the spinors by reversing the
    momenta *)
ReversingFactor[mom_List] := (DeclareSpinorMomentum[spinor, mom];
  DeclareSpinorMomentum[mspinor, -mom];
  NN[La[spinor]][[1, 1]]/NN[La[mspinor]][[1, 1]])
ReversingFactor[sm : {{_, _}, {_, _}}] :=
 ReversingFactor[PfromSm2[sm]]

(* The extra factor needed to correct for how the momenta either side of
     the cut are defined *)
PropogatorFactor[Gluon[_],_]:=1
PropogatorFactor[Quark[p_,_,_],mom_]:=ReversingFactor[-p*mom]


(* Calculate the momenta or particle types to use for the amplitude on
    one side of a cut *)
HelAmplBCFCombineSide[{NonOrd_List, ord_List}, {a_, b_},
  splitNonOrd_List, prop_] := {Part[NonOrd, splitNonOrd],
  Append[LL[ord, LLRange[ord, a, b - 1]], prop]}

(* Evaluate a BCFW term *)
HelAmplNBCFTermImpl[{pNonOrd_List, p_List}, {i_, j_}, {a_,
```

```
    b_}, {leftNonOrd_, rightNonOrd_}, prop_,
   lefttypes : {_List, _List}, righttypes : {_List, _List}] :=
  Module[{z, ii, jj, shiftedp, q, mq, qp, A},
   z = z /.
      Flatten[ExpandSToSpinors[
        SpOpen[Solve[
          0 == ShiftBA[LL[p, i], LL[p, j], z][
            s @@ Join[Part[pNonOrd, leftNonOrd],
              LL[p, LLRange[p, a, b - 1]]]], z]]]] // NN;
   WithSpinors[shiftedp = ReplacePart[p, {i -> ii, j -> jj}];
    qp = PfromSm2[(Sum[
          Sm2[LL[shiftedp, qs]], {qs, LLRange[p, a, b - 1]}] +
         Sum[Sm2[pp], {pp, Part[pNonOrd, leftNonOrd]}]) // NN];
    WithSpinors[
     A = HelAmplN[
        HelAmplBCFCombineSide[{pNonOrd, shiftedp}, {a, b}, leftNonOrd,
         mq], lefttypes]*
       HelAmplN[
        HelAmplBCFCombineSide[{pNonOrd, shiftedp}, {b, a}, rightNonOrd,
          q], righttypes];
     A*I/NN[
        s @@ Join[Part[pNonOrd, leftNonOrd],
          LL[p, LLRange[p, a, b - 1]]]]*PropogatorFactor[prop, qp], {q,
       qp}, {mq, -qp}], {ii, {La[p[[i]]] // NN,
      Lat[p[[i]]] - z*Lat[p[[j]]] //
       NN}}, {jj, {La[p[[j]]] + z*La[p[[i]]] // NN, Lat[p[[j]]] // NN}}]]

(* Declare some terms that are known to vanish based on the particle
   types on each side *)
HelAmplNBCFTerm[{pNonOrd_List, p_List}, {i_, j_}, {a_,
    b_}, {leftNonOrd_, rightNonOrd_}, prop_,
   lefttypes : {_List, _List},
   righttypes : {{}, {Gluon[h_] | Quark[_, h_, _],
       Gluon[h_] | Quark[_, h_, _],
       Gluon[h_] | Quark[_, h_, _]} | {Gluon[1] | Quark[_, 1, _],
       Gluon[1] | Quark[_, 1, _],
       Gluon[-1] | Quark[_, -1, _]} | {Gluon[1] | Quark[_, 1, _],
       Gluon[-1] | Quark[_, -1, _],
       Gluon[1] | Quark[_, 1, _]} | {Gluon[-1] | Quark[_, -1, _],
       Gluon[1] | Quark[_, 1, _], Gluon[1] | Quark[_, 1, _]}}] := 0
HelAmplNBCFTerm[{pNonOrd_List, p_List}, {i_, j_}, {a_,
    b_}, {leftNonOrd_, rightNonOrd_}, prop_,
```

```
  lefttypes : {{}, {Gluon[h_] | Quark[_, h_, _],
      Gluon[h_] | Quark[_, h_, _],
      Gluon[h_] | Quark[_, h_, _]} | {Gluon[1] | Quark[_, 1, _],
      Gluon[-1] | Quark[_, -1, _],
      Gluon[-1] | Quark[_, -1, _]} | {Gluon[-1] | Quark[_, -1, _],
      Gluon[1] | Quark[_, 1, _],
      Gluon[-1] | Quark[_, -1, _]} | {Gluon[-1] | Quark[_, -1, _],
      Gluon[-1] | Quark[_, -1, _], Gluon[1] | Quark[_, 1, _]}},
  righttypes : {_List, _List}] := 0

(* Will swap the sides to put the term in a canonical order. The rule is
    if there is a single non-colour ordered particle it goes in the
    left half, otherwise the left half will be the smaller half in terms
    of number of particles. If neither of these rules choose an order
    the current order is used *)
HelAmplNBCFTerm[{pNonOrd_List, p_List}, {i_, j_}, {a_, b_},
  nonordsplit : {{}, {}} | {{__}, {__}}, prop_,
  lefttypes : {_List, _List}, righttypes : {_List, _List}] :=
 HelAmplNBCFTermImpl[{pNonOrd, p}, {i, j}, {b, a},
   Reverse[nonordsplit], ReverseParticle[prop], righttypes,
   lefttypes] /; Mod[b - a, Length[p]] > Length[p]/2
HelAmplNBCFTerm[{pNonOrd_List, p_List}, {i_, j_}, {a_,
   b_}, {{leftnonordsplit__}, {}}, prop_, lefttypes : {_List, _List},
  righttypes : {_List, _List}] :=
 HelAmplNBCFTermImpl[{pNonOrd, p}, {i, j}, {b,
   a}, {{}, {leftnonordsplit}}, ReverseParticle[prop], righttypes,
  lefttypes]
HelAmplNBCFTerm[{pNonOrd_List, p_List}, {i_, j_}, {a_,
   b_}, {leftNonOrd_, rightNonOrd_}, prop_,
  lefttypes : {_List, _List}, righttypes : {_List, _List}] :=
 HelAmplNBCFTermImpl[{pNonOrd, p}, {i, j}, {a, b}, {leftNonOrd,
   rightNonOrd}, prop, lefttypes, righttypes]

(* Declare some BCFW terms that are known to vanish *)
HelAmplNBCFTerm[{{}, p_List}, {{}, _List}, {_, _}, {a_,
   b_}, {{}, {}}, _] :=
 0 /; a == LLMod[p, b + 1] || LLMod[p, a + 1] == b
HelAmplNBCFTerm[{_List, p_List}, {_List, _List}, {_, _}, {a_,
   b_}, {{}, {__}}, _] := 0 /; LLMod[p, a + 1] == b
HelAmplNBCFTerm[{_List, p_List}, {_List, _List}, {_, _}, {a_,
   b_}, {{__}, {}}, _] := 0 /; a == LLMod[p, b + 1]

```

```
(* Calculate the types of particle on each side of the cut *)
HelAmplNBCFTerm[p : {_List , _List},
  types : {_List , _List}, {i_, j_}, {a_, b_}, {leftNonOrd_,
   rightNonOrd_}, prop_] :=
 HelAmplNBCFTerm[p, {i, j}, {a, b}, {leftNonOrd, rightNonOrd}, prop,
  HelAmplBCFCombineSide[types, {a, b}, leftNonOrd, prop],
  HelAmplBCFCombineSide[types, {b, a}, rightNonOrd,
   ReverseParticle[prop]]]


(* Evaluate an amplitude for a given choice of shift particles *)
HelAmplNBCF[{pNonOrd_List, p_List}, {typesNonOrd_List,
    types_List}, {i_, j_}] := Sum[
   HelAmplNBCFTerm[{pNonOrd, p}, {typesNonOrd, types}, {i, j}, {a, b},
    splitNonOrd, prop], {a, LLRange[p, i + 1, j]}, {b,
    LLRange[p, j + 1, i]}, {splitNonOrd,
     AllSplits[Range[Length[pNonOrd]]]}, {prop,
    PropsForCut[types, {a, b}]}]

(* Try different shifts until one evaluates successfully *)
HelAmplNBCFSearch[{pNonOrd_List, p_List}, {typesNonOrd_List,
    types_List}] :=
 Catch[Do[If[!
      InvalidShiftsQ[types, {types[[i]], types[[j]]}, {i, j}],
     Module[{val =
         HelAmplNBCF[{pNonOrd, p}, {typesNonOrd, types}, {i, j}]},
      If[! MatchQ[
          val, (_ : 1)*Infinity | ComplexInfinity | Indeterminate],
       Throw[val]]]], {i, Length[p]}, {j,
     LLRange[p, i + 1, i - 1]}]; Indeterminate]

(* Evaluate the amplitude using BCFW. Will only be used if none of the
    rules for specific types of amplitude match *)
HelAmplN[{pNonOrd_List, p_List}, {typesNonOrd_List, types_List}] :=
 HelAmplNBCFSearch[{pNonOrd, p}, {typesNonOrd, types}] /; Length[pNonOrd
    ]+Length[p] > 3
```

Listing B.3: Loop-Cuts.txt

```
(* A safe list part which only evaluates when the value to index is a
    list *)
SPart[i_][l_List] := Part[l, i]
\[GothicCapitalP] = SPart;
```

```
(* Declare momenta sets and processes, including which quarks are
    travelling left *)
Process[_] := {{}, {}}
LeftQuarks[_] := {}

DeclareProcess[p : {_List, _List}, leftquarks_: {}] :=
 Module[{process}, Process[process] = p; LeftQuarks[process] =
    leftquarks;
  process]

MomConf[_] := {{}, {}}

DeclareMomConf[p : {_List, _List}] :=
 Module[{momconf}, MomConf[momconf] = p; momconf]

(* Convert names to objects that can be added *)
ToSp[l_List] := ToSp /@ l
ToSp[i_Integer] := Sp[i]
ToSp[Sp[i_]] := Sp[i]
ToSp[s_Symbol /; LVectorQ[s]] := s

(* Declare the reverse spinors for an already defined momenta *)
DeclareReverseSpinor[spinors_List, reverse_List] :=
 MapThread[DeclareReverseSpinor, {spinors, reverse}]
DeclareReverseSpinor[sp : (_Integer | _?SpinorQ),
  r : (_Integer | _Symbol)] := (DeclareSpinorMomentum[r, I*La[sp] // NN,
   I*Lat[sp] // NN]; {sp, r})

(* The factor introduced by reversing a particle *)
LoopReversingFactor[Quark[p_, ___]] := p I
LoopReversingFactor[_Gluon] := -1

(* Calculate the amplitude for a corner given the cut particles and the
    loop momenta defined as pointing around the loop *)
CalculateLoopCornerAmplitude[{pNonOrd_List,
   p_List}, {typesNonOrd_List, {g1 : (Gluon[_] ...), Quark[q1_, h1_, f_
    ],
    mid___, Quark[q2_, h2_, f_], g2 : (Gluon[_] ...)}}, {Quark[q1_, h1_,
     f_],
   Quark[q1_, h1_, f_]}, {l1_, l2_}] :=
 CalculateLoopCornerAmplitude[{pNonOrd,
```

```
    p}, {typesNonOrd, {g1, Quark[q1, h1, f], mid, Quark[q2, h2, -f - 1],
     g2}}, {Quark[q1, h1, f], Quark[q1, h1, -f - 1]}, {l1, l2}]
CalculateLoopCornerAmplitude[{pNonOrd_List, p_List}, {typesNonOrd_List,
    types_List}, {t1_, t2_}, {l1_, l2_}] :=
 WithSpinors[
  Module[{tmp}, DeclareReverseSpinor[ll1, ll1r];
    tmp = HelAmplN[{pNonOrd, Join[p, {ll2, ll1r}]}, {typesNonOrd,
        Join[types, {t2, ReverseParticle[t1]}]}]*LoopReversingFactor[t1];
    FullyUndeclareSpinor[ll1r]; tmp], {ll1, l1}, {ll2, l2}]
CalculateLoopCornerAmplitude[{pNonOrd_List, p_List}, {typesNonOrd_List,
    types_List}, {t1_,
    t2_}, {{(Indeterminate | {(Indeterminate | {Indeterminate ...}) ...})
      ...},
     l2_}] := Indeterminate
CalculateLoopCornerAmplitude[{pNonOrd_List, p_List}, {typesNonOrd_List,
    types_List}, {t1_,
    t2_}, {l1_, {(Indeterminate | {(Indeterminate | {Indeterminate ...})
     ...}) \
...}}] := Indeterminate


(* Split the external particles into the different corners *)
SplitCorners[{Unord_List, Ord_List}, {splits_List, splitUnord_List}] :=
 Transpose[{Extract[Unord, Position[splitUnord, #]] & /@
     Range[Length[splits]],
    RotateLeft[Ord, #1 - 1][[;; #2 - #1]] & @@@
     Partition[splits, 2, 1, 1, Length[Ord] + splits[[1]]]}]

(* Combine the external particles in each corner with the loop
     propagators to give the particles in each corner *)
CombineCornerPropsTypes[corners : {{_List, _List} ...}, props_List] :=
 MapThread[{#1[[1]],
     Join[#1[[2]], #2]} &, {corners, {#[[2]], ReverseParticle[#[[1]]]} &
     /@
     Partition[props, 2, 1, 1]}]

(* Calculate at what index a new split must be inserted to maintain the
     order of the cuts *)
InsertionIndex[split_, corner_, newsplit_] :=
 0 /; corner == Length[split] &&
    newsplit <= split[[1]] && ! newsplit == split[[corner]]
InsertionIndex[split_, corner_, newsplit_] := corner
```

```

(* Calculate the name for a new combined split *)
SplitName[{{}, nonord_List}, {corner_, newsplit_Integer,
    newNonord_List}] := {{newsplit},
  PadLeft[{}, Length[newNonord], 1]}
SplitName[{splits_List, nonord_List}, {corner_Integer,
    newsplit_Integer, newNonord_List}] :=
 Module[{insert = InsertionIndex[splits, corner, newsplit]}, {Insert[
     splits, newsplit, insert + 1],
   MapThread[
    Mod[#1 + If[(#1 == corner) && #2 > 0, 1, 0] +
         If[#1 > insert, 1, 0] - 1, Length[splits] + 1] + 1 &, {nonord,
      newNonord}]}]

(* Replace new splits that are completely invalid with Sequence[] *)
RemoveInvalidSplits[
  process_, {split_, splitNonord_}, {c_, i_, nonordshift_}] :=
 Sequence[] /; split[[c]] == i && Count[nonordshift, 0] == 0
RemoveInvalidSplits[
  process_, {split_, splitNonord_}, {c_, i_, nonordshift_}] :=
 Sequence[] /; (Length[split] !=
      c || (i <= split[[1]] && ! split[[c]] == i)) &&
   split[[Mod[c, Length[split]] + 1]] == i &&
   Count[nonordshift, 1] == 0
RemoveInvalidSplits[process_, {split_, splitNonord_},
  newsplit_] := newsplit

(* Calculate the list of possible new splits given an existing split *)
SplitOptions[process_, {{}, splitNonord_List}] :=
 Table[{None, i, Table[0, {Length[Process[process][[1]]]}]}, {i,
   Length[Process[process][[2]]]}]
SplitOptions[process_, {splits_List, splitNonord_List}] :=
 RemoveInvalidSplits[process, {splits, splitNonord}, #] & /@
  Flatten[Table[
    Module[{nonordindex = Position[splitNonord, c]},
     Table[{c,
       Mod[i + splits[[c]] - 1, Length[Process[process][[2]]]] + 1,
       ReplacePart[PadRight[{}, Length[splitNonord], None],
        MapThread[Rule, {nonordindex, non}]]}, {i, 0,
       If[c == Length[splits],
        splits[[1]] - splits[[c]] + Length[Process[process][[2]]],
        splits[[c + 1]] - splits[[c]]]}, {non,
```

```
        Tuples[{0, 1}, Length[nonordindex]]}]], {c, 1,
      Length[splits]}], 2]

(* Calculate all possible splits into n corners *)
SplitOptions[process_, 0] := {{{}, {}}}
SplitOptions[process_, n_Integer] :=
 Union @@
  Function[name,
    SplitName[name, #] & /@ SplitOptions[process, name]] /@
   SplitOptions[process, n - 1]

MatchQuarks[Quark[p_, _, f_], process_] :=
 MemberQ[LeftQuarks[process], Quark[p, f]]

SplitPropOptions[process_, {splits : {}, _List},
  loopprops : {}, {corner_, newsplit_Integer, _List}] :=
 Module[{quarks = Cases[LeftQuarks[process], Quark[p_, -1] :> p],
   cornertypes =
    RotateLeft[Process[process][[2]], newsplit - 1] /.
      Gluon[_] -> Sequence[] //. {o1___,
       Quark[p1_, h1_, f_]?(MatchQuarks[#, process] &),
       Quark[p2_, h2_, f_], o2___} :> {o1, o2} /;
       p1 == -p2 && h1 == -h2},
  If[Length[cornertypes] == 0,
   If[Length[quarks] ==
     1, {{Quark[quarks[[1]], 1, -1]}, {Quark[
       quarks[[1]], -1, -1]}}, {{Gluon[1]}, {Gluon[-1]}}],
   If[And[Length[quarks] == 0,
     MatchQ[cornertypes, {Quark[p1_, h1_, f_],
        Quark[p2_, h2_, f_]?(MatchQuarks[#, process] &)} /;
       p1 == -p2 && h1 == -h2]], {{cornertypes[[1]]}}, {}]]]

(* Calculate the possible loop propagators given a split and its
    propagators and the new cut being added *)
SplitPropOptions[process_, {splits : {__Integer}, nonord_List},
  loopprops : {_List ...}, {corner_Integer, newsplit_Integer,
   newNonord_List}] :=
 Flatten[SplitPropOptions[
     process, {splits, nonord}, #, {corner, newsplit, newNonord}] & /@
    loopprops, 1]
SplitPropOptions[process_, {splits : {__Integer}, nonord_List},
  loopprops_List, {corner_Integer, newsplit_Integer,
```

```
    newNonord_List}] :=
 Module[{index = InsertionIndex[splits, corner, newsplit],
    name = SplitName[{splits, nonord}, {corner, newsplit, newNonord}],
    cornerparts, options},
   cornerparts =
    CombineCornerPropsTypes[SplitCorners[Process[process], name],
      Insert[loopprops, None, index + 1]][[index + 1, 2]];
   cornerparts =
    cornerparts /.
      Gluon[_] | ReverseParticle[None] -> Sequence[] //. {o1___,
       Quark[p1_, h1_, f_]?(MatchQuarks[#, process] &),
       Quark[p2_, h2_, f_], o2___} :> {o1, o2} /;
       p1 == -p2 && h1 == -h2;
   options =
    If[Length[cornerparts] == 0, {Gluon[1], Gluon[-1]},
     If[Length[cornerparts] == 1, cornerparts, {}]];
   Insert[loopprops, #, index + 1] & /@ options]

(* Calculate all possible propagators for a given split *)
SplitPropOptions[process_, {splits_List, nonord_List}] :=
 Module[{name = {{}, {}}, options = {}},
  Do[options =
     SplitPropOptions[process, name,
      options, {i - 1, splits[[i]], nonord - i + 1}];
    name = SplitName[name, {i - 1, splits[[i]], nonord - i + 1}], {i,
     Length[splits]}]; options]

(* Calculate the helicity of a corner if it has one, or return None *)
CornerHelicity[nonels_List, els_List] :=
 CornerHelicity[Sort[nonels],
   Sort[els]] /; ! (OrderedQ[els] && OrderedQ[nonels])

CornerHelicity[{}, {Gluon[h_], Quark[_, _, _], Quark[_, _, _]}] := h
CornerHelicity[{}, {Gluon[-1], Gluon[-1], Gluon[1]}] := -1
CornerHelicity[{}, {Gluon[-1], Gluon[1], Gluon[1]}] := 1
CornerHelicity[_List, _List] := None

(* Remove from the list of propagators all combinations that are known
    to vanish for any reason *)
RemoveIgnorableOptions[process_, {splits : {__Integer}, nonord_List},
   loopprops : {_List ...}] :=
 RemoveIgnorableOptions[process, {splits, nonord}, #] & /@ loopprops
```

```
RemoveIgnorableOptions[process_, {splits : {_}, nonord_List},
  loopprops_List] := Sequence[]
RemoveIgnorableOptions[process_, {splits : {i_, j_}, {2 ...}},
  loopprops_List] :=
 Sequence[] /; Mod[i, Length[Process[process][[2]]]] + 1 == j
RemoveIgnorableOptions[process_, {splits : {i_, j_}, {1 ...}},
  loopprops_List] :=
 Sequence[] /; i == Mod[j, Length[Process[process][[2]]]] + 1
RemoveIgnorableOptions[process_, {splits : {__Integer}, nonord_List},
  loopprops_List] :=
 RemoveIgnorableOptions[process, {splits, nonord}, loopprops,
  Map[Sort,
   CombineCornerPropsTypes[
    SplitCorners[Process[process], {splits, nonord}], loopprops], 2]]
RemoveIgnorableOptions[process_, {splits : {__Integer}, nonord_List},
  loopprops_List, {___, {{}, {Gluon[h_] ...}}, ___}] := Sequence[]
RemoveIgnorableOptions[process_, {splits : {__Integer}, nonord_List},
  loopprops_List, {___, {{}, {Gluon[h1_], Gluon[h2_], Gluon[h2_],
      Gluon[h2_] ..} | {Gluon[h1_], Gluon[h1_], Gluon[h1_] ..,
      Gluon[h2_]}}, ___}] := Sequence[]
RemoveIgnorableOptions[process_, {splits : {__Integer}, nonord_List},
  loopprops_List, {___, {{}, {Gluon[h_],
     Gluon[h_] .., _Quark, _Quark}}, ___}] := Sequence[]
RemoveIgnorableOptions[process_, {splits : {__Integer}, nonord_List},
  loopprops_List, {___, {{Phi[p_]}, {Gluon[p_],
      Gluon[h_]} | {Gluon[h_], Gluon[p_]}}, ___}] := Sequence[]
RemoveIgnorableOptions[process_, {splits : {__Integer}, nonord_List},
  loopprops_List, {___, {{Phi[
      p_]}, {Gluon[p_] ..., _Quark, _Quark}}, ___}] := Sequence[]
RemoveIgnorableOptions[process_, {splits : {__Integer}, nonord_List},
  loopprops_List, {___, {{}, {Gluon[h1_], Gluon[h : (h1_ | h2_)],
      Gluon[h2_]} | {Gluon[h_], _Quark, _Quark}}, {{}, {Gluon[h1_],
      Gluon[h : (h1_ | h2_)],
      Gluon[h2_]} | {Gluon[h_], _Quark, _Quark}}, ___}] := Sequence[]
RemoveIgnorableOptions[process_, {splits : {__Integer}, nonord_List},
  loopprops_List, {{{}, {Gluon[h1_], Gluon[h : (h1_ | h2_)],
      Gluon[h2_]} | {Gluon[h_], _Quark, _Quark}}, ___, {{}, {Gluon[
       h1_], Gluon[h : (h1_ | h2_)],
      Gluon[h2_]} | {Gluon[h_], _Quark, _Quark}}}] := Sequence[]
RemoveIgnorableOptions[process_, {splits : {__Integer}, nonord_List},
  loopprops_List, {pre___, {{}, {Gluon[h1_], Gluon[h1_],
     Gluon[h2_]}}, _, {{}, {Gluon[h1_], Gluon[h2_], Gluon[h2_]}},
```

```
   post___}] :=
 Sequence[] /; Length[{pre, post}] == 1 (* total length==4 *)
RemoveIgnorableOptions[process_, {splits : {__Integer}, nonord_List},
  loopprops_List, {pre___, {{}, {Gluon[h2_], Gluon[h1_],
     Gluon[h1_]}}, _, {{}, {Gluon[h2_], Gluon[h2_], Gluon[h1_]}},
   post___}] :=
 Sequence[] /; Length[{pre, post}] == 1 (* total length==4 *)

RemoveIgnorableOptions[_, _, loopprops_List, _] := loopprops

SplitValidPropOptions[process_ ,split_ ] := RemoveIgnorableOptions[
  process, split, SplitPropOptions[process, split]]

(* Calculate the momentum solution for a box *)
CalcBoxLoopMom[{{{}, {k1_}}, K2 : {_List, _List}, K3 : {_List, _List},
    K4 : {_List, _List}}, 1] :=
 Module[{k2 = Plus @@ ToSp[K2 // Flatten],
    k3 = Plus @@ ToSp[K3 // Flatten],
    k4 = Plus @@ ToSp[K4 // Flatten]}, {{La[k1],
      CLa[k1].CSm2[k2].Sm2[k3].CSm2[k4]/Spaa[k1, k2, k4, k1]}, {La[k1],
       CLa[k1].CSm2[k4].Sm2[k3].CSm2[k2]/
       Spaa[k1, k2, k4, k1]}, {CSm2[k3].Sm2[k4].La[k1]/
       Spaa[k1, k2, k4, k1],
      CLa[k1].CSm2[k2]}, {CSm2[k3].Sm2[k2].La[k1]/Spaa[k1, k2, k4, k1],
       CLa[k1].CSm2[k4]}} // NN]
CalcBoxLoopMom[{{{}, {k1_}}, K2 : {_List, _List}, K3 : {_List, _List},
    K4 : {_List, _List}}, -1] :=
 Module[{k2 = Plus @@ ToSp[K2 // Flatten],
    k3 = Plus @@ ToSp[K3 // Flatten],
    k4 = Plus @@
      ToSp[K4 // Flatten]}, {{CSm2[k4].Sm2[k3].CSm2[k2].CLat[k1]/
       Spbb[k1, k4, k2, k1],
      Lat[k1]}, {CSm2[k2].Sm2[k3].CSm2[k4].CLat[k1]/
       Spbb[k1, k4, k2, k1], Lat[k1]}, {CSm2[k2].CLat[k1],
      Lat[k1].Sm2[k4].CSm2[k3]/
       Spbb[k1, k4, k2, k1]}, {CSm2[k4].CLat[k1],
      Lat[k1].Sm2[k2].CSm2[k3]/Spbb[k1, k4, k2, k1]}} // NN]

CalcBoxLoopMom[momconf_, {split_List, nonord_List},
  masslessindex_Integer, h_] :=
 CalcBoxLoopMom[momconf, {split, nonord}, masslessindex, h] =
  RotateRight[
```

```
    CalcBoxLoopMom[
     RotateLeft[SplitCorners[MomConf[momconf], {split, nonord}],
      masslessindex - 1], h], masslessindex - 1]

(* Determine if a box has a massless corner and if so where it is *)
MasslessBoxCorner[
  process_, {split : {pre___, i_, j_, ___}, nonord_List}] :=
 Length[{pre}] + 1 /; j == i + 1 && FreeQ[nonord, Length[{pre}] + 1]
MasslessBoxCorner[process_, {split : {1, ___, i_}, nonord_List}] :=
 4 /; Length[Process[process][[2]]] == i && FreeQ[nonord, 4]
MasslessBoxCorner[
  process_, {split : {___Integer}, nonord : {___Integer}}] := None

(* Determine which momenta solution is valid for a given box *)
ValidBoxLoopSolution[{nonord_List, ord_List}] :=
 ValidBoxLoopSolution[Sort[nonord], Sort[ord]]
ValidBoxLoopSolution[{}, {Gluon[h_], Quark[_, _, _],
    Quark[_, _, _]}] := h
ValidBoxLoopSolution[{}, {Gluon[-1], Gluon[-1], Gluon[1]}] := -1
ValidBoxLoopSolution[{}, {Gluon[-1], Gluon[1], Gluon[1]}] := 1
ValidBoxLoopSolution[{}, {_, _, _}] := None

(* Calculate the contribution from a single box *)
(* masslessindex is an integer so there is a massless corner. If there are no massless corners, a version where masslessindex is None must be implemented *)
CalculateBoxContribution[process_,
  momconf_, {split_List, nonord_List}, masslessindex_Integer, props_,
  solutionhel_] :=
 Module[{solution,
     cornertypes = SplitCorners[Process[process], {split, nonord}],
     cornernames = SplitCorners[MomConf[momconf], {split, nonord}]},
    If[solutionhel === None, 0,
     solution =
      CalcBoxLoopMom[momconf, {split, nonord}, masslessindex,
        solutionhel];
     I/2*Product[
       CalculateLoopCornerAmplitude[cornernames[[i]],
         cornertypes[[i]], {props[[i]],
          props[[Mod[i, 4] + 1]]}, {solution[[i]],
          solution[[Mod[i, 4] + 1]]}], {i, Length[split]}]]] /;
  ValidBoxLoopSolution@(CombineCornerPropsTypes[
```

```
    SplitCorners[Process[process], {split, nonord}], props][[
   masslessindex]]) == solutionhel

CalculateBoxContribution[process_,
 momconf_, {split_List, nonord_List}, masslessindex_Integer, props_,
 solutionhel_] :=
 0 /; ValidBoxLoopSolution@(CombineCornerPropsTypes[
    SplitCorners[Process[process], {split, nonord}], props][[
   masslessindex]]) != solutionhel

CalculateBoxContribution[process_,
 momconf_, {split_List, nonord_List}, props_,
 solutionhel_Integer] := (CalculateBoxContribution[process,
  momconf, {split, nonord}, props, solutionhel] =
 CalculateBoxContribution[process, momconf, {split, nonord},
  MasslessBoxCorner[process, {split, nonord}], props, solutionhel])
CalculateBoxContribution[process_,
 momconf_, {split_List, nonord_List}, props_,
 None | PatternSequence[]] :=
 CalculateBoxContribution[process, momconf, {split, nonord},
  props, -1] +
 CalculateBoxContribution[process, momconf, {split, nonord}, props,
  1]

(* Calculate the equation for the triangle loop momentum as a function
    of t, evaluating as much as possible, as early as possible *)
CalcTriLoopMomEqn[{kk1_List, _, kk3_List}, None, h_] :=
 Module[{k1 = Plus @@ ToSp[kk1 // Flatten],
   k3 = Plus @@ ToSp[kk3 // Flatten], K1, K3, S1, S3,
   k1k3, \[CapitalDelta], \[Gamma], K3t, K1t}, K1 = Num4V[k1];
  K3 = Num4V[k3]; S1 = MP2[k1] // NN; S3 = MP2[k3] // NN;
  k1k3 = MP[k1, k3] // NN; \[CapitalDelta] = k1k3^2 - S1 S3; \[Gamma] =
   k1k3 + h*Sqrt[\[CapitalDelta]]; K3t = K3 - S3/\[Gamma] K1;
  K1t = K1 - S1/\[Gamma] K3;
  Function[t,
   Evaluate[
    WithSpinors[{{t La[k1t] +
        S1*(S3 + \[Gamma])/(4*\[CapitalDelta]) La[
          k3t], -S3*(S1 + \[Gamma])/(4*\[CapitalDelta]*t) Lat[k1t] +
        Lat[k3t]}, {t La[k1t] +
        S1*S3*(S1 + \[Gamma])/(4*\[Gamma]*\[CapitalDelta]) La[
          k3t], -\[Gamma]*(S3 + \[Gamma])/(4*\[CapitalDelta]*t) Lat[
```

```
              k1t] + Lat[
            k3t]}, {t La[
              k1t] + \[Gamma]*(S1 + \[Gamma])/(4*\[CapitalDelta]) La[
              k3t], -S1*
            S3*(S3 + \[Gamma])/(4*\[Gamma]*\[CapitalDelta]*t) Lat[k1t] +
            Lat[k3t]}} // NN, {k1t, K1t}, {k3t, K3t}]]]]

CalcTriLoopMomEqn[corners_List, masslessindex_Integer, 1] :=
 Module[{k1 = Plus @@ ToSp[corners[[masslessindex]] // Flatten],
    k3 = Plus @@
      ToSp[corners[[Mod[masslessindex + 1, 3] + 1]] // Flatten], c,
    K3t}, c = MP2[k3]/(2 MP[k1, k3]) // NN;
   K3t = Num4V[k3] - c Num4V[k1];
   Function[t,
    Evaluate[
     RotateRight[
      WithSpinors[{{t La[k1], -c/t Lat[k1] +
           Lat[k3t]}, {t La[k1], -(c + 1)/t Lat[k1] +
           Lat[k3t]}, {t La[k1] + La[k3t], Lat[k3t]}} // NN, {k3t,
        K3t}], masslessindex - 1]]]]
CalcTriLoopMomEqn[corners_List, masslessindex_Integer, -1] :=
 Module[{k1 = Plus @@ ToSp[corners[[masslessindex]] // Flatten],
    k3 = Plus @@
      ToSp[corners[[Mod[masslessindex + 1, 3] + 1]] // Flatten], c,
    K3t}, c = MP2[k3]/(2 MP[k1, k3]) // NN;
   K3t = Num4V[k3] - c Num4V[k1];
   Function[t,
    Evaluate[
     RotateRight[
      WithSpinors[{{-c /t La[k1] + La[k3t],
          t Lat[k1]}, {-(c + 1)/ t La[k1] + La[k3t],
          t Lat[k1]}, {La[k3t], t*Lat[k1] + Lat[k3t]}} // NN, {k3t,
        K3t}], masslessindex - 1]]]]

CalcTriLoopMomEqn[momconf_, {split_List, nonord_List}, masslessindex_,
    h_] := Module[{corners =
     SplitCorners[MomConf[momconf], {split, nonord}], params},
   CalcTriLoopMomEqn[momconf, {split, nonord}, masslessindex, h] =
    CalcTriLoopMomEqn[corners, masslessindex, h]]

CalcTriLoopMom[momconf_, {split_List, nonord_List}, masslessindex_,
   h_, t_] :=
```

```
CalcTriLoopMomEqn[momconf, {split, nonord}, masslessindex, h][t]

(* Calculate the subtraction of a box from a triangle as a function of t
    , evaluating as much as possible, as early as possible *)
CalculateTriSubtractionEqnImpl[process_, momconf_, split_,
  masslessindex_Integer, solution : (1 | -1), props_] :=
  Function[t,
  Evaluate[Module[{cornernames =
       SplitCorners[MomConf[momconf], split], k1, K3, C},
     k1 = Plus @@ ToSp[cornernames[[masslessindex]] // Flatten];
     K3 = Plus @@
       ToSp[cornernames[[Mod[masslessindex + 1, 3] + 1]] // Flatten];
     C = MP2[K3]/2/MP[k1, K3] // NN;
     WithSpinors[
      Sum[Module[{boxname = SplitName[split, boxsplit],
          insindex =
           InsertionIndex[split[[1]], boxsplit[[1]], boxsplit[[2]]], k},
          k =
          Plus @@ ToSp[
            Flatten[If[insindex >= masslessindex,
              SplitCorners[MomConf[momconf], boxname][[
               masslessindex ;; insindex]],
              RotateLeft[SplitCorners[MomConf[momconf], boxname],
                masslessindex][[
               1 ;; insindex - masslessindex + 4]]]]]; (1/
             If[solution == 1,
              Spab[k1, k,
                k3T]*(t - (MP[k, k] + 2 C MP[k, k1])/Spab[k1, k, k3T]),
              Spab[k3T, k,
                k1]*(t - (MP[k, k] + 2 C MP[k, k1])/Spab[k3T, k, k1])] //
            NN)*Sum[CalculateBoxContribution[process, momconf, boxname,
            boxprops,
            solution*(-1)^(masslessindex +
                If[insindex < masslessindex, 1, 0] -
                MasslessBoxCorner[process, boxname])], {boxprops,
            RemoveIgnorableOptions[process, boxname,
             SplitPropOptions[process, split, props,
              boxsplit]]}]], {boxsplit,
         SplitOptions[process, split]}], {k3T,
       Num4V[K3] - C*Num4V[k1]}]]]]

CalculateTriSubtractionEqnImpl[process_, momconf_, split_,
```

```
 masslessindex : None, h_, props_] :=
Module[{cornernames = SplitCorners[MomConf[momconf], split], k1, k3,
  K1, K3, S1, S3, k1k3, \[CapitalDelta]},
 k1 = Plus @@ ToSp[cornernames[[1]] // Flatten];
 k3 = Plus @@ ToSp[cornernames[[3]] // Flatten]; K1 = Num4V[k1];
 K3 = Num4V[k3]; S1 = MP2[k1] // NN; S3 = MP2[k3] // NN;
 k1k3 = MP[k1, k3] // NN; \[CapitalDelta] = k1k3^2 - S1 S3;
 Function[t,
  Evaluate[
   Module[{\[Gamma] = k1k3 + h*Sqrt[\[CapitalDelta]]},
    WithSpinors[
     Sum[Module[{boxname = SplitName[split, boxsplit],
         insindex =
          InsertionIndex[split[[1]], boxsplit[[1]], boxsplit[[2]]], k,
          f, b, c, j, boxmasslessindex},
        k = Plus @@
          ToSp[Flatten[
            If[insindex >= 1,
             SplitCorners[MomConf[momconf], boxname][[1 ;; insindex]],
              RotateLeft[SplitCorners[MomConf[momconf], boxname], 1][[
              1 ;; insindex + 3]]]]]; f = Spab[k1T, k, k3T] // NN;
        b = (MP[k,
              k] + (S3*(S1 + \[Gamma])*MP[k, k1T] -
                S1*(S3 + \[Gamma])*
                 MP[k, k3T])/(2 \[CapitalDelta]))/(2 f) // NN;
        c = -S1*S3*(S1 + \[Gamma])*(S3 + \[Gamma])*
           Spab[k3T, k, k1T]/((4 \[CapitalDelta])^2*f) // NN;
        j = Sqrt[b^2 - c];
        boxmasslessindex = MasslessBoxCorner[process, boxname];
        Sum[CalculateBoxContribution[process, momconf, boxname,
             boxprops, boxh]*
            If[WithSpinors[(MP[ltm, lb] - MP[ltp, lb])/MP[ltp, ltm] //
                  NN // Re, {lb,
                 CalcBoxLoopMom[momconf, boxname, boxmasslessindex,
                   boxh][[1]]}, {ltp, \[GothicCapitalP][1]@
                  CalcTriLoopMom[momconf, split, masslessindex, h,
                   b + j]}, {ltm, \[GothicCapitalP][1]@
                  CalcTriLoopMom[momconf, split, masslessindex, h,
                   b - j]}] <
              0, -(b - j)/(j*(t - (b - j))), (b +
                j)/(j*(t - (b + j)))], {boxh, -1, 1}, {boxprops,
            RemoveIgnorableOptions[process, boxname,
```

```
            SplitPropOptions[process, split, props, boxsplit]]}]/f/
        2 ], {boxsplit, SplitOptions[process, split]}], {k1T,
      K1 - S1/\[Gamma] K3}, {k3T, K3 - S3/\[Gamma] K1}]]]]]

CalculateTriSubtractionEqn[process_,
  momconf_, {split_List, nonord_List}, masslessindex_, soln_,
  props_] :=
 CalculateTriSubtractionEqn[process, momconf, {split, nonord},
   masslessindex, soln, props] =
  CalculateTriSubtractionEqnImpl[process, momconf, {split, nonord},
   masslessindex, soln, props]

CalculateTriSubtraction[args__, t_] :=
 CalculateTriSubtractionEqn[args][
  t] (* args is always process_,momconf_,split_ ,masslessindex_,soln_,
    props_ *)

(* Check which momentum solutions are valid for a given triangle *)
ValidTriLoopSolution[{nonord_List, ord_List}] :=
 ValidTriLoopSolution[Sort[nonord], Sort[ord]]
ValidTriLoopSolution[{}, {Gluon[h_], Quark[_, _, _],
   Quark[_, _, _]}] := h
ValidTriLoopSolution[{}, {Gluon[-1], Gluon[-1], Gluon[1]}] := -1
ValidTriLoopSolution[{}, {Gluon[-1], Gluon[1], Gluon[1]}] := 1
ValidTriLoopSolution[{}, {_, _, _}] := None

(* Try and find a massless corner in a triangle *)
MasslessTriCorner[
  process_, {split : {pre___, i_, j_, ___}, nonord_List},
  props_List] :=
 Length[{pre}] + 1 /;
  j == i + 1 && FreeQ[nonord, Length[{pre}] + 1] &&
   1 == ValidTriLoopSolution@(CombineCornerPropsTypes[
        SplitCorners[Process[process], {split, nonord}], props][[
      Length[{pre}] + 1]])
MasslessTriCorner[process_, {split : {1, ___, i_}, nonord_List},
  props_List] :=
 3 /; Length[Process[process][[2]]] == i && FreeQ[nonord, 3] &&
   1 == ValidTriLoopSolution@(CombineCornerPropsTypes[
        SplitCorners[Process[process], {split, nonord}], props][[3]])
MasslessTriCorner[
  process_, {split : {pre___, i_, j_, ___}, nonord_List},
```

```
  props_List] :=
 Length[{pre}] + 1 /; j == i + 1 && FreeQ[nonord, Length[{pre}] + 1]
MasslessTriCorner[process_, {split : {1, ___, i_}, nonord_List},
  props_List] :=
 3 /; Length[Process[process][[2]]] == i && FreeQ[nonord, 3]
MasslessTriCorner[
  process_, {split : {___Integer}, nonord : {___Integer}},
  props_List] := None

(* Calculate a raw triangle contribution as a function of t *)
CalculateRawTriContributionImpl[cornernames_, cornertypes_, props_,
  solution_] :=
 1/2 Product[
    CalculateLoopCornerAmplitude[\[GothicCapitalP][i]@
      cornernames, \[GothicCapitalP][i]@
      cornertypes, {\[GothicCapitalP][i]@
       props, \[GothicCapitalP][Mod[i, 3] + 1]@
       props}, {\[GothicCapitalP][i]@
       solution, \[GothicCapitalP][Mod[i, 3] + 1]@solution}], {i, 3}]

CalculateRawTriContribution[process_,
  momconf_, {split_List, nonord_List}, masslessindex_Integer, props_,
  solutionh_, t_] :=
 Module[{cornertypes =
     SplitCorners[Process[process], {split, nonord}],
    cornernames = SplitCorners[MomConf[momconf], {split, nonord}]},
  CalculateRawTriContributionImpl[cornernames, cornertypes, props,
      CalcTriLoopMom[momconf, {split, nonord}, masslessindex,
       solutionh, t]] -
     CalculateTriSubtraction[process, momconf, {split, nonord},
      masslessindex, solutionh, props, t] /;
    solutionh ==
     ValidTriLoopSolution@(CombineCornerPropsTypes[cornertypes,
        props][[masslessindex]])]

CalculateRawTriContribution[process_,
  momconf_, {split_List, nonord_List}, masslessindex_Integer, props_,
  solutionh_, t_] :=
 Module[{cornertypes =
     SplitCorners[Process[process], {split, nonord}]},
  0 /; solutionh !=
     ValidTriLoopSolution@(CombineCornerPropsTypes[cornertypes,
```

```
          props ] [ [ masslessindex ] ] ) ]

CalculateRawTriContribution [ process_ ,
  momconf_ , {split_List , nonord_List } , masslessindex : None , props_ ,
  solutionh_ , t_] :=
 Module[{ cornertypes =
     SplitCorners [ Process [ process ] , {split , nonord } ] ,
    cornernames = SplitCorners [MomConf[momconf] , {split , nonord }]} ,
   CalculateRawTriContributionImpl [ cornernames , cornertypes , props ,
     CalcTriLoopMom [ momconf , {split , nonord } , masslessindex , solutionh ,
       t ] ] − CalculateTriSubtraction [ process , momconf , {split , nonord } ,
      masslessindex , solutionh , props , t ] ]

CalculateRawTriContribution [ process_ ,
  momconf_ , {split_List , nonord_List } , props_ , h_ , t_] :=
 CalculateRawTriContribution [ process , momconf , {split , nonord } ,
  MasslessTriCorner [ process , {split , nonord } , props ] , props , h , t ]

(* Set up the constants used for the extraction of the different
    coefficients *)
NNCache [ t0 , 3 Sqrt [ 2 ] − Pi , 2 ]
pt0 [ ___ ] := t0
p = 9; (* Number of points to use *)
pp [ ___ ] := p
mp = 4; (* Maximum power − high enough for a Higgs boson to work *)
pmp [ ___ ] := mp

(* Calculate the equation for the coefficient of any power in a triangle
    *)
CalculateTriContributionEquation [ process_ ,
  momconf_ , {split_List , nonord_List } , props_ , h_] :=
 CalculateTriContributionEquation [ process , momconf , {split , nonord } ,
    props , h] =
   Module[{p = pp [ process ] , t0 = pt0 [ process ]} ,
    Function [ n ,
     Evaluate [
      Sum[ t0^−n*
          CalculateRawTriContribution [ process ,
           momconf , {split , nonord } , props , h ,
           t0*Exp[2 Pi I j /(2 p + 1 ) ] ]*
         Exp[−2 Pi I j n/(2 p + 1 ) ] , {j , −p , p}]/(2 p + 1 ) // Factor ] ] ]
```

```
CalculateTriContribution[process_,
 momconf_, {split_List, nonord_List}, props_, h_Integer, n_: 0] :=
 CalculateTriContributionEquation[process, momconf, {split, nonord},
   props, h][n]
CalculateTriContribution[process_,
 momconf_, {split_List, nonord_List}, props_, h : None, n_: 0] :=
 CalculateTriContribution[process, momconf, {split, nonord}, props, 1,
    n] + CalculateTriContribution[process, momconf, {split, nonord},
   props, -1, n]
CalculateTriContribution[process_,
 momconf_, {split_List, nonord_List}, props_] :=
 CalculateTriContribution[process, momconf, {split, nonord}, props,
  None, 0]

(* The arbitrary vector used in the bubble momentum parametrisation *)
DeclareSpinorMomentum[\[Chi], {1 + 2 I, 1 - 2 I, 1 + 4 I ,
  NN[Sqrt[15], 100]}]

(* Calculate the bubble loop momentum as a function of t and y, evaluating as much as possible, as early as possible *)
CalcBubbleLoopMomEqn[{kk1_List, _}] :=
 Module[{k1 = Plus @@ ToSp[kk1 // Flatten], K1t, S1o\[Chi]},
  S1o\[Chi] = MP2[k1]/2/MP[k1, \[Chi]] // NN;
  K1t = Num4V[k1] - S1o\[Chi] Num4V[\[Chi]];
  Function[{t, y},
   Evaluate[
    WithSpinors[{{t La[k1t] + (1 - y) S1o\[Chi] La[\[Chi]],
        y/t Lat[k1t] + Lat[\[Chi]]}, {-t La[k1t] +
         y S1o\[Chi] La[\[Chi]], (1 - y)/t Lat[k1t] - Lat[\[Chi]]}} //
      NN, {k1t, K1t}]]]]
CalcBubbleLoopMomEqn[momconf_, {split_List, nonord_List}] :=
 Module[{corners = SplitCorners[MomConf[momconf], {split, nonord}],
   params},
  CalcBubbleLoopMomEqn[momconf, {split, nonord}] =
   CalcBubbleLoopMomEqn[corners]]
CalcBubbleLoopMom[momconf_, {split_List, nonord_List}, t_, y_] :=
 CalcBubbleLoopMomEqn[momconf, {split, nonord}][t, y // NN]

(* Calculate the raw and unsubtracted bubble contribution, evaluating as much as possible, as early as possible *)
CalculateRawBubbleContributionImpl[cornernames_, cornertypes_, props_,
   solution_] := -I Product[
```

```
   CalculateLoopCornerAmplitude[cornernames[[i]],
    cornertypes[[i]], {props[[i]],
     props[[Mod[i, 2] + 1]]}, {solution[[i]],
     solution[[Mod[i, 2] + 1]]}], {i, 2}]

(* Calculate the t to use in a triangle from the bubble's t and y *)
CalculateTriTFromBubble[momconf_, k1t_, S1o\[Chi]_, triname_,
  tricorner_Integer, h_Integer, -1, t_, y_] := 0
CalculateTriTFromBubble[momconf_, k1t_, S1o\[Chi]_, triname_,
  tricorner_Integer, 1, power_, t_, y_] :=
 Module[{corners = SplitCorners[MomConf[momconf], triname], tk1, tk3},
   tk1 = Plus @@ ToSp[Flatten[corners[[tricorner]]]];
  tk3 = Plus @@ ToSp[Flatten[corners[[Mod[tricorner + 1, 3] + 1]]]];
  WithSpinors[(t Spaa[k1t, tK3t] -
       S1o\[Chi] (1 - y) Spaa[tK3t, \[Chi]]) (t Spbb[\[Chi], tk1] -
        y Spbb[tk1, k1t])/(t s[tk1, tK3t]) // NN, {tK3t,
    Num4V[tk3] - NN[MP2[tk3]/2 /MP[tk1, tk3]] Num4V[tk1]}]]
CalculateTriTFromBubble[momconf_, k1t_, S1o\[Chi]_, triname_,
  tricorner_Integer, -1, power_, t_, y_] :=
 Module[{corners = SplitCorners[MomConf[momconf], triname], tk1, tk3},
   tk1 = Plus @@ ToSp[Flatten[corners[[tricorner]]]];
  tk3 = Plus @@ ToSp[Flatten[corners[[Mod[tricorner + 1, 3] + 1]]]];
  WithSpinors[(t Spaa[k1t, tk1] -
       S1o\[Chi] (1 - y) Spaa[tk1, \[Chi]]) (t Spbb[\[Chi], tK3t] -
        y*Spbb[tK3t, k1t])/(t s[tk1, tK3t]) // NN, {tK3t,
    Num4V[tk3] - NN[MP2[tk3]/2 /MP[tk1, tk3]] Num4V[tk1]}]]
CalculateTriTFromBubble[momconf_, k1t_, S1o\[Chi]_, triname_, None,
  h_, 1, t_, y_] :=
 Module[{corners = SplitCorners[MomConf[momconf], triname], tk1, tk3,
   tK1, tK3, S1, S3, \[Gamma]},
  tk1 = Plus @@ ToSp[Flatten[corners[[1]]]];
  tk3 = Plus @@ ToSp[Flatten[corners[[3]]]]; tK1 = Num4V[tk1];
  tK3 = Num4V[tk3]; S1 = NN[MP2[tk1]];
  S3 = NN[MP2[tk3]]; \[Gamma] =
   NN[MP[tk1, tk3]] + h Sqrt[NN[MP[tk1, tk3]]^2 - S1 S3];
  WithSpinors[(t Spaa[k1t, tK3t] -
       S1o\[Chi] (1 - y) Spaa[tK3t, \[Chi]]) (t Spbb[\[Chi], tK1t] -
        y Spbb[tK1t, k1t])/(t s[tK1t, tK3t]) // NN, {tK1t,
    tK1 - S1 tK3/\[Gamma]}, {tK3t, tK3 - S3 tK1/\[Gamma]}]]
CalculateTriTFromBubble[momconf_, k1t_, S1o\[Chi]_, triname_, None,
  h_, -1, t_, y_] :=
 Module[{corners = SplitCorners[MomConf[momconf], triname], tk1, tk3,
```

```
  tK1, tK3, S1, S3, \[CapitalDelta], \[Gamma]},
 tk1 = Plus @@ ToSp[Flatten[corners[[1]]]];
 tk3 = Plus @@ ToSp[Flatten[corners[[3]]]]; tK1 = Num4V[tk1];
 tK3 = Num4V[tk3]; S1 = NN[MP2[tk1]];
 S3 = NN[MP2[tk3]]; \[CapitalDelta] =
  MP[tK1, tK3]^2 - S1 S3 // NN; \[Gamma] =
  NN[MP[tk1, tk3]] + h Sqrt[\[CapitalDelta]];
 WithSpinors[-16 \[CapitalDelta]^2 (t Spaa[k1t, tK1t] -
      S1o\[Chi] (1 - y) Spaa[tK1t, \[Chi]]) (t Spbb[\[Chi], tK3t] -
       y Spbb[tK3t, k1t])/(t S1 S3 (S1 + \[Gamma]) (S3 + \[Gamma]) s[
        tK1t, tK3t]) // NN, {tK1t, tK1 - S1 tK3/\[Gamma]}, {tK3t,
    tK3 - S3 tK1/\[Gamma]}]]

(* Calculate the bubble subtraction contributions, evaluating as much as
      possible, as early as possible *)
CalculateBubbleSubtractionEqnImpl[process_, momconf_, split_,
  props_] :=
 Module[{cornernames = SplitCorners[MomConf[momconf], split], K1,
   S1o\[Chi], ypoles, res, tts, ttsy, tripropss, triconts},
  K1 = Plus @@ ToSp[cornernames[[1]] // Flatten];
  S1o\[Chi] = MP2[K1]/2/MP[K1, \[Chi]] // NN;
  WithSpinors[
   Do[Module[{triname = SplitName[split, tri],
       insindex = InsertionIndex[split[[1]], tri[[1]], tri[[2]]], K2,
      S2, \[Chi]K2k1t, a, at, bt, c, sqrtterm},
     K2 =
      Plus @@ ToSp[
        Flatten[If[insindex == 0,
          RotateLeft[SplitCorners[MomConf[momconf], triname], 1][[
           1 ;; insindex + 3 - 1]],
          SplitCorners[MomConf[momconf], triname][[1 ;; insindex]]]]];
      S2 = MP2[K2] // NN;
     c = 1/(S1o\[Chi]*Spab[\[Chi], K2, bk1t]) // NN;
     bt = c*MP[K1, K2] - (2*MP[K2, \[Chi]])/Spab[\[Chi], K2, bk1t] //
        NN; a = c*MP[K2, K2] - 2 MP[K2, \[Chi]]/Spab[\[Chi], K2, bk1t] //
         NN; at = -c*Spab[bk1t, K2, \[Chi]] // NN;
     ypoles[tri, t_, h_] :=
      Evaluate[(1/2 + t bt) + h*Sqrt[(1/2 + t bt)^2 - t (a + t at)]];
     res[tri, t_] :=
      Evaluate[c t/Sqrt[(1/2 + t bt)^2 - t (a + t at)]];
     ttsy[tri, corner_, h_, trih_,
       power_] := (ttsy[tri, corner, h, trih, power] =
```

```
          Function[{t, y},
           Evaluate[
            CalculateTriTFromBubble[momconf, bk1t, S1o\[Chi], triname,
             corner, trih, power, t, y]]]);
      tts[tri, corner_, h_, trih_,
        power_] := (tts[tri, corner, h, trih, power] =
          Function[{t},
           Evaluate[
            ttsy[tri, corner, h, trih, power][t, ypoles[tri, t, h]]]]]);
      tricont s[tri, tricorner_, h_, trih_, triprops_, ttp_, ttm_] :=

            CalculateTriContribution[process, momconf, triname,
          triprops, trih, 0] +
         Sum[ttp^i CalculateTriContribution[process, momconf, triname,
             triprops, trih, i] +
           ttm^i CalculateTriContribution[process, momconf, triname,
             triprops, trih, -i], {i, 1, pmp[process] - 1}];
      tripropss[tri] =
       RemoveIgnorableOptions[process, triname,
        SplitPropOptions[process, split, props, tri]]], {tri,
      SplitOptions[process, split]}], {bk1t,
    Num4V[K1] - S1o\[Chi] Num4V[\[Chi]]}];
  Function[{t,
    y}, -WithSpinors[
     Sum[Module[{triname = SplitName[split, tri],
        insindex = InsertionIndex[split[[1]], tri[[1]], tri[[2]]]},
       res[tri, t] Sum[
          Module[{tricorner =
             MasslessTriCorner[process, triname, triprops], trihs},
           trihs[h_] :=
            Evaluate[
             If[MatchQ[tricorner, _Integer],
              If[(Abs[tts[tri, tricorner, -1, -1, 1][t]] -
                   Abs[tts[tri, tricorner, -1, 1, 1][t]])/(Abs[
                    tts[tri, tricorner, -1, -1, 1][t]] +
                   Abs[tts[tri, tricorner, -1, 1, 1][t]]) <
                 0, {-h}, {h}], {1, -1}]];
           Sum[Sum[CalculateTriSubtraction[process, momconf, triname,
                tricorner, trih, triprops,
                tts[tri, tricorner, h, trih, 1][
                 t]] h/(y - ypoles[tri, t, h]) +
              tricont s[tri, tricorner, h, trih, triprops,
```

```
                    ttsy[tri, tricorner, h, trih, 1][t, y],
                    ttsy[tri, tricorner, h, trih, -1][t,
                     y]] (h/(y - ypoles[tri, t, h]) -
                     h/(y - ypoles[tri, t, -h])) +
                  triconts[tri, tricorner, h, trih, triprops,

                    ttsy[tri, tricorner, h, trih, 1][t,
                     ypoles[tri, t, -h]],
                    ttsy[tri, tricorner, h, trih, -1][t,
                     ypoles[tri, t, -h]]] h/(y -
                      ypoles[tri, t, -h]), {trih, trihs[h]}]/
               Length[trihs[h]], {h, {-1, 1}}]], {triprops,
             tripropss[tri]}]], {tri,
         SplitOptions[process, split]}], {bk1t,
       Num4V[K1] - S1o\[Chi] Num4V[\[Chi]]}]]]

CalculateBubbleSubtractionEqn[process_,
  momconf_, {split_List, nonord_List}, props_] :=
 CalculateBubbleSubtractionEqn[process, momconf, {split, nonord},
   props] =
  CalculateBubbleSubtractionEqnImpl[process, momconf, {split, nonord},
    props]

CalculateBubbleSubtraction[process_,
  momconf_, {split_List, nonord_List}, props_List, t_, y_] :=
 CalculateBubbleSubtractionEqn[process, momconf, {split, nonord},
   props][t, y]

(* Calculate the raw but subtracted bubble contribution, evaluating as
    much as possible, as early as possible *)
CalculateRawBubbleContribution[process_,
  momconf_, {split_List, nonord_List}, props_, t_, y_] :=
 Module[{cornertypes =
    SplitCorners[Process[process], {split, nonord}],
   cornernames = SplitCorners[MomConf[momconf], {split, nonord}]},
  CalculateRawBubbleContributionImpl[cornernames, cornertypes, props,
    CalcBubbleLoopMom[momconf, {split, nonord}, t, y]] -
   CalculateBubbleSubtraction[process, momconf, {split, nonord},
    props, t, y]]

(* The coefficients and values for y needed to extract the correct
    combinations for different maximum powers of y *)
```

```
GetyExtraction[1 | 2] := {{1, 1/2}}
GetyExtraction[
  3 | 4] := {{1/2, (3 - Sqrt[3])/6}, {1/2, (3 + Sqrt[3])/6}}
GetyExtraction[
  5 | 6] := {{4/9,
   1/2}, {5/18, (5 - Sqrt[15])/10}, {5/18, (5 + Sqrt[15])/10}}
GetyExtraction[
  7 | 8] := {{(18 + Sqrt[30])/72, (35 - Sqrt[525 - 70 Sqrt[30]])/
     70}, {(18 + Sqrt[30])/72, (35 + Sqrt[525 - 70 Sqrt[30]])/
     70}, {(18 - Sqrt[30])/72, (35 - Sqrt[525 + 70 Sqrt[30]])/
     70}, {(18 - Sqrt[30])/72, (35 + Sqrt[525 + 70 Sqrt[30]])/70}}
GetyExtraction[
  9 | 10] := {{64/225,
   1/2}, {(322 + 13 Sqrt[70])/1800, (21 - Sqrt[245 - 14 Sqrt[70]])/
     42}, {(322 + 13 Sqrt[70])/1800, (21 + Sqrt[245 - 14 Sqrt[70]])/
     42}, {(322 - 13 Sqrt[70])/1800, (21 - Sqrt[245 + 14 Sqrt[70]])/
     42}, {(322 - 13 Sqrt[70])/1800, (21 + Sqrt[245 + 14 Sqrt[70]])/42}}

(* Calculate the needed bubble contribution *)
CalculateBubbleContribution[process_,
  momconf_, {split_List, nonord_List}, props_] :=
 CalculateBubbleContribution[process, momconf, {split, nonord},
   props] =
  Module[{p = pp[process], t0 = pt0[process], yex},
   yex = GetyExtraction[p];
   Sum[yex[[k, 1]] CalculateRawBubbleContribution[process,
        momconf, {split, nonord}, props, t0*Exp[2 Pi I j/(2 p + 1)],
        yex[[k, 2]]], {k, Length[yex]}, {j, -p, p}]/(2 p + 1) //
    Factor]
```

B.2 Comparing Mathematica Implementation and BlackHat

Listing B.4: Test.txt

```
SetAttributes[PrintTiming, HoldFirst]
PrintTiming[expr_] := Module[{tmp = AbsoluteTiming[expr]},
  Print[tmp[[1]], " to evalute ", HoldForm[expr]]; tmp[[2]]]

FailedTests = {};
TestIndex = 0;

```

```
(* Run a test, show its result and if it failed record for later *)
SetAttributes[TestEqual, HoldFirst]
TestEqual[expr_, value_] := Module[{i, result},
  i = TestIndex = TestIndex + 1; Print["Starting Test ", i,
    ": "*Defer[expr], "=", value]; result = PrintTiming[expr];
   If[TrueQ[result == value], Print["Test succeeded"],
    Print["Test ", i, " Failed: ", result, "=/=", value];
     FailedTests = Append[FailedTests, i]]; ]
TestEqual[expr_, value_, error_] := Module[{i, result},
  i = TestIndex = TestIndex + 1; Print["Starting Test ", i,
    ": "*Defer[expr], "=", value]; result = PrintTiming[expr];
   If[TrueQ[If[TrueQ[value == 0], Abs[result], (Abs[result - value]*2)/
       (Abs[result] + Abs[value])] < error], Print["Test succeeded"],
    Print["Test ", i, " Failed: ", result, "=/=", value];
     FailedTests = Append[FailedTests, i]]; ]

(* Show any tests that have failed *)
ShowFailedTests[] := If[Length[FailedTests] == 0,
  Print["No Tests Failed"], Print["Test(s) ",
   Sequence @@ Riffle[FailedTests, ", "], " Failed"]]
```

Listing B.5: Rules.txt

```
epsilon = 10^-7;

(* Compare two sets of rules recursively. Rules match if the values for
    the same keys match within tolerance except for keys that are only
    in one set of rules whose values must all be zero within tolerance
    *)
CompareRules[keys_, math : _Real | _Complex | _Integer,
  bh : _Real | _Complex | _Integer] :=
 If[TrueQ[Abs[math - bh]*2/(0.1 + Abs[math] + Abs[bh]) < epsilon],
  True, Print["Difference in valid values for ", keys, " ", math,
   "!=", bh]; False]
CompareRules[keys_, math : {}, bh : _Real | _Complex | _Integer] :=
 If[TrueQ[Abs[bh] < epsilon], True,
  Print["bh value not 0 and math missing for ", keys, " ", bh];
  False]
CompareRules[keys_, math : _Real | _Complex | _Integer, bh : {}] :=
 If[TrueQ[Abs[math] < epsilon], True,
  Print["math value not 0 and bh missing for ", keys, " ", math];
  False]
CompareRules[keys_, math_List, bh_List] :=
```

```
 Module[{result = True},
  If[Not[And @@ (CompareRules[Append[keys, #], # /. math, # /. bh] & /@
         Intersection[First /@ math, First /@ bh])], result = False;
   Print["error in common elements for ", keys]];
  If[Not[And @@ (CompareRules[Append[keys, #], # /. math, {}] & /@
       Complement[First /@ math, First /@ bh])], result = False;
   Print["error in elements only in math ", keys]];
  If[Not[And @@ (CompareRules[Append[keys, #], {}, # /. bh] & /@
       Complement[First /@ bh, First /@ math])], result = False;
   Print["error in elements only in bh ", keys]]; result]

(* Generate a set of nested rules with split as the outer key and
    propagators as the inner key that map to the value of the
    corresponding coefficient *)
GenerateMathRules[process_, momconf_, n_, genfunc_] :=
 Module[{tree = HelAmplN[MomConf[momconf], Process[process]]},
  Function[split,
    split -> ((# :>
          genfunc[process, momconf, split, #, None]/tree) & /@
       RemoveIgnorableOptions[process, split,
        SplitPropOptions[process, split]])] /@
   SplitOptions[process, n]]
GenerateMathRules[process_, momconf_, n : 4] :=
 GenerateMathRules[process, momconf, n, CalculateBoxContribution]
GenerateMathRules[process_, momconf_, n : 3] :=
 GenerateMathRules[process, momconf, n, CalculateTriContribution]
GenerateMathRules[process_, momconf_, n : 2] :=
 GenerateMathRules[process, momconf, n, CalculateBubbleContribution]

$UseBHSigns = False;
```

Listing B.6: rambo.py by Daniel Maitre

```
""" Generate random sets of on-shell momenta that conserve momenta """

import random
import math
from numpy import array
random.seed(1234)

def doublePi():
    return math.pi

```

```python
math.get_pi=doublePi

def dot(a,b):
    return sum([a[i]*b[i] for i in range(len(a))],type(a[0])(0) )

def getRandomQ(RandomGenerator=random.random,Type=float,mathLib=math):
    c=Type(2)*RandomGenerator()-Type(1)
    phi=Type(2)*mathLib.get_pi()*RandomGenerator()
    q0=-mathLib.log(RandomGenerator()*RandomGenerator())
    qx=q0*mathLib.sqrt(Type(1)-c*c)*mathLib.cos(phi)
    qy=q0*mathLib.sqrt(Type(1)-c*c)*mathLib.sin(phi)
    qz=q0*c
    return (q0,qx,qy,qz)

def boost(q,x,gamma,b):
    Type=type(x)
    p0=x*(gamma*q[0]+dot(b,q[1:]))
    p= array(q[1:])
    p+= b*q[0]
    f=Type(1)/(Type(1)+gamma)
    f*=dot(b,q[1:])
    p+=b*f
    p*=x
    return (p0,)+tuple(p)

def finalStatePS(w,n,Type=float,mathLib=math,**kargs):
    qs = [ getRandomQ(Type=Type,mathLib=mathLib,**kargs) for i in range(
    n)]
    Q = array([ sum([q[j] for q in qs ],Type(0)) for j in range(4)])
    M = mathLib.sqrt(Q[0]*Q[0]-dot(Q[1:],Q[1:]))
    b = array( [-q/M for q in Q[1:] ])
    x = Type(w)/M
    gamma = Q[0]/M
    ps=[ boost(q,x,gamma,b) for q in qs ]
    return ps

def PS(n,Type=float,mathLib=math,RandomGenerator=random.random,**kargs):
    ctheta=Type(2)*RandomGenerator()-Type(1)
    stheta=mathLib.sqrt(Type(1)-ctheta*ctheta)
    phi=Type(2)*RandomGenerator()*mathLib.get_pi()
    sphi=mathLib.sin(phi)
    cphi=mathLib.cos(phi)
```

```
    w=Type(n)
    E=w/Type(2)
    c=mathLib
    p1=(-E,
        -E*stheta*sphi,
        -E*stheta*cphi,
        -E*ctheta
    )
    p2 = ( -E , -p1[1], -p1[2], -p1[3])
    ps=finalStatePS(n,n-2,RandomGenerator=RandomGenerator,Type=Type,
    mathLib=mathLib,**kargs)
    return [ p1,p2] + ps
```

Listing B.7: BHtools.py based on code by Daniel Maitre

```
""" Load BlackHat and declare a few functions to map between strings and
    BlackHat types """
## uncomment this and insert the correct path to find the
## blackhat python library if it is not already in the
## module import path
#import sys
#sys.path.append('/path/to/blackhat/python/library')
import BH

import re
import itertools

import rambo

def getRandomMC(n):
    ps =rambo.PS(n)
    cms = [ BH.Cmomd(*p) for p in ps ]
    return BH.mcd(*cms)

stringToParticleMap={
    'm' : BH.cvar.m ,
    'p' : BH.cvar.p ,
    'qm' : BH.cvar.qm ,
    'qp' : BH.cvar.qp ,
    'Qm' : BH.cvar.q2m ,
    'Qp' : BH.cvar.q2p ,
    'qbm' : BH.cvar.qbm ,
    'qbp' : BH.cvar.qbp ,
```

```
    'Qbm' : BH.cvar.qb2m ,
    'Qbp' : BH.cvar.qb2p ,
    'ym' : BH.cvar.ym ,
    'yp' : BH.cvar.yp ,
    'lp' : BH.cvar.lp ,
    'lm' : BH.cvar.lm ,
    'lbp' : BH.cvar.lbp,
    'lbm' : BH.cvar.lbm,
    'Um' : BH.cvar.Qm ,
    'Up' : BH.cvar.Qp ,
    'Ubm' : BH.cvar.Qbm ,
    'Ubp' : BH.cvar.Qbp ,
    'ph': BH.cvar.ph ,
    'phd': BH.cvar.phd ,
    'H': BH.cvar.H ,
    'q0m' : BH.cvar.qm ,
    'q0p' : BH.cvar.qp ,
    'qb0m' : BH.cvar.qbm ,
    'qb0p' : BH.cvar.qbp ,
    'q1m' : BH.cvar.q2m ,
    'q1p' : BH.cvar.q2p ,
    'qb1m' : BH.cvar.qb2m ,
    'qb1p' : BH.cvar.qb2p ,
    'q2m' : BH.cvar.q3m ,
    'q2p' : BH.cvar.q3p ,
    'qb2m' : BH.cvar.qb3m ,
    'qb2p' : BH.cvar.qb3p ,
    'q3m' : BH.cvar.q4m ,
    'q3p' : BH.cvar.q4p ,
    'qb3m' : BH.cvar.qb4m ,
    'qb3p' : BH.cvar.qb4p
}

def stringToParticles(st):
    ps=st.split(' ')
    return [ stringToParticleMap[p] for p in ps ]

def stringToProcess(st):
    return BH.process(BH.vectorpID(stringToParticles(st)))
```

Listing B.8: BHMathLink.py

```python
""" Functions to convert between BlackHat and Mathematica implementation representations of various objects """
import BHtools as BHT
import BH
from math import log10,ceil
import numpy as np
from collections import defaultdict

def genmc(pro):
  """ Generate a set of momenta for the given process taking account of massive particles """
  nparticles=len(pro)
  imassive=[i for i in range(nparticles) if pro[i].mass()!=0]
  nmassive=len(imassive)
  imassless=[i for i in range(nparticles) if pro[i].mass()==0]
  nmassless=len(imassless)
  n=nparticles+nmassive #nmassless+2*nmassive
  mc=BHT.getRandomMC(n)
  inds=np.zeros((nparticles),dtype=int)
  inds[imassless]=range(1,nmassless+1)
  inds[imassive]=[mc.Sum(i,i+1) for i in range(nparticles-nmassive+1,n+1,2)]
  return mc,[int(i) for i in inds]

def dasmath(d):
  """ Write a double in a form that can be read as Mathematica input """
  if d==0:
    return "0"
  exp=int(log10(abs(d)))
  return str(d/10**exp)+"*^"+str(exp)
def casmath(c):
  """ Write a complex number in a form that can be read as Mathematica input """
  return dasmath(c.real)+"+I*"+dasmath(c.imag)

def makequarkmap(cut,n):
  """
  Calculate a map to correct quark flavours to Mathematica versions.

  In corners of a cut, BlackHat relabels quarks to ensure that if the same quark line traverses a corner twice, the correct pairs of quarks will connect up. The Mathematica implementation does this at
```

```
      the last possible step before evaluating and therefore expects
      quarks to be labelled using their external flavours
    """
    quarkmap={}
    cutquarks=defaultdict(set)
    for i in range(n):
      for j in range(cut.corner_size(i+1)):
        ind=cut.corner_ind(i+1,j+1)
        externpart=cut.extern_process().p(ind)
        internpart=cut.get_process(i+1).p(j+2)
        if externpart.is_a(BH.cvar.quark):
          # if the particle is a quark then correct from the
          # type used in the corner to the type used in the
          # external process
          quarkmap[internpart.flavor()]=externpart.flavor()
      cutpart=cut.get_process(i+1).front()
      # map cut quarks from the type in one corner to the
      # type used in the previous corner
      if cutpart.is_a(BH.cvar.quark):
        otherpart=cut.get_process((i-1)%n+1).back()
        cutquarks[cutpart.flavor()].add(otherpart.flavor())
        cutquarks[otherpart.flavor()].add(cutpart.flavor())
    while cutquarks:
      # if the quark is not found then there must be a closed quark loop
      value=-1

      # start from a cut quark and repeatedly search for quarks
      # that are equivalent and also check if any match an
      # external quark
      quarkset=[cutquarks.keys()[0]]
      newquarks=quarkset[:]
      while newquarks:
        quark=newquarks.pop()
        if quark in quarkmap:
          value=quarkmap[quark]
        for otherquark in cutquarks.pop(quark):
          if otherquark not in quarkset:
            quarkset.append(otherquark)
            newquarks.append(otherquark)
      for quark in quarkset:
        quarkmap[quark]=value
    return quarkmap
```

```
_partToMathMap={
    BH.cvar.m:"Gluon[-1]" ,
    BH.cvar.p:"Gluon[1]" ,
    BH.cvar.ph:"Phi[1]" ,
    BH.cvar.phd:"Phi[-1]" ,
    BH.cvar.H:"Higgs[]"
}

def partToMathMap(p,quarkmap={}):
  """ Convert a BlackHat particle to its Mathematica type name """
  if p.is_a(BH.cvar.quark):
    return "Quark["+("-1" if p.is_anti_particle() else "1")+","+str(p.helicity())+","+str(quarkmap.get(p.flavor(),p.flavor()))+"]"
  else:
    return _partToMathMap[p]


def partname(p,i):
  """ Generate a name for a particle compatible with S@M """
  if p.mass()==0:
    return str(i)
  else:
    return "P"+str(i)


def mcmathprint(pro,mc,inds):
  """ Convert the momentum configuration to the code to declare the corresponding S@M momenta """
  s=""
  for p,i in zip(pro.particle_IDs(),inds):
    if p.mass()==0:
      s=s+"DeclareSpinorMomentum["+str(i)+",{"+casmath(mc.p(i).E())+","+casmath(mc.p(i).X())+","+casmath(mc.p(i).Y())+","+casmath(mc.p(i).Z())+"}]\n"
    else:
      s=s+"DeclareLVectorMomentum[P"+str(i)+",{"+casmath(mc.p(i).E())+","+casmath(mc.p(i).X())+","+casmath(mc.p(i).Y())+","+casmath(mc.p(i).Z())+"}]\n"
  return s

def generate_label_maps(pro):
```

```
  """ Generate a pair of maps containing information needed to map cuts from BlackHat to their Mathematica names """
  ordered_label_map={}
  unordered_label_map={}
  i=0
  j=0
  for k in range(len(pro)):
    if pro[k].get_ordered() == 0:
      j=j+1
      ordered_label_map[k+1]=j
    else:
      i=i+1
      unordered_label_map[k+1]=i
  return ordered_label_map, unordered_label_map

def get_label_indexes((ordered_map,unordered_map)):
  """ Generate lists containing the indexes of the ordered and unordered particles in the BlackHat process """
  ordered_indexes=[-1]*len(ordered_map)
  unordered_indexes=[-1]*len(unordered_map)
  for k,v in ordered_map.items():
    ordered_indexes[v-1]=k
  for k,v in unordered_map.items():
    unordered_indexes[v-1]=k
  return ordered_indexes,unordered_indexes

def calculate_raw_code(c,(ordered_map,unordered_map)):
  """ Calculate the split code from a BlackHat cut, in a form that matches the Mathematica implementation """
  cut_labels=[]
  unordered_labels=[-1]*len(unordered_map)
  skipped = 0
  for i in range(c.nbr_props()):
    corner = [c.corner_ind(i+1,j+1) for j in range(c.corner_size(i+1))]
    for ind in [ind for ind in corner if ind in unordered_map]:
      unordered_labels[unordered_map[ind]-1]=i+1
      corner.remove(ind)
    if len(corner) == 0:
      skipped = skipped+1
    else:
      cut_labels.extend([ordered_map[corner[0]]]*(skipped+1))
      skipped=0
```

```
  cut_labels.extend([cut_labels[0]]*skipped)
  shifted = 0;
  target = min(cut_labels)
  while cut_labels[0] != target or cut_labels[-1] == target:
    shifted = shifted+1
    cut_labels.append(cut_labels.pop(0))
  if shifted != 0:
    unordered_labels = [((i+len(cut_labels)-shifted-1) % len(cut_labels)) + 1 for i in unordered_labels]
  return cut_labels,unordered_labels,shifted

def calculate_mycode(c,maps):
  """ Calculate the split code as a string from a BlackHat cut, in a form that matches the Mathematica implementation """
  cut_labels,unordered_labels,shifted=calculate_raw_code(c,maps)
  cuts = "".join(str(el) for el in cut_labels)
  if len(unordered_labels) != 0:
    cuts += "-"+"".join(str(el) for el in unordered_labels)
  return cuts,shifted

def calculate_mathsplit(cut,maps):
  """ Calculate the Mathematica code for the label for a split """
  cut_labels,unordered_labels,shifted=calculate_raw_code(cut,maps)
  return "{{"+",".join(str(s) for s in cut_labels)+"},{"+",".join(str(s) for s in unordered_labels)+"}}",shifted

def sortedcuts(cutfunction,cutcount,maps):
  """
  Sort the cuts into a consistent order.

  Returns a list of tuples containing the BlackHat index of the cut and the cut itself
  """
  cuts=[(i+1,cutfunction(i+1)) for i in range(cutcount)]
  def key((i,cut)):
    return (calculate_mycode(cut,maps),)+tuple(cut.l(i+1).conjugate() for i in range(cut.size()))
  cuts.sort(key=key)
  return cuts

def getprops(cut,n,shift):
```

```
    """ Return the propagators for a cut in the order needed for the
      Mathematica implementation """
    return tuple(p.p(p.n()) for p in (cut.get_process((i+shift-1)%n+1) for
      i in range(n)))

# Dictionaries to map colour structures to strings and back
colourstructToBH={
  "glue":BH.glue,
  "nf":BH.nf,
  "LT":BH.LT,
  "RT":BH.RT,
  "LLT":BH.LLT,
  "LRT":BH.LRT,
  "RLT":BH.RLT,
  "RRT":BH.RRT}

colourstructureFromBH={v:k for k,v in colourstructToBH.iteritems()}

# Dictionary to map colour structures to the Mathematica code for them
colourstructureBHtoMath={
  BH.glue:"{}",
  BH.nf:"{Quark[1,-1]}",
  BH.LT:"{Quark[-1,1]}",
  BH.RT:"{Quark[1,1]}",
  BH.LLT:"{Quark[-1,1],Quark[-1,2]}",
  BH.LRT:"{Quark[-1,1],Quark[1,2]}",
  BH.RLT:"{Quark[1,1],Quark[-1,2]}",
  BH.RRT:"{Quark[1,1],Quark[1,2]}"}
```

Listing B.9: TreeTests.py

```
#!/usr/bin/python

import BHtools as BHT
import BH
from BHMathlink import *
import sys
import os
import random
random.seed()
from itertools import combinations
import numpy as np

```

```
def calctest(pro,mc,inds):
  ep=BH.ep(mc,inds)
  A=BH.TreeHelAmpl(pro)
  return A.eval(ep)
def mathcode(pro,mc,inds,val):
  s=mcmathprint(pro,mc,inds)
  s=s+"\n"
  s=s+"TestEqual[HelAmplN[{"+",".join(partname(p,i) for p,i in zip(pro.
    particle_IDs(),inds))+"},{"+",".join(partToMathMap(p) for p in pro.
    particle_IDs())+"}],"+casmath(val)+",1*^-9]\n"
  return s
def runtest(pro):
  mc,inds=genmc(pro)
  val=calctest(pro,mc,inds)
  return mathcode(pro,mc,inds,val)

def allcombs(l):
  for i in range(0,len(l)+1):
    for c in combinations(l,i):
      yield c

def shuffle(els,counts):
  if sum(counts)==0:
    yield ()
  else:
    for i in range(len(els)):
      counts[i]-=1
      for s in shuffle(els,counts):
        yield (el[i],)+s
      counts[i]+=1

def noshuffle(els,counts):
  return [tuple(e for e,c in zip(els,counts) for i in range(c))]

shuffle=noshuffle

def rungluetests():
  s=""
  for n in range(4,10):
    for nneg in range(n+1):
      for shuf in shuffle(("p","m"),(n-nneg,nneg)):
        s=s+runtest(BHT.stringToProcess(" ".join(shuf)))+"\n\n"
```

```
  return s

def runphigluetests():
  s=""
  for n in range(4,10):
    for nneg in range(n+1):
      for shuf in shuffle(("p","m"),(n-nneg,nneg)):
        s=s+runtest(BHT.stringToProcess("ph "+" ".join(shuf)))+"\n\n"
  return s

def quarkoptions(nquarks,i=0):
  if nquarks==0:
    yield []
  else:
    qo=[("qp","qbm"),("qm","qbp"),("qbp","qm"),("qbm","qp")]
    for option in quarkoptions(nquarks-1,i+1):
      for o in qo:
        yield [tuple(q[:-1]+str(i)+q[-1] for q in o)]+option

def insertquarks(others,quarks):
  others=tuple(others)
  if len(quarks)==0:
    yield others
  else:
    for i in range(len(others)+1):
      for rest in insertquarks(others[i:],quarks[1:]):
        yield others[:i]+quarks[0]+rest

def pad(it,el):
  for e in it:
    yield e
  while True:
    yield el

def grouper(l,n):
  for i in range(0,len(l),n):
    yield l[i:i+n]

def insertquarksone(others,quarks):
  others=tuple(others)
  per=max(len(others)/len(quarks),1)
  l=[e for o,q in zip(pad(grouper(others,per),()),quarks) for e in q+o]
```

```
    return [tuple(l)+others[len(l):]]

insertquarks=insertquarksone

def runquarkgluetests():
  s=""
  for N in range(4,10):
    for nneg in range(N-2,-1,-1):
      for nquarks in range(1,min((N-nneg)/2,3)+1):
        n=N-nquarks*2
        for quarks in quarkoptions(nquarks):
          for shuf in shuffle(("p","m"),(n-nneg,nneg)):
            for withq in insertquarks(shuf,quarks):
              print withq;
              s=s+runtest(BHT.stringToProcess(" ".join(withq)))+"\n\n"
  return s

def runphiquarkgluetests():
  s=""
  for N in range(4,10):
    for nneg in range(N-2,-1,-1):
      for nquarks in range(1,min((N-nneg)/2,3)+1):
        n=N-nquarks*2
        for quarks in quarkoptions(nquarks):
          for shuf in shuffle(("p","m"),(n-nneg,nneg)):
            for withq in insertquarks(shuf,quarks):
              print withq;
              s=s+runtest(BHT.stringToProcess("ph "+" ".join(withq)))+"\n\n"
  return s

def mainglue():
  from subprocess import Popen,PIPE

  p = Popen(["math8","-noprompt"], cwd=os.path.dirname(os.path.realpath(__file__)), stdin=PIPE)
  pipe = p.stdin
  pipe.write("SetOptions[#,FormatType->OutputForm]&/@Streams[]\n")

  pipe.write("<<\"Common.txt\"\n")
  pipe.write("<<\"HelAmplN.txt\"\n")
  pipe.write("<<\"Test.txt\"\n")
```

```

  pipe.write(rungluetests())

  pipe.write("ShowFailedTests[]\n")
  pipe.write("Exit[]")
  pipe.close()
  p.wait()

def mainphiglue():
  from subprocess import Popen,PIPE

  p = Popen(["math8","-noprompt"], cwd=os.path.dirname(os.path.realpath(__file__)), stdin=PIPE)
  pipe = p.stdin
  pipe.write("SetOptions[#,FormatType->OutputForm]&/@Streams[]\n")

  pipe.write("<<\"Common.txt\"\n")
  pipe.write("<<\"HelAmplN.txt\"\n")
  pipe.write("<<\"Test.txt\"\n")

  pipe.write(runphigluetests())

  pipe.write("ShowFailedTests[]\n")
  pipe.write("Exit[]")
  pipe.close()
  p.wait()

def mainquarkglue():
  from subprocess import Popen,PIPE

  p = Popen(["math8","-noprompt"], cwd=os.path.dirname(os.path.realpath(__file__)), stdin=PIPE)
  pipe = p.stdin
  pipe.write("SetOptions[#,FormatType->OutputForm]&/@Streams[]\n")

  pipe.write("<<\"Common.txt\"\n")
  pipe.write("<<\"HelAmplN.txt\"\n")
  pipe.write("<<\"Test.txt\"\n")

  pipe.write(runquarkgluetests())

  pipe.write("ShowFailedTests[]\n")
```

```
  pipe.write("Exit[]")
  pipe.close()
  p.wait()

def mainphiquarkglue():
  from subprocess import Popen,PIPE

  p = Popen(["math8","-noprompt"], cwd=os.path.dirname(os.path.realpath(
    __file__)), stdin=PIPE)
  pipe = p.stdin
  pipe.write("SetOptions[#,FormatType->OutputForm]&/@Streams[]\n")

  pipe.write("<<\"Common.txt\"\n")
  pipe.write("<<\"HelAmplN.txt\"\n")
  pipe.write("<<\"Test.txt\"\n")

  pipe.write(runphiquarkgluetests())

  pipe.write("ShowFailedTests[]\n")
  pipe.write("Exit[]")
  pipe.close()
  p.wait()


if __name__=="__main__":
  import sys
  name=sys.argv[1] if len(sys.argv)>=2 else "glue"
  globals()["main"+name]()
```

Listing B.10: OneLoopTests.py

```
#!/usr/bin/python
print "(*"
import sys
import os.path
import BHtools as BHT
import BH
from BHMathlink import *
from collections import defaultdict
from subprocess import Popen,PIPE

BH.use_setting("USE_KNOWN_FORMULAE no")
print "*)"
```

```
def getprocesses(cut,n):
  return [cut.get_process(i+1) for i in range(n)]

def collectcuts(cuts,ncuts,n,maps):
  splits=defaultdict(dict)
  for i in range(ncuts):
    cut=cuts(i+1)
    split,shifted = calculate_mathsplit(cut,maps)
    splits[split][getprops(cut,n,shifted)]=(makequarkmap(cut,n),cut)
  return splits

def run(PRO,mode=BH.glue,doprint=True,mathtestcode=True,plotgraph=False,
    outfile=sys.stdout):
  mc,inds=genmc(PRO)

  ep=BH.ep(mc,inds)

  print "(*"#make the next few lines output be treated by mathematica as
     a comment

  A=BH.TreeHelAmpl(PRO)

  tree=A.eval(ep)

  if doprint:
    print PRO
    print tree

  AA=BH.One_Loop_Helicity_Amplitude(PRO,mode)

  cc=AA.cut_part()
  cp=cc.makeDarrenCutPart()
  if cp==None:
    cp=cc.makeHiggsCutPart()
  if cp==None:
    print "can't convert cut part to known type"
    return
  print cp

  print "Generating Maps: "+str(PRO)

```

```
print len(PRO)

maps = generate_label_maps(PRO)

print maps

if doprint:
  for (i,c) in sortedcuts(cp.bubble,cp.nbr_bubbles(),maps):
      res=c.eval(ep)
      print "Bubble {i:{pad}} code {code}/{mycode}  [ {processes[0]} | {processes[1]} ]: {res.real:< 16.10g}{res.imag:<+16.10g}i | {norm.real:< 16.10g}{norm.imag:<+16.10g}i".format(i=i,pad=len(str(cp.nbr_bubbles())),code=c.get_code(),mycode=calculate_mycode(c,maps)[0],processes=getprocesses(c,2),res=res,norm=res/tree)


  for (i,c) in sortedcuts(cp.triangle,cp.nbr_triangles(),maps):
      res=c.eval(ep)
      print "Triangle {i:{pad}} code {code}/{mycode}  [ {processes[0]} | {processes[1]} | {processes[2]} ]: {res.real:< 16.10g}{res.imag:<+16.10g}i | {norm.real:< 16.10g}{norm.imag:<+16.10g}i".format(i=i,pad=len(str(cp.nbr_triangles())),code=c.get_code(),mycode=calculate_mycode(c,maps)[0],processes=getprocesses(c,3),res=res,norm=res/tree)

  for (i,c) in sortedcuts(cp.box,cp.nbr_boxes(),maps):
      res=c.eval(ep)
      print "Box {i:{pad}} code {code}/{mycode}  [ {processes[0]} | {processes[1]} | {processes[2]} | {processes[3]} ]: {res.real:< 16.10g}{res.imag:<+16.10g}i | {norm.real:< 16.10g}{norm.imag:<+16.10g}i".format(i=i,pad=len(str(cp.nbr_boxes())),code=c.get_code(),mycode=calculate_mycode(c,maps)[0],processes=getprocesses(c,4),res=res,norm=res/tree)


print "*)"

if mathtestcode:
  try:
    if mode==BH.nf:
      sign=-1
    else:
```

```
            sign=1

        s=mcmathprint(PRO,mc,inds)+"\n"

        ordered_indexes,unordered_indexes=get_label_indexes(maps)

        s+="process=DeclareProcess[{{"+",".join(partToMathMap(PRO[i-1]) for i in unordered_indexes)+"},{"+",".join(partToMathMap(PRO[i-1]) for i in ordered_indexes)+"}}, "+colourstructureBHtoMath[mode]+"]\n"
        s+="momconf=DeclareMomConf[{{"+",".join(partname(PRO[i-1],inds[i-1]) for i in unordered_indexes)+"},{"+",".join(partname(PRO[i-1],inds[i-1]) for i in ordered_indexes)+"}}]\n"
        s+="\n"
        s+="CompareRules[\"TREE\",Abs[HelAmplN[MomConf[momconf],Process[process]]],"+dasmath(abs(tree))+"]\n"
        s+="HelAmplN[MomConf[momconf],Process[process]]/("+casmath(tree)+")\n\n"

        if not doprint:
            # force the cuts to evaluate themselves (only first copy is apparently needed)
            for (i,c) in sortedcuts(cp.bubble,cp.nbr_bubbles(),maps):
                res=c.eval(ep)
            for (i,c) in sortedcuts(cp.triangle,cp.nbr_triangles(),maps):
                res=c.eval(ep)

        splits=collectcuts(cp.box,cp.nbr_boxes(),4,maps)
        s+="bh={"+",".join(split+"->{"+",".join("{"+", ".join(partToMathMap(p,quarkmap) for p in prop)+"}->"+casmath(sign*cut.eval(ep)/tree) for prop,(quarkmap,cut) in els.iteritems())+"}\n" for split,els in splits.iteritems())+"};\n"
        s+="CompareRules[{},GenerateMathRules[process,momconf,4],bh]\n"

        splits=collectcuts(cp.triangle,cp.nbr_triangles(),3,maps)
        s+="bh={"+",".join(split+"->{"+",".join("{"+", ".join(partToMathMap(p,quarkmap) for p in prop)+"}->"+casmath(sign*cut.eval(ep)/tree) for prop,(quarkmap,cut) in els.iteritems())+"}\n" for split,els in splits.iteritems())+"};\n"
        s+="CompareRules[{},GenerateMathRules[process,momconf,3],bh]\n"

        splits=collectcuts(cp.bubble,cp.nbr_bubbles(),2,maps)
```

```
      s+="bh={"+",".join(split+"->{"+",".join("{"+", ".join(partToMathMap(p,quarkmap) for p in prop)+"}->"+casmath(sign*cut.eval(ep)/tree) for prop,(quarkmap,cut) in els.iteritems())+"}\n" for split,els in splits.iteritems())+"};\n"
      s+="CompareRules[{},GenerateMathRules[process,momconf,2],bh]\n"


    except Exception as ex:
      print "(* ERROR *)"
      print s
      raise

    print "\n\n(********MATH CODE********)\n\n"
    if not (outfile==sys.stdout or outfile==sys.stderr):
      print s;
    print >>outfile,s,"\n\n"
    print "(********END MATH CODE********)\n"

  if plotgraph:
    path='tree_structure-'+str(PRO)+'-'+colourstructureFromBH[mode]

    import os
    os.system('mkdir -p \'%s\'' % path)
    BH.print_cut_part_graph(cp,path)
    os.system('cd \'%s\' ; make all > /dev/null' % path)

  return A,AA,cp

class MathStream(object):
  def __init__(self):
    self.p=None
  def connect(self):
    if self.p==None:
      self.p = Popen(["math8","-noprompt"], cwd=os.path.dirname(os.path.realpath(__file__)), stdin=PIPE)
      self.p.stdin.write("SetOptions[#,FormatType->OutputForm]&/@Streams[]\n")
      self.p.stdin.write("<<\"Common.txt\"\n")
      self.p.stdin.write("<<\"HelAmplN.txt\"\n")
      self.p.stdin.write("<<\"Loop-Cuts.txt\"\n")
      self.p.stdin.write("<<\"Rules.txt\"\n")
  def write(self,st):
```

```
      if self.p==None:
        self.connect()
      self.p.stdin.write(st)
    def close(self):
      if self.p!=None:
        self.p.stdin.close()
        self.p.wait()
        self.p=None

def parseargs(args,mathstream):
  if len(args) == 0:
    print "Script to run tests of the BlackHat and Mathematica
    implementations of one loop cut amplitudes."
    print
    print "Arguments are: ([OPTIONS] PROCESS|'!' COLOUR_STRUCTURE|'*')
    ..."
    print "Valid options are '--[no]print' to turn on/off printing
    output to the console"
    print "                   '--[no]mathtestcode' to turn on/off
    generating Mathematica code"
    print "                   '--[no]plotgraph' to turn on/off generating
     the graph of cuts    contributing to the amplitude."
    print "                   '--out' FILE to specify where the
    Mathematia code should be written to."
    print "                   FILE can be one of the special values '
    stdout' or 'stderr'. It can also be '/math/' to request that the
    Mathematica code be sent directly to a command line instance of
    Mathematica"

  files={"stdout":sys.stdout,"stderr":sys.stderr,"/math/":mathstream}
  currentoptions={"doprint":True,"mathtestcode":True,"plotgraph":False,"
    outfile":sys.stdout}
  longoptions={"print":{"doprint":True},
               "noprint":{"doprint":False},
               "mathtestcode":   {"mathtestcode":True},
               "nomathtestcode":{"mathtestcode":False},
               "plotgraph":   {"plotgraph":True},
               "noplotgraph":{"plotgraph":False}}
  shortoptions={"t":{"mathtestcode":True},
                "n":{"mathtestcode":False},
                "T":{"doprint":False,"mathtestcode":True,"plotgraph":
    False},
```

```
                "p":{"doprint":True},
                "q":{"doprint":False},
                "g":{"plotgraph":True}}
  changesincelast=False
  torun=[]
  args=list(args)
  while args:
    arg=args.pop(0)
    if arg[0]=="-":
      if arg[1]=="-":
        if arg[2:]=="out":
          fname=args.pop(0);
          if fname not in files:
            try:
              files[fname]=open(fname,"w")
            except IOExpetion as ex:
              print ex
          if fname in files:
            currentoptions["outfile"]=files[fname]
        else:
          currentoptions.update(longoptions[arg[2:]])
      else:
        for a in arg[1:]:
          currentoptions.update(shortoptions[a])
      changesincelast=True
    else:
      if arg!="!":
        PRO=BHT.stringToProcess(arg)
      arg=args.pop(0)
      if arg!="*":
        mode=colourstructToBH[arg]
      torun.append(((PRO,mode),dict(currentoptions)))
      changesincelast=False
  if changesincelast:
    print "(* WARNING: Trailing options what will be ignored *)"
  if len(torun)==0:
    print "(* ERROR: Nothing to run *)"
  return torun

def main(args):
  mathstream=MathStream()
  for a,kwa in parseargs(args,mathstream):
```

```
    run(*a,**kwa)
  mathstream.close()

import sys
main(sys.argv[1:])
```

Listing B.11: DrawCuts.txt

```
LaTeXValToSign[1] := "+"
LaTeXValToSign[-1] := "-"
MakeLaTeXName[Gluon[h_], name_] :=
 "g_{" <> ToString[name] <> "}^" <> LaTeXValToSign[h]
MakeLaTeXName[Phi[1], name_] := "\\phi_{" <> ToString[name] <> "}"
MakeLaTeXName[Phi[-1], name_] :=
 "\\bar\\phi_{" <> ToString[name] <> "}"
MakeLaTeXName[Quark[1, h_, f_], name_] :=
 "q_{" <> ToString[f] <> ",," <> ToString[name] <> "}^" <>
  LaTeXValToSign[h]
MakeLaTeXName[Quark[-1, h_, f_], name_] :=
 "\\bar q_{" <> ToString[f] <> ",," <> ToString[name] <> "}^" <>
  LaTeXValToSign[h]
MakeLaTeXName[_, name_] := "?_{" <> ToString[name] <> "}"
LaTeXLineType[_Phi] := "dashes"
LaTeXLineType[Quark[-1, _, _]] := "plain"
LaTeXLineType[Quark[1, _, _]] := "fermion"
LaTeXLineType[_Gluon] := "gluon"
LaTeXLineType[_] := "dots"
ReverseLine[Quark[-1, _, _]] := True;
ReverseLine[_] := False;

MakeFeynMFSide[0] := ""
MakeFeynMFSide[x_?Positive] := ",left=" <> ToString[x];
MakeFeynMFSide[x_?Negative] := ",left=" <> ToString[x];

MakeFeynMFLine[v1_, v2_, prop_, options_: "", side_: 0] :=
 MakeFeynMFLine[v2, v1, ReverseParticle[prop], options, -side] /;
  ReverseLine[prop]
MakeFeynMFLine[v1_, v2_, prop_, options_: "", side_: 0] :=
 StringJoin["\\fmf{", LaTeXLineType[prop], options,
  MakeFeynMFSide[side], "}{", v1, ",", v2, "}"]
MakeFeynMFLabels[i_, v1_, v2_, prop_] :=
 MakeFeynMFLabels[i, v2, v1, ReverseParticle[prop], "right"] /;
  ReverseLine[prop]
```

```
MakeFeynMFLabels[i_, v1_, v2_, prop_, side_: "left"] :=
 Module[{path = StringJoin["vpath",ToString[i],"(__", v1, ",__", v2, ")"]},
  Sow[StringJoin["\\fmfi{phantom,label=${",
    MakeLaTeXName[ReverseParticle[prop], "l" <> ToString[i]],
    "}$,label.side=", side, "}{subpath (0.6 length(", path,
    "),0.8 length(", path, ")) of ", path, "}"]];
  Sow[StringJoin["\\fmfi{phantom,label=${",
    MakeLaTeXName[prop, "l" <> ToString[i]], "}$,label.side=", side,
    "}{subpath (0.2 length(", path, "),0.4 length(", path, ")) of ",
    path, "}"]]]

MakeFeynMF[process_, momconf_, split_, props_] :=
 StringJoin[
  Riffle[Reap[
     Module[{vname, cornernames, cornertypes},
      cornernames = SplitCorners[MomConf[momconf], split];
      cornertypes = SplitCorners[Process[process], split];
      Sow[StringJoin["\\fmfsurround{",
        Riffle[ToString /@ Range[Length[Flatten[cornernames]]] //
          Reverse, ","], "}"]];
      Table[vname[Flatten[cornernames][[i]]] = ToString[i];
       Sow[StringJoin["\\fmfv{label=${",
         MakeLaTeXName[Flatten[cornertypes][[i]],
          Flatten[cornernames][[i]]], "}$}{", ToString[i], "}"]], {i,
        Length[Flatten[cornernames]]}];
      Table[Table[
        Sow[MakeFeynMFLine["v" <> ToString[i],
          vname[Flatten[cornernames[[i]]][[j]]],
          Flatten[cornertypes[[i]]][[j]], "", 0]], {j,
         Length[Flatten[cornertypes[[i]]]]}];
       Sow[MakeFeynMFLine[
         "v" <> ToString[Mod[i - 2, Length[split[[1]]]] + 1],
         "v" <> ToString[i], props[[i]], ",tension=0.4,tag="<>ToString[i],
         If[Length[split[[1]]] == 2, 0.6, 0]]], {i,
        Length[split[[1]]]}]; Sow["\\fmffreeze"];
      Table[
       MakeFeynMFLabels[i,
        "v" <> ToString[Mod[i - 2, Length[split[[1]]]] + 1],
        "v" <> ToString[i], props[[i]]], {i,
        Length[split[[1]]]}]]][[2, 1]], "\n"]]
```

```

MyRun[cmd_] :=
 Module[{val = Run[cmd]},
  If[TrueQ[val == 0], 0, Throw[{cmd, val}, Run]]]

MakeFeyn[process_, momconf_, split_, props_] := Module[{code = "
     \\nonstopmode
     \\pdfminorversion=3
     \\documentclass{letter}
     \\usepackage[usenames]{color} %used for font color
     \\usepackage{amssymb} %maths
     \\usepackage{amsmath} %maths
     \\usepackage[utf8]{inputenc} %useful to type directly diacritic \
characters
     \\usepackage{graphicx}
     \\usepackage[outdir=./]{epstopdf}
     \\DeclareGraphicsRule{*}{mps}{*}{}
     \\usepackage{feynmp}
     \\begin{document}
     \\thispagestyle{empty}
     \\begin{fmffile}{mathfig}
     \\begin{fmfgraph*}(120,120)
     " <> MakeFeynMF[process, momconf, split, props] <> "
     \\end{fmfgraph*}
     \\end{fmffile}
     \\end{document}", dir = CreateDirectory[], img},
  SetDirectory[dir]; Export[dir <> "/math.tex", code, "Text"];
  Catch[MyRun["pdflatex math"]; MyRun["mpost mathfig.mp"];
   MyRun["pdflatex math"]; MyRun["pdflatex math"];
   MyRun["pdfcrop --margin 10 math math-crop.pdf"];
   img = Import[dir <> "/math-crop.pdf", ImageSize -> 1000][[1]];
   ResetDirectory[]; DeleteDirectory[dir, DeleteContents -> True];
   Show[img, ImageSize -> 300],
   Run, (ResetDirectory[];
     Print[Row[{"Error: ", #1, " for ", dir}]]) &]]

GetFeyn[process_, momconf_, split_, props_] :=
 Module[{val = MakeFeyn[process, momconf, split, props]},
  If[TrueQ[val[[0]] == Graphics],
   GetFeyn[process, momconf, split, props] = val, val]]

GenerateAllDiagrams[proc_, momconf_, n_, f_: (Grid[#1] -> #3 &)] :=
```

```
 Function[split,
    f[split, #, GetFeyn[proc, momconf, split, #]] & /@
     RemoveIgnorableOptions[proc, split,
      SplitPropOptions[proc, split]]] /@ SplitOptions[proc, n] //
  Flatten

ForAllDiagrams[proc_, momconf_, n_, f_: (#1 -> #2 &)] :=
 Function[split,
    f[split, #] & /@
     RemoveIgnorableOptions[proc, split,
      SplitPropOptions[proc, split]]] /@ SplitOptions[proc, n] //
  Flatten

CalculateDiagramDependancies[proc_, momconf_, n_] :=
 ForAllDiagrams[proc, momconf, n,
  Function[{split, props},
   Function[
     newSplit, ({SplitName[split, newSplit], #} -> {split, props}) & /@
       RemoveIgnorableOptions[proc, SplitName[split, newSplit],
       SplitPropOptions[proc, split, props, newSplit]]] /@
    SplitOptions[proc, split]]]
ShowDependancyGraph[proc_, momconf_] :=
 Module[{FMFVertexShapeFunction},
  FMFVertexShapeFunction[pos_, {split_, props_}, size_, args___] :=
   Inset[GetFeyn[proc, momconf, split, props], pos, {0, 0}, size];
  Graph[{CalculateDiagramDependancies[proc, momconf, 2],
     CalculateDiagramDependancies[proc, momconf, 3]} // Flatten,
   VertexShapeFunction -> FMFVertexShapeFunction, VertexSize -> 1.5,
   DirectedEdges -> False, GraphLayout -> "SpringEmbedding"]]

GenerateDependancyGraphDot[procP_, momconfP_, name_: "graph"] :=
 Module[{ToN$Value = 0, ToN$Values = {}, ToN, dir = CreateDirectory[],
    spec, graph}, ToN[l_] := (ToN$Value = ToN$Value + 1;
    ToN$Values = Union[Append[ToN$Values, l]];
    ToN[l] = ToString[ToN$Value]);
  spec = StringJoin[
    Riffle[{"digraph{", "node[label=\"\",shape=none]",
       ToN[#[[1]]] <> "->" <> ToN[#[[2]]] <> ";" & /@
        CalculateDiagramDependancies[procP, momconfP, 2],
       ToN[#[[1]]] <> "->" <> ToN[#[[2]]] <> ";" & /@
        CalculateDiagramDependancies[procP, momconfP, 3],
       StringJoin["{rank=same;", Riffle[#, ";"], "}"] & /@
```

```
         Map[ToString[#[[1]]] &,
           GatherBy[
            SortBy[{ToN[#], Length[#[[1, 1]]]} & /@ ToN$Values, Last],
            Last], {2}],
        ToN[#] <> "[image=\"" <>
            Export[dir <> "/" <> ToN[#] <> ".eps",
             GetFeyn[procP, momconfP, #[[1]], #[[2]]]] <> "\"];" & /@
          ToN$Values, "}"} // Flatten, "\n"]];
  graph = Export[dir <> "/graph.txt", spec];
  Catch[MyRun[
     "dot -Tps < " <> graph <> " > " <> dir <> "/graph.eps"];
   MyRun["eps2eps " <> dir <> "/graph.eps " <> dir <> "/graph2.eps"];
   MyRun["ps2pdf -dEPSCrop " <> dir <> "/graph2.eps " <> name <>
      ".pdf"];DeleteDirectory[dir, DeleteContents -> True];,
   Run, Print[Row[{"Error: ", #1, " for ", dir}]] &]]
```

Appendix C

6 Dimensional Spinor Helicity Implementation

The code created as part of this project can also be downloaded from http://bit.ly/2oXSBSu.

Listing C.1: 6DSpinorHelicity.txt

```
(* Declare the dimension of vectors to enable extra simplifications *)
Dimension[p_] := None;
DeclareVectorDimension[p_, d_?EvenQ] := Dimension[p] = d;
UndeclareVectorDimension[p_] := Dimension[p] =.;

(* TraditionalForm and/or StandardForm representations for the various
    objects *)
SetAttributes[DoWith, HoldAll]
DoWith[s_, v_, e_] := Module[{tmp}, s = v; tmp = e; s =.; tmp]

MakeBoxes[spinor[p_, hs__], f : TraditionalForm | StandardForm] ^:=
 MakeBoxes[Subscript[u, hs][p], f]
MakeBoxes[spinorbar[p_, hs__], f : TraditionalForm | StandardForm] ^:=
  MakeBoxes[Subscript[UnderBar[u], hs][p], f]
Subscript[u, hs__][p_] := spinor[p, hs]
Subscript[UnderBar[u], hs__][p_] := spinorbar[p, hs]
AngleBracket = Sp;
MakeBoxes[Sp[a___], f : TraditionalForm] ^:=
 MakeBoxes[AngleBracket[a], f]
MpIndexSymbolIndex = 0;
MakeBoxes[Mp[(ma : Mom | MomM)[a_], (mb : Mom | MomM)[b_]],
```

```
  f : TraditionalForm] ^:=
 RowBox[{SubscriptBox[If[TrueQ[ma == Mom], "p", "P"],
    MakeBoxes[a, f]], "\[CenterDot]",
   SubscriptBox[If[TrueQ[mb == Mom], "p", "P"], MakeBoxes[b, f]]}]
MakeBoxes[Mp[a_, b_], f : TraditionalForm] ^:=
 RowBox[{Block[{MpIndexSymbol = MakeBoxes[\[Mu], f],
     MpIndexUp = True}, MakeBoxes[a, f]], "\[CenterDot]",
   Block[{MpIndexSymbol = MakeBoxes[\[Mu], f], MpIndexUp = False},
    MakeBoxes[b, f]]}]
MakeBoxes[\[Gamma], f : TraditionalForm] :=
 SubscriptBox["\[Gamma]", MpIndexSymbol] /; ! MpIndexUp
MakeBoxes[\[Gamma], f : TraditionalForm] :=
 SuperscriptBox["\[Gamma]", MpIndexSymbol] /; MpIndexUp
MakeBoxes[\[Sigma][1], f : TraditionalForm] :=
 SubscriptBox["\[Sigma]", MpIndexSymbol] /; ! MpIndexUp
MakeBoxes[\[Sigma][1], f : TraditionalForm] :=
 SuperscriptBox["\[Sigma]", MpIndexSymbol] /; MpIndexUp
MakeBoxes[\[Sigma][-1], f : TraditionalForm] :=
 SubscriptBox[OverscriptBox["\[Sigma]", "~"], MpIndexSymbol] /; !
   MpIndexUp
MakeBoxes[\[Sigma][-1], f : TraditionalForm] :=
 SuperscriptBox[OverscriptBox["\[Sigma]", "~"], MpIndexSymbol] /;
  MpIndexUp
MakeBoxes[Mom[i_], f : TraditionalForm] :=
 SubscriptBox["p", RowBox[{MakeBoxes[i, f], MpIndexSymbol}]] /; !
   MpIndexUp
MakeBoxes[Mom[i_], f : TraditionalForm] :=
 SubsuperscriptBox["p", MakeBoxes[i, f], MpIndexSymbol] /; MpIndexUp
MakeBoxes[MomM[i_], f : TraditionalForm] :=
 SubscriptBox["P", RowBox[{MakeBoxes[i, f], MpIndexSymbol}]] /; !
   MpIndexUp
MakeBoxes[MomM[i_], f : TraditionalForm] :=
 SubsuperscriptBox["P", MakeBoxes[i, f], MpIndexSymbol] /; MpIndexUp
MakeBoxes[Mp[a_, b_, i_], f : TraditionalForm] ^:=
 RowBox[{DoWith[MpIndexUp$$[i], True, MakeBoxes[a, f]],
   DoWith[MpIndexUp$$[i], False, MakeBoxes[b, f]]}]
MakeBoxes[\[Gamma][i_], f : TraditionalForm] :=
 SubscriptBox["\[Gamma]", MakeBoxes[i, f]] /; ! MpIndexUp$$[i]
MakeBoxes[\[Gamma][i_], f : TraditionalForm] :=
 SuperscriptBox["\[Gamma]", MakeBoxes[i, f]]
MakeBoxes[\[Sigma][1][i_], f : TraditionalForm] :=
 SubscriptBox["\[Sigma]", MakeBoxes[i, f]] /; ! MpIndexUp$$[i]
```

```
MakeBoxes[\[Sigma][1][i_], f : TraditionalForm] :=
 SuperscriptBox["\[Sigma]", MakeBoxes[i, f]]
MakeBoxes[\[Sigma][-1][i_], f : TraditionalForm] :=
 SubscriptBox[OverscriptBox["\[Sigma]", "~"], MakeBoxes[i, f]] /; !
   MpIndexUp$$[i]
MakeBoxes[\[Sigma][-1][i_], f : TraditionalForm] :=
 SuperscriptBox[OverscriptBox["\[Sigma]", "~"], MakeBoxes[i, f]]
MakeBoxes[Mom[p_][i_], f : TraditionalForm] :=
 SubscriptBox["p", RowBox[{MakeBoxes[p, f], MakeBoxes[i, f]}]] /; !
   MpIndexUp$$[i]
MakeBoxes[Mom[p_][i_], f : TraditionalForm] :=
 SubsuperscriptBox["p", MakeBoxes[p, f], MakeBoxes[i, f]]
MakeBoxes[Metric[i_, j_], f : TraditionalForm] :=
 SuperscriptBox["g", RowBox[{MakeBoxes[i, f], MakeBoxes[j, f]}]]
MakeBoxes[Metric[i_, j_], f : TraditionalForm] :=
 SubscriptBox["g",
   RowBox[{MakeBoxes[i, f], MakeBoxes[j, f]}]] /; !
    MpIndexUp$$[i] && ! MpIndexUp$$[j]
MakeBoxes[Metric[i_, j_], f : TraditionalForm] :=
 SubscriptBox[SuperscriptBox["g", MakeBoxes[i, f]],
   MakeBoxes[j, f]] /; ! MpIndexUp$$[j]
MakeBoxes[Metric[i_, j_], f : TraditionalForm] :=
 SuperscriptBox[SubscriptBox["g", MakeBoxes[i, f]],
   MakeBoxes[j, f]] /; ! MpIndexUp$$[i]
MakeBoxes[CalculatedP[mom_], f : TraditionalForm] :=
 RowBox[{SubscriptBox["p", "calc"], "[", MakeBoxes[mom, f], "]"}]
MakeBoxes[Mom[i_], f : TraditionalForm] :=
 SubscriptBox["p", MakeBoxes[i, f]]
MakeBoxes[Mom[i_], f : TraditionalForm] :=
 SubscriptBox["p", MakeBoxes[i, f]]

MakeBoxes[shift[{p_, q_}, z_, h_List, l_List][p_],
  f : TraditionalForm | StandardForm] :=
 MakeBoxes[Subsuperscript[OverHat[p] -> q, Row[h], Row[l]][z], f]
MakeBoxes[shift[{p_, q_}, z_, h_List, l_List][q_],
  f : TraditionalForm | StandardForm] :=
 MakeBoxes[Subsuperscript[p -> OverHat[q], Row[h], Row[l]][z], f]
MakeBoxes[Mom[shift[{p_, q_}, z_, h_List, l_List][p_]],
  f : TraditionalForm] :=
 SubscriptBox[
   MakeBoxes[
    Subsuperscript[OverHat[Subscript["p", p]] -> Subscript["p", q],
```

```
      Row[h], Row[l]][z], f], MpIndexSymbol] /; ! MpIndexUp
MakeBoxes[Mom[shift[{p_, q_}, z_, h_List, l_List][q_]],
  f : TraditionalForm] :=
 SubscriptBox[
   MakeBoxes[
    Subsuperscript[Subscript["p", p] -> OverHat[Subscript["p", q]],
      Row[h], Row[l]][z], f], MpIndexSymbol] /; ! MpIndexUp
MakeBoxes[Mom[shift[{p_, q_}, z_, h_List, l_List][p_]],
  f : TraditionalForm] :=
 SuperscriptBox[
   MakeBoxes[
    Subsuperscript[OverHat[Subscript["p", p]] -> Subscript["p", q],
      Row[h], Row[l]][z], f], MpIndexSymbol] /; MpIndexUp
MakeBoxes[Mom[shift[{p_, q_}, z_, h_List, l_List][q_]],
  f : TraditionalForm] :=
 SuperscriptBox[
   MakeBoxes[
    Subsuperscript[Subscript["p", p] -> OverHat[Subscript["p", q]],
      Row[h], Row[l]][z], f], MpIndexSymbol] /; MpIndexUp
MakeBoxes[Mom[shift[{p_, q_}, z_, h_List, l_List][p_]][i_],
  f : TraditionalForm] :=
 SubscriptBox[
   MakeBoxes[
    Subsuperscript[OverHat[Subscript["p", p]] -> Subscript["p", q],
      Row[h], Row[l]][z], f], MakeBoxes[i, f]] /; ! MpIndexUp$$[i]
MakeBoxes[Mom[shift[{p_, q_}, z_, h_List, l_List][q_]][i_],
  f : TraditionalForm] :=
 SubscriptBox[
   MakeBoxes[
    Subsuperscript[Subscript["p", p] -> OverHat[Subscript["p", q]],
      Row[h], Row[l]][z], f], MakeBoxes[i, f]] /; ! MpIndexUp$$[i]
MakeBoxes[Mom[shift[{p_, q_}, z_, h_List, l_List][p_]][i_],
  f : TraditionalForm] :=
 SuperscriptBox[
  MakeBoxes[
   Subsuperscript[OverHat[Subscript["p", p]] -> Subscript["p", q],
     Row[h], Row[l]][z], f], MakeBoxes[i, f]]
MakeBoxes[Mom[shift[{p_, q_}, z_, h_List, l_List][q_]][i_],
  f : TraditionalForm] :=
 SuperscriptBox[
  MakeBoxes[
   Subsuperscript[Subscript["p", p] -> OverHat[Subscript["p", q]],
```

```
      Row[h], Row[l]][z], f], MakeBoxes[i, f]]
MakeBoxes[Mom[shift[{p_, q_}, z_, h_List, l_List][p_]],
  f : TraditionalForm] :=
 MakeBoxes[
  Subsuperscript[OverHat[Subscript["p", p]] -> Subscript["p", q],
    Row[h], Row[l]][z], f]
MakeBoxes[Mom[shift[{p_, q_}, z_, h_List, l_List][q_]],
  f : TraditionalForm] :=
 MakeBoxes[
  Subsuperscript[Subscript["p", p] -> OverHat[Subscript["p", q]],
    Row[h], Row[l]][z], f]
Subsuperscript[OverHat[p_] -> q_, Row[h_], Row[l_]][z_] :=
 shift[{p, q}, z, h, l][p]
Subsuperscript[p_ -> OverHat[q_], Row[h_], Row[l_]][z_] :=
 shift[{p, q}, z, h, l][q]

Subscript["\[PlusMinus]", p_] := HelicitySign[p]
MakeBoxes[HelicitySign[p_], f : TraditionalForm | StandardForm] ^:=
 MakeBoxes[Subscript["\[PlusMinus]", p], f]

(* An object representing a sign whose value may not be known yet but
    which can still be simplified since it is known that it must be
    either 1 or -1 *)
HelicitySign[a_, b__] := HelicitySign[a] HelicitySign[b]
HelicitySign[a_^n_] := HelicitySign[a]^n
HelicitySign[a_ b_] := HelicitySign[a, b]
HelicitySign[-a_] := -HelicitySign[a]
HelicitySign[1] := 1
HelicitySign[-1] := -1
HelicitySign[a_]^n_?OddQ ^:= HelicitySign[a]
HelicitySign[a_]^n_?EvenQ ^:= 1

(* Declare spinors, conjugate spinors, slashed matrices, momenta and
    their products along with many basic simplifications that are always
     applied *)
Attributes[Sp] = {Flat};
Attributes[Metric] = {Orderless};
Default[Mp, 3] := Sequence[]
Mp[a_, b_, i_.] := Mp[b, a, i] /; Order[a, b] == 1

Sp[pre___, spinorbar[p1_, hs1__], mid___, spinor[p2_, hs2__],
  post__] := Sp[pre, post] Sp[spinorbar[p1, hs1], mid, spinor[p2, hs2]]
```

```
Sp[pre__, spinorbar[p1_, hs1__], mid___, spinor[p2_, hs2__],
  post___] :=
 Sp[pre, post] Sp[spinorbar[p1, hs1], mid, spinor[p2, hs2]]
Sp[pre___, a_ + b_, post___] := Sp[pre, a, post] + Sp[pre, b, post]
Sp[pre___, a_ b_, post___] :=
 a Sp[pre, b, post] /;
  FreeQ[a //.
    Sp[spinorbar[p1_, hs1__], mid___, spinor[p2_, hs2__]] :> 1,
   spinor | spinorbar | \[Gamma] | Sm | SmM]
Mp[a_ + b_, c_, i_.] := Mp[a, c, i] + Mp[b, c, i]
Mp[a_, b_ + c_, i_.] := Mp[a, b, i] + Mp[a, c, i]
(* Factors that contain no free objects with a space-time index can be
    moved to outside of the product *)
Mp[a_ b_, c_] :=
 a Mp[b, c] /;
  FreeQ[a //. Mp[__] :> 1, Mom | MomM | \[Gamma] | \[Sigma] | Metric]
Mp[a_, b_ c_] :=
 b Mp[a, c] /;
  FreeQ[b //. Mp[__] :> 1, Mom | MomM | \[Gamma] | \[Sigma] | Metric]
Mp[a_ b_, c_, i_] :=
 a Mp[b, c, i] /;
  FreeQ[a, Mom[_][i] | MomM[_][i] | \[Gamma][i] | \[Sigma][_][i] |
    Metric[i, _] | Metric[_, i]]
Mp[a_, b_ c_, i_] :=
 b Mp[a, c, i] /;
  FreeQ[b,
   Mom[_][i] | MomM[_][i] | \[Gamma][i] | \[Sigma][_][i] |
    Metric[i, _] | Metric[_, i]]
Mp[a_[i_], b_[i_], i_] := Mp[a, b]
Mp[a_[i_], Metric[i_, j_], i_] := a[j]
Mp[Metric[i_, j_], a_[i_], i_] := a[j]
Mp[Metric[i_, j_], Metric[i_, k_], i_] := Metric[j, k]
Mp[Sp[pre___, \[Gamma][i_], post___], b_[i_], i_] :=
 Mp[Sp[pre, \[Gamma], post], b]
Mp[a_[i_], Sp[pre___, \[Gamma][i_], post___], i_] :=
 Mp[a, Sp[pre, \[Gamma], post]]
Mp[Sp[pre___, \[Gamma][i_], post___],
  Sp[pre2___, \[Gamma][i_], post2___], i_] :=
 Mp[Sp[pre, \[Gamma], post], Sp[pre2, \[Gamma], post2]]
Mp[Sp[pre___, \[Gamma][i_], post___], Metric[i_, j_], i_] :=
 Sp[pre, \[Gamma][j], post]
Mp[Metric[i_, j_], Sp[pre___, \[Gamma][i_], post___], i_] :=
```

```
 Sp[pre, \[Gamma][j], post]
Mp[Sp[pre___, \[Sigma][l_][i_], post___], b_[i_], i_] :=
 Mp[Sp[pre, \[Sigma][l], post], b]
Mp[a_[i_], Sp[pre___, \[Sigma][l_][i_], post___], i_] :=
 Mp[a, Sp[pre, \[Sigma][l], post]]
Mp[Sp[pre___, \[Sigma][l_][i_], post___],
  Sp[pre2___, \[Sigma][l2_][i_], post2___], i_] :=
 Mp[Sp[pre, \[Sigma][l], post], Sp[pre2, \[Sigma][l2], post2]]
Mp[Sp[pre___, \[Sigma][l_][i_], post___], Metric[i_, j_], i_] :=
 Sp[pre, \[Sigma][l][j], post]
Mp[Metric[i_, j_], Sp[pre___, \[Sigma][l_][i_], post___], i_] :=
 Sp[pre, \[Sigma][l][j], post]
Mp[Mom[a_], Mom[a_]] := 0
Mp[0, _, i_.] := 0
Mp[_, 0, i_.] := 0
Mp2[a_] := Mp[a, a]

Sp[pre___, s1 : {___List}, s2 : {___List}, post___] :=
 Sp[pre, s1.s2, post]
b_ Sp[{a___}] ^:= Sp[b {a}]
Sp /: Sp[{a___}] + Sp[{b___}] := Sp[{a} + {b}]
Sp[{a___}] == Sp[{b___}] ^:= {a} == {b}
Sp[{{n_}}] := n
Mp[p1_List, p2_List] :=
 p1[[1]] p2[[1]] -
  Sum[p1[[$$i]] p2[[$$i]], {$$i, 2, Min[Length[p1], Length[p2]]}]

(* Convert the Minkowski product of momenta with a spinor chain
    containing a gamma matrix to a slashed matrix in the appropriate
    location *)
DoMpToSm[expr_] :=
 expr //. {Mp[Mom[a_], Sp[pre__, \[Gamma], post___]] |
      Mp[Sp[pre__, \[Gamma], post___], Mom[a_]] :>
     Sp[pre, Sm[a], post],
    Mp[Mom[a_], Sp[pre__, \[Sigma][l_], post___]] |
      Mp[Sp[pre__, \[Sigma][l_], post___], Mom[a_]] :>
     Sp[pre, Sm[a, l], post],
    Mp[MomM[a_], Sp[pre__, \[Gamma], post___]] |
      Mp[Sp[pre__, \[Gamma], post___], MomM[a_]] :>
     Sp[pre, SmM[a], post],
    Mp[MomM[a_], Sp[pre__, \[Sigma][l_], post___]] |
      Mp[Sp[pre__, \[Sigma][l_], post___], MomM[a_]] :>
```

```
    Sp[pre, SmM[a, 1], post]}

(* Convert the Minkowski product of two spinor chains, both containing a
      gamma matrix, to products of spinor chains *)
DoMpToSpinorChains[expr_] :=
 expr //.
   {Mp[Sp[pre1___, Sm[p_, mh_], \[Sigma][h_], post1___],
      Sp[pre2___, \[Sigma][h_],
       post2___]] :> -Mp[Sp[pre1, \[Sigma][mh], Sm[p, h], post1],
         Sp[pre2, \[Sigma][h], post2]] +
       2 Sp[pre1, post1] Sp[pre2, Sm[p, h], post2] /; mh == -h,
    Mp[Sp[pre1___, \[Sigma][h_], Sm[p_, mh_], post1___],
      Sp[pre2___, \[Sigma][h_],
       post2___]] :> -Mp[Sp[pre1, Sm[p, h], \[Sigma][mh], post1],
         Sp[pre2, \[Sigma][h], post2]] +
       2 Sp[pre1, post1] Sp[pre2, Sm[p, h], post2] /; mh == -h,
    Mp[Sp[pre2___, \[Sigma][h_], post2___],
      aa : Sp[pre1___, Sm[p_, mh_], \[Sigma][h_],
        post1___]] :> -Mp[Sp[pre1, \[Sigma][mh], Sm[p, h], post1],
         Sp[pre2, \[Sigma][h], post2]] +
       2 Sp[pre1, post1] Sp[pre2, Sm[p, h], post2] /; mh == -h,
    Mp[Sp[pre2___, \[Sigma][h_], post2___],
      aa : Sp[pre1___, \[Sigma][h_], Sm[p_, mh_],
        post1___]] :> -Mp[Sp[pre1, Sm[p, h], \[Sigma][mh], post1],
         Sp[pre2, \[Sigma][h], post2]] +
       2 Sp[pre1, post1] Sp[pre2, Sm[p, h], post2] /;
      mh == -h} //.
  {Mp[
     Sp[spinorbar[p1_, hp1_, lp1_],
      midpre : (Sm[_, _] ...), \[Sigma][h_], midpost : (Sm[_, _] ...),
       spinor[p2_, hp2_, lp2_]],
     Sp[pre___, \[Sigma][mh_], post___]] :>
    2 (Sp[pre, midpost, spinor[p2, hp2, lp2], spinorbar[p1, hp1, lp1],
         midpre, post] -
       HelicitySign[lp1, lp2, (-1)^(Length[{midpre, midpost}])] Sp[
         pre, Sequence @@ Reverse[{midpre}], spinor[p1, hp1, lp1],
         spinorbar[p2, hp2, lp2], Sequence @@ Reverse[{midpost}],
         post]) /; mh == -h,
   Mp[Sp[pre___, \[Sigma][mh_], post___],
     aaaa : Sp[spinorbar[p1_, hp1_, lp1_],
       midpre : (Sm[_, _] ...), \[Sigma][h_],
       midpost : (Sm[_, _] ...), spinor[p2_, hp2_, lp2_]]] :>
```

```
    2 (Sp[pre, midpost, spinor[p2, hp2, lp2], spinorbar[p1, hp1, lp1],
          midpre, post] -
        HelicitySign[lp1, lp2, (-1)^(Length[{midpre, midpost}])] Sp[
          pre, Sequence @@ Reverse[{midpre}], spinor[p1, hp1, lp1],
          spinorbar[p2, hp2, lp2], Sequence @@ Reverse[{midpost}],
          post]) /; mh == -h,
  Mp[Sp[spinorbar[p1_, hp1_, lp1_],
      midpre : (Sm[_, _] ...), \[Sigma][h_], post___],
     Sp[pre___, \[Sigma][mh_], midpost : (Sm[_, _] ...),
      spinor[p2_, hp2_, lp2_]]] :>
    2 (Sp[spinorbar[p1, hp1, lp1], midpre, midpost,
          spinor[p2, hp2, lp2]] Sp[pre, post] -
        HelicitySign[lp1, lp2, (-1)^(Length[{midpre, midpost}])] Sp[
          pre, Sequence @@ Reverse[{midpre}], spinor[p1, hp1, lp1],
          spinorbar[p2, hp2, lp2], Sequence @@ Reverse[{midpost}],
          post]) /; mh == -h,
  Mp[Sp[pre___, \[Sigma][mh_], midpost : (Sm[_, _] ...),
      spinor[p2_, hp2_, lp2_]],
     aaaa : Sp[spinorbar[p1_, hp1_, lp1_],
       midpre : (Sm[_, _] ...), \[Sigma][h_], post___]] :>
    2 (Sp[spinorbar[p1, hp1, lp1], midpre, midpost,
          spinor[p2, hp2, lp2]] Sp[pre, post] -
        HelicitySign[lp1, lp2, (-1)^(Length[{midpre, midpost}])] Sp[
          spinorbar[p2, hp2, lp2], Sequence @@ Reverse[{midpost}],
          post] Sp[pre, Sequence @@ Reverse[{midpre}],
          spinor[p1, hp1, lp1]]) /; mh == -h}

(* Declare more simplifications and rearrangements for spinor products
    that are always applied *)
Sp[___, spinorbar[p_, hs1__], spinor[p_, hs2__], ___] := 0
Sp[___, Sm[p_, _], spinor[p_, hs__], ___] := 0
Sp[___, spinorbar[p_, hs__], Sm[p_, _], ___] := 0
Sp[___, Sm[p_], Sm[p_], ___] := 0
Sp[pre___, SmM[p_], SmM[p_], post___] := Sp[pre, post] Mp2[MomM[p]]
Sp[___, Sm[p_, _], Sm[p_, _], ___] := 0
Sp[pre___, SmM[p_, _], SmM[p_, _], post___] :=
 Sp[pre, post] Mp2[MomM[p]]

Sp[pre___, (Sm | SmM)[p1_, h1_] | \[Sigma][h1_],
  Sm[p2_, h2_] | \[Sigma][h2_], post___] := 0 /; h1 == h2
Sp[pre___, (sm : Sm | SmM)[p1_],
  o : ((Sm | SmM)[_, h2_] | \[Sigma][h2_]), post___] :=
```

```
 Sp[pre, sm[p1, -h2], o, post]
Sp[pre___,
  o : ((Sm | SmM)[_, h1_] | \[Sigma][h1_]), (sm : Sm | SmM)[p2_],
  post___] := Sp[pre, o, sm[p2, -h1], post]
Sp[pre___, \[Gamma], o : ((Sm | SmM)[_, h2_] | \[Sigma][h2_]),
  post___] := Sp[pre, \[Sigma][-h2], o, post]
Sp[pre___, o : ((Sm | SmM)[_, h1_] | \[Sigma][h1_]), \[Gamma],
  post___] := Sp[pre, o, \[Sigma][-h1], post]

Sp[pre___, (Sm | SmM)[p2_, h2_], spinor[p_, h_, hs___], post___] :=
 0 /; h2 == -h
Sp[pre___, \[Sigma][h2_], spinor[p_, h_, hs___], post___] :=
 0 /; h2 == -h
Sp[pre___, spinorbar[p_, hs__], (Sm | SmM)[p2_, h2_], post___] :=
 0 /; h2 == -{ConvertHels[hs]}[[1]]
Sp[pre___, spinorbar[p_, hs__], \[Sigma][h2_], post___] :=
 0 /; h2 == -{ConvertHels[hs]}[[1]]
Sp[pre___, (sm : Sm | SmM)[p2_], spinor[p_, h_, hs___], post___] :=
 Sp[pre, sm[p2, h], spinor[p, h, hs], post]
Sp[pre___, \[Gamma], spinor[p_, h_, hs___], post___] :=
 Sp[pre, \[Sigma][h], spinor[p, h, hs], post]
Sp[pre___, spinorbar[p_, hs__], (sm : Sm | SmM)[p2_], post___] :=
 Sp[pre, spinorbar[p, hs], sm[p2, {ConvertHels[hs]}[[1]]], post]
Sp[pre___, spinorbar[p_, hs__], \[Gamma], post___] :=
 Sp[pre, spinorbar[p, hs], \[Sigma][{ConvertHels[hs]}[[1]]], post]

Sp[pre___, spinorbar[p1_, hs1___], spinor[p2_, h2_, hs2___],
  post___] := 0 /; h2 == {ConvertHels[hs1]}[[1]]
Sp[pre___, spinorbar[p1_, hs1___], spinor[p2_, hs2___], post___] :=
 0 /; Dimension[p1] =!= None && Dimension[p2] =!= None &&
   MemberQ[Drop[{hs2}, -(Max[Dimension[p1], Dimension[p2]]/2 - 2)] -
     Drop[{ConvertHels[
        hs1]}, -(Max[Dimension[p1], Dimension[p2]]/2 - 2)], 0]
Sp[pre___, spinorbar[p1_, hs1___], sms : Sm[_, _] ...,
  spinor[p2_, hs2___], post___] :=
 0 /; Dimension[p1] =!= None &&
    Dimension[p2] =!= None && !
     MemberQ[Dimension[#[[1]]] & /@ {sms}, None] &&
   MemberQ[Drop[{hs2}, -(Max[Dimension[p1],
           Dimension[#[[1]]] & /@ {sms}, Dimension[p2]]/2 -
          2)] - (-1)^
       Length[{sms}] Drop[{ConvertHels[
```

```
          hs1]}, -(Max[Dimension[p1], Dimension[#[[1]]] & /@ {sms},
             Dimension[p2]]/2 - 2)], 0]

CompareHels[{}, {}] := False
CompareHels[{h1_, hh1___}, {h2_, hh2___}] :=
 Module[{order = Order[h1, h2]},
  If[order == 0, CompareHels[{hh1}, {hh2}], order > 0]]

CalculatedP /: Mom[CalculatedP[mom_]] := mom
CalculatedP /: MomM[CalculatedP[mom_]] := mom
CalculatedP /: Sm[CalculatedP[a_. Mom[b_] + mom_], h_] :=
 a Sm[b, h] + Sm[CalculatedP[mom], h]
CalculatedP /: Sm[CalculatedP[a_. Mom[b_]], h_] := a Sm[b, h]

RevP /: Mom[RevP[mom_]] := -Mom[mom]
RevP /: MomM[RevP[mom_]] := -MomM[mom]
RevP /: Sm[RevP[a_], h_] := -Sm[a, h]
RevP /: spinor[RevP[a_], hs__] := I spinor[a, hs]
RevP /: spinorbar[RevP[a_], hs__] := I spinorbar[a, hs]

FlipHelicities[Sp[els___]] := Sp @@ FlipHelicities[{els}]
FlipHelicities[{el_, rest___}] :=
 Join[{FlipHelicities[el]}, FlipHelicities[{rest}]]
FlipHelicities[{}] := {}
FlipHelicities[Sm[p_, i_]] := Sm[p, -i]
FlipHelicities[SmM[p_, i_]] := SmM[p, -i]
FlipHelicities[\[Sigma][i_]] := \[Sigma][-i]

Sp[spinorbar[p1_, hs1__], pre : (Sm | SmM)[__] ..., \[Sigma][h_],
  post : (Sm | SmM)[__] ..., spinor[p2_, hs2__]] :=
 SpFlipSign[{hs1}, {hs2}, Length[{pre, post}] + 1] Sp[
    spinorbar[p2, hs2],
    ConvertMidHelicities[Reverse[Sp[pre, \[Sigma][h], post]],
     Length[{hs1}]], spinor[p1, hs1]] /;
  CompareHels[{-Length[{pre}], hs1, {pre}, p1}, {-Length[{post}], hs2,
     Reverse[{post}], p2}]
Sp[spinorbar[p1_, hs1__], sms : (Sm | SmM)[__] ...,
  spinor[p2_, hs2__]] :=
 SpFlipSign[{hs1}, {hs2}, Length[{sms}]] Sp[spinorbar[p2, hs2],
    ConvertMidHelicities[Reverse[Sp[sms]], Length[{hs1}]],
    spinor[p1, hs1]] /;
  CompareHels[{hs1, Sp[sms], p1}, {hs2,
```

```
    ConvertMidHelicities[Reverse[Sp[sms]], Length[{hs1}]], p2}]

barspinor[p_, hs__] := spinorbar[p, ConvertHels[hs]]

(* Replace slashed matrices by spinors. d_ is the number of dimensions
    the expression is in. It is also possible for the replacement to be
    in terms of a specified set of helicity labels, in which case the
    dimension is set by the number of labels needed for a spinor *)
ConvertSmToSpinors[d_, Sm[p_, h_]] :=
 ConvertSmToSpinors[Sm[p, h], PadLeft[{}, d/2 - 2, 1]]
ConvertSmToSpinors[Sm[p_, h_], hs_List] :=
 Sum[Sp[spinor[p, -h, Sequence @@ (hs is)],
   barspinor[p, -h, Sequence @@ (hs is)]], {is,
   Tuples[{1, -1}, {Length[hs]}]}]
ConvertSmToSpinors[d_, p_][expr_] :=
 expr //. Sp[pre___, spinorbar[p2_, h2_, hs__], Sm[pp : p, h_],
      post___] :>
     Sp[pre, spinorbar[p2, h2, hs],
      ConvertSmToSpinors[Sm[pp, h], {hs}], post] //.
   Sp[pre___, Sm[pp : p, h_], spinor[p2_, h2_, hs__], post___] :>
    Sp[pre, ConvertSmToSpinors[Sm[pp, h], {hs}], spinor[p2, h2, hs],
     post] //. Sm[pp : p, h_] :> ConvertSmToSpinors[d, Sm[pp, h]]
ConvertSmToSpinors[p_, {h_, hs___}][expr_] :=
 expr //. Sm[pp : p, hh : h] :> ConvertSmToSpinors[Sm[pp, hh], {hs}]

(* Commute the given pair of slashed matrices wherever they occur in the
     expression *)
CommuteMatricies[m1_\[Sigma], m2p_][expr_] :=
 expr /. {Sp[pre___, m1, Sm[m2 : m2p, _], post___] :>
    2 Mom[m2] Sp[pre, post] - Sp[pre, Sm[m2], \[Gamma], post],
   Sp[pre___, m1, SmM[m2 : m2p, _], post___] :>
    2 MomM[m2] Sp[pre, post] - Sp[pre, SmM[m2], \[Gamma], post]}
CommuteMatricies[m1p_, m2_\[Sigma]][expr_] :=
 expr /. {Sp[pre___, Sm[m1 : m1p, _], m2, post___] :>
    2 Mom[m1] Sp[pre, post] - Sp[pre, \[Gamma], Sm[m1], post],
   Sp[pre___, SmM[m1 : m1p, _], m2, post___] :>
    2 MomM[m1] Sp[pre, post] - Sp[pre, \[Gamma], SmM[m1], post]}
CommuteMatricies[m1p_, m2p_][expr_] :=
 expr /. {Sp[pre___, Sm[m1 : m1p, a_], Sm[m2 : m2p, _], post___] :>
    2 Mp[Mom[m1], Mom[m2]] Sp[pre, post] -
     Sp[pre, Sm[m2], Sm[m1], post],
   Sp[pre___, SmM[m1 : m1p, a_], Sm[m2 : m2p, _], post___] :>
```

```
   2 Mp[MomM[m1], Mom[m2]] Sp[pre, post] -
    Sp[pre, Sm[m2], SmM[m1], post],
  Sp[pre___, Sm[m1 : m1p, a_], SmM[m2 : m2p, _], post___] :>
   2 Mp[Mom[m1], MomM[m2]] Sp[pre, post] -
    Sp[pre, SmM[m2], Sm[m1], post],
  Sp[pre___, SmM[m1 : m1p, a_], SmM[m2 : m2p, _], post___] :>
   2 Mp[MomM[m1], MomM[m2]] Sp[pre, post] -
    Sp[pre, SmM[m2], SmM[m1], post]}

(* Commute matrices if this can cause a term to vanish by placing two
    elements for the same massless momenta next to each other *)
CommuteMatriciesAway[expr_] :=
 expr //. {Sp[pre___, Sm[m1_, h_], Sm[m2_, l_], Sm[m1_, h_],
      post___] :> 2 Mp[Mom[m2], Mom[m1]] Sp[pre, Sm[m1, h], post],
    Sp[pre___, Sm[m1_, h_], SmM[m2_, l_], Sm[m1_, h_], post___] :>
     2 Mp[MomM[m2], Mom[m1]] Sp[pre, Sm[m1, h], post],
    Sp[pre___, SmM[m1_, h_], Sm[m2_, l_], SmM[m1_, h_], post___] :>
     2 Mp[Mom[m2], Mom[m1]] Sp[pre, Sm[m1, h], post] -
      Mp2[MomM[m1]] Sp[pre, SmMp[m2, h], post],
    Sp[pre___, SmM[m1_, h_], SmM[m2_, l_], SmM[m1_, h_], post___] :>
     2 Mp[MomM[m2], Mom[m1]] Sp[pre, Sm[m1, h], post] -
      Mp2[MomM[m1]] Sp[pre, SmMp[m2, h], post],
    Sp[pre___, Sm[m1_, l_], Sm[m2_, l_], spinor[m1_, h___],
      post___] :> 2 Mp[Mom[m2], Mom[m1]] Sp[pre, spinor[m1, h], post],
     Sp[pre___, Sm[m1_, l_], SmM[m2_, l_], spinor[m1_, h___],
      post___] :>
     2 Mp[MomM[m2], Mom[m1]] Sp[pre, spinor[m1, h], post],
    Sp[pre___, spinorbar[m1_, h___], Sm[m2_, l_], Sm[m1_, m_],
      post__] :>
     2 Mp[Mom[m2], Mom[m1]] Sp[pre, spinorbar[m1, h], post],
    Sp[pre___, spinorbar[m1_, h___], SmM[m2_, l_], Sm[m1_, m_],
      post__] :>
     2 Mp[MomN[m2], Mom[m1]] Sp[pre, spinorbar[m1, h], post]} //. {Sp[
     pre___, Sm[a_, h_], Sm[b_, l_], Sm[c_, m_], Sm[a_, n_],
     post___] :>
   4 Mp[Mom[a], Mom[b]] Mp[Mom[c], Mom[a]] Sp[pre, post] -
    2 Mp[Mom[a], Mom[b]] Sp[pre, Sm[a], Sm[c], post] -
    2 Mp[Mom[a], Mom[c]] Sp[pre, Sm[b], Sm[a], post],
  Sp[pre___, Sm[a_, h_], SmM[b_, l_], Sm[c_, m_], Sm[a_, n_],
     post___] :>
   4 Mp[Mom[a], MomM[b]] Mp[Mom[c], Mom[a]] Sp[pre, post] -
    2 Mp[Mom[a], MomM[b]] Sp[pre, Sm[a], Sm[c], post] -
```

```
    2 Mp[Mom[a], Mom[c]] Sp[pre, SmM[b], Sm[a], post],
  Sp[pre___, Sm[a_, h_], Sm[b_, l_], SmM[c_, m_], Sm[a_, n_],
    post___] :>
   4 Mp[Mom[a], Mom[b]] Mp[MomM[c], Mom[a]] Sp[pre, post] -
    2 Mp[Mom[a], Mom[b]] Sp[pre, Sm[a], SmM[c], post] -
    2 Mp[Mom[a], MomM[c]] Sp[pre, Sm[b], Sm[a], post],
  Sp[pre___, Sm[a_, h_], SmM[b_, l_], SmM[c_, m_], Sm[a_, n_],
    post___] :>
   4 Mp[Mom[a], MomM[b]] Mp[MomM[c], Mom[a]] Sp[pre, post] -
    2 Mp[Mom[a], MomM[b]] Sp[pre, Sm[a], SmM[c], post] -
    2 Mp[Mom[a], MomM[c]] Sp[pre, SmM[b], Sm[a], post],
  Sp[pre___, SmM[a_, h_], Sm[b_, l_], Sm[c_, m_], SmM[a_, n_],
    post___] :>
   4 Mp[Mom[a], Mom[b]] Mp[Mom[c], Mom[a]] Sp[pre, post] -
    2 Mp[Mom[a], Mom[b]] Sp[pre, Sm[a], Sm[c], post] -
    2 Mp[Mom[a], Mom[c]] Sp[pre, Sm[b], Sm[a], post] +
    Mp2[MomM[a]] Sp[pre, Sm[b], Sm[c], post],
  Sp[pre___, SmM[a_, h_], SmM[b_, l_], Sm[c_, m_], SmM[a_, n_],
    post___] :>
   4 Mp[Mom[a], MomM[b]] Mp[Mom[c], Mom[a]] Sp[pre, post] -
    2 Mp[Mom[a], MomM[b]] Sp[pre, Sm[a], Sm[c], post] -
    2 Mp[Mom[a], Mom[c]] Sp[pre, SmM[b], Sm[a], post] +
    Mp2[MomM[a]] Sp[pre, SmM[b], Sm[c], post],
  Sp[pre___, SmM[a_, h_], Sm[b_, l_], SmM[c_, m_], SmM[a_, n_],
    post___] :>
   4 Mp[Mom[a], Mom[b]] Mp[MomM[c], Mom[a]] Sp[pre, post] -
    2 Mp[Mom[a], Mom[b]] Sp[pre, Sm[a], SmM[c], post] -
    2 Mp[Mom[a], MomM[c]] Sp[pre, Sm[b], Sm[a], post] +
    Mp2[MomM[a]] Sp[pre, Sm[b], SmM[c], post],
  Sp[pre___, SmM[a_, h_], SmM[b_, l_], SmM[c_, m_], SmM[a_, n_],
    post___] :>
   4 Mp[Mom[a], MomM[b]] Mp[MomM[c], Mom[a]] Sp[pre, post] -
    2 Mp[Mom[a], MomM[b]] Sp[pre, Sm[a], SmM[c], post] -
    2 Mp[Mom[a], MomM[c]] Sp[pre, SmM[b], Sm[a], post] +
    Mp2[MomM[a]] Sp[pre, SmM[b], SmM[a], post]}

(* Commute gamma matrices if it can cause a term to vanish by placing
   two elements for the same massless momenta next to each other *)
CommuteSigmaMatriciesAway[expr_] :=
 expr //. {Sp[pre___, Sm[m1_, h_], \[Sigma][l_], Sm[m1_, h_],
    post___] :> 2 Mom[m1] Sp[pre, Sm[m1, h], post],
  Sp[pre___, SmM[m1_, h_], \[Sigma][l_], SmM[m1_, h_], post___] :>
```

```
      2 Mom[m1] Sp[pre, Sm[m1, h], post] -
       Mp2[MomM[m1]] Sp[pre, \[Sigma][h], post],
    Sp[pre___, Sm[m1_, l_], \[Sigma][l_], spinor[m1_, h___],
       post___] :> 2 Mom[m1] Sp[pre, spinor[m1, h], post],
    Sp[pre___, spinorbar[m1_, h___], \[Sigma][l_], Sm[m1_, m_],
       post___] :> 2 Mom[m1] Sp[pre, spinorbar[m1, h], post]}

(* Dimension independant code for evaluating expressions numerically *)
SpinorExpression[d_, p_List, hs___] :=
 SpinorExpression[d, PadRight[p, d, 0], hs] /; Length[p] != d
SpinorExpression[d_, p_List, hs___] :=
 SpinorExpression[d, PadRight[p, d, 0], hs] /; Length[p] != d
AdjointSpinor[d_, s_, hs__] :=
 AdjointSign[d, hs] Transpose[s].AdjointMetric[d]

\[Gamma]Expression[2, _, 0] := {{1}}
\[Gamma]Expression[2, h_, _] := {{h}}
\[Gamma]Expression[d_, h_, i_] := ArrayFlatten[{{
      \[Gamma]Expression[d - 2, h, i], 0}, {0, -
       \[Gamma]Expression[d - 2, -h, i]}}] /;
  d > 2 && EvenQ[d] && i < d - 2
\[Gamma]Expression[d_, s_, i_] :=
 ArrayFlatten[{{0,
      I IdentityMatrix[2^(d/2 - 2)]}, {I IdentityMatrix[2^(d/2 - 2)],
      0}}] /; d > 2 && EvenQ[d] && i == d - 2
\[Gamma]Expression[d_, s_, i_] :=
 ArrayFlatten[{{0,
      IdentityMatrix[2^(d/2 - 2)]}, {-IdentityMatrix[2^(d/2 - 2)],
      0}}] /; d > 2 && EvenQ[d] && i == d - 1

shift$$reps = {};
ReplaceShiftedSpinors[expr_] := expr //. shift$$reps

vectors$$momentums$$defined[_, _] := False
vectors$$spinors[tag_, p_] :=
 Module[{spinors$},
   spinors$[d_,
      hs__] := (spinors$[d, hs] =
       SpinorExpression[d, vectors$$momentums[tag, p], hs]);
   tag /: vectors$$spinors[tag, p] = spinors$] /;
  vectors$$momentums$$defined[tag, p]
```

```
vectors$$spinors[tag_, p : CalculatedP[pexpr_]] :=
 Module[{spinors$},
  spinors$[d_,
    hs__] := (spinors$[d, hs] =
     SpinorExpression[d, pexpr // Ev[d, tag], hs]);
  tag /: vectors$$spinors[tag, p] = spinors$]

vectors$$momentums[tag_,
  shift[{p_, q_}, z_, {h_}, {l_}][
   p_]] := (Mom[p] + z Sp[spinorbar[p, h], \[Gamma], spinor[q, l]]/2 //
     Ev[4, tag]) /;
  h == -1 && MatchQ[vectors$$momentums[tag, p], _List] &&
   MatchQ[vectors$$momentums[tag, q], _List]
vectors$$momentums[tag_,
  shift[{p_, q_}, z_, {h_}, {l_}][
   q_]] := (Mom[q] - z Sp[spinorbar[p, h], \[Gamma], spinor[q, l]]/2 //
     Ev[4, tag]) /;
  h == -1 && MatchQ[vectors$$momentums[tag, p], _List] &&
   MatchQ[vectors$$momentums[tag, q], _List]
shift$$reps =
  Join[shift$$reps, {Mom[shift[{p_, q_}, z_, {h_}, {l_}][p_]] :>
     Mom[p] + z Sp[spinorbar[p, h], \[Gamma], spinor[q, l]]/2,
    Mom[shift[{p_, q_}, z_, {h_}, {l_}][q_]] :>
     Mom[q] - z Sp[spinorbar[p, h], \[Gamma], spinor[q, l]]/2,
    Sm[shift[{p_, q_}, z_, {h_}, {l_}][p_], h_] :>
     Sm[p, h] + z Sp[spinor[q, l], spinorbar[p, h]],
    Sm[shift[{p_, q_}, z_, {h_}, {l_}][p_], l_] :>
     Sm[p, l] + z Sp[spinor[p, h], spinorbar[q, l]],
    Sm[shift[{p_, q_}, z_, {h_}, {l_}][q_], h_] :>
     Sm[q, h] - z Sp[spinor[q, l], spinorbar[p, h]],
    Sm[shift[{p_, q_}, z_, {h_}, {l_}][q_], l_] :>
     Sm[q, l] - z Sp[spinor[p, h], spinorbar[q, l]]}];
vectors$$spinors[tag_, shift[{p_, q_}, z_, {h_}, {l_}][p_]][4,
  a_] := (spinor[p, a] + z spinor[q, a] // Ev[4, tag]) /;
  h == -1 && a == 1 && MatchQ[vectors$$momentums[tag, p], _List] &&
   MatchQ[vectors$$momentums[tag, q], _List]
vectors$$spinors[tag_, shift[{p_, q_}, z_, {h_}, {l_}][p_]][4, a_] :=
  (spinor[p, a] // Ev[4, tag]) /;
  h == -1 && a == -1 && MatchQ[vectors$$momentums[tag, p], _List] &&
   MatchQ[vectors$$momentums[tag, q], _List]
vectors$$spinors[tag_, shift[{p_, q_}, z_, {h_}, {l_}][q_]][4,
  a_] := (spinor[q, a] - z spinor[p, a] // Ev[4, tag]) /;
```

```
  h == -1 && a == h && MatchQ[vectors$$momentums[tag, p], _List] &&
   MatchQ[vectors$$momentums[tag, q], _List]
vectors$$spinors[tag_, shift[{p_, q_}, z_, {h_}, {l_}][q_]][4,
  a_] := (spinor[q, a] // Ev[4, tag]) /;
  h == -1 && a == -h && MatchQ[vectors$$momentums[tag, p], _List] &&
   MatchQ[vectors$$momentums[tag, q], _List]
shift$$reps =
  Join[shift$$reps, {(ss : spinor | spinorbar)[
       shift[{p_, q_}, z_, {h_}, {l_}][p_], h_] :>
     ss[p, h], (ss : spinor | spinorbar)[
       shift[{p_, q_}, z_, {h_}, {l_}][p_], l_] :>
     ss[p, l] + z ss[q, l], (ss : spinor | spinorbar)[
       shift[{p_, q_}, z_, {h_}, {l_}][q_], l_] :>
     ss[q, l], (ss : spinor | spinorbar)[
       shift[{p_, q_}, z_, {h_}, {l_}][q_], h_] :>
     ss[q, h] - z ss[p, h]}];

vectors$$momentums[tag_,
  shift[{p_, q_}, z_, {h_, l_}, {m_, n_}][
   p_]] := (Mom[p] -
     HelicitySign[h,
       n] z Sp[spinorbar[p, h, l], \[Gamma], Sm[q], spinor[p, m, n]]/
       2 // Ev[6, tag]) /;
  h == -m && MatchQ[vectors$$momentums[tag, p], _List] &&
   MatchQ[vectors$$momentums[tag, q], _List]
vectors$$momentums[tag_,
  shift[{p_, q_}, z_, {h_, l_}, {m_, n_}][
   q_]] := (Mom[q] +
     HelicitySign[h,
       n] z Sp[spinorbar[p, h, l], \[Gamma], Sm[q], spinor[p, m, n]]/
       2 // Ev[6, tag]) /;
  h == -m && MatchQ[vectors$$momentums[tag, p], _List] &&
   MatchQ[vectors$$momentums[tag, q], _List]
shift$$reps =
  Join[shift$$reps, {Mom[
       shift[{p_, q_}, z_, {h_, l_}, {m_, n_}][p_]] :>
     Mom[p] -
      HelicitySign[h,
        n] z Sp[spinorbar[p, h, l], \[Gamma], Sm[q], spinor[p, m, n]]/
        2, Mom[shift[{p_, q_}, z_, {h_, l_}, {m_, n_}][q_]] :>
     Mom[q] +
      HelicitySign[h,
```

```
       n] z Sp[spinorbar[p, h, l], \[Gamma], Sm[q], spinor[p, m, n]]/
       2,
    Sm[shift[{p_, q_}, z_, {h_, l_}, {m_, n_}][p_], h_] :>
     Sm[p, h] +
      z  (HelicitySign[h, l] Sp[Sm[q], spinor[p, h, l],
          spinorbar[p, m, n]] -
        HelicitySign[m, n] Sp[spinor[p, m, n], spinorbar[p, h, l],
          Sm[q]]),
    Sm[shift[{p_, q_}, z_, {h_, l_}, {m_, n_}][p_], m_] :>
     Sm[p, m] +
      z (-HelicitySign[h, l] Sp[spinor[p, h, l], spinorbar[p, m, n],
          Sm[q]] +
        HelicitySign[m, n] Sp[Sm[q], spinor[p, m, n],
          spinorbar[p, h, l]]),
    Sm[shift[{p_, q_}, z_, {h_, l_}, {m_, n_}][q_], h_] :>
     Sm[q, h] -
      z (HelicitySign[h, l] Sp[Sm[q], spinor[p, h, l],
          spinorbar[p, m, n]] -
        HelicitySign[m, n] Sp[spinor[p, m, n], spinorbar[p, h, l],
          Sm[q]]),
    Sm[shift[{p_, q_}, z_, {h_, l_}, {m_, n_}][q_], m_] :>
     Sm[q, m] -
      z (-HelicitySign[h, l] Sp[spinor[p, h, l], spinorbar[p, m, n],
          Sm[q]] +
        HelicitySign[m, n] Sp[Sm[q], spinor[p, m, n],
          spinorbar[p, h, l]])}];
vectors$$spinors[tag_, shift[{p_, q_}, z_, {h_, l_}, {m_, n_}][p_]][6,
   a_, b_] := (spinor[p, a, b] -
     z HelicitySign[a, n] Sm[q, m].spinor[p, m, n] // Ev[6, tag]) /;
  m == -h && h == a && l == -b &&
   MatchQ[vectors$$momentums[tag, p], _List] &&
   MatchQ[vectors$$momentums[tag, q], _List]
vectors$$spinors[tag_, shift[{p_, q_}, z_, {h_, l_}, {m_, n_}][p_]][6,
   a_, b_] := (spinor[p, a, b] -
     z HelicitySign[a, l] Sm[q, h].spinor[p, h, l] // Ev[6, tag]) /;
  m == -h && m == a && n == -b &&
   MatchQ[vectors$$momentums[tag, p], _List] &&
   MatchQ[vectors$$momentums[tag, q], _List]
vectors$$spinors[tag_, shift[{p_, q_}, z_, {h_, l_}, {m_, n_}][p_]][6,
   a_, b_] := (spinor[p, a, b] // Ev[6, tag]) /;
  m == -h && ((h == a && l == b) || (m == a && n == b)) &&
   MatchQ[vectors$$momentums[tag, p], _List] &&
```

```
    MatchQ[vectors$$momentums[tag, q], _List]
vectors$$spinors[tag_, shift[{p_, q_}, z_, {h_, l_}, {m_, n_}][q_]][6,
    a_, b_] := (spinor[q, a, b] +
      z HelicitySign[h, l] spinor[p, h, l] Sp[spinorbar[p, m, n],
        spinor[q, a, b]] // Ev[6, tag]) /;
  m == -h && h == a && MatchQ[vectors$$momentums[tag, p], _List] &&
    MatchQ[vectors$$momentums[tag, q], _List]
vectors$$spinors[tag_, shift[{p_, q_}, z_, {h_, l_}, {m_, n_}][q_]][6,
    a_, b_] := (spinor[q, a, b] +
      z HelicitySign[m, n] spinor[p, m, n] Sp[spinorbar[p, h, l],
        spinor[q, a, b]] // Ev[6, tag]) /;
  m == -h && m == a && MatchQ[vectors$$momentums[tag, p], _List] &&
    MatchQ[vectors$$momentums[tag, q], _List]
shift$$reps =
   Join[shift$$reps, {(ss : spinor | spinorbar)[
        shift[{p_, q_}, z_, {h_, l_}, {m_, n_}][p_], h_, l_] :>
      ss[p, h, l], (ss : spinor | spinorbar)[
        shift[{p_, q_}, z_, {h_, l_}, {m_, n_}][p_], m_, n_] :>
      ss[p, m, n],
     spinor[shift[{p_, q_}, z_, {h_, l_}, {m_, n_}][p_], h_, b_] :>
      spinor[p, h, b] -
        z HelicitySign[h, n] Sp[Sm[q, m], spinor[p, m, n]] /; b == -l,
     spinor[shift[{p_, q_}, z_, {h_, l_}, {m_, n_}][p_], m_, b_] :>
      spinor[p, m, b] -
        z HelicitySign[m, l] Sp[Sm[q, h], spinor[p, h, l]] /; b == -n,
     spinorbar[shift[{p_, q_}, z_, {h_, l_}, {m_, n_}][p_], h_, b_] :>
      spinorbar[p, h, b] -
        z HelicitySign[h, l] Sp[spinorbar[p, m, n], Sm[q, m]] /;
       b == -l,
     spinorbar[shift[{p_, q_}, z_, {h_, l_}, {m_, n_}][p_], m_, b_] :>
      spinorbar[p, m, b] -
        z HelicitySign[m, n] Sp[spinorbar[p, h, l], Sm[q, h]] /;
       b == -n, (ss : spinor | spinorbar)[
        shift[{p_, q_}, z_, {h_, l_}, {m_, n_}][q_], h_, b_] :>
      ss[q, h, b] +
       z If[TrueQ[ss == spinorbar], HelicitySign[h, b],
         HelicitySign[h, l]] ss[p, h, l] Sp[spinorbar[p, m, n],
         spinor[q, h, b]], (ss : spinor | spinorbar)[
        shift[{p_, q_}, z_, {h_, l_}, {m_, n_}][q_], m_, b_] :>
      ss[q, m, b] +
       z If[TrueQ[ss == spinorbar], HelicitySign[m, b],
         HelicitySign[m, n]] ss[p, m, n] Sp[spinorbar[p, h, l],
```

```
        spinor[q, m, b]]}];

(* Function to declare momenta for names on the momenta set 'tag' *)
DeclareVectorMomentum[tag_, p_,
  pp_List] := (tag /: vectors$$momentums$$defined[tag, p] = True;
  tag /: vectors$$momentums[tag, p] = pp)
$CheckSpinorsConsistent = True;
SetAttributes[CheckSpinorsMomentum, HoldFirst]
CheckSpinorsMomentum::Inconsistent =
  "The spinors `3` and `4` give a momentum of `5` which is not the \
same as the momenta `6` for `1` `2`.";
CheckSpinorsMomentum[tag_,
  p_, {hs : (1 | -1) ...} -> {spp : {{_} ..}, spm : {{_} ..}}] :=
 Message[CheckSpinorsMomentum::Inconsistent, tag, p, spp, spm,
   CalculateMomenta[spp, spm, {hs}],
   vectors$$momentums[tag, p]] /; $CheckSpinorsConsistent &&
   Simplify[
    CalculateMomenta[spp, spm, {hs}] != vectors$$momentums[tag, p]]
DefineSpinors[
  spinors$_, {hs : (1 | -1) ...} -> {spp : {{_} ..},
    spm : {{_} ..}}] := (spinors$[_, hs, 1] = spp;
  spinors$[_, hs, -1] = spm)
DeclareVectorMomentum[tag_, p_,
  sp : ({hs : (1 | -1) ...} -> {spp : {{_} ..}, spm : {{_} ..}}),
  sps : ({(1 | -1) ...} -> {{{_} ..}, {{_} ..}}) ...] := (tag /:
   vectors$$momentums[tag,
    p] := (tag /: vectors$$momentums[tag, p] =
      CalculateMomenta[spp, spm, {hs}]);
  CheckSpinorsMomentum[tag, p, #] & /@ {sp, sps};
  Module[{spinors$}, DefineSpinors[spinors$, #] & /@ {sp, sps};
   tag /: vectors$$spinors[tag, p] = spinors$; spinors$])
DeclareVectorMomentum[tag_, p_, spspre : ({(1 | -1) ...} -> _) ...,
  sph : {(1 | -1) ...} -> {sp_Sp, other_} ,
  spspost : ({(1 | -1) ...} -> _) ...] :=
 DeclareVectorMomentum[tag, p, spspre, sph -> {other, sp[[1]]},
   spspost] /; Length[sp] == 1
DeclareVectorMomentum[tag_, p_, spspre : ({(1 | -1) ...} -> _) ...,
  sph : {(1 | -1) ...} -> {other_, sp_Sp} ,
  spspost : ({(1 | -1) ...} -> _) ...] :=
 DeclareVectorMomentum[tag, p, spspre, sph -> {sp[[1]], other},
   spspost] /; Length[sp] == 1
DeclareVectorMomentum[tag_, p_,
```

```
  sp : {({(1 | -1) ...} -> ({{_} ..} | _Sp)) ..}] :=
 Module[{sp$},
  DeclareVectorMomentum[tag, p,
   Sequence @@ (#[[1,
          1, ;; -2]] -> ({sp$[1],
           sp$[-1]} //. ((sp$[#[[1, -1]]] -> #[[2]]) & /@ #)) &) /@
     GatherBy[sp, #[[1, ;; -2]] &]]]

(* Remove the declaration of a momenta for a specified 'tag' *)
UndeclareVectorMomentum[tag_,
  p_] := (Remove[Evaluate[vectors$$spinors[tag, p]]];
  tag /: vectors$$spinors[tag, p] =.;
  tag /: vectors$$momentums[tag, p] =.;
  tag /: vectors$$momentums$$defined[tag, p] =.;)

(* Declare that a 'tag' inherits all momenta values defined for another
    tag *)
DeclareInheritingTag[tag_,
  parent_] := (tag /: vectors$$momentums[tag, p_] :=
   vectors$$momentums[parent, p];
  tag /: vectors$$spinors[tag, p_] := vectors$$spinors[parent, p];
  tag /: vectors$$momentums[tag, p_] :=
   vectors$$momentums$$defined[parent, p];)

(* Evaluate expressions numerically *)
Ev[d_, moms_][expr_] :=
 Module[{e = expr},
  e //. {spinor[p_, hs__Integer] :>
     vectors$$spinors[moms, p][d, hs] /;
      MatchQ[vectors$$spinors[moms, p][d, hs], {__List}],
    spinorbar[p_, hs__Integer] :>
     AdjointSpinor[d, vectors$$spinors[moms, p][d, hs], hs] /;
      MatchQ[vectors$$spinors[moms, p][d, hs], {__List}], (Sm | SmM)[
      p_, h_] :> \[Gamma]Expression[d, h,
        0] vectors$$momentums[moms, p][[1]] -
       Sum[\[Gamma]Expression[d,
          h, $$i] vectors$$momentums[moms, p][[$$i + 1]], {$$i, 1,
         Length[vectors$$momentums[moms, p]] - 1}] /;
      MatchQ[vectors$$momentums[moms, p], _List], (Mom | MomM)[p_] :>
     PadRight[vectors$$momentums[moms, p], d, 0] /;
      MatchQ[vectors$$momentums[moms, p], _List],
    Sp[pre__, \[Sigma]ma][h_], post__] :>
```

```
    Table[Sp[pre, \[Gamma]Expression[d, h, \[Mu]], post], {\[Mu], 0,
      d - 1}]}]

(* Implementations for functions that depend on the number of dimensions
    *)
ConvertHels[i_] := Sequence[-i] (* 4d *)

ConvertHels[i_, j_] := Sequence[i, -j] (* 6d *)

ConvertMidHelicities[sp_, 1] := FlipHelicities[sp]
ConvertMidHelicities[sp_, 2] := sp

SpFlipSign[{_}, {_}, n_] := HelicitySign[(-1)^(n + 1)] (* 4d *)

SpFlipSign[{_, h1_}, {_, h2_}, n_] :=
 HelicitySign[(-1)^n, h1, h2] (* 6d *)

Sp[pre___, spinorbar[p_, hh_, h_], Sm[__], spinor[p_, hh_, h_],
  post___] := 0
Sp[pre___, spinorbar[p_, hh_, h_], \[Sigma][_], spinor[p_, hh_, h_],
  post___] := 0

SpinorExpression[4, {p0_, p1_, p2_, p3_}, 1] :=
 If[TrueQ[Simplify[p0 + p1 == 0]],
  If[TrueQ[p0 - p1 == 0],
   Module[{sqrtp3 =
      Sqrt[2 p3]}, {{(p3 + I p2)/Sqrt[2 p3]}, {(p3 - I p2)/
       Sqrt[2 p3]}}],
   Module[{sqrtpm =
      Sqrt[p0 - p1]}, {{(p3 + I p2)/sqrtpm}, {sqrtpm}}]],
  Module[{sqrtpp = Sqrt[p0 + p1]}, {{sqrtpp}, {(p3 - I p2)/sqrtpp}}]]
SpinorExpression[4, {p0_, p1_, p2_, p3_}, -1] :=
 If[TrueQ[Simplify[p0 + p1 == 0]],
  If[TrueQ[p0 - p1 == 0],
   Module[{sqrtp3 =
      Sqrt[2 p3]}, {{(p3 + I p2)/Sqrt[2 p3]}, {(p3 - I p2)/
       Sqrt[2 p3]}}],
   Module[{sqrtpm =
      Sqrt[p0 - p1]}, {{sqrtpm}, {(p3 - I p2)/sqrtpm}}]],
  Module[{sqrtpp = Sqrt[p0 + p1]}, {{(p3 + I p2)/sqrtpp}, {sqrtpp}}]]
AdjointMetric[4] = {{0, -1}, {1, 0}};
AdjointSign[4, i_] := 1
```

```
CalculateMomenta[{{ppp_}, {ppmp_}}, {{pppm_}, {ppm_}}, {}] := {(ppm \
ppp + ppmp pppm)/2, (ppm ppp - ppmp pppm)/2,
  I (ppm ppmp - ppp pppm)/2, (ppm ppmp + ppp pppm)/2}

SpinorExpression[6, {p0_, p1_, p2_, p3_, p4_, p5_}, 1, 1] :=
 If[TrueQ[Simplify[p0 + p1 == 0]],
  If[TrueQ[Simplify[p0 - p1 == 0]],
   If[TrueQ[Simplify[p3 + I p2 == 0]],
    If[TrueQ[
      Simplify[
       p3 - I p2 == 0]], {{(p5 + I p4)/
        Sqrt[2 p5]}, {0}, {0}, {(p5 - I p4)/Sqrt[2 p5]}},
     Module[{sqrtppm =
        Sqrt[p3 - I p2]}, {{sqrtppm}, {0}, {0}, {-(p5 - I p4)/
         sqrtppm}}]],
    Module[{sqrtppp =
       Sqrt[p3 + I p2]}, {{sqrtppp}, {0}, {0}, {-(p5 - I p4)/
        sqrtppp}}]],
   Module[{sqrtpm =
      Sqrt[p0 - p1]}, {{(p3 + I p2)/
       sqrtpm}, {sqrtpm}, {0}, {-(p5 - I p4)/sqrtpm}}]],
  Module[{sqrtpp =
     Sqrt[p0 + p1]}, {{sqrtpp}, {(p3 - I p2)/sqrtpp}, {(p5 - I p4)/
      sqrtpp}, {0}}]]
SpinorExpression[6, {p0_, p1_, p2_, p3_, p4_, p5_}, 1, -1] :=
 If[TrueQ[Simplify[p0 + p1 == 0]],
  If[TrueQ[Simplify[p0 - p1 == 0]],
   If[TrueQ[Simplify[p3 + I p2 == 0]],
    If[TrueQ[
      Simplify[
       p3 - I p2 == 0]], {{0}, {-(p5 + I p4)/
        Sqrt[2 p5]}, {-(p5 - I p4)/Sqrt[2 p5]}, {0}},
     Module[{sqrtppm =
        Sqrt[p3 - I p2]}, {{0}, {-(p5 + I p4)/
         sqrtppm}, {sqrtppm}, {0}}]],
    Module[{sqrtppp =
       Sqrt[p3 + I p2]}, {{0}, {-(p5 + I p4)/
        sqrtppp}, {sqrtppp}, {0}}]],
   Module[{sqrtpm =
      Sqrt[p0 - p1]}, {{(p5 + I p4)/
       sqrtpm}, {0}, {sqrtpm}, {(p3 - I p2)/sqrtpm}}]],
```

```
  Module[{sqrtpp =
      Sqrt[p0 + p1]}, {{0}, {-(p5 + I p4)/sqrtpp}, {(p3 + I p2)/
        sqrtpp}, {sqrtpp}}]]
SpinorExpression[6, {p0_, p1_, p2_, p3_, p4_, p5_}, -1, 1] :=
 If[TrueQ[Simplify[p0 + p1 == 0]],
  If[TrueQ[Simplify[p0 - p1 == 0]],
   If[TrueQ[Simplify[p3 + I p2 == 0]],
    If[TrueQ[
      Simplify[
       p3 - I p2 == 0]], {{0}, {-(p5 + I p4)/
         Sqrt[2 p5]}, {-(p5 - I p4)/Sqrt[2 p5]}, {0}},
     Module[{sqrtppm =
        Sqrt[p3 - I p2]}, {{0}, {-(p5 + I p4)/
          sqrtppm}, {sqrtppm}, {0}}]],
    Module[{sqrtppp =
       Sqrt[p3 + I p2]}, {{0}, {-(p5 + I p4)/
         sqrtppp}, {sqrtppp}, {0}}]],
   Module[{sqrtpm =
      Sqrt[p0 - p1]}, {{0}, {-(p5 + I p4)/sqrtpm}, {(p3 + I p2)/
        sqrtpm}, {sqrtpm}}]],
  Module[{sqrtpp =
     Sqrt[p0 + p1]}, {{(p5 + I p4)/
       sqrtpp}, {0}, {sqrtpp}, {(p3 - I p2)/sqrtpp}}]]
SpinorExpression[6, {p0_, p1_, p2_, p3_, p4_, p5_}, -1, -1] :=
 If[TrueQ[Simplify[p0 + p1 == 0]],
  If[TrueQ[Simplify[p0 - p1 == 0]],
   If[TrueQ[Simplify[p3 + I p2 == 0]],
    If[TrueQ[
      Simplify[
       p3 - I p2 == 0]], {{(p5 + I p4)/
         Sqrt[2 p5]}, {0}, {0}, {(p5 - I p4)/Sqrt[2 p5]}},
     Module[{sqrtppm =
        Sqrt[p3 - I p2]}, {{sqrtppm}, {0}, {0}, {-(p5 - I p4)/
          sqrtppm}}]],
    Module[{sqrtppp =
       Sqrt[p3 + I p2]}, {{sqrtppp}, {0}, {0}, {-(p5 - I p4)/
         sqrtppp}}]],
   Module[{sqrtpm =
      Sqrt[p0 - p1]}, {{sqrtpm}, {(p3 - I p2)/sqrtpm}, {(p5 - I p4)/
        sqrtpm}, {0}}]],
  Module[{sqrtpp =
     Sqrt[p0 + p1]}, {{(p3 + I p2)/
```

```
      sqrtpp}, {sqrtpp}, {0}, {-(p5 - I p4)/sqrtpp}}]]
AdjointMetric[
    6] = {{0, 0, 0, 1}, {0, 0, -1, 0}, {0, 1, 0, 0}, {-1, 0, 0, 0}};
AdjointSign[6, i_, j_] := HelicitySign[i j]

CalculateMomenta6dI[pp_, pm_, ppp_, ppm_, pdp_, pdm_,
   h_] := {pm + pp, pp - pm, I (ppm - ppp), ppm + ppp, h I (pdm - pdp),
     pdm + pdp}/2
CalculateMomenta[{{sp1_}, {sp2_}, {sp3_}, {sp4_}}, {{sm1_}, {sm2_}, \
{sm3_}, {sm4_}}, {h : 1}] :=
 CalculateMomenta6dI[sm4 sp1 - sm1 sp4, sm3 sp2 - sm2 sp3,
  sm3 sp1 - sm1 sp3, sm4 sp2 - sm2 sp4, -sm2 sp1 + sm1 sp2,
  sm4 sp3 - sm3 sp4, h]
CalculateMomenta[{{sp3_}, {sp4_}, {sp1_}, {sp2_}}, {{sm3_}, {sm4_}, \
{sm1_}, {sm2_}}, {h : -1}] :=
 CalculateMomenta6dI[sm4 sp1 - sm1 sp4, sm3 sp2 - sm2 sp3,
  sm3 sp1 - sm1 sp3, sm4 sp2 - sm2 sp4, -sm2 sp1 + sm1 sp2,
  sm4 sp3 - sm3 sp4, h]

(* The polarisation vector in any number of dimensions. The number of
    dimensions is set by the number of signs given as each of i__ and
    j__ *)
PolVec[p_, q_, i__, j__] :=
 Sp[spinorbar[p, i], \[Gamma], Sm[q], spinor[p, j]]/2^(3/2)/
   Mp[Mom[p], Mom[q]] /; Length[{i}] == Length[{j}]
PolVec[p_, q_, i__, j__, \[Mu]_] :=
 Sp[spinorbar[p, i], \[Gamma][\[Mu]], Sm[q], spinor[p, j]]/2^(3/2)/
   Mp[Mom[p], Mom[q]] /; Length[{i}] == Length[{j}]

(* Replacements for reducing 6 dimensional momenta to 4 dimensional
    momenta and the reverse *)
rep = {spinor[pp : p | q | n6p | n6m | shift[{p, q}, z][p | q], h_,
      l_] :> spinor[pp[a], h, l] -
      HelicitySign[h,
        l] (pp[5] -
         I HelicitySign[h, l] pp[4]) Sp[Sm[a, -h],
          spinor[pp[a], -h, l]]/2/Mp[Mom[a], Mom[pp[a]]],
    spinorbar[pp : p | q | n6p | n6m | shift[{p, q}, z][p | q], h_,
      l_] :> spinorbar[pp[a], h, l] +
      HelicitySign[h,
        l] (pp[5] -
         I HelicitySign[h, l] pp[4]) Sp[spinorbar[pp[a], -h, l],
```

```
          Sm[a, -h]]/2/Mp[Mom[a], Mom[pp[a]]],
    Mom[pp : p | q | n6p | n6m | shift[{p, q}, z][p | q]] :>
     Mom[pp[a]] + Mom[a] (pp[4]^2 + pp[5]^2)/2/Mp[Mom[pp[a]], Mom[a]] +
       Sum[h Sp[spinorbar[pp[a], 1, h], \[Sigma][1], Sm[a, -1],
           spinor[pp[a], -1, -h]] (pp[5] + I h pp[4]), {h, {1, -1}}]/4/
       Mp[Mom[pp[a]], Mom[a]],
    Sm[pp : p | q | n6p | n6m | shift[{p, q}, z][p | q], h_] :>
     Sp[Sm[pp[a], h]] +
      Sum[l HelicitySign[
           h] (pp[5] -
            I l HelicitySign[h] pp[4]) (Sp[spinor[pp[a], -h, l],
             spinorbar[pp[a], h, -l], Sm[a, h]] -
            Sp[Sm[a, h], spinor[pp[a], h, -l],
             spinorbar[pp[a], -h, l]] ), {l, {1, -1}}]/2/
       Mp[Mom[pp[a]], Mom[a]] + (pp[4]^2 + pp[5]^2) Sp[Sm[a, h]]/2/
        Mp[Mom[pp[a]], Mom[a]]};
reph[ll_] := {spinor[pp : p | q | n6p | n6m | shift[{p, q}, z][p | q],
       h_, l_] :>
     spinor[pp[a], h, l] -
      HelicitySign[h,
        l] (pp[5] -
         I HelicitySign[h, l] pp[4]) Sp[Sm[a, -h],
          spinor[pp[a], -h, l]]/2/Mp[Mom[a], Mom[pp[a]]],
    spinorbar[pp : p | q | n6p | n6m | shift[{p, q}, z][p | q], h_,
      l_] :> spinorbar[pp[a], h, l] +
      HelicitySign[h,
        l] (pp[5] -
         I HelicitySign[h, l] pp[4]) Sp[spinorbar[pp[a], -h, l],
          Sm[a, -h]]/2/Mp[Mom[a], Mom[pp[a]]],
    Mom[pp : p | q | n6p | n6m | shift[{p, q}, z][p | q]] :>
     Mom[pp[a]] + Mom[a] (pp[4]^2 + pp[5]^2)/2/Mp[Mom[pp[a]], Mom[a]] +
       Sum[HelicitySign[h] Sp[spinorbar[pp[a], 1, h], \[Sigma][1],
           Sm[a, -1],
           spinor[pp[a], -1, -h]] (pp[5] +
            I HelicitySign[h] pp[4]), {h, {ll, -ll}}]/4/
       Mp[Mom[pp[a]], Mom[a]],
    Sm[pp : p | q | n6p | n6m | shift[{p, q}, z][p | q], h_] :>
     Sp[Sm[pp[a], h]] +
      Sum[HelicitySign[l] HelicitySign[
           h] (pp[5] -
            I HelicitySign[l] HelicitySign[h] pp[4]) (Sp[
             spinor[pp[a], -h, l], spinorbar[pp[a], h, -l], Sm[a, h]] -
```

```
              Sp[Sm[a, h], spinor[pp[a], h, -1],
               spinorbar[pp[a], -h, 1]] ), {1, {11, -11}}]/2/
        Mp[Mom[pp[a]], Mom[a]] + (pp[4]^2 + pp[5]^2) Sp[Sm[a, h]]/2/
         Mp[Mom[pp[a]], Mom[a]]};
unrep = {spinor[(pp : p | q | shift[{p, q}, z][p | q])[a], h_, l_] :>
    spinor[pp, h, l] +
     HelicitySign[h,
       l] (pp[5] -
        I HelicitySign[h, l] pp[4]) Sp[Sm[a, -h], spinor[pp, -h, l]]/
        2/Mp[Mom[a], Mom[pp]],
   spinorbar[(pp : p | q | shift[{p, q}, z][p | q])[a], h_, l_] :>
    spinorbar[pp, h, l] -
     HelicitySign[h,
       l] (pp[5] -
        I HelicitySign[h, l] pp[4]) Sp[spinorbar[pp, -h, l],
         Sm[a, -h]]/2/Mp[Mom[a], Mom[pp]],
   Mom[(pp : p | q | shift[{p, q}, z][p | q])[a]] :>
    Mom[pp] + Mom[a] (pp[4]^2 + pp[5]^2)/2/Mp[Mom[pp[a]], Mom[a]] -
     Sum[h Sp[spinorbar[pp, 1, h], \[Sigma][1], Sm[a, -1],
          spinor[pp, -1, -h]] (pp[5] + I h pp[4]), {h, {1, -1}}]/4/
       Mp[Mom[pp], Mom[a]],
   Sm[(pp : p | q | shift[{p, q}, z][p | q])[a], h_] :>
    Sp[Sm[pp, h]] -
     Sum[l HelicitySign[
          h] (pp[5] -
            I l HelicitySign[h] pp[4]) (Sp[spinor[pp, -h, l],
              spinorbar[pp, h, -l], Sm[a, h]] -
             Sp[Sm[a, h], spinor[pp, h, -l],
              spinorbar[pp, -h, l]] ), {l, {1, -1}}]/2/
       Mp[Mom[pp], Mom[a]] + (pp[4]^2 + pp[5]^2) Sp[Sm[a, h]]/2/
        Mp[Mom[pp], Mom[a]]};

tomu = {(pp : p | q)[
     4] :> (I/2)*(Subsuperscript[\[Mu], pp, "-"] -
        Subsuperscript[\[Mu], pp, "+"]), (pp : p | q)[
     5] :> (Subsuperscript[\[Mu], pp, "-"] +
        Subsuperscript[\[Mu], pp, "+"])/2};
tomuh[h_] := {(pp : p | q)[4] :> (I/2)*
     HelicitySign[
      h] (Subsuperscript[\[Mu], pp, -h] -
       Subsuperscript[\[Mu], pp, h]), (pp : p | q)[
    5] :> (Subsuperscript[\[Mu], pp, -h] +
```

```
        Subsuperscript[\[Mu], pp, h])/2}
tomuhp[h_] := {(pp : p | q)[4] :> (I/2)*
      HelicitySign[
       h[pp]] (Subsuperscript[\[Mu], pp, -h[pp]] -
        Subsuperscript[\[Mu], pp, h[pp]]), (pp : p | q)[
      5] :> (Subsuperscript[\[Mu], pp, -h[pp]] +
        Subsuperscript[\[Mu], pp, h[pp]])/2}
frommu = {Subsuperscript[\[Mu], p_, "+"] :> p[5] + I p[4],
  Subsuperscript[\[Mu], p_, "-"] :> p[5] - I p[4],
  Subsuperscript[\[Mu], p_, h_] :> p[5] + I HelicitySign[h] p[4]}
```