



Universiteit
Leiden
The Netherlands

Advances in computational methods for Quantum Field Theory calculations

Ruijl, B.J.G.

Citation

Ruijl, B. J. G. (2017, November 2). *Advances in computational methods for Quantum Field Theory calculations*. Retrieved from <https://hdl.handle.net/1887/59455>

Version: Not Applicable (or Unknown)

License: [Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden](#)

Downloaded from: <https://hdl.handle.net/1887/59455>

Note: To cite this publication please use the final published version (if applicable).

Cover Page



Universiteit Leiden



The following handle holds various files of this Leiden University dissertation:

<http://hdl.handle.net/1887/59455>

Author: Ruijl, B.J.G.

Title: Advances in computational methods for Quantum Field Theory calculations

Issue Date: 2017-11-02

In this chapter we focus on answering

RQ2: *How can we construct a program that can compute four-loop massless propagator integrals more efficiently?*

Over the years particle physics experiments have become more and more precise. This creates the need for more accurate calculations of the underlying processes. Quantum Field Theory (QFT) has proven to be a successful framework for making predictions about scattering experiments in particle accelerators such as the Large Hadron Collider (LHC). Especially Quantum Chromodynamics (QCD), the theory that describes the strong nuclear force mediated by gluons, is essential. One key feature of QCD is that interactions between particles can be described as a perturbative series, where every order improves the accuracy. Since the coupling constant is relatively large, higher order calculations are not negligible. For processes such as the production of Higgs bosons, three-loop calculations have recently been performed [81, 82]. This in turn necessitates the evaluation of the four-loop splitting functions to determine the parton distributions inside the proton. A complete calculation of the four-loop splitting functions is currently out of the question, in part due to the complexity of the integral reductions. The next best solution is to evaluate a number of Mellin moments as was done at the three-loop level over the past 25 years [83–87]. One way to obtain such moments is by converting them to integrals of a massless propagator nature by expansions in terms of the parton momentum. The computer program that could deal with the resulting three-loop integrals is called MINCER [32, 33] and its algorithms are based on Integration by Parts (IBP) identities [43]. To obtain higher moments MINCER has been optimised heavily. This eventually resulted in an $N = 29$ moment calculation for polarised scattering [88].

The construction of a similar program for four-loop propagator integrals is a far more formidable task. The attempts to solve the problem have led to the exploration of different techniques, such as the $1/D$ expansions of Baikov [89–91]. Instead of solving the systems of IBP equations parametrically as was done in MINCER, Laporta developed a method to solve the system by substituting values for its parameters [42]. This method has been used to create generic programs that can handle integrals in a flexible way [92–96]. The drawback of these programs is that it is in essence a brute-force Gaussian reduction, that needs to reduce many subsystems that will drop out of the final answer. An extra complication is the fact that the system is riddled with ‘spurious poles’ which are powers in $1/\epsilon = 2/(4 - D)$ that only cancel by the time all contributions to the coefficient of a master integral have been added. If it is not known in advance how many of these spurious poles will occur, one cannot safely perform a fixed expansions in ϵ . In the three-loop MINCER program

the spurious poles could be avoided thanks to the resolved triangle formula by Tkachov [97], but for the all- N calculation these spurious poles caused significant issues [34, 35, 98]. In general, spurious pole avoidance is considered too complicated and is resorted to the very slow but exact arithmetic of rational polynomials.

A method capable for a parametric reduction of massless four-loop propagator integrals has been developed by Roman Lee in a series of papers [99, 100]. It resulted in the LITERED program, which is a MATHEMATICA package that constructs reduction programs (also in MATHEMATICA code). Although it is extremely elegant and as a method powerful, the resulting four-loop propagator programs are too slow for most practical applications (reductions take months or years).

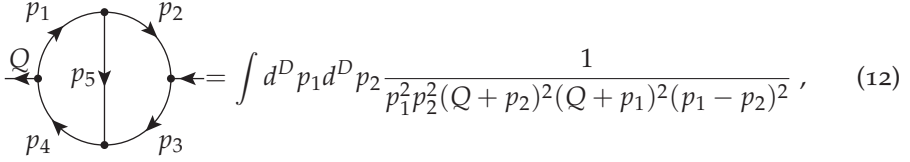
In this chapter we describe FORCER, a FORM [101, 102] program that is a hybrid between various approaches. We discuss the construction of a precomputed reduction graph that describes how to reduce each topology and how to map topologies with missing propagators into each other. Most topologies have straightforward reduction rules due to known reducible substructures, such as triangles or insertions. However, 21 topologies require a custom reduction rule (see section 3.6). Our research methodology consists of three steps. (1) We construct the diamond rule, which is able to efficiently reduce more topologies. Next, (2) we provide several heuristics for custom reductions. Finally, (3) we construct a program that applies the reduction rules to every topology.

The remainder of this chapter is structured as follows. In section 3.1 we define our Feynman diagrams. We discuss the reductions of simple substructures in sections 3.2 (the insertion rule) and 3.3 (the carpet rule). Next, we introduce Integration by Parts identities (IBPs) and construct the diamond rule in section 3.4 (step 1). In section 3.5 we show how to solve parametric IBPs by using heuristics (step 2). Section 3.6 lists all 21 topologies that are considered to be irreducible by lacking simple substructures. These involve the master integrals and a few more topologies that cannot be resolved in a single reduction step. In section 3.7 the superstructure of the program and its derivation are described (step 3). The usage of the program is discussed in section 3.8. In section 3.9 we show how to use expansions with FORCER. We describe how to transform physical diagrams to input for FORCER in section 3.10. In section 3.11 we show examples and we study the performance. Finally, section 3.12 provides the chapter conclusion.

3.1 GENERALISED FEYNMAN DIAGRAMS

We start by describing the objects we would like to compute. Each scattering process requires the computation of Feynman integrals, which have an intuitive graph representation. The order of perturbations is captured by the number of

fundamental cycles ('loops') in the graph. An example of a two-loop Feynman diagram with its corresponding integral is



$$\int d^D p_1 d^D p_2 \frac{1}{p_1^2 p_2^2 (Q + p_2)^2 (Q + p_1)^2 (p_1 - p_2)^2} , \quad (12)$$

where D is the dimension. Each propagator is rewritten in terms of the momentum basis $\{Q, p_1, p_2\}$. For physical calculations, the integrals should be computed in four dimensions, but sadly many integrals are divergent in $D = 4$. To regulate the infinities, which we should subtract at some point, we shift the space-time dimension to $D = 4 - 2\varepsilon$, where ε is a small positive number.

The diagram in eq. (12) has two loops and is scalar. In QCD, each propagator (line) and vertex are functions, defined by Feynman rules, that can be rewritten into a sum of scalar diagrams. For now we assume that the Feynman rules have been applied, so that we are now concerned with computing diagrams of the form:

$$F = \int d^D l_1 \cdots d^D l_m \frac{(p_{r_1} \cdot p_{r_2})^{n_{r_{12}}} \cdots (p_{r_3} \cdot p_{r_4})^{n_{r_{34}}}}{(p_1 \cdot p_1)^{n_1} \cdots (p_k \cdot p_k)^{n_k}} , \quad (13)$$

where D is the dimension, l_i are loop momenta, and p_i are momenta of propagators. Potentially, the powers of the propagators n_i contain powers of ε . These diagrams with arbitrary powers on the numerator are not physical, but can be obtained from them by rewriting momenta. Thus we consider a class of generalized Feynman diagrams.

All dot products can always be rewritten into a basis. For an L -loop diagram with K external momenta, the basis size is $(L + 1)L/2 + LK$. If the number of edges is E , then there are $(L + 1)L/2 + LK - E$ irreducible numerators for that topology.

In order to compute these diagrams, we shall rewrite them into a sum over simpler integrals. A simpler integral is an integral with fewer number of lines or loops, or with fewer numerators. Especially a reduction in the number of loops makes the integral easier, as the degrees of freedom are lowered.

An example of a simplification of the numerator structure is given below:

$$\begin{aligned} \int d^D p \frac{p \cdot Q}{p^2 (p + Q)^2} &= \int d^D p \frac{1}{2} \frac{(p + Q)^2 - p^2 - Q^2}{p^2 (p + Q)^2} \\ &= \int d^D p \frac{1}{2} \frac{1}{p^2} + \int d^D p \frac{1}{2} \frac{1}{(p + Q)^2} + \int d^D p \frac{1}{2} \frac{-Q^2}{p^2 (p + Q)^2} \\ &= \frac{-Q^2}{2} \int d^D p \frac{1}{p^2 (p + Q)^2} , \end{aligned} \quad (14)$$

where the first two terms are dropped since they are massless tadpoles. The last term has no numerator structure and is thus considered simpler.

In the following sections we describe simplifications of integrals that can always be performed if the integral has a certain substructure.

3.2 INTEGRATION OF ONE-LOOP TWO-POINT FUNCTIONS

If the Feynman graph has a massless one-loop two-point function, as shown in figure 16, the bubble can be integrated. This structure is also referred to as an *insertion*.

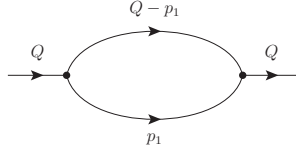


Figure 16: A massless one-loop two-point function, also referred to as an insertion.

An efficient formula that works for an insertion with an arbitrary numerator structure and arbitrary (including non-integer) powers for the propagators is the following formula [103]:

$$\int \frac{d^D P}{(2\pi)^D} \frac{\mathcal{P}_n(P)}{P^{2\alpha}(P-Q)^{2\beta}} = \frac{1}{(4\pi)^2} (Q^2)^{D/2-\alpha-\beta} \sum_{\sigma=0}^{[n/2]} G(\alpha, \beta, n, \sigma) Q^{2\sigma} \left\{ \frac{1}{\sigma!} \left(\frac{\square}{4} \right)^\sigma \mathcal{P}_n(P) \right\}_{P=Q}, \quad (15)$$

where

$$\mathcal{P}_n(P) = P_{\mu_1} P_{\mu_2} \cdots P_{\mu_n}, \quad \square = \partial^2 / \partial P_\mu \partial P_\mu, \quad (16)$$

and G can be expressed in terms of Γ -functions:

$$G(\alpha, \beta, n, \sigma) = (4\pi)^\epsilon \frac{\Gamma(\alpha + \beta - \sigma - D/2) \Gamma(D/2 - \alpha + n - \sigma) \Gamma(D/2 - \beta + \sigma)}{\Gamma(\alpha) \Gamma(\beta) \Gamma(D - \alpha - \beta + n)}. \quad (17)$$

The function G is normalised to a function G in which (1) the powers of the denominators are one plus potentially a multiple of ϵ and (2) there are no numerators. The difference is a number of Pochhammer symbols in ϵ which can either be expressed as rational polynomials or can be expanded in terms of ϵ , depending on what is needed. When finite expansions are used it is easy to generate tables of these Pochhammer symbols. The remaining function G is basically part of the master integral and kept for the end of the program when the master integrals are substituted.

In the presence of powers of the loop momentum P in the numerator, it is much faster to apply the above formula than to use IBP identities to lower the numerator. This holds in particular when one works with the FORM system, because its instruction set allows the evaluation of powers of the d'Alembertians with perfect efficiency.

It means that each term will be generated with the proper combinatoric factor and hence never gets generated more than once. The latter is owing to the combinatoric functions `distrib_` and `dd_`. The code is displayed in listing 3.1.

LISTING 3.1: FORM code for one-loop insertion

```

Tensor Ptensor,del;
Vector P,Q,p1,p2,p3,p4;
Symbols dAlembertian,j;
Local F = dAlembertian^15*P.p1^15*P.p2^15*P.p3^15*P.p4^15;
ToTensor,P,Ptensor;
id dAlembertian^j?*Ptensor(?a) = distrib_(1,2*j,del,Ptensor,?a);
ToVector,Ptensor,Q;
id del(?a) = dd_(?a);
Print +f +s;
.end

Time = 3.09 sec Generated terms = 1133616
      F Terms in output = 1133616
      Bytes used = 140937744

F =
+ 3092470075094400000*Q.p1*Q.p2*Q.p3^13*Q.p4^15*p1.p1*
p1.p2^10*p1.p3^2*p2.p2^2
.....
+ 1451044943048200500000*Q.p1^7*Q.p2^10*Q.p3^13*p1.p1^3*
p1.p4^2*p2.p2*p2.p3^2*p2.p4*p4.p4^6
etc.

```

When computing Mellin moments of structure functions (see 4.3), the efficient combinatoric functions described above are essential, since they are used both in the one-loop integrals and the harmonic projections.

Two drawbacks of the approach is that (1) one may need to rewrite the dot products or invariants in the numerator to such a form that they are usable for the above formula. (2) The result of integrating out the insertion is that the remaining graph edge will have a non-integer power. Since the two-point function was embedded in a larger diagram, this edge may be internal. Consequently, this line can never be reduced to zero again by IBP identities.

3.3 CARPET RULE

For integrals where a subgraph is embedded in an outer one-loop graph, scaling and Lorenz invariance arguments [43] allow us to integrate out the outer one first. In figure 17, we show this structure.

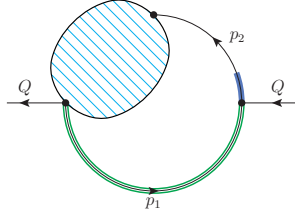


Figure 17: An subgraph (blob) embedded in a one-loop graph. The green triple line can be integrated out, regardless of the content of the shaded blob.

The generalized rule that works for these topologies with a generic numerator structure, we call the ‘carpet’ rule:

$$\begin{aligned}
 & \int \frac{d^D p}{(2\pi)^D} \frac{1}{(p^2)^\alpha [(p-q)^2]^\beta} \left[\prod_{i=1}^{L_{\text{sub}}} \int \frac{d^D l_i}{(2\pi)^D} \right] \left[\prod_{i=1}^{N_{\text{sub}}} \frac{1}{(p_i^2)^{a_i}} \right] \mathcal{P}_n(\{p_i\}, q) \\
 &= \frac{1}{(4\pi)^2} (q^2)^{D/2-\alpha-\beta} \sum_{\sigma=0}^{\lfloor n/2 \rfloor} \left(\frac{D}{2} + n - \sigma \right)_{-\sigma} \\
 & \quad \times G \left(\alpha + \sum_{i=1}^{N_{\text{sub}}} a_i - \frac{D}{2} L_{\text{sub}} - \sigma, \beta, n - 2\sigma, 0 \right) \\
 & \quad \times \sum_{j=0}^{\lfloor n/2-\sigma \rfloor} (-1)^j \left(\frac{D}{2} + n - 2\sigma - 1 \right)_{-j} (q^2)^{\sigma+j} \\
 & \quad \times \left[\prod_{i=1}^{L_{\text{sub}}} \int \frac{d^D l_i}{(2\pi)^D} \right] \left[\prod_{i=1}^{N_{\text{sub}}} \frac{1}{(p_i^2)^{a_i}} \right] \left[\frac{1}{\sigma! j!} \left(\frac{\square_q}{4} \right)^{\sigma+j} \mathcal{P}_n(\{p_i\}, q) \right]_{p=q}.
 \end{aligned} \tag{18}$$

Here L_{sub} is the number of loops in the embedded subgraph. The integrand of the subgraph consists of two parts: a product of $1/(p_i^2)^{a_i}$ and $\mathcal{P}_n(\{p_i\}, q)$. Each p_i^2 indicates not only a squared propagator in the subgraph but also any quadratic Lorentz scalar that becomes p^2 after the integrations in the subgraph, e.g., $p_i \cdot p_j$ and $p_i \cdot p$. If $\mathcal{P}_n(\{p_i\}, q) = 1$ (and $n = 0$), the formula just describes that the knowledge of the dimension of the subgraph is sufficient to write down the result of the outer loop integral. In general, $\mathcal{P}_n(\{p_i\}, q)$ is a polynomial with degree n both in p_i and q , which are taken as dot products of $p_i \cdot q$ in FORCER. In the right-hand side of the formula, $(x)_n = \Gamma(x+n)/\Gamma(x)$ is the Pochhammer symbol and the function G is given by eq. (17). The d’Alembertian $\square_q = \partial^2/\partial q_\mu \partial q^\mu$ can be efficiently implemented by `distrib_` and `dd_` functions in FORM as explained in sec. 3.2.

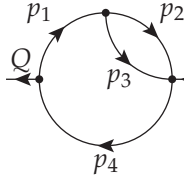
3.4 INTEGRATION BY PARTS

A whole class of simplification identities can be derived from the following rule stemming from partial integration of any Feynman integral F [43]:

$$\frac{\partial}{\partial p_i} p_j \circ F = 0, \quad (19)$$

where the operator \circ means that the preceding factors should be applied to the integrand of F (so before integration). Here we remark that p_i and p_j can be any momentum. However, the equation will yield different relations only if p_i and p_j are part of the same loop momentum basis, or p_j is the external momentum. Thus, in total there are $L(L + K)$ independent partial integration equations.

Let us consider the following (generalised) diagram:



$$= \int d^D p_1 d^D p_2 \frac{1}{(p_1^2)^{n_1} (p_2^2)^{n_2} (p_1 - p_2)^{2n_3} (Q + p_1)^{2n_4}}, \quad (20)$$

to which we apply the IBP identity $Q^\mu \frac{\partial}{\partial p_2^\mu}$:

$$\begin{aligned} Q^\mu \frac{\partial}{\partial p_2^\mu} \circ F &= \int d^D p_1 d^D p_2 Q^\mu \frac{\partial}{\partial p_2^\mu} \left[\frac{1}{(p_1^2)^{n_1} (p_2^2)^{n_2} (p_1 - p_2)^{2n_3} (Q + p_1)^{2n_4}} \right] \\ &= \int d^D p_1 d^D p_2 \left[-2n_2 \frac{Q \cdot p_2}{(p_1^2)^{n_1} (p_2^2)^{n_2+1} (p_3^2)^{n_3} (p_4^2)^{n_4}} - 2n_3 \frac{Q \cdot p_1 - Q \cdot p_2}{(p_1^2)^{n_1} (p_2^2)^{n_2} (p_3^2)^{n_3+1} (p_4^2)^{n_4}} \right] \\ &= \left[-2n_2 \frac{Q \cdot p_2}{p_2^2} + 2n_3 \frac{Q \cdot p_2}{p_3^2} - n_3 \frac{p_4^2}{p_3^2} + n_3 \frac{p_1^2}{p_3^2} + n_3 \frac{Q^2}{p_3^2} \right] \circ F = 0, \end{aligned} \quad (21)$$

where we have suppressed the integration and written the diagram in the basis $\{p_1^2, p_2^2, p_3^2, p_4^2, 2Q \cdot p_2\}$. For each index, described by n_i , we introduce a raising and lowering operator N_i^+ and N_i^- (capitalising the index letter and making it bold), which raise or lower the power of the index. By definition, we consider the powers of the numerator to be negative. Thus, the raising operator on a numerator will decrease its power. We get:

$$\left[-n_2 N_5^+ N_2^+ + n_3 \left(N_5^+ N_3^+ - N_4^- N_3^+ + N_1^- N_3^+ + Q^2 N_3^+ \right) \right] F = 0. \quad (22)$$

Alternatively, we can write:

$$\begin{aligned} &n_2 F(n_1, n_2 + 1, n_3, n_4, n_5 + 1) + n_3 F(n_1, n_2, n_3 + 1, n_4, n_5 + 1) \\ &- n_3 F(n_1, n_2, n_3 + 1, n_4 - 1, n_5) - n_3 F(n_1 - 1, n_2, n_3 + 1, n_4, n_5) \end{aligned}$$

$$+ n_3 Q^2 F(n_1, n_2, n_3 + 1, n_4, n_5) = 0. \quad (23)$$

Systems of IBP identities can be solved such that repeated application of a rule always decreases one of the lines to zero. Below we show two IBP identities that can always be applied if the integral has a certain substructure. In section 3.4.1 we discuss the rule of the triangle, and in section 3.4.2 we construct the diamond rule. Finally, we sketch the construction of custom rules in section 3.4.3.

3.4.1 The rule of the triangle

Since the 1980s, the so-called *triangle rule* [43, 103] has been used for removing a propagator line from diagrams corresponding to a certain class of integrals. Any topology that has the following substructure can be simplified using the triangle rule:

$$F(a_1, a_2, b, c_1, c_2) = \int d^D k \frac{k^{\mu_1} \dots k^{\mu_N}}{[(k + p_1)^2 + m_1^2]^{a_1} [(k + p_2)^2 + m_2^2]^{a_2} (k^2)^b (p_1^2 + m_1^2)^{c_1} (p_2^2 + m_2^2)^{c_2}}, \quad (24)$$

where b, c_1, c_2 are positive integers. The diagram corresponding to this integral is shown in fig. 18.

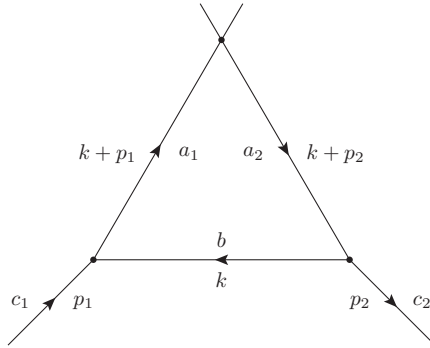


Figure 18: A triangle subtopology where the loop momentum k is assigned to the central line. p_1 and p_2 are external momenta. a_1, a_2, b, c_1 , and c_2 represent the powers of their associated propagators.

We write out the IBP relation $\frac{\partial}{\partial k_\mu} k_\mu \circ F = 0$ to obtain

$$1 = \frac{1}{D + N - a_1 - a_2 - 2b} \left[a_1 A_1^+ (B^- - C_1^-) + a_2 A_2^+ (B^- - C_2^-) \right], \quad (25)$$

where again A_i^+ , B^- , and C_i^- are operators acting on an integral that increase the power a_i by one, decrease the power b by one, and decrease the power c_i by one, respectively. Numerators that are expressed in dot products of k and an external line,

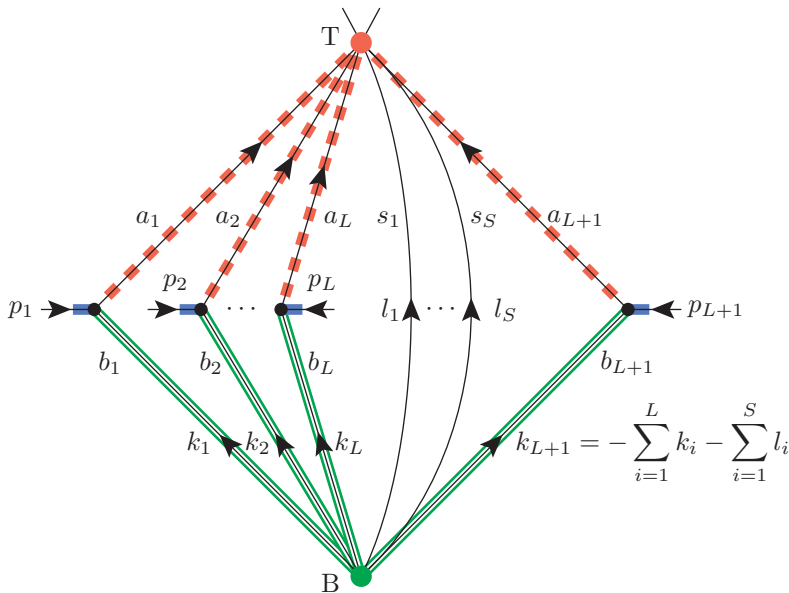


Figure 19: $(L + S)$ -loop diamond-shaped diagram. $(L + 1)$ -lines have external connections and S -lines do not. Red with dashed lines, green with double lines, and blue with thick lines represent upper, lower, and external lines of the diamond, respectively. Label T represents the top vertex, and B the bottom vertex. k_i , p_i , and l_i are momenta, and a_i , b_i , and s_i are the powers of their associated propagators.

contribute as a constant N to the rule. The rule of the triangle can be recursively applied to remove one of the propagators associated with k , p_1 , or p_2 from the system.

The recursion in the triangle rule can be explicitly ‘solved’ [97], such that the solution is expressed as a linear combination of integrals for which either b , c_1 , or c_2 is 0. The advantage of the summed system over the recursion is that it generates fewer intermediate terms and it cannot have *spurious poles*: terms in which the factor $D + N - a_1 - a_2 - 2b$ becomes proportional to ϵ more than once during the full recursion.

3.4.2 The rule of the diamond

Next, we derive a rule for a new substructure we call a *diamond*. Consider the following family of Feynman integrals in D -dimensions arising from the $(L + S)$ -loop diagram in Fig. 19:

$$F(\{a_i\}, \{b_i\}) = \left[\prod_{i=1}^L \int d^D k_i \right] \left[\prod_{i=1}^S \int d^D l_i \right]$$

$$\times \left[\prod_{i=1}^{L+1} \frac{k_i^{\mu_1^{(i)}} \dots k_i^{\mu_{N_i}^{(i)}}}{[(k_i + p_i)^2 + m_i^2]^{a_i} (k_i^2)^{b_i}} \right] \left[\prod_{i=1}^S \frac{l_i^{\nu_1^{(i)}} \dots l_i^{\nu_{R_i}^{(i)}}}{(l_i^2)^{s_i}} \right]. \quad (26)$$

The diagram consists of $(L + 1)$ paths from the top vertex T to the bottom vertex B with an external connection in between, and S lines without external connections. The upper, lower, and external lines of the diamond are represented by red with dashed lines, green with double lines, and blue with thick lines, respectively. The lines without external connections, we call *spectator lines*. In principle any pair of spectator lines can be seen as a two point function which can be reduced to a single line by integration. This line would then have a power that is not an integer. Depending on the complete framework of the reductions this may or may not be desirable. Hence we leave the number of spectators arbitrary. In any case, the contribution of the spectators is a constant (see below), which allows us to characterise integrals in the family only by $2(L + 1)$ indices a_i and b_i , and not by s_i . Without loss of generality, we assign loop momenta k_i to the lower lines of the diamond as well as l_i to the spectator lines, except the last diamond line which is fixed by momentum conservation:

$$k_{L+1} = - \sum_{i=1}^L k_i - \sum_{i=1}^S l_i. \quad (27)$$

In contrast, we do not require any constraints on the momentum conservation at the top vertex in the arguments below, hence any number of external lines can be attached to this point. In the middle of the diamond, external lines with momentum p_i are attached by three-point vertices. The upper lines in the diamond may have masses m_i , whereas the lower lines in the diamond and the spectator lines have to be massless. In addition, we allow arbitrary tensor structures of k_i and l_i with homogeneous degrees N_i and R_i , respectively, in the numerator.

Constructing the IBP identity corresponding to the operator

$$\sum_{i=1}^L \frac{\partial}{\partial k_i} \cdot k_i + \sum_{i=1}^S \frac{\partial}{\partial l_i} \cdot l_i, \quad (28)$$

straightforwardly gives the following operator identity:

$$(L + S)D + \sum_{i=1}^{L+1} (N_i - a_i - 2b_i) + \sum_{i=1}^S (R_i - 2s_i) = \sum_{i=1}^{L+1} a_i A_i^+ [B_i^- - (p_i^2 + m_i^2)]. \quad (29)$$

Here A_i^+ and B_i^- are understood as operators increasing a_i and decreasing b_i by one, respectively, when acting on $F(\{a_i\}, \{b_i\})$. Note that operators changing the spectator indices s_i are absent in the identity.

For a typical usage of eq. (29), one may identify a diamond structure as a subgraph in a larger graph. If the line with the momentum p_i has the same mass m_i as the corresponding upper line, the term $(p_i^2 + m_i^2)$ in the identity reads as an operator C_i^-

decreasing the corresponding index c_i of the power of the propagator $(p_i^2 + m_i^2)^{-c_i}$ in the larger graph by one. Applying the rule

$$1 = \frac{1}{E} \sum_{i=1}^{L+1} a_i A_i^+ (B_i^- - C_i^-), \quad (30)$$

where

$$E = (L + S)D + \sum_{i=1}^{L+1} (N_i - a_i - 2b_i) + \sum_{i=1}^S (R_i - 2s_i), \quad (31)$$

decreases $\sum_{i=1}^{L+1} (b_i + c_i)$ of integrals appearing in the right-hand side, at the cost of increasing $\sum_{i=1}^{L+1} a_i$. Starting from positive integer indices b_i and c_i , one can repeatedly use the rule until one of either b_i or c_i is reduced to zero.¹

The above diamond rule contains the conventional triangle rule as a special case. For the one-loop case $L = 1$ and $S = 0$, the two lower lines may be identified as a single line and the triangle integral in eq. (24) can be reproduced. Correspondingly, the IBP identity (30) becomes eq. (25).

Below we will (A) derive an explicit summation formula for the recursion in the diamond rule. Then (B) we will provide four examples of the diamond structure.

(A) Summation rule

We start by considering the possible connectivities in the diamond structure. If we allow for some external lines to be directly connected to each other, we get at least one triangle that can be used for the triangle rule: suppose the external momenta of k_i and of k_j are connected and identified with p_{ij} , then this triangle is k_i, k_j, p_{ij} . In this case, the triangle rule generates fewer terms and is preferred to the diamond rule. Thus, we only consider the case where the diamond does not have direct connections of external lines.

We follow the same procedure as outlined in [97]. First, we rewrite eq. (30) as:

$$F = \left[\sum_{i=1}^{L+1} a_i A_i^+ (B_i^- - C_i^-) \right] E^{-1} F, \quad (32)$$

where E is the operator $(L + S)D + \sum_{i=1}^{L+1} (N_i - a_i - 2b_i) + \sum_{i=1}^S (R_i - 2s_i)$. We split our solution in two classes $A_i^+ B_i^-$ and $A_i^+ C_i^-$ satisfying

$$E^{-1}(A_i^+ B_i^-) = (A_i^+ B_i^-)(E + 1)^{-1}, \quad E^{-1}(A_i^+ C_i^-) = (A_i^+ C_i^-)(E - 1)^{-1}. \quad (33)$$

We identify the first class with the label $+$, since it increases E by 1, and the latter with the label $-$, since it decreases E by 1. The remaining part of the derivation is analogous to the one in [97].

¹ Note that a_i are allowed to be non-integers provided the denominator in the right-hand side of eq. (30) never vanishes.

Finally, we obtain the explicit summation formula:

$$\begin{aligned}
F(\{a_i\}, \{b_i\}, \{c_i\}) = & \sum_{r=1}^{L+1} \left[\left(\prod_{\substack{i=1 \\ i \neq r}}^{L+1} \sum_{k_i^+=0}^{b_i-1} \right) \left(\prod_{i=1}^{L+1} \sum_{k_i^-=0}^{c_i-1} \right) (-1)^{k^-} \frac{k_r^+ (k^+ + k^- - 1)!}{\prod_{i=1}^{L+1} k_i^+! k_i^-!} (E + k^+)_{-k^+ - k^-} \right. \\
& \times \left. \left(\prod_{i=1}^{L+1} (a_i)_{k_i^+ + k_i^-} \right) F(\{a_i + k_i^+ + k_i^-\}, \{b_i - k_i^+\}, \{c_i - k_i^-\}) \right]_{k_r^+ = b_r} \\
& + \sum_{r=1}^{L+1} \left[\left(\prod_{\substack{i=1 \\ i \neq r}}^{L+1} \sum_{k_i^+=0}^{b_i-1} \right) \left(\prod_{\substack{i=1 \\ i \neq r}}^{L+1} \sum_{k_i^-=0}^{c_i-1} \right) (-1)^{k^-} \frac{k_r^- (k^+ + k^- - 1)!}{\prod_{i=1}^{L+1} k_i^+! k_i^-!} (E + k^+ + 1)_{-k^+ - k^-} \right. \\
& \times \left. \left(\prod_{i=1}^{L+1} (a_i)_{k_i^+ + k_i^-} \right) F(\{a_i + k_i^+ + k_i^-\}, \{b_i - k_i^+\}, \{c_i - k_i^-\}) \right]_{k_r^- = c_r},
\end{aligned} \tag{34}$$

where $k^+ = \sum_{i=1}^L k_i^+$, $k^- = \sum_{i=1}^L k_i^-$, and $(a)_b$ is the rising Pochhammer symbol $\Gamma(a+b)/\Gamma(a)$. The first term decreases the power b_r to 0, and the second term decreases c_r to 0. The only significant difference between the two terms is the +1 in the Pochhammer symbol.

Because the Pochhammer symbol that depends on E only appears once in each term, powers of $1/\epsilon^2$ or higher cannot occur. Thus, the explicit summation formula for the diamond rule does not have spurious poles.

(B) Examples

Four examples of diamond structures are displayed in Fig. 20. The role of each line in the diamond rule is highlighted by different colours and shapes. Red dashed lines, green double lines, and blue thick lines represent upper, lower, and external lines of the diamond, respectively. Label T represents the top vertex, and B the bottom vertex. In Fig. 20a a four-loop diagram is displayed. For this diagram, the line of either p_5 , p_6 , p_7 , p_8 , p_9 , or p_{10} can be removed by recursive use of the diamond rule or by the explicit formula given in the previous section. The irreducible numerators of this diagram are selected as $Q \cdot p_8$, $Q \cdot p_{10}$, $p_5 \cdot p_{10}$, and $p_5 \cdot p_7$, such that they adhere to the tensorial structure in the diamond rule. The last numerator, $p_5 \cdot p_7$, lies outside of the diamond and does not interfere with the rule.

If, in this figure, we draw an additional line from the top (T) to the bottom (B) vertex, we obtain the simplest non-trivial propagator topology with a spectator line. As a five-loop diagram it is unique.

In Fig. 20b the three-loop master topology NO is displayed. $Q \cdot p_5$ is chosen as irreducible numerator. One of the lines attached to the diamond is actually an off-shell external line. In general, if the line with momentum p_{L+1} is one of the external momenta of the larger graph, the factor $(p_{L+1}^2 + m_{L+1}^2)$ is just a constant with respect to the loop integration and has no role for reducing the complexity of

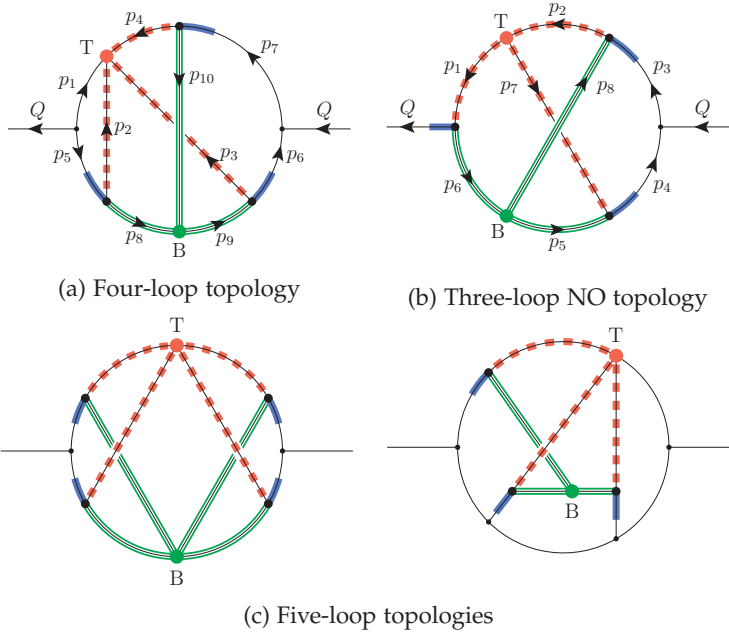


Figure 20: Four topologies with highlighted diamond structures. Red with dashed lines, green with double lines, and blue with thick lines represent upper, lower, and external lines of the diamond, respectively. Label T represents the top vertex, and B the bottom vertex. (a) shows a four-loop topology which can be completely reduced. (b) shows the three-loop NO master topology, for which a modified form of the diamond rule can be applied to lower the power of line p_1 to 1. (c) shows two five-loop topologies, which the diamond rule can be applied to.

the integral. As a result, the rule (30) is not applicable to remove one of the internal lines. Even for such cases, one can still find a useful rule by shifting $a_{L+1} \rightarrow a_{L+1} - 1$:

$$1 = \frac{1}{p_{L+1}^2 + m_{L+1}^2} \left[\sum_{i=1}^L \frac{a_i}{a_{L+1} - 1} A_i^+ A_{L+1}^- (B_i^- - C_i^-) - \frac{E+1}{a_{L+1} - 1} A_{L+1}^- + B_{L+1}^- \right], \quad (35)$$

which decreases at least a_{L+1} or b_{L+1} by one. Repeated use of this rule from positive integer a_{L+1} and b_{L+1} reduces a_{L+1} or b_{L+1} to 1. For the NO topology, this variant yields the rule to reduce the line p_1 to 1 in Mincer [32, 33].²

In Fig. 20c we show two five-loop topologies for which the diamond rule can eliminate one line. The first diagram is unique in the sense that it is the simplest diagram for which $L = 3$, $S = 0$. The second diagram is a typical representative

² The triangle rule counterpart of this variant was used to reduce the peripheral lines of the massless two-loop propagator-type diagrams with non-integer powers of the central line to unity.

of the 29 five-loop topologies with $L = 2$, $S = 0$ and all three p -momenta of the diamond internal.

3.4.3 Custom solutions

Using the combination of above rules, many topologies can be simplified. However, if the Feynman diagram does not contain a triangle, a diamond, an insertion, or a carpet, the IBPs have to be solved manually. One well-known example is the reduction of the three-loop non-planar diagram [43], which paved the way for fast three-loop reductions.

Let us consider a straightforward example with a system of two equations:

$$\begin{aligned} 1) \quad Z(n_1, n_2, n_3) &= n_1 Z(n_1 - 1, n_2, n_3) + \mathbf{Z}(\mathbf{n}_1, \mathbf{n}_2 + \mathbf{1}, \mathbf{n}_3) \\ 2) \quad Z(n_1, n_2, n_3) &= n_3 Z(n_1, n_2, n_3 - 1) + n_2 \mathbf{Z}(\mathbf{n}_1, \mathbf{n}_2 + \mathbf{1}, \mathbf{n}_3) \end{aligned} \quad (36)$$

Here we see that we have two terms that only increase indices without lowering any others, highlighted in bold. These terms have to be removed in order to have a reduction scheme that, when repeatedly applied, always lowers one of the indices. Solving it analytically, we get:

$$Z(n_1, n_2, n_3) = \frac{1}{1 - n_2} \left(n_3 Z(n_1, n_2, n_3 - 1) - n_1 n_2 Z(n_1 - 1, n_2, n_3) \right) \quad (37)$$

This rule can be applied until $n_2 = 1$ or until $n_1 = 0$ or $n_3 = 0$, since the latter cases mean we have removed a propagator. The next step would be to solve $Z(n_1, \mathbf{1}, n_3)$, since any configuration with $n_2 > 1$ will have $n_1 = 0$ or $n_3 = 0$.

In general, it is quite difficult to solve these systems parametrically. Therefore, many computations are performed using the Laporta method [42, 94, 95, 100]. In its basic form, each parameter is substituted by a number, and every configuration is solved separately. The solutions to the system are brute force Gaussian eliminations, which is a time consuming process.

The Laporta system above for $Z(1, 2, 1)$ looks like:

$$\begin{aligned} 1) \quad Z(1, 2, 1) &= Z(0, 2, 1) + \mathbf{Z}(\mathbf{1}, \mathbf{3}, \mathbf{1}) \\ 2) \quad Z(1, 2, 1) &= Z(1, 2, 0) + 2\mathbf{Z}(\mathbf{1}, \mathbf{3}, \mathbf{1}) \\ \Rightarrow Z(1, 2, 1) &= 2Z(0, 2, 1) - Z(1, 2, 0). \end{aligned} \quad (38)$$

If we had to solve $Z(2, 2, 1)$, we would have to solve the system of eq. (36) for $Z(2, 2, 1)$ and for $Z(1, 2, 1)$.

Some topologies are called *master topologies*: these cannot be guaranteed to get a line removed through IBPs, and they have to be solved using different methods. An example is the simple system above, that could be used to reduce n_2 to 1, but not to 0. In all cases we have encountered, the IBP reduction rules can be used to simplify the edge powers to 1, and the numerator powers to 0.

3.5 SOLVING PARAMETRIC IBP IDENTITIES BY HEURISTICS

In this section we will give heuristics on how to solve more complicated IBP systems parametrically. In an N -loop propagator graph we have $N + 1$ independent vectors: the external vector Q and N loop momenta p_i , where $i = 1, \dots, N$. Together there are $(N + 2)(N + 1)/2$ independent variables. One of them, Q^2 , can be used to set the scale. Hence there are $(N + 2)(N + 1)/2 - 1$ variables in the loops. Because there are at most $3N - 1$ propagators, the remaining variables will be in the numerator and there is often quite some freedom as to which variables to choose. In topologies in which there are fewer propagators there will correspondingly be more variables in the numerator. The efficiency of the reduction depends critically on the selected numerators. In the MINCER program the numerators were chosen to be dot products, such as $2p_7 \cdot p_8$ for the ladder topology or $2Q \cdot p_2$ for the Benz topology. Alternatively, one could use extra squared momenta such as p_9^2 with $p_9 = p_7 - p_8$. The advantage of the invariant method is that when rewriting the numerators to a new basis after a line removal, more invariants of the old basis can be a part of the new basis. The advantage of using dot products is that integration of one-loop subintegrals and the use of the rule of the triangle/diamond generates fewer terms compared to using invariants. Especially the simpler structure for integrating one-loop two-point subgraphs is important, since we apply this rule as early as possible to reduce the number of loops (and thus the number of parameters). Hence we choose to use dot products for the variables in the numerator in FORCER.

In the reduction routines we represent the integrals by a function Z with 14 variables (for fewer than four loops there will naturally be fewer variables) in which powers of variables in the denominators are given by positive numbers and powers in the numerator by negative numbers, as is commonly used. For example:

$$Z(1, 1, 1, 1, 1, 1, 1, 1, 1, -1, -2, -2, -2) \quad (39)$$

is a four-loop integral with four dot products. One dot product has power one, and the other three have two powers. Each of the 10 denominators has power 1. Note that all information about the topology or the choice of dot products is erased in this notation. Once the IBP relations are constructed, such information should be kept by different means. We note that some indices may be associated with propagators that have non-integer powers if insertions are involved (see section 3.5.3).

We define the integral in which all denominators have power one (possibly with an extra multiple of ϵ) and all numerators have power zero to have *complexity* zero. For each extra power of a denominator or of a numerator the complexity is increased by one. When we construct the IBPs parametrically the variables are represented by parameters $n_1 \dots n_{14}$ in which at least three represent numerators. Now we define the integral with just $n_1 \dots n_{14}$ as arguments to have complexity zero and again raising the value of a denominator by one, or subtracting one from a numerator raises the complexity by one. To improve readability, we represent denominators by parameters n and numerators by parameters k in the examples.

We redefine Z by adding *minus* the complexity as the first argument.³ For example:

$$Z(-3; 2, 1, 1, 1, 1, 1, 1, 1, 1, 0, -1, -1), \quad (40)$$

$$Z(-1; n_1, n_2, n_3 + 1, n_4, n_5, n_6, n_7, n_8, n_9, n_{10}, n_{11} - 1, k_{12} + 1, k_{13} - 1, k_{14} - 1). \quad (41)$$

In general, the goal is to construct a rule under which the basic complexity 0 integral $Z(0; n_1, \dots, n_{14})$ is expressed in terms of other complexity 0 integrals or in terms of integrals with negative complexity.

The remainder of this section is structured as follows. We provide heuristics to find reduction rules in section 3.5.1. Next, we generate new IBP rules in section 3.5.2. We describe special rules for diagrams with non-integer powers in section 3.5.3. Finally, we summarise our solving strategy in section 3.5.4.

3.5.1 Heuristics and equation generation

For four-loop diagrams there are at first instance 20 unique IBP relations, formed from the operation $\frac{\partial}{\partial p^\mu} q^\mu$, where p is one of the four loop momenta and q is one of the four loop momenta or the external momentum. This set of equations can often be simplified by a Gaussian elimination of the more complex integrals. We call the simplified set of equations S_0 . The most complex terms in S_0 have complexity 2 and have one raised denominator and one raised irreducible numerator. This is a direct consequence of the IBP structure.

In the set S_0 one can distinguish several types of reduction identities. The nicest identities are the ones that lower the complexity, sometimes even by more than one unit. An example is

$$\begin{aligned} 0 = & Z(-2; \dots, n+1, \dots, k-1, \dots) \cdot n \\ & + Z(0; \dots, n, \dots, k, \dots) + \dots, \end{aligned} \quad (42)$$

where both a propagator and numerator are raised in the complexity 2 term. By shifting $n \rightarrow n-1$ and $k \rightarrow k+1$, we find the reduction rule:

$$Z(0; \dots, n, \dots, k, \dots) = \frac{-1}{n-1} [Z(2; \dots, n-1, \dots, k+1, \dots) + \dots]. \quad (43)$$

Such identities are used for the simultaneous reduction of two variables. Since the equation will vanish once $n = 1$ or ‘overshoot’ when $k = 0$, it can only be used to speed up a reduction. Consequently, rules for the individual reduction of n and k are still required.

In what follows we will omit the last step of shifting the equation such that the highest complexity term becomes complexity 0. We also omit the coefficients of the Z functions when they are deemed irrelevant and we do not consider integrals with lines missing to be Z -integrals.

³ We use minus the complexity, so that FORM prints the integrals with the highest complexity first.

We now study relations that raise only one coefficient:

$$0 = Z(-1; \dots, n+1, \dots) \cdot n \\ + Z(0; \dots) + \dots \quad (44)$$

Repeated application of this relation will either take the variable n down to one, or eventually create integrals in which one or more of the other lines are missing. For non-master topologies, at some point we find an equation that looks like

$$0 = Z(-1; \dots, n+1, \dots) \cdot (\epsilon + \dots) \\ + Z(0; \dots) + \dots \quad (45)$$

This equation has an ϵ in the coefficient, which means it does not vanish if $n = 1$. As a result, it can be used to reduce n to 0.

If after there is an equation in which the highest complexity is zero and the integral for which none of the parameters has been raised or lowered is present, there is a good chance that one can eliminate at least one line in that topology by repeated application of this identity, provided that there are no lines with a non-integer power. One example of such an equation is the rule of the triangle. The finding of more such equations while investigating the IBP systems of five-loop propagator diagrams led to the discovery of the diamond rule [10].

3.5.2 Reduction rules beyond S_0

Even though the triangle and diamond rule can be derived from equations in the set S_0 , the set generally does not contain enough equations to reduce a topology straight away. Therefore, we expand our system by taking the set S_0 and constructing all equations in which either one denominator has been raised by one or one numerator lowered by one (which means that there is one more power of that variable because the numerators ‘count negative’). This set is called S_1 , since the IBPs are constructed from a complexity one integral. In total, we now have $20 + 280$ equations. Similarly we could construct the set S_2 by raising the complexity of one of the variables in the set S_1 in all possible ways, generating an additional 2100 equations. Usually S_2 is not needed. In some cases we may need a set like S_{-1} in which the complexity of one of the variables has been lowered, or even $S_{1,-1}$ in which one has been raised and one has been lowered.

The essence of our method is to construct the combined sets S_0 and S_1 and use Gaussian elimination to remove all objects of complexities 3 and 2 from the equations. The remaining equations only have objects of complexity one or lower. Out of these equations we construct an elimination scheme by defining an order of the variables, and we select for each variable an equation to eliminate it. For a denominator variable this is ideally an equation with a single term in which a variable n has been raised and all other parameters are at their default values: $Z(-1, n_1, n_2, \dots, n+1, \dots)$. Once we have such an equation we can lower $n + m$, with m being a positive integer,

in all other equations to n . Since we know that after this either one of the other n_i will be 0 (meaning the reduction is done) or $n = 1$, we can assume from this point on that $n = 1$. Thus, in all other equations we now set n to 1, lowering the number of parameters by one. Similarly the numerator variables are worked up from $n - m$ to n after which this variable is given the value zero. The order of elimination and the selection of the equations is critical: one of our early carefully selected schemes resulted in a benchmark run of 53000 seconds, whereas a scheme with a different variable order and a more sophisticated combination of the equations, performed the same test in 555 seconds.

Above we gave an example of a simple, useful equation. However, sometimes these equations are not there. Below we discuss several other types of equations one may encounter. One example is if there are more integrals of complexity one:

$$\begin{aligned}
 0 = & Z(-1; \dots) \\
 & + Z(-1; \dots) \\
 & + Z(-1; \dots) \\
 & + Z(-1; \dots, n+1, \dots) \cdot n \\
 & + Z(0; \dots) + \dots,
 \end{aligned} \tag{46}$$

where the last complexity one term can be used to eliminate the variable n (provided that all other parameters have not been raised/lowered in this term), but this goes at the cost of increasing the number of terms with the same complexity. When the scheme is not carefully selected, the number of terms in the intermediate stages may become very large and the rational polynomials could become complicated.

A convenient subclass of the type shown in eq. (46) is one that increases an index in only a single term in the equation, independent of the complexity:

$$\begin{aligned}
 0 = & Z(-1; n_1 + 1, \dots) \cdot n_1 \\
 & + Z(-1; n_1, \dots) \\
 & + Z(0; n_1, \dots) \\
 & + Z(1; n_1, \dots) \\
 & + \dots
 \end{aligned} \tag{47}$$

As a result, the equation can be used to lower the value of this variable at any level of complexity c :

$$\begin{aligned}
 0 = & Z(-c; n_1 + 1, \dots) \cdot n_1 \\
 & + Z(-c; n_1, \dots) \\
 & + Z(-c + 1; n_1, \dots) \\
 & + Z(-c + 2; n_1, \dots) \\
 & + \dots
 \end{aligned} \tag{48}$$

We emphasise that we apply these equations to any value of n_1 , so also to terms that look like $Z(-2, n_1 + 2, \dots)$. In FORM this can be done with a pattern match:

$$\begin{aligned} \text{id } Z(-c?, n_1?, \dots, n_{14}?) = & Z(-c, n_1-1, \dots)/(n_1-1) \\ & + Z(-c+1, n_1-1, \dots)/(n_1-1) \\ & + Z(-c+2, n_1-1, \dots)/(n_1-1) \\ & + \dots; \end{aligned}$$

These equations are convenient because after applying them and after setting the variable to 1, there will not be a single term in the remaining equations in which there is a number greater than 1 in its position. We will later see why this is desirable.

The next type of equations also has more than one term at complexity one, but there is no clean reduction of a given variable:

$$\begin{aligned} 0 = & Z(-1; n_1 + 1, n_2 - 1, n_3 + 1, \dots) \cdot n_1 \\ & + Z(-1; n_1 + 1, \dots) \cdot n_1 \\ & + Z(-1; n_1, \dots) \\ & + Z(-1; n_1, \dots) \\ & + Z(0; \dots) + \dots \end{aligned} \quad (49)$$

In the numerical case, one just moves the second term to the left and either n_1 will be reduced to one or n_2 will eventually become zero. However, in the derivation of the scheme one needs to apply this equation inside other parametric equations and more care is called for. One should apply the equation as many times as needed until terms either have n_1 (or $n_1 - 1$, etc.) or the n_2 position has $n_2 - 1$. This means that for the integral

$$Z(-1; n_1 + 1, n_2 + 2, \dots) \quad (50)$$

equation (49) will have to be used up to three times. Once n_1 has been set equal to one, one may end up with terms such as

$$Z(-1; 2, n_2 - 1, \dots), \quad (51)$$

which are undesirable. The solution to this problem is to try to deal with n_2 immediately after n_1 . Once we can put n_2 equal to one, $n_2 - 1$ becomes zero and hence it is an integral with a missing line. If one waits with the n_2 reduction and does another variable first, one risks that n_2 is raised because the equation for the other variable could have a term with $n_2 + 1$ and then one would end up with an integral of the type

$$Z(-1; 2, n_2, \dots). \quad (52)$$

This introduces either unresolved integrals or loops in the reduction scheme. It is also possible that one has reductions with two such conditions as we saw above. This requires great care in the selection of the next equation. We have not run into impossible situations at this stage.

Another case is one where the coefficient limits its application. For example

$$\begin{aligned} 0 = & Z(-1; n_1 + 1, \dots) \cdot (n_1 - n_2) \\ & + Z(0; \dots) + \dots \end{aligned} \quad (53)$$

cannot be applied when $n_1 = n_2$. Such rules could be very compact and are therefore used as a special case while the case in which n_1 is equal to n_2 is handled by a more general rule with less favourable properties but effectively one parameter less.

By far the most difficult equations are of the type

$$\begin{aligned}
 0 = & Z(-1; n_1 + 1, n_2 - 1, n_3 + 1, \dots) \cdot a(n_1, n_2, n_3) \\
 & + Z(-1; n_1 + 1, n_2 + 1, n_3 - 1, \dots) \cdot b(n_1, n_2, n_3) \\
 & + Z(-1; n_1 + 1, \dots) \\
 & + Z(-1; n_1, \dots) \\
 & + Z(0; \dots) + \dots,
 \end{aligned} \tag{54}$$

where a and b are coefficient functions. We call this type a *yoyo*. As a recursion it will never end, because the values of n_2 and n_3 will keep going up and down. There are various ways to resolve this. The first is to construct a new type of recursion. This is done by applying the equation twice:

$$\begin{aligned}
 & Z(n_1 + 1, n_2, n_3, \dots) \rightarrow \\
 & \quad + a(n_1, n_2, n_3) Z(n_1 + 1, n_2 - 1, n_3 + 1, \dots) \\
 & \quad + b(n_1, n_2, n_3) Z(n_1 + 1, n_2 + 1, n_3 - 1, \dots) + \dots \\
 \rightarrow & \quad + a(n_1, n_2, n_3) a(n_1, n_2 - 1, n_3 + 1) Z(n_1 + 1, n_2 - 2, n_3 + 2, \dots) \\
 & \quad + b(n_1, n_2, n_3) b(n_1, n_2 + 1, n_3 - 1) Z(n_1 + 1, n_2 + 2, n_3 - 2, \dots) \\
 & \quad + (a(n_1, n_2, n_3) b(n_1, n_2 + 1, n_3 - 1) + \\
 & \quad \quad b(n_1, n_2, n_3) a(n_1, n_2 - 1, n_3 + 1)) Z(n_1 + 1, n_2, n_3, \dots) + \dots
 \end{aligned} \tag{55}$$

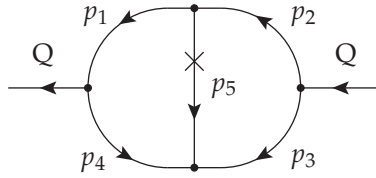
By moving the third term to the left one has a new recursion with a shift of two units. This procedure can be repeated i times until both $n_2 - 2^i$ and $n_3 - 2^i$ are less than one. The price to pay for this solution is high: fractions become enormously complicated and the number of terms could become very large.

An improved solution is to find another equation with a similar yoyo and combine the equations in such a way that one of the yoyo terms is eliminated. After this, one has a regular condition. We call this 'breaking the yoyo'. There is another way to break the yoyo that will be introduced below. We had to apply both methods of breaking the yoyo several times in the creation of the reduction schemes for the master topologies.

A final consideration is the structure of the coefficients of the integrals. In principle it is not very difficult to construct a reduction scheme from the available equations. The problem is that most schemes will end up with rational coefficients that take many megabytes to store because there are still quite a few variables in them. During the derivation this may cause problems with the limitations of the computer algebra system that is used (in our case FORM). More importantly, the evaluation of such rational polynomials in the application of the reduction scheme to millions of integrals will render the reductions impossibly slow and hence useless for all practical purposes. Thus, if the coefficients are too large, an alternative reduction has to be found.

3.5.3 Identities for topologies with insertions

When a topology contains a line that does not have an integer power, the method of the previous section has to be slightly extended. Such cases occur either when the input diagram(s) can be written with a higher-order propagator in it, or when during the reduction a two-point function can be integrated out. If the resulting topology needs a custom reduction, we not only have to lower powers of denominators and numerators, but we also have to bring the powers of the non-integer lines to a canonical value, which we take to be $1 + m\epsilon$ for some positive integer m . As an example, we consider the two-loop $\mathbf{t1star05}$ topology (see also refs. [32, 33])



which has an ϵ in index 5, indicated by a single cross. We call such a cross an *insertion*. We have the relation:

$$Z_{\mathbf{t1star05}}(n_1, n_2, n_3, n_4, n_5) = Z_{\mathbf{t1}}(n_1, n_2, n_3, n_4, n_5 + \epsilon), \quad (56)$$

where the topology $\mathbf{t1}$ is the same two-loop topology but without any implicit non-integer powers. Since the ϵ can never be removed from the index during the reduction, we suppress it in our notation for $\mathbf{t1star05}$. The IBPs for $\mathbf{t1star5}$ are generated from those of $\mathbf{t1}$ by a substitution $n_5 \rightarrow n_5 + \epsilon$. Typically, one tries to first reduce the integer indices n_1, \dots, n_4 to 1. During these reductions, the contribution to the integral complexity from n_5 could be taken as the absolute value of the difference to penalise any change of n_5 , or just be ignored to allow any change:

$$\mathbf{Complexity}(n_5 + m_5) = |m_5|, \quad \text{or} \quad \mathbf{Complexity}(n_5 + m_5) = 0. \quad (57)$$

After all n_1, \dots, n_4 are 1, we reduce the remaining index n_5 to 1, which may be positive or negative at this point. To derive a rule for the positive n_5 case, the complexity of n_5 can be defined as usual for a propagator:

$$\mathbf{Complexity}(n_5 + m_5) = m_5, \quad \text{for a rule with } n_5 > 1. \quad (58)$$

On the other hand, for the negative n_5 case, the complexity of n_5 can be defined as usual for a numerator:

$$\mathbf{Complexity}(n_5 + m_5) = -m_5, \quad \text{for a rule with } n_5 < 1. \quad (59)$$

In this way, all integrals belonging to $\mathbf{t1star05}$ can be reduced to the master integral $Z_{\mathbf{t1star05}}(1, 1, 1, 1, 1)$ and integrals with simpler topologies.

3.5.4 Solving strategy

The heuristics for ‘solving’ a topology can now be outlined in a list of nine steps.

1. Select a numerator basis. The quality of the IBPs will depend on this choice.
2. Construct the IBP identities.
3. (Important) Use a type of Gaussian elimination to simplify the IBP identities, minimising the number of terms with the highest complexity. We call this set S_0 . Most of the time this simplification can be done in an automated way. Only for the most difficult cases we have applied manual interference to obtain better results.
4. Construct the set S_1 by generating all possible options of raising an index in S_0 . This gives terms of complexity 2 and 3. Use Gaussian elimination to eliminate all those terms. The remaining set of equations has terms of at most complexity 1.
5. (Important) Use the equations of the set S_0 (applying it to any complexity and configuration as in eq. 48) to eliminate as many complexity one terms as possible. This can simplify the following task and results in simpler formulas in the final reduction program. It also breaks up some difficult yoyos.
6. Determine an order of elimination of the variables. Often the first variables are rather obvious from the presence of simple reduction equations. Some variables may not be so obvious and one may have to experiment. The resulting programs may differ by orders of magnitude in their efficiency. Here is where either human intelligence, or a cleverly written AI program can help.
7. In many cases, one cannot find a decent equation for the last variable. This can be because either the results have become extremely lengthy, or one has discarded some long equations that seemed of no further relevance. In that case, the almost complete reduction scheme is applied to the set S_0 . This will give a number of varieties of the final reduction(s). One can select the shortest one.
8. (Checking) Now apply the custom reduction scheme to the set S_0 with numbers for the variables and make sure that master integrals are indeed irreducible, and that the program does not get caught in loops. There may be equations remaining which only consist of integrals with missing lines. We did not take relations between those into account.
9. Combine all reductions and useful double reduction equations (equations that need at least two variables that are above their minimal complexity) based on S_0 or substitutions made during the Gaussian elimination. Together this forms the reduction procedure for the given topology.

In some cases the resulting schemes were still deemed too slow and more exhaustive methods were asked for. In such cases the sets S_2 and S_{-1} were also constructed and many different ways of combining the equations were tried automatically. Such programs could take much time because of the very complicated rational polynomials in the parameters of the integrals, but they eventually did result in a number of shorter reductions.

A number of FORM procedures has been constructed to execute the above steps. The most laborious step is to determine a proper order for the elimination sequence, and which equations to use for each. Furthermore, we had a case (the bebe topology of section 3.6) in which there were no good reductions for two of the variables, unless we used two of the equations in the set S_0 to eliminate them with a complexity raising operation. It also reduced the number of remaining equations to 18 and hence left fewer options during the remaining parts of the derivation.

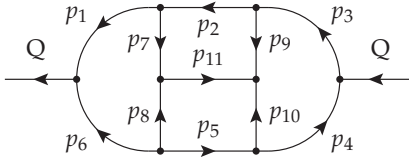
There are two major reasons why some reduction rules perform faster than others. The first reason is that even though a rule may have only one $Z(-1; \dots)$ term, it could be that the sub-leading terms increase the value of a variable that was set to 1 in one of the early steps of the scheme (see eq. 51). This forces the program back to an earlier reduction rule of the scheme, even though now at a lower complexity. The second reason is the coefficient growth: if a rule has a particularly complicated overall coefficient, it multiplies every term in the RHS and all subsequent terms will have rather lengthy rational polynomials in ϵ . Expanding in ϵ (see appendix 3.9) can alleviate some of these problems, provided one expands deep enough to avoid issues with spurious poles.

Determining the order of elimination seems suited for AI techniques, such as Monte Carlo Tree Search (see, e.g., [67]). One could use the number of top complexity terms, the number of lower complexity terms, the number of spectators and the size of the most complicated rational polynomial as parameters for an evaluation function for a given scheme and then use this in a MCTS to find an optimal scheme. This is currently under investigation. It should be noted that such type of use of AI for precisely this purpose was already hinted at in ref. [104].

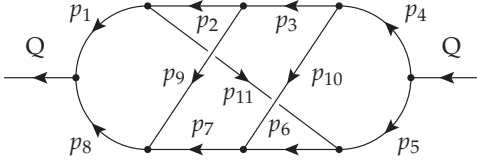
3.6 THE 21 TOPOLOGIES THAT NEED CUSTOM REDUCTIONS

Because we integrate over one-loop two-point functions, our classification of the master integrals differs from refs. [105, 106]. In general, any diagram that factorises we do not consider a master topology. The master diagrams that contain one-loop two-point functions that cannot be factorised, will have slightly different values, since we integrate out the bubble. The full list with the values of the master integrals in our convention are given in [1].

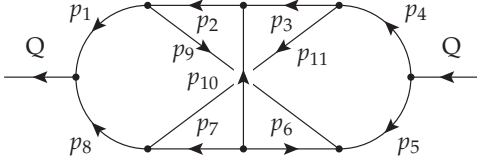
There are eight four-loop master integrals, excluding the diagrams in which a 2-point function can be integrated out.



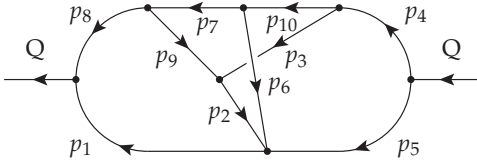
Topology name: **haha**, master.
 Momenta: p_1, p_2, p_4, p_5 .
 Numerators: $2Q \cdot p_2, 2Q \cdot p_5, 2p_1 \cdot p_4$



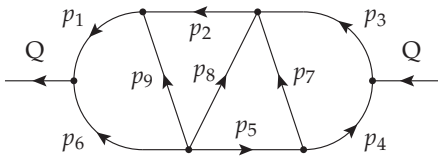
Topology name: **no1**, master.
 Momenta: p_1, p_2, p_3, p_4 .
 Numerators: $2p_2 \cdot p_4, 2Q \cdot p_2, 2Q \cdot p_3$.



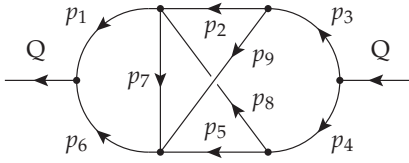
Topology name: **no2**, master.
 Momenta: p_1, p_2, p_3, p_4 .
 Numerators: $2Q \cdot p_2, 2p_1 \cdot p_4, 2Q \cdot p_3$.



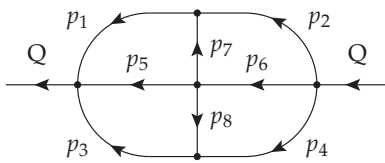
Topology name: **no6**, master.
 Momenta: p_1, p_2, p_3, p_4 .
 Numerators: $2p_1 \cdot p_2, 2p_2 \cdot p_4, 2Q \cdot p_2, 2Q \cdot p_3$.



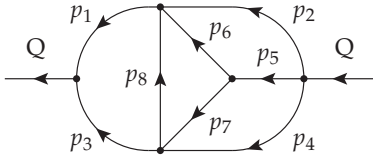
Topology name: **lala**, master.
 Momenta: p_1, p_2, p_4, p_5 .
 Numerators: $2Q \cdot p_5, 2Q \cdot p_2, 2p_1 \cdot p_4, 2p_1 \cdot p_5, 2p_2 \cdot p_4$.



Topology name: **nono**, master.
 Momenta: $p_1, p_2, p_3, p_{10} = p_2 + p_8$.
 Numerators: $2p_2 \cdot p_8, 2p_6 \cdot p_7, 2Q \cdot p_2, 2p_1 \cdot p_2, 2p_7 \cdot p_9$.



Topology name: **cross**, master.
 Momenta: p_1, p_2, p_3, p_4 .
 Numerators: $2Q \cdot p_1, 2Q \cdot p_2, 2Q \cdot p_3, 2Q \cdot p_4, 2p_1 \cdot p_4, 2p_2 \cdot p_3$.



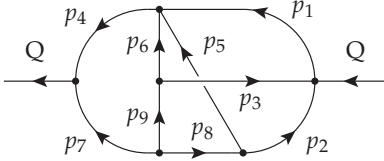
Topology name: **bebe**, master.

Momenta: p_1, p_2, p_4, p_6 .

Numerators: $2Q \cdot p_2, 2Q \cdot p_4, 2Q \cdot$

$p_6,$

$2p_1 \cdot p_2, 2p_2 \cdot p_6, 2p_1 \cdot p_4.$



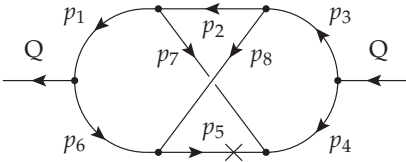
Topology name: **bubu**, not a master.

Momenta: p_2, p_3, p_8, p_9 .

Numerators: $2Q \cdot p_2, 2Q \cdot p_8, 2p_2 \cdot$

$p_3,$

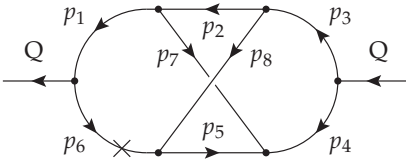
$2p_2 \cdot p_9, 2p_3 \cdot p_8.$



Topology name: **nostar5**, master.

Momenta: p_1, p_2, p_3 .

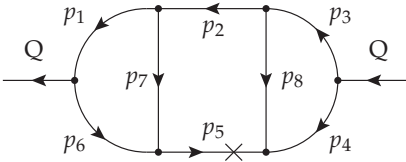
Numerators: $2Q \cdot p_2,$



Topology name: **nostar6**, master.

Momenta: p_1, p_2, p_3 .

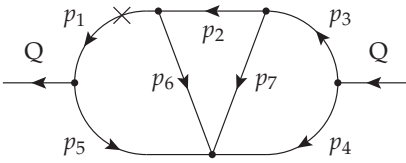
Numerators: $2Q \cdot p_2,$



Topology name: **lastar5**, not a master.

Momenta: p_1, p_2, p_3 .

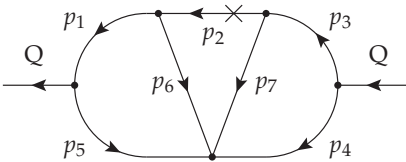
Numerators: $2p_1 \cdot p_3,$



Topology name: **fastar1**, not a master.

Momenta: p_1, p_2, p_3 .

Numerators: $2p_1 \cdot p_3, 2Q \cdot p_2,$



Topology name: **fastar2**, master.

Momenta: p_1, p_2, p_3 .

Numerators: $2p_1 \cdot p_3, 2Q \cdot p_2,$

	<p>Topology name: bustar5, not a master. Momenta: p_4, p_5, p_6. Numerators: $2Q \cdot p_4, 2Q \cdot p_5$.</p>
	<p>Topology name: t1star55, master. Momenta: p_1, p_2.</p>
	<p>Topology name: t1star24, master. Momenta: p_1, p_2.</p>
	<p>Topology name: t1star34, not a master. Momenta: p_1, p_2.</p>
	<p>Topology name: t1star45, master. Momenta: p_1, p_2.</p>
	<p>Topology name: no, master. Momenta: p_1, p_2, p_3. Numerators: $2Q \cdot p_2$, Remarks: Already in MINCER.</p>
	<p>Topology name: t1star05, master. Momenta: p_1, p_2. Remarks: Already present in MINCER.</p>

Table 3: Table of all the topologies that require a custom reduction.

For these master integrals we have to design a custom scheme in which the parameters are reduced, one by one, to the value they have in the master integral. In addition (and perhaps surprisingly) there are four non-master topologies that need such a custom reduction.

Only when all but a few parameters are set to 1, do we find a relation to reduce an edge to 0. In this category there is one at the four-loop level, two at the three-loop level (with one non-integer edge) and one at the two-loop level (with two non-integer edges). In total we need 21 custom reduction schemes. All other topologies can be dealt with using generic formulas that can either eliminate a line or integrate out a loop. We list all topologies that need a custom reduction scheme in table 3.

In order to choose the best reduction schemes for the topologies in table 3, we measure the performance of a complete calculation of the integrals with all indices raised by 1 (a complexity 14 integral at four loops). By performing a complete calculation, we confirm that the number of terms with a simpler topology created by the reduction rules does not cause bottlenecks. Additionally, we confirm that for the case where all indices are raised by 2 (a complexity 28 integral at four loops), the reduction is still performing well.

We note that the ordering of variables in the reductions scheme is not the only relevant parameter. The choice of numerators can influence the presence of non-leading terms, which after the Gaussian elimination become leading terms. Such terms can spoil the efficiency of certain reduction rules. In particular the three complicated topologies *nono*, *bebe*, and *no2* are sensitive to the choice of dot products.

Most schemes could be derived using the heuristics introduced in section 3.5, by selecting the reduction variable that corresponds to the shortest reduction rule. However, there are a few derivations that need more care. For *nono*, one needs to avoid a circular path in a special way. The formulas for the last two variables, n_4 and n_8 , can only be obtained by reusing the original set S_0 . At this point one uses either combinations of nearly all equations to obtain very lengthy formulas (> 1000 lines) or one uses a relatively short formula with a term that sends the reduction back to a previous rule, because it contains a term with $n_{11} = -1$. This would normally introduce a loop, but by sending only this term through the unfinished scheme and combining the result with the remaining part of the formula, we obtain a compact reduction formula for n_4 (39 terms).

The *bebe* reduction is more complicated as it does not yield a regular reduction for n_1 and n_3 . However, in the set S_0 there are equations that can be used for their reduction, provided we are willing to raise the complexity. This does not agree with the automatic nature of our derivation tools, and hence some work needs to be guided by hand. Furthermore, we can no longer use a number of equations from the S_0 set for generating reduction rules for other variables. As a consequence, we are left with far fewer equations after the Gaussian eliminations, although their number is still sufficient for the next 11 variables. Eventually the n_2 variable has to be obtained again from the S_0 set.

For the construction of a reduction scheme the *bubu* topology is by far the most complicated, even though it is not a master topology. There are five different

numerators and the elimination of the last numerator needs to be split into several cases, each with a rather lengthy formula involving complicated rational polynomials. In order to prevent a blow-up of terms, the order of elimination of the variables is critical, as well as using the equations obtained during the Gaussian elimination that give a direct reduction of the complexity. It took more than two months to find a first suitable reduction scheme.

We use the S_0 set and equations that come from the Gaussian elimination before we start with the 14 reduction identities of the complete schemes. This speeds up the reduction by a factor two or more, because these equations are usually much more compact and will often reduce the complexity immediately. It turns out that the final result is very sensitive to how we use these equations, because sometimes there are options when there is more than one term with the highest complexity, and also the order in which they are applied is relevant. Additionally one has to be careful with this ordering to avoid loops in the reduction. Unfortunately, it is not always possible to indicate which ordering is optimal, because some orderings may yield a faster scheme at the cost of more spectator terms and/or higher powers of ϵ in the rational polynomials.

Considering the amount of work involved in deriving the schemes, it is quite conceivable that better schemes will be found. It seems to be a good candidate for the application of automated AI techniques.

3.7 THE FORCER FRAMEWORK

In essence, the FORCER program provides a method to reduce each topology to simpler ones. There is quite some freedom: sometimes multiple reduction rules can be applied, sometimes it is best to use a different set of independent momenta, etc. In order to obtain the best performance, all decisions in the FORCER program are pre-computed by a script: for each topology the action is hard-coded and the momentum rewrites are known. The advantage of this method is that costly optimisations, such as choosing an optimal basis for each topology, have no runtime cost.

The remainder of this section is structured as follows. In section 3.7.1 we describe the construction of the reduction graph. Next, the execution of the reduction graph is shown in section 3.7.2. Finally, we give an example of the treatment of a topology in section 3.7.3.

3.7.1 Reduction graph generation

Before going into details, we first give an overview of the program. The program structure can be viewed as a directed acyclic graph (DAG), the *reduction graph*, where the nodes are topologies and each edge indicates a transition from one topology to another when a propagator is removed. As a result, each node may have more than one parent. The root nodes of the reduction graph are the top-level topologies, which are topologies that only have three-point vertices. All tadpole topologies will

be zero, so they are not included in the graph. To reduce the number of topologies, propagators with the same momentum are always merged.

Each node represents a topology, which consists of a graph with a certain fixed labelling of all the propagators, including momentum directions, and a fixed set of irreducible numerators. Each topology also has an action that determines how it can be reduced. They are, in order of preference: integration of a two-point function, carpet rule, triangle/diamond rule, or a custom reduction. Each topology contains transitions to other topologies for all removable edges (edges with integer power). Even though the specific rule may not be able to nullify any propagator in the graph, the dot product rewrites may, so therefore we generate all possible transitions. If there are lines missing, in most cases the topology action is not executed and the topology is automatically rewritten to another. The exception is for integrating insertions: insertions are guaranteed to reduce the number of loops, which simplifies the dot product basis. Thus, first rewriting the dot products to a new topology would be wasteful.

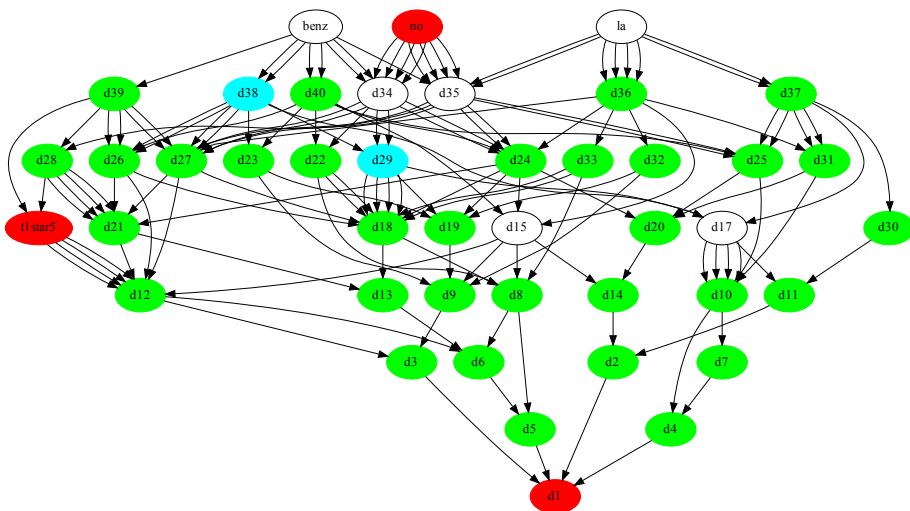


Figure 21: The three-loop reduction graph. Each node represents a topology, and each arrow a transition if a certain line is removed. The colour defines the topology action: white means the triangle or diamond rule, cyan the carpet rule, green the insertion rule, and red a custom reduction.

In figure 21 the reduction graph is displayed for three-loop massless propagator graphs. The names of the topologies are automatically generated. Every arrow denotes a transition that occurs when a propagator is removed. Multiply arrows could point to the same node if the resulting diagram is isomorphic. An example of this is `t1star5` (same as `t1star05` in section 3.6), where removing any of the four

outside lines results in the same topology. The central line cannot be removed, since it has a non-integer power. The four-loop reduction graph, with over 400 nodes, is far too large to display.

The reduction graph is generated from the top-level topologies down. For every topology, a new one is generated where a particular line is missing. For this new topology, we determine its action. Next, we generate a dot product basis that is compatible with the action, e.g., for the insertion rule all dot products should only involve at most one of the two momenta. We also determine its automorphisms (graph symmetries), so that we can map every topology instance to the same unique form (we will go into more detail about this in the next section).

The dot product basis is chosen according to the following three rules: (1) it is compatible with the action, (2) it minimises the number of terms created when rewriting from the parent topology. As a criterion we choose the sum of the square of the number of terms that are created in rewriting each dot product. (3) The dot products are chosen in line with the symmetries of the topology.

We summarise the generation of the reduction graph in algorithm 2.

```

Input : top-level topologies  $T$ 
Output: reduction tree  $T_{\text{all}}$ 
 $T_{\text{all}} \leftarrow T$ ;
foreach  $t \in T$  do determine action and
    automorphisms;
while  $T \neq \emptyset$  do
    take  $t \in T$ ;
    foreach propagator  $p \in t$  do
         $h \leftarrow$  new topology without  $p$ ;
        if  $h' \in T_{\text{all}}$  isomorphic to  $h$  then
            construct mapping from  $h \rightarrow h'$ ;
        else
            determine action for  $h$ ;
            generate dot product basis for  $h$ ;
            generate automorphisms for  $h$ ;
            generate mapping of dot products from
                 $t$  to  $h$ ;
            add node  $h$  to tree  $T_{\text{all}}$ ;
             $T \leftarrow T \cup \{h\}$ ;
        end
    end
end

```

Algorithm 2: Reduction graph generation.

The reduction graph is generated with a Python script, using `igraph` [107] for a basic graph representation of the topologies and for the isomorphism algorithm. Since by default only simple graphs (without self-edges and duplicate edges) are

supported by the isomorphism algorithm, we merge all double edges and use a custom function to determine if the topologies are truly isomorphic (one could view duplicate edges with possible insertions as a special edge colouring). This function enforces that the number of duplicate edges is the same, and that the distribution of insertions over duplicate edges is the same. Additionally, we generate all possible permutations over similar duplicate edges, to generate the edge isomorphisms. The reduction graph contains 438 topologies and requires 40 000 lines of FORM code.

3.7.2 Reduction graph execution

So far, we have discussed the generation of the reduction graph. Now we consider how the graph is processed in runtime.

As input, we have integrals that are labelled by the name of their topology in a symbol. In contrast to MINCER, the input expressions can contain multiple topologies. In FORCER, every topology is put in a separate expression and is hidden. The topologies are processed one by one, in the order of the number of edges. When a topology is treated, the expression is unhidden, the integrals are symmetrised using automorphisms, the topology action is executed, and finally, the resulting integrals are rewritten to their new topology. The topologies in the output are either master integrals, which require no further reductions, or topologies with fewer lines. These topologies will be merged into the designated expression for that topology. All the masters integrals are stored in their own expression.

After rewriting dot products, multiple edges could have vanished. Some of the integrals that remain could have become massless tadpoles, which are zero in dimensional regularisation. A table is used that maps the topology and a list of missing edges to zero if the resulting topology is a tadpole.

The execution of the reduction graph is summarised in algorithm 3.

3.7.3 Example

Below we give an example of the treatment of a topology. The topology is depicted in figure 22, and is internally called d366.

In the input, the integral is represented as follows by a compact notation in terms of symbols only:

`Md366/i1/i2/i3/i4^2/i5/i6/i7/i8/i9*i10*i11*i13;`

where Md366 is the marker of the topology and the powers of i_n represent the propagator and numerator powers. In this example we have three additional powers: $1/p_4^2$, $Q \cdot p_4$, $p_1 \cdot p_6$, and $p_1 \cdot p_4$. Since all rules are precomputed, the information of the topology such as the vertex structure, momentum flow, non-integer powers of lines and which dot products are in the basis, is never stored in the terms that are processed. Instead, the topology marker Md366 will be used to call the correct routines.

```

Input : input integrals  $I$ 
Input : reduction graph  $T_{\text{all}}$ 
convert  $I$  to FORCER topologies ;
foreach  $t \in I$  do put in its own expression  $E_t$  and
    deactivate;
for  $l = 11$  to  $1$  do
    foreach  $t \in T_{\text{all}}$  with  $l$  edges (any order) do
        activate expression with topologies  $t$  ;
        symmetrise terms (apply automorphisms) ;
        perform reduction operation (triangle,
            carpet, etc.) ;
        rewrite result with missing lines to FORCER
            topologies  $h_i \in T_{\text{all}}$  ;
        move the terms with topology  $h_i$  to  $E_{h_i}$ ;
    end
end

```

Algorithm 3: Reduction graph execution.

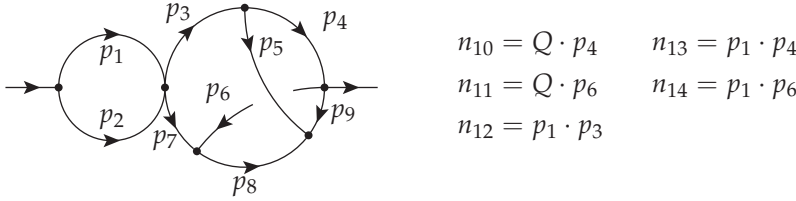


Figure 22: FORCER topology d366.

When treating topology d366, we first apply symmetries to make sure that similar configurations of d366 are merged. We use the automorphisms of the graph, of which there are four: $(p_1 \leftrightarrow p_2) \times (p_4 \leftrightarrow p_6, p_3 \leftrightarrow p_7, p_7 \leftrightarrow p_8)$. However, since there may be dot products in these momenta, the symmetry may be broken unless the set of dot products maps into itself. For the symmetry $(p_1 \leftrightarrow p_2)$, the dot products $p_1 \cdot p_3, p_1 \cdot p_4, p_1 \cdot p_6$ should be absent. The other symmetry can only be applied when $p_1 \cdot p_3$ is absent.

To find the smallest isomorphism, we hash the powers of the i , and take the smallest. The code is displayed in listing 3.2.

LISTING 3.2: FORM code for finding the smallest isomorphism

```

if (match(Md366*<1/i1~n1?n1>*...*<1/i14~n14?n14>));
if (($n12==0)&&($n13==0)&&($n14==0));
    #call hash(0,$n14,$n13,$n12,$n11,$n10,$n9,$n8,$n7,$n6,$n5,$n4,$n3,
        $n1,$n2)

```

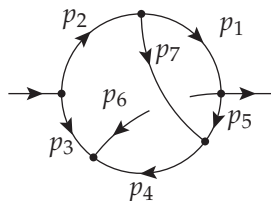
```

    #call hash(1,$n14,$n13,$n12,$n10,$n11,$n9,$n5,$n3,$n4,$n8,$n6,$n7,
    $n1,$n2)
endif;
if (($n12==0));
    #call hash(2,$n14,$n13,$n12,$n11,$n10,$n9,$n8,$n7,$n6,$n5,$n4,$n3,
    $n2,$n1)
    #call hash(3,$n13,$n14,$n12,$n10,$n11,$n9,$n5,$n3,$n4,$n8,$n6,$n7,
    $n2,$n1)
endif;
* stores best hash in $bestiso
#call smallesthash(0,1,2,3)
if ($bestiso == 0); Multiply replace_(i1,i2,i2,i1);
elseif ($bestiso == 1); Multiply sign_($n10+$n11+$n13+$n14)
    *replace_(i1,i2,i2,i1,i3,i7,i4,i6,i5,i8,i6,i4,i7,i3,i8,i5,i10,i11,
    i11,i10);
elseif ($bestiso == 3); Multiply sign_($n10+$n11+$n13+$n14)
    *replace_(i3,i7,i4,i6,i5,i8,i6,i4,i7,i3,i8,i5,i10,i11,i11,i10,i13,
    i14,i14,i13);
endif;
endif;

```

The action that will be performed in d366 is the integration of the left bubble, p_1 and p_2 . As can be seen in figure 22, all relevant dot products are written only in terms of p_1 and none in terms of p_2 , in alignment with the insertion rule. The dot products that involve p_1 can all be re-expressed in terms of inverse propagators after integrating the insertion. The two dot products that remain, $Q \cdot p_4$, and $Q \cdot p_6$ (represented by i10 and i11 respectively) have to be rewritten to the new topology.

The new topology is called d118:



$$n_8 = Q \cdot p_4$$

$$n_9 = Q \cdot p_7$$

where we have suppressed the ϵ power of the external line.

In listing 3.3, we display the mapping from d366 to d118, which includes rewriting the old dot products.

LISTING 3.3: FORM code for rewriting dot products

```

Multiply replace_(i3,j2,i4,j1,i5,j7,i6,j6,i7,j3,i8,j4,i9,j5);
id i10 = Q^2/2+j2/2-j3/2-j9;

```

```
id i11 = -Q^2/2+j2/2-j3/2+j8;
Multiply replace_(Md366,Md118,<j1,i1>,...,<j7,i7>,j8,-i8,j9,i9);
```

3.8 USAGE

The FORCER program can be downloaded from <https://github.com/benruijl/forcer>. Currently, the latest development version of FORM is required, which can be obtained from <https://github.com/vermaseren/form>. The generation scripts require Python 2.7, Python 3 or higher as well igraph [107], numpy [108] and sympy [109].

An example of FORCER input is displayed in listing 3.4.

LISTING 3.4: Example input for FORCER

```
#-
#include forcer.h

L F =
+1/<p1.p1>/.../<p6.p6>*Q.p3*Q.p4*vx(Q,p1,p5,p6)*vx(-p1,p2,p3)*vx(-p5,-
p6,p4)*vx(-Q,-p2,-p3,-p4)
+1/<p1.p1>/.../<p5.p5>*vx(-Q,p2,p3)*vx(p1,-p2,p5)*vx(-p1,p4,Q)*vx(-p3,-
p4,-p5)*ex(p1,p4)
;

#call Forcer(msbareexpand=4)
B ep;
P +s;
.end
```

After `forcer.h` is included, the input integral can be defined. This is done by specifying the vertex structure using `vx`. The external momentum should be called `Q`. The propagators and momenta can simply be multiplied in, as shown in the example above. Insertions on lines can be specified using the `ex` function. In the second integral above `ex(p1,p4)` means that there is a single ϵ on the propagator associated with momentum p_1 , and one on p_4 . The topologies will automatically be matched to FORCER's internal topologies. The dot products will also automatically be rewritten (see subsection 3.10.4).

By calling the `Forcer` procedure, the integrals are computed. The optional argument `msbareexpand` can give the (unrenormalised) answer expanded in \overline{MS} . Otherwise, the result will be given exactly in terms of the master integrals and rational coefficients. Other options include `polyratfunexpand=div` and `polyratfunexpand=`

`maxpow`, which enable the expansions of rational coefficients in ϵ at intermediate steps using the FORM statement `PolyRatFun` (see 3.9).

3.9 EXPANSIONS

In principle the coefficients of the integrals can be kept as rational polynomials in D or ϵ . However, the nature of the reductions is such that these polynomials can contain very high powers in their numerators and denominators. Adding such rational polynomials is easily the most costly operation during the reductions. During the development of the FORCER program, we have encountered polynomials with powers of ϵ that went over 700, and that was not even for a complete reduction. In practice one needs such ‘precision’ only in very rare cases, such as when one needs to change dimensions during or after the reduction. In our program this is not necessary, and hence a better strategy is to expand these polynomials to a finite power of ϵ . The main problem is that we do not know in advance how many powers are needed. The reductions will at times generate extra powers of $1/\epsilon$ (spurious poles) that will only cancel near the end of the reduction when all terms that contribute to a given master integral are added. An exact solution for the spurious problem is a denominator notation [110], but to make this workable FORM still needs supporting facilities.

We have opted for a method in which the reduction formulas still use rational polynomials, but after each step they are expanded to sufficient depth. It is possible to make a special trial run to determine how many powers are needed. In this trial run only the minimum power of ϵ is kept with the coefficient one, to avoid that such terms can cancel. Avoiding all calculations, such a run can be relatively fast, provided that the main computational effort is in the FORCER part of the program (it usually is). After the run, one can see how deep the expansions have to be. We usually take the worst value that we encounter for all diagrams and add one ‘guard power’. With this value the program generates the proper tables for the various Pochhammer symbols and other objects that may need expansions. Then during the actual reductions the rational polynomials will be expanded to the proper depth.

A simpler and safer method is to run the whole calculation twice with different settings for the expansion depth and observe at which power of ϵ the coefficients change. This is similar to running numerical programs with different floating point precisions to study the numerical instabilities.

FORM has options to use expansions in its coefficients. The command `PolyRatFun, rat(divergence, variable)` keeps only the lowest power of ϵ . Generally, the program is quite fast in this mode. To expand, the statement `PolyRatFun, rat(expand, variable, maxpow)` can be used. These commands are implemented in the latest development version of FORM.

3.10 FROM PHYSICAL DIAGRAMS TO FORCER

The interface provided in the previous section expects scalar integrals as input. In order to compute Feynman diagrams, process-specific preprocessing has to be performed. Since the actual implementation is highly dependent on conventions, we will only sketch certain parts.

The program QGRAF [111] provides a convenient way to generate the Feynman graphs that are needed for the actual calculations, because it can generate FORM compatible output. However, the challenge remains of converting the diagrams as presented by QGRAF to something that the FORCER program can deal with. This involves mapping the topology and momenta of the diagrams to FORCER's internal notation. For this purpose, the Python program that generates the reduction graph also generates a file called `notation.h` which contains a specification of all topologies in such a way that a conversion program can use it for

1. topology recognition,
2. labelling the momenta and their directions for each line,
3. using symmetry transformations.

Each topology is represented by a term in FORM notation. Two typical terms are displayed in listing 3.5.

LISTING 3.5: Two entries in the notation file

```
+vx(-Q,p4,p5)
  *vx(p3,-p4,p11)
  *vx(p6,p7,p10)
  *vx(p2,-p3,-p10)
  *vx(p1,-p2,p9)
  *vx(-p5,-p6,-p9)
  *vx(-p7,p8,-p11)
  *vx(-p1,-p8,Q)
  *SYM()
  *SYM(Q,-Q,p1,-p5,p2,p6,p3,-p7,p4,-p8,p5,-p1,p6,p2,p7,-p3,p8,-p4,p9,-
    p9,p10,-p10,p11,-p11)
  *SYM(Q,-Q,p1,-p4,p2,-p3,p3,-p2,p4,-p1,p5,-p8,p6,p7,p7,p6,p8,-p5,p9,
    p11,p11,p9)
  *SYM(p1,p8,p2,p7,p3,-p6,p4,p5,p5,p4,p6,-p3,p7,p2,p8,p1,p9,-p11,p10,-
    p10,p11,-p9)
  *TOP0(Mno2)

+vx(-Q,p3,p4)
  *vx(p2,-p3,p7)
  *vx(p1,-p2,p6)
  *vx(-p1,p5,Q)
```

```

*vx(-p4,-p5,-p6,-p7)
*ex(p2)
*SYM()
*SYM(Q,-Q,p1,-p3,p2,-p2,p3,-p1,p4,p5,p5,p4,p6,p7,p7,p6)
*TOP0(Mfastar2)

```

The first term indicates the no2 topology. The function vx indicates the vertices and the momenta belonging to that vertex. Negative momenta are incoming. The function TOP0 has a symbol as an argument that indicates the topology. In the FORCER program terms that are in the notation of a given topology are labelled with one power of the corresponding symbol. The function SYM describes a symmetry operation of the topology. The FORM statement

```
id,once,SYM(?a) = replace_(?a);
```

will execute such an operation. In practice one could use it as described in listing 3.6.

LISTING 3.6: FORM code for applying symmetries

```

id vx(?a) = f1(vx(?a));
repeat id f1(x1?)*f1(x2?) = f1(x1*x2);
repeat id SYM(?a)*f1(x?) = f1(x)*f2(x*replace_(?a));
id f1(x1?)*f2(x2?) = x2;
id f2(x?) = 1;

```

This process makes for each occurrence of the function SYM a copy of the contents of the function f1 in which the corresponding symmetry operation has been applied. Because the normal ordering algorithm of FORM puts the smallest of the functions f2 first, we end up with the smallest representation of the term. If this is applied at a later stage in the program more statements may be needed, because there may be more objects than vx.

The notation file includes more topologies than actually exist in the FORCER reduction graph, since physical diagrams can have duplicate momenta. If this is the case, the term in the notation file also contains a function ID, for example ID(p4,-p5), indicating that p_4 and $-p_5$ are actually the same momentum. After the topology is matched and the labelling is done, the following ID function can be applied.

```
id ID(p1?,p2?) = replace_(p1,p2);
```

The first step in determining the topology of a diagram is to read the notation.h file, number its topologies, and store each of them in a dollar variable with a name that is labelled by this number. We also store the names of the topologies in such an array of dollar variables. The topology of a diagram can now be determined by trying whether one of the topologies can be substituted in the term. If this pattern

matching involves wildcards, and the match of the wildcards is stored inside dollar variables we can use this to relabel the diagram itself and bring it to the notation of the topology. The main problem is creating the match structure, since we need wildcards for all the momenta followed by the name of a dollar variable. This issue is resolved with the dictionary feature of FORM. The essential part of the code is shown in listing 3.7.

LISTING 3.7: FORM code for topology matching

```
#OpenDictionary wildmom
  #do i = 1, '$MAXPROPS'
    #add p'i': "p'i'?$p'i'"
  #enddo
#CloseDictionary

#do i = 1, '$MAXPROPS'
  $p'i' = p'i';
#enddo

#UseDictionary wildmom($)
#do i = 1, '$numtopo'
  if ( match('$topo'i') );
    $toponum = 'i';
    goto caught;
  endif;
#enddo
#CloseDictionary
label caught;

Multiply replace_(Q,Q,<$p1,p1>,...,<$p'$MAXPROPS',p'$MAXPROPS'>)*topo(
  $toponum);
```

When we try to match, the printing of the '\$topo'i' variable will result in objects like $vx(p1?sp1,p2?sp2,p3?sp3)*\dots$ rather than the $vx(p1,p2,p3)*\dots$ that it actually contains. This way the \$-variables get the value of the momenta in the diagram for which we want to determine the topology and the notation. The final replace substitutes these momenta by the value they have in the topology file.

It is possible to speed up the process considerably by hashing the topologies by the number of vertices and by first stripping the signs of the momenta. These signs can be recovered in a later step.

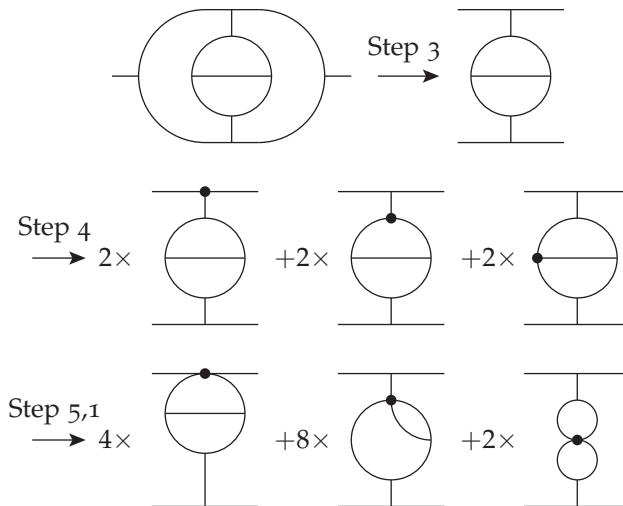
The remainder of this section is structured as follows. In section 3.10.1 we describe how to filter self-energies. The colour split-off is shown in section 3.10.2. In section 3.10.3 we describe the diagram database. Finally, we construct the momentum substitution routines in section 3.10.4.

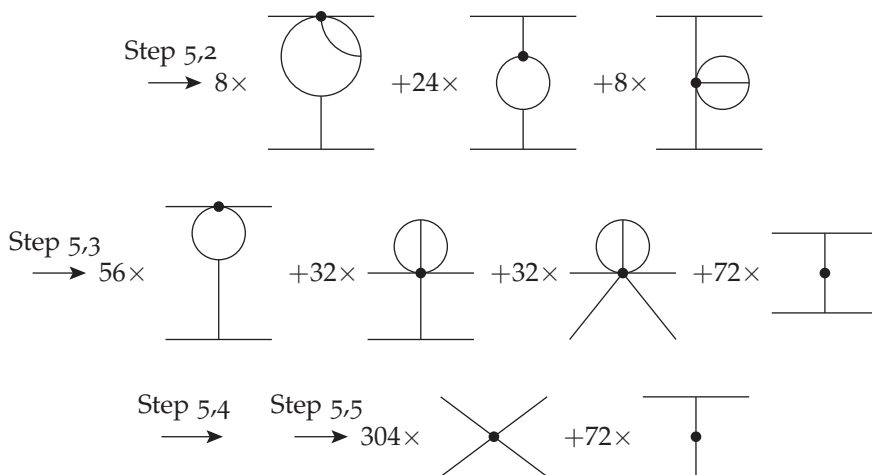
3.10.1 Self-energy filtering

An optimisation is to filter self-energy insertions from the QGRAF output. Here we present an algorithm that can detect one particle *reducible* propagator insertions.

1. Select a representative for a one-loop propagator. A representative is a single diagram that occurs in this propagator. For the ghost and the quark propagators this is trivial, since there is only a single diagram. For the gluon we select the diagram with the ghost loop (not forgetting the minus sign).
2. In the propagators we indicate the number of loops with an extra parameter. Adjacent loop representatives are combined and their number of loops is the sum of those parameters. This means that the representative of a three-loop gluon propagator is a chain of three one-loop diagrams, each with a ghost loop.
3. Next we make a copy of all remaining vertices into a function acc. In this function we remove all vertices that have an external line.
4. In the function acc we start selecting one vertex in all possible ways.
5. If this special vertex has more than two lines, it 'consumes' in all possible ways one of its neighbouring vertices, removing the connecting momentum. If the same momentum connects twice to the new vertex, it is removed as well.
6. We keep doing this until either the super-vertex in one of the terms has two lines remaining in which case we can eliminate the whole diagram as it is part of a propagator, or we cannot remove any more lines. If all possibilities end in the last way we keep the diagram.

Let us show this diagrammatically for a non-trivial diagram:





In the example, the diagram can be eliminated at the moment the super-vertex with just two lines appears. This is at step 5,3. We did not stop at that point because we wanted to show how the other possibilities develop for diagrams that would survive.

The above algorithm can be programmed rather easily in FORM with the new `id,all` option of the `id` statement. For instance step 4 is just the statement

```
id,all,v(?a) = w(?a);
```

in which v represents the vertices and w is the super-vertex. This is followed by a symmetrisation to reduce the number of different diagrams. A complete procedure that works for all types of diagrams, independent of the number of external lines or loops contains 30 FORM statements. The elimination of insertions simplifies the calculation considerably, because multi-loop gluon propagator insertions have many diagrams. This is particularly important when calculating moments of splitting and coefficient functions in DIS.

3.10.2 Colour split-off

We split each diagram in its colour part and its 'Lorentz' part before applying the Feynman rules. The 4-gluon vertex is split up into three terms with their own overall colour factor. Technically it is not required to do the split-off at this stage, but the remaining program will be considerably faster when the colour is a global factor.

To compute the colour factor we use a modified version of the `color` package of ref. [112]. It has been observed that even when one may have 100 000 diagrams or more, there are usually at most a few hundred different colour factors to be worked out. Hence, the way to process these factors is by pulling all colour objects into a function `color` and then, after using colour projectors on the external lines, only working out the colour bracket. The process is shown in listing 3.8.

LISTING 3.8: FORM code for bracketing in a function

```
Normalize color;  
B color;  
.sort: Prepare color;  
Keep brackets;  
Argument color;  
  #call color  
  #call simpli  
EndArgument;
```

Replacing every `.sort` by the procedure described in listing 3.9, guarantees that each different colour object is worked out only once.

LISTING 3.9: FORM code for a bracketed sort

```
#procedure SORT(text)  
  EndArgument;  
  B color;  
  .sort: 'text';  
  Keep Brackets;  
  Argument color;  
#endprocedure
```

3.10.3 *Diagram database*

Diagrams with the same topology and colour factor are grouped together in superdiagrams. The superdiagrams provide a convenient way to distribute the work over multiple computers. This grouping can speed up the calculation by a factor three.

We use the minos database program provided (with its source code) in the FORM pages to store the superdiagrams. After each superdiagram is computed, it is multiplied with its colour factors. Finally, the values of all superdiagrams are added. Only at this stage do we substitute the formulas for the insertion propagators and the master integrals. Up until the substitution of the master integrals the results are exact to all orders in ϵ if one uses rational polynomials in ϵ for the coefficients of the terms.

3.10.4 *Momentum substitutions*

After the Feynman rules have been applied, the integrals are in a form in which they can be converted to FORCER's basis for the topologies. The reduction to this basis needs to be done with great care as it is quite easy to generate an extremely

large number of terms. This process is split up into two components: rewriting the momenta to a momentum basis and rewriting the dot products to FORCER's basis.

The momentum basis should contain all the momenta of the irreducible dot products belonging to this FORCER topology. The other basis elements are obtained by an exhaustive search that tries to minimise the number of terms that will be created when rewriting to the basis. The optimisation criterion is the sum of the square of the number of terms that get created for all the momentum and dot product rewrites.

In order to prevent a blow-up in the number of terms, we create a layered rewrite of momenta. This layering is constructed automatically and makes the momentum rewrites order dependent:

<pre>p9.p?!{p9,}=+p2.p+p7.p+p11.p-Q.p; p5.p?!{p5,}=-p11.p-p3.p+Q.p; p6.p?!{p6,}=-p2.p+p3.p-p7.p; p1.p?!{p1,p4}=+Q.p-p8.p; p10.p?!{p10,}=+p2.p-p3.p; p4.p?!{p4,p1}=+p11.p+p3.p; p7.p?!{p7,}=+p8.p-p11.p;</pre>	\longrightarrow	<pre>p9.p?!{p9,}=-p6.p-p5.p; p5.p?!{p5,}=-p4.p+Q.p; p6.p?!{p6,}=-p10.p-p7.p; p1.p?!{p1,p4}=+Q.p-p8.p; p10.p?!{p10,}=+p2.p-p3.p; p4.p?!{p4,p1}=+p11.p+p3.p; p7.p?!{p7,}=+p8.p-p11.p;</pre>
---	-------------------	---

Because some terms will merge during the momentum rewrites, the layered approach is much faster. Note that dot products will not be rewritten if they are elements of the dot product basis.

Finally, the dot products are rewritten, straight to the internal FORCER notation. This is displayed in listing 3.10.

LISTING 3.10: FORM code for dot product rewriting

```
id Q.p1 = Q^2/2+i1/2-i8/2;
id p1.p2 = i1/2+i2/2-i9/2;
id p2.p3 = -i10/2+i2/2+i3/2;
id Q.p4 = Q^2/2+i4/2-i5/2;
id p3.p4 = -i11/2+i3/2+i4/2;
id p1.p3 = -Q^2/2+i11/2+i13+i14-i4/2+i5/2-i7/2+i8/2;
id p2.p4 = -Q^2/2-i1/2+i12+i13+i5/2-i6/2+i8/2+i9/2;
```

We note that in the actual code there will be `.sort` statements between the `id` statements and that there are extra optimisations in place to prevent excessive term generation.

3.11 EXAMPLES AND PERFORMANCE

The FORCER program has recently been used in many large calculations. As a first demonstration of its capabilities, the four-loop QCD beta function has been

recomputed [7, 9], and it agrees with refs. [113, 114]. Other major computations (refs. [2, 4, 5, 115, 116]) will be discussed in chapter 4 and chapter 6.

Below we demonstrate some benchmarks of the FORCER program. We start with some specific configurations, displayed in table 4. We have chosen top-level topologies for the benchmark, since these are the most time-consuming ones. In their reduction, many other master topologies (and thus custom reductions) are encountered. The topology la4 is the four-loop ladder topology.

ID	Configuration	Time (s)
no1	$Z(-14; 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, -1, -1, -1)$	10476
no2	$Z(-14; 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, -1, -1, -1)$	147
haha	$Z(-14; 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, -1, -1, -1)$	338
la4	$Z(-14; 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, -1, -1, -1)$	68
no2	$Z(-17; 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, -2, -2, -2)$	370
la4	$Z(-20; 2, 2, 3, 2, 2, 2, 2, 3, 2, 2, -2, -2, -3)$	2848
haha	$Z(-20; 2, 1, 2, 2, 1, 2, 1, 2, 2, 2, -4, -4, -4)$	12943
la4	$Z(-20; 2, 1, 2, 2, 1, 2, 1, 2, 2, 2, -4, -4, -4)$	117906

Table 4: Benchmark for several specific configurations, using 4 cores.

Next, we compute samples of configurations with a specific complexity of the top-level non-planar master integral no1 and no2. In figure 23, we show the total wall-clock time of computing 10 and 100 samples for a given complexity at the same time, using 4 cores. We observe that even though the difference in number of samples is a factor 10, the computation time increases only by about 20%. This demonstrates that the FORCER program makes use of symmetries and grouping, which cause shared configurations deeper in the reduction process to merge. Additionally, the graph shows that the computation time scales exponentially in the complexity with a base of about 2.5.

Finally, figure 24 shows the timings for computing four-loop QCD self-energies for a certain maximum power of the (unrenormalised) gauge parameter ξ . Here $\xi = 0$ corresponds to the Feynman gauge. In our setup, all techniques discussed in section 3.10 are applied.

The background field propagator in figure 24 can be used to obtain the beta function without computing an additional propagator and a vertex [117, 118]. Interestingly, the curve for the background-gluon is quite similar to that for the gluon, even though one may expect that the background-gluon to be more time consuming than the gluon propagator because of extra vertices. The high performance can be understood by the fact that we are using superdiagrams; as we have seen in figure 23, the increase of the number of terms does not matter much, provided complexities of integrals are similar, and there are many chances for merges and cancellations of coefficients of the integrals at intermediate stages in the reduction.

Using the background field method, we are able to compute the four-loop beta function for Yang-Mills theory with fermions in less than three minutes in the

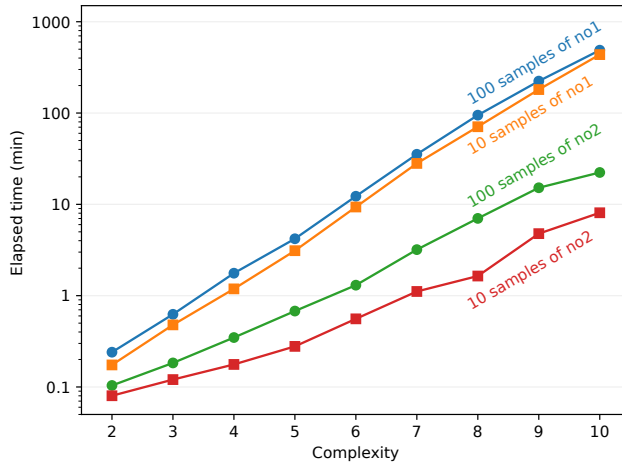


Figure 23: A benchmark (wall time) for the complete reduction of no1 and no2 configurations, using 4 cores (2.6 GHz). The line with the dots indicates the joint computation time of 100 sampled configurations, the line with the squares the computation time of 10 samples. Even though 10 times more integrals are computed, the computation time is only 20% longer. The scaling in complexity is exponential: each increase in complexity increases the computation time by 2.5.

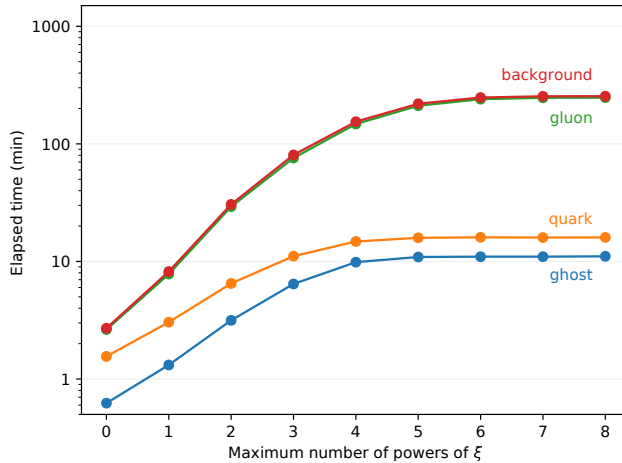


Figure 24: A benchmark (wall time) for computing four-loop QCD self-energies (ghost, quark, gluon and background-gluon) on a 32-core machine. The two curves for the gluon and background-gluon almost coincide.

Feynman gauge, and in about four hours for all powers of the gauge parameter on a single machine with 32 cores.

3.12 CHAPTER CONCLUSION

We are now able to answer

RQ2: *How can we construct a program that can compute four-loop massless propagator integrals more efficiently?*

We have shown how the FORCER program has been constructed, what algorithms it uses and demonstrated its performance [1]. We have derived the diamond rule, which is able to efficiently reduce integrals with a certain substructure [10]. Next, we have shown how to derive parametric reduction rules. In addition, we have outlined how FORCER may be used for computing physical diagrams.

3.12.1 Findings and main conclusion

We have shown that the FORCER program is able to compute the four-loop beta function in only three minutes on a 32-core machine. We have given benchmarks of configurations that would take months to compute with Laporta-like methods. Additionally, the FORCER program has already been used for some large calculations at four- and five-loop accuracy [2, 4, 5, 7, 9, 115, 116]. These computations will be discussed in chapter 4 and chapter 6. From our benchmarks and the use in large calculations, we may conclude that FORCER is an efficient program for computing four-loop massless propagator integrals.

3.12.2 Future research

An interesting area for future research is to extend FORCER to five loops. Most parts of the program are easily extended, since the predominantly automatic construction of the program is not limited to four loops. We have even computed 30% of the diagrams of the five-loop gluon propagator using the five-loop FORCER. The challenge is that over 200 topologies have an unknown reduction scheme. If the heuristics for deriving reduction schemes can be extended and fully automated, a full five-loop FORCER can be constructed shortly after. The idea is quite challenging: the number of parameters that have to be reduced grows from 14 at four loops to 20 at five loops.

