NIKHEF-H/91-18

NIKHEF

The FORM version of MINCER

S.A.Larin[1]
NIKHEF-H
P.O.Box 41882
1009 DB Amsterdam

F.V.Tkachov[2,3]
Fermi National Accelerator Laboratory
P.O.Box 500, Batavia, Illinois 60510 USA

J.A.M. Vermaseren
NIKHEF-H
P.O.Box 41882
1009 DB Amsterdam

Abstract

The program MINCER [1] for massless three-loop Feynman diagrams of the propagator type has been reprogrammed in the language of FORM [2]. The new version is thoroughly optimized and can be run from a utility like the UNIX *make*, which allows one to conveniently process large numbers of diagrams. It has been used for some calculations that were previously not practical.

[1] on leave from Institute for Nuclear Research (INR) of the Academy of Sciences of the USSR, Moscow 117312, USSR.

[2] on leave from Physics Department, McGill University, Montreal, Canada H3A 2T8 and Institute for Nuclear Research (INR) of the Academy of Sciences of the USSR, Moscow 117312, USSR (permanent address).

[3] Work supported in part by the Natural Sciences and Research Council of Canada and Quebec Committee of Education.

1 Introduction

The interaction between theory and experiment in particle physics relies for a large part on the ability of theorists to perform computations in perturbative field theory. The perturbation expansion runs in powers of $\hbar$ which in field theory corresponds to the number of closed loops in the Feynman diagrams. As each closed loop corresponds to an extra degree of freedom in space-time that has to be integrated over, each extra term in the expansion involves more integrals than the previous order, while the integrands become increasingly more complicated. Another important measure of complexity of multiloop Feynman integrals is the number of dimensionless parameters which the quantity being calculated depends on. Already the one-loop integrals can be very difficult when the particles involved have masses that cannot be ignored. Such masses result in complicated analytical dependences on several parameters (ratios of masses and external momenta). Exact solutions of these integrals are known [3], [4], [5], but they are very untransparent. The situation is much worse already for two-loop integrals involving masses—the most general integrals that can occur here have not yet been solved.

On the other hand, if the quantity to be calculated does not depend on dimensionless parameters (or if it depends on only one dimensional parameter, which is practically the same) then one can expect to be able to do all one-, two- and often some three-loop integrals analytically, and still higher orders can be attacked numerically. A good example is the calculation of g-2 in QED where many three-loop diagrams can be done analytically, while numerically one can go as far as four loops [6]. Another example is the calculation of the critical exponents of second order phase transitions by methods of quantum field theory, where calculations were done analytically through five loops [7].

In general, reduction in the number of parameters of the problem is achieved through one or another kind of asymptotic approximation. In this respect the situation is rather favourable in QCD where perturbation theory is expected to work only at high energies so that quark and hadronic masses can often be neglected, which simplifies the integrals greatly. Moreover, certain kinematic regions can be studied by using operator product expansions (OPE) and similar techniques (for a review see e.g. [8]) which allow one to further reduce the number of independent parameters of the problem. For example, calculation of coefficient functions of the short distance OPE can be reduced [9] to the so-called p-integrals—massless propagator-type (i.e. depending on only one non-zero external momentum) integrals obtained by differentiations in, and putting to zero, masses and momenta of more complicated diagrams.

The p-integrals were identified and studied in [10], [11] and [12]. It was shown [10] that a judicious use of integration by parts can reduce all two-loop p-integrals to convolutions of one-loop ones, and for the latter there exista a convenient closed formula in terms of Euler Γ-functions. Moreover, the same techniques allowed one to find an algorithm [11] which, theoretically, allows one to evaluate any three-loop p-integral analytically.

Practical implementation of the algorithm of [11] is, however, far from trivial. The algorithm consists of the application of a fairly large number of recursion relations in order to express an arbitrary p-integral in terms of simpler integrals that can be analytically evaluated in a more or less straightforward manner. Moreover, each diagram in realistic problems gives rise to a large number of terms after evaluation of traces over fermion lines and/or substitution of explicit expressions for gluon vertices. As a result, the volume of algebraic manipulations involved becomes prohibitive for hand calculations, and even the use of powerful symbolic manipulation programs like SCHOONSCHIP [13] or FORM [2] requires great care to make the algorithm work for realistic three-loop p-integrals.

The first implementation of the algorithms for p-integrals was done for SCHOONSCHIP. That package of programs was called MINCER [1] and used, among other problems, to evaluate the total crosssection for the process $e^+e^- \to hadrons$ to order α_S^3 [14], which took more then 200 hours of CPU time on a CDC 6500. Actually, in order to obtain a reliable result, that calculation had to be performed several times with some variations at the stage of reducing the original problem to p-integrals, which required a large amount of hand work. On the other hand, one could introduce an arbitrary gauge parameter for the same purpose[4] but then expressions exceed the size that could be reasonably managed by the old version of MINCER. Another important drawback of the implementation of [1] was that the rounding errors that are due to the fact that SCHOONSCHIP uses floating point arithmetic internally, would make it impossible to obtain meaningful results in calculations of OPE relevant for studies of deep inelastic lepton-hadron scattering.

Such were the motivations for reprogramming MINCER in the language of FORM. The new symbolic manipulation program FORM [2] is available on a variety of computers including ones that are much faster than the computers on which SCHOONSCHIP is available. Moreover, the language of FORM includes rather powerful pattern-matching and control-flow primitives as well as means for procedural structuring of programs. As a result, it proved possible to organize the recursions of the algorithm for p-integrals in more efficient manner. Thus, for the most complicated integral in the e^+e^- calculation a factor of 40 was measured between the original SCHOONSCHIP version on the CDC and the FORM version on a NeXTstation. This made it possible to perform calculations for some reactions [15], [16], [17] that would otherwise not be feasible. One stage of one of those calculations took as much as 154 CPU hours on an Apollo DN10000 computer which is nearly as fast as the NeXTstation for this type of computations.

The present paper describes the FORM implementation for the MINCER program. Its setup is somewhat different from the original SCHOONSCHIP version. Whereas the nature of the recursion relations dictates a certain structure for the inner layer of the program, there is more flexibility in the design of the interface layer. FORM version 2 has a more advanced interface with the operating system, so that it is rather easy to run the new implementation of MINCER from the UNIX utility *make*. For example, it is now relatively easy to set up a physical problem by specifying the whole set of relevant integrals in a single data base; a single procedure then defines the particular physics of the problem; after that the program can call the appropriate procedures that evaluate the particular integrals. It was thus possible to make a database with 378 diagrams and use it for a number of problems in studies of the deep inelastic lepton-hadron scattering in rather different ways.

The plan of the paper is as follows. Section 2 describes the auxiliary procedures of the lowest level—the ones used in calculations of convoluted one-loop p-integrals at the final stages of the algorithm. Section 3 deals with two-loop integrals and the so-called triangle recursion and its implementation. The algorithms for three-loop p-integrals are discussed in section 4. The routines constituting the package are listed and discussed in section 5. In section 6 various aspects of efficiency of the programs of the package are considered. The final section 7 contains examples of calculations that use all the routines of the package and thus can be used for checks and future references.

2 The lowest level

The innermost level of MINCER implements some basic identities and tabulated quantities. This includes the one-loop p-integrals and their reduction to polynomials in ϵ, the basic recursion relation for diagrams that have a triangular subgraph and the tables for normalized Γ-functions, as well as some integrals.

[4] such a parameter should cancel out in the final result, which constitutes a rather stringent test of the calculation.

2.1 One-loop p-integrals

The basic one-loop p-integral corresponds to the graph in fig.1 in which P and Q indicate the

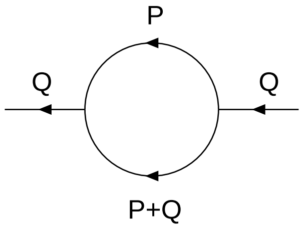

fig.1 The basic one-loop diagram

relevant 4-momenta. P is the so-called loop momentum that has to be integrated over. The integrand consists of powers of P^2 and $(P+Q)^2$ and an arbitrary number of individual momenta P^{μ_i}. First it should be noted that the integration is not exactly over 4-space but over a D-dimensional space with $D = 4 - 2\epsilon$ where we assume ϵ to be infinitesimally small. In addition we assume that the powers of P^2 and $(P+Q)^2$ are noninteger and may contain ϵ.

That this will be the case when two-loop p-integrals are reduced to convolutions of one-loop p-integrals can be seen on dimensional grounds. Indeed, even if we have integer powers in one p-integral, the fact that the integration measure $d^D P = d^{4-2\epsilon} P$ has a noninteger dimension causes the factor $(Q^2)^{-\epsilon}$ (where Q is the momentum that is external w.r.t. the one-loop subgraph) to appear in the result. Such factor may further combine with e.g. the "normal" propagator $1/Q^2$ to produce a propagator with a noninteger power in the remaining one-loop p-integral—cf. the discussion of the topology T2 in section 3.

In general the numerator of the integrand contains a polynomial in P, defined by

$$\mathcal{P}_n(P) = P_{\mu_1} P_{\mu_2} \cdots P_{\mu_n}. \tag{1}$$

The general solution of such one-loop p-integral is known and can be given in a closed formula [10]:

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

where $\Box = \partial^2/\partial P_\mu \partial P_\mu$ and G is given in terms of Γ-functions:

$$G(\alpha,\beta,n,\sigma) =$$
$$(4\pi)^\epsilon \, \frac{\Gamma(\alpha+\beta-\sigma-D/2)\Gamma(D/2-\alpha+n-\sigma)\Gamma(D/2-\beta+\sigma)}{\Gamma(\alpha)\Gamma(\beta)\Gamma(D-\alpha-\beta+n)}. \tag{3}$$

Note that we are interested not so much in G but in the ratio $G(\alpha,\beta,n,\sigma)$ $/G(1,1,0,0)$, because it is convenient to factor out powers of $\epsilon G(1,1,0,0)$, one per each momentum integration. Redefining the surface of the unit sphere in the space of $D = 4 - 2\epsilon$ dimensions so as to make this factor equal to 1, one obtains what is called the G scheme [18]. In the end one can easily convert the results to the standard $\overline{\text{MS}}$ scheme [19].

The main problem is now how to use the above equations in a practical sense. There are two ways to proceed.

2.2 Recursive evaluation of one-loop p-integral

The first method consists in defining a few recursion relations for the G's. Since G can be written in terms of Γ-functions one can relate G-functions with arguments differing by 1:

$$\begin{aligned}
G(\alpha,\beta,n,\sigma) &= \frac{1+\beta-\sigma-D/2}{(D/2-\alpha-\beta+\sigma)(D/2-\alpha+n-\sigma)}\,G(\alpha,\beta,n,\sigma-1), \\
G(\alpha,\beta,n,0) &= \frac{D/2-1-\alpha+n}{D-1-\alpha-\beta+n}\,G(\alpha,\beta,n-1,0), \\
G(\alpha,\beta,0,0) &= \frac{(\alpha+\beta-1-D/2)(D-\alpha-\beta)}{(\beta-1)(D/2-\beta)}\,G(\alpha,\beta-1,0,0) \\
G(\alpha,\beta,0,0) &= G(\beta,\alpha,0,0).
\end{aligned} \tag{4}$$

By means of these recursions one can reduce each G-function to one of the type $G(1+m\epsilon,1+n\epsilon,0,0)$. There are only four of these functions that we actually need: $(m,n)=(0,0),(1,0),(2,0),(1,1)$. The division by $G(1,1,0,0)$ normalizes the first to 1 and the other three can be expanded in terms of epsilon after they are written in terms of the Γ-functions. This expansion is needed to order ϵ^5 in calculations of the finite part of three-loop p-integrals. In practice one can give these p-integrals a name and substitute them in the end. This makes it easier to convince oneself that their expansion to $\mathcal{O}(\epsilon^5)$ was indeed deep enough.

Another problem with expansions is due to the presence of fractions in the recursion. The denominators are all of the type $m+n\epsilon$, so one can define a function e by:

$$e(x) = \frac{1}{1+x\epsilon} \tag{5}$$

and express the recursions in terms of this function. This results in many terms with different combinations of e's. The number of different terms, however, can be greatly reduced by means of partial fractioning:

$$\begin{aligned}
\epsilon\ e(x) &= \frac{1}{x}(1-e(x)) \\
\frac{e(x)}{\epsilon} &= \frac{1}{\epsilon} - x\ e(x) \\
e(x)\ e(y) &= \frac{x\ e(x) - y\ e(y)}{x-y} \qquad x \neq y.
\end{aligned} \tag{6}$$

By a systematic use of these relations one can transform the initial expression to a sum of terms containing either a power of ϵ but no e's or a power of a single $e(x)$. Since there is only a limited number of $e(x)$ with different x's, the volume of the expression can be considerably reduced.

This scheme has the important advantage that no expansions of e's are done at intermediate steps of the calculation. This minimizes the risk of a loss of precision due to insufficient number of terms retained in a power expansion of the e's. The e-functions are kept unexpanded till the end. The "fundamental" integrals are all finite (once $G(1,1,0,0)$ has been factored out), so in the end each e is in a term that has no powers of $1/\epsilon$ and all e's can be simply replaced by 1. Then the expansions for $G(1+m\epsilon,1+n\epsilon,0,0)$ can be substituted.

The main problem with this method is that it is relatively slow. Both the recursion and, especially, the partial fractioning take much time when the recursion generates many e's.

2.3 The tabulated Γ's

The second method is designed for speed. The fastest way would of course be to have all possible G's tabulated. Unfortunately this is not very practical. Because α and β can contain a part with ϵ there are 6 nearly free integer parameters. Even though some of them are restrained and the ϵ parts take only few different values, the number of G's can easily be more than 10000 when for instance higher moments of deep inelastic structure functions are studied. On the other hand all G's can be expressed in terms of Γ-functions and the normalized G's can be expressed in terms of normalized Γ-functions (so-called Pochhammer symbols). This function is defined by:

$$Po(n,x) = \Gamma(n+x\epsilon)/\Gamma(1+x\epsilon) \tag{7}$$

It can be tabulated best using a hybrid between a table and a function. Foreach value of n Po is tabulated as a function of x and an expansion in ϵ. We need the expansion to be accurate to $\mathcal{O}(\epsilon^6)$. In addition it is best to make separate expansions for the factors $1/Po(n,x)$ instead of inverting expansions for the factors in the denominator at run-time. This alone, however, is not enough to make the program fast. It could actually even make it slower! Indeed, if one G-function produces six Γ's then the expansion would lead to 7^6 terms.

There is, however, a more localized way to deal with this problem. Let us define a function A such that the normalized Γ is replaced by $A(Po(n,x))$ where Po represents the series expansion from a table. We thus obtain the product of 6 A's and, repeatedly applying the relation

$$A(x)A(y) = A(xy) \tag{8}$$

reduces it to just one A. The difference between this procedure and the straightforward multiplication of all Po is rather subtle: after each application of (8) FORM sorts the argument of the A on the r.h.s. and terms with excessive powers of ϵ (i.e. higher than ϵ^6) are discarded. For this reason each A has never more than 7 terms in the argument. This has to be done 5 times before all A's are reduced to a single A. This way of doing expansions avoids the buildup of 7^6 terms and eliminates the need for an overall sort, which leads to gains in time.

This is the method actually used in the program. Its main drawback is that now the depth of the expansion has to be specified in advance, and there is no check on whether this depth was sufficient. The advantage, however, is a superior speed. The gain in computer time increases even more when the higher deep inelastic moments are considered. In that case the coefficients become fractions with very big numbers in numerators and denominators and much time is spent on the arithmetic with these numbers. Once the expansions have been tabulated in advance, much of the worst work has been done.

2.4 Derivatives

Another tricky problem—which at first sight might look trivial—concerns evaluation of derivatives in the form of powers of the d'Alembertian on the r.h.s. of (3). Indeed, for higher powers of the d'Alembertian and a high value of n in $\mathcal{P}_n(P)$ the straightforward algorithm for derivatives based on the product rule generates a lot of like terms. For example, the product rule can be applied by replacing each p_{μ_i} by a noncommuting function $F_p(\mu_i)$. Then one multiplies the expression from the left by a derivative operator d_ν and repeats the substitution $d_\nu F_p(\mu) = \delta_{\nu\mu} + F_p(\mu)d_\nu$. In the end the d_ν that made it all the way to the right is replaced by zero. If one has to do this for three d'Alembertians and $n = 8$ one generates 20160 terms to obtain an answer with only 420 terms.

It is, however, possible to generate the appropriate tensor almost directly with no superfluous terms by using the special function `distrib_` in FORM-2 which can distribute indices over various functions. For the above example this makes a difference in speed of nearly a factor of 30. For higher values of n and more powers of the d'Alembertian the factor becomes even larger. For the simplest cases ($n \leq 2$), however, it is fastest to write the formula out by hand[5]. Therefore, the program first tests for the value of n and then selects either the 'simple' solution or the general one that constructs the resulting tensor directly.

The above method is described in more detail when the evaluation of harmonic projections for the three-loop p-integrals of the topology O4 is dicussed in section 4.1. It should be noted that general harmonic projections can be solved with this method quite efficiently. This is important for many problems involving higher moments of deep inelastic structure functions. Even so it should be clear that high order harmonic projections of tensors with a large value of n will eventually lead to expressions with a large number of terms.

3 Two-loop p-integrals

The treatment of two-loop p-integrals is more complicated as compared to theone-loop case and involves additional methods. There are three basic topologies of two-loop integrals: T1, T2 and T3. For integrals of each topology there is a special optimal way to treat them. Hence a separate routine was created for each topology. Let us consider them in turn.

The topology T1 has no trivial subgraphs and appears to be an irreducible integral that would have to be done by a completely new method.

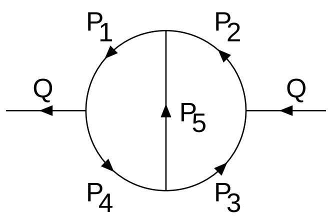

fig.2 The topology T1.

There does, however, exist a rather simple method [10], [11] to reduce p-integrals of this type to simpler integrals of the types T2 and T3. This method relies on the existence of identities (obtained using integration by parts within dimensional regularization) which relate different integrals within one topology and even integrals with different topologies. For the present case the method can be summarized in the form of the 'rule of triangle'.

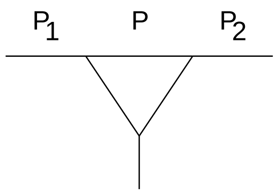

fig.3 The triangle subgraph.

[5]This is a first example of a general tactics that is often used in what follows: for simple cases one adopts a straightforward approach which, though less efficient in general, carries less overhead, while the more complicated cases are treated using more sophisticated algorithms.

In the notation of fig.3, one of the subintegrals can be written as:

$$I(\alpha_0,\beta_1,\beta_2,\alpha_1,\alpha_2) = \int d^DP\, \frac{1}{(P^2)^{\alpha_0}((P+p_1)^2)^{\beta_1}(p_1^2)^{\alpha_1}((P+p_2)^2)^{\beta_2}(p_2^2)^{\alpha_2}}. \tag{9}$$

This integral can be studied by first considering the following identity:

$$\int d^DP \left\{ \frac{\partial}{\partial P_\mu} \frac{P_\mu}{(P^2)^{\alpha_0}((P+p_1)^2)^{\beta_1}(p_1^2)^{\alpha_1}((P+p_2)^2)^{\beta_2}(p_2^2)^{\alpha_2}} \right\} = 0. \tag{10}$$

Working out the derivative one obtains the recursion relation:

$$\begin{aligned} I(\alpha_0,\beta_1,\beta_2,\alpha_1,\alpha_2) = (& \\ &+\beta_1(I(\alpha_0-1,\beta_1+1,\beta_2,\alpha_1,\alpha_2) - I(\alpha_0,\beta_1+1,\beta_2,\alpha_1-1,\alpha_2)) \\ &+\beta_2(I(\alpha_0-1,\beta_1,\beta_2+1,\alpha_1,\alpha_2) - I(\alpha_0,\beta_1,\beta_2+1,\alpha_1,\alpha_2-1)) \\ &)/(D-2\alpha_0-\beta_1-\beta_2). \end{aligned} \tag{11}$$

This relation can be used to make one of the powers α_i equal to zero,provided that the α_i were integers at the start. When α_0 is eliminated we obtain an integral of type T3, and when either α_1 or α_2 is eliminated the remaining integral is of type T2. Those integrals, as we will see, are fairly easy. This recursion is the cornerstone of the whole treatment of two- and three-loop diagrams.

Let us turn to the integrals of type T2 which can be represented by the following diagram:

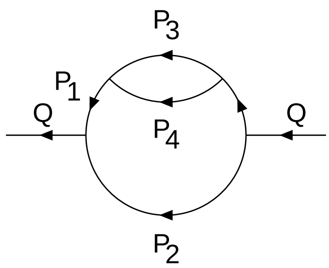

fig.4 The topology T2.

This integral is rather trivial. It is a convolution of two one-loop integrals. One should note, however, that when the inner integral has been done, the outer integral has a line whose power involves an ϵ. This is an example of how non-integer powers of propagators are generated.

The integrals of type T3 are even easier. The diagram is as follows:

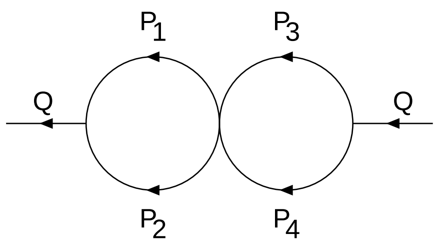

fig.5 The topology T3.

It is essentially a direct product of two one-loop p-integrals and its evaluation is straightforward.

The above algorithms are, in principle, sufficient for the evaluation of an arbitrary two-loop p-integral. In practice, however, there remain some problems in connection with the integrals of type T1. The first problem is the nastiest. Although the coefficient by which we divide in (11) can never be zero, it can be proportional to ϵ. Hence it is possible to generate one power of $1/\epsilon$ this way. Normally one will not generate more than one power of $1/\epsilon$ in the recursion and hence things will work out fine. The problem emerges when there are extra powers of P in the numerator. When we define the integral

$$\begin{aligned} I(n,\alpha_0,\beta_1,\beta_2,\alpha_1,\alpha_2) \\ = \int d^DP\, \frac{P_{\mu_1}\cdots P_{\mu_n}}{(P^2)^{\alpha_0}((P+p_1)^2)^{\beta_1}(p_1^2)^{\alpha_1}((P+p_2)^2)^{\beta_2}(p_2^2)^{\alpha_2}}, \end{aligned} \tag{12}$$

then the recursion relation picks up an extra term and the denominator becomes

$$D+n-2\alpha_0-\beta_1-\beta_2. \tag{13}$$

The effect of n is that during the recursion it is possible to generate $1/\epsilon$ more than once. This would present no problem if we were not working with a truncated power series in ϵ. Although it can be shown that the total expression—after the recursion has been completed and all terms added—may contain no more than a certain limited power of $1/\epsilon$, this may not be true for individual terms at intermediate steps. Hence if each term is multiplied by a power series which contains a power series in ϵ truncated at ϵ^6, there may eventually occur a loss of precision in the power expansion in ϵ. This indeed happened during the testing of an early version of the program.

There are several solutions to this problem. The first would be to make theexpansions deeper, at the cost of much computer time, not only due to the increased number of terms, but also due to the fact that the terms with the higher powers of ϵ have much more complicated coefficients. Another solution would be to go back to the scheme with the function e as explained in the section on the one-loop integral. In that scheme no approximations with respect to the order of expansions are done. That scheme, however, is much costlier in terms of computer time.

The actual solution, as implemented in the current version of the program, both saves computer time and eliminates the problem once and for all. It is based on the explicit solution for the recursion (11), that was found in [12] and has the following form:

$$\begin{aligned}
& I(n,\alpha_0,\beta_1,\beta_2,\alpha_1,\alpha_2)\Gamma(\alpha_1)\Gamma(\alpha_2) = \\
& + \sum_{i=0}^{\beta_2-1}\sum_{j=0}^{\beta_1-1}\sum_{k=0}^{\beta_1-j-1}(-1)^{\alpha_0+i} I(n,0,\beta_1-j-k,\beta_2-i,\alpha_0+\alpha_1+j,\alpha_2+i+k) \\
& \quad \frac{\Gamma(\alpha_0+i+j+k)\Gamma(\alpha_0+\alpha_1+j)\Gamma(D+1+n-2\beta_1-\alpha_0-\alpha_1-\alpha_2-i)}{\Gamma(D+1+n-2\beta_1-\alpha_1-\alpha_2+j+k)\Gamma(\alpha_0)\; i!\; j!\; k!} \\
& \qquad \times\Gamma(\alpha_2+k+i) \\
& + \sum_{i=0}^{\alpha_0-1}\sum_{j=0}^{\beta_2-1}\sum_{k=0}^{\beta_1}(-1)^{i+j} I(n,\alpha_0-i,0,\beta_2-j,\alpha_1+i+k,\alpha_2+\beta_1+j-k) \\
& \quad \frac{\beta_1\Gamma(\beta_1+i+j)\Gamma(D+n-2\beta_1-\alpha_1-\alpha_2-i-j)\Gamma(\alpha_1+k+i)}{(\beta_1-k)!\; i!\; j!\; k!\; \Gamma(D+n-\beta_1-\alpha_1-\alpha_2)} \\
& \qquad \times\Gamma(\beta_1+\alpha_2-k+j) \\
& + \sum_{i=0}^{\alpha_0-1}\sum_{j=0}^{\beta_1-1}\sum_{k=0}^{\beta_1-j-1}(-1)^{i+\beta_2} I(n,\alpha_0-i,\beta_1-k-j,0,\alpha_1+i+k,\alpha_2+\beta_2+j)
\end{aligned}$$

$$\frac{\Gamma(D+1+n-2\beta_1-\beta_2-\alpha_1-\alpha_2-i)\Gamma(i+j+k+\beta_2)\Gamma(\alpha_1+i+k)}{\Gamma(D+1+n-2\beta_1-\alpha_1-\alpha_2+k+j)\Gamma(\beta_2)i!\ j!\ k!} \times\Gamma(\beta_2+\alpha_2+j). \tag{14}$$

The advantage of this formula is that there are no direct cancellations between the terms and hence the spurious poles in ϵ cannot occur. All terms are unique, so that exactly the right number of terms is generated. The only disadvantage is that for simple cases the implementation of the formula carries more overhead than the straightforward recursion.

Hence, the best solution is to use the original recursion for the simple cases that cannot develop the spurious poles, and to use the resolved recursion for all other cases.

The other problem with integrals of the type T1 is as follows. When evaluating three-loop p-integrals one can obtain two-loop p-integrals with one denominator having a noninteger power. This causes no problems when the noninteger power is on one of the peripheral denominators. One can choose the triangle so that the noninteger power is inside the triangle, and the recursion will have the desired effect. If on the other hand α_0 is noninteger, then it is impossible to apply the recursion (11) to make the power of the central denominator become zero.

It is still, however, possible to find recursions similar to (11) to do the job. We can apply two extra types of recursions. The first recursion is obtained by applying the triangle rule with one of the peripheral momenta playing the role of the integration momentum, in place of the central one:

$$\begin{aligned}(\beta_1-1)I(\alpha_0,\beta_1,\beta_2,\alpha_1,\alpha_2)Q^2 = \\ &+(\alpha_0+\beta_1+2\beta_2-D-1)I(\alpha_0,\beta_1-1,\beta_2,\alpha_1,\alpha_2)\\ &+(\beta_1-1)I(\alpha_0,\beta_1,\beta_2-1,\alpha_1,\alpha_2)\\ &+\alpha_0(I(\alpha_0+1,\beta_1-1,\beta_2-1,\alpha_1,\alpha_2)\\ &\quad -I(\alpha_0+1,\beta_1-1,\beta_2,\alpha_1,\alpha_2-1)).\end{aligned} \tag{15}$$

This gives a recursion that lowers the sum of the powers in the peripheral denominators at least by one, provided we started with a denominator with at least two powers. Hence this recursion can be used to reduce all powers of the peripheral denominators to one (or zero). This then leaves integrals in which the internal propagator has a noninteger power of the type $(n+\epsilon)$ where n can in principle take any integer value. With one more recursion one can reduce this integral to integrals of the types T2, T3 and $n=1$. Such a recursion can be set up in one of several ways. One is by applying the original rule of triangle (11) if n needs to be lowered, and a trivially modified version if n needs to be raised. A direct recursion has been derived in [11] and is given by:

$$\begin{aligned}(\alpha_0-1+\epsilon)I(\alpha_0,1,1,1,1) = \\ &-(\alpha_0-2+2\epsilon)I(\alpha_0-1,1,1,1,1)Q^{-2}\\ &-(3-3\epsilon-2\alpha_0)G(1,\alpha_0,0,0)G(1,\alpha_0+\epsilon,0,0)Q^{-2(2+\alpha_0)}.\end{aligned} \tag{16}$$

The integral with $n=1$ is not known exactly, but its value to the desired accuracy $\mathcal{O}(\epsilon^2)$ can be derived using the formulae of [11]:

$$I(1+\epsilon,1,1,1,1) = (6\zeta_3+9\zeta_4\epsilon+102\zeta_5\epsilon^2+\cdots)(1-2\epsilon) \tag{17}$$

In ref. [20] this integral was worked out to $\mathcal{O}(\epsilon^3)$ and in [21] the integral was expressed as a sum over gamma functions and the term $O(\epsilon^4)$ was calculated. The actual solution we use for the last recursion is that we tabulate these integrals in a range of values of n that is more than sufficient. This means that the above recursion and the one-loop integrals after it have to be executed only once when the table is made. This saved a considerable amount of computer time.

4 Three-loop p-integrals

The three-loop p-integrals have more topologies than the two-loop p-integrals. We distinguish the topologies LA (for ladder), BE (for Benz), NO (for non-planar), BU, FA, Y1, Y2, Y3, Y4, Y5, O1, O2, O3 and O4. Some of these topologies are rather trivial, others are very difficult to deal with. Contrary to the original MINCER program we treat all these topologies separately. This has the advantage that each topology routine can be optimized independently. In the original MINCER program it was decided to handle the simpler cases using the more universal routines due to complexity of that program and difficulties of debugging. This resulted in much overhead. Programming with FORM proved to be so much easier that it took very little effort to provide individually optimized routines for the simpler topologies. We will deal with them later on in section 4.7.

4.1 The topology O4

In its most general form this topology is given in fig.6:

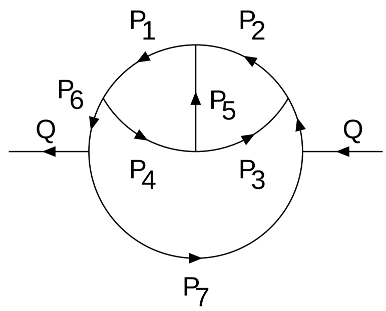

fig.6 The topology O4

In principle, this topology is a trivial convolution of a two-loop p-integral of type T1 and a one-loop integral. But that would not be the most economical way to set up the routine. It is actually best to do the outer integral over p_6 first. This is possible due to the fact that the inner two-loop integral will give an expression whose dependence on p_6 can be determined beforehand using Lorentz and scale covariance. This is particularly simple if the numerator of the inner two-loop subintegral does not contain dotproducts involving momenta that are external with respect to the subintegral because the latter will then produce just a (non-integer) power of p_6^2. Although integrals of this type account for a majority of terms in actual calculations, one still has to decide what to do with those dotproducts in the initial integrand that cannot be reexpanded in terms of the propagators of the initial topology.

The solution (see the appendices in [11]) is to notice that any symmetric traceless tensor structure in the integrand remains such after integration. Therefore, if one represents the numerator of the two-loop subintegral as a linear combination of such tensors one can then proceed to evaluate the outer one-loop integral prior to doing the inner two-loop one. We will describe the whole procedure in some detail because its implementation is not entirely trivial.

First, all dotproducts—with the exception of $Q.p_1$ and $Q.p_2$—are reexpanded in terms of p_i^2 and Q^2; for example, one can replace $p_7 \to Q - p_6$ in the numerator and then rewrite all dotproducts in the numerator that contain p_6 in terms of the p_i^2 and Q^2. After that $p_i.p_j$ can be reexpanded exactly as in the generic two-loop case (topology T1) and we have only $Q.p_1$, … $Q.p_5$ left. After the inner loops would be integrated out these dotproducts would be transformed into powers of

$Q.p_6$ and $Q^2 p_6^2$ so that the remaining one-loop integral would assume a rather simple form allowing a straightforward integration using the formulae of the one-loop case.

Before we present an explicit formula for doing the outer one-loop integration, it should be noted that if at this point (i.e. after reducing the numerator to $Q.p_1, \ldots Q.p_5$) one of the denominators is missing we have a simpler integral and in principle we could pass on to a different topology. This is not needed though, because the general procedure described here works equally well for all topologies of the two-loop subintegrals and allows one to get rid of complicated numerators in one sweep.

The outer one-loop integration is implemented as follows. For technical reasons we replace all $Q.p_i$ by $x\ R(p_i)$. The scalar symbol x is used as a bookkeeping device to tell us how many powers of p_i there are in the numerator. The outer one-loop integral is then done as follows[6]:

$$\begin{aligned}\frac{x^n d^D p_6}{(p_6^2)^{x_1}(p_7^2)^{x_2}} &= \frac{1}{(Q^2)^{x_1+x_2-D/2}}\sum_{s=0}^{n/2} G(x_1-s, x_2, n-2s, 0) \\ & \sum_{j=0}^{n/2-s}\frac{\Gamma(n+2-\epsilon-2s)\Gamma(n+1-\epsilon-2s-j)(-1)^j}{\Gamma(n+2-\epsilon-s)\Gamma(n+1-\epsilon-2s)\ s!\ j!} \\ & \times\left(\frac{Q^2\Box_R}{4}\right)^{s+j}\end{aligned} \tag{18}$$

The d'Alembertian $\Box_R$ acts only on the function R. If it is defined by $\Box_R = \partial^\mu\partial_\mu$ then $\partial^\mu R(p_i) = p_i^\mu$.

At this point we have to worry about evaluating derivatives. How one can do that was explained in general terms in sect.2.4. Now we wish to present more detail since occurence of high-order derivatives is rather usual in three-loop integrals (due to more complicated numerators) while it is rare in the one-loop case (but it does happen!).

It seems unavoidable to present some code:

```
repeat id R(??)*R(???) = R(..,...);
id  dal^j?*R(??) = 2^j*fac_(j)*distrib_(1,2*j,f1,f2,..);
repeat id f2(i1?,??) = Q(i1)*f2(..);
  id f2 = 1;
  repeat;
    id f1(i1?,??) = f3(i1)*distrib_(1,1,f3,f1,..);
    id f3(i1?)*f3(i2?) = d_(i1,i2);
  endrepeat;
  id f1 = 1;
```

The first `repeat` statement combines all occurrences of R into one tensor R with n indices (or contractions with p_i which is nearly the same). This can be done in a single statement because FORM provides a rather powerful primitive for pattern-matching involving lists of arguments of indefinite length. Then the j powers of the d'Alembertian will lead to $2j$ indices being absorbed by j pairs of Kronecker delta's. These $2j$ indices are extracted in all possible ways and put into the argument field of the function $f1$, while the remaining indices are put into function $f2$. This is done by means of the `distrib_` function and the result contains $\binom{n}{2j}$ terms. There is of course an overall factor. The indices in $f2$ are now written back into contractions with the vector Q. Then the indices in $f1$ are distributed in all possible ways over Kronecker δ's (`d_`). Such `d_`'s are taken out one at a time, till there is no more. As explained above, this method generates exactly the right number of different terms.

[6] The relevant formula (A.16) from [11] contains a misprint.

With the above techniques p-integrals of the type O4 are reduced to two-loop integrals that can be dealt with in a general two-loop routine.

4.2 The topology LA

This topology is also called the ladder topology. Its generic notation is given in fig.7:

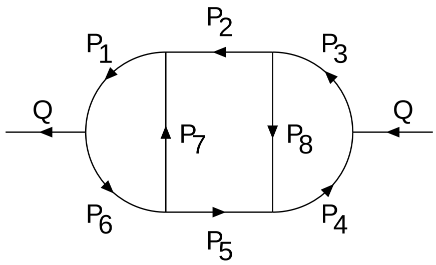

fig.7 The ladder or LA topology.

It is relatively easy to deal with, because either the left side or the right side can be treated with the triangle routine. SInce the diagram is so symmetric the best strategy is to look at the powers of the propagators with the momenta p_7 and p_8. If there are fewer powers of p_7^2 in the denominator than of p_8^2 one can apply a left-right symmetry operation to give the triangle routine as little work as possible. The triangle is then applied to the right side, involving p_8, p_3, p_4, p_2 and p_5. Of course, the application of the triangle routine only works well if the only occurrence of the momenta in the loop is in the denominators (with the exception of p_8 which can occur in the numerator). Hence the first part of a ladder routine should be to reexpand all dotproducts in the numerator in terms of squares of the various four-vectors.

The scheme for this is rather trivial: first p_1, p_3, p_4 and p_6 are rewritten in terms of the momenta of the inner box. Then the dotproduct of two adjacent momenta can be rewritten in terms of squares as in the following expression:

$$p_2.p_7 = \frac{1}{2}(p_1^2 - p_2^2 - p_7^2). \tag{19}$$

At this point the relation $Q = p_2 - p_5$ can be used, eliminating all dotproducts between these three vectors. Finally $Q.p_7$ and $Q.p_8$ can be removed by using for instance $p_7 = p_1 - p_2$ and the fact that $Q.p_2$ and $Q.p_1$ can be rewritten trivially. This leaves only squares and the dotproduct $p_7.p_8$ which is irreducible.

The reduction of the momenta has to be done very carefully. For the more complicated diagrams this can be the most time consuming step. It also involves the largest expansion of intermediate expressions. In this and similar cases it proved best to insert sort operations after each few substitutions.

After application of the triangle routine either the denominator with p_2^2 or p_5^2 or the denominator with p_8^2 will have disappeared, resulting in a FA topology or the trivial O3 topology which will be treated later on. If before the application of the triangle one (or more) of the denominators was missing already we would already have a simpler topology and no action would be necessary.

4.3 The topology BE

The diagram in fig.8 represents the generic Benz topology which is abbreviated to BE.

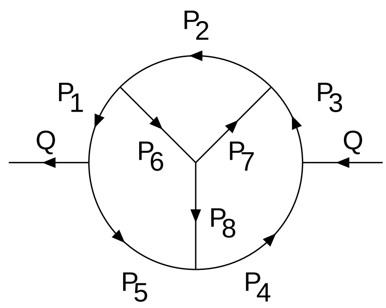

fig.8 The Benz or BE topology.

Of the three highest level topologies this one is conceptually the easiest, because the triangle is so clearly visible. Using techniques similar to the one explained for the topology LA we reduce the dotproducts in the numerator to combinations of squares and the single mixed dotproduct $Q.p_2$. Since we use p_2 as the central momentum in the triangle this does not complicate the recursion. The triangle routine will eliminate either p_2^2 or p_1^2 or p_3^2 from the denominators leaving either a topology of the trivial type O1 (which will be considered later) or a topology of the type BU. In principle we could have used either p_6 or p_7 as the central momentum in the triangle, but that would have resulted in topologies of the types FA and BU, and FA still needs extra attention before we can do the first integration. This is not the case for O1 which is ready for the first integration.

As before, when any of the propagators are missing after the dotproducts have been rewritten we start out with a simpler topology. This can be a topology of the types BU, O1, O2, O4 or FA. If more than one propagator is missing the topology can be even simpler.

4.4 The topology BU

This topology can in principle occur in either of two ways.

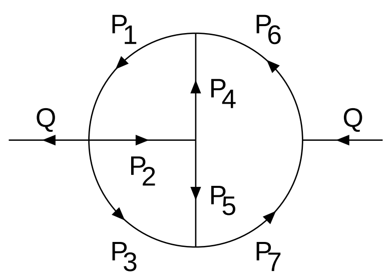

fig.9 The topology BU.

A diagram can have this topology from the very beginning, or it can emerge as a result of the triangle operation in the topologies BE or NO. In the first case we can just proceed as if the diagram had a topology BE; the reduction of the momenta in the numerator would follow the same algorithms and eventually it would look as a reduced BE topology. This scheme looses very little in efficiency over a completely dedicated routine. Therefore, we have to consider this topology only as a derivate from higher topologies for which there are only a few special dotproducts that still survive in the numerator.

Because we can apply the triangle routine either to the 1-4-2 triangle or to the 2-5-3 triangle it is best to make sure that the only nontrivial dotproducts in the numerator are $Q.p_4$ and $Q.p_5$.

The second irreducible dotproduct is due to the fact that we have to rewrite the momentum of the missing denominator in terms of other momenta. The number of powers of p_6^2 and p_7^2 will determine whether it is more economical to do the upper triangle or the lower triangle. Assuming that there are fewer powers of p_6^2 in the denominator we do the upper triangle and either p_6^2, or p_4^2 or p_5^2 gets eliminated, resulting in a topology O4 or O2. The last topology is trivial and will be discussed later (a topology is called trivial if one of the integrals can be performed directly) and O4 has been discussed before.

4.5 The topology FA

As in the case of the BU topology we will only consider this topology as a

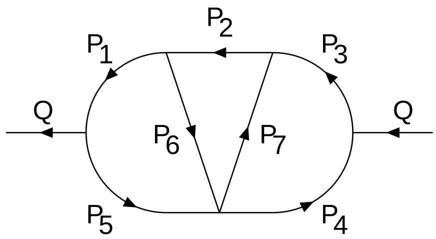

fig.10 The topology FA.

derivate of either the BE or the LA topology. When this topology is encountered in an actual diagram we will use the reduction scheme that is also used for the BE topology which it resembles most. The only complication with this topology is that when it occurs as a derivate of the topology LA there can be the dotproduct $p_6.p_7$ which comes from $p_7.p_8$ in the LA topology, while if we deal with a BE derivate there may still remain p_8^2 in the numerator. The p_8^2 is rather harmless. It can be rewritten in terms of p_4 and p_5, so it does not affect the triangle reduction. For this reason we can eliminate the $p_6.p_7$ by reexpanding it in terms of p_6^2, p_7^2 and p_8^2. Then we apply the triangle routine and, finally, p_8^2 is reexpanded in terms of p_6 and p_7 again. This will eliminate either p_2^2 or p_1^2 or p_3^2 or p_6^2 or p_7^2. The last two can be eliminated in some terms due to the reexpansion of dotproducts. The resulting topologies are all of the trivial types O1, O2 and O3.

4.6 The topology NO

The NO or non-planar topology is the most complicated one by far.

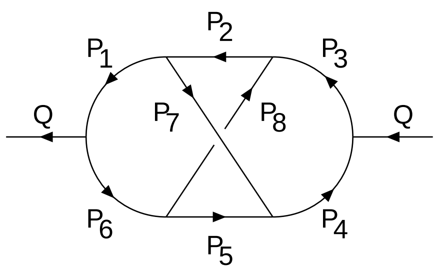

fig.11 The non-planar or NO topology.

The major impediment is due to the fact that there are no triangle subintegrals here and hence it is impossible to apply the triangle routine. On the other hand, this topology possesses a large number of symmetries which can be used to reduce the number of operations that has to be programmed. In addition, the symmetries can somewhat reduce the number of terms that have to be processed, thereby saving some computer time.

The absence of triangle subgraphs forces now a complicated scheme in which several similar relations for box graphs are used. Such relations are less useful by themselves, but a combination of several box identities can eliminate offending terms and yield useful relations. This is explained in [11].

The algorithm consists of two major parts. First, one reexpands the numerator in terms of a minimal set of scalar products and uses integration-by-parts identities to get rid of the irreducible scalar product. One thus obtains a sum of nontrivial integrals of the non-planar topology without numerators[7]. In the second part these 'scalar' integrals are reduced to simpler topologies by means of a set of recursions.

To accomplish the first part, we have the choice of two schemes: the one described in [11] and that of [1].

The scheme implemented in the original MINCER program was supposedly less advantageous in terms of overall efficiency of MINCER than the one described in [11], as the latter results in a lesser number of, and simpler, terms to be processed in the second part. The reason for such a decision was the fear that the scheme of [11], containing more—and more complicated—recursions, was much harder to debug, because for this type of algorithms the old version of SCHOONSCHIP is rather hard to program. Hence considerations of reliability won over run-time efficiency. In the FORM version we have experimented with both schemes. The conclusion is that for very complicated integrals the scheme of [11] is inherently faster, but for actual diagrams this advantage is often rather insignificant. We will describe the two schemes here as they have been implemented. Both routines are provided with the package.

The scheme of the original MINCER algorithm [1] begins, as usual, with reexpansion of the dotproducts in terms of an independent set. This is not quite as easy as it was for the LA and BE topologies. The first steps are still straightforward: First p_5 is rewritten as $p_6 - p_8$, unless it occurs in a square. Then p_4, p_6, p_7 and p_8 can be eliminated, again unless they occur already in squares. In FORM each of these can be done in a single statement:

```
id  p7.p?!{p7} = p2.p-p1.p;
```

where the wildcarding excludes the set with the single element p_7. The dotproducts with two vectors that flow through lines incident to the same three-line vertex are also easy, so this leaves the two mixed dotproducts $Q.p_2$ and $p_1.p_3$. The elimination of this last dotproduct is more difficult. It is given by the relation:

$$p_1.p_3 = \frac{1}{2}(Q^2 + p_1^2 + p_2^2 + p_3^2 + p_5^2 - p_4^2 - p_6^2 - p_7^2 - p_8^2) - Q.p_2\,. \tag{20}$$

Whether we use the reduction scheme with $Q.p_2$ or with $p_1.p_3$ hardly makes any difference. We define the integral N by:

$$N(\alpha_1, \cdots, \alpha_8, \alpha_9) = \int \frac{d^D p_1\, d^D p_2\, d^D p_3\ (Q.p_2)^{\alpha_9}}{(p_1.p_1)^{\alpha_1} \cdots (p_8.p_8)^{\alpha_8}}. \tag{21}$$

The recursion now is as follows:

$$N(\alpha_1, \cdots, \alpha_9) \quad = \quad \Big(+\frac{1}{2}(D - 1 - \alpha_1 - 2\alpha_2 - \alpha_3 + \alpha_9)\mathbf{9}^-$$

[7]plus, as usual, terms with derivative, simpler topologies for which the presence of a numerator is inessential.

$$+\frac{1}{2}\alpha_1\mathbf{1^+7^-9^-} - \frac{1}{2}\alpha_1\mathbf{1^+2^-9^-}$$
$$+\frac{1}{2}\alpha_3\mathbf{3^+8^-9^-} - \frac{1}{2}\alpha_3\mathbf{3^+2^-9^-}$$
$$+\frac{\alpha_9-1}{2}\mathbf{2^-9^-9^-}\Bigg)\Bigg/\left(2D-1+\alpha_9-\sum_{i=1}^{8}\alpha_i\right). \qquad (22)$$

The notation here is somewhat schematic: $\mathbf{1^+7^-9^-}$ means that in N the α_1 should be incremented by one, the α_7 should be decremented by one and the α_9 should also be lowered by one. This equation can be applied as many times as needed to eliminate all powers of $Q.p_2$. Note that there will never occur negative powers of $Q.p_2$ because the term that could generate such a negative power will have a zero coefficient. One should also note that in the denominator, the factor ϵ can be generated—but only once, whatever the starting values of the α's. If this were not the case we would have the same problems that forced us to use the explicit solution of the triangle relation in the previous section.

In the scheme of [11] the extra irreducible scalar product is chosen to be $p_2.p_5$ which is slightly more symmetric than $Q.p_2$ used above. The reduction of scalar products in the numerator to the irreducible ones can be conveniently done in the same way as in the previous scheme but the following two substitutions should be used in the end:

$$\begin{aligned} p_1.p_3 &= (p_2^2 - p_4^2 + p_5^2 - p_6^2 + Q^2)/2 + p_2.p_5, \\ Q.p_2 &= (p_1^2 + p_3^2 - p_7^2 - p_8^2)/2 - p_2.p_5. \end{aligned} \qquad (23)$$

These substitutions should be used only when all 8 denominators are present. If at least one denominator is missing we use the old substitution that results in $Q.p_2$. For those terms no reduction is needed at this stage. If $p_2.p_5$ is used consistently the algorithm may actually be slower in many cases, because the rewriting is more costly for the 'spectator' integrals and afterwards we have to write everything in terms of $Q.p_2$ anyway.

Next we start a giant sequence of five recursions that work as long as there is a positive power of $p_2.p_5$. The first recursion is executed as long as p_1^2, p_3^2, p_4^2 or p_6^2 have a power that is less than -1:

$$\begin{aligned} N(\alpha_1,\cdots,\alpha_9) &= +\frac{1}{2}(\mathbf{3^-9^-} - \mathbf{2^-9^-} - \mathbf{8^-9^-}) \\ &\quad +\frac{1}{2(\alpha_6-1)}(D-1-\alpha_1-2\alpha_2-\alpha_8+\alpha_9)\,\mathbf{6^-9^-} \\ &\quad +\frac{\alpha_1}{2(\alpha_6-1)}(\mathbf{1^+6^-7^-9^-} - \mathbf{1^+2^-6^-9^-}) \\ &\quad +\frac{\alpha_8}{2(\alpha_6-1)}(\mathbf{3^-6^-8^+9^-} - \mathbf{2^-6^-8^+9^-}). \end{aligned} \qquad (24)$$

We have assumed here that α_9 refers to powers of $p_2.p_5$. Repeated use of this and its three variants for α_1, α_4 and α_3 will eventually result in an integral either with one of its denominators missing, or with $p_2.p_5$ missing in the numerator, or with all powers of p_1^2, p_3^2, p_4^2 and p_6^2 equal to -1. After this the second recursion is used:

$$\begin{aligned} &N(1,\alpha_2,1,1,\alpha_5,1,\alpha_7,\alpha_8,\alpha_9) = \\ &\qquad -\tfrac{D-1+\alpha_9-2\alpha_2-\alpha_7-\alpha_8}{2(\alpha_5-1)}\mathbf{5^-9^-} + \tfrac{\alpha_9-1}{2(\alpha_5-1)}\mathbf{2^-5^-9^-9^-} \\ &+\tfrac{\alpha_7}{2(\alpha_5-1)}(\mathbf{2^-5^-7^+9^-} - \mathbf{1^-5^-7^+9^-}) + \tfrac{\alpha_8}{2(\alpha_5-1)}(\mathbf{2^-5^-8^+9^-} - \mathbf{3^-5^-8^+9^-}). \end{aligned} \qquad (25)$$

This recursion and its variant for α_2 will either remove $p_2.p_5$, or remove one of the denominators entirely or leave the powers of p_2^2 and p_5^2 equal to -1. The third and fourth recursions are as follows:

$$
\begin{aligned}
&N(1,1,1,1,1,1,\alpha_7,\alpha_8,\alpha_9) = \\
&\quad +\frac{1}{2}(2\ \mathbf{1^-9^-} + 2\ \mathbf{3^-9^-} - 2\ \mathbf{2^-9^-} - \mathbf{7^-9^-} - Q^2\ \mathbf{9^-}) \\
&\quad +\frac{1}{2}\frac{\alpha_9-1}{\alpha_8-1}(\mathbf{1^-8^-9^-9^-} - \mathbf{2^-8^-9^-9^-} - \mathbf{7^-8^-9^-9^-}) \\
&\quad +\frac{D-1-2\alpha_7-\alpha_8}{2(\alpha_8-1)}\mathbf{8^-9^-} + \frac{1}{\alpha_8-1}(\mathbf{1^-2^+8^-9^-} - \mathbf{2^+7^-8^-9^-}), \qquad (26) \\
&N(1,1,1,1,1,1,\alpha_7,1,\alpha_9) = \\
&\quad +\frac{1}{2}(2\ \mathbf{4^-9^-} + 2\ \mathbf{6^-9^-} - 2\ \mathbf{5^-9^-} - \mathbf{8^-9^-} - Q^2\ \mathbf{9^-}) \\
&\quad +\frac{1}{2}\frac{\alpha_9-1}{\alpha_7-1}(\mathbf{6^-7^-9^-9^-} - \mathbf{5^-7^-9^-9^-} - \mathbf{7^-8^-9^-9^-}) \\
&\quad +\frac{D-3-\alpha_7}{2(\alpha_7-1)}\mathbf{7^-9^-} + \frac{1}{\alpha_7-1}(\mathbf{5^+6^-7^-9^-} - \mathbf{5^+7^-8^-9^-}). \qquad (27)
\end{aligned}
$$

At this point either one of the denominators is missing, or there is no $p_2.p_5$ left or all denominators have a single power only. This brings us to the last recursion:

$$
\begin{aligned}
N(1,1,1,1,1,1,1,1,\alpha_9) &= \mathbf{3^-9^-} - \mathbf{8^-9^-} + Q^2\frac{1+2\epsilon-\alpha_9}{2(\alpha_9-1-4\epsilon)}\mathbf{9^-} \\
&\quad + \frac{Q^2}{\alpha_9-1-4\epsilon}(\mathbf{2^-3^+9^-} - \mathbf{3^+8^-9^-}). \qquad (28)
\end{aligned}
$$

It should be clear by now that the second scheme for the first part of the non-planar algorithm is much more cumbersome, but one can also see that in principle it is more useful: In the first scheme the combined sum of powers of propagators is mostly a constant, while in the second scheme it decreases together with the powers of α_9 (except for equations (26) and (27) where there is one term in which it is a constant, but for many integrals these two equations will not even be needed). Thus, the benefit of the second scheme consists in the fact that it does already much of the work that has to be done in the second part of the non-planar algorithm. One may also appreciate the fact that relations like eq.(24) are relatively easy to program in FORM. One can take this relation and its three variants for α_1, α_3 and α_4 and put them together into one `repeat` loop (one does not need to know in advance how many times each equation is to be applied). Similarly, eq.(25) and the corresponding recursion for α_2 together make up the body of one `repeat` loop.

At this point all the nontrivial (i.e. having no denominators missing) terms in the expression have $\alpha_9 = 0$ (so that we drop the argument α_9 in N), and we can proceed to the second part of the non-planar algorithm.

There are two types of propagators: the 'outer' propagators 1,3,4 and 6, and the 'inner' propagators 2,5,7 and 8. For each set there is a relation that improves the integral:

$$
\begin{aligned}
Q^2 N(\alpha_1,\cdots,\alpha_8) &= \mathbf{1^-} + \frac{-2D-1+2\alpha_1+2\alpha_2+\alpha_5+\alpha_6+2\alpha_7+\alpha_8}{\alpha_6-1}\mathbf{6^-} \\
&\quad + \frac{\alpha_8}{\alpha_6-1}\mathbf{6^-8^+}(\mathbf{2^-} - \mathbf{3^-}) + \frac{\alpha_5}{\alpha_6-1}\mathbf{6^-5^+}(\mathbf{7^-} - \mathbf{4^-}). \qquad (29)
\end{aligned}
$$

This recursion lowers the power of an outer propagator at least by one, so when this recursion and its three variants for and α_4 are put together into one `repeat` loop, in the end all outer propagators will have only a single power (or at least one propagator is missing). This leaves then only inner propagators with a power greater than one. The following recursion takes care of such terms:

$$
\begin{aligned}
&(\alpha_5-1)(2-D+2\alpha_5)N(1,\alpha_2,1,1,\alpha_5,1,\alpha_7,\alpha_8) = \\
&\quad(-2D+4+\beta)Q^2\mathbf{5}^-(+4-3D+2\beta+\alpha_4\mathbf{3}^-\mathbf{4}^+ + \alpha_6\mathbf{1}^-\mathbf{6}^+) \\
&\quad+(\alpha_5-1)(\alpha_4\mathbf{7}^-\mathbf{4}^+ + \alpha_6\mathbf{8}^-\mathbf{6}^+) \\
&\quad-(3-2D+\beta+\alpha_5)(\alpha_4\mathbf{5}^-\mathbf{4}^+ + \alpha_6\mathbf{5}^-\mathbf{6}^+),
\end{aligned}
\tag{30}
$$

where $\beta = \alpha_2+\alpha_5+\alpha_7+\alpha_8$.

The last recursion has some annoying properties: most terms do not change the sum of the powers of the propagators. Only the sum of the powers of the inner propagators is lowered by one, but it is then possible that one of the outer propagators gets an additional power. This means that for each of those terms we have to reapply the recursion for the outer propagators.

After all these reductions have taken place we have only integrals that have either all 8 propagators with a unit power or one (or more) of the propagators is missing resulting in an integral of the type FA or BU. The integral with the 8 single powers cannot be reduced any further. However, we only need the constant term without an ϵ for this integral, which is given by (see [18], [11]):

$$
N(1,1,1,1,1,1,1,1) = 20\zeta_5 + \mathcal{O}(\epsilon). \tag{31}
$$

The term in $O(\epsilon)$ has been computed in [21].

It should be noted that the implementation of the above algorithms requires great care. Debugging was partly facilitated by the fact that the results for some non-planar integrals were given in [1]—but not all parts of the algorithm can be tested that way. We have found it very useful to run tests involving several diagrams with a gauge parameter (whereby higher powers for the propagators are also introduced). For sums of such diagrams that correspond to a physically observable quantity the dependence on the gauge parameter should cancel in the final result, which is a rather stringent test for the program. In the last section we present a set of integrals that use all parts of the integration routines, in order to facilitate testing of future versions of the algorithm.

When very complicated diagrams have to be calculated the above recursions can become very involved. We evaluated the integral $N(2,2,2,2,2,2,2,2,2)$. This took more than 40 hours on a NeXTstation for the first method, and almost 19 hours for the second method! Integrals of this complexity can occur when higher moments of structure functions in deep inelastic scattering are computed. For the other topologies such integrals are not nearly as costly in time, because the recursions involve only the rule of the triangle in its resolved version. It is however possible to improve the speed of difficult non-planar diagrams by a spectacular amount with a partial tabulation. Unfortunately the number of different integrals is too large to tabulate all integrals that can occur during the computation of for instance a tenth moment. We can however tabulate all integrals that have $\alpha_9 = 0$ and $\alpha_1 = \alpha_3 = \alpha_4 = \alpha_6 = 1$. If we have 8 excess denominators (which means that the sum of the powers of all denominators is 8 more than the case in which all denominators have power 1), there are still 165 integrals, but symmetries reduce this to 29 different integrals. For smaller numbers of excess propagators we have fewer different integrals: 1,3,4,8,10,16,20 for the number of excess propagators being 1-7. It is a one time effort to compute all these integrals and put them in a table. For each new number of excess propagators we could use the results of 'lower' integrals. Hence the whole operation of constructing the tables up to 8 excess powers took only 6.5 hours on the same computer and after this the same integral took 1572 sec. and 407 sec. respectively. When even more complicated integrals are needed the routines will break them down to integrals that can be looked up in the table. Both routines for the non-planar integrals have been equiped with this mechanism. In the case that it is necessary to depend on this mechanism—either when the integrals are very complicated, or when the table is extended—one can speed up the evaluation by defining a preprocessor variable 'LONGINT' with the value 1. This variable causes the inclusion of some extra code that is superfluous in the case that the table is sufficient in which case this code would only slow the execution down.

4.7 The trivial topologies of the type O and Y

These topologies are "trivial" in the sense that each of them contains a one-loop subgraph that can be done immediately. The Y-topologies consist of the so-called recursively one-loop integrals, i.e. such that can be done by repeated application of the one-loop formula:

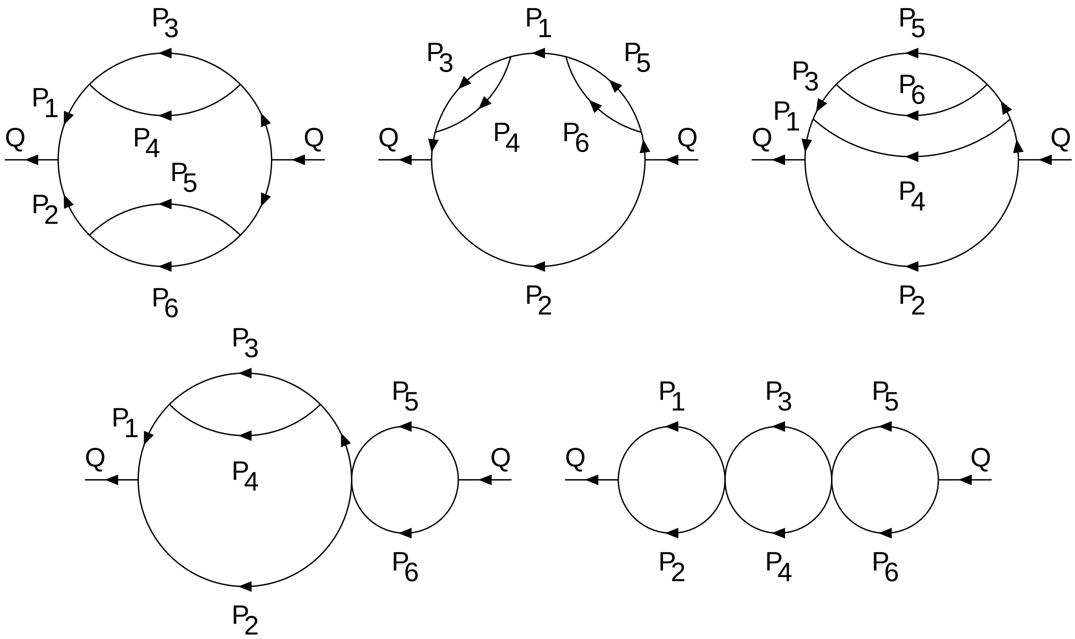

fig.12 The topologies Y1-Y5.

The topologies O1,. . . ,O3 contain a one-loop propagator-type subintegral embedded in a two-loop integral (the topology O4 can also be regarded as belonging to this class because, as explained in sect.4.1, the outer one-loop subintegral can be done prior to evaluating the two-loop insertion):

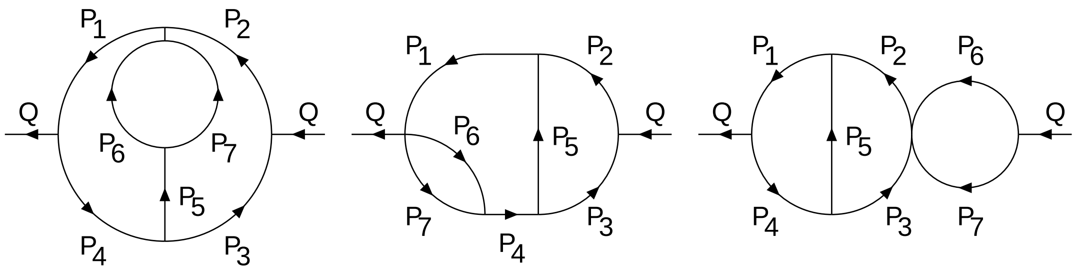

fig.13 The topologies O1, O2 and O3.

Evaluation of the two-loop integrals resulting from taking the one-loop insertions are discussed in section 3.

5 The routines

The present version of the package MINCER written for FORM consists of a fairly large number of FORM procedures (`procedure`). All variables that are needed in all procedures are declared in the header file `mdeclare.h`. `Q` always stands for the "external" momentum. The variable `ep` stands for ϵ, etc. The procedures are:

- `ACCU(`*text*`)`
  We postpone the discussion of this routine till the section on efficiency. The *text* will be displayed as commentary in the statistics when the `.sort` instruction in this procedure is executed.

- `benz(p1,p2,p3,p4,p5,p6,p7,p8,Q)`
  The routine that breaks down topologies of type BE to simpler topologies like BU. The routine reexpands scalar products in the numerator to obtain a minimal set. Then the routine `triangle` is called.

- `benzbar(p1,p2,p3,p4,p5,p6,p7,Q)`
  This routine breaks down topologies of the type FA. It calls the routine `triangle` after (temporarily) rewriting powers of $p_6.p_7$. Since the routine contains no end-of-module instructions it can be used inside the range of an `if` statement.

- `benzbu(p1,p2,p3,p4,p5,p6,p7,Q)`
  This routine breaks down topologies of the type BU. It tests whether the p_6 or the p_7 propagator has the smallest power and depending on the result of the test calls the routine `triangle` for the upper (1-4-2) or for the lower (2-5-3) triangle. The routine contains no end-of-module instructions, so it can be used inside the range of an `if` statement.

- `dotwo`
  This routine does general two-loop p-integrals. It starts by reexpressing all dotproducts in terms of powers of the denominators and Q^2. Then it applies symmetry relations to reduce the number of different integrals (and, hence, the number of terms in the expression). Then it calls the procedure `newtwo` which takes care of the two-loop topology T1. Then the two remaining one-loop integrals are done by calling the routines `one` and `simplify`.

  It should be noted that this procedure expects the input in notations different from the standard labelling of momenta for the topology T1. The input momenta should be labeled p_1, p_3, p_4, p_6 and p_7 respectively, rather than $p_1 \cdots p_5$. This is for historical reasons.

- `finish(`*loops,scheme*`)`
  This routine substitutes the various tabulated integrals to a sufficiently high power of ϵ. These integrals are `F321` for the integral in eq.(31), `G311` for the integral in (17) and `exp10`, `exp20` and `exp11` for the integrals $(n+m+1)G(1+m\epsilon,1+n\epsilon,0,0)/G(1,1,0,0)$ where we have $(m,n)=(1,0)$, $(2,0)$ and $(1,1)$ respectively. Actually there exists a relation between these integrals:
  $$(1-3\epsilon)exp_{1,0}exp_{2,0} = (1-2\epsilon)exp_{1,1}. \tag{32}$$
  This relation is used at an earlier stage (in routine `simplify`) to simplify the intermediate results. The parameters *loops* and *scheme* specify the form in which the output should be presented. When *scheme* is zero the output will be for the $\overline{\text{MS}}$ scheme. In that case we have to multiply the result by a constant that is raised to a power where the power is equal to the

number of loops in the integral. The parameter *loops* provides this number. When *scheme* is equal to one the answer will be in the so-called G scheme. Other values are currently illegal.

In addition to the above integrals this routine contains also expressions for the one- and two-loop gluon, fermion (quark) and ghost propagators in QCD. The use of these objects requires care: The one-loop fermion propagator for the momentum p_1 is:

```
g_(1,p1)/p1.p1*fermi1*e1/eq
```

where `e1` is $(p_1^2)^{-\epsilon}$ and `eq` is $(Q^2)^{-\epsilon}$. For the two-loop propagator we get these objects squared and the object `fermi2`. The gluon propagator is now transverse so it becomes:

```
(d_(i1,i2)-p1(i1)*p1(i2)/p1.p1)/p1.p1*gluon1*e1/eq
```

All these integrals are given in the G scheme, so the conversion to the $\overline{\mathrm{MS}}$ scheme presents no problems.

- `ladder(p1,p2,p3,p4,p5,p6,p7,p8,Q)`
  The ladder recursion. It results in integrals of a simpler type like FA. The routine takes care of the reexpansion of scalar products, one symmetry transformation and the application of the rule of triangle.

- `newplane(p1,p2,p3,p4,p5,p6,p7,p8,Q,ints)`
  The version of the non-planar recursion following the scheme of [11]. The momenta are as in fig. 11. The variable `ints` corresponds basically to $d^Dp_1\, d^Dp_2\, d^Dp_3$. It is needed to indicate which terms should still be integrated and some integrals are substituted immediately by a value in a table. The result is a sum of tabulated integrals and integrals of simpler types.

- `newtwo(p1,e1,p2,e2,p3,e3,p4,e4,p5,e5,Q,eq,reduction,ints)`
  This routine does the two-loop integrals of topology T1 in such a way that it can be used in higher level routines. This means that it will accept one of the propagators with a non-integer power. If the non-integer power is in one of the peripheral propagators, the procedure `triangl2` is used. This procedure differs from the regular `triangle` in that it uses the solution of the triangle recursion that takes into account the non-integer power by using Γ-functions instead of factorials wherever appropriate. As this is slightly slower, a special routine was written to handle the more complicated case. The variables e_i are defined by $e_i = (p_i^2)^{-\epsilon}$. When the inner line has the non-integer power we use symmetry and then the integral is looked up in a table (`TabTwo`). For this last substitution we need the quantity `ints` which stands for $d^Dp_1\, d^Dp_2$. When `reduction` is not zero the routine will rewrite dotproducts in terms of powers of the propagators. The quantity `eq` is defined by $e_Q = (Q^2)^{-\epsilon}$.

- `noplane(p1,p2,p3,p4,p5,p6,p7,p8,Q,ints)`
  The original MINCER version of the non-planar recursion. The momenta are as explained in fig. 11. The variable `ints` corresponds, basically, to $d^Dp_1 d^Dp_2 d^Dp_3$. It is needed to indicate which terms should still be integrated and some integrals are substituted immediately by a value in a table. The result is a sum of tabulated integrals and integrals of simpler types.

- `one(p,pq,Q,ep,epq,eq,int)`
  This is the basic one-loop integral. First it makes sure—by performing appropriate substitutions—that the only occurrence of the loop momentum p in the numerator is as `p`. The other momentum is `pq` and it is reexpanded in terms of `p`. Then the one-loop integral is evaluated. The objects e_i stand for $(p_i^2)^{-\epsilon}$. `int` stands for d^Dp. It is redundant, and we use this redundancy to make sure that no integral is done twice and no terms are left unintegrated. Further, quite a bit of attention is given to the derivatives with the d'Alembertians.

  Actually there are two versions of the procedure `one`. The procedure `onet` is to be used when very high moments of structure functions in deep inelastic scattering [22] are used. In that case even the trick with the `distrib_` function is not enough, because the same momentum P will occur many times, multiplied with `p` and in addition P^2 can be set to zero. A rather complicated construction with several instances of the `distrib_` function improves the situation dramatically. The output of the routine contains the appropriate tensor structures and the G-functions. They have to be replaced by explicit expressions in the procedure `simplify`.

- `one4(p,pq,Q,ep,epq,eq,p1,p2,p3,p4,p5,int)`
  This routine does the one-loop integration in the topology O4. Hence the momenta p_i are the five momenta of the two-loop subintegral of the topology T1. The scheme that is used is described in the section on the O4 topology. The variable `int` stands for d^Dp and $e_i = (p_i^2)^{-\epsilon}$.

- `pochtabl(n)`
  This file contains two tables. It should be included in the main declarations file `mdeclare.h`. The tables are:

$$
\begin{aligned}
PO(i,x) &= \frac{\Gamma(i+x\epsilon)}{\Gamma(1+x\epsilon)}, \quad (1 \le i \le n), \\
PO(i,x) &= \frac{\Gamma(i+x\epsilon)}{x\epsilon\Gamma(1+x\epsilon)}, \quad (-n+1 \le i \le 0), \\
POINV(i,x) &= \frac{\Gamma(1+x\epsilon)}{\Gamma(i+x\epsilon)}, \quad (-n+1 \le i \le 1), \\
POINV(i,x) &= \frac{x\epsilon\Gamma(1+x\epsilon)}{\Gamma(i+x\epsilon)}, \quad (1 < i \le n).
\end{aligned} \tag{33}
$$

  Since the values of the elements of the table differ for the cases $i \ge 1$ and $i < 1$, the definitions for these different regions are kept separate in the file. The factor $x\epsilon$ is introduced in half the cases so as to keep the pole structure from interfering with the depth of the expansion. This way each expansion starts with a constant term and they can all be cut off at the same power of ϵ. This allows us to use an automatic power cutoff for ϵ, thereby improving the speed of the program.

  The parameter `n` is used to limit the number of elements in the table. Usually `n` $= 10$ is more than enough. The maximum value of `n` is 24.

- `simplify`
  This routine replaces the G-functions by the formula in terms of normalized Γ-functions (eq.(3)). To insure that we have only normalized Γ-functions the G-functions are normalized by dividing out the function $G(1+m\epsilon, 1+n\epsilon, 0, 0)$. These functions are then written as $exp_{m,n}G(1,1,0,0)/(m+n+1)$. In the G scheme of renormalization ([18]) this last G-function is $1/\epsilon$. Conversion to $\overline{MS}$ takes place in the routine `finish` and is essentially a multiplication

by a constant. The $exp_{m,n}$ are also substituted in the routine `finish`. The rest of the routine is a rather elegant substitution of the elements of the tables with the normalized Γ's. If this were done in a blunt fashion each G-function would give (7^6) terms.

- `tabtwo(n)`
  Tabulates the integrals $I(m+\epsilon,1,1,1,1)$ that could have been reduced with (16). The table will include at least the values $-n+2 \leq m \leq n$. The maximum value for `n` is 12. If more entries are needed one can use (16) till the table can be used. The new elements can be added to the table. The subintegrals that are still in the table were not substituted because the expressions would become longer due to the presence of the various ζ_i.

- `triangl2(p,pa,pb,p1,p3)`
  This routine is almost identical to the routine `triangle`. The difference is in the possibility to use a non-integer power for either p_a^2 or p_b^2 (or both). In the routine `triangle` these powers enter a factorial function (which is faster) while in `triangl2` they are handled in a Γ-function.

- `triangle(p,pa,pb,p1,p2)`
  This is the implementation of the rule of the triangle. If p_1 and p_2 are outgoing we define $p_a = p - p_1$ and $p_b = p + p_2$, so p_a and p_b are the "other" momenta in the triangle. Actually the direction of the external momenta is not important. For very simple cases the recursion relation (11) is used. For more complicated cases or when there is a chance to pick up a spurious pole the closed formula (14) is used. This involves some Γ-functions which are substituted from the tables with normalized Γ's.

- `two(p1,e1,p2,e2,p3,e3,p4,e4,p5,e5,Q,eq,reduction,ints)`
  This routine is like the routine `newtwo`, except for that it does not use the table in `TabTwo`. Instead it uses the recursion in equation 16. This routine is only needed when the table in `TabTwo` has to be extended.

In addition to these routines there is the file 'notabl.h' which contains a partial tabulation of non-planar integrals. This table contains only different integrals. The routines newplane and noplane use the symmetries of the non-planar diagrams to reduce the integrals to the integrals that can be found in the table. This file in included in 'mdeclare.h', so the user does not need it directly.

The `include`-files each can handle complete integrals after all vertices and propagators have been substituted and all traces have been taken. Each file expects an expression in terms of the vectors p_i as defined in the figures presented above, while the external momentum is Q. In addition the dimension D (or n) has to be in terms of $\epsilon = 2-D/2$, which is represented by `ep`. All momentum dependences should be in terms of dotproducts between the given vectors. Each file has a name of 5 characters, the first 3 of which are always `inp`, while the other two indicate the topology. The file contains the statements that will result in the integrals being evaluated to enough powers in ϵ that the results can be used for three-loop calculations. This means that for the three-loop integrals the constant term is kept but the term $\mathcal{O}(\epsilon)$ eliminated. For the two-loop integrals the terms $\mathcal{O}(\epsilon)$ are kept while the terms $\mathcal{O}(\epsilon^{\in})$ eliminated. Similarly the one-loop results are extended to ϵ^2 and the tree level results go to ϵ^3.

- `inpno`: The non-planar integration. First it uses the procedure `noplane` unless the preprocessor variable NEWPLANE has been set in which case the routine `newplane` is used. Then symmetry relations are used to reduce the number of terms. After this the reduction of the BU and FA topologies is prepared by eliminating the momenta that do not appear in those topologies. The procedure `triangle` is called for the reduction. Again, symmetry relations are used to minimize the number of terms. Then the first integral can be evaluated, which is done either by a call to the procedure `one` or by a call to the procedure `one4`. The resulting G-functions are removed in a call to the procedure `simplify`. Now we have a standard two-loop integral which can be handled by the procedure `dotwo`. Finally, substitutions of the basic integrals are done as well as conversion to the chosen renormalization scheme. This is done in the procedure `finish`.

- `inpla`: The ladder integration. This routine is rather similar in structure to `inpno`. It uses the procedures `ladder`, `benzbar`, `one`, `simplify`, `dotwo` and `finish`.
- `inpbe`: The Benz integration. In essence the setup is as in `inpno` and `inpla` but there is less symmetry. It uses the procedures `benz`, `benzbu`, `benzbar`, `one`, `one4`, `simplify`, `dotwo` and `finish`.
- `inpbu`: The BU integration. After a relabelling it invokes `inpbe`.
- `inpfa`: The FA integration. It just invokes `inpbe`.
- `inpo1`: The integration of type O1. All simpler topologies are worked out in dedicated programs, rather than considered as special cases of the more general toplogies as was done in the original MINCER program. Hence the one-loop subintegral is done directly with a call to `one` and it is simplified with a call to `simplify`. Then the resulting two-loop topology is taken care of by the procedure `dotwo` and a call to `finish` completes the job.
- `inpo2`: The integration of type O2. It is very similar to `inpo1`.
- `inpo3`: The integration of type O3. It is very similar to `inpo1` and `inpo2`.
- `inpo4`: The integration of type O4. The strategy is rather similar to the one used for the other O topologies. Here we have to call the procedure `one4`, but before we can do this we have to reexpand the dotproducts in terms of a set that is more suitable for the harmonic projections that are used in this procedure. After this everything is straightforward.
- `inpy1`: The integration of type Y1. The Y topologies are even easier than the O topologies. They are pure combinations of one-loop integrals and no reduction tricks are needed. Therefore, only a minimal amount of reexpanding scalar products is needed before each call to the procedure `one`.
- `inpy2`: The integration of type Y2. Very similar to `inpy1`.
- `inpy3`: The integration of type Y3. Very similar to `inpy1`.
- `inpy4`: The integration of type Y4. Very similar to `inpy1`.
- `inpy5`: The integration of type Y5. Very similar to `inpy1`.
- `inpt1`: The integration of type T1. Because of the strange momentum assignments in the procedure `dotwo` it has to relabel the momenta. In addition, it multiplies the expression by $1/\epsilon$ to insure that the answer will be accurate to one power in ϵ more than in the three-loop integrals. This may be done because the fixed depth in the expansions is actually overkill for two-loop diagrams. In the end the result is multiplied by ϵ. The actual integration is done in the procedure `dotwo`.

- inpt2: The integration of type T2. The trick with the ϵ is the same as in inpt1. After that the integral is worked out in the same way as with the Y topologies. All calls to one and simplify are done in this file.

- inpt3: The integration of type T3. Similar to inpt2.

- inpl1: The integration of type L1. The ϵ trick of inpt1 is now used with two powers of ϵ to insure that the answer is correct to order ϵ^2. Of course, the procedures one and simplify are called directly from this file.

- inptr: The tree level routine. It uses only the routine finish. We have to divide the expression by ϵ^3 at the beginning and multiply by the same factor in the end to ensure that the result is accurate to order ϵ^3.

In all programs the result would in principle contain a power of the quantity eq $= (Q^2)^{-\epsilon}$. This quantity is divided out and all results are normalized so that they are dimensionless. Hence the integrals give only ζ_3, ζ_4 and ζ_5 and powers of ϵ.

An interesting observation is that in the 5 physical observables that have been calculated so far using MINCER, the ζ_4 dropped out in the final results. We can offer no explanation to such miraculous cancellations.

6 Efficiency

In this section we will discuss the reasons for some decisions in the implementation. Most of these decisions were taken because it is important for MINCER to be as fast as possible, as the actual calculations tend to require computer resources measured in hundreds of hours of CPU time and many Mbytes of disk storage.

In first tests most of the computer time was spent in the recursions and the simplification stages. One of the reasons for this was that each term takes much attention during simplification. On the other hand, if we study such expressions we notice that many terms are identical except for different powers of ϵ. Indeed, calculations of this kind can be naturally regarded as arithmetic with (truncated) power series. Hence it would be very economical to collect polynomials in ϵ into the argument of a single function. If that function can be treated as a polynomial coefficient all the work on the Γ-functions etc. has to be done only once, rather than about seven times. To implement this, we work with a function named acc (for accumulator) (in the section on tabulated Γ's it was called A) that contains polynomials in ϵ in its argument. FORM will allow one function to be treated as a polynomial coefficient when the sorting takes place at the end of each module, so that just a few lines of code are needed in order to absorb various powers of ϵ into acc. We have done this in the procedure ACCU:

```
#procedure ACCU(TEXT)
if ( count(ep,1) );
    if ( count(acc,1) == 1 );
        id  ep^x?*acc(y?) = acc(ep^x*y);
    else;
        id  ep^x? = acc(ep^x);
    endif;
endif;
.sort(PolyFun = acc):'TEXT';
#endprocedure
```

We see that when there is only a single occurrence of the function acc additional powers of ϵ that are outside this polynomial coefficient are pulled inside. Then the PolyFun option of the

`.sort` instruction causes two things to happen. First the coefficient is pulled inside the function if there is just a single occurrence of it in the term. If there is none, a function `acc` is created with the coefficient as its argument. If there is more than one occurence of the function `acc` in the term or when a single function `acc` has more than one argument, nothing special is done with this term. Then, during the sorting, the arguments of these single functions `acc` are added if two terms are otherwise identical. This is just like adding coefficients and is exactly what we need here. The `TEXT` is placed as commentary with the statistics that are printed each time a sort buffer is full. This allows one to trace the progress of long programs.

Another problem is connected with higher derivatives in the procedure `onet` (see the description of the procedure `one`). Evaluation of higher moments of the structure functions for deep inelastic scattering involves multiple powers of a momentum that is "foreign" to the graph in the sense that it is independent of all the momenta flowing through the lines of the graph. This means that dot-products involving this vector cannot be reexpanded in terms of powers of the propagators. Hence all these dotproducts survive the reexpansion and for instance the 10-th moment of a longitudinal crosssection can involve 12 powers of $P.p$ with p being the loop momentum and P, the "foreign" momentum. If we use the regular trick with the `distrib_` function we would still generate enormous numbers of terms (for 6 powers of the d'Alembertian there would be $11!! = 10395$ terms), all of them proportional to P^2. If we take into account that actually $P^2 = 0$ in such an analysis, we see an even greater waste. A better solution works as follows:

All other vectors that are multiplied by p (or indices) are collected inside one function while the vectors P are collected separately. If the function with the P's has more arguments than the other function there will be restrictions on the number of d'Alembertians that give a non-zero result. The P's will have to be contracted either with the arguments of the other function or with the external vector Q. By avoiding the various permutations of the P's generating separate terms we save a factor $n!$ in work for n vectors P in the case when there are no P^2. This turned out to make the calculation of even twelfth moments possible. Of course all these considerations concern only two-loop integrals, because the computation of moments for three-loop integrals cannot proceed with a "foreign" momentum (because some more relations would be needed than described above and this would be very difficult for the non-planar topology). In the three-loop case one would have to project the moments out, and a harmonic projection for higher moments is exceptionally costly. Hence the highest moment in three-loops that has been calculated thus far is the second moment [17], although it may be feasible to compute the fourth, sixth and maybe even the eighth moment in the near future. The last two moments would need parallel processing on a supercomputer.

Making use of the function `acc`, the explicit solution for the triangle recursion and the tables, it proved possible to achieve that the computer time is now more or less evenly divided between the integration stages and the reexpansions of dotproducts at the initial stages of the program. The reepansion of dotproducts needs by far the largest amount of temporary diskspace, especially when gauge parameters are used in vector propagators. We have had a diagram for which FORM needed about 200 Mbytes of diskspace for such a reexpansion. Therefore, one needs to pay special attention to these otherwise trivial operations.

There are two places that allow an easy improvement. The first one concerns triple gluon vertices. When a triple gluon vertex is contracted with the momentum that belongs to the contracted index there are simplifications:

$$
\begin{aligned}
V^{ggg}_{\mu\nu\rho} p_3^\rho &= \delta_{\mu\nu}(p_2^2 - p_1^2) + p_{1_\mu} p_{1_\nu} - p_{2_\mu} p_{2_\nu}, \\
V^{ggg}_{\mu\nu\rho} p_2^\nu p_3^\rho &= -p_1^2 p_{2_\mu} + p_{1_\mu} p_1 \cdot p_2, \\
V^{ggg}_{\mu\nu\rho} p_1^\mu p_2^\nu p_3^\rho &= 0.
\end{aligned}
\tag{34}
$$

Here we have rewritten most of the dotproducts in such a way that they give squares of momenta, which is exactly what we need eventually. This saves on reexpansions later. When gauge parameters are used there are many of the above contractions and savings can be enormous.

The second improvement concerns gauge terms and fermion lines. Consider for instance the following subgraph:

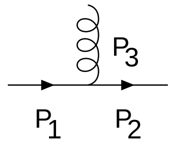

fig.14 A simple subgraph.

The gauge term in the gluon propagator will give a term with the momentum p_3 and the trace will contain the substring

$$\begin{aligned} \not{p}_2\not{p}_3\not{p}_1 &= \not{p}_2\not{p}_2\not{p}_1 - \not{p}_2\not{p}_1\not{p}_1, \\ &= p_2^2\not{p}_1 - p_1^2\not{p}_2. \end{aligned} \tag{35}$$

In this case the placement of p_3 makes the substitution very obvious and even though there are now twice as many strings of gamma matrices each trace will be simpler and we have separated out one power of a propagator automatically. This makes the resulting expression much shorter and much time (and diskspace) is saved during the rewriting of the dotproducts. There is however one complication: the momentum assignments for each topology are different and hence the application of these tricks should be done separately for each topology. This can be achieved with a special file that contains all the various replacements, each in a fold whose name contains the name of the topology. In that case the same mechanism that will allow us to select the various files `inp`*xx* will allow us to select a single fold from this file. Folding is originally a mechanism used for outlining by some editors. FORM uses the folding mechanism of the STedi editor to put only selected parts of a header file in its input stream. The file with these folds is called `reduce.h`.

After all the above improvements have been introduced, the program is rather well balanced in the sense that there are no obvious bottlenecks left. In addition, the needed diskspace went down to less than 39 Mbytes for the diagram that first needed about 200 Mbytes.

7 Some examples

In this section we will first look at some individual diagrams. Then we will look at how a complete calculation can be set up.

The first example is rather trivial. We compute a simple ladder diagram in which the rungs of the ladder are photons and we contract the indices of the external photons. The program is rather short:

```
#include mdeclare.h
.global
#define SCHEME "0"
Local Dia =
    g_(1,MU,p1,mu,p2,nu,p3,MU,p4,nu,p5,mu,p6)
    /p1.p1/p2.p2/p3.p3/p4.p4/p5.p5/p6.p6/p7.p7/p8.p8;
Tracen,1;
id  [n-4] = -2*ep;
id  n = 4 - 2*ep;
```

```
#call ACCU{traces}
#include inpla
multiply 1/Q.Q;
print;
.end
```

We start with specifying all declarations by including the file `mdeclare.h` and then we make all declarations `global`. This last step is not necessary in this example but for more complicated programs it would be. Next we define the renormalization scheme in which we would like to see the output. A choice of zero means that the output will be in the $\overline{MS}$ scheme. The next step is to define the diagram in terms of the variables of fig.7. Here we work in the Feynman gauge in which the photon propagator with momentum p is just $\delta^{\mu\nu}/p^2$. At this point we should take the trace and make the input ready for the file `inpla`. The trace is in n dimensions and we convert to ϵ. It is of course best to simplify the expression before we start the real work. Hence the call to `ACCU` which collects the powers of ϵ inside the function `acc` and treats this as a polynomial coefficient during the sorting that is executed. In the end we divide by Q^2 to make the result dimensionless. In principle there should also be powers of $(Q^2)^{-\epsilon}$ but those are systematically removed by the program. The result of this program is

```
Dia =
   2583/2 + 8/3*ep^-3 + 82/3*ep^-2 - 96*ep^-1*z3
   + 207*ep^-1 - 1168/3*z3 - 144*z4 - 320*z5;
```

and the execution time is 11.5 sec on a NeXTstation (68040 chip at 25 Mhz). Calculations in a more general gauge are also easy, although they will usually require more computer time. If we replace the definition of the diagram in the above example by

```
Local Dia =
    g_(1,MU,p1,mu,p2,nu,p3,MU,p4,nu1,p5,mu1,p6)
    /p1.p1/p2.p2/p3.p3/p4.p4/p5.p5/p6.p6
    *Dg(mu,mu1,p7)*Dg(nu,nu1,p8);
id  Dg(i1?,i2?,p?)=(d_(i1,i2)-xi*p(i1)*p(i2)/p.p)/p.p;
#include reduce.h # reducela
```

then it should be easy to guess that the extra powers of the momenta in the denominator will lead to more complicated expressions after the rule of triangle has been applied. The result is now

```
Dia =
    + 2583/2 + 8/3*ep^-3 + 82/3*ep^-2 - 96*ep^-1*z3
    + 207*ep^-1 - 1168/3*z3 - 144*z4 - 320*z5
    + xi * (  - 1504 - 16/3*ep^-3 - 128/3*ep^-2
       + 96*ep^-1*z3 - 276*ep^-1 + 1808/3*z3 + 144*z4 )
    + xi^2 * ( 2179/6 + 8/3*ep^-3 + 46/3*ep^-2
       + 77*ep^-1 - 256/3*z3 );
```

and this took 21.5 sec on the same computer. The use of the reductions in the file `reduce.h` saved about 3 sec.

Any calculation of such complexity requires additional checks to make sure that no simple mistake (like a typo in the input expression) has been introduced. Cancellation of divergences after renormalization is one such check. Another is the introduction of the above gauge parameter. (Depending on the graph, such a check can require far more computer time, but it invariably turns out that calculations without adequate built-in checks contain errors and recalculations done after publication of wrong results tend to be extremely costly.) If the answer for a physical quantity contains the gauge parameter explicitly, its presence makes it much easier to locate the problem. But even such tests are not sufficient, because the most difficult integrals seem to occur in the terms without ξ. In an earlier version of the program we have run across an error that occurred only in the gauge independent finite part of some diagrams.

Next we present some results that can be used as a test whether the programs perform properly. First we take some non-planar integrals.

$$\begin{aligned} N(2,2,2,2,2,2,2,2,2) &= -\frac{368808805411}{81000} - \frac{1348}{3}\epsilon^{-3} - \frac{42409}{9}\epsilon^{-2} \\ &\quad -\frac{13820447}{900}\epsilon^{-1} - \frac{36312616}{3}\zeta_3 + 18432000\zeta_5 \end{aligned} \tag{36}$$

This integral tests mainly the table in the file `notabl.h`, unless one plays around with the value of the variable INNOTABL. Note however that when its value is made smaller the time that is needed increases rapidly. The following integral is a good test when the value of INNOTABL is set to 4. In that case only one integral is taken from the tables. It is a good test of the later stages of the recursion.

$$\begin{aligned} N(1,2,1,1,2,1,2,2,0) &= -\frac{130909}{27} - 64\epsilon^{-3} - \frac{1652}{3}\epsilon^{-2} \\ &\quad -\frac{11978}{9}\epsilon^{-1} - 9120\zeta_3 + 17280\zeta_5 \end{aligned} \tag{37}$$

The earlier parts of the non-planar integrals can be tested with the integral

$$N(2,1,2,2,1,2,1,1,0) = -\frac{1460}{3} - 4\epsilon^{-3} - 22\epsilon^{-2} + 20\epsilon^{-1} - 940\zeta_3 + 1920\zeta_5 \tag{38}$$

When a more complicated input is required it is usually a good idea to design an extra procedure. We will call it generically procedure `treat`. It should handle the transformation of the input (often in an abstract notation) to the notation in which the formula is ready for processing by one of the `inp`xx files. The input could for instance be in terms of vertices and propagators. `treat` should then substitute those propagators and vertices in such a way that the expression is still manageable. After that it may have to deal with color matrices and evaluate traces. The dimension (called `n` in the program) is then rewritten in terms of the variable `ep`. In the case of the computation of moments of structure functions in deep inelastic scatterring the propagators have to be expanded in terms of the expansion momentum P, and then the proper number of powers of this momentum has to be selected. In the case of two-loop integrals one can leave it at that, but in the case of three-loop integrals one would have to make a harmonic projection.

In the next example we will show how to compute the quark propagator to three loops within the framework of QCD[8]. The diagrams are put into a single file. For each diagram that we compute, this file is included and only the needed diagram is taken, one at a time. This is done by selecting individual folds. The diagrams are built up from vertices, propagators and overall factors. The routine `treat` provides explicit expressions for these objects. Then we multiply the total expression by an overall factor $\frac{1}{4}\not{Q}/Q^2$ and take the trace. This allows one to projects out the coefficient of $\not{Q}$ from the answer.

The diagrams can be found in the file `fermidia`. A typical diagram looks like

```
*--#[ dia2 :
*
        +S(i5,p1,i6,p2,j3,p7,j6,-p8,j5,-p4,i3)
        *Dg(i3,j3,p3)
```

[8]It could be possible to make a FORM program to generate the necessary diagrams, but that is outside the scope of the present paper.

```
    *Dg(i5,j5,p5)
    *Dg(i6,j6,p6)
    ;
#define TOPOLOGY "be"
*
*--#] dia2 :
*--#[ fac2 :
    (2*cf*(cf-ca/2)^2)
*--#] fac2 :
```

The lines with dia2 and fac2 mark the ranges of the folds. The function S represents a string of gamma matrices and it is essential that the indices of the vertices alternate with the momenta of the propagators. This way the routine treat will replace the momenta by the appropriate Γ-function divided by the square of the momentum. Dg is the gluon propagator. Then the topology of the diagram is defined. The content of the second fold is the overall factor, which is a combination of color factors, combinatorics and statistics. This factor is used at a later stage when all diagrams are added. The basic treat routine looks as follows:

```
#procedure treat
repeat;
 id  SS(i1?index_,??) = g_(2,i1)*SS(..);
 id  SS(p1?,??) = g_(2,p1)*Ds(p1)*SS(..);
endrepeat;
id   SS = 1;
repeat;
 id  S(i1?index_,??) = g_(1,i1)*S(..);
 id  S(p1?,??) = g_(1,p1)*Ds(p1)*S(..);
endrepeat;
id   S = g_(1,Q)/4/Q.Q;
id   Ds(p1?) = 1/p1.p1;
id   Vgh(i1?,p1?,p2?,p3?) = p2(i1);
id   Dg(i1?,i2?,p?) = d_(i1,i2)/p.p-xi(p(i1)*p(i2)/p.p^2;
id   DL(i1?,j1?,p1?) = d_(i1,j1)/p1.p1-p1(i1)*p1(j1)/p1.p1^2;
id   V3g(i1?,p1?,i2?,p2?,i3?,p3?) =
                    p1(i3)*d_(i1,i2)-p2(i3)*d_(i1,i2)
                   +p2(i1)*d_(i2,i3)-p3(i1)*d_(i2,i3)
                   +p3(i2)*d_(i3,i1)-p1(i2)*d_(i3,i1);
id   V4g(i1?,i2?,i3?,i4?,x1?,x2?,x3?) =
    +x1*(d_(i1,i3)*d_(i2,i4)-d_(i1,i4)*d_(i2,i3))
    +x2*(d_(i1,i2)*d_(i3,i4)-d_(i1,i4)*d_(i2,i3))
    +x3*(d_(i1,i3)*d_(i2,i4)-d_(i1,i2)*d_(i3,i4));
id  n = 4-2*ep;
.sort: expansions;
tracen,2;
tracen,1;
id   [n-4] = -2*ep;
id   n = 2-2*ep;
.sort:traces;
#endprocedure
```

For the sake of simplicity, we have not used in this setup some of the improvements that have been explained in the previous section. The actual file does contain those improvements. Note that Vgh is the ghost vertex and V4g is the 4-gluon vertex. For this vertex we note that the color factors will be identical (up to a constant factor) for each of the three parts. Hence we have included the parameters x_1 to x_3 to keep the diagram itself relatively simple. When more than one 4-gluon vertex is involved it is necessary to become slightly more sophisticated about it.

The final steps are the construction of the main program that adds the results, and of the `make`-file. The main program 'dofermi' is now rather simple:

```
#include mdeclare.h
I   i;
#define SCHEME "0"
#define GAUGE "1"
.global
G   'DIA' =
#include fermidia # 'DIA'
#call treat
#include inp'TOPOLOGY'
print;
.store
save f'DIA'.res;
.end
```

The variable `DIA` will be specified in the call of FORM. Note that inclusion of `fermidia` is done in such a way that only the proper fold is selected. Then the procedure `treat` does its job and the whole is finished with a call to the proper integration program. In the end the result is stored in a file. We can pick those results up in a later program and put in the color factors there. Assume that this program is called `sumfermi` and its result is stored in the file `sumfermi.sav`. If the user works under UNIX a possible `make`-file might look as follows:

```
DIAS = fdia1.res fdia2.res fdia3.res fdia4.res fdia5.res  \
       fdia6.res fdia7.res fdia8.res fdia9.res fdia10.res \
       fdia11.res fdia12.res fdia13.res fdia14.res        \
       fdia16.res fdia17.res fdia18.res fdia19.res        \
       fdia20.res fdia21.res fdia22.res fdia23.res
       fdia24.res

sumfermi.sav:   $(DIAS) sumfermi
   form -l sumfermi

fdia1.res:   dofermi
   form -d DIA=dia1 dofermi > dia1.log

fdia2.res:   dofermi
   form -d DIA=dia2 dofermi > dia2.log

 .....etc......
```

In this case the only further command that has to be given is `make` and all diagrams are computed (after some waiting, of course). The advantage of doing it this way is that, on the one hand, one does not have to submit each diagram separately (which in the actual case of 347 diagrams in the calculations in deep inelastic scatterring of [15], [16] is definitely a bonus)—on the other hand, if something goes wrong with the computer we loose only (part of) one diagram. Such a precaution is by no means excessive, because some calculations may take days, even on very fast computers.

As a test of the programs we give the results here for the three-loop quark propagator. This result was previously used as a part of the computations in [16]:

$$+C_F C_A n_f \quad \Big(+\tfrac{260075}{1296} - \tfrac{1}{6}\epsilon^{-3}\xi + \tfrac{1}{6}\epsilon^{-3} - \tfrac{13}{12}\epsilon^{-2}\xi + \tfrac{227}{36}\epsilon^{-2} + 2\epsilon^{-1}\zeta_3\xi - 6\epsilon^{-1}\zeta_3$$

$$
\begin{aligned}
& -\tfrac{155}{24}\epsilon^{-1}\xi+\tfrac{8267}{216}\epsilon^{-1}+\tfrac{61}{6}\zeta_3\xi-\tfrac{137}{6}\zeta_3+3\zeta_4\xi-9\zeta_4-\tfrac{4483}{144}\xi\Big)\\
+C_FC_A^2 \quad & \Big(-\tfrac{2008997}{2592}+\tfrac{55}{24}\epsilon^{-3}\xi-\tfrac{5}{8}\epsilon^{-3}\xi^2+\tfrac{1}{12}\epsilon^{-3}\xi^3-\tfrac{7}{4}\epsilon^{-3}+\tfrac{623}{48}\epsilon^{-2}\xi\\
& -\tfrac{49}{16}\epsilon^{-2}\xi^2+\tfrac{3}{8}\epsilon^{-2}\xi^3-\tfrac{1841}{72}\epsilon^{-2}-18\epsilon^{-1}\zeta_3\xi+\tfrac{11}{8}\epsilon^{-1}\zeta_3\xi^2+\tfrac{83}{2}\epsilon^{-1}\zeta_3\\
& +\tfrac{6869}{96}\epsilon^{-1}\xi-\tfrac{497}{32}\epsilon^{-1}\xi^2+\tfrac{89}{48}\epsilon^{-1}\xi^3-\tfrac{66629}{432}\epsilon^{-1}-\tfrac{2917}{24}\zeta_3\xi+23\zeta_3\xi^2\\
& -\tfrac{29}{12}\zeta_3\xi^3+\tfrac{1015}{4}\zeta_3-27\zeta_4\xi+\tfrac{33}{16}\zeta_4\xi^2+\tfrac{249}{4}\zeta_4+5\zeta_5\xi\\
& -\tfrac{5}{4}\zeta_5\xi^2-45\zeta_5+\tfrac{189061}{576}\xi-\tfrac{4403}{64}\xi^2+\tfrac{49}{6}\xi^3\Big)\\
+C_Fn_f^2 \quad & \Big(-\tfrac{1108}{81}-\tfrac{4}{9}\epsilon^{-2}-\tfrac{68}{27}\epsilon^{-1}\Big)\\
+C_F^2C_A \quad & \Big(-\tfrac{1199}{16}+\tfrac{9}{4}\epsilon^{-3}\xi-\tfrac{3}{2}\epsilon^{-3}\xi^2+\tfrac{1}{4}\epsilon^{-3}\xi^3-\epsilon^{-3}+\tfrac{49}{4}\epsilon^{-2}\xi\\
& -\tfrac{47}{8}\epsilon^{-2}\xi^2+\tfrac{7}{8}\epsilon^{-2}\xi^3-\tfrac{43}{12}\epsilon^{-2}-9\epsilon^{-1}\zeta_3\xi+3\epsilon^{-1}\zeta_3\xi^2+2\epsilon^{-1}\zeta_3\\
& +\tfrac{365}{8}\epsilon^{-1}\xi-\tfrac{81}{4}\epsilon^{-1}\xi^2+\tfrac{11}{4}\epsilon^{-1}\xi^3-\tfrac{93}{8}\epsilon^{-1}-\tfrac{101}{4}\zeta_3\xi+\tfrac{3}{2}\zeta_3\xi^2\\
& +\tfrac{3}{4}\zeta_3\xi^3+23\zeta_3-\tfrac{27}{2}\zeta_4\xi+\tfrac{9}{2}\zeta_4\xi^2+3\zeta_4+20\zeta_5\xi\\
& +\tfrac{2147}{16}\xi-\tfrac{217}{4}\xi^2+\tfrac{27}{4}\xi^3\Big)\\
+C_F^2n_f \quad & \Big(+\tfrac{163}{8}-\tfrac{1}{2}\epsilon^{-2}\xi-\tfrac{1}{6}\epsilon^{-2}-\tfrac{9}{4}\epsilon^{-1}\xi+\tfrac{11}{12}\epsilon^{-1}-16\zeta_3-\tfrac{67}{8}\xi\Big)\\
+C_F^3 \quad & \Big(+\tfrac{79}{48}+\tfrac{1}{2}\epsilon^{-3}\xi-\tfrac{1}{2}\epsilon^{-3}\xi^2+\tfrac{1}{6}\epsilon^{-3}\xi^3-\tfrac{1}{6}\epsilon^{-3}+\tfrac{3}{4}\epsilon^{-2}\xi\\
& -\tfrac{3}{2}\epsilon^{-2}\xi^2+\tfrac{1}{2}\epsilon^{-2}\xi^3+\tfrac{1}{4}\epsilon^{-2}+\tfrac{13}{8}\epsilon^{-1}\xi-3\epsilon^{-1}\xi^2+\epsilon^{-1}\xi^3\\
& -\tfrac{1}{8}\epsilon^{-1}-\tfrac{5}{2}\zeta_3\xi-\tfrac{7}{2}\zeta_3\xi^2+\tfrac{7}{6}\zeta_3\xi^3+\tfrac{29}{6}\zeta_3+\tfrac{103}{16}\xi-3\xi^2+\xi^3\Big).
\end{aligned}
\tag{39}
$$

The total execution time for this computation was less than 35 minutes on a NeXTstation with the routine `treat` as given in the text, and slightly more than 27 minutes with the routine included in the package.

References

[1] S.G. Gorishny, S.A. Larin and F.V. Tkachov, INR preprint P-0330 (Moscow, 1984); S.G. Gorishny, S.A. Larin, L.R. Surguladze and F.V. Tkachov,Comp. Phys. Comm. 55 (1989) 381.

[2] J.A.M. Vermaseren, Symbolic Manipulation with FORM, published by CAN (Computer Algebra Nederland), Kruislaan 413, 1098 SJ Amsterdam, 1991, ISBN 90-74116-01-9. An earlier version of FORM can be obtained for free by means of anonymous ftp from nikhefh.nikhef.nl. Version 1 is however insufficient to run the program described in this publication.

[3] G.'t Hooft and M. Veltman, Nucl. Phys. B153 (1979) 365

[4] G.J. van Oldenborgh and J.A.M. Vermaseren, Zeitschr. für Physik C46 (1990) 425.

[5] E.E. Boos and A.I. Davydychev, INP MSU preprint 90-11/157 (Moscow, 1990); A.I. Davydychev, J.Math.Phys. 32 (1991) 1052.

[6] See for instance S.Laporta and E.Remiddi, Bologna preprint DFUB 91-3 and references therein.

[7] S.G.Gorishny, S.A.Larin and F.V.Tkachov, Phys. Lett. 101A (1984) 120.

[8] J.C.Collins, "Renormalization", Cambridge Univ. Press, Cambridge, 1984.

[9] S.G.Gorishny, S.A.Larin and F.V.Tkachov, Phys. Lett. 124B (1983) 217.

[10] F.V.Tkachov, Phys. Lett. 100B (1981).

[11] K.G. Chetyrkin and F.V. Tkachov, Nucl. Phys. B192 (1981) 159.

[12] F.V. Tkachov, Teor. Mat. Fiz. 56 (1983) 350.

[13] M.Veltman, SCHOONSCHIP, CERN preprint, 1967; H.Strubbe, Comp. Phys. Commun. 8 (1974) 1.

[14] S.G.Gorishny, A.L.Kataev and S.A.Larin, Phys. Lett. B212 (1988) 238, ibid. B259 (1991) 144; L.R.Surguladze and M.A.Samuel, Phys.Rev.Lett. 66 (1991) 560, erratum ibid, 2416.

[15] S.A. Larin, F.V. Tkachov and J.A.M. Vermaseren, Phys. Rev. Lett. 66 (1991) 862.

[16] S.A. Larin and J.A.M. Vermaseren, Phys. Lett. B259 (1991) 345.

[17] S.A. Larin, F.V. Tkachov and J.A.M. Vermaseren, preprint NIKHEF-H/91-19.

[18] K.G. Chetyrkin, A.L. Kataev and F.V. Tkachov, Nucl. Phys. B174 (1980) 345.

[19] W.A.Bardeen, A.J.Buras, D.Duke, T.Muta, Phys. Rev. D18 (1978) 3998.

[20] K.G.Chetyrkin, S.G.Gorishny, S.A.Larin and F.V.Tkachov, INR preprint P-0453, (Moscow, 1986).

[21] D.I.Kazakov, Phys.Lett. B 133 (1983) 406; Teor.Mat.Fiz. 58 (1984) 343.

[22] S.A. Larin and J.A.M. Vermaseren, preprint NIKHEF-H/91-17.