*****DRAFT*****

Guide to FeynCalc 4.2

Rolf Mertig
and
Frederik Orellana

CERN
March 1st, 2002

Contents

1 Introduction

FeynCalc is a (≥ 3) collection of utilities for algebraic calculations in high energy physics. It is implemented as a Mathematica package; that is, a number of extensions to the programming language Mathematica, themselves written in Mathematica. In the section below are described some of the main new capabilities provided.

This document is the technical manual of FeynCalc; explanations are sometimes rather short. The user is assumed to know Quantum Field Theory and Mathematica. It is strongly recommended to study the Mathematica book [1] by Stephen Wolfram before starting with FeynCalc.

The original idea of FeynCalc was to provide convenient tools for radiative corrections in the Standard Model of particle physics. The input for FeynCalc, the analytical expressions for the diagrams, can be entered by hand or can be taken directly from the output of another package, FeynArts [2] . The user can provide certain additional information about the process under consideration, i.e. the kinematics and the choice of the standard matrix elements may be defined. Once this is done, FeynCalc performs the algebraic calculations like tensor algebra, tensor integral decomposition and reduction, yielding a polynomial in standard matrix elements, special functions, kinematical variables and scalars.

FeynCalc also provides calculator-type like features. You enter a Dirac trace in a very similar notation you use by hand and get back an answer, suitable for further manipulation. These features include basic operations like contraction of tensors, simplification of products of Dirac matrices and trace calculation.

More complex algebraic operations have also been implemented, notably derivation of polynomials in quantum fields with respect to such fields.

Over the years, the detailed course of and motivation for the development of FeynCalc has changed somewhat both in scope and focus. This has several reasons: 1) the different research interests of the two authors; 2) the relative slowness of Mathematica and the rapid development of multi-loop calculations carried out with tools more suited for very large scale calculations; 3) the rapid development of the raw computing power of hardware. The first two points have had as consequence that the focus has moved away from providing cutting edge results of loop calculations in the Standard Model and in QCD. The last point has had as consequence that it has been possible to explore other types of calculations, namely complex algebraic manipulations of quantum fields and lagrangians, traditionally done by hand only.

Despite the motivations driving development, the central vision remains largely the same: FeynCalc was conceived as and continues to be developed as a general-purpose modular package for quantum field theory calculations. Dirac algebra utilities, functional derivation utilities, integral tables etc. have been clearly separated off in distinct functions which do well-defined operations. Such basic functions can then be used on a higher level to build sophisticated model dependent functions.

In the following we list what we consider the most important missions of the FeynCalc project:

- **Code reuse.** In calculations as complex as quantum field theory calculations, the code used should be thoroughly checked and reinventing the wheel for every calculation should be avoided.
- **Standardization.** In order to reuse code, clear standards should be formulated, thoroughly documented and followed. This should also make it easier to reach common standards with other similar or related projects (using other programming languages) for fields, lagrangians, 4-vectors, etc. in the end allowing easy comparison of calculations of the same process.
- **Repository of lagrangians, amplitudes and integrals.** The agreement on standards allows the creation of a database of quantum field theory results stored in a standardized electronic form. We encourage users

of FeynCalc to contribute back the results they obtain.

- **Algorithm optimization.** The more people using (an implementation of) an algorithm, the better the chances of improving (the implementation of) it.
- **Fast prototyping.** Quickly trying out new quantum field theory models should be made easier by using the building blocks provided by a clear standardization as well as a set of functions for the most common calculational tasks.

Notice in this connection that the standards put forth in this document are what we consider to be the most general but still useful standards for doing quantum fields theory with Mathematica. This means that that standards are to be considered as relatively fixed, but not that we are unwilling to change them given sufficiently good arguments.

From all of the above it should be clear that we consider interaction with users to be crucial for the further evolving of FeynCalc. Therefore, suggestions and in particular code contributions are warmly welcomed. Smaller contributions can be emailed to feyncalc@feyncalc.org; larger contributions should be coordinated with the developers who can be reached at the same email address. Though much testing of the code has been done, there is absolutely no claim that FeynCalc is bug-free. You should be sceptical about the results, and when you are sure the program returned a wrong answer, you are encouraged to send email to feyncalc@feyncalc.org.

1.1 Version History

The roots of FeynCalc go back to 1987. During a stay as a graduate student in Albuquerque, New Mexico, Rolf Mertig learned to program in Macsyma [3] from the experts Stanly Steinberg and Michael Wester. Meanwhile in Germany, elementary particle physicists needed automation of the calculation of Feynman diagrams of eletroweak processes to one loop. The algorithms were provided mostly by Ansgar Denner and Manfred Böhm and implemented in a purely functional way by Rolf Mertig during the years 1987 - 1989.

The basic idea was to have general functions for some of the more mechanical parts of the diagram calculations, generalizable tools for use in calculations of different 1-loop processes, especially 1→2 and 2→2 processes. These included tools for:

- Lorentz algebra. $g^{\mu\nu}\ p_\mu \ \to\ p^\nu\ ,\ \ g^{\alpha\beta}g_{\alpha\beta}\ \to\ n\ \cdots$,
- Dirac algebra. $\gamma^\alpha\ \gamma^\nu\gamma_\alpha\ \to\ (2-n)\,\gamma^\nu,\ \ \mathrm{v}(p,m)\ \not{p} \to$ - m $\mathrm{v}(p,m)\ \cdots$,
- Passarino-Veltman - tensor integral decomposition. Applications of these tools included complete 1-loop processes in the Standard Model, $e^+e^-\ \to\ Z\,H\ ,\ e^+e^-\ \to\ W^+W^-$ and the 2-loop photon self-energy in QED.

However, at the end of 1989 several problems showed up with the Macsyma implementation. The purely functional programming style proved to be difficult to debug and, in fact, inappropriate. The rudimentary pattern matcher in Macsyma was not useful. There was no way to incorporate new functions easily into the whole

Macsyma system, and no possibility of providing online documentation. Furthermore, Macsyma's memory managment ("garbage collection") was slow when handling large expressions.

In early 1990 it became clear that Mathematica was a much more natural (programming) environment for FeynCalc.

1990-1991 : The first version of FeynCalc in Mathematica

In 1990, user-friendly packages were built with extended automatic capabilities (OneLoop, OneLoopSum), and SU(3) algebra capabilities were added (SU3Simplify).

In 1991 initial documentation was written and the program was made available on anonymous ftp-servers: mathsource.wolfram.com and canc.can.nl

Applications from 1990 - 1996 included:

- 1-loop 2 →2 processes in the Standard Model, such as:
  $gg \to t\bar{t}$, $e^+e^- \to Z\,H$, $W^+W^- \to W^+W^-$, $ZZ \to ZZ$
- background field gauge calculations,
- High-energy approximation of $e^+e^- \to W^+W^-$.
- 2-loop Standard Model self-energies.

No attempt was made to provide tools for tree-level calculations. For this purpose other programs appeared, among them another Mathematica package, HIP [4], developed at SLAC.

1992-1995 : FeynCalc 2.0-2.2, unification and simplification.

During this period of development the SU(3) algebra was changed to SU(N). Several tools for automatic tree-level calculation were added, for example the function **`SquareAmplitude`**. (Unfortunately, the documentation was not updated.) All sub-packages were put into one file ($\simeq 10^4$ lines). The result was FeynCalc2.2beta.m.

1993-1996 : FeynCalc 3.0, modularization, typesetting

Due to the rapidly increasing amount of code, FeynCalc has been reorganized in a completely modular way. Each function in a package is a file which is loaded only on demand. For the maintenance of hundreds of packages, totalling 2.5 MegaByte, software engineering was needed.

- The new typesetting capabilities of Mathematica 3 (TraditionalForm) were used to substantially improve the look of the output. Typesetting rules were added for e.g., $\vec{\partial}_\mu$.

- Code was written to allow new abstract datatypes, for example for noncommutative algebra and for special integrals.
- QCD tools for the Operator Product Expansion (OPE) were added.
- Automatic Feynman rule derivation (with functional differentation) was coded, in order to get special Feynman rules for twist-2 (and higher) operators.

1997-2000 : FeynCalc 4.0-4.1, QCD and OPE, ChPT, tables

In these years Rolf Mertig made many improvements to the code, driven by his own work in perturbative Quantum Chromo Dynamics (QCD).

One effort was to build data bases: Convolutions, integrals, tensor integral transformation formulas, Feynman rules, and Feynman parameterizations. Applications include 2-loop spin-dependent and spin-independent Altarelli-Parisi Splitting functions.

Another was the package TARCER, which was initially an independent project, but then integrated into FeynCalc. TARCER [6] adds two-loop functionality for propagator-type integrals using the recurrence relations of Tarasov [5].

In 2000, maintanance of the package was taken over by Frederik Orellana. He worked in Chiral Perturbation Theory [7] (ChPT) and made some changes, as well as adding code to FeynCalc in order to support ChPT and effective theories in general, and interfacing with FeynArts. Also, a first implementation of the 'tHooft-Veltman formulae [8] for numerical evaluation of the one-loop integrals $B0, C0$ and $D0$ was added.

1.2 Installation

- Download the file "HighEnergyPhysics-4.2.tar.gz" or "HighEnergyPhysics-4.2.zip" (if you're using using Windows you'll probably want the zip file).
- Place the file in the directory "mathhome/AddOns/Applications", where "mathhome" is the directory containing your Mathematica installation. If you're on a Unix system, "mathhome" can also be " /.Mathematica/x", where x is the version of your Mathematica installation. (if " /.Mathematica/x/AddOns/Applications" does not exist, you can safely create it).
- Make sure there is not already a directory "HighEnergyPhysics" (and move it out of the way if there is).
- Unpack the file: Under UNIX, type `tar -xvzf HighEnergyPhysics-4.2.tar.gz`; under Windows and MacOS, use some utility like WinZip or StuffIt Expander.
- If you've made any customizations in the configuration file FCConfig.m, merge them from the file you've moved away into the new file.

- Start Mathematica.
- Choose 'Rebuild Help Index' from the 'Help' menu.
- Load FeynCalc with **«HighEnergyPhysics`FeynCalc`**

2 Input Functions

The FeynCalc syntax for metric tensors, four-vectors, Dirac matrices, etc., is such that the positioning of the arguments automatically defines the nature of the variables. No declaration of Lorentz indices or four-vectors has to be made. Covariant and contravariant Lorentz indices are treated on an equal basis, assuming summation over (twice) repeated indices, where each pair of indices are assumed to be co- and contravariant respectively. Since explicit components of Lorentz tensors are not used, this convention causes no problems. The ambiguity in the sign of the Levi-Civita tensor can be fixed by an option of the function **DiracTrace** and **EpsChisholm**, see page 42.

The input functions are macros that are immediately translated into the internal representation of FeynCalc. But these sometimes lengthy internal representations are difficult to recognize when working interactively. Therefore, some effort has been put into making the output of FeynCalc look as much as possible like typeset formulas. The typesetting of FeynCalc depends on the input/output interface in use.

- **Terminal interface.** Per default FeynCalc sets the global Mathematica variable **$PrePrint** to **FeynCalcForm**, forcing the display to be in a more readable manner. It is important to realize that the display given by **FeynCalcForm** is different from the input syntax. Thus, you may not give as input, for example, a scalar product in the result of a trace calculation by copying the form you see on the screen. To have the output be the internal representation, set **$PrePrint=.**. To return to formatted output, set **$PrePrint = FeynCalcForm**.
- **Notebook interface.** FeynCalc per default automatically sets the Mathematica default output format type to **TraditionalForm**. Because FeynCalc defines the **TraditionalForm**-look of its objects, the output looks very much like typeset formulas. Again, the screen output cannot be copied and used as input. If the Mathematica default output format type [1] is set to **InputForm** or **StandardForm**, the screen output is the internal representation of FeynCalc and can be copied and used as input. To obtain the same output as with a terminal interface, set the Mathematica default output format type to **OutputForm** and set **$PrePrint = FeynCalcForm**. To return to formatted output, set the Mathematica default output format type to **TraditionalForm**.

In the rest of this guide, the examples given will use the notebook interface.

The internal structure of FeynCalc is explained briefly in section 3 and in the reference section 8. Usually it is not necessary to know it. Notice simply that there are three layers of FeynCalc: The input functions, an output representation of those and the internal structure.

In order to do calculations in the framework of D-dimensional regularization, FeynCalc can be used for calculations in 4, D dimensions. The dimension of metric tensors, Dirac matrices and four-vectors is specified by setting the option **Dimension** of the corresponding input functions. The default is four. If FeynCalc is used for interactive D-dimensional calculations, you should change the option of the input functions. Note that Lorentz indices,

[1] The Mathematica default output format type is set using the menu "Cell" → "Default Output Format Type".

momenta and Dirac matrices carry their dimension locally, i.e., for calculating a Dirac trace in D dimensions the dimension of the Dirac matrices has to be set accordingly, thus there is no need of a dimension option for e.g. the function **DiracTrace**.

2.1 Lorentz Tensors and Scalar Products

FourVector[p**,** μ**]**	four-vector p^μ
LeviCivita[μ**,** ν**,** ρ**,** σ**]**	Levi-Civita tensor $\varepsilon^{\mu\nu\rho\sigma}$
MetricTensor[μ**,** ν**]**	metric tensor $g^{\mu\nu}$
ScalarProduct[p**,** q**]**	$p \cdot q$

Basic Lorentz tensor input functions.

option name	*default value*	
Dimension	**4**	number of space-time dimensions

Option for **FourVector**, **MetricTensor**, **LeviCivita** and **ScalarProduct**.

Internally, the input given with **FourVector**, **MetricTensor**, **LeviCivita** and **ScalarProduct** is represented in terms of the lower level functions **Pair**, **Momentum** and **LorentzIndex** (this is elaborated in section 3.4).

The following examples show the input syntax and the typesetting of FeynCalc.

First we have to load FeynCalc.

```
In[1]:= «HighEnergyPhysics‘FeynCalc‘
```

Enter a scalar product $p \cdot q$.

```
In[2]:= ScalarProduct[p, q]
```

This is the typeset output.

```
Out[2]= p.q
```

This is the input for $g^{\mu\nu}$.

```
In[3]:= MetricTensor[μ, ν]
```

```
Out[3]= g^μν
```

A metric tensor $g^{\mu\nu}$ in **D** dimensions is entered in this way. The number of dimensions is suppressed in the screen output (for notebook output. For terminal output, the number of dimensions is displayed as an index, g $_{\text{D}}$**[mu, nu]**).

```
In[4]:= MetricTensor[μ, ν, Dimension → D]
Out[4]= g^μν
```

A four-dimensional p_μ.

```
In[5]:= FourVector[p, μ]
Out[5]= p_μ
```

You may also enter a linear combination of four-vectors: $(p-2\,q)_\mu$.

```
In[6]:= FourVector[p - 2 q, μ]
Out[6]= (p - 2 q)_μ
```

This is a D-dimensional q_μ.

```
In[7]:= FourVector[q, μ, Dimension → D]
Out[7]= q_μ
```

A polarization vector $\varepsilon_\mu(k)$ is a special four-vector.

```
In[8]:= PolarizationVector[k, μ]
Out[8]= ε_μ(k)
```

This is how to enter a $\varepsilon^*_\mu(k)$.

```
In[9]:= Conjugate[PolarizationVector[k, μ]]
Out[9]= ε*_μ(k)
```

Polarization vectors are represented in a special way (see page 42). The transversality condition $\varepsilon(k)\cdot k = 0$ is automatically fulfilled (see page 31). Polarization vectors are defined in FeynCalc in four dimensions only. For **FourVector**, **MetricTensor**, **LeviCivita**, and **ScalarProduct** other dimensions may be specified with the option **Dimension**.

```
PolarizationVector[k, mu]                    ε_μ(k)
Conjugate[PolarizationVector[k, mu]]         ε*_μ(k)
```

The input for a polarization vector and polarization.

2.2 Dirac Matrices

DiracMatrix[*mu***, *nu*, ...]**	Dirac matrices $\gamma^\mu\ \gamma^\nu \ldots$
DiracSlash[p, q, ...]	Feynman slashes $\not{p}\not{q}\ldots$
DiracMatrix[5]	γ^5
ChiralityProjector[+1]	$\omega_+ = (1+\gamma^5)/2$
DiracMatrix[6]	$\gamma^6 = (1+\gamma^5)/2$
ChiralityProjector[-1]	$\omega_- = (1-\gamma^5)/2$
DiracMatrix[7]	$\gamma^7 = (1-\gamma^5)/2$

Various input functions for Dirac matrices.

A Dirac matrix γ^μ is represented by **DiracMatrix[mu]**. For $\not{p} = p_\mu \gamma^\mu$ you may use **DiracSlash[p]**. Products of Dirac matrices or slashes can either be entered by adding subsequent arguments, i.e., **DiracMatrix[***mu***, *nu*, ...]** and **DiracSlash[p, q, ...]**, or by multiplying the **DiracMatrix** and **DiracSlash** with the Mathematica **Dot**, ".".

The Mathematica **Dot**, ".", is used in the input as noncommutative multiplication operator for the objects **DiracMatrix**, **DiracSlash**, **Spinor**, **LeptonSpinor**, **QuarkSpinor** and **GellMannMatrix**. The "." may also be used as a delimiter instead of the "," within the functions **DiracMatrix**, **DiracSlash** and **GellMannMatrix**.

In the output with **FeynCalcForm** the "." as noncommutative multiplication operator is suppressed.

This is how you enter $(\not{p}+\not{q}+m)\gamma^\mu$.

```
In[10]:= (DiracSlash[p + q] + m) . DiracMatrix[μ]
Out[10]= (m + γ · (p + q)) γ^μ
```

Here is $\gamma^\mu \gamma^5 \gamma^\nu \gamma^6 \gamma^\rho \gamma^7$.

```
In[11]:= DiracMatrix[μ, 5, ν, 6, ρ, 7]
Out[11]= γ^μ γ^5 γ^ν γ^6 γ^ρ γ^7
```

This is a four-dimensional product: $2\not{b}\not{a}2\,(\not{d}-\not{c})\,(6\not{q}-3\not{p})$.

```
In[12]:= DiracSlash[2b, a, 2(d - c), 6q - 3p]
Out[12]= 2γ · b  γ · a  2 (γ · d − γ · c)  (6 γ · q − 3  γ · p)]
```

Slashed polarization vectors $\not{\varepsilon}(k)$ are entered in this way.

```
In[13]:= DiracSlash[Polarization[k]]
Out[13]= γ · ε(k)
```

option name	*default value*	
Dimension	**4**	space-time dimension

Option for **DiracMatrix** and **DiracSlash**.

As setting for **Dimension**, a Mathematica **Symbol** *dim*, or *dim*-4 are possible.

Entering of a D-dimensional γ^μ. The number of dimensions is suppressed in the screen output (for notebook output).

```
In[14]:= DiracMatrix[μ, Dimension → D]
Out[14]= γ^μ
```

This is $\gamma^\mu \gamma^\mu$ in $D-4$ dimensions.

```
In[15]:= DiracMatrix[mu, mu, Dimension → D - 4]
Out[15]= γ^μ γ^μ
```

2.3 Spinors

In FeynCalc spinors are entered with the four functions **SpinorU**, **SpinorUBar**, **SpinorV**, **SpinorVBar**. Products of spinors with spinors or Dirac matrices (Fermion chains) are built with the Mathematica **Dot**, ".".

SpinorU[p**,** m**]**	$u(p,m)$
SpinorUBar[p**,**m**]**	$\overline{u}(p,m)$
SpinorV[p**,** m**]**	$v(p,m)$
SpinorVBar[p**,**m**]**	$\overline{v}(p,m)$

The input functions for fermionic spinors.

This is $\overline{u}(p,m)\not{p}$.

```
In[16]:= SpinorUBar[p, m] . DiracSlash[p]
Out[16]= u-bar(p, m) γ · p
```

On the right of $\not{p}$ the spinor, $u(p,m)$ is used.

```
In[17]:= DiracSlash[p] . SpinorU[p, m]
Out[17]= γ · p u(p, m)
```

This is $\overline{v}(p,m)\not{q}\not{p}$.

```
In[18]:= SpinorVBar[p, m] . DiracSlash[q, p]
Out[18]= v-bar(p, m) γ · q γ · p
```

Here we have $\gamma \cdot p\, v(p,m)$.

```
In[19]:= DiracSlash[p] . SpinorV[p, m]
Out[19]= γ · p v(p, m)
```

Enter a spinor obeying the massless Dirac equation. The **FeynCalcForm** suppresses the 0.

```
In[20]:= SpinorU[p, 0]
Out[20]= u(p)
```

You may also omit the second argument for a massless spinor.

```
In[21]:= SpinorU[p]
Out[21]= u(p)
```

2.4 SU(N) Matrices and Structure Constants

FeynCalc provides functions for the input of SU(N) matrices and the corresponding structure constants. More specialized input functions in SU(2) and SU(3) (Gell-Mann and Pauli matrices) are provided by the subpackage PHI. These are described in the PHI user's guide ???.

SUNT[a]	the a'th generator, T_a, of SU(N) in the fundamental representation

Input of SU(N) matrices.

For noncommutative multiplication of SU(N) matrices, use the Mathematica **Dot**, ".".

This is $T_a\, T_b\, T_c$.

```
In[22]:= SUNT[a] . SUNT[b] . SUNT[c]
Out[22]= T_a T_b T_c
```

The above can also be entered like this.

```
In[23]:= SUNT[a, b, c]
Out[23]= T_a T_b T_c
```

This is how you enter $T_a\, T_b\, T_c\, \delta_{cd}$.

```
In[24]:= SUNT[a, b, c] SUNDelta[c, d]
Out[24]= T_a T_b T_c δ_cd
```

SUNDelta[i,j]	Kronecker delta function in SU(N)
SUNF[i,j,k]	antisymmetric structure constants of SU(N)
SUND[i,j,k]	symmetric structure constants of SU(N)

Input of SU(N) structure constants.

option name	*default value*	
Explicit	**True**	replace **SUNF** or **SUND** by traces

An option for the structure constants **SUNF** and **SUND**.

This is how you enter f_{abc}.

```
In[25]:= SUNF[a, b, c]
Out[25]= f_abc
```

This is how you enter d_{abc}.

```
In[26]:= SUND[a, b, c]
Out[26]= d_abc
```

They can be expressed in terms of the generating matrices (see section 4.4 for a discussion of the trace).

```
In[27]:= SUNF[a, b, c, Explicit → True]
Out[27]= 2 i ( tr(T_a T_c T_b) - tr(T_a T_b T_c) )
```

Antisymmetry, $f_{acb} = -f_{abc}$.

```
In[28]:= SUNF[a, b, c, Explicit → True] +
         SUNF[a, c, b, Explicit → True] //
         Expand
Out[28]= 0
```

Symmetry, $d_{acb} = d_{abc}$.

```
In[29]:= SUND[a, b, c, Explicit → True] -
         SUND[a, c, b, Explicit → True] //
         Expand
Out[29]= 0
```

2.5 Denominators of Propagators

The denominators of propagators are entered in a special way. Each one-loop Feynman amplitude has factors of the form

$$df = \frac{1}{[q^2 - m_0^2]\,[(q+p_1)^2 - m_1^2]\,[(q+p_2)^2 - m_2^2]\,\ldots}$$

with q as loop momentum and the p_i denoting linear combinations of the external four-vectors.

FeynAmpDenominator[PropagatorDenominator[q**,** m_0**], PropagatorDenominator[**q **+** p**,** m_1**], ...]**	$1/([q^2 - m_0^2][(q+p)^2 - m_1^2]) \ldots$

Entering denominators of propagators.

This is $1/([q^2 - m_1^2][(q+p)^2 - m_2^2])$.

```
In[30]:= FeynAmpDenominator[ PropagatorDenominator[q,
         m1], PropagatorDenominator[q + p, m2]]
```

$$\text{Out[30]=} \quad \frac{1}{(q^2 - m_1^2)((q+p)^2 - m_2^2)}$$

2.6 Small Variables

For radiative corrections in the standard model, fermionic masses often can be neglected with respect to the gauge boson masses. Since, however, some of the Passarino-Veltman integrals may be infrared divergent, the fermionic (and photonic) small masses must remain as arguments of them and cannot be set to 0. In order to achieve this behavior, these small masses have to be entered with head **Small**. The idea is that every variable with head **Small** evaluates to 0, unless it is an argument of a scalar integral.

You can avoid to explicitly wrap the head **Small** around a mass by using the option **SmallVariables** of the function **OneLoop** (the function **OneLoop** is described in section 5.1).

Small[m**]**	head of a small mass m, $m \ll M$

A head for small variables.

2.7 Quantum Fields

FeynCalc is meant to be a general framework for calculations in quantum field theory. Thus, FeynCalc defines notation not only for working with what comes out of a Feynman amplitude calculation, namely loop integrals,

momenta, Dirac and SU(*N*) structures, propagators, etc., but also for the quantities needed to work out the input to such calculations, namely quantum fields and lagrangians.

QuantumField[*par1***,** *par2***, ... ,** *ftype***, {***lorind***}, {***sunind***}]**	denotes a quantum field of type *ftype* with possible Lorentz indices *lorind* and SU(*N*) indices *sunind*. The optional first arguments *par1*, *par2*, $\cdots$, are partial derivatives acting on the field
PartialD[μ]	denotes a space-time derivative when given as first argument to **QuantumField**

Input of quantum fields.

Some field ψ.

```
In[31]:= QuantumField[ψ]
Out[31]= ψ
```

If the field is considered as part of an SU(*N*) multiplet and the SU(*N*) index is *i*, this is the notation.

```
In[32]:= QuantumField[ψ, {i}]
Out[32]= ψ_i
```

Some field *g* with an SU(*N*) index *i* and a Lorentz index ν.

```
In[33]:= QuantumField[g, {ν}, {i}]
Out[33]= g^i_μ
```

This is the notation for the space-time derivative $\partial/\partial x^\mu$ of the same field.

```
In[34]:= QuantumField[PartialD[μ],g, {ν}, {i}]
Out[34]= ∂_μ g^i_ν
```

3 Inside FeynCalc

This section, which may be skipped on a first read, deals with the internals of FeynCalc. For advanced users and people considering contributing to FeynCalc, it is a must-read. ??? fill in some more bla bla ???

3.1 Startup Sequence, Modules, Add-ons and Extensibility

FeynCalc is loaded with the following command:

```
«HighEnergyPhysics‘FeynCalc‘
```

This causes the program file "FeynCalc.m" to be loaded. This file contains definitions of the core objects, used by higher level, functions. The more important of these objects are: **DiracGamma**, **FeynAmpDenominator**, **FourVector**, **LorentzIndex**, **Momentum**, **NonCommutative**, **Pair**, **PartialD**, **PropagatorDenominator**, **QuantumField**, S**UNIndex**, **ScalarProduct**, **Spinor**.

"FeynCalc.m" also scans the subdirectories of the directory "HighEnergyPhysics" for files with an extension ".m". Each such file, e.g. "FeynRule.m", should contain the definition of a function, e.g. **FeynRule**.

Each definition, whether in one of these files or in "FeynCalc.m" is, however, not loaded into memory. Instead the function name is declared using **DeclarePackage**, so that when this function is used the first time, the definition is loaded into memory. This standard technique significantly reduces startup time and memory consumption. Which subdirectories are scanned is defined in "FeynCalc.m". The idea is that each directory corresponds to a module, that is, a group of functions pertaining to some subject within quantum field theory. The core modules are:

- **fctables.** Databases of lagrangians, amplitudes and integrals. The more important functions are: **B0**, **C0**, **Lagrangian**, **Amplitude**, **Integrate2**.
- **fcloops.** Tools for calculating loop integrals. The more important functions are: **OneLoop**, **PaVeReduce**.
- **fctools.** Various tools for working with quantum fields and amplitudes. The more important functions are: **Contract**, **DiracReduce**, **FunctionalD**, **FeynRule**, **Tr**, **FermionSpinSum**.
- **general.** Mathematical tools of a general nature that are used by one or more functions of the other subpackages, but may be useful in other contexts as well.
- **qcd.** Tools for working with QCD. The more important functions are: **???**.

The governing idea of the file layout of FeynCalc is modularity. The files belonging to each module are in a separate directory; each file corresponds to one function. If a function uses some support functions or options that are not used by other functions but desirable to have globally accessible, one may put these in the same context as the function and add them to the list **multifunpack** in "FeynCalc.m". The dependence on functions in other contexts should be clearly stated with **MakeContext** statements at the beginning of each file.

The modules listed above all adhere to these conventions.

As will be noticed there are some files and directories not mentioned so far. Apart from the directory "Documentation", which is a directory for standard Mathematica package documentation, they are listed below under the packages to which they belong. These packages are not loaded by default, but must be enabled by setting certain configuration variables (see section 3.2). The reasons for this are: These packages are not considered essential for all users and also don't follow the loading conventions described above, but load all definitions on startup and thus take up memory and take some time in loading (still of the order of seconds though).

- **FeynArts** is made up of the file "FeynArts.m" , the directories "GraphInfo", "Models" and some more files on the top level of "HighEnergyPhysics". It is an independent package by Sepp Kueblbeck, Hagen Eck and Thomas Hahn for generating Feynman graphs and corresponding amplitudes with Feynman rules as input. Integration within FeynCalc is not intended, but can, with a few modifications, be achieved (see PHI below). For more information on FeynArts, see ref 2.
- **PHI** is a package by Frederik Orellana providing utilities for working with effective field theories and interacting with FeynArts. All files belonging to PHI are contained in the directory "Phi". It was not conceived as a FeynCalc subpackage and the file layout does not adhere to the FeynCalc conventions. Never the less, it has been modified to effectively act as a FeynCalc subpackage and is now distributed only with FeynCalc. For more information on PHI, see ref. ??? and the PHI user's guide ???.
- **TARCER** is a package by Rolf Mertig for working with 2-loop propagator type integrals. Its file layout also does not adhere to the FeynCalc conventions, but it also is integrated tightly within FeynCalc. The files belonging to TARCER are contained in the directory "Tarcer". For more information on FeynArts, see ref. 6.
- **fcdevel** is code which is not considered production ready and is meant to be used by developers only.

3.2 Configuration and Runtime Variables

As we have seen, the behaviour of most functions is controlled by options like e.g. **`Dimension`**. Additionally the behaviour of some functions depends on the setting of certain environment variables. A few of these should be set before loading FeynCalc, namely **`$LoadTARCER, $LoadPhi, $LoadFeynArts`**.

variable name	*default value*	
`$LoadTARCER`	**`False`**	load TARCER
`$LoadPhi`	**`False`**	load PHI
`$LoadFeynArts`	**`False`**	load FeynArts

Variables switching loading of extra packages on and off.

Others can be set at runtime; e.g. **$Color, $Covariant, $Gauge, $NonComm, $Abbreviations**. **$BreitMaison** and **$Larin** are special cases in that they are set at runtime, but, once set cannot be changed.

variable name	*default value*	
$Color	**False**	colour special variables
$Covariant	**True**	display lorentz indices as upper or lower indices
$Gauge	**1** (Feynman gauge)	the gauge fixing parameter of QED in Lorentz gauge. Notice that **$Gauge** is used by some functions, the option **Gauge** by others
$NonComm	**{DiracGamma, DiracGammaT, ..., OPESum}**	a list of all noncommutative heads
$Abbreviations	**{"^" → "", "*" → "", ..., "Vector" → "V"}**	a list of string substitution rules used when generating names for storing intermediate results. Used by **OneLoop** and **PaVeReduce**
$BreitMaison	**False**	use the Breitenlohner-Maison γ^5 scheme (currently not supported)
$Larin	**False**	use the Larin-Gorishny-Atkyampo-DelBurgo γ^5 scheme
$VeryVerbose	**0**	display intermediate messages from FeynCalc
$MemoryAvailable	**8**	the amount of available main memory in megabytes
$SpinorMinimal	**False**	a global switch for an additional simplification attempt in **DiracSimplify** for more than one Spinor line

Runtime variables.

The variable **$VeryVerbose** may be set to 0, 1, 2 or 3. The higher the setting the more intermediate information is printed during the calculations.

FeynCalc is designed in such a way that certain intermediate results are stored in order to speed up the computations. This of course may become too memory consuming. The storing stops, when the value of **10^(-6)*MemoryInUse[ ]** gets bigger than **$MemoryAvailable**-1.

The default of FeynCalc is to use the naive γ^5 prescription, i.e., γ^5 anticommutes with D - dimensional Dirac

matrices. Setting **$BreitMaison** to **True** changes the commutation behavior of γ^5[2]: It anti-commutes with Dirac matrices in four dimensions, but commutes with the (D-4)-dimensional part. The basic features of the Breitenlohner-Maison scheme are implemented in FeynCalc, but they have not very thoroughly been tested (some simple calculations are given in section 4.2). Therefore one should check the results for correctness.

3.3 Data Types

In quantum field theory, a number types of variables are needed. E.g. variables representing Lorentz indices, momenta, SU(*N*) indices and quantum fields. As we have seen in previous sections, the way to tell FeynCalc that a variable is of one of these types is to wrap some head around it. However, FeynCalc provides also a method for having variables be of a certain type without wrapping a head around them. The knowledge about such variables is carried by the function **DataType**. The default setting is

```
DataType[__, _] := False
```

For example, to assign the data-type **PositiveInteger** to the variable **x**, do

```
DataType[x, PositiveInteger] = True
```

DataType[*x, type***] = True**	defines the symbol *x* to have data-type *type*
DataType[*x1, x2, ..., type***]**	defines the symbols *x1*, *x2*, ... to have data-type *type*

The function carrying the knowledge of FeynCalc data-types.

[2] In fact, how FeynCalc treats γ^5 depends on the setting of the variables **$BreitMaison**, **$Larin** and **$West**. They are all boolean variables; the first two specify the γ^5 scheme. None of them have been neither thoroughly implemented nor tested. If **$West** is set to True (which is the default), traces involving more than 4 Dirac matrices and a γ^5 are calculated recursively according to formula (A.5) from [11], which is based on the Breitenlohner Maison -scheme.

NonCommutative	a variable treated as noncommutative by **Dot**, "."
PositiveInteger	a positive integer variable
NegativeInteger	a negative integer variable
PositiveNumber	a positive number variable
FreeIndex	a Lorentz index not to be contracted by **contract**
GrassmannParity	**DataType[F, GrassmannParity] = 1** declares a field **F** to be bosonic, **DataType[F, GrassmannParity] = -1** declares it to be fermionic.

FeynCalc data-types.

Declare some symbols **f** and **g** to be noncommutative.

```
In[35]:= DataType[f, g, NonCommutative] = True;
```

An algebraic expression using **Dot**, ".".

```
In[36]:= t = f.g - g.(2a).f
```

```
Out[36]= f.g − g.(2a).f
```

DotSimplify only extracts **a** out of the noncommutative product.

```
In[37]:= DotSimplify[t]
```

```
Out[37]= f.g − 2a g.f
```

Declare some symbols **m** and **a** to be of data-types **even** and **odd**.

```
In[38]:= DataType[m, odd] = DataType[a, even] =
           True;
```

Define functions for reducing powers of -1.

```
In[39]:= ptest1[x_] := x /.  (-1)^n_ :> -1 /;
           DataType[n, odd] ;
         ptest2[x_] := x /.  (-1)^n_ :> 1 /;
           DataType[n, even];
```

Redefine the expression **t**.

```
In[40]:= t = (-1)^m + (-1)^a + (-1)^z
```

```
Out[40]= (−1)^a + (−1)^m + (−1)^z
```

Apply the functions to the expression **t**.

```
In[41]:= ptest1[t]
```

```
Out[41]= −1 + (−1)^a + (−1)^z
```

```
In[42]:= ptest2[%]

Out[42]= (-1)^z
```

Clear **t**.

```
In[43]:= Clear[t];
```

DeclareNonCommutative[*x***,** *y***, ...]**	sets **DataType[***x***, NonCommutative] = True; DataType[***y***, NonCommutative] = True ; ...**

Declaration of noncommutative variables.

3.4 Output Forms and Internal Representation

By default, the internal representation is hidden. For some purposes, like debugging or hacking the FeynCalc source, however, it may be useful to know how the various objects are represented in FeynCalc.

Some input functions are transformed right away to the internal representation, others only on being acted upon by utility functions. An example of the first kind is **DiracMatrix[μ]**. An example of the second kind is **FourVector[p, μ]**; during the internal evaluation of e.g. **Contract[FourVector[p, μ], FourVector[p, μ]]**, however, a conversion to the internal representation is done. To see the internal representation of an object, the function **FeynCalcInternal** can always be used.

The basic idea is to wrap a special head around each special input variable. For example, the arguments of **MetricTensor[mu, nu]** each get a head **LorentzIndex**. These heads have only a few functional definitions. A **D**-dimensional **LorentzIndex[mu, D]**, for example, simplifies to **LorentzIndex[mu]** when **D** is replaced by 4.

Inside FeynCalc the functions and rules are specified in such a way that they only apply to those objects with the right head. This programming style, which makes strong use of the pattern matching capabilities of Mathematica, especially the so-called upvalue function definitions, has proven to be useful and reliable for the purpose of algebraic calculations in physics.

Pair[*a***, *b***]**	is a special pairing used in the internal representation: *a* and *b* may have heads **LorentzIndex** or **Momentum**. If both *a* and *b* have head **LorentzIndex**, the metric tensor is understood. If *a* and *b* have head **Momentum**, a scalar product is meant. If one of *a* and *b* has head **LorentzIndex** and the other **Momentum**, a Lorentz vector (e.g. p_μ) is understood
Momentum[*p***]**	is a four-dimensional momentum. For other than four dimensions: **Momentum[***p***, *d***]**. **Momentum[***p***, 4]** simplifies to **Momentum[***p***]**
LorentzIndex[μ]	is a four-dimensional Lorentz index. When μ is an integer, it evaluates to mbExplicitLorentzIndex[μ]. For other than four dimensions: **LorentzIndex[***p***, *d***]**. **LorentzIndex[***p***, 4]** simplifies to **LorentzIndex[***p***]**
ExplicitLorentzIndex[μ]	is an explicit Lorentz index, i.e., μ is an integer

Internal representation of Lorentz input functions.

Now we shall study the internal representation through a few examples. **FeynCalcInternal** shall be applied when necessary.

Suppress FeynCalc typesetting by going from **TraditionalForm** output to **StandardForm** output. This can also be achieved via the menu setting of "Cell" → "Default Output Format Type".

```
In[44]:= SetOptions[$FrontEnd,
         "CommonDefaultFormatTypes"->"Output"
         -> StandardForm]
```

This is the internal representation of a γ^μ in four dimensions.

```
In[45]:= DiracMatrix[μ]

Out[45]= DiracGamma[LorentzIndex[μ]]
```

Here is a γ^μ in D dimensions.

```
In[46]:= DiracMatrix[μ, Dimension→D]

Out[46]= DiracGamma[LorentzIndex[μ, D], D]
```

This is a product of a four-dimensional $\not{q}$ and a D-dimensional $\not{q}$.

```
In[47]:= DiracSlash[q] . DiracSlash[q, Dimension → D]

Out[47]= DiracGamma[Momentum[q]] . DiracGamma[Momentum[q, D], D]
```

This shows the convention for a four-dimensional scalar product.

```
In[48]:= DiracSimplify[%]

Out[48]= Pair[Momentum[q], Momentum[q]]
```

The head **Pair** is also used for metric tensors; in this case in **D** dimensions. The dimension is provided as an argument.

```
In[49]:= MetricTensor[μ, ν, Dimension → D]

Out[49]= Pair[LorentzIndex[μ, D], LorentzIndex[ν, D]]
```

Changing **D** to 4 recovers the default for a four-dimensional $g^{\mu\nu}$.

```
In[50]:= % /. D → 4

Out[50]= Pair[LorentzIndex[μ], LorentzIndex[ν]]
```

A four-vector does not immediately evaluate to the internal representation.

```
In[51]:= FourVector[p, μ]

Out[51]= FourVector[p, μ]
```

By applying **FeynCalcInternal** we force the internal representation to be used: A four-vector is also represented as a **Pair** internally.

```
In[52]:= FourVector[p, μ] // FeynCalcInternal

Out[52]= Pair[LorentzIndex[μ], Momentum[p]]
```

By applying some utility function, transformation to the internal representation is automatically done.

```
In[53]:= FourVector[p, μ] FourVector[p, μ] // Contract

Out[53]= Pair[Momentum[p], Momentum[p]]
```

A $(D-4)$-dimensional four-vector.

```
In[54]:= FourVector[p, μ, Dimension → D - 4] // FeynCalcInternal

Out[54]= Pair[LorentzIndex[μ, -4 + D], Momentum[p, -4 + D]]
```

It vanishes after changing the dimension.

```
In[55]:= % /. D → 4

Out[55]= 0
```

A special kind of 4-vectors are polarization vectors. This shows the internal representation of these.

```
In[56]:= PolarizationVector[k, μ]

Out[56]= Pair[LorentzIndex[μ], Momentum[Polarization[k]]]
```

DiracGamma[*x***]**	head of a four-dimensional γ^μ or $\not{p}$
DiracGamma[*x***, *D*]**	head of a D-dimensional γ^μ or $\not{p}$

Internal representation of Dirac matrices.

Polarization[*k***]**	A polarization vector $\varepsilon(k)$ is understood when the argument of **Momentum** has head **Polarization**

Internal representation of Polarization vectors.

Internally in FeynCalc, all spinors are represented with the same function, **Spinor**. Which of the Dirac spinors u, v and the conjugate spinors $\overline{u}, \overline{v}$ are understood, depends on the position of the spinors in the Dirac chain and the sign of the momentum argument.

Spinor[*p***, *m*]**	$u(p, m)$ or $\overline{u}(p, m)$
Spinor[-*p***, *m*]**	$v(p, m)$ or $\overline{v}(p, m)$

Internal representation of spinors.

The internal representation of $v(-p,m)$ and $u(p,m)$ is the same.

```
In[57]:= SpinorV[-p, m] - SpinorU[p, m] //
         FeynCalcInternal

Out[57]= 0
```

Analogously to Lorentz indices, SU(N) indices are also in the internal representation wrapped with a head.

SUNIndex[*i***]**	is an SU(N) index. When i is an integer, it evaluates to ExplicitSUNIndex[i]
ExplicitSUNIndex[*i***]**	is an explicit SU(N) index, i.e., i is an integer

Internal representation of SU(N) input functions.

After having completed a calculation with FeynCalc and obtained some screen output, a natural question will often arise: How does one export the results to a file or to some other application. The screen output in **TraditionalForm** is not very useful; copying and pasting it to a text editor will produce some very garbled string with all the typesetting codes of Mathematica included. Saving an expression with **Put** or **save** will produce the same as copying the screen output in **InputForm**, namely the internal representation of FeynCalc, which is also not very useful, because it contains all the heads, options, etc. FeynCalc needs to make sense of the expression. For these reasons the FeynCalc external output format exists. An expression in the internal representation is translated to the FeynCalc external output format with the command **FeynCalcExternal**. The inverse translation is done with **FeynCalcInternal**.

FeynCalcExternal[*exp***]**	translates *exp* from the internal FeynCalc representation to the simpler external one (i.e., FV, GA, GS, etc.)
FeynCalcInternal[*exp***]**	translates *exp* into the internal FeynCalc representation

Translation to and from external output format.

option name	*default value*	
FinalSubstitutions	**{}**	additional translation rules

Option of **FeynCalcExternal** and **FeynCalcInternal**.

The external output form of γ^{μ}.

```
In[58]:= FeynCalcExternal[DiracMatrix[μ],
         FinalSubstitutions → {μ → mu}]

Out[58]= GA[mu]
```

Translation back to the internal representation.

```
In[59]:= FeynCalcInternal[%, FinalSubstitutions
         → {mu → σ}]

Out[59]= DiracGamma[LorentzIndex[σ]]
```

Switch back to FeynCalc typesetting by going from **StandardForm** output to **TraditionalForm** output. This can also be achieved via the menu setting of "Cell" → "Default Output Format Type".

```
In[60]:= SetOptions[$FrontEnd,
         "CommonDefaultFormatTypes"->"Output"
         -> TraditionalForm]
```

3.5 Help system

The help system follows standard Mathematica conventions: Description of an object **obj** is obtained by evaluating **?obj**.

E.g. the description of the function **FeynRule** is obtained by evaluating **?FeynRule**.

```
In[61]:= ?FeynRule

Out[61]= FeynRule[lag, fields] gives the Feynman rule corresponding
         to the field configuration fields of the lagrangian lag.
```

More in-depth information about e.g. **FeynRule** can be obtained by typing in FeynRule in the AddOns part of the help browser (see figure 1).

The files configuring what appears in the help browser are: "FeynCalcBook.nb" and "BrowserCategories.m" in the directory "HighEnergyPhysics/Documentation/English". The first file is a notebook containing the actual

content organized in cells, all of which carry a cell tag. The second file defines the organization of the contents in terms of the cell tags. After a new install of FeynCalc or a change to one of these files the, "Rebuild Help Index" from the "Help" menu must be selected. Then, after clicking the navigation button "Add-ons" in the help browser window, in the left pane "High Energy Physics" will appear. Clicking this and the resulting fields in the other three panes will cause FeynCalc help information to appear. The second pane has various introductory material and the field "Functions". Clicking this field causes the appearance in the third pane of the fields "Core", "Tools", "Loops", "Tables", "QCD" and "General" corresponding to the file organization of FeynCalc as described in section 3.1. Clicking each of these will display a list of the corresponding objects in the fourth pane. Each of these lists is organized in groups of objects, with the groups in the following order: Functions, abbreviations, constants, options and finally internal functions which are described only for completeness, but are of no interest for other than developers.

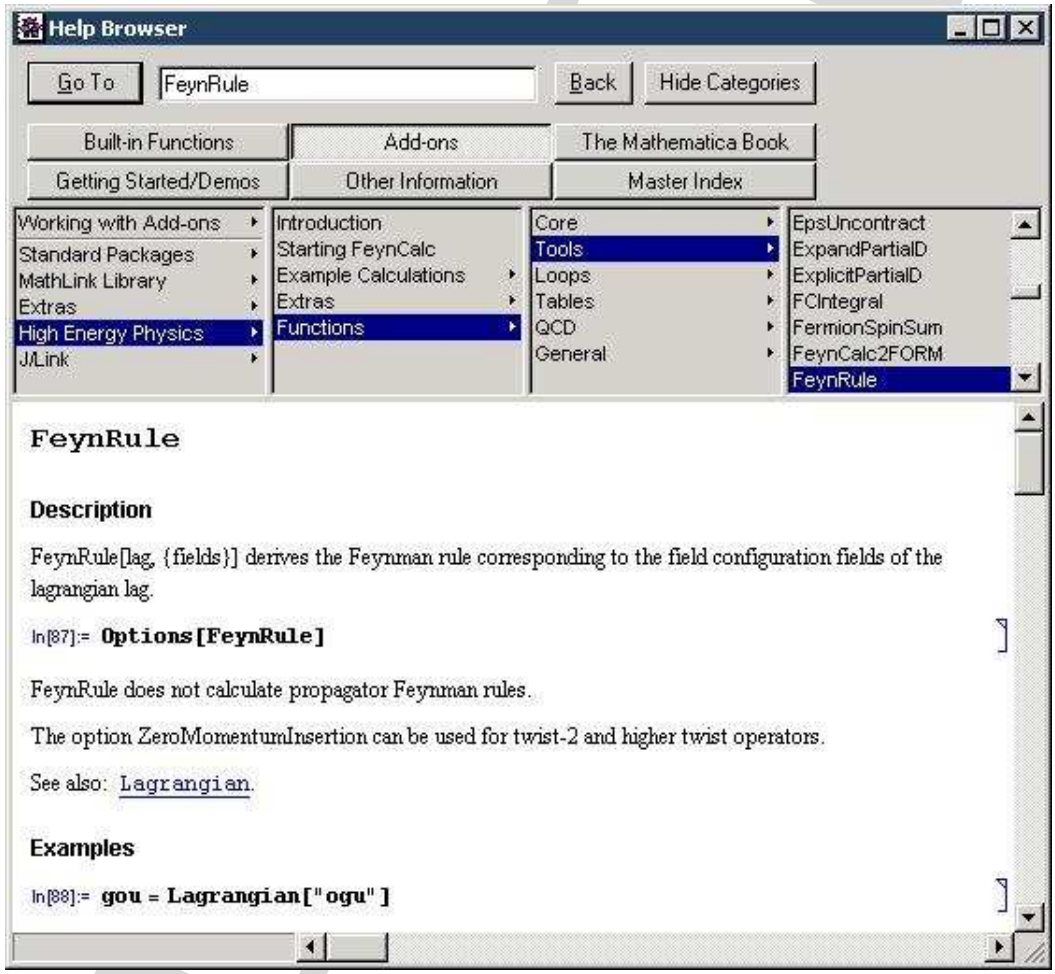

Figure 1: The help system of FeynCalc.

4 Elementary Calculations

You can use FeynCalc for basic calculations like Lorentz algebra and Dirac and color trace evaluations. This chapter contains simple examples of such calculations.

4.1 Lorentz Algebra

The **Contract** function contracts equal Lorentz indices, if at least one belongs to a metric tensor, a four-vector or a Levi-Civita tensor.

Contract[*expr***]** contract double Lorentz indices in *expr*

The function for contraction of tensors.

In four dimensions $g^{\mu}_{\mu} = 4$.

```
In[62]:= Contract[MetricTensor[μ, μ]]

Out[62]= 4
```

While in D dimensions $g^{\mu}_{\mu} = D$.

```
In[63]:= Contract[MetricTensor[μ, μ, Dimension → D]]

Out[63]= D
```

Contract $g^{\alpha\beta} p^{\beta}$.

```
In[64]:= Contract[MetricTensor[α, β] FourVector[p, β]]

Out[64]= p^α
```

Contract $q^{\alpha} (p-q)^{\alpha}$.

```
In[65]:= Contract[FourVector[q, α] FourVector[p - q, α]]

Out[65]= (p - q)·q
```

Numerical factors are pulled out when calculating with **FourVector**s.

```
In[66]:= FourVector[2 p, μ] FourVector[2 p, μ] // Contract

Out[66]= 4 p^2
```

This contracts $g^{\alpha\beta}\,\gamma^{\alpha}$.

```
In[67]:= Contract[MetricTensor[α, β]
         DiracMatrix[α]]
```

$$\text{Out[67]=}\quad \gamma^{\beta}$$

Contracting $q^{\alpha}\,\gamma^{\alpha}$ yields a Feynman slash.

```
In[68]:= Contract[FourVector[q, al]
         DiracMatrix[al]]
```

$$\text{Out[68]=}\quad \gamma\cdot q$$

Contracting $\varepsilon^{\mu\nu\rho\sigma}\,p^{\sigma}$ gives $\varepsilon^{\mu\nu\rho p}$.

```
In[69]:= Contract[LeviCivita[mu, nu, ro, si]
         FourVector[p, si]]
```

$$\text{Out[69]=}\quad \varepsilon^{\mu\nu\rho p}$$

The contraction of $\varepsilon^{\alpha\nu\rho\sigma}\,\varepsilon^{\beta\nu\rho\sigma}$ yields $-6\,g^{\alpha\beta}$.

```
In[70]:= Contract[LeviCivita[α, ν, ρ, σ]
         LeviCivita[β, ν, ρ, σ], ]
```

$$\text{Out[70]=}\quad -6\,g^{\alpha\beta}$$

option name	*default value*	
EpsContract	**False**	contract Levi-Civita **Eps**
Expanding	**True**	expand the input
Factoring	**False**	factor canonically

Options for **Contract**.

Contracting only $g^{\alpha\sigma}\,p^{\alpha}\,p^{\sigma}$ in $g^{\alpha\sigma}\,p^{\alpha}\,p^{\sigma}\,(q^{\beta}+r^{\beta})\,(p^{\beta}-s^{\beta})$.

```
In[71]:= Contract[MetricTensor[α, σ] *
         FourVector[p, α] FourVector[p, σ] *
         (FourVector[q, β] + FourVector[r, β]) *
         (FourVector[p, β] - FourVector[q, β]),
         Expanding → False]
```

$$\text{Out[71]=}\quad (p^{\beta}-q^{\beta})\,(q^{\beta}+r^{\beta})\,p^{2}$$

FeynCalc uses the transversality condition $(k^\mu \cdot \varepsilon^\mu(k)) = 0$ for polarization vectors.

```
In[72]:= Contract[FourVector[k, μ]
         PolarizationVector[k, μ]]

Out[72]= 0
```

ExpandScalarProduct[*expr***]**	expand scalar products and four-vectors in *expr*
MomentumExpand[*expr***]**	expand **Momentum[a+b+ ...]** in *expr* into **Momentum[a] + Momentum[b] + ...**
MomentumCombine[*expr***]**	invert the operation of **MomentumExpand** and **ExpandScalarProduct**

Functions for expansion and combination of scalar products and four-vectors.

As an example, expand $(a + b) \cdot (c - 2d)$.

```
In[73]:= ExpandScalarProduct[ScalarProduct[a + b,
         c - 2 d]]

Out[73]= a·c − 2 a·d + b·c − 2 b·d
```

Combine again.

```
In[74]:= MomentumCombine[%]
Out[74]= (a + b)·(c − 2d)
```

Consider again $q^\alpha (p - q)^\alpha$. **Contract** expands scalar products.

```
In[75]:= Contract[FourVector[q, α] FourVector[p -
         q, α]]

Out[75]= (p − q)·q
```

This is how you can substitute $q^2 \to 0$ afterwards.

```
In[76]:= % /. ScalarProduct[q, q] → 0

Out[76]= p·q
```

Instead of substituting scalar products at the end of the calculation another possibility is to assign special values for scalar products first. These special values are inserted immediately whenever possible during the calculation.

Set $q^2 = 0$ before a calculation.

```
In[77]:= ScalarProduct[q, q] = 0;
```

Contracting (and expanding) $q^\alpha\,(p-q)^\alpha$ now yields $(p\cdot q)$.

```
In[78]:= Contract[FourVector[q, α] FourVector[p -
           q, α]]

Out[78]= p·q
```

Clear the value of $(p\cdot q)$.

```
In[79]:= DownValues[Pair] = Select[DownValues[Pair],
           FreeQ[#, q]&];
```

This expands $(a+b)^\mu$ to $a^\mu + b^\mu$.

```
In[80]:= ExpandScalarProduct[FourVector[a + b,
           μ]]

Out[80]= a^μ + b^μ
```

4.2 Dirac Algebra

For the manipulation of noncommutative products of Dirac matrices and spinors, a number of functions are provided (see also section 7.1). **DiracEquation** applies the Dirac equation without expanding. **DiracOrder** orders products of Dirac matrices in a canonical way: Basically it is just the implementation of the anticommutator relation $\{\gamma^\mu, \gamma^\nu\} = 2\,g^{\mu\nu}$. **Chisholm** substitutes products of three Dirac matrices or slashes in *expr* using the Chisholm identity.

DiracEquation[*expr***]**	apply the Dirac equation
Chisholm[*expr***]**	apply the Chisholm identity
DiracOrder[*expr***]**	alphabetical ordering of Dirac matrices in *expr*
DiracOrder[*expr***, {***a***, *b*, ...}]**	ordering according to $a, b, \ldots$

Manipulation functions for Dirac matrices and spinors.

All functions take as *expr* any expression with "." as the noncommutative multiplication operator between Dirac matrices or Dirac slashes.

The Dirac equation.

```
In[81]:= (# == DiracEquation[#])&[DiracSlash[p] .
           Spinor[p, m]]

Out[81]= (γ·p).φ(p, m) == m φ(p, m)
```

The Chisholm identity.

```
In[82]:= (# == Chisholm[#])&[DiracMatrix[μ,ν,ρ]]
```

$$\text{Out[82]=}\ \gamma^\mu \gamma^\nu \gamma^\rho == i\,\gamma^{\$MU\$26}\,.\,\gamma^5\,\epsilon^{\mu\mu\rho\$MU\$26} + \gamma^\rho\, g^{\mu\nu} - \gamma^\nu\, g^{\mu\rho} + \gamma^\mu\, g^{\nu\rho}$$

Order the product $\gamma^\beta \gamma^\alpha \to 2\,g^{\alpha\beta} - \gamma^\alpha \gamma^\beta$.

```
In[83]:= DiracOrder[DiracMatrix[β, α]]
```

$$\text{Out[83]=}\ 2\,g^{\alpha\beta} - \gamma^\alpha \gamma^\beta$$

Anticommute back to $\gamma^\beta \gamma^\alpha$.

```
In[84]:= DiracOrder[%, {β, α}]
```

$$\text{Out[84]=}\ \gamma^\beta \gamma^\alpha$$

Simplifications like $\gamma^\mu \gamma^\mu \not{p}\not{p} = 4\,p^2$ are built in.

```
In[85]:= DiracOrder[DiracMatrix[μ, μ],
           DiracSlash[p, p]]
```

$$\text{Out[85]=}\ 4\,p^2$$

$\gamma^\alpha \gamma^\mu \gamma^\alpha = (2 - D)\,\gamma^\mu$ in D dimensions.

```
In[86]:= DiracOrder[DiracMatrix[a, m, a,
           Dimension → D]]
```

$$\text{Out[86]=}\ 2\,\gamma^m - D\,\gamma^m$$

$-\not{p}\not{q}\not{p} = \not{q}\,p^2 - 2\,\not{p}\,(p \cdot q)$.

```
In[87]:= DiracOrder[DiracSlash[-p, q, p]]
```

$$\text{Out[87]=}\ \gamma \cdot q\ \ p^2 - 2\,\gamma \cdot p\ \ p \cdot q$$

DotSimplify expands and reorders noncommutative terms using relations specified by the option **DotSimplifyRelations** or by **Commutator** and/or **AntiCommutator** definitions. Whether noncommutative expansion is done depends on the option **Expanding**. Notice that in the rules of the setting of **DotSimplifyRelations**, **Condition** should not be used and patterns should be avoided on the right-hand sides. Also, the performance of **DotSimplify** scales inversely and very badly with the complexity of **DotSimplifyRelations** and the number of terms of the expression to be simplified.

DotSimplify[*expr***]**	expand and reorder noncommutative terms

Simplification of noncommutative products.

option name	*default value*	
Expanding	**True**	noncommutatively expand the result
DotSimplifyRelations	**{}**	a list of substitution rules of the form **DotSimplifyRelations → {a . b → c, b^2 → 0, ...}**
DotPower	**False**	whether noncommutative powers are represented by successive multiplication or by Power.

Options for **DotSimplify**.

This is a four-dimensional product: $2 b a 2 (d - c)(6 q - 3 p)$. **DotSimplify** pulls common numerical factors out.

```
In[88]:= DiracSlash[2 b, a, 2 (d - c), (6 q - 3
         p)] // DotSimplify
```

$$\text{Out[88]}= -12\,\gamma\cdot b\,\gamma\cdot a\,\gamma\cdot(d - c)\,\gamma\cdot(p - 2\,q)$$

Here is another product. Notice that we are using the shorthand **FeynCalcExternal** form (see section 3.4) for input.

```
In[89]:= GA[μ].(a GS[p] - b GS[q]).GS[q].GA[ν]]
```

$$\text{Out[89]}= \gamma^{\mu}.(a\ \gamma\cdot p - b\ \gamma\cdot q).(\gamma\cdot q).\gamma^{\nu}$$

With **Expanding -> True** sums are distributed over. This also causes symbols not known as being noncommutative (see section 3.3) to be pulled out.

```
In[90]:= DotSimplify[%, Expanding -> True]
```

$$\text{Out[90]}= a\,\gamma^{\mu}.(\gamma\cdot p)(\gamma\cdot q).\gamma^{\nu} - b\,\gamma^{\mu}(\gamma\cdot q).(\gamma\cdot q).\gamma^{\nu}$$

With **DotPower -> True** dot products of identical objects are replaced with powers.

```
In[91]:= DotSimplify[%, DotPower -> True]
```

$$\text{Out[91]}= a\,\gamma^{\mu}.(\gamma\cdot p)(\gamma\cdot q).\gamma^{\nu} - b\,\gamma^{\mu}(\gamma\cdot q)^2.\gamma^{\nu}$$

One may specify relations to be included in the simplification process.

```
In[92]:= DotSimplify[%, DotPower -> True,
         DotSimplifyRelations -> {GS[q]^2 -> 1}]
```

$$\text{Out[92]}= a\,\gamma^{\mu}.(\gamma\cdot p)(\gamma\cdot q).\gamma^{\nu} - b\,\gamma^{\mu}.\gamma^{\nu}$$

Declare some symbols as noncommuting and define a commutator.

```
In[93]:= DeclareNonCommutative[a, b, c];
         Commutator[a, c] = 1;
```

DotSimplify uses this information.

```
In[94]:= DotSimplify[a . (b - z c) . a]

Out[94]= a.b.a - z(a + c.a.a)
```

Clear definitions.

```
In[95]:= UnDeclareNonCommutative[a, b, c];
         Commutator[a, c] = 0;
```

DiracSimplify[*expr***]**	contract all Lorentz indices and simplify
DiracSimplify2[*expr***]**	like **DiracSimplify** but leaves any γ^5 untouched. γ^6 and γ^7 are replaced with their definitions
DiracReduce[*expr***]**	reduce Dirac matrices to the standard basis (S, P, V, A, T) using the Chisholm identity
DiracTrick[*expr***]**	contracts gamma matrices with each other and performs several simplifications, but no expansion

Simplification functions for Dirac matrices and spinors.

All functions take as *expr* any expression with "." as the noncommutative multiplication operator between Dirac matrices or Dirac slashes.

DiracBasis	a head wrapped around Dirac structures (and 1) by **DiracReduce**)

A head used by **DiracReduce**.

DiracSimplify contracts Dirac matrices with equal indices, moves γ^5, γ^6 and γ^7 to the right, applies the Dirac equation and expands noncommutative products (see section 3.2). The Dirac matrices in the result of **DiracSimplify** are only ordered in a canonical way if they are between spinors. See below and section 3.2 for the treatment of γ^5 in D dimensions.

This is $\gamma^\mu \gamma^\mu = D$.

```
In[96]:= DiracSimplify[DiracMatrix[μ, μ,
           Dimension → D]]

Out[96]= D
```

Here the Kahane algorithm is used.
$\gamma^\mu \gamma^\nu \gamma^\rho \gamma^\sigma \gamma^\mu = -2 \gamma^\sigma \gamma^\rho \gamma^\nu$.

```
In[97]:= DiracSimplify[DiracMatrix[μ, ν, ρ, σ, μ]]

Out[97]= -2 γ^σ . γ^ρ . γ^ν
```

Kahane also gives this identity:
$\frac{1}{2} \gamma^\mu \gamma^\alpha \gamma^\beta \gamma^\gamma \gamma^\delta \gamma^\mu = \gamma^\gamma \gamma^\beta \gamma^\alpha \gamma^\delta + \gamma^\delta \gamma^\alpha \gamma^\beta \gamma^\gamma$.

```
In[98]:= DiracSimplify[1/2 DiracMatrix[μ, α, β, γ,
           δ, μ]]

Out[98]= γ^γ . γ^β . γ^α . γ^δ + γ^δ . γ^α . γ^β . γ^γ
```

This is
$\not{p}(m-\not{q})\not{p} = \not{q} p^2 + p^2 m - 2\not{p}(p \cdot q)$.

```
In[99]:= DiracSimplify[DiracSlash[p],
           DiracSlash[-q] + m, DiracSlash[p]]

Out[99]= γ · q  p^2 + m p^2 - 2 γ · p  p · q
```

This is $\gamma^5 \gamma^\mu = -\gamma^\mu \gamma^5$.

```
In[100]:= DiracSimplify[DiracMatrix[5],
            DiracMatrix[μ]]

Out[100]= -γ^μ γ^5
```

$\gamma^6 \gamma^\nu \gamma^7 \gamma^\mu = \gamma^\nu \gamma^\mu \gamma^6$.

```
In[101]:= DiracSimplify[DiracMatrix[6, ν, 7, μ]]

Out[101]= γ^ν γ^μ γ^6
```

This is $(\not{p} - m)\, u(p) = 0$.

```
In[102]:= DiracSimplify[(DiracSlash[p] - m) .
            SpinorU[p, m]]

Out[102]= 0
```

Here is the Dirac equation for $v(p)$: $(\not{p} + m)\, v(p) = 0$.

```
In[103]:= DiracSimplify[(DiracSlash[p] + m) .
            SpinorV[p, m]]

Out[103]= 0
```

For the conjugate spinor:
$\overline{u}(\not{p} - m) = 0$.

```
In[104]:= DiracSimplify[SpinorUBar[p, m] .
            (DiracSlash[p] - m)]

Out[104]= 0
```

This is $\overline{v}\,\not{q}\,(\not{p}-m) = 2\,\overline{v}\,p\cdot q$.

```
In[105]:= DiracSimplify[SpinorVBar[p, m] .
            DiracSlash[q] .  (DiracSlash[p] - m)]

Out[105]= 2  p·q  φ(−p, m)
```

Also more complicated structures are simplifi ed; for example, $\overline{v}(p)\,\not{q}\,\not{p}\,u(q) = \overline{v}(p)\;u(p)\,[\,2\,(p\cdot q) + m\,M\,]$.

```
In[106]:= DiracSimplify[SpinorVBar[p, m] .
            DiracSlash[q, p] . SpinorU[q, M]]

Out[106]= φ(−p, m) . φ(q, M) (m M + 2 p·q)
```

The behaviour of **DiracSimplify** may be tuned with the setting of various options.

option name	*default value*	
DiracCanonical	**False**	use **DiracOrder** internally
DiracSigmaExplicit	**True**	substitute the explicit representation of σ (also a function)
DiracSimpCombine	**False**	try merging **DiracGamma**'s in **DiracGamma[.. + .. +]**'s
DiracSubstitute67	**False**	substitute the explicit representation of γ^6 and γ^7
Expanding	**True**	when set to **False** only a limited set of simplification rules are used
Factoring	**False**	factor canonically
InsideDiracTrace	**False**	assume the expression being simplified is inside a Dirac trace

Options for **DiracSimplify**.

DiracReduce reduces all four-dimensional Dirac matrices to the standard basis (S,P,V,A,T) using the Chisholm identity. In the result the basic Dirac structures are wrapped with a head **DiracBasis**. I.e., S corresponds to **DiracBasis[1]**, P : **DiracBasis[DiracMatrix[5]]**, V: **DiracBasis[DiracMatrix[μ]]**, A: **DiracBasis[DiracMatrix[μ, 5]]**, T: **DiracBasis[DiracSigma[DiracMatrix[μ, ν]]]**. By default **DiracBasis** is substituted with **Identity**. Notice that the result of **DiracReduce** is given in **FeynCalcExternal** notation, i.e., evtl. you may want to use **FeynCalcInternal** on the result.

option name	*default value*	
Factoring	**False**	factor canonically
FinalSubstitutions	**{}**	substitutions done at the end of the calculation

Options for **DiracReduce**.

Reducing a product of two Dirac matrices to a standard basis.

```
In[107]:= DiracReduce[DiracMatrix[μ, ν]]
```

$$\text{Out[107]}= g^{\mu\nu} - \sigma^{\mu\nu}$$

Reducing a product of three Dirac matrices to a standard basis.

```
In[108]:= DiracReduce[DiracMatrix[μ, ν, ρ]]
```

$$\text{Out[108]}= i\gamma^{\text{Mu(1)}}.\gamma^5\epsilon^{\mu\nu\rho\text{Mu(1)}} + \gamma^\rho g^{\mu\nu} - \gamma^\nu g^{\mu\rho} + \gamma^\mu g^{\nu\rho}$$

Reducing a product of four Dirac matrices to a standard basis.

```
In[109]:= DiracReduce[DiracMatrix[μ, ν, ρ, σ]]
```

$$\text{Out[109]}= -i\gamma^5\epsilon^{\mu\nu\rho\sigma} - i\sigma^{\rho\sigma}g^{\mu\nu} + i\sigma^{\nu\sigma}g^{\mu\rho} - i\sigma^{\nu\rho}g^{\mu\sigma} - i\sigma^{\mu\sigma}g^{\nu\rho} + g^{\mu\sigma}g^{\nu\rho} + i\sigma^{\mu\rho}g^{\nu\sigma} - g^{\mu\rho}g^{\nu\sigma} - i\sigma^{\mu\nu}g^{\rho\sigma} + g^{\mu\nu}g^{\rho\sigma}$$

Contract the square of the this.

```
In[110]:= Contract[DiracSimplify[% . %]]
```

$$\text{Out[110]}= -128$$

Calc does the full job.

```
In[111]:= Calc[%% . %%]
```

$$\text{Out[111]}= -128$$

Calc[*expr***]**	applies **DotSimplify**, **DiracSimplify**, **EpsEvaluate**, **Contract** and other functions to *expr*, trying to reduce to the simplest form

The highest-level FeynCalc simplification function.

As mentioned in section 3.2, basic features of the Breitenlohner-Maison scheme [14] (for a short explanation of the Breitenlohner-Maison symbols like $\hat{\gamma}^\mu$, see e.g. ref. 15) are implemented. Below are given some simple illustrations.

Setting the Breitenlohner-Maison γ^5 scheme.

```
In[112]:= $BreitMaison = True;
```

Entering $\gamma^5\,\gamma^\mu$, with γ^μ D-dimensional.

```
In[113]:= DiracMatrix[5] . DiracMatrix[μ,
          Dimension →D]

Out[113]= γ5 . γμ
```

Now only the 4-dimensional part of γ^μ anticommutes, while the $D-4$ dimensional part $2\,\hat{\gamma}^\mu$ commutes.

```
In[114]:= DiracSimplify[%]

Out[114]= 2 γ̂μ . γ5 − γμ . γ5
```

Project out the positive chirality part of γ^μ.

```
In[115]:= DiracMatrix[6] . DiracMatrix[μ,
          Dimension→D]

Out[115]= γ6 . γμ
```

The expression is expanded and γ^5 is moved to the right.

```
In[116]:= DiracSimplify[%]

Out[116]= γμ/2 + γ̂μ . γ5 - (γμ . γ5)/2
```

Go back to naive γ^5 scheme (for some operations a kernel restart is necessary).

```
In[117]:= $BreitMaison = True;
```

4.3 Dirac Traces

The function **DiracTrace** takes as *expr* either subsequent **DiracMatrix** or **DiracSlash** separated by "," or any expression with "." as noncommutative multiplication operator.

The default of **DiracTrace** is not to evaluate Dirac traces directly. For direct calculation the function **Tr** can be used.

DiracTrace[*expr***]**	head of a Dirac trace
Tr[*expr***]**	calculate the trace directly

Two Dirac trace functions.

$\mathrm{tr}(\gamma^\alpha \gamma^\beta) = 4 g^{\alpha\beta}$.

```
In[118]:= Tr[DiracMatrix[α, β]]

Out[118]= 4 g^(α β)
```

$\mathrm{tr}(\not{a}\not{b}\not{c}\not{d}) = 4\{(a\cdot d)(b\cdot c) - (a\cdot c)(b\cdot d) + (a\cdot b)(c\cdot d)\}$.

```
In[119]:= Tr[DiracSlash[a, b, c, d]]

Out[119]= 4 (a·d b·c - a·c b·d + a·b c·d)
```

$\mathrm{tr}(\gamma^a \gamma^b \gamma^c \gamma^d \gamma^5) = -4 i \varepsilon^{a b c d}$.

```
In[120]:= Tr[DiracMatrix[α, β, γ, δ, 5]]

Out[120]= -4 I ε^(α β γ δ)
```

You may include metric tensors or four-vectors, for example, $\mathrm{tr}(\frac{1}{4} g^{\alpha\beta} \gamma^\mu \gamma^\alpha p^\mu) = p^\beta$.

```
In[121]:= Tr[MetricTensor[α, β]/4
            DiracMatrix[μ].DiracMatrix[α]
            FourVector[p, μ]] // Contract

Out[121]= p^β
```

If you want to do more complicated traces it is often convenient to introduce your own abbreviations. The following examples, some of which verify results given in [19], show how to do this.

Consider a trace corresponding to the square of the s-channel diagram for γe scattering:

$$T_1 = \frac{1}{16}\,\mathrm{tr}[\,(\not{p}' + m)\,\gamma^\alpha\,(\not{p} + \not{k} + m)\,\gamma^\beta\,(\not{p} + m)\,\gamma^\beta\,(\not{p} + \not{k} + m)\,\gamma^\alpha\,]$$

Set the abbreviations for Dirac matrices and slashes here.

```
In[122]:= (pps = DiracSlash[p'];
           ps = DiracSlash[p];
           ks = DiracSlash[k];
           a = DiracMatrix[α];
           b = DiracMatrix[β];)
```

This is the input for the trace T_1. The CPU time needed for the calculation is of the order of seconds.

```
In[123]:= Tr[(pps + m).a.(ps + ks + m).b.(ps +
          m).b.(ps + ks + m).a/16]//Expand
```

$$\text{Out[123]}= 4\,m^4 + 4\,k^2\,m^2 + 4\,k\cdot p\,m^2 - 4\,k\cdot p'\,m^2 - 3\,p\cdot p'\,m^2 + 2\,k\cdot p\,k\cdot p' + 2\,k\cdot p'\,p^2 - k^2\,p\cdot p' + p^2\,p\cdot p'$$

Clear symbols used.

```
In[124]:= Clear[pps, ps, ks, a, b];
```

Another nontrivial example is a D-dimensional trace involving 14 Dirac matrices:

$$T_2 = \mathrm{tr}(\gamma^\beta\,\gamma^\alpha\,\not{p}_1\,\not{p}_2\,\gamma^\nu\,\gamma^\beta\,\not{p}_2\,\not{p}_3\,\gamma^\alpha\,\not{p}_1\,\gamma^\nu\,\not{p}_3\,\not{p}_1\,\not{p}_2)$$

This defines abbreviations for trace T_2. a, b, n denote $\gamma^\alpha, \gamma^\beta, \gamma^\nu$ in D dimensions. The last command sets $ps1 = \not{p}_1$, $ps2 = \not{p}_2$, $ps3 = \not{p}_3$.

```
In[125]:= a = DiracMatrix[α, Dimension -> D];
          b = DiracMatrix[β, Dimension -> D];
          n = DiracMatrix[ν, Dimension -> D];
          {ps1, ps2, ps3} = Map[DiracSlash, {p1,
          p2, p3}];
```

Here is the input for trace T_2. The result is again collected with respect to scalar products.

```
In[126]:= Tr[b.a.ps1.ps2.n.b.ps2.ps3.a.ps1.n.ps3.ps1.ps2,
          PairCollect → True]
```

$$\begin{aligned}\text{Out[126]}= {} & 4\,((-288 + 224\,D - 56\,D^2 + 4\,D^3)\,p_1\cdot p_2\,p_1\cdot p_3{}^2\,p_2^2 + \\ & (256 - 128\,D + 16\,D^2)\,p_1\cdot p_2{}^2\,p_1\cdot p_3\,p_2\cdot p_3 + (112 - \\ & 104\,D + 28\,D^2 - 2\,D^3)\,p_1^2\,p_1\cdot p_3\,p_2^2\,p_2\cdot p_3 + (-128 + 64\,D - \\ & 8\,D^2)\,p_1^2\,p_1\cdot p_2\,p_2\cdot p_3{}^2 + (-128 + 64\,D - 8\,D^2)\,p_1\cdot p_2{}^3\,p_3^2 + \\ & (168 - 104\,D + 20\,D^2 - D^3)\,p_1^2\,p_1\cdot p_2\,p_2\cdot p_2\,p_3^2)\end{aligned}$$

This calculates T_2 in four dimensions. Since the "." is used, the replacement `D → 4` applies to all Dirac matrices. The time needed would be twice as much without calculating the D-dimensional case before.

```
In[127]:= Tr[b.a.ps1.ps2.n.b.ps2.ps3.a.ps1.n.ps3.ps1.ps2
          /. D →4, PairCollect → True]
```

$$\text{Out[127]}= 4\,(-32\,p1\cdot p2\,p2^2\,p1\cdot p3^2 + 16\,p1^2\,p2^2\,p2\cdot p3\,p1\cdot p3 + 8\,p1^2\,p1\cdot p2\,p2^2\,p3^2)$$

Clear symbols used.

```
In[128]:= Clear[a, b, n, ps1, ps2, ps3];
```

Sometimes you do not want a trace to be evaluated immediately. Here you get the input $\mathrm{tr}(\gamma^\alpha\,\gamma^\beta\,\gamma^\rho\,\gamma^\sigma)$ back (typeset).

```
In[129]:= DiracTrace[DiracMatrix[α, β, ρ, σ]]
```

$$\text{Out[129]}= \mathrm{tr}(\gamma^\alpha\,\gamma^\beta\,\gamma^\rho\,\gamma^\sigma)$$

You may then contract, e.g., with $g^{\alpha\beta}$.

```
In[130]:= Contract[% MetricTensor[α, β]]
```

$$\text{Out[130]}= \mathrm{tr}(\gamma^\beta\,\gamma^\beta\,\gamma^\rho\,\gamma^\sigma)$$

This evaluates the Dirac trace.

```
In[131]:= % /.  DiracTrace → Tr

Out[131]=  16 g^{ρσ}
```

option name	*default value*	
DiracTraceEvaluate	**False**	evaluate the trace
LeviCivitaSign	**-1**	which sign convention to use in the result of tr($\gamma^a \gamma^b \gamma^c \gamma^d \gamma^5$). The default gives $(-1) 4\, i\, \varepsilon^{a\,b\,c\,d}$
Factoring	**False**	factor canonically
Mandelstam	**{}**	utilize the Mandelstam relation
PairCollect	**True**	collect **Pair**s
Schouten	**0**	maximum number of terms on which to apply the Schouten identity
TraceOfOne	**0**	the trace of an identity matrix
FeynCalcExternal	**0**	give output in **FeynCalcExternal** form (see section 3.4)
EpsContract	**False**	contract Levi-Civita **Eps**

Options for **DiracTrace**.

Tr takes the options of **DiracTrace**, but the default setting of *DiracTraceEvaluate* is **True**. Additionally, **Tr** takes the two options **SUNTrace** and **SUNNToCACF**, which control if and how SU(N) traces are evaluated. This is elaborated upon in section 4.4.

The option **PairCollect** determines whether the resulting polynomial is collected with respect to metric tensors, four-vectors and scalar products. In the internal representation these three objects have the same head **Pair**, hence the name **PairCollect**.

For $2 \to 2$ processes the traces are often expressed in terms of Mandelstam variables. In order to replace these for the scalar products you can use **SetMandelstam**.

SetMandelstam[s**,** t**,** u**,** p_1**,** p_2**,** p_3**,** p_4**,** m_1**,** m_2**,** m_3**,** m_4**]**	define scalar products in terms of Mandelstam variables and put the p_i on-shell

A function for introducing Mandelstam variables.

Assuming all p_i incoming, i.e., $p_1 + p_2 + p_3 + p_4 = 0$, the Mandelstam variables are defined by

$$s = (p_1 + p_2)^2, t = (p_1 + p_3)^2, u = (p_1 + p_4)^2.$$

Using these three equations and the on-shell conditions, $p_i^2 = m_i^2$, **SetMandelstam** sets the 10 possible scalar products $(p_i \cdot p_j)$ in terms of s, t, u and m_i^2.
For calculation of traces the Mandelstam relation

$$s + t + u = m_1^2 + m_2^2 + m_3^2 + m_4^2$$

can often be used to get a compact result. If you set the option

```
Mandelstam → {s, t, u, m1^2 + m2^2 + m3^2 + m4^2}
```

FeynCalc tries to figure out the best choice of s, t or u in each factor of the result.
As an example for calculating a trace in terms of Mandelstam variables, consider the following squared amplitude from the process $gg \to t\bar{t}$, with $\Sigma_1^{\alpha\rho}$ and $\Sigma_2^{\beta\rho}$ as polarization sums for the gluons.

$$\begin{aligned}
T_3 &= \mathrm{tr}(\gamma^\sigma (\not{k}_1 - \not{p}_1 - m_t)\gamma^\rho (\not{p}_1 + m_t)(\not{p}_2 - m_t))\, p_1^\alpha\, p_2^\beta\, \Sigma_1^{\alpha\rho}\, \Sigma_2^{\beta\sigma}, \\
\Sigma_1^{\alpha\rho} &= -g^{\alpha\rho} + \tfrac{4}{(u-t)^2}(4m_t^2 - s)\, k_1^\alpha\, k_1^\rho + \tfrac{2}{u-t}[k_1^\rho (p_1 - p_2)^\alpha + k_1^\alpha (p_1 - p_2)^\rho] \\
\Sigma_2^{\beta\sigma} &= -g^{\beta\sigma} + \tfrac{4}{(t-u)^2}(4m_t^2 - s)\, k_2^\beta\, k_2^\sigma + \tfrac{2}{t-u}[k_2^\sigma (p_1 - k_2)^\beta + k_2^\beta (p_1 - p_2)^\sigma]
\end{aligned}$$

Set up s,t,u for $gg \to t\bar{t}$, with k_1, k_2 as gluon and p_1, p_2 as fermion momenta. Again abbreviations with capital letters are introduced for the Dirac matrices and slashes. *polsum*1 and *polsum*2 are the polarization sums for the gluons. As external momentum the choice $n = p_1 - p_2$ has been made.

```
In[132]:= SetMandelstam[s,t,u, k1,k2,-p1,-p2,
          0,0,m,m];
          {ks1, ps1, ps2} = Map[ DiracSlash, {k1,
          p1, p2} ];
          {si, ro} = Map[ DiracMatrix,{σ, ρ} ];
          polsum1 = PolarizationSum[α, ρ, k1, p1 -
          p2];
          polsum2 = PolarizationSum[β, σ, k2, p1 -
          p2];
          p1al = FourVector[p1, alpha];
          p2be = FourVector[p2, β];
```

This is a possible input for trace T_3. FeynCalc contracts first all Lorentz indices and then calculates the trace. Since the option **Mandelstam** has been specified, the result is given in a factored form, where in each factor one of s,t or u is eliminated via the Mandelstam relation. Note that a factor $(t-u)$ has been cancelled.

```
In[133]:= Tr[ (polsum1 polsum2 p1al p2be si) .
          (ks1 - ps1 - m) . ro . (ps1 + m) .
          (ps2 - m), Mandelstam → {s, t, u, 2 m^2}
          ]
```

$$\text{Out[133]} = -\frac{2\,m\,s\,(m^4 - t\,u)\,(8\,m^4 - t^2 - u^2 - 6\,t\,u)}{(t-u)^3}$$

An alternative method would be to first calculate the trace without the polarization sums.

```
In[134]:= Tr[si, ks1 - ps1 - m, ro, ps1 + m, ps2 -
          m]
```

$$\text{Out[134]} = (-2\,m\,t + 2\,m\,u)\,g^{\rho\sigma} - 4\,m\,\text{k1}^{\sigma}\,\text{p1}^{\rho} - 4\,m\,\text{k1}^{\rho}\,\text{p1}^{\sigma} + 8\,m\,\text{p1}^{\rho}\,\text{p1}^{\sigma} + 4\,m\,\text{k1}^{\sigma}\,\text{p2}^{\rho} + 4\,m\,\text{k1}^{\rho}\,\text{p2}^{\sigma} - 8\,m\,\text{p1}^{\rho}\,\text{p2}^{\sigma}$$

Then contract the result with the polarization sums, expand the scalar products and use **TrickMandelstam** (see section 7.7) in order to get the Mandelstam variable substitution. This method is faster; but that is not the case for all trace calculations.

```
In[135]:= TrickMandelstam[ExpandScalarProduct[Contract[
          % polsum1 polsum2 p1al p2be]], {s, t, u,
          2 m^2}]
```

$$\text{Out[135]} = -\frac{2\,m\,s\,(m^4 - t\,u)\,(8\,m^4 - t^2 - u^2 - 6\,t\,u)}{(t-u)^3}$$

Clear symbols used.

```
In[136]:= Clear[ks1,ps1,ps2,si,ro,polsum1,polsum2,p1al,p2be];
```

Since Dirac matrices can be given in any dimensions, FeynCalc is also able to calculate traces in $D-4$ dimensions. Defining $T(n) = \text{tr}(\gamma_{\mu_1}\gamma_{\mu_2}...\gamma_{\mu_n}\gamma_{\mu_1}\gamma_{\mu_2}...\gamma_{\mu_n})$ we give a list of timings and results for $T(8)$ to $T(11)$. The trace $T(10)$ is a verification of the result given in [18].

This is a little program defi ning T. The dimension of each particular Dirac matrix is set to $d - 4$. The calculations were done with Mathematica 4.2 under Linux on a 1.8 GHz Pentium 4 box with 256 MB of RAM.

```
In[137]:= T[n_] := T[n] = Block[{gammas, calc},
          gammas = Dot @@ Table[
          DiracMatrix[a[i], Dimension → (d - 4)],
          {i, 1, n} ];
          calc = Timing[ Tr[ gammas . gammas ] //
          Expand ];
          Print["Time = ", calc[[1]] ];
          calc[[2]]];
```

This calculates a trace of 16 matrices.

```
In[138]:= T[8]

Time = 0.58 Second
```

$$\text{Out[138]}= 123469824 - 135962624\,d + 63224832\,d^2 - 16145920\,d^3 + 2461760\,d^4 - 227584\,d^5 + 12320\,d^6 - 352\,d^7 + 4\,d^8$$

Here we have 18.

```
In[139]:= T[9]

Time = 1.6 Second
```

$$\text{Out[139]}= -1879576576 + 2220901376\,d - 1127626752\,d^2 + 321806848\,d^3 - 56625408\,d^4 + 6331584\,d^5 - 446208\,d^6 + 18912\,d^7 - 432\,d^8 + 4\,d^9$$

The trace of 20 Dirac matrices.

```
In[140]:= T[10]

Time = 5.88 Second
```

$$\text{Out[140]}= -31023169536 + 38971179008\,d - 21328977920\,d^2 + 6679521280\,d^3 - 1320732160\,d^4 + 171464832\,d^5 - 14710080\,d^6 + 816960\,d^7 - 27840\,d^8 + 520\,d^9 - 4\,d^{10}$$

With 22 Dirac matrices it gets slow.

```
In[141]:= T[11]

Time = 23.15 Second
```

$$\text{Out[141]}= 551768735744 - 731506905088\,d + 427299186688\,d^2 - 144858475520\,d^3 + 31576821760\,d^4 - 4629805312\,d^5 + 463655808\,d^6 - 31521600\,d^7 + 1415040\,d^8 - 39600\,d^9 + 616\,d^{10} - 4\,d^{11}$$

4.4 SU(N) Traces and Algebra

The functions for calculations with SU(N) matrices and the corresponding structure constants were developed for calculations in N-dimensional color space. More specialized functions developed for calculations in 2- and

3-dimensional flavor space (Pauli and Gell-Mann matrices) are proveded by the subpackage PHI. These are described in the PHI user's guide ???.

SUNTrace[*expr***]**	calculate the trace of SU(N) matrices

Trace calculation of SU(N) matrices in the fundamental representation.

Like Dirac traces, traces of the SU(N) matrices T_i are calculated algebraically. The matrices are assumed to be in the fundamental representation and traces are given in terms of N.

option name	*default value*	
Explicit	**False**	evaluate the trace

Option for **SUNTrace**.

SUNDeltaContract[*expr***]**	contract **SUNDelta**s
SUNSimplify[*expr***]**	simplify polynomials in the SU(N) Kronecker delta δ_{ij} and structure functions f_{ijk}, d_{ijk} and generating matrices T_i

Functions for simplifying expressions with SU(N) matrices and structure functions.

The result of **SUNSimplify** involves either N or the Casimir invariants C_A and C_F. Which, depends on the setting of the option **SUNNToCACF**.

SUNN	the N of SU(N)
CA	$C_A = N$ in the funcamental representation
CF	$C_F = (N^2 - 1)/(2N)$ in the funcamental representation

Casimir invariants of SU(N).

option name	*default value*	
SUNTrace	**False**	if set to False, then any SUNT-matrices are taken out of **DiracTrace[...]**; otherwise a color-trace is taken (by **SUNTrace**) before taking the SU(N) objects in front of **DiracTrace[...]**
Explicit	**False**	express structure functions (f_{ijk}, d_{ijk}) in terms of traces of generator matrices (T_i)
Factoring	**False**	factor the result
SUNIndexRename	**True**	rename contracted SU(N) indices systematically
SUNFJacobi	**False**	use the Jacobi identity
SUNNToCACF	**True**	express the result in terms of the Casimir invariants C_A and C_F instead of N
Expanding	**False**	do noncommutative expansion

Options for **SUNSimplify**.

To evaluate f_{abc} use the option **Explicit → True**.
The input form of the trace in the output is **Tr**.

```
In[142]:= SUNF[a, b, c, Explicit → True]
```

$$\text{Out[142]=}\ 2i\,(\mathrm{tr}(T_a\,T_c\,T_b) - \mathrm{tr}(T_a\,T_b\,T_c))$$

$\mathrm{tr}(T_a\,T_b) = \delta_{ab}/2$.

```
In[143]:= SUNTrace[SUNT[a] .  SUNT[b]]]
```

$$\text{Out[143]=}\quad \frac{\delta_{ab}}{2}$$

$\mathrm{tr}(3\,T_a\,T_b\,T_a\,T_b) = 1/(4N) - N/4$.

```
In[144]:= SUNTrace[SUNT /@ (a.b.a.b)]
```

$$\text{Out[144]=}\quad \frac{1}{4N} - \frac{N}{4}$$

$\mathrm{tr}(T_a\,T_b\,T_c)$ stays as it is.

```
In[145]:= SUNTrace[SUNT /@ (a.b.c)]
```

$$\text{Out[145]=}\quad \mathrm{tr}(T_a\,T_b\,T_c)$$

$\mathrm{tr}(T_a\,T_b\,T_c\;f_{abc}) = i\,C_\mathrm{A}^2\,C_\mathrm{F}/2$.

```
In[146]:= SUNTrace[SUNF[a, b, c] SUNT /@ (a.b.c)]
          // SUNSimplify
```

$$\text{Out[146]=}\quad \frac{1}{2}\,i\,C_\mathrm{A}^2\,C_\mathrm{F}$$

$\mathrm{tr}(f_{ars}\,f_{brs}) = C_\mathrm{A}\,f_{ars}\,f_{brs} = C_\mathrm{A}^2\,\delta_{ab}$.

```
In[147]:= SUNF[a, r, s] SUNF[b, r, s] //
          SUNSimplify
```

$$\text{Out[147]=}\quad C_\mathrm{A}^2\,\delta_{ab}$$

The Jacobi identity: $f_{abr}\,f_{rcs} + f_{bcr}\,f_{ras} + f_{car}\,f_{rbs} = 0$.

```
In[148]:= SUNSimplify[SUNF[a,b,r] SUNF[r,c,s] +
          SUNF[b,c,r] SUNF[r,a,s] + SUNF[c,a,r]
          SUNF[r,b,s], SUNFJacobi → True]
```

$$\text{Out[148]=}\quad 0$$

$T_a\,T_b\,T_a = -\frac{1}{2}\,(C_\mathrm{A} - 2C_\mathrm{F})\,T_b$.

```
In[149]:= SUNT[a, b, a] // SUNSimplify
```

$$\text{Out[149]=}\quad -\frac{1}{2}\,(C_\mathrm{A} - 2C_\mathrm{F})\,T_b$$

This is $f_{abc}\,T_b\,T_c = i\,C_\mathrm{A}\,T_a/2$.

```
In[150]:= SUNF[c, a, b] SUNT[b, c] // SUNSimplify
```

$$\text{Out[150]=}\quad \frac{1}{2}\,i\,C_\mathrm{A}\,T_a$$

4.5 Green's functions

Quantum fields (see section 2.7) can be combined in polynomials to form lagrangians. From such lagrangians, the Green's function or Feynman rule of an interaction vertex can be found by calculating functional derivatives with respect to the fields of the vertex. As a very simple example, consider the vertex obtained from the following term of the QED lagrangian:

Mathematica (FeynCalc) notation for $eA_\mu\bar{\psi}\gamma^\mu\psi$.

```
In[151]:= e QuantumField[γ, {μ}].QuantumField[ψ̄].
          DiracMatrix[{μ}].QuantumField[ψ]
```

$$\text{Out[151]=}\quad eA_\mu\bar{\psi}\gamma^\mu\psi$$

Calculation of the $\psi\bar{\psi}\gamma_{\mu_3}$ Feynman rule.

```
In[152]:= FeynRule[%,
          {QuantumField[ψ][p1],
          QuantumField[ψ̄][p2],
          QuantumField[γ, {μ3}][p3]}]
```

$$\text{Out[152]=}\quad ie\gamma^{\mu 3}$$

Notice that **FeynRule** does not write out the momentum conserving $\delta(p_1 + p_2 + p_3)$. As we shall see later, this is anyway enforced if the vertex is used for Feynman diagram calculations.

Besides calculating the functional derivative, **FeynRule** does some simplification on the result and transforms to momentum space. To calculate the functional derivative **FeynRule** uses a lower level function: **FunctionalD**. Contrary to **FeynRule** **FunctionalD** does do any cleaning up on the result and therefore may be faster but return longer expressions.

By default **FunctionalD** operates in position space. However, no explicit space-time symbols should be given and none will be returned. Thus, a few conventions are used: Instead of the usual $\delta\phi(x)/\delta\phi(y) = \delta^{(D)}(x-y)$ the arguments and the δ function are omitted, i.e., for simplicity $\delta\phi/\delta\phi$ is taken to be 1. Similarly, instead of the usual $\delta\partial_\mu\phi(x)/\delta\phi(y) = \partial_\mu\delta^{(D)}(x-y)$ the arguments are omitted, and the ∂_μ operator is specified by default to be an integration by parts operator, i.e., the right-hand side will be just $-\partial_\mu$ or, more precisely (by default), $-\overrightarrow{\partial}_\mu$.

If the **QuantumField**s of the first argument (e.g. a lagrangian) of **FunctionalD** are given an extra argument (e.g. **QuantumField[...][p]**), the argument is assumed to be a momentum and transformation to momentum space is done.

FunctionalD[*expr*, *fi 1***[***p1***], ...]**	calculates the functional derivative of *expr* with respect to quantum fields *fi 1*, *fi 2*, ⋯ and does the Fourier transform to field momenta *p1*, *p2*, ⋯
FunctionalD[*expr*, *fi 1*, *fi 2*, ...]**	calculates the functional derivative of *expr* and does partial integration but omits the x-space delta functions.
FeynRule[*lag*, *fi 1***[***p1***]**, *fi 2***[***p2***], ...]**	calculates the Feynman rule of *lag* with respect to fields *fi 1*, *fi 2*, ⋯ and momenta *p1*, *p2*, ⋯

Calculation of Green's functions.

Define a two-gluon product **aa** with contracted indices.

```
In[153]:= aa = QuantumField[PartialD[LorentzIndex[β]],
           GaugeField, LorentzIndex[α],
           SUNIndex[a]].QuantumField[
           PartialD[LorentzIndex[β]], GaugeField,
           LorentzIndex[α], SUNIndex[a]]
```

$$\text{Out[153]=}\ \partial_\beta A^a_\alpha\, \partial_\beta A^a_\alpha$$

Calculate the functional derivative.

```
In[154]:= dd = FunctionalD[aa,
           QuantumField[GaugeField, {μ1}, {i1}],
           QuantumField[GaugeField, {μ2}, {i2}]]
```

$$\text{Out[154]=}\ -2\overrightarrow{\partial}_\beta.\overrightarrow{\partial}_\beta g^{\mu 1\mu 2}\delta_{\text{i1i2}}$$

See the internal representation. Notice the symbol **RightPartialD**.

```
In[155]:= dd // StandardForm
```

```
Out[155]= -2 RightPartialD[LorentzIndex[β]] .
          RightPartialD[LorentzIndex[β]]
          Pair[LorentzIndex[μ1], LorentzIndex[μ2]]
          SUNDelta[SUNIndex[i1], SUNIndex[i2]]
```

We can apply **dd** to some other field ψ.

```
In[156]:= dd .  QuantumField[ψ, {μ1}, {i1}] //
           ExpandPartialD
```

$$\text{Out[156]=}\ -2g^{\mu 1\mu 2}(\partial_\beta\partial_\beta\psi^{\text{i2}}_{\mu 1})$$

See the internal representation. Notice that applying **ExpandPartialD** causes **RightPartialD** to disappear in favour of **PartialD**.

```
In[157]:= dd // StandardForm

Out[157]= -2 RightPartialD[LorentzIndex[β]] .
          RightPartialD[LorentzIndex[β]]
          Pair[LorentzIndex[μ1], LorentzIndex[μ2]]
          SUNDelta[SUNIndex[i1], SUNIndex[i2]]
```

Calculate the corresponding Feynman rule (in momentum space).

```
In[158]:= FunctionalD[aa, {QuantumField[GaugeField,
          {μ1}, {i1}][p1], QuantumField[GaugeField,
          {μ2}, {i2}][p2]}]

Out[158]= (-ig^(αμ1) p1^β δ_(αi1)) . (-ig^(αμ2) p2^β δ_(αi2)) +
          (-ig^(αμ2) p2^β δ_(αi2)) . (-ig^(αμ1) p1^β δ_(αi1))
```

Clear intermediate variables.

```
In[159]:= Clear[aa];
```

RightPartialD[μ**]**	denotes partial space-time differentiation $\partial/\partial x^\mu$, acting to the right.
LeftPartialD[μ**]**	denotes partial space-time differentiation $\partial/\partial x^\mu$, acting to the left.
LeftRightPartialD[μ**]**	denotes partial space-time differentiation $\partial/\partial x^\mu$, acting to the left and right. **ExplicitPartialD[LeftRightPartialD[**μ**]]** gives **1/2 (RightPartialD[**μ**] - LeftPartialD[**μ**])**
LeftRightPartialD2[μ**]**	denotes partial space-time differentiation $\partial/\partial x^\mu$, acting to the left and right. **ExplicitPartialD[LeftRightPartialD[**μ**]]** gives **(RightPartialD[**μ**] + LeftPartialD[**μ**])**
ExplicitPartialD[*exp***]**	inserts in *exp* the definitions for **LeftRightPartialD** and **LeftRightPartialD2**.
ExpandPartialD[*exp***]**	expands **DOT** products of **RightPartialD**'s and/or **LeftPartialD**'s with **QuantumField**'s in *exp* using the Leibniz rule.

Partial space-time derivatives.

As another example consider the strong QCD 4-gluon vertex. The relevant part of the QCD lagrangian is already known by FeynCalc.

The gluon self-interaction part of the QCD lagrangian .

```
In[160]:= Lagrangian["QCD"]
```

$$\text{Out[160]= } -\frac{1}{4} F^a_{\alpha\beta} . F^a_{\alpha\beta}$$

We can write out the fi eld strength tensors.

```
In[161]:= Lagrangian["QCD"] /.
          FieldStrength[x__] :→
          FieldStrength[x, Explicit → True] //
          DotExpand
```

$$\text{Out[161]= } \frac{1}{4}\left(- A^{b11}_{\alpha} . A^{c15}_{\beta} . A^{b12}_{\alpha} . A^{c16}_{\beta} \, f_{ab11c15} \, f_{ab12c16} \, g_s^2 - A^{b11}_{\alpha} . A^{c15}_{\beta} . \partial_{\alpha} A^a_{\beta} \, f_{ab11c15} \, g_s + A^{b11}_{\alpha} . A^{c15}_{\beta} . \partial_{\beta} A^a_{\alpha} \, f_{ab11c15} \, g_s - \partial_{\alpha} A^a_{\beta} . A^{b12}_{\alpha} . A^{c16}_{\beta} \, f_{ab12c16} \, g_s + \partial_{\beta} A^a_{\alpha} . A^{b12}_{\alpha} . A^{c16}_{\beta} \, f_{ab12c16} \, g_s - \partial_{\alpha} A^a_{\beta} . \partial_{\alpha} A^a_{\beta} + \partial_{\alpha} A^a_{\beta} . \partial_{\beta} A^a_{\alpha} + \partial_{\beta} A^a_{\alpha} . \partial_{\alpha} A^a_{\beta} - \partial_{\beta} A^a_{\alpha} . \partial_{\beta} A^a_{\alpha} \right)$$

Calculate the 4-gluon Feynman rule.

```
In[162]:= FeynRule[Lagrangian["QCD"], {
          QuantumField[GaugeField, {μ1},
          {i1}][p1],
          QuantumField[GaugeField, {μ2},
          {i2}][p2],
          QuantumField[GaugeField, {μ3},
          {i3}][p3],
          QuantumField[GaugeField, {μ4},
          {i4}][p4]}]
```

$$\text{Out[162]= } i \left(g^{\mu 1 \mu 3} g^{\mu 2 \mu 4} - g^{\mu 1 \mu 2} g^{\mu 3 \mu 4}\right) f_{i1i4si3} \, f_{i2i3si3} \, g_s^2 + i \left(g^{\mu 1 \mu 4} g^{\mu 2 \mu 3} - g^{\mu 1 \mu 2} g^{\mu 3 \mu 4}\right) f_{i1i3si3} \, f_{i2i4si3} \, g_s^2 + i \left(g^{\mu 1 \mu 4} g^{\mu 2 \mu 3} - g^{\mu 1 \mu 3} g^{\mu 2 \mu 4}\right) f_{i1i2si3} \, f_{i3i4si3} \, g_s^2$$

The function **Lagrangian** reads a database of lagrangians and returns the one corresponding to the name (a string) given as argument. Currently the following names are known by **Lagrangian**:

- **Lagrangian["oqu"]** gives the unpolarized OPE quark operator.
- **Lagrangian["oqp"]** gives the polarized quark OPE operator.
- **Lagrangian["ogu"]** gives the unpolarized gluon OPE operator.
- **Lagrangian["ogp"]** gives the polarized gluon OPE operator.
- **Lagrangian["ogd"]** gives the sigma-term part of the QCD lagrangian.
- **Lagrangian["QCD"]** gives the gluon self interaction part of the QCD lagrangian.

More can be added easily (as done e.g. by the optional subpackage PHI).

`Lagrangian["`*name***`"]`** returns a lagrangian *name* in FeynCalc notation.

A database of lagrangians.

5 One-Loop Calculations

In this section is described the capabilities of FeynCalc to reduce one-loop Feynman diagrams. If is also discussed how to provide input for FeynCalc and how to further process the output of FeynCalc.

The methods and conventions implemented in FeynCalc for the evaluation of one-loop diagrams are described in [9] and [10]. The usual Passarino-Veltman scheme for the one-loop integrals is adapted to a large extent [9]. The coefficient functions of the tensor integrals are defined similar to [9], except that the Passarino-Veltman integrals take internal masses squared as arguments.

FeynCalc can reduce all n-point integrals with $n \leq 4$ to scalar integrals A_0, B_0, C_0 and D_0.

For the numerical evaluation of scalar integrals, several possibilities exist: 1) The program FormF by M. Veltman. Unfortunately this only runs in CDC and 68000 assembler and is thus not a realistic alternative. 2) The library FF by G.J. van Oldenborgh [12] written in Fortran 77. The library can be put to use with the LoopTools package [13] written by Thomas Hahn, which features a Fortran, C++, and Mathematica interface. Loading the mathematica package will simply cause the functions A_0, B_0, C_0 and D_0 (and E_0 and F_0) to return numerical values when given numerical arguments. This is the recommended procedure for obtaining numerical output from FeynCalc results. A more simple wrapper Fortran program to link FF and FeynCalc is available from Rolf Mertig. 3) The optional subpackage PHI provides some functions for evaluating B_0 and C_0. These are written purely in Mathematica but not very well tested.

5.1 Passarino-Veltman Integrals and Reduction of Coefficient Functions

The scalar integrals A_0, B_0, C_0 and D_0 are represented in FeynCalc as functions with all arguments consisting of scalar products or masses squared. Thus A_0 has one, B_0 three, C_0 six, and D_0 ten arguments. The symmetry properties of the arguments are implemented, i.e., a standard representative of all possible argument permutations of each B_0, C_0 and D_0 is returned. For example, `B0[pp, m2^2, m1^2]` → `B0[pp, m1^2, m2^2]`, where `pp` denotes the scalar product $(p \cdot p) = p^2$.

A0[*m02***]**	$(i\pi^2)^{-1}\int d^Dq(q^2-m_0^2)^{-1}$
B0[*p10***,** *m02***,** *m12***]**	$(i\pi^2)^{-1}\int d^Dq([(q^2-m_0^2][(q+p_1)^2-m_1^2])^{-1}$
DB0[*p10***,** *m02***,** *m12***]**	$\partial B_0(p_1^2,m_0^2,m_1^2)/\partial p_1^2$
C0[*p10***,** *p12***,** *p20***,** *m02***,** *m12***,** *m22***]**	$(i\pi^2)^{-1}\int d^Dq([q^2-m_0^2][(q+p_1)^2-m_1^2][(q+p_2)^2-m_2^2])^{-1}$
D0[*p10***,** *p12***,** *p23***,** *p30***,** *p20***,** *p13***,** *m02***,** *m12***,** *m22***,** *m32***]**	$(i\pi^2)^{-1}\int d^Dq([q^2-m_0^2][(q+p_1)^2-m_1^2][(q+p_2)^2-m_2^2][(q+p_3)^2-m_3^2])^{-1}$

Scalar Passarino-Veltman functions A_0, B_0,B_0', C_0 and D_0.

In FeynCalc and in the mathematical definitions given above, the factor $(2\pi\mu)^{(4-D)}$ with the scaling variable μ is suppressed. The convention for the scalar arguments is $pi0 = p_i^2 \quad pij = (p_i - p_j)^2, \quad mi2 = m_i^2$.

The B_0 function is symmetric in the mass arguments.

```
In[163]:= B0[s, mz2, mw2]
```

$$\text{Out[163]= } B_0(s,\ \text{mw2},\ \text{mz2})$$

Taking the derivative with respect to the first argument yields **DB0**.

```
In[164]:= D[%, s]
```

$$\text{Out[164]= } \text{DB0}(s,\ \text{mw2},\ \text{mz2})$$

The tensor-integral decomposition is automatically done by FeynCalc when calculating one-loop amplitudes, but extra functions are provided to reduce the coefficients of the tensor-integral decomposition.

For fixing the conventions of the coefficient functions the definitions of the tensor-integrals and the decomposition are given below. In general the one-loop tensor integral is

$$T^N_{\mu_1\ldots\mu_P}(p_1,\ldots,p_{N-1},m_0,\ldots,m_{N-1}) = \frac{(2\pi\mu)^{4-D}}{i\pi^2}\int d^Dq\frac{q_{\mu_1}\cdots q_{\mu_P}}{\mathcal{D}_0\mathcal{D}_1\cdots\mathcal{D}_{N-1}}$$

with the denominator factors

$$\mathcal{D}_0 = q^2 - m_0^2 \qquad \mathcal{D}_i = (q+p_i)^2 - m_i^2, \qquad i = 1,\ldots,N-1$$

originating from the propagators in the Feynman diagram. The $i\varepsilon$ part of the denominator factors is suppressed.

The tensor integral decompositions for the integrals that FeynCalc can do are listed below. The coefficient functions $B_i, B_{ij}, C_i, C_{ij}, C_{ijk}, D_i, D_{ij}, D_{ijk}$ and D_{ijkl} are totally symmetric in their indices.

$$
\begin{aligned}
B_\mu &= p_{1\mu} B_1 \\
B_{\mu\nu} &= g_{\mu\nu} B_{00} + p_{1\mu} p_{1\nu} B_{11} \\
C_\mu &= p_{1\mu} C_1 + p_{2\mu} C_2 = \sum_{i=1}^{2} p_{i\mu} C_i \\
C_{\mu\nu} &= g_{\mu\nu} C_{00} + p_{1\mu} p_{1\nu} C_{11} + p_{2\mu} p_{2\nu} C_{22} + (p_{1\mu} p_{2\nu} + p_{2\mu} p_{1\nu}) C_{12} \\
&= g_{\mu\nu} C_{00} + \sum_{i,j=1}^{2} p_{i\mu} p_{j\nu} C_{ij} \\
C_{\mu\nu\rho} &= (g_{\mu\nu} p_{1\rho} + g_{\nu\rho} p_{1\mu} + g_{\mu\rho} p_{1\nu}) C_{001} + (g_{\mu\nu} p_{2\rho} + g_{\nu\rho} p_{2\mu} + g_{\mu\rho} p_{2\nu}) C_{002} \\
&\quad + p_{1\mu} p_{1\nu} p_{1\rho} C_{111} + p_{2\mu} p_{2\nu} p_{2\rho} C_{222} \\
&\quad + (p_{1\mu} p_{1\nu} p_{2\rho} + p_{1\mu} p_{2\nu} p_{1\rho} + p_{2\mu} p_{1\nu} p_{1\rho}) C_{112} \\
&\quad + (p_{2\mu} p_{2\nu} p_{1\rho} + p_{2\mu} p_{1\nu} p_{2\rho} + p_{1\mu} p_{2\nu} p_{2\rho}) C_{122} \\
&= \sum_{i=1}^{2} (g_{\mu\nu} p_{i\rho} + g_{\nu\rho} p_{i\mu} + g_{\mu\rho} p_{i\nu}) C_{00i} + \sum_{i,j,k=1}^{2} p_{i\mu} p_{j\nu} p_{k\rho} C_{ijk} \\
D_\mu &= \sum_{i=1}^{3} p_{i\mu} D_i \\
D_{\mu\nu} &= g_{\mu\nu} D_{00} + \sum_{i,j=1}^{3} p_{i\mu} p_{j\nu} D_{ij} \\
D_{\mu\nu\rho} &= \sum_{i=1}^{3} (g_{\mu\nu} p_{i\rho} + g_{\nu\rho} p_{i\mu} + g_{\mu\rho} p_{i\nu}) D_{00i} + \sum_{i,j,k=1}^{3} p_{i\mu} p_{j\nu} p_{k\rho} D_{ijk} \\
D_{\mu\nu\rho\sigma} &= (g_{\mu\nu} g_{\rho\sigma} + g_{\mu\rho} g_{\nu\sigma} + g_{\mu\sigma} g_{\nu\rho}) D_{0000} \\
&\quad + \sum_{i,j=1}^{3} (g_{\mu\nu} p_{i\rho} p_{j\sigma} + g_{\nu\rho} p_{i\mu} p_{j\sigma} + g_{\mu\rho} p_{i\nu} p_{j\sigma} + g_{\mu\sigma} p_{i\nu} p_{j\rho} + g_{\nu\sigma} p_{i\mu} p_{j\rho} + g_{\rho\sigma} p_{i\mu} p_{j\nu}) D_{00ij} \\
&\quad + \sum_{i,j,k,l=1}^{3} p_{i\mu} p_{j\nu} p_{k\rho} p_{l\sigma} D_{ijkl}
\end{aligned}
$$

All coefficient functions and the scalar integrals are summarized in one generic function, **`PaVe`**.

PaVe[*i, j, ...,* **{***P10, P12, ...***}, {***m02, m12, ...***}]**	Passarino-Veltman coefficient functions

Passarino-Veltman coefficient functions of the tensor integral decomposition.

The first set of arguments *i, j, ...* are exactly those indices of the coefficient functions of the tensor integral decomposition. If only a **0** is given as first argument, the scalar integrals are understood. The last argument, the list of inner masses m_0^2, m_1^2, ..., determines whether a one-, two-, three- or four-point function is meant. **PaVe** is totally symmetric in the *i, j, ...* arguments. The foremost argument is the list of scalar products of the p_i. They are the same as defined above for the scalar B_0, C_0 and D_0 functions. For A_0 an empty list has to be given.

A certain set of special **PaVe** shown in the following examples simplify to the usual notation.

To shorten the input squared masses are abbreviated with a suffix 2, i.e., a mass m^2 is denoted by **m2**. The scalar quantity p^2 is entered as **pp**, p_i^2 as **pi0** and $(p_i - p_j)^2$ as **pij**.

For the following examples some options are set.

```
In[165]:= SetOptions[A0, A0ToB0 → False];
          SetOptions[{B1, B00, B11}, BReduce →
          False];
```

This is the scalar Passarino-Veltman one-point function.

```
In[166]:= PaVe[0, {}, {m02}]

Out[166]= A0(m02)
```

This is the two-point function $B_0(p^2, m_0^2, m_1^2)$.

```
In[167]:= PaVe[0, {pp}, {m02, m12}]

Out[167]= B0(pp, m02, m12)
```

Here is the coeffi cient function $B_1(p^2, m_0^2, m_1^2)$.

```
In[168]:= PaVe[1, {pp}, {m12, m22}]

Out[168]= B1(pp, m12, m22)
```

Coeffi cient functions of metric tensors have two "0" indices for each $g^{\mu\nu}$.

```
In[169]:= PaVe[0,0, {pp}, {m02, m12}]

Out[169]= B00(pp, m02, m12)
```

This is $B_{11}(p_1^2, m_0, m_1)$, the coeffi cient function of $p_{1\mu}\, p_{1\nu}$ of $B_{\mu\nu}$.

```
In[170]:= PaVe[1,1, {p10}, {m12, m22}]

Out[170]= B11(p10, m12, m22)
```

PaVe with one 0 as first argument are scalar Passarino-Veltman integrals.

```
In[171]:= PaVe[0, {p10, p12, p20}, {m12, m22,
            m32}]

Out[171]= C0(p10, p12, p20, m12, m22, m32)
```

This is D_0 with 10 arguments.

```
In[172]:= PaVe[0, {p10, p12, p23, p30, p20, p13},
            {m12, m22, m32, m42}]

Out[172]= D0(p10, p12, p23, p30, p20, p13, m12, m22, m32, m42)
```

B1[*p10***,** *m02***,** *m12***]**	coefficient function $B_1(p_1^2, m_0^2, m_1^2)$
B00[*p10***,** *m02***,** *m12***]**	coefficient function $B_{00}(p_1^2, m_0^2, m_1^2)$
B11[*p10***,** *m02***,** *m12***]**	coefficient function $B_{11}(p_1^2, m_0^2, m_1^2)$

Two-point coefficient functions.

The two-point coefficient functions can be reduced to lower-order ones. For special arguments also B_0 is expressed in terms of A_0, if the option **BReduce** is specified. Setting the option **B0Unique** to **True** simplifies $B_0(m^2, 0, m^2) \to 2 + B_0(0, m^2, m^2)$ and $B_0(0, 0, m^2) \to 1 + B_0(0, m^2, m^2)$.

option name	*default value*	
A0ToB0	**False**	express $A_0(m^2)$ by $m^2(1 + B_0(0, m^2, m^2))$
BReduce	**True**, **False** for **B0**	reduce **B0**, **B1**, **B00**, **B11** to **A0** and **B0**
B0Unique	**False**	simplify $B_0(a, 0, a)$ and $B_0(0, 0, a)$

Reduction options for **A0** and the two-point functions.

The default is to reduce B_1 to B_0.

```
In[173]:= B1[pp, m12, m22]
```

$$\text{Out[173]= } -\frac{B_0(\text{pp},\ \text{m12},\ \text{m22})}{2} + \frac{(-\text{m12} + \text{m22})\,(-B_0(0,\ \text{m12},\ \text{m22}) + B_0(\text{pp},\ \text{m12},\ \text{m22}))}{2\ \text{pp}}$$

Arguments of two-point functions with head **Small** are replaced by 0, if the other arguments have no head **Small** and are nonzero. **A0[0]** and **A0[SmallVariable[m]^2]** simplify to 0.

The small mass **m** is set to 0, since the other arguments are non-zero and not **SmallVariable**.

```
In[174]:= B1[pp, SmallVariable(me2), m22]
```

$$\text{Out[174]= } -\frac{B_0(\text{pp},\ 0,\ \text{m22})}{2} + \frac{\text{m22}\,(-B_0(0,\ 0,\ \text{m22}) + B_0(\text{pp},\ 0,\ \text{m22}))}{2\ \text{pp}}$$

But in this case no arguments are replaced by 0. **SmallVariable** heads are not displayed int **TraditionalForm**

```
In[175]:= B1[SmallVariable[me2],
           SmallVariable(me2), 0]
```

$$\text{Out[175]= } -\frac{1}{2} - \frac{\text{B0(me2, 0, me2)}}{2}$$

In **StandardForm** we see them.

```
In[176]:= % // StandardForm
```

$$\text{Out[176]= } -\frac{1}{2} - \frac{\text{B0[SmallVariable[me2], 0, SmallVariable[me2]]}}{2}$$

Any mass with head **SmallVariable** is neglected if it appears in a sum, but not as an argument of Passarino-Veltman (**PaVe**) functions or **PropagatorDenominator**.

SmallVariable[*var***]**	is a small (negligible) variable

Head for small variables.

PaVeReduce[*expr***]**	reduces coefficient functions **PaVe** to **A0**, **B0**, **C0**, **D0**
KK[*i***]**	abbreviations in **HoldForm** in the result of **PaVeReduce**

Reduction function for Passarino-Veltman coefficient functions.

Depending on the option **BReduce** **B1**, **B00** and **B11** may also remain in the result of **PaVeReduce**.

option name	*default value*	
IsolateNames	**False**	use **Isolate** with this option
Mandelstam	**{}**	Mandelstam relation, e.g., **{s, t, u, 2 mw^2}**

Options for **PaVeReduce**.

The function **Isolate** is explained in section 7.3.

Reduce $C_2(m_e^2, m_W^2, t, m_e^2, 0, m_W^2)$ to scalar integrals.

```
In[177]:= PaVeReduce[ PaVe[2, {SmallVariable[me2],
            mw2, t},{SmallVariable[me2], 0, mw2} ]]
```

$$\text{Out[177]=} \quad \frac{B_0(\text{mw2},\ 0,\ \text{mw2})}{\text{mw2} - t} - \frac{B_0(t,\ 0,\ \text{mw2})}{\text{mw2} - t}$$

Break down the coeffi cient function $C_{12}(s, m^2, m^2, m^2, m^2, M^2)$. This is the result in **HoldForm**.

```
In[178]:= c12 = PaVeReduce[ PaVe[1, 2, {s, m2,
          m2}, {m2, m2, M2}],
          IsolateNames → c ]

Out[178]= c(11)
```

The **FullForm** of the assignment to **c12** is **HoldForm[c[11]]**. If you want to get the value of **c[11]**, you can either type **ReleaseHold[c12]** or **c[11]** as done in the example.

```
In[179]:= c[11]
```

$$\text{Out[179]= } \frac{2\,\text{m2}\,c(3)}{s\,c(6)} + \frac{c(1)\,c(7)}{s\,c(6)^2} + \frac{c(2)\,c(8)}{2\,c(6)^2} + \frac{c(4)\,c(9)}{s\,c(6)} + \frac{\text{M2}\,c(5)\,c(10)}{c(6)^2} + \frac{1}{2\,c(6)}$$

Have B_1's be reduced to B_0's.

```
In[180]:= SetOptions[B1, BReduce → True];
```

Repeated application of **ReleaseHold** reinserts all **K**.

```
In[181]:= Simplify /@ Collect[FixedPoint[ReleaseHold,
          c12], _B0, _C0]
```

$$\text{Out[181]= } \frac{1}{2\,(4\,\text{m2} - s)} + \frac{(\text{m2} - \text{M2})\,B_0(0,\ \text{m2},\ \text{M2})}{2\,\text{m2}\,(4\,\text{m2} - s)} - \frac{(8\,\text{m2}^2 - 10\,\text{m2}\,\text{M2} - 2\,\text{m2}\,s + \text{M2}\,s)\,B_0(\text{m2},\ \text{m2},\ \text{M2})}{2\,\text{m2}\,(4\,\text{m2} - s)^2} + \frac{(4\,\text{m2} - 6\,\text{M2} - s)\,B_0(s,\ \text{m2},\ \text{m2})}{2\,(4\,\text{m2} - s)^2} + \frac{\text{M2}\,(8\,\text{m2} - 3\,\text{M2} - 2\,s)\,C_0(\text{m2},\ \text{m2},\ s,\ \text{m2},\ \text{M2},\ \text{m2})}{(4\,\text{m2} - s)^2}$$

Take a coeffi cient function from $D_{\mu\nu\rho}$: $D_{122}(m_e^2, m_w^2, m_w^2, m_e^2, s, t, 0, m_e^2, 0, m_e^2)$.

```
In[182]:= d122 = PaVeReduce[
          PaVe[1, 2, 2, {SmallVariable[me2], mw2,
          mw2, SmallVariable[me2], s, t},
          {0, SmallVariable[me2], 0,
          SmallVariable[me2]}],
          Mandelstam → {s, t, u, 2 mw2},
          IsolateNames → f ]

Out[182]= f[16]
```

Write the result out into a Fortran file.

```
In[183]:= Write2[ "d122.for", d122res = d122,
          FormatType → FortranForm ];
```

This shows the resulting Fortran file.
The first abbreviations are always the scalar integrals.
The partially recursive definitions of the abbreviations are not fully optimized.
The function **Write2** is explained in Section 7.
Note that the head **SmallVariable** is eliminated in the Fortran output automatically.

```
In[184]:= !!d122.for

      f(1)= B0(mw2,me2,0D0)
      f(2)= B0(s,0D0,0D0)
      f(3)= B0(t,me2,me2)
      f(4)= C0(mw2,mw2,t,me2,0D0,me2)
      f(5)= C0(mw2,s,me2,me2,0D0,0D0)
      f(6)= C0(t,me2,me2,me2,me2,0D0)
      f(7)= D0(mw2,mw2,me2,me2,t,s,me2,0D0,me2,0D0)
      f(8)= mw2 + s
      f(9)= 4*mw2 - t
      f(10)= mw2**2 - s*u
      f(11)= mw2 - s
      f(12)= 4*mw2**5-5*mw2**4*s-16*mw2**3*s**2+
     & 4*mw2**2*s**3 + 4*mw2*s**4 - mw2**4*u -
     & 4*mw2**2*s**2*u+8*mw2*s**3*u+4*mw2**2*s*u**2+
     & s**3*u**2 + s**2*u**3
      f(13)= mw2**2 - 4*mw2*s + 2*s**2 + s*u
      f(14)= 4*mw2**3-9*mw2**2*s+2*s**3-mw2**2*u-
     & 4*mw2*s*u + 5*s**2*u + 3*s*u**2
      f(15)= 2*mw2**6-8*mw2**5*s+12*mw2**4*s**2-
     & 8*mw2**3*s**3+2*mw2**2*s**4-2*mw2**5*t+
     & 20*mw2**4*s*t-36*mw2**3*s**2*t+20*mw2**2*s**3*t-
     & 2*mw2*s**4*t-6*mw2**3*s*t**2+6*mw2**2*s**2*t**2-
     & 6*mw2*s**3*t**2+4*mw2*s**2*t**3-s**2*t**4
      f(16)= -f(8)/(2D0*f(9)*f(10)) -
     & (s**2*t**2*f(6)*f(11))/(2D0*f(10)**3) +
     & (s**3*t**2*f(7)*f(11))/(2D0*f(10)**3) +
     & (s**2*t*f(5)*f(11)**2)/f(10)**3 -
     & (f(1)*f(12))/(2D0*f(9)**2*f(10)**2*f(11)) +
     & (s*f(2)*f(13))/(2D0*f(10)**2*f(11)) +
     & (f(3)*f(8)*f(14))/(2D0*f(9)**2*f(10)**2) -
     & (f(4)*f(8)*f(15))/(2D0*f(9)**2*f(10)**3)
      d122res = f(16)
```

Delete the output file.

```
In[185]:= DeleteFile["d122.for"];
```

The Fortran code generated by **Write2** should be checked with care. All integer numbers (except 0 as argument of **B0**, **C0**, **D0**) are translated to integers. This causes problems when translating variables with rational powers and must be corrected in the Fortran output by hand.

5.2 A One-Loop Self Energy Diagram

The function **OneLoop** performs the algebraic simplifications of a given amplitude. The result is given in a polynomial of standard matrix elements, invariants of the process under consideration, and Passarino-Veltman integrals.

OneLoop[*q*, *amp***]**	calculates the one-loop amplitude *amp* with *q* as loop momentum
OneLoop[*name*, *q*, *amp***]**	calculates the one-loop amplitude *amp* and gives it a name

Calculating one-loop amplitudes.

PropagatorDenominator[Momentum[*q***], *m*]**	a factor of the denominator of a propagator
PropagatorDenominator[Momentum[*q*, *D***], *m*]**	a factor of the denominator of a propagator in *D* dimensions
FeynAmpDenominator[PropagatorDenominator[...], PropagatorDenominator[...]]	a propagator

Representation of integrands.

The first argument to **OneLoop** is optional. It indicates a name for the amplitude for bookkeeping reasons. The second argument *q* is the loop momentum, i.e., the integration variable.

As last argument the analytical expression for the graph is given. It may be given in four dimensions. **OneLoop** performs the necessary extension to *D* dimensions automatically.

This is $A_0 = -i\,\pi^{-2}\,(2\,\pi\mu)^{4-D}\int d^D q\,(q^2-m^2)^{-1}$, corresponding to a tadpole diagram.
The scaling variable μ is suppressed in FeynCalc.

```
In[186]:= -I/Pi^2 FeynAmpDenominator[
            PropagatorDenominator[q, m]]
```

$$\text{Out[186]= } -\frac{i}{\pi^2(q^2 - m^2)}$$

This calculates the tadpole diagram.

```
In[187]:= a0 = OneLoop[q, a0]
```

$$\text{Out[187]= } A_0(m^2)$$

Have **A0** written in terms of **B0**.

```
In[188]:= SetOptions[A0, A0ToB0 → True];
```

This calculates the tadpole diagram again. Now the result is given in terms of **B0**.

```
In[189]:= OneLoop[q, a0]
```

$$\text{Out[189]= } m^2 B_0(0,\ m^2,\ m^2) + m^2$$

Return to the default.

```
In[190]:= SetOptions[A0, A0ToB0 → False];
```

For a most compact result the factoring option of **OneLoop** is set. For a description of all options of **OneLoop** see section 5.4.

```
In[191]:= SetOptions[OneLoop, Factoring → True];
```

This is the transversal part of a photon self energy diagram with a fermion loop.
$ie^2/((2\pi)^4\,(1-D))$
$\int d^4q\,[q^2-m_f^2]^{-1}[(q-k)^2-m_f^2]^{-1}$
$tr[\,(m_f+\not{q}-\not{k})\,\gamma^\nu\,(\not{q}+m_f)\,\gamma^\nu\,]$
$=$
$-[e^2(k^2+6m_f^2B_0(0,m_f^2,m_f^2)-$
$-3(k^2+$
$2m_f^2)B_0(k^2,m_f^2,m_f^2))]/(36\pi^2)$

```
In[192]:= OneLoop[ q, (I el^2)/(16 Pi^4)/(1 - D) *
          FeynAmpDenominator[
          PropagatorDenominator[q, mf],
          PropagatorDenominator[q - k, mf] ] *
          DiracTrace[(mf + DiracSlash[q - k]) .
          DiracMatrix[mu] .
          (mf + DiracSlash[q]) . DiracMatrix[mu]]
          ] /. ScalarProduct[k, k] → k2 /.
          (mf^2) → mf2
```

$$\text{Out[192]= } \frac{\text{el}^2\ (-\text{k2} + 6\,\text{mf2} - 6\,A_0(\text{mf2}) + 3\,(\text{k2} + 2\,\text{mf2})\,B_0(\text{k2},\ \text{mf2},\ \text{mf2}))}{36\,\pi^2}$$

Note that in this example, where the dimension is entered explicitly as a parameter (D), the option **Dimension** of **OneLoop** must also be set to D (this is the default).

5.3 Generic Diagrams for W → $f_i\bar{f}'_j$ with OneLoop

As an example for calculating triangle diagrams the result for two generic one-loop diagrams of the decay W → $f_i\bar{f}'_j$ for massless fermions given in [9] is verified with FeynCalc.

For the two diagrams different approaches are taken. In the first one FeynCalc introduces standard matrix elements, i.e., that part of the diagram containing polarization dependencies. In the other approach the set of standard matrix elements is defined by the user before FeynCalc calculates the diagrams. The last possibility is usually preferable, since the choices of FeynCalc for the standard matrix elements may have physical significance only by accident.

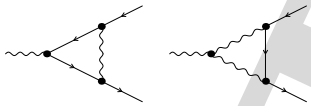

Figure 2: Two generic diagrams for the decay of $W \to f_i \bar{f}'_j$, generated by FeynArts.

This defines a function for abbreviation purposes: $g_i^c = g_i^- \omega_- + g_i^+ \omega_+$. Set the p_i on-shell and $(p_1 \cdot p_2) = k^2/2$, where k denotes the momentum of the W.

```
In[193]:= gc[i_] := g[i, "-"] DiracMatrix[7] +
          g[i, "+"] DiracMatrix[6];
          ScalarProduct[p1, p1] = 0;
          ScalarProduct[p2, p2] = 0;
          ScalarProduct[p1, p2] = k2/2;
          MakeBoxes[g[i_, j_], TraditionalForm] :=
          SubsuperscriptBox[g, i, j];
```

The analytical expression for the generic diagram is:

$\delta m_1 = i(2\pi)^{-4} \int (2\pi\mu)^{4-D} d^D q$
$[(q^2 - m^2)(q + p_1)^2 (q - p_2)^2]^{-1}$
$\overline{u}(p_1) \gamma^\nu (g_1^- \omega_- + g_1^+ \omega_+) (\not{q} + \not{p})$
$\not{\varepsilon}(g_3^- \omega_- + g_3^+ \omega_+) (\not{q} -$
$\not{p}_2) \gamma^\nu (g_2^- \omega_- + g_2^+ \omega_+) v(p_2)$

```
In[194]:= wff1 = OneLoop[ q, I/(2 Pi)^4
          FeynAmpDenominator[
          PropagatorDenominator[q, m],
          PropagatorDenominator[q + p1],
          PropagatorDenominator[q - p2]] *
          Spinor[p1] . DiracMatrix[nu] . gc[1] .
          DiracSlash[q + p1] .
          DiracSlash[Polarization[k]] .
          gc[3] . DiracSlash[q - p2] .
          DiracMatrix[nu] . gc[2] .
          Spinor[p2]  ] /. (m^2) → m2
```

Translated into the usual notation the result reads:

$(1 - 2B_0(k^2, 0, 0) +$
$2k^2 C_0(0, 0, k^2, 0, m^2, 0) +$
$2k^2 C_1(k^2, 0, 0, 0, 0, m^2) +$
$4C_{00}(k^2, 0, 0, 0, 0, m^2)$
$(g_1^+ g_2^+ g_3^+ \overline{u}(p_1) \not{\varepsilon} \omega_+ v(p_2) +$
$g_1^- g_2^- g_3^- \overline{u}(p_1) \not{\varepsilon} \omega_- v(p_2)))/(16\pi^2)$

```
Out[194]= -1/(16 π^2) ((4 B0(0, 0, m2) - 2 B0(k2, 0, 0) -
          2 k2 C0(0, 0, k2, 0, m2, 0) - 2 m2 C0(0, 0, k2, 0, m2, 0) -
          4 PaVe(0, 0, {0, k2, 0}, {m2, 0, 0}) - 1)
          (g1+ g2+ g3+ ||φ(p1) . (γ · ε(k)) . γ^6 . φ(p2)|| +
          g1- g2- g3- ||φ(p1) . (γ · ε(k)) . γ^7 . φ(p2)||))
```

The remaining Dirac structure is wrapped with the head **`StandardMatrixElement`**, which displays like $\| \cdots \|$.

This reduces the result to scalar integrals.

```
In[195]:= wff1a = PaVeReduce[wff1] // Simplify
```

$$\text{Out[195]}= \frac{1}{16\,\text{k2}\,\pi^2}$$
$$\Big(\big(-2\,(2\,\text{k2}+\text{m2})\,B_0(0,0,\text{m2})+(3\,\text{k2}+2\,\text{m2})\,B_0(\text{k2},0,0)+$$
$$2\,\big(C_0(\text{k2},0,0,0,0,\text{m2})\,(\text{k2}+\text{m2})^2+\text{k2}\big)\big)$$
$$\big(g(1,+)\,g(2,+)\,g(3,+)\,\text{II}\varphi(\text{p1})\,.\,(\gamma\cdot\varepsilon(k))\,.\,\gamma^6\,.\,\varphi(\text{p2})\text{II}+$$
$$g(1,-)\,g(2,-)\,g(3,-)\,\text{II}\varphi(\text{p1})\,.\,(\gamma\cdot\varepsilon(k))\,.\,\gamma^7\,.\,\varphi(\text{p2})\text{II}\big)\Big)$$

With this command you can extract the standard matrix elements.

```
In[196]:= var = Select[Variables[wff1a],
          (Head[#]===StandardMatrixElement)&]
```

$$\text{Out[196]}= \{\text{II}\varphi(\text{p1})\,.\,(\gamma\cdot\varepsilon(k))\,.\,\gamma^6\,.\,\varphi(\text{p2})\text{II},$$
$$\text{II}\varphi(\text{p1})\,.\,(\gamma\cdot\varepsilon(k))\,.\,\gamma^7\,.\,\varphi(\text{p2})\text{II}\}$$

Here the **StandardMatrixElement**s are set to some abbreviations.

```
In[197]:= Set @@ {var, {ma[1], ma[2]} }
```

In this way you can generate a Fortran file.
With replacements you can adapt the result to your other Fortran code.

```
In[198]:= Write2["wff1a.for", vert = wff1a /.
          g[i_, "+"] → gp[i] /.
          g[i_, "-"] → gm[i], FormatType →
          FortranForm];
```

Show the content of the file.

```
In[199]:= !!wff1a.for
```

```
vert = (((3*k2 + 2*m2)*B0(k2,Null,Null) -
& 2*(2*k2 + m2)*B0(Null,m2,Null) +
& 2*(k2 + (k2 + m2)**2*
& C0(k2,Null,Null,Null,Null,m2)))*
& (gp(1)*gp(2)*gp(3)*ma(1)+gm(1)*gm(2)*gm(3)*ma(2)))
& /(16D1*k2*Pi**2)
```

Clean up.

```
In[200]:= DeleteFile["wff1a.for"];
          Clear[gc, g, wff1, wff1a, ma];
```

StandardMatrixElement[*expr***]**	head of a standard matrix element
SetStandardMatrixElements[{ ***{****sm1* → *abb1****}, {****sm2* → *abb2****}, ...}]***	set abbreviations for standard matrix elements
SetStandardMatrixElements[{ ***{****sm1* → *abb1****}, {****sm2* → *abb2****}, ...}*** , $k_2 \to p_1 + p_2 - k_1$**]**	set abbreviations for standard matrix elements by using energy momentum conservation

A head for identifying standard matrix elements; *sm*1, *sm*2 are the standard matrixelements, *abb*1, *abb*2 the abbreviations.

The function **SetStandardMatrixElements** introduces **StandardMatrixElement[***abb1* **]** for *sm1*. The abbreviations *abb1*, *abb2*, ... may be numbers or strings.

For calculating the generic triangle diagram with a non-abelian gauge coupling the standard matrix elements are set ahead using **SetStandardMatrixElements**.

In addition to the first example the option **ReduceToScalars** is set to **True**, which will produce directly a result in terms of B_0 and C_0.
The other definitions are convenient abbreviations: **r** for the right-handed projection operator γ_6, **l** for the left-handed projection operator γ_7, short mnemonic functions like **mt**, **fv** and **feynden** stand for metric tensors, four-vectors and denominators of propagators.

```
In[201]:= r = DiracMatrix[6]; l = DiracMatrix[7];
          ScalarProduct[p1, p1] =
          ScalarProduct[p2, p2] = 0;
          ScalarProduct[p1, p2] = k2/2 ;
          SetOptions[ OneLoop,
          Factoring → True, FormatType →
          FortranForm,
          ReduceToScalars → True, WriteOut → True,
          FinalSubstitutions → {g[i_, "+"] →
          gp[i], g[i_, "-"] → gm[i],
          StandardMatrixElement →  ma} ];
          mt = MetricTensor; fv = FourVector;
          feynden[x:{_, _}..] :=
          FeynAmpDenominator @@
          Map[Apply[PropagatorDenominator, #]&,
          {x}];
```

This sets the standard matrix elements:
$\mathcal{M}_1^+ = \overline{u}(p_1) \not{\varepsilon}_1 \omega_+ v(p_2)$
$\mathcal{M}_1^- = \overline{u}(p_1) \not{\varepsilon}_1 \omega_- v(p_2)$

```
In[202]:= SetStandardMatrixElements[
          { (  Spinor[p1] . DiracSlash[Polarization[k]].
          r . Spinor[p2] ) → {1},
          (  Spinor[p1] . DiracSlash[Polarization[k]].
          l . Spinor[p2] ) → {2}}];
```

Here is the second generic diagram.
Note that in the result **StandardMatrixElement** is replaced by **mat**, as specified in the option **FinalSubstitutions** of **OneLoop** on the previous page.

```
In[203]:= OneLoop[q, I/(2 Pi)^4 *
          feynden[{q, 0}, {q + p1, m1}, {q - p2,
          m2}] *
          Spinor[p1] . DiracMatrix[nu] .
          (g[1, "-"] l + g[1, "+"] r) .
          DiracSlash[-q] . DiracMatrix[ro] .
          (g[2, "-"] l + g[2, "+"] r) .
          Spinor[p2] *
          g3 ( mt[ro, mu] fv[p1 + 2 p2 - q, nu] -
          mt[mu, nu] fv[2 p1 + p2 + q, ro] +
          mt[nu, ro] fv[2 q + p1 - p2, mu] ) *
          PolarizationVector[k, mu]
          ] /. (m1^2) → m12 /. (m2^2) → m22
```

$$\text{Out[203]= } -\frac{1}{(16\,\text{k2}\,\pi^2)}$$
$$(\text{g3}\,((2\,\text{k2} + \text{m12})\,B_0(0,\,0,\,\text{m12}) +$$
$$(2\,\text{k2} + \text{m22})\,B_0(0,\,0,\,\text{m22}) -$$
$$(\text{k2} + \text{m12} + \text{m22})\,B_0(\text{k2},\,\text{m12},\,\text{m22}) + 2\,(\text{k2}\,\text{m12} +$$
$$\text{m22}\,\text{m12} + \text{k2}\,\text{m22})\,C_0(\text{k2},\,0,\,0,\,\text{m12},\,\text{m22},\,0))$$
$$(\text{gp(1) gp)(2) ma(1)} + \text{gm(1) gm(2) ma(2)}))$$

As specified above in the options for **OneLoop**, a Fortran output is written into a file.

```
In[204]:= !!"wff2.for"

Out[204]= wff2 = -(g3*(-((k2 + m1**2 +
          m2**2)*B0(k2,m1**2,m2**2)) +
          & (2*k2 + m1**2)*B0(Null,m1**2,Null) +
          & (2*k2 + m2**2)*B0(Null,m2**2,Null) +
          & 2*(k2*m1**2 + k2*m2**2 + m1**2*m2**2)*
          & C0(k2,Null,Null,m1**2,m2**2,Null))*
          & (gp(1)*gp(2)*ma(1) + gm(1)*gm(2)*ma(2)))/
          & (16D1*k2*Pi**2)
```

Clean up.

```
In[205]:= DeleteFile["wff2.for"];
          DeleteFile /@ FileNames["PaVe*"];
          Clear[r, l, mt, fv, feynden, wff2];
```

5.4 The Options of OneLoop

Several options of **OneLoop** have already been introduced in the previous section. Here the full list of available options is briefly discussed. The example in Section 5.6 shows the use of some options.

In the automatic calculation of one-loop amplitudes it does not matter in which order the arguments of **FeynAmpDenominator** are given. Therefore the default setting of **DenomatorOrder** is **True**. In case you want to verify a result obtained by hand calculation, you can set this option to **False**, which will preserve the order of the propagators as entered. If you want to include the dimension D explicitly in the input, as in the

example in Section 5.2, you have to set **Dimension** → **D**.

With the default setting of **Dimension** you can enter four-dimensional objects to **OneLoop**, which are automatically extended to D dimensions inside **OneLoop**. In case you want to calculate a finite amplitude, you can set **Dimension** → **4**.

The option **FinalSubstitutions** indicates substitutions that are done at the very end of the calculation, which may be useful to adapt the output to your Fortran program.

The **Factoring** option should be used only for relatively small problems, since it may be very time consuming to factor the result. Unless the result of **OneLoop** is very short, only the coefficients of **StandardMatrixElement** are factored.

FormatType takes **InputForm**, **FortranForm**, **MacsymaForm** or **MapleForm** as settings. If the option **WriteOut** is set to **True**, the result is written out into a file using **Write2** with the setting of **FormatType**.

Replacements are done with **InitialSubstitutions** and **FinalSubstitutions**. Especially energy momentum conservation should be included, e.g., **InitialSubstitutions** → **{k2** → **- k1 + p1 + p3}**. Note that the rules listed in **FinalSubstitutions** are not applied as one list of rules, but sequentially in a loop.

If **IsolateNames** is set to **c**, for example, the result will be given as a **c[i]** in **HoldForm**. See **Isolate** for more information. The setting of **Mandelstam** may be, e.g., **Mandelstam** → **{***s***,** *t***,** *u***,** *m1*^**2 +** *m2*^**2 +** *m3*^**2 +** *m4*^**2}**, where $s + t + u = m_1^2 + m_2^2 + m_3^2 + m_4^2$.

The option **ReduceToScalars** should not be set to **True** when calculating several complicated diagrams involving $D_{\mu\nu\rho}$ or $D_{\mu\nu\rho\sigma}$. Depending on the computer you are using it may nevertheless work, but it is usually better to use **OneLoopSum** with the appropriate options. Note that depending on the setting of the option **BReduce** also two-point coefficient functions may remain in the result.

For processes with light external fermions it is best not to neglect the fermion masses everywhere, but to keep them in the arguments of the scalar Passarino-Veltman functions. This set of masses should be supplied as a list to the option **SmallVariables**, see section 5.6.

If **WriteOut** is set to **True**, the result is written out into a file composed of the first argument of **OneLoop**, i.e., the *name*. In which language, i.e., Mathematica, **Fortran**, **Macsyma** or **Maple** the result is written, depends on the setting of the option **FormatType**. You may also set **WriteOut** to a string, which denotes the directory to write the result files to (actually this string is simply prepended to the file names).

option name	*default value*	
Apart2	**True**	use **Apart2** to partial fraction denominators
CancelQP	**True**	cancel $q \cdot p$ and q^2
DenominatorOrder	**False**	order the arguments of **FeynAmpDenominator** canonically
Dimension	**D**	number of space-time dimensions
Factoring	**False**	factor the result
FinalSubstitutions	**{}**	substitutions done at the end of the calculation
FormatType	**InputForm**	how to write out the result file
InitialSubstitutions	**{}**	substitutions done at the beginning of the calculation, i.e., energy momentum conservation
IntermediateSubstitutions	**{}**	substitutions done at an intermediate stage of the calculation
IsolateNames	**False**	use **Isolate** on the result
Mandelstam	**{}**	indicate the Mandelstam relation
OneLoopSimplify	**False**	use **OneLoopSimplify** at the beginning of the calculation
Prefactor	**1**	extra prefactor of the amplitude
ReduceGamma	**False**	insert for γ_6 and γ_7 their definitions
ReduceToScalars	**False**	reduce to B_0, C_0, D_0
SmallVariables	**{}**	a list of masses, which will get wrapped with the head **SmallVariable**
WriteOut	**False**	write out a result file carrying the name of the optional first argument of **OneLoop**
WriteOutPaVe	**False**	store **PaVe**s in files
Sum	**True**	compatibility with FeynArts; sum terms multiplied with FeynArts's **SumOver**

Options of **OneLoop**.

5.5 OneLoopSum and Its Options

To sum a list of amplitudes two different methods are provided by **OneLoopSum**. Either the provided list of amplitudes is calculated and subsequently summed, or the summation occurs partially before calculation. This can be specified with the option **CombineGraphs**.

OneLoopSum[List[FeynAmp[**GraphName[...,** *N1*, *q*, *amp1*], **FeynAmp[GraphName[...,** *N2*, *q*, *amp2*], ...]]	calculate a list of amplitudes
OneLoopSum[*expr***]**	sum already calculated amplitudes

A function for summing one-loop amplitudes.

The input of **OneLoopSum** is adapted to the output of FeynArts [3] . After saving a list of Feynman diagrams (that is, unintegrated amplitudes) created with FeynArts using the function **CreateFeynAmp**, i.e. **CreateFeynAmp[***ins***] // PickLevel[Particles] // ToFA1Conventions »** *filename*, you can start a new Mathematica session [4], load FeynCalc, get the amplitudes by *amps* **= «** *filename* , calculate the amplitude with **OneLoopSum** and finally save the result e.g. as a Fortran file.

To actually use the Fortran file, obviously the constants and functions used, like masses and scalar integrals, have to be defined. This is discussed in section 5.7.

Instead of supplying a list of not yet calculated amplitudes you can also give the sum of already calculated ones as argument (*expr*) to **OneLoopSum**.

The output of **OneLoopSum** is typically given as a short expression wrapped in **HoldForm**. In order to get the full expression, the function **FRH** can be applied.

FRH[*exp***]**	removes all **HoldForm** and **Hold** in *exp*

A utility function.

[3] Actually to version 1 of FeynArts, but the current version 3 provides translation to version 1 syntax (**ToFA1Conventions**). The examples given here all use FeynArts version 3,

[4] It is not possible to load FeynCalc and FeynArts simultaneously into one Mathematica session because some functions of FeynArts and FeynCalc have the same name but are in different contexts (name spaces). A more sophisticated approach is provided by the optional subpackage PHI, which patches FeynArts slightly, allowing the two packages to be loaded simultaneously.

option name	*default value*	
CombineGraphs	**{}**	combine amplitudes
Dimension	**True**	number of space-time dimensions
ExtraVariables	**{}**	list of variables which are bracketed out in the result like **B0**, **C0**, **D0** and **PaVe**
FinalFunction	**Identity**	function applied to the result
FinalSubstitutions	**{}**	substitutions done at the end of the calculation
FormatType	**InputForm**	format used when saving results
InitialSubstitutions	**{}**	substitutions done at the beginning of the calculation
IntermediateSubstitutions	**{}**	substitutions done at an intermediate stage in the calculation
IsolateNames	**KK**	**Isolate** the result
Mandelstam	**{}**	use the Mandelstam relation
Prefactor	**1**	multiply the result by a pre-factor
ReduceToScalars	**True**	reduce to scalar integrals
SelectGraphs	**All**	which graphs to select
WriteOutPaVe	**False**	write out the reduced **PaVe**

Options of **OneLoopSum**.

With the default options **OneLoopSum** calculates each amplitude separately by substituting **OneLoop** for **FeynAmp**. Then each single **PaVe** is reduced to scalar integrals. The hard final part consists in the simplification of the rational coefficients of the scalar integrals. This may involve thousands of factorizations and can therefore take hours of CPU time. But the algebraic simplifications achieved by putting all coefficients of the scalar integrals over a common denominator and to factor them, possibly cancelling factors and reducing the singularity structure, may be very significant. These calculations may need quite a lot of RAM space, therefore the options of **OneLoopSum** allow you to split up the task of summing lots of diagrams.

First you can select a certain subclass of diagrams with the option **SelectGraphs**. You may set, e.g., **SelectGraphs** → **{1, 2, 5, 8}**, which selects the amplitudes at positions 1, 2, 5 and 8 of the argument list of **OneLoopSum**. The setting **SelectGraphs** → **{1, 2, 5, 8, {10, 40}}** also includes the range of all amplitudes from position 10 to 40.

With the option **CombineGraphs** a possibility is given to sum the graphs before calculation. This is especially useful for combining a graph with its crossed counterpart. In general it makes sense to combine all graphs with the same propagators before calculation, but for very big sums this may reduce the performance considerably. The possible settings for **CombineGraphs** are the same as for **SelectGraphs**. If you use the FeynArts syntax

for the first argument of **FeynAmp**, i.e. **GraphName[..., N1]**, the last arguments of **GraphName** for combined graphs are concatenated and a new **GraphName** for the summed amplitude is created.

By setting the option **WriteOutPaVe** you can save the result of the reduction of each **PaVe** in a file for later use. The names of the corresponding files are generated automatically. In case you use **OneLoopSum** several times it recognizes previously saved reductions and loads these results automatically. This may save a considerable amount of time. Instead of setting the option **WriteOutPaVe** to an empty string (which means that the files are written in the current directory), you can specify another directory (a string prepended to the file names).

Note that these options together with the possibility of using **OneLoopSum** on already calculated graphs gives you a lot freedom to split up the calculation, which may be necessary in order to avoid memory overflow. It can also be a good idea to set **$VeryVerbose** to 1 or 2 for monitoring the calculation.

The option **Prefactor** may be used to extract global factors. You can, i.e., put **SetOptions[OneLoop, Prefactor → (2 SW^2)**, which multiplies each single amplitude by $2s_w^2$, and set simultaneously **SetOptions[OneLoopSum, Prefactor → 1/(2 SW^2)**. The result of **OneLoopSum** has then as a global factor $1/(2s_w^2)$.

5.6 Box Graphs of $e^+e^- \to ZH$

In this section it is shown how to calculate a sum of amplitudes with **OneLoopSum**. The input consists of a one page program with process-dependent definitions. This program reads in a file with unmodified output of FeynArts for the amplitudes. The Fortran file produced is obtained without any interactive action.

The six standard matrix elements are:

$$\begin{aligned}
\mathcal{M}_0^1 &= \bar{v}(p_1)\not{\epsilon}\,\omega_+\,u(p_2)\\
\mathcal{M}_0^2 &= \bar{v}(p_1)\not{\epsilon}\,\omega_-\,u(p_2)\\
\mathcal{M}_1^1 &= \bar{v}(p_1)\not{k}_1\,\omega_+\,u(p_2)\,\epsilon\,p_1\\
\mathcal{M}_1^2 &= \bar{v}(p_1)\not{k}_1\,\omega_-\,u(p_2)\,\epsilon\,p_1\\
\mathcal{M}_2^1 &= \bar{v}(p_1)\not{k}_1\,\omega_+\,u(p_2)\,\epsilon\,p_2\\
\mathcal{M}_2^2 &= \bar{v}(p_1)\not{k}_1\,\omega_-\,u(p_2)\,\epsilon\,p_2
\end{aligned}$$

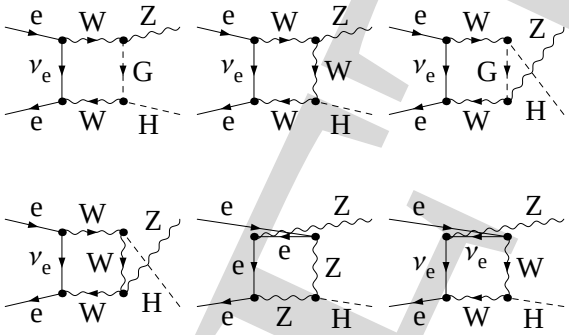

Figure 3: Box diagrams of $e^+e^- \to ZH$, generated by FeynArts. G denotes the unphysical charged Higgs.

```
(*Generate the box topologies*)
tops = CreateTopologies[1, 2 -> 2, Adjacencies -> {3},
      ExcludeTopologies -> {SelfEnergies, WFCorrections, Tadpoles,
          Boxes[3]}];

(*Check that the right topologies were generated*)
Paint[tops, ColumnsXRows -> {3, 1}];

(*Insert fields*)
inserttops =
    InsertFields[tops, {F[2, {1}], -F[2, {1}]} -> {V[2], S[1]}, Model -> "SM",
       GenericModel -> "Lorentz", InsertionLevel -> Particles,
      ExcludeFieldPoints -> {FieldPoint[F, -F, S]}];

(*Display the graphs*)
graphs = Paint[inserttops, PaintLevel -> {Particles}, AutoEdit -> False,
      SheetHeader -> False, Numbering -> False, ColumnsXRows -> {3, 2}];

(*Generate amplitude*)
eezhb = CreateFeynAmp[inserttops, AmplitudeLevel -> Particles];

(*Save amplitude in a format understood by FeynCalc*)
PickLevel[Particles][eezhb] // ToFA1Conventions >> "eezhb.amp";
```

For completeness here is the input program for generating boxes of $e^+e^- \to ZH$ with FeynArts.

```
(*Define the Mandelstam variables and put momenta on - shell*)
  SetMandelstam[s, t, u, p1, p2, -k1, -k2, SmallVariable[ME],
    SmallVariable[ME], MZ, MH];

(*Set the option for the ordering of D0's*)
SetOptions[PaVeOrder, PaVeOrderList -> {{s, t}, {s, u}, {t, u}}];

(*Set the options for OneLoop*)
SetOptions[OneLoop, Mandelstam -> {s, t, u, MH^2 + MZ^2},
    Prefactor -> 1/ALPHA2,
    InitialSubstitutions -> {k2 -> p1 + p2 - k1, CW -> MW/MZ,
        EL -> Sqrt[4 Pi Sqrt[ALPHA2]]}, SmallVariables -> {ME, ME2}];

(*The option for OneLoopSum introduces abbreviations at the end*)
SetOptions[OneLoopSum, Prefactor -> 2 ALP4PI FLUFAC,
    Mandelstam -> {s, t, u, MH^2 + MZ^2},
    FinalSubstitutions -> {SW -> Sqrt[SW2], ME -> Sqrt[ME2], MW -> Sqrt[MW2],
        MZ -> Sqrt[MZ2], MH -> Sqrt[MH2],
        ME2^n_ :> ME^(2 n) /; Head[n] =!= Integer,
        MZ2^n_ :> MZ^(2 n) /; Head[n] =!= Integer,
        MW2^n_ :> MW^(2 n) /; Head[n] =!= Integer,
        MH2^n_ :> MH^(2 n) /; Head[n] =!= Integer,
        SW2^n_ :> SW^(2 n) /; Head[n] =!= Integer,
        StandardMatrixElement -> MBM}, WriteOutPaVe -> ""];

(*Define the standard matrix elements*)
SetStandardMatrixElements[{Spinor[p1].DiracSlash[
           Conjugate[Polarization[k1]]].ChiralityProjector[+1].Spinor[
           p2] -> {0, 1},
      Spinor[p1].DiracSlash[
           Conjugate[Polarization[k1]]].ChiralityProjector[-1].Spinor[
           p2] -> {0, 2},
      ScalarProduct[Conjugate[Polarization[k1]], p1]*
          Spinor[p1].DiracSlash[k1].ChiralityProjector[+1].Spinor[p2] -> {1,
          1}, ScalarProduct[Conjugate[Polarization[k1]], p1]*
          Spinor[p1].DiracSlash[k1].ChiralityProjector[-1].Spinor[p2] -> {1,
          2}, ScalarProduct[Conjugate[Polarization[k1]], p2]*
          Spinor[p1].DiracSlash[k1].ChiralityProjector[+1].Spinor[p2] -> {2,
          1}, ScalarProduct[Conjugate[Polarization[k1]], p2]*
          Spinor[p1].DiracSlash[k1].ChiralityProjector[-1].Spinor[p2] -> {2,
          2}}, {k2 -> (p1 + p2 - k1)}];

(*get the amplitudes, which have been written to a file by FeynArts*)
eezhamp = << eezhb.amp;

(*This calculates the amplitudes and sums them up*)
eezhboxes = OneLoopSum[eezhamp, CombineGraphs -> {1, 2, 3, 4, 5, 6}];

(*Here the result is written into a Mathematica file*)
Write2["eezhb.m", EEZHBOXES = FRH[eezhboxes]];

(*Here the result is written into a Mathematica program*)
Write2["eezhb.s", EEZHBOXES = eezhboxes];

(*Here the result is written into a Fortran file*)
Write2["eezhb.for", EEZHBOXES = eezhboxes, FormatType -> FortranForm];
```

Input program for calculating boxes of $e^+e^- \to ZH$.

```
      KK(1)= B0(MH2,MZ2,MZ2)
      KK(2)= B0(MZ2,ME2,ME2)
      KK(3)= B0(MH2,MW2,MW2)
      KK(4)= B0(MZ2,0D0,0D0)
      KK(5)= B0(MZ2,MW2,MW2)
      KK(6)= D0(MH2,ME2,MZ2,ME2,t,u,MZ2,MZ2,ME2,ME2)
      KK(7)= D0(MH2,MZ2,ME2,ME2,s,t,MW2,MW2,MW2,0D0)
      KK(8)= D0(MH2,MZ2,ME2,ME2,s,u,MW2,MW2,MW2,0D0)
      KK(9)= D0(MH2,ME2,MZ2,ME2,t,u,MW2,MW2,0D0,0D0)
      KK(10)= C0(MH2,t,ME2,MZ2,MZ2,ME2)
      KK(11)= C0(MH2,u,ME2,MZ2,MZ2,ME2)
      KK(12)= C0(MZ2,t,ME2,ME2,ME2,MZ2)
      KK(13)= C0(MZ2,u,ME2,ME2,ME2,MZ2)
      KK(14)= C0(MH2,MZ2,s,MW2,MW2,MW2)
      KK(15)= C0(MH2,t,ME2,MW2,MW2,0D0)
      KK(16)= C0(MH2,u,ME2,MW2,MW2,0D0)
      KK(17)= C0(MZ2,t,ME2,0D0,0D0,MW2)
      KK(18)= C0(MZ2,t,ME2,MW2,MW2,0D0)
      KK(19)= C0(MZ2,u,ME2,0D0,0D0,MW2)
      KK(20)= C0(MZ2,u,ME2,MW2,MW2,0D0)
      KK(21)= C0(s,ME2,ME2,MW2,MW2,0D0)
      KK(22)= B0(t,ME2,MZ2)
      KK(23)= B0(u,ME2,MZ2)
      KK(24)= B0(t,MW2,0D0)
      KK(25)= B0(u,MW2,0D0)
      KK(26)= 2*MH2 - MZ2
      KK(27)= MH2 - 5*MZ2
      KK(28)= 1 - 2*SW2
      KK(29)= MW2 + MZ2
      KK(30)= 2*MH2 - MW2
      KK(31)= MW2 + 2*MZ2
      KK(32)= 4*MW2**2 + 3*MW2*MZ2 - MZ2**2*SW2
      KK(33)= 5*MH2*MW2 - 2*MW2**2 + 2*MW2*MZ2 - MH2*MZ2*SW2
      KK(34)= MH2 - MW2 + 2*MZ2
      KK(35)= 2*MW2**2 + MW2*MZ2 - MZ2**2*SW2
      KK(36)= MW2 - MZ2*SW2
      KK(37)= MW2 + MZ2*SW2
      KK(38)= MH2 + MZ2
      KK(39)= 3*MW2 + MZ2*SW2
      KK(40)= MH2*MW2 + 4*MW2**2 + 2*MW2*MZ2 - 2*MZ2**2 + MH2*MZ2*SW2
      KK(41)= 2*MH2*MW2 - 2*MW2**2 + 4*MW2*MZ2 + MZ2**2*SW2
      KK(42)= 4*MH2 - 2*MW2 + 3*MZ2
      KK(43)= 2*MW2 - MZ2
      KK(44)= MH2*MW2 - 2*MW2**2 - 3*MW2*MZ2 + 2*MZ2**2 - MH2*MZ2*SW2
      KK(45)= MH2*MW2 + 4*MW2**2 + 4*MW2*MZ2 + MH2*MZ2*SW2
      KK(46)= 5*MW2 + MZ2*SW2
      KK(47)= 2*MH2 - MW2 + 2*MZ2
      KK(48)= 3*MH2*MW2 + 2*MW2**2 + 6*MW2*MZ2 + MH2*MZ2*SW2
      KK(49)= 9*MW2 + MZ2*SW2
      KK(50)= 2*MW2 + MZ2
      KK(51)= 3*MW2 + 2*MZ2
      KK(52)= MW2**2 - MH2*MZ2 + 3*MW2*MZ2 + MZ2**2
      KK(53)= 6*MH2*MW2 - 8*MW2**2 - MH2*MZ2*SW2 + MZ2**2*SW2
      KK(54)= MH2 - 2*MW2 + MZ2
      KK(55)= 4*MH2*MW2 - 4*MW2**2 - 2*MW2*MZ2 - MH2*MZ2*SW2 + MZ2**2*SW2
      KK(56)= MH2 - MZ2
      KK(57)= 2*MH2 + MZ2
      KK(58)= 2*MH2**2 + 4*MH2*MZ2 + MZ2**2
      KK(59)= MH2 - 2*MZ2
---> ... 418 lines omitted ...
      KK(173)= ALP4PI*FLUFAC*KK(172)
      EEZHBOXES = 2*KK(173)
```

Fortran output file "eezhb.for" generated by **OneLoopSum** for $e^+e^- \to ZH$.

```
KK[1] = (B0[MH2, MZ2, MZ2]
        );
KK[2] = (B0[MZ2, SmallVariable[ME2], SmallVariable[ME2]]
        );
KK[3] = (B0[MH2, MW2, MW2]
        );
KK[4] = (B0[MZ2, 0, 0]
        );
KK[5] = (B0[MZ2, MW2, MW2]
        );
KK[6] = ( D0[MH2, SmallVariable[ME2], MZ2, SmallVariable[ME2], t, u, MZ2, MZ2,
 SmallVariable[ME2], SmallVariable[ME2]]
        );
KK[7] = ( D0[MH2, MZ2, SmallVariable[ME2], SmallVariable[ME2], s, t, MW2, MW2, MW2, 0]
        );
KK[8] = ( D0[MH2, MZ2, SmallVariable[ME2], SmallVariable[ME2], s, u, MW2, MW2, MW2, 0]
        );
KK[9] = ( D0[MH2, SmallVariable[ME2], MZ2, SmallVariable[ME2], t, u, MW2, MW2, 0, 0]
        );
KK[10] = (C0[MH2, t, SmallVariable[ME2], MZ2, MZ2, SmallVariable[ME2]]
        );
KK[11] = (C0[MH2, u, SmallVariable[ME2], MZ2, MZ2, SmallVariable[ME2]]
        );
KK[12] = ( C0[MZ2, t, SmallVariable[ME2], SmallVariable[ME2], SmallVariable[ME2], MZ2]
        );
KK[13] = ( C0[MZ2, u, SmallVariable[ME2], SmallVariable[ME2], SmallVariable[ME2], MZ2]
        );
KK[14] = (C0[MH2, MZ2, s, MW2, MW2, MW2]
        );
KK[15] = (C0[MH2, t, SmallVariable[ME2], MW2, MW2, 0]
        );
KK[16] = (C0[MH2, u, SmallVariable[ME2], MW2, MW2, 0]
        );
KK[17] = (C0[MZ2, t, SmallVariable[ME2], 0, 0, MW2]
        );
KK[18] = (C0[MZ2, t, SmallVariable[ME2], MW2, MW2, 0]
        );
KK[19] = (C0[MZ2, u, SmallVariable[ME2], 0, 0, MW2]
        );
KK[20] = (C0[MZ2, u, SmallVariable[ME2], MW2, MW2, 0]
        );
KK[21] = (C0[s, SmallVariable[ME2], SmallVariable[ME2], MW2, MW2, 0]
        );
KK[22] = (B0[t, MZ2, SmallVariable[ME2]]
        );
KK[23] = (B0[u, MZ2, SmallVariable[ME2]]
        );
KK[24] = (B0[t, 0, MW2]
        );
KK[25] = (B0[u, 0, MW2]
        );
KK[26] = (2*MH2 - MZ2
        );
KK[27] = (MH2 - 5*MZ2
        );
KK[28] = (1 - 2*SW2
        );
KK[29] = (MW2 + MZ2
        );
KK[30] = (2*MH2 - MW2
        );
(* ............. 564 lines omitted ..................... *)
KK[173] = (ALP4PI*FLUFAC*HoldForm[KK[172]]
        );
EEZHBOXES = 2*HoldForm[KK[173]]
```

Mathematica output file "eezhb.s" generated by **OneLoopSum** for $e^+e^- \to ZH$.

5.7 Processing Amplitudes

Calculating loop integrals will in most cases lead to very large expressions. The same goes for the application of other algorithms like Dirac tracing algorithms. The systematization and reduction of such expressions is a process which requires much more human planing, control and intervention than the process leading to the expressions. However, the computer is still of help. In this section we shall consider a few examples of how one may proceed, taking advantage of both tools provided by FeynCalc as well as constructing small Mathematica programs.

Load a store amplitude (see section 5.6).

```
In[206]:= « "eezhb.m"
```

Check the "size" of the expression.

```
In[207]:= EEZHBOXES // LeafCount
Out[207]= 10415
```

Expand the expressions, collect with respect to factors we know will be overall or "interesting", simplify what the non-overall factors multiply.

```
In[208]:= eezhoxes = Collect[EEZHBOXES // Expand,
          ALP4PI, FLUFAC, _MBM, _D0, _C0, _B0,
          If[FreeQ[#, _MBM | _D0 | _C0 | _B0 |
          _PaVe], FullSimplify[#], #] &];
```

Check the "size" of the resulting expression.

```
In[209]:= eezhoxes // LeafCount
Out[209]= 8881
```

Clean up.

```
In[210]:= DeleteFile /@ FileNames["eezhb*"];
          DeleteFile /@ FileNames["PaVe*"];
```

In the example above, notice that instead of simply applying **Simplify** to the whole expression, some grouping is first done and then **Simplify** is applied to individual terms. This is often advantageous because the performance of **Simplify** naturally scales very badly with the size of expressions. We remark that although **OneLoopSum** does a decent job in structuring the expression, it can still be reduced somewhat.

In the one-loop calculations considered thus far, amplitudes have been computed. The extra step of computing the (differential) cross section can, however, also be done with FeynCalc. To demonstrate this we pick a very simple example, namely the famous Møller cross section. This example also demonstrates that a full calculation from Feynman rules to cross section can be carried out with FeynArts and FeynCalc [5].

[5] In fact, as we have seen in section 4.5, the calculation of the Feynman rule from the lagrangian can also be automatized with FeynCalc.

```
(*Construction of topologies*)
tops = CreateTopologies[0, 2 -> 2, Adjacencies -> {3},
      ExcludeTopologies -> {SelfEnergies, WFCorrections}];

(*Check*)
Paint[tops, ColumnsXRows -> {3, 1}, AutoEdit -> False];

(*Field insertion*)
inserttops =
    InsertFields[tops, {F[1, {1}], F[1, {1}]} -> {F[1, {1}], F[1, {1}]},
      Model -> "QED", GenericModel -> "QED", InsertionLevel -> Particles];

(*Check*)
treegraphs =
    Paint[inserttops, PaintLevel -> {Particles}, AutoEdit -> False,
      SheetHeader -> False, Numbering -> False, ColumnsXRows -> {2, 1}];

(*Calculate the amplitudes*)
amps = CreateFeynAmp[inserttops, AmplitudeLevel -> Particles];

(*Save result*)
PickLevel[Classes][amps] // ToFA1Conventions >> "moelleramps.m"
```

For completeness here is the input program for generating the (leading order) diagrams of Møller scattering with FeynArts.

```
(*Define the Mandelstam variables and put momenta on - shell*)
  SetMandelstam[s, t, u, p1, p2, -k1, -k2, ME, ME, ME, ME];

(*get the amplitudes, which have been written to a file by FeynArts*)
amps = << moelleramps.m;

(*Sum the graphs and contract Lorentz indices*)
amp = (OneLoopSum[amps, CombineGraphs -> {1, 2}] // FRH) /. D :> Sequence[] //
     Contract

(*Square the amplitude,
  transform the spin sums into traces and evaluate the traces*)
squaredamp =
  FermionSpinSum[amp ComplexConjugate[amp /. li2 -> li1] // Expand] /.
          DiracTrace -> Tr // DiracSimplify //
      TrickMandelstam[#, {s, t, u, 4ME^2}] & // Simplify
squaredamp1 =
  squaredamp // Contract // PropagatorDenominatorExplicit // Simplify

(*The kinematical factor in the center of mass frame*)
kinfac = 1/(64 \[Pi]^2s);

(*The full differential cross section in the center of mass frame expressed \
in terms of Mandelstam variables*)
dcrosssection = 1/4*kinfac*squaredamp1 // Simplify
(*Shift to other variables : Scattering angle and half the CMS energy*)
dc = dcrosssection /. u -> 4ME^2 - s - t /. {s -> 4 \[Omega]^2,
          t -> -2 q2(1 - Sqrt[1 - sin[\[Theta]]^2])} /.
      q2 -> \[Omega]^2 - ME^2 // Simplify

(*Clean up*)
DeleteFile["moelleramps.m"];
```

Calculation of the Møller cross section.

Notice that the argument given to **`FermionspinSum`** is a product of two amplitudes. These both contain contracted indices; therefore, in one of them, these indices have to be renamed. The functions **`TrickMandelstam`** and **`PropagatorDenominatorExplicit`** are described in section 7.

ComplexConjugate[*exp***]**	conjugates the expression *exp*, operating on fermion lines
FermionSpinSum[*exp***]**	constructs traces out of squared amplitudes

Calculation of squares of fermion amplitudes.

6 Advanced Calculations

6.1 QCD with FeynCalc

6.2 ChPT with FeynCalc

6.3 Two-Loop Calculations

6.4 Amplitude and Integral Tables

7 Miscellaneous Functions

7.1 Low Level Dirac Algebra Functions

There are a number of lower level functions for doing Dirac algebra accessible (defined in contexts belongint to **$Path**). Some are used by the functions described in section 4.2, some may be useful to perform more controlled calculations. They are described here also for completeness: **DiracGammaCombine** **DiracGammaExpand** **DiracGammaT** **DiracSigma** **DiracSigmaExplicit** **DiracSpinor** **EpsChisholm** **ChisholmSpinor**

7.2 Functions for Polynomial Manipulations

Unfortunately the built-in Mathematica 2.0 functions **Factor**, **Collect** and **Together** are either not general enough for the purposes needed in FeynCalc or consume too much CPU time for certain polynomials with rational coefficients. The FeynCalc extensions **Factor2**, **Collect2** and **Combine** should be used instead when manipulating the rational polynomials emerging from **OneLoop** or **PaVeReduce**.

Collect2[*expr*, *x*$_1$, *x*$_2$, ...**]**	group together powers of terms containing $x_1, x_2, ...$
Combine[*expr***]**	put terms in a sum over a common denominator
Factor2[*expr***]**	factor a polynomial in a canonical way

Modifications of **Collect**, **Together** and **Factor**.

This collects **f[x]** and **p[y]**. The default setting is not to expand terms like $(b-2)d$.

```
In[211]:= Collect2[ (b - 2) d f[x] + f[x] + c p[y]
          + p[y], {f, y}]

Out[211]= (1 + (-2 + b) d) f[x] + (1 + c) p[y]
```

With the setting **ProductExpand → True** the product $3(s+m^2)$ gets expanded.

```
In[212]:= Collect2[3 B0[pp,m1,m2] (m^2 + s) +
          B0[pp,m1,m2] s,
          B0, ProductExpand → True]

Out[212]= (3 m^2 + 4 s) B0[pp, m1, m2]
```

This puts terms over a common denominator without expanding the numerator.

```
In[213]:= Combine[(a - b) (c - d)/e + g]
```

$$\text{Out[213]=}\ \frac{(a-b)(c-d)+e\,g}{e}$$

Consider a generic polynomial.

```
In[214]:= test = (a - b) x + (b - a) y

Out[214]= (a - b) x + (-a + b) y
```

Mapping **Factor** on the summands shows that no canonical factorization of integers in sums takes place.

```
In[215]:= Map[Factor, test]

Out[215]= (a - b) x + (-a + b) y
```

If a canonical factorization is desired, you may use **Factor2** instead.

```
In[216]:= Map[Factor2, test]

Out[216]= (a - b) x - (a - b) y
```

This is the overall factorization of **Factor**.

```
In[217]:= Factor[test]

Out[217]= (-a + b) (-x + y)
```

Here the convention used by **Factor2** becomes clear: the first term in a subsum gets a positive prefactor.

```
In[218]:= Factor2[test]

Out[218]= (a - b) (x - y)
```

option name	*default value*	
ProductExpand	**False**	expand products

An expansion controlling option for **Collect2** and **Combine**.

7.3 An Isolating Function for Automatically Introducing Abbreviations

Isolate[*expr***]**	if **Length[***expr***] > 0, substitute an abbreviation K[***i***] for** *expr*
Isolate[*expr***, {***x*$_1$**, ***x*$_2$**, ...}]**	substitute abbreviations for subsums free of $x_1, x_2, ...$

A function for isolating variables by introducing abbreviations for common subexpressions.

option name	*default value*	
IsolateHead	**K**	the head of the abbreviations
IsolateSplit	**442**	a limit beyond which **Isolate** splits the expression into two sums

Options for **Isolate** and **Collect2**.

Isolate with one argument introduces a single **K[***i***]** as abbreviation.

```
In[219]:= Isolate[a + b]

Out[219]= K[1]
```

Here *f* gets isolated with **K[1]** and **K[2]** replacing the bracketed subsums.

```
In[220]:= test = Isolate[(a + b) f + (c + d) f +
          e, f]

Out[220]= e + f K[1] + f K[2]
```

Looking at the **FullForm** reveals that the **K[*i*]** are given in **HoldForm**.

```
In[221]:= FullForm[test]

Out[221]= Out[221]//FullForm=
          Plus[e, Times[f, HoldForm[K[1]]], Times[f, Hold-
          Form[K[2]]]]
```

Asking for **K[1]** returns its value, but in **test** **K[1]** is held.

```
In[222]:= { K[1], test, ReleaseHold[test] }

Out[222]= {a + b, e + f K[1] + f K[2],

          e + (a + b) f + (c + d) f}
```

For the term **(b + c (y + z))** a single abbreviation **F[2]** is returned.

```
In[223]:= Isolate[a[z] (b + c (y + z)) + d[z] (y +
          z), {a, d},
          IsolateHead → F]

Out[223]= d[z] F[1] + a[z] F[2]
```

With the help of an additional function it is easy to introduce identical abbreviations for each subsum.
This trick of selectively substituting functions for sums is also quite useful in special reordering of polynomials.

```
In[224]:= ( und[x__] := Isolate[Plus[x],
          IsolateHead → H]/;
          FreeQ2[{x}, {a, d}];
          (a[z] (b + c (y + z)) + d[z] (y + z))/.
          Plus → und /.  und → Plus
          )

Out[224]= d[z] H[1] + a[z] H[2]
```

Here it is clear that **H[1]** = (y + z) is part of **H[2]**.

```
In[225]:= ReleaseHold[%]

Out[225]= (y + z) d[z] + a[z] (b + c H[1])
```

Decreasing the option **IsolateSplit** signifi cantly forces **Isolate** to use more than one **L** for **a - b - c - d - e**.

```
In[226]:= Isolate[ a - b - c - d - e, IsolateHead
          → L, IsolateSplit → 15 ]

Out[226]= L[2]
```

This shows the values of the **L**.

```
In[227]:= {L[2], L[1]}

Out[227]= {-c - d - e + L[1], a - b}
```

The importance of **Isolate** is significant, since it gives you a means to handle very big expressions. The use of **Isolate** on big polynomials before writing them to a file is especially recommended.

A subtle issue of the option **IsolateSplit**, whose setting refers to the **Length[Characters[ToString[FortranForm[***expr***]]]]**, is that it inhibits too many continuation lines in the Fortran file when writing out expressions involving **K[i]** with **Write2**. See Section 7.5 for the usage of **Write2**.

You may want to change **IsolateSplit** when working with big rational polynomials in order to optimize the successive (**FixedPoint**) application of functions like **Combine[ReleaseHold[#]]&**.

7.4 An Extension of FreeQ and Two Other Useful Functions

FreeQ2[*expr***, {***f_1***, ***f_2***, ...}]**	yields **True** if *expr* does not contain any occurrence of f_1, f_2, ...
NumericalFactor[*expr***]**	gives the numerical factor of *expr*
PartitHead[*expr***, *h*]**	returns a list **{***a***, *h***[b]}** with *a* free of expressions with head *h*, and *h*[*b*] having head *h*

Three useful functions.

FreeQ2 is an extension of **FreeQ**, allowing a list as second argument.

```
In[228]:= FreeQ2[M^2 + m^2 B0[pp, m1, m2], {M,
            B0}]

Out[228]= False
```

This gives the numerical factor of the expression.

```
In[229]:= NumericalFactor[-137 x]

Out[229]= -137
```

The action of **PartitHead** on a product is to split the product apart.

```
In[230]:= PartitHead[f[m] (s - u), f]

Out[230]= {s - u, f[m]}
```

A sum gets separated into subsums.

```
In[231]:= PartitHead[s^2 + M^2 - f[m], f]

Out[231]= {M^2 + s^2, -f[m]}
```

7.5 Writing Out to Mathematica, Fortran, Macsyma and Maple

The Mathematica functions **Write** and **Save** may on rare occasions create files that cannot be read in again correctly. What sometimes happens is that, for example, a division operator is written out as the first character of a line. When the file is loaded again, Mathematica may simply ignore that part of the file before this character. You can easily work around this bug by editing the file and wrapping the expressions with brackets "( )". Since this is a cumbersome procedure for lots of expressions, it has been automatized in **Write2** for writing out expressions. If you want to use **Save**, you have to check the output file for correctness.

An option can be given to **Write2** allowing the output to be written in **FortranForm**, **MacsymaForm** or **MapleForm**. These facilities are elementary and mostly limited to the type of polynomials usually encountered in FeynCalc.

The **FortranForm** option should not be used if the expression to be written out contains a term $(-x)^n$, where n is a symbol, (which is never produced by FeynCalc, unless you enter it). The Mathematica **FortranForm** is also incapable of recognizing some elementary functions like **Exp[x]**. Therefore if you want to generate more elaborate Fortran code you may want to use *Maple*, which provides an optimization option when translating *Maple* code to Fortran, or the most sophisticated package for Fortran code generation from computer algebra systems: *Gentran* (by B. Gates and H. van Hulzen), which is available in some *Macsyma* versions. To this end the **Write2** option **FormatType** may be set to **MacsymaForm** or **MapleForm**. You may achieve a rudimentary optimization by applying **Isolate** on the expression to be written out. With the option setting **FortranForm** and **InputForm** the function **Write2** recognizes variables in **HoldForm** and writes these out with their definitions first.

Write2[*fi***, *x* = *y*]**	write the setting $x = y$ into a file *fi*
Write2[*fi***, *x* = *y*, *a* = *b*, ...]**	write the setting $x = y$ into a file *fi*

A modification of **Write**.

option name	*default value*	
FormatType	**InputForm**	language in which to write the output
D0Convention	0	which convention to use for the scalar integrals

Options for **Write2**.

With the default setting 0 of **D0Convention** the arguments of the scalar integrals are not changed when written into a Fortran program. Another possible setting is 1, which interchanges the fifth and sixth arguments of **D0** and writes the mass arguments of **A0**, **B0**, **C0** and **D0** without squares.

InputForm	write out in Mathematica form
FortranForm	write out in Fortran form
MacsymaForm	write out in *Macsyma* form
MapleForm	write out in *Maple* form

The four possible settings of the option **FormatType** for **Write2**.

The output possibilities for the two other computer algebra systems *Macsyma* and *Maple* are very limited: square brackets in Mathematica are changed to round brackets, additional replacements are for **MacsymaForm { Pi → %pi, I → %i, = → : }** and for **MapleForm { = → := }**.

The reasons to include these interface abilities are to utilize the Fortran optimization possibility of *Maple* and the excellent package Gentran for Fortran code generation.

Create a test polynomial.

```
In[232]:= tpol = z + Isolate[Isolate[2 Pi I + f[x]
            (a - b), f]]

Out[232]= z + K[2]
```

The default writes out in Mathematica **InputForm**.

```
In[233]:= Write2["test.m", test = tpol];
```

Show the content of the file.

```
In[234]:= !!test.m

Out[234]= K[1] = a - b
          K[2] = 2*I*Pi + f[x]*HoldForm[K[1]]

          test = z + HoldForm[K[2]]
```

This writes a file in Fortran format.

```
In[235]:= Write2["test.for", test = tpol,
            FormatType → FortranForm];
```

Show the content of the Fortran file.

```
In[236]:= !!test.for

Out[236]= K(1) = a - b
          K(2) = (0,2)*Pi + f(x)*K(1)
          test = z + K(2)
```

This writes a file in *Macysma* format.

```
In[237]:= Write2["test.mac", test = tpol,
          FormatType → MacsymaForm];
```

Here the *Macsyma* format can be seen.

```
In[238]:= !!test.mac

Out[238]= test : ( 2*%i*%pi + z + (a - b)*f(x) )$
```

Here we get a file with *Maple* conventions.

```
In[239]:= Write2["test.map", test = tpol,
          FormatType → MapleForm];
```

The *Maple* conventions are only slightly different.

```
In[240]:= !!test.map

Out[240]= test := ( 2*I*Pi + z + (a - b)*f(x) );
```

7.6 More on Levi-Civita Tensors

EpsEvaluate[*expr***]** evaluate Levi-Civita **Eps**

A function for total antisymmetry and linearity with respect to **Momentum** .

This is $\varepsilon^{\mu\nu\rho\sigma}(p+q)_\sigma = \varepsilon^{\mu\nu\rho(p+q)}$.

```
In[241]:= Contract[LeviCivita[m, n, r, s]
          FourVector[p + q, s]]

Out[241]= eps[m, n, r, p + q]
```

In this way linearity is enforced: $\varepsilon^{\mu\nu\rho(p+q)} = \varepsilon^{\mu\nu\rho p} + \varepsilon^{\mu\nu\rho q}$.

```
In[242]:= EpsEvaluate[%]

Out[242]= eps[m, n, r, p] + eps[m, n, r, q]
```

This does not evaluate directly to 0.

```
In[243]:= LeviCivita[a, b, c, d] /.  d → c

Out[243]= eps[a, b, c, c]
```

Like this you can always evaluate ε's.

```
In[244]:= EpsEvaluate[%]

Out[244]= 0
```

EpsChisholm[*expr***]** utilize $\gamma_\mu \varepsilon^{\mu\nu\rho\sigma} = +i\,(\gamma^\nu\gamma^\rho\gamma^\sigma - g^{\nu\rho}\gamma^\sigma - g^{\rho\sigma}\gamma^\nu + g^{\nu\sigma}\gamma^\rho)\gamma^5$

A function for the Chisholm identity.

option name	*default value*	
LeviCivitaSign	-1	sign convention for the ε-tensor

An option for **EpsChisholm**.

Use the Chisholm identity for $\varepsilon^{abcd}\,\gamma^\mu\,\gamma^a$.

```
In[245]:= EpsChisholm[LeviCivita[a, b, c, d]
          DiracMatrix[mu, a] ]

Out[245]= +I (-(g[c, d] ga[mu] ga[b] ga[5]) +
           g[b, d] ga[mu] ga[c] ga[5] -
           g[b, c] ga[mu] ga[d] ga[5] +
           ga[mu] ga[b] ga[c] ga[d] ga[5])
```

Eps[*a***, *b*, *c*, *d*]** internal representation of Levi-Civita tensors, where the arguments must have head **LorentzIndex** or **Momentum**

The internal function for Levi-Civita tensors, see also section 3.4.

This clears **$PrePrint**.

```
In[246]:= $PrePrint=.
```

This shows the internal structure.

```
In[247]:= LeviCivita[m, n, r, s]

Out[247]= Eps[LorentzIndex[m], LorentzIndex[n],
          LorentzIndex[r], LorentzIndex[s]]
```

Contracting $\varepsilon^{\mu\nu\rho\sigma}\, p^{\sigma}$ yields $\varepsilon^{\mu\nu\rho p}$.

In the internal representation the p has the head **Momentum** wrapped around it.

```
In[248]:= Contract[% FourVector[p, s]]

Out[248]= Eps[LorentzIndex[m], LorentzIndex[n],
          LorentzIndex[r], Momentum[p]]
```

7.7 Simplifications of Expressions with Mandelstam Variables

TrickMandelstam[*expr*, **{***s*, *t*, *u*, *m1*^2 + *m2*^2 + *m3*^2 + *m4*^2}]** simplifies *expr* to the shortest form removing s, t or u in each sum using $s + t + u = m_1^2 + m_2^2 + m_3^2 + m_4^2$

For tricky Mandelstam variable substitution.

This is an easy example of simplifications done by **TrickMandelstam**.

```
In[249]:= TrickMandelstam[(s + t - u) (2 mw2 - t -
            u), {s, t, u, 2 mw2}]

Out[249]= 2 s (mw2 - u)
```

The result is always given in a factorized form.

```
In[250]:= TrickMandelstam[M^2 s - s^2 + M^2 t - s t
            + M^2 u - s u, {s, t, u, 2 M^2}]

Out[250]= 2 M^2 (M^2 - s)
```

7.8 Manipulation of Propagators

FeynAmpDenominatorCombine[*expr***]**
expands *expr* w.r.t. to **FeynAmpDenominator** and combines products of **FeynAmpDenominator** into one **FeynAmpDenominator**

FeynAmpDenominatorSimplify[*expr***]**
simplifies each PropagatorDenominator in a canonical way

FeynAmpDenominatorSimplify[*expr***, sl q]**
includes some translation of momenta

FeynAmpDenominatorSimplify[*expr***, sl q1, sl q2]**
additionally removes 2-loop integrals with no mass scale

FeynAmpDenominatorSplit[*expr***]**
splits every **FeynAmpDenominator[a,b, ...]** into **FeynAmpDenominator[a]*FeynAmpDenominator[b] ...**

FeynAmpDenominatorSplit[*expr***, sl q]**
splits every **FeynAmpDenominator[a,b, ...]** into a product of two, one with sl q and other momenta, the other without sl q

PropagatorDenominatorExplicit[*expr***]**
writes out **FeynAmpDenominator[a,b, ...]** explicitly as a fraction

Functions for manipulating propagators.

7.9 Polarization Sums

PolarizationSum[*mu***, *nu***]**	$-g^{\mu\nu}$
PolarizationSum[*mu***, *nu***, *k***]**	$-g^{\mu\nu} + k^\mu k^\nu/k^2$
PolarizationSum[*mu***, *nu***, *k***, *n***]**	$-g^{\mu\nu} - k_\mu k_\nu n^2/(k\cdot n)^2 + (n_\mu k_\nu + n_\nu k_\mu)/(k\cdot n)$

Polarization sums.

This is the polarization sum for massive bosons: $\Sigma\varepsilon_\mu\varepsilon_\nu^* = -g^{\mu\nu} + k^\mu k^\nu/k^2$.

```
In[251]:= PolarizationSum[mu, nu, k]

Out[251]= -g[mu, nu] + k[mu] k[nu]/k.k
```

Here the polarization sum for gluons is given; with external momentum $n = p_1 - p_2$.

```
In[252]:= PolarizationSum[mu, nu, k, p1 - p2]

Out[252]= -g[mu, nu] - (p1.p1 - 2 p1.p2 + p2.p2) k[mu] k[nu]/(k.p1 - k.p2)^2 +

  k[nu] (p1 - p2)[mu] + k[mu] (p1 - p2)[nu]
  ----------------------------------------
                 k.p1 - k.p2
```

7.10 Permuting the Arguments of the Four-Point Function

The arguments of the Passarino-Veltman four-point function **D0** are permuted by FeynCalc into an alphabetical order. If you want a specific permutation of the 24 possible ones, you can use the function **PaVeOrder**.

PaVeOrder[*expr***]** order the arguments of **C0** and **D0** in *expr* canonically.

An ordering function for C_0 and D_0.

option name	*default value*	
PaVeOrderList	**{}**	order D_0 according to **{..., *a*, *b*, ... }**

An ordering option for **PaVeOrder**.

With the default setting of **PaVeOrder** a canonical ordering is chosen.

It is sufficient to supply a subset of the arguments.

```
In[253]:= PaVeOrder[D0[me2, me2, mw2, mw2, t, s,
            me2, 0, me2, 0],
            PaVeOrderList → {me2, me2, 0, 0}]

Out[253]= D0[me2, s, mw2, t, mw2, me2, me2, 0, 0, me2]
```

This interchanges the f and e.

```
In[254]:= PaVeOrder[D0[a, b, c, d, e, f, m12, m22,
            m32, m42],
            PaVeOrderList → {f, e}]

Out[254]= D0[a, d, c, b, f, e, m22, m12, m42, m32]
```

This shows how to permute several D_0.

```
In[255]:= PaVeOrder[D0[a, b, c, d, e, f, m12, m22,
            m32, m42] +  D0[me2, me2, mw2, mw2, t,
            s, me2, 0, me2, 0], PaVeOrderList → {
            {me2, me2, 0, 0}, {f, e}}]

Out[255]= D0[a, d, c, b, f, e, m22, m12, m42, m32] + D0[me2, s, mw2,
          t, mw2, me2, me2, 0, 0, me2]
```

8 Reference Guide for FeynCalc

■ **A0**

A0[*m*^**2]** is the scalar Passarino-Veltman one-point function.

■ **A0[***m*^**2]** is an abbreviation for $A_0(m^2) = -i\pi^{-2} \int d^D q\, (2\mu\pi)^{4-D} [q^2 - m^2]^{-1}$. ■ **A0[0]** and **A0[Small[***mass*^**2]]** give 0. ■ The following option can be given:

A0ToB0	True	replace $A_0(m^2)$ by $m^2(1 + B_0(0, m^2, m^2))$

■ **A0ToB0**

A0ToB0 is an option for **A0**. If set to **True**, $A_0(m^2)$ is replaced by $m^2(1 + B_0(0, m^2, m^2))$.

■ **B0**

B0[*pp*, m_1^**2**, m_2^**2]** is the scalar Passarino-Veltman two-point function.

■ **B0[***pp*, m_1^**2**, m_2^**2]** is an abbreviation for $B_0(p^2, m_1^2, m_2^2) = -i\pi^{-2} \int d^D q\, (2\mu\pi)^{4-D} ([q^2 - m_1^2][(q+p)^2 - m_2^2])^{-1}$. ■ **B0** is symmetric in its second and third argument. ■ The following options can be given:

BReduce	**False**	reduce **B0** in special cases to **A0**
B0Unique	**False**	replace $B_0(a, 0, a)$ by $(2 + B_0(0, a, a))$ and $B_0(0, 0, a)$ by $(1 + B_0(0, a, a))$.

■ **B00**

B00[*pp*, m_1^**2**, m_2^**2]** is the coefficient of $g^{\mu\nu}$ of the tensor integral decomposition of $B^{\mu\nu}$.

■ The following option can be given:

BReduce	**True**	reduce **B00** to **B1** and **A0**

■ If **BReduce** is set to **True** the following simplification holds:
$B_{00}(a, b, c) = 1/6(A_0(b) + (B_1(a, b, c)(a - c + b) + a/3B_0(a, b, c) + 1/6(a + b - c/3)))$.

■ **B0Unique**

B0Unique is an option for **B0**.

■ Sometimes it is useful to set this option to **True**, since only then all simplifications between different B_0 occur.

B1

B1[*pp*, m_1^2, m_2^2**]** is the coefficient of p^μ of the tensor integral decomposition of B^μ.

■ The following option can be given:

BReduce	**True**	reduce **B1** to **B0** and **A0**

■ If a variable of B_1 has head **Small** and at least one other variable does not (and is different from 0), the variable with head **Small** is set to 0. ■ If the option **BReduce** is set to **True**, **B1** simplifies in the following way, where a, b, c are m^2 with no head **Small**. The same simplifications are performed if instead of 0 a **Small** variable is supplied as an argument.
$B_1(a,b,c) = 1/2(A_0(b) - A_0(b) - (a - c + b)B_0(a,b,c))$.
$B_1(a,b,b) = -1/2B_0(a,b,b)$.
$B_1(a,a,0) = -1/2B_0(a,a,0) - 1/2$.
$B_1(a,0,a) = 1/2 - 1/2B_0(a,0,m)$.
$B_1(0,0,a) = -1/2B_0(0,0,a) + 1/4$.
$B_1(0,a,0) = -1/2B_0(0,a,0) - 1/4$.

B11

B11[*pp*, m_1^2, m_2^2**]** is the coefficient of $p^\mu\, p^\nu$ of the tensor integral decomposition of $B^{\mu\nu}$.

■ The following option can be given:

BReduce	**True**	reduce **B11** to **B1** and **A0**

■ If the option **BReduce** is set to **True**, **B11** simplifies in the following way, where a, b, c are m^2 with no head **Small**.
$B_{11}(a,b,c) = 1/(3a)(A_0(b) - 2(a - c + b)B_1(a,b,c) - aB_0(a,b,c) - 1/2(a + b - c/3))$.
$B_{11}(0,a,a) = 1/3B_0(0,a,a)$.

BReduce

BReduce is an option for **B0**, **B00**, **B1** and **B11**, determining whether reductions to lower-order **A** and **B** are done.

■ The default setting of **BReduce** is **True**.

CancelQ2

CancelQ2 is an option for **OneLoop**. If set to **True**, cancellation of all q^2 with the first propagator via $q^2 \to ((q^2 - m^2) + m^2)$ is performed, where q denotes the integration momentum.

■ With the default **True** the translation of the integration momentum in the lower order Passarino Veltman functions is done such that the third mass argument of the 4-point integral is put at position 1.

CancelQP

CancelQP is an option for **OneLoop**. If set to **True**, cancellation of $q \cdot p$ with propagators is performed, where q denotes the integration momentum.

ChiralityProjector

ChiralityProjector[+1] is an alternative input for $\omega_+ = \frac{1}{2}(1+\gamma^5)$. **ChiralityProjector[-1]** denotes $\omega_- = \frac{1}{2}(1-\gamma^5)$.

■ **ChiralityProjector[1]** is identical to **DiracMatrix[6]**. **ChiralityProjector[-1]** is identical to **DiracMatrix[7]**. The internal representation is **DiracGamma[6]** and **DiracGamma[7]**.
■ See also: **DiracMatrix**, **DiracGamma**.

C0

C0[p_{10}**,** p_{12}**,** p_{20}**,** m_1**^2,** m_2**^2,** m_3**^2]** is the scalar Passarino-Veltman three-point function. The first three arguments of **C0** are the scalar products $p_{10} = p_1^2$, $p_{12} = (p_1 - p_2)^2$, $p_{20} = p_2^2$.

■ **C0[**p_{10}**,** p_{12}**,** p_{20}**,** m_1**^2,** m_2**^2,** m_3**^2]** is an abbreviation for $C_0 = -i\pi^{-2} \int d^D q\, (2\mu\pi)^{4-D} ([q^2 - m_1^2][(q+p_1)^2 - m_2^2][(q+p_2)^2 - m_3^2])^{-1}$. ■ A standard representative of the six equivalent argument permutations is chosen.
■ See also: **PaVeOrder**.

Collect2

Collect2[*expr***,** x**]** collects together terms which are not free of any occurrence of x.

Collect2[*expr***, {**x_1**,** x_2**, ...}]** collects together terms which are not free of $x_1, x_2, \ldots$.

■ The following option can be given.

ProductExpand	**False**	expand products in *expr* free of x (x_1, x_2, ...)
IsolateHead	**False**	isolate with respect to **{**x_1**,** x_2**, ...}**
IsolateSplit	**442**	the limit before **Isolate** splits

■ See also: **Isolate**.

Combine

Combine[*expr***]** puts terms in a sum over a common denominator. **Combine** is similar to **Together**, but works better on certain polynomials with rational coefficients.

■ The following option can be given:

ProductExpand	**False**	expand products

■ CombineGraphs

CombineGraphs is an option for **OneLoopSum**.

■ The utilization of this option may speed up FeynCalc. But depending on the available memory there is a turning point where FeynCalc becomes quite inefficient when forced to calculate too may complicated diagrams at once.

■ Contract

Contract[*expr***]** contracts pairs of Lorentz indices in **expr**.

■ For the contraction of two Dirac matrices **DiracSimplify** has to be used. ■ The result of **Contract** is not fully expanded. ■ The following options can be given:

EpsContract	**False**	contract products of Levi-Civita (Eps) tensors via the determinant formula
Expanding	**True**	expand all sums containing Lorentz indices
Factoring	**False**	factor the result canonically

■ Examples:

```
Contract[ FourVector[p, mu]^2 ] → p.p.
Contract[ FourVector[p + q, mu] FourVector[p' + q', mu] ]//
ExpandScalarProduct → p.p' + p.q' + q.p' + q.q'.
Contract[ LeviCivita[m, n, r, a] LeviCivita[m, n, r, b], EpsContract → True
] → -6 g[a, b].
Contract[ Pair[ LorentzIndex[mu, Dim], LorentzIndex[mu, Dim] ] ] → Dim.
Contract[ Pair[ LorentzIndex[mu, D - 4], LorentzIndex[mu] ] ] → 4.
Contract[ Pair[ LorentzIndex[mu, D - 4], LorentzIndex[mu] ] ] → 0.
Contract[ Pair[ LorentzIndex[mu, D - 4], LorentzIndex[mu, D - 4] ] ]  →
-4 + D.
Contract[ f[ ___, LorentzIndex[mu], ___ ] MetricTensor[mu, al] ]  →  f[
___, LorentzIndex[al], ___ ].
Contract[ f[ ___, LorentzIndex[mu], ___ ] FourVector[p, mu] ] →  f[ ___,
FourVector[p], ___ ].
Contract[ DiracMatrix[mu, Dimension → D] MetricTensor[mu, al] ] →
DiracGamma[LorentzIndex[nu]].
Contract[ DiracGamma[LorentzIndex[mu, D-4], D-4] MetricTensor[mu, al] ] →
0.
```

■ If big expressions are contracted and substitutions for the resulting scalar products are made, it is best not to do these replacements after contraction, but to set the values of the relevant scalar products *before* invoking **Contract**; in this way the intermediate expression swell is minimized.
■ See also: **ExpandScalarProduct**.

■ D0

D0[p_{10}**,** p_{12}**,** p_{23}**,** p_{30}**,** p_{20}**,** p_{13}**,** m_1**^2,** m_2**^2,** m_3**^2,** m_4**^2]** is the scalar Passarino-Veltman four-point function. The first six arguments of **D0** are the scalar products $p_{10} = p_1^2,\ p_{12} = (p_1 - p_2)^2,\ p_{23} = (p_2 - p_3)^2,\ p_{30} = p_3^2,\ p_{20} = p_2^2,\ p_{13} = (p_1 - p_3)^2$.

■ **D0[**p_{10}**,** p_{12}**,** p_{23}**,** p_{30}**,** p_{20}**,** p_{13}**,** m_1**^2,** m_2**^2,** m_3**^2,** m_4**^2]** is an abbreviation for $D_0 = -i\pi^{-2} \int d^4q\,([q^2 - m_1^2][(q + p_1)^2 - m_2^2][(q + p_2)^2 - m_3^2][(q + p_3)^2 - m_4^2])^{-1}$.
■ See also: **PaVe**, **PaVeOrder**.

■ D0Convention

D0Convention is an option for **Write2**. Possible settings are 0 or 1. With the last setting the fifth and sixth arguments of **D0** are interchanged and all (internal) mass arguments of the scalar Passarino Veltman integrals are given square free.

■ See also: **Write2**, **D0**.

■ DB0

DB0[p_{10}**,** m_0**^2,** m_1**^2]** is the derivative $\partial B_0(p^2, m_0^2, m_1^2)/\partial p^2$ of the two-point function B_0.

■ See also: **B0**.

■ DenominatorOrder

DenominatorOrder is an option for **OneLoop**, if set to **True** the **PropagatorDenominator** in **FeynAmpDenominator** will be ordered in a standard way.

■ You may want to set this option to **False** when checking hand calculations.

■ Dimension

Dimension is an option for **DiracMatrix**, **DiracSlash**, **FourVector**, **MetricTensor**, **OneLoop** and **ScalarProduct**.

■ The setting of **Dimension** may be **4**, **dim** or **dim-4**, where **dim** must be a Mathematica **Symbol** .

■ DiracGamma

DiracGamma[*x***,** *optdim***]** is the head of all Dirac matrices and Feynman slashes $\not{p}$ ($= \gamma_\mu p^\mu$) in the internal representation.

A four-dimensional Dirac matrix γ_μ is **DiracGamma[LorentzIndex[***mu***]]**, a four-dimensional Feynman slash is **DiracGamma[Momentum[***p***]]**.

γ_5 is represented as `DiracGamma[5]`, the helicity projectors $\gamma_6 = (1 + \gamma_5)/2$ and $\gamma_7 = (1 - \gamma_5)/2$ as **DiracGamma[6]** and **DiracGamma[7]** respectively.

For other than four dimensions an additional argument is necessary: **DiracGamma[LorentzIndex[***mu***,** *Dim***], ***Dim* **]** and **DiracGamma[Momentum[***q***,** *Dim***], ***Dim* **]**.

■ For standard input **DiracMatrix** and **DiracSlash** are more convenient. ■ Note that **DiracGamma[***exp***,** *Dim***]** projects out the smaller dimension of the objects **exp** and **Dim**. There are special relationships if **Dim** takes the form **D-4**, e.g., **DiracGamma[LorentzIndex[***mu***,** *D***-4],***4* **]** → **0**.

■ DiracMatrix

DiracMatrix[*mu***]** is an input function for a Dirac matrix. A product of Dirac matrices $\gamma^\mu\gamma^\nu\gamma^\rho \ldots$ is entered as **DiracMatrix[mu, nu, ro, ...]** or equivalently as **DiracMatrix[mu] . DiracMatrix[nu] . DiracMatrix[ro] . ...**

γ^5 may be entered as **DiracMatrix[5]**, $\gamma^6 = (1+\gamma^5)/2$ as **DiracMatrix[6]** and $\gamma^7 = (1-\gamma^5)/2$ as **DiracMatrix[7]**

■ The following option can be given:

Dimension	4	space-time dimension

■ γ^5, γ^6 and γ^7 are defined purely in four dimensions. ■ See also: **DiracGamma**, **DiracSlash**, **ChiralityProjector**, **$BreitMaison**.

■ DiracOrder

DiracOrder[*expr***,** *orderlist***]** orders the Dirac matrices in **expr** according to *orderlist*.

DiracOrder[*expr***]** orders the Dirac matrices in **expr** alphabetically.

■ γ_5, γ_6 and γ_7 are not ordered; use `DiracSimplify` to push them all to the right. ■ Example:
DiracOrder[DiracSlash[a, q, a, p]] → **-2 a.a p.q + 2 a.q gs[a] . gs[p] + a.a gs[p] . gs[q]**.
DiracOrder[DiracSlash[p], DiracSlash[q], {q,p}] → **2 p.q - gs[q] . gs[p]**.
■ This function is just the implementation of the anticommutator relation for Dirac matrices.
See also: **DiracSimplify**.

■ DiracSimplify

DiracSimplify[*expr***]** simplifies products of Dirac matrices in **expr**. Double Lorentz indices and four vectors are contracted. The Dirac equation is applied. All **DiracMatrix[5]**,

DiracMatrix[6] and **DiracMatrix[7]** are moved to the right. The order of the Dirac matrices is not changed.

■ $(\not{p} - m)\,u(p) = 0$, $(\not{p} + m)\,v(p) = 0$ and $\overline{u}(p)(\not{p} - m) = 0$, $\overline{v}(p)(\not{p} + m) = 0$ are represented by: **DiracSimplify[(DiracSlash[p] - m) . Spinor[p, m]]** → 0, **DiracSimplify[(DiracSlash[p] + m) . Spinor[-p,m]]** → 0, **DiracSimplify[Spinor[p, m] . (DiracSlash[p] - m)]** → 0, **DiracSimplify[Spinor[-p,m] . (DiracSlash[p] + m)]** → 0
■ Examples:
DiracSimplify[DiracMatrix[mu, mu]] → 4.
DiracSimplify[DiracSlash[p, p]] → **p.p**.
DiracSimplify[DiracMatrix[al, be, al]] → **-2 ga[be]**.
DiracSimplify[DiracMatrix[al, be, al, Dimension→D]]//Factor → **(2-D) ga[be]**.
DiracSimplify[DiracSlash[p], (DiracSlash[-q]+m), DiracSlash[p]] → **gs[q] p.p - 2 gs[p] p.q + p.p m**.
DiracSimplify[DiracMatrix[5, mu]] → **-ga[mu] . ga[5]**.
DiracSimplify[DiracMatrix[6, mu]] → **ga[mu] . ga[7]**.
■ See also: **DiracOrder**.

■ DiracSlash

DiracSlash[*p***]** is an input function for a Feynman slash $\not{p} = \gamma^\mu p_\mu$. A product of slashes may be entered by **DiracSlash[p, q, ...]** or **DiracSlash[p], DiracSlash[q], ...** .

■ The following option can be given:

Dimension	4	space-time dimension

■ The internal representation of a four-dimensional **DiracSlash[p]** is **DiracGamma[Momemtum[***p***]]**, a **D**-dimensional **DiracSlash[** *p*, *Dimension* → **D]** is transformed into **DiracGamma[Momemtum[p, D], D]**.
■ See also: **DiracGamma**, **DiracMatrix**.

■ DiracTrace

DiracTrace[*expr***]** is the head of a Dirac trace. Whether the trace is evaluated depends on the option **DiracTraceEvaluate**. The argument **expr** may be a product of Dirac matrices or slashes separated by ".".

■ The following options can be given:

DiracTraceEvaluate	**False**	evaluating the trace
LeviCivitaSign	**-1**	sign convention for the ε-tensor
Factoring	**False**	factor the result
Mandelstam	**{}**	if set to **{***s*, *t*, *u*, m_1**^2 +** m_2**^2 +** m_3**^2 +** m_4**^2}**, **TrickMandelstam** will be used

PairCollect	**True**	collect the results with respect to products of **Pair**

■ Examples:
DiracTrace[DiracMatrix[al, be]] → **4 g[al, be]**.
DiracTrace[DiracSlash[p, q]] → **4 p.q**.
DiracTrace[DiracMatrix[mu, al, be, mu, Dimension→D]] → **4 D g[al, be]**.
DiracTrace[DiracMatrix[a, b, c, d, 5]] → **-4 I Eps[a, b, c, d]**.
DiracTrace[MetricTensor[al, be] DiracMatrix[si, al, ro, si]] → **16 g[be, ro]**.
DiracTrace[DiracSlash[p - q] . (DiracSlash[q] + m) + DiracSlash[k, 2 p]] → **8 k.p + 4 p.q - 4 q.q**.
With **PP=DiracSlash[p']**; **P=DiracSlash[p]**; **MU=DiracMatrix[mu]**; **K=DiracSlash[k]**; **NU=DiracMatrix[nu]**: **DiracTrace[(PP+m).MU.(P+K+m).NU.(P+m).NU.(P+K+m).MU/16]/.m^2→m2/.m^4→m4** → **4 m4 + k.p (4 m2 + 2 k.p') + k.p' (-4 m2 + 2 p.p) + k.k (4 m2 - p.p') - 3 m2 p.p' + p.p p.p'**.
■ To replace scalar products two possibilities exist: either set the corresponding **ScalarProduct** *before* calculating the trace: **ScalarProduct[p, q] = t/2;** (**DiracTrace[p . q]**) → **2 t**, which is preferable, or substitute afterwards: (**DiracTrace[DiracSlash[a, b]]/.ScalarProduct[a, b]→s/2**) → **2 s**.
■ See also: **DiracMatrix**, **DiracSlash**, **ScalarProduct**, **TrickMandelstam**.

■ DiracTraceEvaluate

DiracTraceEvaluate is an option for **DiracTrace**. If set to **False**,**DiracTrace** remains unevaluated.

■ The resaon for this option is that **OneLoop** needs traces in unevaluated form.

■ Eps

Eps[*a***, *b*, *c*, *d***]** is the head of the totally antisymmetric four-dimensional epsilon (Levi-Civita) tensor. The *a*, *b*, *c*, *d must* have head **LorentzIndex** or **Momentum**.

■ All entries are transformed to four dimensions. ■ For user-friendly input of an **Eps** just with **LorentzIndex**' use **LeviCivita**. ■ For $\varepsilon^{\mu\nu\rho\sigma} q_\sigma$ the compact notation $\varepsilon^{\mu\nu\rho q}$ is used, i.e.: **Eps[LorentzIndex[mu], LorentzIndex[nu], LorentzIndex[ro], Momentum[q]]**. ■ **Eps** is just a head not having any functional properties. In order to exploit linearity ($\varepsilon^{\mu\nu\rho(p+q)} \to \varepsilon^{\mu\nu\rho p} + \varepsilon^{\mu\nu\rho q}$) and the total antisymmetric property of the Levi-Civita tensor use **EpsEvaluate**.

■ EpsChisholm

EpsChisholm[*expr***]** substitutes for a gamma matrix contracted with a Levi Civita tensor (**Eps**) the Chisholm identity: $\gamma_\mu \varepsilon^{\mu\nu\rho\sigma} = +i(\gamma^\nu \gamma^\rho \gamma^\sigma - g^{\nu\rho}\gamma^\sigma - g^{\rho\sigma}\gamma^\nu + g^{\nu\sigma}\gamma^\rho)\gamma^5$.

■ With the option **LeviCivitaSign** the sign of the right hand side of the equation above can be altered. ■ The following option can be given:

LeviCivitaSign	-1	sign convention

■ **EpsContract**

EpsContract is an option for **Contract** specifying whether Levi-Civita tensors **Eps** will be contracted, i.e., products of two **Eps** are replaced via the determinant formula.

■ **EpsEvaluate**

EpsEvaluate[*expr***]** applies total antisymmetry and linearity (with respect to **Momentum**) to all Levi-Civita tensors (**Eps**) in *expr*.

■ **EvaluateDiracTrace**

EvaluateDiracTrace[*expr***]** evaluates **DiracTrace** in *expr*.

■ **Expanding**

Expanding is an option for **Contract** specifying whether expansion will be done in **Contract**. If set to **False**, not all Lorentz indices might get contracted.

■ **ExpandScalarProduct**

ExpandScalarProduct[*expr***]** expands scalar products in *expr*.

■ Example:
ExpandScalarProduct[ScalarProduct[p-q,r+2 s]] → **p.r + 2 p.s - q.r - 2 q.s**.
■ At the interal level **ExpandScalarProduct** expands actually everything with head **Pair**. ■ Since a four-vector has head **Pair** internally also, **ExpandScalarProduct[FourVector[p - 2 q, mu]]** → **p[mu] - 2 q[mu]**.

■ **Factor2**

Factor2[*expr***]** factors a polynomial in a standard way. **Factor2** works better than **Factor** on polynomials involving rationals with sums in the denominator.

■ In general it is better to use **Factor2** in Mathematica 2.0.

■ **Factoring**

Factoring is an option for **Contract**, **DiracTrace**, **DiracSimplify** and **OneLoop**. If set to **True** the result will be factored, using **Factor2**.

■ **FeynAmp**

FeynAmp[*name*, *amp*, *q***]** is the head of the Feynman amplitude given by FeynArts. The first argument **name** is for bookkeeping, **amp** is the analytical expression for the amplitude, and **q** is the integration variable. In order to calculate the amplitute replace **FeynAmp** by **OneLoop**. In the output of FeynArts *name* has the head **GraphName**.

■ See also: **GraphName**, Guide to FeynArts.

■ **FeynAmpDenominator**

FeynAmpDenominator[PropagatorDenominator[...], PropagatorDenominator[...], ...] is the head of the denominators of the propagators, i.e., **FeynAmpDenominator[x]** is the representation of $1/x$.

■ Example:
$1/([q^2-m_1^2][(q+p_1)^2-m_2^2])$ is represented as **FeynAmpDenominator[PropagatorDenominator[***q*, *m1***], PropagatorDenominator[***q***+p1,***m2***]]**.
■ See also: **PropagatorDenominator**.

■ **FeynAmpList**

FeynAmpList[*info* **...]***FeynAmp***[...],** *FeynAmp***[...], ...]** is the head of a list of **FeynAmp** in the result of FeynArts.

■ See also: Guide to FeynArts.

■ **FeynCalcForm**

FeynCalcForm[*expr***]** changes the printed output of *expr* to an easy to read form. The default setting of **$PreRead** is **$PreRead = FeynCalcForm**, which forces to display everything after applying **FeynCalcForm**.

■ **Small**, **Momentum** and **LorentzIndex** are set to **Identity** by **FeynCalcForm**. ■ **PaVe** are abbreviated. ■ The action of **FeynCalcForm** is: **DiracMatrix[al]** → **ga[al]**.
DiracSlash[p] → **gs[p]**.
FeynAmpDenominator[PropagatorDenominator[q,m1], PropagatorDenominator[q+p,m2]] → **1/(q^2 - m1^2) ((p + q)^2 - m2^2)**.
FourVector[p, mu] → **p[mu]**.

FourVector[p-q, mu] → **(p - q)[mu]**.
GellMannMatrix[a] → **la[a]**.
GellMannTrace[x] → **tr[x]**.
DiracTrace[x] → **tr[x]**.
LeviCivita[a, b, c, d] → **eps[a, b, c, d]**.
MetricTensor[mu, nu] → **g[mu nu]**.
Momentum[Polarization[p]] → **ep[p]**.
Conjugate[PolarizationVector[k, mu]] → **ep(*)[k, mu]**.
PolarizationVector[p, mu] → **ep[p][mu]**.
ScalarProduct[p, q] → **p.q**.
Spinor[p, m] → **u[p, m]**.
Spinor[-p, m] → **v[p, m]**.
QuarkSpinor[p, m] → **u[p, m]**.
QuarkSpinor[-p, m] → **v[p, m]**.
SU3F[i, j, k] → **f[i, j, k]**.

■ **FinalSubstitutions**

FinalSubstitutions is an option for **OneLoop** and **OneLoopSum**. All substitutions given to this opton will be performed at the end of the calculation.

■ Example: **FinalSubstitutions** → **{mw^2** → **mw2, B0** → **B0R}**.

■ **FourVector**

FourVector[*p*, *mu***]** is the input for a four vector p_μ.

■ **FourVector[***p*, *mu***]** is directly transformed to the internal representation: **Pair[Momentum[***p***], LorentzIndex[***mu***]]**. ■ The following option can be given:

Dimension	4	space-time dimension

■ See also: **LorentzIndex**, **Momentum**, **Pair**.

■ **FreeQ2**

FreeQ2[*expr*, **{***form1*, *form2*, **...}]** yields **True** if *expr* does not contain any occurrence of *form*1, *form2*, ... **FreeQ2[***expr*, *form***]** is the same as **FreeQ[***expr*, **{***form1*, *form2*, **...}]**.

■ Stephen Wolfram pointed out that you can use alternatively **FreeQ[***expr*, *form1* **||** *form2* **|| ...]**.

■ **GellMannMatrix**

GellMannMatrix[*a***]** is the Gell-Mann matrix λ_a. A product of Gell-Mann matrices may be entered as **GellMannMatrix[a, b, ...]** or as **GellmannMatrix[a] .**

GellMannMatrix[b] . .... **GellMannMatrix[1]** denotes the unit-matrix in color space.

■ See also: **SU3Delta**, **SU3F**, **GellMannTrace**.

■ GellMannTrace

GellMannTrace[*expr***]** calculates the trace of *expr*. All color indices should occur twice and *expr* must be a product of **SU3F**, **SU3Delta** and **GellMannMatrix**.

■ The Cvitanovic algorithm is used. ■ Examples: **[GellMannTrace[GellMannMatrix[i.i]]** → 16. **GellMannTrace[GellMannMatrix[a . b . c] SU3F[a, b, c]]** → 48 I. **GellMannTrace[GellMannMatrix[a . c . e . d] SU3F[a, b, e] SU3F[b, c, d]]** → 0. **GellMannTrace[GellMannMatrix[1]]** → 3.

■ GraphName

GraphName[*a***, *b*, *c*, *d***]** is the first argument of **FeynAmp** given by FeynArts. It may be used also as first argument of **OneLoop**. The arguments *a*, *b*, *c*, *d* indicate information of the graph under consideration.

■ InitialSubstitutions

InitialSubstitutions is an option for **PaVeReduce** and **OneLoop**. All substitutions hereby indicated will be performed at the beginning of the calculation. Energy momentum conservation may be especially indicated in the setting.

■ Example: **InitialSubstitutions** → **{CW^2** → **1 - SW^2, k2** → **- k1 + p1 + p2 }**.

■ Isolate

Isolate[*expr***, {***x*$_1$**, *x*$_2$**, ... }]** substitutes **K[***i***]** for all subsums in *expr* which are free of any occurrence of *x*$_1$, *x*$_2$, ..., if **Length[***expr***]>0**.

Isolate[*expr***]** substitutes an abbreviation **K[***i***]** in **HoldForm** for *expr*, if **Length[***expr***]>0**.

■ The following options can be given:

IsolateHead	**K**	the head of the abbreviations
IsolateSplit	**442**	a limit beyound which **Isolate** splits the expression in two sums

■ **IsolateSplit** is the maximum of the characters of the **FortranForm** of the expression being isolated by **Isolate**. The default setting inhibits **Write2** from producing too many continuation

lines when writing out in **FortranForm**. ■ The result of **Isolate** can always be recovered by **MapAll[ReleaseHold, result]**. ■ Example: **Isolate[a[z] (b+c) + d[f] (x+y), {a,d}]** → **a[z] K[1] + d[f] K[2]**.
■ See also: **Write2**.

■ **IsolateHead**

IsolateHead is an option for **Isolate**.

■ **IsolateSplit**

IsolateSplit is an option for **Isolate**.

■ **K**

K[*i***]** are abbreviations which may result from **PaVeReduce**, depending on the option **IsolateHead**. The **K[***i***]** are returned in **HoldForm** and may be recovered by **ReleaseHold**.

■ **LeptonSpinor**

LeptonSpinor[*p***,** *mass***]** specifies a Dirac spinor. Which of the spinors $u, v, \overline{u}$ or $\overline{v}$ is understood, depends on the sign of the mass argument and the relative position of **DiracSlash[p]**: **LeptonSpinor[***p***,** *mass***]** is that spinor which yields **mass*LeptonSpinor[***p***,** *mass***]** if the Dirac equation is applied to **DiracSlash[***p***] . LeptonSpinor[***p***,** *mass***]** or **LeptonSpinor[***p***,** *mass***] . DiracSlash[***p***]**.

If a spinor is multiplied by a Dirac matrix or another spinor, the multiplication operator "." must be used.

■ See also: **DiracSimplify**.

■ **LeviCivita**

LeviCivita[*mu***,** *nu***,** *ro***,** *si***]** is an input function for the totally antisymmetric Levi-Civita tensor.

■ **LeviCivita[mu, nu, ro, si]** transforms to the internal representation **Eps[LorentzIndex[mu], LorentzIndex[nu], LorentzIndex[ro], LorentzIndex[si]]**. ■ For simplification of Levi-Civita tensors use **EpsEvaluate**.
■ See also: **Eps**, **EpsEvaluate**.

■ **LeviCivitaSign**

LeviCivitaSign is an option for **DiracTrace**. The possible settings are (+1) or (-1). This option determines the sign convention of the result of $tr(\gamma^a \gamma^b \gamma^c \gamma^d \gamma^5)$.

■ **LorentzIndex**

LorentzIndex[*mu*, *optdim***]** is the head of Lorentz indices. The internal representation of a four-dimensional μ is **LorentzIndex[mu]**. For other than four dimensions enter **LorentzIndex[***mu*, *dim***]**.

■ **LorentzIndex[***mu*, **4]** simplifies to **LorentzIndex[mu]**.

■ **MacsymaForm**

MacsymaForm is an option for **FormatType** in **Write2**.

■ See also: **Write2**, **MapleForm**.

■ **Mandelstam**

Mandelstam is an option for **DiracTrace**, **OneLoop** and **TrickMandelstam**. A typical setting is **Mandelstam** → **{s, t, u, m1^2 + m2^2 + m3^2 + m4^2}**, which stands for $s + t + u = m_1^2 + m_2^2 + m_3^2 + m_4^2$.

■ **MapleForm**

MapleForm is an option for **Write2**.

■ See also: **Write2**, **MacsymaForm**.

■ **MetricTensor**

MetricTensor[*mu*, *nu***]** is the input for a metric tensor $\mathbf{g}^{\mu\nu}$.

■ The following option can be given:

Dimension	4	space-time dimension

■ See also: **Contract**, **LorentzIndex**, **Pair**.

■ **Momentum**

Momentum[*p***, *optdim*]** is the head of a momentum in the internal representation. A four-dimensional momentum **p** is represented by **Momentum[p]**. For other than four dimensions an extra argument must be given: **Momentum[***q***, *dim*]**.

- **Momentum[***p***, *4*]** is automatically transformed to **Momentum[***p***]**.

NumericalFactor

NumericalFactor[*expr***]** gives the numerical factor of *expr*.

OneLoop

OneLoop[*name***, *q*, *amplitude***]** calculates the one-loop Feynman diagram **amplitude**. The argument *q* denotes the integration variable, i.e., the loop momentum.

- The following options can be given:

ReduceToScalars	**False**	reduce to B_0, C_0, D_0
DenominatorOrder	**False**	order the entries of **FeynAmpDenominator**
Dimension	**True**	dimension of integration
FinalSubstitutions	**{}**	substitutions done at the end of the calculation
Factoring	**False**	factor the result
InitialSubstitutions	**{}**	substitutions done at the beginning of the calculation
Mandelstam	**{}**	indicate the Mandelstam relation
Prefactor	**1**	additional prefactor of the amplitude
CancelQ2	**True**	cancel q^2
CancelQP	**False**	cancel $q \cdot p$
ReduceGamma	**False**	eliminate γ_6 and γ_7
SmallVariables	**{}**	a list of masses, which will get wrapped around the head **Small**
WriteOut	**True**	write out a result file **name.m**

- Energy momentum conservation may be given as rule of **InitialSubstitutions**.

OneLoopResult

OneLoopResult[*name***]** is the variable in the result file written out by **OneLoop** to which the corresponding result is assigned. *name* is constructed from the first argument of **OneLoop**.

OneLoopSum

OneLoopSum[FeynAmp[...], FeynAmp[...], ...] calculates a list of Feynman amplitudes by replacing **FeynAmp** step by step by **OneLoop** and sums the result.

■ The following options can be given:

CombineGraphs	**{}**	which amplitudes to sum before invoking **OneLoop**
FinalSubstitutions	**{}**	substitutions done at the end of the calculation
IsolateHead	**K**	**Isolate** the result
Mandelstam	**{}**	use the Mandelstam relation
Prefactor	**1**	multiply the result by a pre-factor
ReduceToScalars	**True**	reduce the summed result to scalar integrals
SelectGraphs	**All**	which amplitudes to select
WriteOutPaVe	**False**	write out the reduced **PaVe**

Possible settings for **CombineGraphs** and **SelectGraphs** are lists of integers. For indicating a range of graphs also a list **{i, j}** instead of a single integer may be provided.

■ **Pair**

Pair[*a*, *b***]** is the head of a special pairing used in the internal representation. The arguments *a* and *b* may have heads **LorentzIndex** or **Momentum**. If both *a* and *b* have head **LorentzIndex**, the metric tensor is understood. If *a* and *b* have head **Momentum**, a scalar product is meant. If one of *a* and *b* has head **LorentzIndex** and the other head **Momentum**, a Lorentz vector p^μ is understood.

■ **Pair** has only one functional definition: any integers multiplied with *a* or *b* will be pulled out.
■ See also: **FourVector**, **LorentzIndex**, **MetricTensor**, **ExpandScalarProduct**, **ScalarProduct**.

■ **PairCollect**

PairCollect is an option for **Contract**.

■ See also: **Pair**, **ScalarProduct**, **Momentum**.

■ **PaVe**

PaVe[*i*, *j*, ..., *plist*, *mlist***]** denotes the Passarino-Veltman integrals. The length of the mass list *mlist* indicates if a one-, two-, three- or four-point integral is understood. The first set of arguments *i, j,...* signifies that the coefficient of $p_i^\mu\, p_j^\nu, \ldots$ of the tensor integral decomposition is meant, where $p_0^\mu\, p_0^\nu = g^{\mu\nu}$. Joining *plist* and *mlist* gives the same conventions as for **A0**, **B0**, **C0** and **D0**.

■ For the corresponding arguments of **PaVe** the special cases **A0**, **B0**, **C0**, **D0**, **B1**, **B00**, **B11** are returned.

■ **PaVeOrder**

PaVeOrder[*expr***]** brings all arguments of **C0** and **D0** into a canical order.

■ The following option can be given:

PaVeOrderList	**{}**	order according to a list of arguments

■ **PaVeOrderList**

PaVeOrderList is an option for **PaVeOrder** allowing to specify a specific order of the **D0** functions.

■ Possible settings are a sublist of the arguments of a **D0**, or a list of such lists.

■ **PaVeReduce**

PaVeReduce[*expr***]** reduces Passarino-Veltman integrals **PaVe** to scalar integrals **B0**, **C0** and **D0**, depending on the option **BReduce** eventually also **A0**, **B1**, **B00** and **B11**.

■ The class of invariant Passarino-Veltman integrals which can be currently reduced consists of all coefficients of the Lorentz invariant decomposition of $B^{\mu}, B^{\mu\nu}, C^{\mu}, C^{\mu\nu}, C^{\mu\nu\rho}, D^{\mu}, D^{\mu\nu}, D^{\mu\nu\rho}, D^{\mu\nu\rho\sigma}$.
■ The following options can be given:

IsolateHead	**False**	use **Isolate**
Mandelstam	**{}**	Mandelstam relation, e.g., **{s, t, u, 2 mw^2}**

■ See also: **BReduce**, **K**, **PaVe**.

■ **Polarization**

Polarization[*p***, *optarg*]** is the head of a polarization momentum$\varepsilon(p)$. **Polarization** must always occur inside **Momentum**. The full internal representation of $\varepsilon(p)$ is **Momentum[Polarization[p]]**. With this notation transversality of polarization vectors is provided.

■ **Polarization[***p***, -1]** stands for the complex conjugate $\varepsilon(p)^*$.
■ See also: **PolarizationVector**.

■ **PolarizationSum**

PolarizationSum[*mu***, *nu*, ...]** defines different polarization sums.

■ **PolarizationSum** does not calculate any polarization sum, it is just an abbreviation function. ■ **PolarizationSum[***mu***, *nu***]**$= -g_{\mu\nu}$. ■ **PolarizationSum[***mu***, *nu***, *k***]**$= -g_{\mu\nu} + k_\mu k_\nu/k^2$.
■ **PolarizationSum[***mu***, *nu***, *k***, *n***]**$= -g_{\mu\nu} - k_\mu k_\nu n^2/(k\cdot n)^2 + (n_\mu k_\nu + n_\nu k_\mu)/(k\cdot n)$, where n_μ denotes an external four vector.

■ **PolarizationVector**

PolarizationVector[*p***, *mu***]** is an input function for a polarization vector $\varepsilon(k)^\mu$.

■ **Conjugate{PolarizationVector[***p***, *mu***]}** is the input for $\varepsilon^*_\mu(k)$. ■ The internal representation of **PolarizationVector[***p***, *mu***]** is **Pair[Momentum[Polarization[***p***]],**

LorentzIndex[*mu***]]**. ■ The internal representation of **Conjugate{PolarizationVector[***p***, *mu*]}** is **Pair[Momentum[Polarization[***p***,-1]], LorentzIndex[***mu***]]**.
■ See also: **Polarization**.

■ **Prefactor**

Prefactor is an option for **OneLoop** and **OneLoopSum**. If set as option of **OneLoop**, the amplitude is multiplied by **Prefactor** before calculation; if specified as option of **OneLoopSum**, it appears in the final result as a global factor.

■ A possible setting is $1/(1-D)$ for calculating the transvere part of self energies. The option **Dimension** of **OneLoop** must then be set to D.

■ **ProductExpand**

ProductExpand is an option for **Collect2** and **Combine**.

■ **PropagatorDenominator**

PropagatorDenominator[*q***, *m*]** is the denominator of a propagator, i.e., $(q^2 - m^2)$.

PropagatorDenominator[*q***]** evaluates to **PropagatorDenominator[***q***, 0]**.

■ If q is supposed to be D-dimensional enter: **PropagatorDenominator[Momentum[q,D], m]**. ■ **PropagatorDenominator** must always occur only as an argument of **FeynAmpDenominator**.

■ **QuarkSpinor**

QuarkSpinor[*p***, *mass*]** specifies a Dirac spinor. Which of the spinors $u, v, \overline{u}$ or $\overline{v}$ is understood, depends on the sign of the mass argument and the relative position of **DiracSlash[p]**: **QuarkSpinor[***p***, *mass*]** is that spinor which yields **mass*QuarkSpinor[***p***, *mass*]** if the Dirac equation is applied to **DiracSlash[***p***] . QuarkSpinor[***p***, *mass*]** or **QuarkSpinor[***p***, *mass*] . DiracSlash[***p***]**.

If a spinor is multiplied by a Dirac matrix or another spinor, the multiplication operator "." must be used.

■ See also: **DiracSimplify**.

■ **ReduceGamma**

ReduceGamma is an option for **OneLoop** determining whether γ_6 and γ_7 are removed by their definitions $1/2(1+\gamma_5)$ and $1/2(1-\gamma_5)$.

■ This option may be needed for certain standard matrixelements.

■ **ReduceToScalars**

ReduceToScalars is an option for **OneLoop** and **OneLoopSum** that specifies whether the result is reduced to scalar integrals.

■ Depending on the option **BReduce** the Passarino-Veltman functions **B1**, **B11**, **B00** and **B11** may also remain.
■ See also: **PaVeReduce**.

■ **ScalarProduct**

ScalarProduct[*p, q***]** is the input for a scalar product.

■ The following option can be given:

Dimension	4	space-time dimension

■ The internal representation is: **Pair[Momentum[p], Momentum[q]]**. ■ Scalar products may be set, e.g., **ScalarProduct[a, b] = c**; but **a** and **b** must not contain sums.
■ See also **ExpandScalarProduct**.

■ **SelectGraphs**

SelectGraphs is an option for **OneLoopSum**. The default setting is **All**. It may be set to a list indicating that only a subclass of all graphs supplied to **OneLoopSum** should be calculated.

■ **SetMandelstam**

SetMandelstam[*s, t, u, p_1, p_2, p_3, p_4, m_1, m_2, m_3, m_4***]** defines the Mandelstam variables $s = (p_1 + p_2)^2$, $t = (p_1 + p_3)^2$, $u = (p_1 + p_4)^2$ and sets the p_i on-shell: $p_i^2 = m_i^2$, where the p_i satisfy $p_1 + p_2 + p_3 + p_4 = 0$.

■ If $p_3 = -k_1$ and $p_4 = -k_2$, i.e., $p_1 + p_2 = k_1 + k_2$, the input is: **SetMandelstam[s, t, u,** p_1, p_2, **-** k_1, **-** k_2, m_1, m_2, m_3, m_4**]**

■ **SetStandardMatrixElements**

SetStandardMatrixElements[{*sma1* → *abb1***}, {***sma2* → *abb2***}, ... ,***enmoconrule* **]** defines the standard matrix elements *sma*1, *sma*2, .. (e.g., **Spinor[p1] . DiracSlash[k] . Spinor[p2]**), as **StandardMatrixelement[***abb1***]**, **StandardMatrixelement[***abb2***]**, ... The last argument *enmomcon* defines energy momentum conservation; e.g., *enmomcon* = **{k2** → **p1 + p2 - k1}**.

■ **SetStandardMatrixElements** should be invoked only once for a whole process. It is

most conveniently used in a separate specification batch file. In this file also the settings of the scalar products should be done either directly and/or with **SetMandelstam** before applying **SetStandardMatrixElements**. ■ It is not necessary to predefine standard matrix elements. ■ See also: **SetMandelstam**, **StandardMatrixelement**.

■ **Small**

Small[*me***]** is the head of a small mass *me*. The effect is that masses with this head are set to zero, if they occur outside a Passarino-Veltman function.

■ **SmallVariables**

SmallVariables is an option for **OneLoop**.

■ The setting of **SmallVariables** is a list containing masses which are small compared to others. If present the photon mass should always be listed.

■ **Spinor**

Spinor[*p***, *mass*]** specifies a Dirac spinor. Which of the spinors $u, v, \overline{u}$ or $\overline{v}$ is understood, depends on the sign of the mass argument and the relative position of **DiracSlash[p]**: **Spinor[***p***, *mass*]** is that spinor which yields **mass*Spinor[***p***, *mass*]** if the Dirac equation is applied to **DiracSlash[***p***] . Spinor[***p***, *mass*]** or **Spinor[***p***, *mass*] . DiracSlash[***p***]**.

If a spinor is multiplied by a Dirac matrix or another spinor, the multiplication operator "." must be used.

■ See also: **DiracSimplify**.

■ **StandardMatrixElement**

StandardMatrixElement[...] is the head of the standard matrix elements. The standard matrix elements are a basis for the Feynman amplitude which contain the spinor structure and, eventually, the dependence on the polarization vectors.

If **SetStandardMatrixElements** has been used to define abbreviations for the non commutative and/or scalar-product structure, the arguments of **StandardMatrixElement** are these abbreviations itself. Without invoking **SetStandardMatrixElements** the arguments of **StandardMatrixElement** contain the basis for the amplitude, i.e., the non commutative structure and/or scalar-product structure.

■ **SU3Delta**

SU3Delta[*a, b***]** is the Kronecker δ with color indices *a* and *b*.

■ **SU3Delta[i, i]** → 8.
■ See also: **SU3F**, **GellMannMatrix**, **GellMannTrace**.

■ SU3F

SU3F[*a, b, c***]** are the structure constants f_{abc} of SU(3).

■ Only algebraic properties are implemented. ■ The following option can be given:

SU3FToTraces	**True**	replace f_{abc} by $\frac{i}{4}(tr(\lambda_a\lambda_c\lambda_b) - tr(\lambda_a\lambda_b\lambda_c))$.

■ See also: **SU3Delta**, **GellMannMatrix**, **GellMannTrace**.

■ SU3FToTraces

SU3FToTraces is an option for **SU3F**.

■ Tr

Tr[*expr***]** calculates the Dirac trace of *expr* directly. **Tr** is identical to **DiracTrace** up to the default setting of **DiracTraceEvaluate**.

■ The following options can be given:

DiracTraceEvaluate	**True**	evaluating the trace
LeviCivitaSign	**-1**	sign convention for the ε-tensor
Factoring	**False**	factor the result
Mandelstam	**{}**	if set to **{***s, t, u, m*$_1$**^2 + ***m*$_2$**^2 + ***m*$_3$**^2 + ***m*$_4$**^2}**, **TrickMandelstam** will be used
PairCollect	**True**	collect the results with respect to products of **Pair**

■ See also: **DiracTrace**.

■ TrickMandelstam

TrickMandelstam[*expr***, {***s, t, u, m*$_1$**^2 + ***m*$_2$**^2 + ***m*$_3$**^2 + ***m*$_4$**^2}]** simplifies all sums in *expr* in such a way that one of the Mandelstam variables s, t or u is eliminated by the relation $s + t + u = m_1^2 + m_2^2 + m_3^2 + m_4^2$. The trick is that the resulting sum has the shortest number of terms.

■ Example: **TrickMandelstam[s + t + u, {s,t,u,2 mw^2}]** → **2 mw^2**.

■ Write2

Write2[*channel*, val_1 = $expr_1$, val_2 = $expr_2$, ...**]** writes the settings $val_1 = expr_1$, $val_2 = expr_2$ in sequence followed by a newline, to the specified output channel.

■ The following options can be given:

FormatType	**InputForm**	in which language to write out to the result file
DOConvention	**0**	convention for scalar Passarino Veltman integrals

Other possible settings for **FormatForm** are **FortranForm**, **MacsymaForm** and **MapleForm**. ■ Be careful on the ouput when generating Fortran files. There might be problems like powers of ratios of integers which you have to correct by hand. ■ If the expressions contain variables in **HoldForm**, their values are written out first, if the output language is Mathematica or Fortran.

■ WriteOut

WriteOut is an option for **OneLoop**, **OneLoopSum**.

■ Possible settings are:

True	write output into a file *name*, where *name* is the first argument of **OneLoop**
False	write no output
"/usr/hep/weinberg/"	write result files in your local directory

■ WriteOutPaVe

WriteOutPave is an option for **OneLoopSum**. If set to a string the reduced **PaVe** are written into the file indicated.

■ $BreitMaison

$BreitMaison is a global variable determining whether the Breitenlohner-Maison scheme is used.

■ The default setting is **False**, which implies the "naive" γ_5 prescription: γ_5 is assumed to anticommute with Dirac matrices in all dimensions. ■ If **$BreitMaison** is set to **True**, γ_5 will anticommute with the four-dimensional part of a Dirac matrix and commute with the (D-4)-dimensional part. Reset **$PrePrint** for experimenting with the Breitenlohner-Maison scheme. **$BreitMaison** must be set to **True** before loading FeynCalc. The command is : **FeynCalc'$BreitMaison = True**. ■ Not every operation has been tested thoroughly, therefore beware!

■ **`$MemoryAvailable`**

`$MemoryAvailable` is a global variable which is set to an integer *n*, where *n* is the available amount of main memory in MB. The default is 8. It should be increased if possible. The higher **`$MemoryAvailable`** can be, the more intermediate steps do not have to be repeated by FeynCalc.

■ **`$VeryVerbose`**

`$VeryVerbose` is a global variable with default setting **`0`**. If set to **`1, 2, ...`**, more and more intermediate comments and informations are displayed during calculations.

Bibliography

[1] , Steven Wolfram (2000),
"The Mathematica Book", Fourth edition,
Cambridge University Press,
"http://www.mathematica.com/".

[2] J. Küblbeck, M. Böhm and A. Denner, Comp. Phys. Comm. **60** (1990) 165.
"FeynArts web site".

[3] R. D. Drinkard and N. K. Sulinski, "Macsyma: A Program For Computer Algebraic Manipulation (Demonstrations And Analysis)," NUSC-6401 "SPIRES entry".

[4] E. Yehudai and A. C. K. Hsieh, *HIP* — Symbolic High-Energy Physics Calculations,
SLAC-PUB- **5576** July 1991.

[5] O. V. Tarasov, "Generalized recurrence relations for two-loop propagator integrals with arbitrary masses," Nucl. Phys. B **502** (1997) 455 "arXiv:hep-ph/9703319".

[6] R. Mertig and R. Scharf, "TARCER: A mathematica program for the reduction of two-loop propagator integrals," Comput. Phys. Commun. **111** (1998) 265 "arXiv:hep-ph/9801383".

[7] J. Gasser and H. Leutwyler, "Chiral Perturbation Theory: Expansions In The Mass Of The Strange Quark," Nucl. Phys. B **250** (1985) 465.

[8] G. 't Hooft and M. J. Veltman, "Scalar One Loop Integrals," Nucl. Phys. B **153** (1979) 365.

[9] A. Denner, Techniques for the Calculation of Electroweak Radiative Corrections at the One-Loop Level and Results for W-Physics at LEP200, to appear in *Fortschritte der Physik 1992, 41* and references therein.

[10] R. Mertig, M. Böhm, and A. Denner, Comp. Phys. Comm. **64** (1991) 345.

[11] Comp. Phys. Comm **77** (1993) 286-298

[12] G. J. van Oldenborgh, FF – a package to evaluate one-loop Feynman diagrams, Comp. Phys. Comm. **66** (1991) 1.
Z. Phys. C **46** (1990) 425,
"Scanned version from KEK".

[13] T. Hahn and M. Perez-Victoria, "Automatized one-loop calculations in four and D dimensions," Comput. Phys. Commun. **118** (1999) 153 "arXiv:hep-ph/9807565".

[14] P. Breitenlohner and D. Maison, Commun. Math. Phys. **52** (1977) 11.

[15] C. P. Martin and D. Sanchez-Ruiz, renormalization with a non-anticommuting gamma(5)," Nucl. Phys. B **572** (2000) 387 [arXiv:hep-th/9905076].

[16] P.Cvitanovic, Phys. Rev. D **14** (1976) 1536.

[17] A.P. Kryukov and A.Ya. Rodionov, Comp. Phys. Comm. **48** (1988) 327.

[18] Matthias Jamin and E. Lautenbacher, TRACER, A Mathematica Package for γ-Algebra in Arbitrary Dimensions, preprint Technische Universität München, TUM-T31-20/91.

[19] S. Wolfram, MACSYMA Tools for Feynman Diagram Calculations, *Proceedings of the 1979 MACSYMA Users Conference*.