Guide to *FeynCalc* 1.0

Rolf Mertig
Physikalisches Institut
Am Hubland
Universität Würzburg
8700 Würzburg, Germany

March 1992

Contents

0 Introduction to *FeynCalc*

FeynCalc 1.0 is a *Mathematica* 2.0 package for algebraic calculations in high energy physics. The basic idea of *FeynCalc* is to provide convenient tools for radiative corrections in the standard model. 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* [1], which produces all diagrams for a given process. 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. With an appropiate option setting this polynomial is directly converted to an optimized Fortran file.

The computational methods and algorithms implemented in *FeynCalc* can be found in [2] and [3], where further references are given.

FeynCalc provides also 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 with *Mathematica* or ready to translate into Fortran. Apart from these basic operations like contraction of tensors, simplification of products of Dirac matrices and trace calculation no attempt has been made to provide tools for tree-level calculations in *FeynCalc* 1.0. For this purpose a lot of programs exist, among them another *Mathematica* package, HIP [4], developped at SLAC.

A free copy of *FeynCalc* may be obtained from the author. You may redistribute *FeynCalc* with no charge, but please do not change the program. *FeynCalc* 1.0 is the first public release of a set of programs which were developped and used at the University of Würzburg since 1988. Though a lot of 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 rolfm@wri.com or mertig@vax.rz.uni-wuerzburg.dbp.de . Suggestions and improvements are also welcome.

The explanations in this preliminary manual are rather short. The user is assumed to be skilled in high energy phyiscs and *Mathematica*. It is strongly recommended to study Sections 1.2, 1.4, 1.7, 1.8, 2.1 - 2.4 and 3.3 of the book *Mathematica*, Second Edition, by Stephen Wolfram.

In *FeynCalc* 1.0 no facilities for the numeric evaluation of Feynman diagrams are provided. Planned extensions of *FeynCalc* are tree level and two-loop facilities, the use of the program *MathLink* for the symbolic-numeric interface, and links to other programs for the evaluation of Feynman diagrams.

0.1 Acknowledgments

I would like to thank M. Böhm and A. Denner for their support and helpful discussions, which contributed considerably to the algorithms implemented in *FeynCalc*. I am grateful to J. Küblbeck and H. Eck for providing *FeynArts*, which was used for the graphics. Testing of the results of *FeynCalc* has been done by A. Denner, S. Dittmaier, J. Küblbeck, G. Weiglein and T. Sack.

I am very grateful to Stephen Wolfram for an invitation to Wolfram Research Inc., where most of this manual was written. Special thanks go to Jan Progen for the invaluable task of proof reading and to Joe Kaiping, who is a magician in typesetting. It is a pleasure to thank Matthew Markert, Tom Wickham-Jones, Stephen Wolfram and Dave Withoff for their comments and help on subtleties of *Mathematica*. Partial financial support by the Deutsche Forschungsgemeinschaft is greatly acknowledged.

0.2 Installation

The file `FeynCalc.m` should be put in a directory called `/usr/users/mydir/FeynCalc`. In this directory you have to open a subdirectory *Code* and put the subpackages `GellMann.m, General.m, Main.m, OneLoop.m` and `PaVe.m` into this directory (`/usr/users/mydir/FeynCalc/Code`).

From the directory `/usr/users/mydir/FeynCalc` you can start a *Mathematica* session and load *FeynCalc* by typing `<<FeynCalc.m`.

If you want to work from another directory, you should add the following command in the `init.m` (or `FeynCalc.m`) file:

```
$Path = Append[$Path, "/usr/users/mydir"]
```

If you add the command `<</usr/users/mydir/FeynCalc/FeynCalc.m` to your initialization file `init.m`, *FeynCalc* will be loaded automatically upon invoking *Mathematica*. Throughout this manual *FeynCalc* is always automatically loaded in this way.

1 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. You can only enter algebraic objects; there is no possibiltiy of working with explicit components of vectors or with specific Dirac matrices like γ_0 or γ_2. Note that no declaration of Lorentz indices or four-vectors has to be made. Covariant and contravariant indices are treated on an equal basis, assuming summation over repeated indices. Since explicit components of tensors will not be used at all, 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 17.

The input functions are macros that translate into the internal representation of *FeynCalc*. But these sometimes lengthy internal representations are difficult to recognize when working interactively. Therefore as a default in *FeynCalc* the global *Mathematica* variable `$PrePrint` is set to `FeynCalcForm`, forcing the display to be in a more readable manner. It is important to realize that the display given by `FeynCalcForm` is always different from the input syntax. Thus you may not substitute, for example, a scalar product in the result of a trace calculation by copying the form you see on the screen.

The internal structure of *FeynCalc*, which you may explore by clearing `$PrePrint`, is explained briefly in Section 4 and in the Reference Guide. Usually it is not necessary to know it. Thus there are three layers of *FeynCalc*: the input set of functions, a `FeynCalcForm` 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 or $D-4$ 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 dimension is four. If *FeynCalc* is used for interactive D-dimensional calculations you should change the option of the input functions. Note that Lorentz indices, 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 the function `DiracTrace`.

For the automatic calculation of one-loop amplitudes the input for the function `OneLoop` may always be given in four dimensions. `OneLoop` extends the amplitude to D dimensions automatically.

1.1 Entering Tensors and Scalar Products

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

Basic input functions.

Levi-Civita tensors and polarization vectors are defined in *FeynCalc* in four dimensions only. For `MetricTensor`, `FourVector` and `ScalarProduct` other dimensions may be specified with the option `Dimension`.

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

Option for `FourVector`, `MetricTensor` and `ScalarProduct`.

The following examples show the input syntax and the result, which is displayed by *FeynCalc* in an abbreviated form with the default setting of `$PrePrint` to `FeynCalcForm`. Thus `FeynCalcForm` is similar to the *Mathematica* `OutputForm`, but specialized to the new objects defined in *FeynCalc*.

Enter a scalar product $p \cdot q$.

```
In[1]:= ScalarProduct[p, q]
```

This is the `FeynCalcForm`.

```
Out[1]= p.q
```

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

```
In[2]:= MetricTensor[mu, nu]
```

```
Out[2]= g[mu, nu]
```

A metric tensor $g^{\alpha\beta}$ in `D` dimensions is entered in this way. The dimension is displayed as an index.

```
In[3]:= MetricTensor[al, be, Dimension -> D]
```

```
Out[3]= g     [al, be]
         D
```

Changing `D` to 4 recovers the default `FeynCalcForm` for a four-dimensional $g^{\alpha\beta}$.

```
In[4]:= % /. D -> 4
```

```
Out[4]= g[al, be]
```

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

```
In[5]:= MetricTensor[al, al]
```

```
Out[5]= 4
```

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

```
In[6]:= MetricTensor[al, al, Dimension -> D]
```

```
Out[6]= D
```

Enter a four-dimensional p^{μ}.

```
In[7]:= FourVector[p, mu]
```

```
Out[7]= p[mu]
```

You may also enter a sum of four-vectors: $(p - 2q)^{\mu}$.

```
In[8]:= FourVector[p - 2 q, mu]
```

```
Out[8]= (p - 2 q)[mu]
```

Providing $(2q - p)^{\mu}$ yields $-(p - 2q)^{\mu}$.

```
In[9]:= FourVector[2 q - p, mu]
```

```
Out[9]= -(p - 2 q)[mu]
```

In general numerical factors are pulled out in a standard way from `FourVector`.

```
In[10]:= FourVector[2 p, mu]
```

```
Out[10]= 2 p[mu]
```

This is a D-dimensional q^{α}.

```
In[11]:= FourVector[q, al, Dimension -> D]
```

```
Out[11]= q     [al]
          D
```

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

```
In[12]:= PolarizationVector[k, mu]
```

```
Out[12]= ep[k, mu]
```

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

```
In[13]:= Conjugate[PolarizationVector[k, mu]]
```

```
Out[13]= ep(*)[k, mu]
```

Polarization vectors are entered in a special way. The transversality condition $\varepsilon(k) \cdot k = 0$ is automatically fulfilled, see page 13.

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

The input for a polarization vector and polarization.

1.2 Entering 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 always 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[1]:= (DiracSlash[p + q] + m) . DiracMatrix[mu]
Out[1]= (gs[p + q] + m) ga[mu]
```

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

```
In[2]:= DiracMatrix[ro, 5, mu, 6, al, 7]
Out[2]= ga[ro] ga[5] ga[mu] ga[6] ga[al] ga[7]
```

This is an alternative input variety. Note that the spaces around the "." are essential when γ^5, γ^6 and γ^7 are involved.

```
In[3]:= DiracMatrix[mu . 5 . nu . 6 . ro . 7]
Out[3]= ga[mu] ga[5] ga[nu] ga[6] ga[ro] ga[7]
```

This is a four-dimensional product: $2\not{b}\not{a}\,2\,(\not{d}-\not{c})\,(6\not{q}-3\not{p})$. `DiracSlash` pulls common numerical factors out.

```
In[4]:= DiracSlash[2 b, a, 2 (d - c) , 6 q - 3 p]
Out[4]= 12 gs[b] gs[a] gs[c - d] gs[p - 2 q]
```

Here is an alternative input with `Dot`. Commutative products have to be grouped with parentheses here.

```
In[5]:= DiracSlash[(2 b) . a . (2 (d - c)) . (6 q - 3 p)]
Out[5]= 12 gs[b] gs[a] gs[c - d] gs[p - 2 q]
```

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

```
In[6]:= DiracSlash[Polarization[k]]
Out[6]= gs[ep[k]]
```

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

Option for `DiracMatrix` and `DiracSlash`.

As settings for `Dimension` 4, a *Mathematica* `Symbol` *dim*, or *dim* − 4 are possible.

Entering of a D-dimensional γ^μ.

The dimension is displayed as an index in the `FeynCalcForm`.

```
In[7]:= DiracMatrix[mu, Dimension -> D]
Out[7]= ga    [mu]
          D
```

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

```
In[8]:= DiracMatrix[mu, mu, Dimension -> D - 4]
Out[8]= ga    [mu] ga    [mu]
          D-4        D-4
```

■ 1.3 Entering Gell-Mann Matrices and SU(3) Structure Constants

FeynCalc is not a package especially designed for QCD calculations. But some basic features for Gell-Mann matrices are included.

`GellMannMatrix[a, b, ...]`	Gell-Mann matrices $\lambda_a\,\lambda_b\,\ldots$
`SU3Delta[a, b]`	Kronecker δ_{ab} with color indices a, b
`SU3F[a, b, c]`	structure constants f_{abc} of SU(3)

Input of Gell-Mann matrices, δ_{ab} and SU(3) structure constants.

For noncommutative multiplication use the *Mathematica* `Dot` ".".

Products of `GellMannMatrix` have to be separated by a ".".

```
In[9]:= GellMannMatrix[b] . GellMannMatrix[a]
Out[9]= la[b] la[a]
```

This is how you enter $\lambda_c\,\lambda_b\,\lambda_a\,\delta_{de}$.

```
In[10]:= GellMannMatrix[c, b, a] SU3Delta[d, e]
Out[10]= d[d, e] la[c] la[b] la[a]
```

The default for the f_{abc} is: $i/4\,[\,tr(\,\lambda_a\,\lambda_c\,\lambda_b\,) - tr(\,\lambda_a\,\lambda_b\,\lambda_c\,)\,]$.

The `trace` in the output is the `FeynCalcForm` for `GellMannTrace`.

```
In[11]:= SU3F[a, b, c]
             I
Out[11]= -   - (-tr[la[a] la[b] la[c]] + tr[la[a] la[c] la[b]])
             4
```

option name	*default value*	
`SU3FToTraces`	`True`	whether to replace `SU3F` by traces

An option for the structure constants `SU3F`.

The structure constants are totally antisymmetric: $f_{acb} = -f_{abc}$.

```
In[12]:= SU3F[a, c, b, SU3FToTraces -> False]
Out[12]= -f[a, b, c]
```

$f_{aab} = 0$.

```
In[13]:= SU3F[a, a, b]
Out[13]= 0
```

■ 1.4 Entering Spinors

In *FeynCalc* 1.0 two different types of spinors can be entered. For leptonic spinors you can use `LeptonSpinor` or `Spinor`. The convention for a quark spinor is to suppress the color index. The only difference between `QuarkSpinor` and `LeptonSpinor` is that Gell-Mann matrices do not commute with quark spinors.

`Spinor[p, m]`	$u(p,m)$ or $\overline{u}(p,m)$ for leptons
`Spinor[-p,m]`	$v(p,m)$ or $\overline{v}(p,m)$ for leptons
`LeptonSpinor[p, m]`	$u(p,m)$ or $\overline{u}(p,m)$ for leptons
`LeptonSpinor[-p,m]`	$v(p,m)$ or $\overline{v}(p,m)$ for leptons
`QuarkSpinor[p, m]`	$u(p,m)$ or $\overline{u}(p,m)$ for quarks
`QuarkSpinor[-p,m]`	$v(p,m)$ or $\overline{v}(p,m)$ for quarks

The functions for leptonic and hadronic spinors.

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.

This is a leptonic spinor $u(p,m)$ or $\overline{u}(p,m)$.

```
In[14]:= Spinor[p, m]
Out[14]= u[p, m]
```

The minus sign of the first argument indicates that $v(p,m)$ or $\overline{v}(p,m)$ is understood.

```
In[15]:= Spinor[-p, m]
Out[15]= v[p, m]
```

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

```
In[16]:= Spinor[p, m] . DiracSlash[p]
Out[16]= u[p, m] gs[p]
```

Since $\not{p}$ is now left of the spinor, $\not{p}\,u(p,m)$ is understood.

```
In[17]:= DiracSlash[p] . QuarkSpinor[p, m]
Out[17]= gs[p] u[p, m]
```

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

```
In[18]:= Spinor[-p, m] . DiracSlash[q, p]
Out[18]= v[p, m] gs[q] gs[p]
```

Here we have $\not{p}\,v(p,m)$.

```
In[19]:= DiracSlash[p] . QuarkSpinor[-p, m]
Out[19]= gs[p] v[p, m]
```

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

```
In[20]:= Spinor[p, 0]
Out[20]= u[p]
```

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

```
In[21]:= Spinor[p]
Out[21]= u[p]
```

1.5 Entering 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, m0],
                   PropagatorDenominator[q + p, m1], ...]
```

$$1/([q^2 - m_0^2][(q+p)^2 - m_1^2])\;\ldots$$

Entering denominators of propagators.

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

The `FeynCalcForm` imitates the usual notation.

```
In[22]:= FeynAmpDenominator[ PropagatorDenominator[q, m1],
                             PropagatorDenominator[q + p, m2]]
```

$$\text{Out[22]=}\ \frac{1}{(q^2 - m1^2)\,((p+q)^2 - m2^2)}$$

■ 1.6 Entering 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 evtl. 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 basic idea is that every variable with head `Small` evaluates to 0, unless it is an argument of a scalar integral.

You can avoid to write explicitly the head `Small` around a mass by using the option `SmallVariables` of the function `OneLoop`.

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

A head for small variables.

In spinors a mass with head `Small` is replaced by 0.

```
In[23]:= Spinor[p, Small[m]]
Out[23]= u[p]
```

2 Elementary Calculations

You can use *FeynCalc* for basic calculations like Trace evaluations. In the following examples *FeynCalc* was loaded automatically by including `<<FeynCalc.m` in the `init.m` file. The output is given in `FeynCalcForm`, *i.e.*, the default of `$PrePrint` is `FeynCalcForm`.

2.1 Contraction of Metric Tensors, Four-Vectors and Levi-Civita Tensors

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*`]`	contracts double Lorentz indices

The function for contraction of tensors.

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

```
In[1]:= Contract[MetricTensor[a, b] FourVector[p, b]]
Out[1]= p[a]
```

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

```
In[2]:= Contract[FourVector[q, al] FourVector[p - q, al]]
Out[2]= (p - q).q
```

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

```
In[3]:= Contract[MetricTensor[al, be] DiracMatrix[al]]
Out[3]= ga[be]
```

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

```
In[4]:= Contract[FourVector[q, al] DiracMatrix[al]]
Out[4]= gs[q]
```

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

```
In[5]:= Contract[LeviCivita[mu, nu, ro, si] FourVector[p, si]]
Out[5]= eps[mu, nu, ro, p]
```

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

```
In[6]:= Contract[LeviCivita[a, n, r, s] LeviCivita[b, n, r, s],
        EpsContract -> True ]
Out[6]= -6 g[a, b]
```

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

Options for `Contract`.

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

```
In[7]:= Contract[ MetricTensor[al, si] *
            FourVector[p, al] FourVector[p, si] *
            ( FourVector[q, be] + FourVector[r, be] ) *
            ( FourVector[p, be] - FourVector[q, be] ),
            Expanding -> False ]
Out[7]= p.p (p[be] - q[be]) (q[be] + r[be])
```

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

```
In[8]:= Contract[FourVector[k, mu] PolarizationVector[k, mu]]
Out[8]= 0
```

2.2 Simplification of Scalar Products and Four-Vectors

`ExpandScalarProduct[`*expr*`]`	expand scalar products and four-vectors in *expr*

A function for expansion of scalar products and four-vectors.

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

```
In[1]:= ExpandScalarProduct[ScalarProduct[a + b, c - 2 d]]
Out[1]= a.c - 2 a.d + b.c - 2 b.d
```

Consider again $q^\alpha\ (p-q)^\alpha$.

```
In[2]:= Contract[FourVector[q, al] FourVector[p - q, al]]
Out[2]= (p - q).q
```

This expands the scalar products.

```
In[3]:= ExpandScalarProduct[%]
Out[3]= p.q - q.q
```

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

```
In[4]:= % /. ScalarProduct[q, q] -> 0
Out[4]= 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[5]:= ScalarProduct[q, q] = 0
Out[5]= 0
```

Expanding the result of the contraction $q^\alpha\ (p-q)^\alpha$ now yields $(p\cdot q)$.

```
In[6]:= ExpandScalarProduct[Out[2]]
Out[6]= p.q
```

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

```
In[7]:= ExpandScalarProduct[FourVector[a + b, mu]]
Out[7]= a[mu] + b[mu]
```

2.3 Simplification of Products of Dirac Matrices and Spinors

For the simplification of noncommutative products of Dirac matrices and spinors, two functions are provided: `DiracOrder`, which orders products of Dirac matrices in a canonical way, and `DiracSimplify`, which contracts Dirac matrices with equal indices, moves γ^5, γ^6 and γ^7 to the right end applies the Dirac equation and expands noncommutative products, see section 4.4.11. The Dirac matrices in the result of `DiracSimplify` are only ordered in a canonical way if they are between spinors.

`DiracOrder[`*expr*`]`	alphabetical ordering of Dirac matrices
`DiracOrder[`*expr*`, {`*a*`, `*b*`, ...}]`	ordering according to $a, b, \ldots$
`DiracSimplify[`*expr*`]`	contract all Lorentz indices and simplify

Simplification functions for Dirac matrices and spinors.

Both functions take as *expr* either subsequent `DiracMatrix` or `DiracSlash` separated by "," or any expression with "." as the noncommutative multiplication operator between Dirac matrices or Dirac slashes.

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

```
In[1]:= DiracOrder[DiracMatrix[be, al]]
Out[1]= 2 g[al, be] - ga[al] ga[be]
```

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

```
In[2]:= DiracOrder[%, {be, al}]
Out[2]= ga[be] ga[al]
```

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

```
In[3]:= DiracOrder[DiracMatrix[mu, mu], DiracSlash[p, p]]
Out[3]= 4 p.p
```

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

```
In[4]:= DiracOrder[DiracMatrix[a, m, a, Dimension -> D]]
Out[4]= (2 - D) ga [m]
                   D
```

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

```
In[5]:= DiracOrder[DiracSlash[-p, q, p]]
Out[5]= gs[q] p.p - 2 gs[p] p.q
```

Basically `DiracOrder` is just the implementation of the anticommutator relation $\{\gamma^\mu, \gamma^\nu\} = 2\,g^{\mu\nu}$.

For calculations involving γ^5 you can use the function `DiracSimplify`. All γ^5, γ^6 and γ^7 are moved to the right end of the product of Dirac matrices; see section 4.4.11 for the treatment of γ^5 in D dimensions. `DiracSimplify` applies the Dirac equation. The result of `DiracSimplify` is not canonically ordered.

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

```
In[1]:= DiracSimplify[DiracMatrix[mu, mu, Dimension -> D]]
Out[1]= 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[2]:= DiracSimplify[DiracMatrix[mu, nu, ro, si, mu]]
Out[2]= -2 ga[si] ga[ro] ga[nu]
```

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[3]:= DiracSimplify[1/2 DiracMatrix[mu, a, b ,c ,d ,mu]]
Out[3]= ga[c] ga[b] ga[a] ga[d] + ga[d] ga[a] ga[b] ga[c]
```

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

```
In[4]:= DiracSimplify[DiracSlash[p], DiracSlash[-q] + m,
                      DiracSlash[p]]
Out[4]= gs[q] p.p + m p.p - 2 gs[p] p.q
```

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

```
In[5]:= DiracSimplify[DiracMatrix[5], DiracMatrix[mu]]
Out[5]= -ga[mu] ga[5]
```

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

```
In[6]:= DiracSimplify[DiracMatrix[6, nu, 7, mu]]
Out[6]= ga[nu] ga[mu] ga[6]
```

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

```
In[7]:= DiracSimplify[(DiracSlash[p] - m) . Spinor[p, m]]
Out[7]= 0
```

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

```
In[8]:= DiracSimplify[(DiracSlash[p] + m) . Spinor[-p, m]]
Out[8]= 0
```

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

```
In[9]:= DiracSimplify[Spinor[p, m] . (DiracSlash[p] - m)]
Out[9]= 0
```

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

```
In[10]:= DiracSimplify[Spinor[-p, m] . DiracSlash[q] .
                                       (DiracSlash[p] - m)]
Out[10]= 2 v[p, m] p.q
```

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

```
In[11]:= DiracSimplify[Spinor[-p, m] . DiracSlash[q, p] .
                       Spinor[q, M]] // Factor
Out[11]= v[p, m] u[q, M] (m M + 2 p.q)
```

`DiracSimplify` orders products of Gell-Mann matrices from the Dirac structure.

```
In[12]:= DiracSimplify[QuarkSpinor[-p1] . GellMannMatrix[b] .
                          DiracSlash[q] . GellMannMatrix[a] .
                       QuarkSpinor[p2]]
Out[12]= u[p1] la[b] la[a] gs[q] u[p2]
```

2.4 Dirac Traces

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

Two functions for 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` should be used.

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

```
In[1]:= Tr[DiracMatrix[al, be]]
Out[1]= 4 g[al, be]
```

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

```
In[2]:= Tr[DiracSlash[a, b, c, d]]
Out[2]= 4 a.d b.c - 4 a.c b.d + 4 a.b c.d
```

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

```
In[3]:= Tr[DiracMatrix[a, b, c, d, 5]]
Out[3]= -4 I eps[a, b, c, d]
```

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

```
In[4]:= Tr[MetricTensor[al, be]/4 DiracMatrix[mu],
           DiracMatrix[al] FourVector[p, mu]]
Out[4]= p[be]
```

If you want to do more complicated traces it is convenient to introduce abbreviations for the several objects. The following examples show how to do this. The CPU time needed for each trace calculation is for all examples less than one minute (on a NeXT). Some of the examples here verify the results given in [9].

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

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

Set the abbreviations for Dirac matrices and slashes here.

```
In[5]:= ( PP = DiracSlash[p'];
          P = DiracSlash[p];    K = DiracSlash[k];
          A = DiracMatrix[al];  B = DiracMatrix[be]
        )
Out[5]= ga[be]
```

This is the input for trace T_1.

```
In[6]:= Tr[PP + m,A,P + K + m,B,P + m,B,P + K + m,A/16]
```

The CPU time needed for the calculation is of the order of seconds.

```
                4       2            2            2
Out[6]= 4 m  + 4 m  k.k + 4 m  k.p - 4 m  k.p' +
                                    2
  2 k.p k.p' + 2 k.p' p.p - 3 m  p.p' - k.k p.p' + p.p p.p'
```

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

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

This defines abbreviations for trace T_2. a, b, n denote $\gamma^\alpha, \gamma^\beta, \gamma^\nu$ in D dimensions.

The last command sets $P1 = \not{p}_1$, $P2 = \not{p}_2$, $P3 = \not{p}_3$.

```
In[7]:= ( a = DiracMatrix[al, Dimension -> D];
          b = DiracMatrix[be, Dimension -> D];
          n = DiracMatrix[nu, Dimension -> D];
          {P1, P2, P3} = Map[ DiracSlash, {p1, p2, p3} ])
Out[7]= {gs[p1], gs[p2], gs[p3]}
```

Here is the input for trace T_2.

```
In[8]:= Tr[b,a,P1,P2,n,b,P2,P3,a,P1,n,P3,P1,P2]
```

The result is again collected with respect to scalar products.

```
Out[8]= (-1152 + 896 D - 224 D^2 + 16 D^3) p1.p2 p1.p3^2
 
   p2.p2 + (1024 - 512 D + 64 D^2) p1.p2^2 p1.p3 p2.p3 +
 
  (448 - 416 D + 112 D^2 - 8 D^3) p1.p1 p1.p3 p2.p2 p2.p3 +
 
  (-512 + 256 D - 32 D^2) p1.p1 p1.p2 p2.p3^2 +
 
  (-512 + 256 D - 32 D^2) p1.p2^3 p3.p3 +
 
  (672 - 416 D + 80 D^2 - 4 D^3) p1.p1 p1.p2 p2.p2 p3.p3
```

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[9]:= Tr[b.a.P1.P2.n.b.P2.P3.a.P1.n.P3.P1.P2 /. D ->4]
```

```
Out[9]= -128 p1.p2 p1.p3^2 p2.p2 +
 
  64 p1.p1 p1.p3 p2.p2 p2.p3 + 32 p1.p1 p1.p2 p2.p2 p3.p3
```

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

```
In[10]:= DiracTrace[DiracMatrix[al, be, ro, si]]
Out[10]= tr[ga[al] ga[be] ga[ro] ga[si]]
```

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

```
In[11]:= Contract[% MetricTensor[al, be]]
Out[11]= tr[ga[be] ga[be] ga[ro] ga[si]]
```

This evaluates the Dirac trace.

```
In[12]:= EvaluateDiracTrace[%]
Out[12]= 16 g[ro, si]
```

`EvaluateDiracTrace[`*expr*`]`	evaluate `DiracTrace` in *expr*

Evaluation of Dirac traces.

option name	*default value*	
`DiracTraceEvaluate`	`False`	whether to 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 $(-1) 4\, i\, \varepsilon^{abcd}$
`Factoring`	`False`	whether to factor canonically
`Mandelstam`	`{}`	utilize the Mandelstam relation
`PairCollect`	`True`	whether to collect `Pair`

Options for `DiracTrace`.

`Tr` takes the same options as `DiracTrace`, but the default setting of *DiracTraceEvaluate* is `True`.

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 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 often the Mandelstam relation $s + t + u = m_1^2 + m_2^2 + m_3^2 + m_4^2$ can 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* figures 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.

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

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[13]:= (SetMandelstam[s,t,u, k1,k2,-p1,-p2, 0,0,mt,mt];
          {K1, P1, P2} = Map[ DiracSlash, {k1, p1, p2} ];
             {SI, RO} = Map[ DiracMatrix,{si, ro} ];
          polsum1 = PolarizationSum[al, ro, k1, p1 - p2];
          polsum2 = PolarizationSum[be, si, k2, p1 - p2];
             p1al = FourVector[p1, al];
             p2be = FourVector[p2, be])
Out[13]= p2[be]
```

This is a possible input for trace T_3. *FeynCalc* contracts first all Lorentz indices and then calculates the trace.

```
In[14]:= Tr[ polsum1 polsum2 p1al p2be SI,
             K1 - P1 - mt, RO, P1 + mt, P2 - mt,
             Mandelstam -> {s, t, u, 2 mt^2} ]
```

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.

$$\text{Out[14]=}\ \frac{-2\ mt\ s\ (mt^4 - t\ u)\ (8\ mt^4 - t^2 - 6\ t\ u - u^2)}{(t - u)^3}$$

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

```
In[15]:= temp = Tr[SI, K1 - P1 - mt, RO, P1 + mt, P2 - mt]
Out[15]= (-2 mt t + 2 mt u) g[ro, si] - 4 mt k1[si] p1[ro] -
  4 mt k1[ro] p1[si] + 8 mt p1[ro] p1[si] +
  4 mt k1[si] p2[ro] + 4 mt k1[ro] p2[si] -
  8 mt p1[ro] p2[si]
```

Then contract the result with the polarization sums, expand the scalar products and use `TrickMandelstam` in order to get the tricky Mandelstam variable substitution.

This method is faster; but that is not the case for all trace calculations.

```
In[16]:= TrickMandelstam[ ExpandScalarProduct[ Contract[
                          temp polsum1 polsum2 p1al p2be]],
                          {s, t, u, 2 mt^2} ]
```

$$\text{Out[16]=}\ \frac{-2\ mt\ s\ (mt^4 - t\ u)\ (8\ mt^4 - t^2 - 6\ t\ u - u^2)}{(t - u)^3}$$

The function `TrickMandelstam` is explained in section 4.4.7.

Since Dirac matrices can be given in any dimensions, *FeynCalc* is also able to calculate traces in $D-4$ dimensions.

Defining $T(n) = Tr(\gamma_{\mu_1}\gamma_{\mu_2}\ldots\gamma_{\mu_n}\gamma_{\mu_1}\gamma_{\mu_2}\ldots\gamma_{\mu_n})$ 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 [8].

This is a little program defining T. The dimension of each particular Dirac matrix is set to $d-4$.

The timings are also displayed in `FeynCalcForm`, which yields more usual time units.

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

This calculates a trace of 16 matrices.

```
In[2]:= T[8]
Time = 38. s

                                                  2
Out[2]= 123469824 - 135962624 d + 63224832 d  -
                                                 7
  16145920 d^3 + 2461760 d^4 - 227584 d^5 + 12320 d^6 - 352 d^7 +
                                                 8
  4 d
```

Here we have 18.

```
In[3]:= T[9]
Time = 1.8 min

                                                   2
Out[3]= -1879576576 + 2220901376 d - 1127626752 d  +
                                                 6
  321806848 d^3 - 56625408 d^4 + 6331584 d^5 - 446208 d^6 +
                                                 9
  18912 d^7 - 432 d^8 + 4 d
```

The trace of 20 Dirac matrices.

```
In[4]:= T[10]
Time = 7.2 min

                                                    2
Out[4]= -31023169536 + 38971179008 d - 21328977920 d  +
                                                 5
  6679521280 d^3 - 1320732160 d^4 + 171464832 d  -
                                                 10
  14710080 d^6 + 816960 d^7 - 27840 d^8 + 520 d^9 - 4 d
```

With 22 Dirac matrices it gets slow.

```
In[5]:= T[11]
Time = 55. min

                                                        2
Out[5]= 551768735744 - 731506905088 d + 427299186688 d  -
                                                 5
  144858475520 d^3 + 31576821760 d^4 - 4629805312 d  +
                                                 9
  463655808 d^6 - 31521600 d^7 + 1415040 d^8 - 39600 d^9 +
                                                 11
  616 d^10 - 4 d
```

2.5 Gell-Mann Traces and Contraction of Color Indices

`GellMannTrace[`*expr*`]`	calculates traces of Gell-Mann matrices

The function for trace calculation of λ_a matrices.

The matrices $\frac{1}{2}\lambda_a$ are the infinitesimal generators of the group SU(3). Similar to the case of Dirac traces, also traces of the Gell-Mann matrices λ_a are calculated algebraically. The Cvitanovic algorithm [6] is implemened similar to [7]. Traces with no free color indices are always evaluated to a number. Otherwise only special cases are simplified.

$tr(\lambda_a \lambda_b) = 2\,\delta_{ab}$.

```
In[1]:= GellMannTrace[GellMannMatrix[a, b]]
Out[1]= 2 d[a, b]
```

$tr(3\,\lambda_a \lambda_b \lambda_a \lambda_b) = -32$.

```
In[2]:= GellMannTrace[ 3 GellMannMatrix[a, b, a, b] ]
Out[2]= -32
```

$tr(\lambda_a \lambda_b \lambda_c)$ stays as it is.

```
In[3]:= GellMannTrace[GellMannMatrix[a, b, c]]
Out[3]= tr[la[a] la[b] la[c]]
```

$tr(\lambda_a \lambda_b \lambda_c f_{abc}) = 48\,i$.

```
In[4]:= GellMannTrace[SU3F[a, b, c] GellMannMatrix[a, b, c]]
Out[4]= 48 I
```

$tr(f_{ars} f_{brs}) = 3 f_{ars} f_{brs} = 9\,\delta_{ab}$.

```
In[5]:= GellMannTrace[SU3F[a, r, s] SU3F[b, r, s]]
Out[5]= 9 d[a, b]
```

A non trivial case:
$tr(\lambda_a \lambda_c \lambda_e \lambda_d f_{abe} f_{bcd}) = 0$.

This takes a few seconds.

```
In[6]:= GellMannTrace[ GellMannMatrix[a, c, e, d] *
                       SU3F[a, b, e] SU3F[b, c, d] ]
Out[6]= 0
```

Repeated indices of δ_{ab} and f_{abc} can always be contracted with the Cvitanovic algorithm.

$16 f_{ars} f_{brs} = 48\,\delta_{ab}$.

The structure constants are replaced by traces.

```
In[7]:= 16 SU3F[a, r, s] SU3F[b, r, s]
Out[7]= -((-tr[la[a] la[r] la[s]] + tr[la[a] la[s] la[r]])
    (-tr[la[b] la[r] la[s]] + tr[la[b] la[s] la[r]]))
```

Expanding forces the products of traces to simplify, if there are indices to contract.

```
In[8]:= ExpandAll[%]
Out[8]= 48 d[a, b]
```

Define `f` as `SU3F`.

```
In[9]:= f[x__] := SU3F[x]
```

$f_{abr} f_{rcs} + f_{bcr} f_{ras} + f_{car} f_{rbs} = 0$.

```
In[10]:= ExpandAll[ f[a,b,r] f[r,c,s] + f[b,c,r] f[r,a,s] +
                    f[c,a,r] f[r,b,s] ]
Out[10]= 0
```

$\lambda_a \lambda_b \lambda_a = -\frac{2}{3}\lambda_b$.

```
In[11]:= GellMannMatrix[a, b, a]
```

$$\text{Out[11]=}\ \frac{-2\ \text{la[b]}}{3}$$

This is $f_{abc}\,\lambda_b \lambda_c = 3\,i\,\lambda_a$.

```
In[12]:= SU3F[c, a, b, SU3FToTraces -> False] *
         GellMannMatrix[b, c]
Out[12]= 3 I la[a]
```

3 One-Loop Calculations

The methods and conventions implemented in *FeynCalc* for the evaluation of one-loop diagrams are described in [2] and [3]. The usual Passarino-Veltman scheme for the one-loop integrals is adapted to a large extent [2]. The coefficient functions of the tensor integrals are defined similar to [2], except that the Passarino-Veltman integrals take internal masses squared as arguments. A wrapper Fortran program to link FF — a Fortran program by G.J. van Oldenborgh [5] for the evaluation of the scalar n-point integrals — and *FeynCalc* is available from the author. Currently *FeynCalc* is limited to four-point integrals.

3.1 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]`	$(i\pi^2)^{-1} \int d^D q (q^2 - m_0^2)^{-1}$
`B0[p10,m02,m12]`	$(i\pi^2)^{-1} \int d^D q ([(q^2 - m_0^2][(q+p_1)^2 - m_1^2])^{-1}$
`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]`	$(i\pi^2)^{-1} \int d^D q ([q^2 - m_0^2][(q+p_1)^2 - m_1^2][(q+p_2)^2 - m_2^2])^{-1}$
`D0[p10,p12,p23,p30,p20,p13,m02,m12,m22,m32]`	$(i\pi^2)^{-1} \int d^D q ([q^2 - m_0^2][(q+p_1)^2 - m_1^2][(q+p_2)^2 - m_2^2][(q+p_3)^2 - m_3^2])^{-1}$

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[1]:= B0[s, mz2, mw2]
Out[1]= B0[s, mw2, mz2]
```

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

```
In[2]:= D[%, s]
Out[2]= DB0[s, mw2, 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

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

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.

$$
\begin{aligned}
B_\mu &= p_{1\mu} B_1 \\
B_{\mu\nu} &= g_{\mu\nu} B_{00} + p_{1\mu} p_{1\nu} B_{11}
\end{aligned}
$$

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

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[3]:= (SetOptions[A0, A0ToB0 -> False];
         SetOptions[{B1, B00, B11}, BReduce -> False]);
```

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

```
In[4]:= PaVe[0, {}, {m02}]
Out[4]= A0[m02]
```

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

```
In[5]:= PaVe[0, {pp}, {m02, m12}]
Out[5]= B0[pp, m02, m12]
```

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

```
In[6]:= PaVe[1, {pp}, {m12, m22}]
Out[6]= B1[pp, m12, m22]
```

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

```
In[7]:= PaVe[0,0, {pp}, {m02, m12}]
Out[7]= B00[pp, m02, m12]
```

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

```
In[8]:= PaVe[1,1, {p10}, {m12, m22}]
Out[8]= B11[p10, m12, m22]
```

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

```
In[9]:= PaVe[0, {p10, p12, p20}, {m12, m22, m32}]
Out[9]= C0[p10, p12, p20, m12, m22, m32]
```

This is D_0 with 10 arguments.

```
In[10]:= PaVe[0, {p10, p12, p23, p30, p20, p13},
                 {m12, m22, m32, m42}]
Out[10]= 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	True	whether to express $A_0(m^2)$ by $m^2(1 + B_0(0, m^2, m^2))$
BReduce	True	reduce B0, B1, B00, B11 to A0 and B0
B0Unique	False	whether to 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[1]:= B1[pp, m12, m22]
```

$$\text{Out[1]=}\ \frac{\text{-B0[pp, m12, m22]}}{2} + \frac{\text{(-m12 + m22) (-B0[0, m12, m22] + B0[pp, m12, m22])}}{\text{2 pp}}$$

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[Small[m]^2]` simplify to 0.

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

```
In[2]:= B1[pp, Small[me2], m22]
```

$$\text{Out[2]=}\ \frac{\text{-B0[pp, 0, m22]}}{2} + \frac{\text{m22 (-B0[0, 0, m22] + B0[pp, 0, m22])}}{\text{2 pp}}$$

But in this case no arguments are replaced by 0.

```
In[3]:= B1[Small[me2], Small[me2], 0]
```

$$\text{Out[3]=}\ -\left(\frac{1}{2}\right) - \frac{\text{B0[Small[me2], 0, Small[me2]]}}{2}$$

`PaVeReduce[`*expr*`]`	reduces coefficient functions `PaVe` to `A0`, `B0`, `C0`, `D0`
`K[`*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*	
`IsolateHead`	`False`	whether to 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 4.4.2.

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

```
In[4]:= PaVeReduce[ PaVe[2, {Small[me2], mw2, t},
                         {Small[me2], 0, mw2} ]
                 ]
```

$$\text{Out[4]=}\ \frac{\text{B0[mw2, 0, mw2]}}{\text{mw2 - t}} - \frac{\text{B0[t, 0, mw2]}}{\text{mw2 - t}}$$

Break down the coefficient function $C_{12}(s, m^2, m^2, m^2, m^2, M^2)$.

```
In[5]:= c12 = PaVeReduce[ PaVe[1, 2, {s, m2, m2}, {m2, m2, M2}],
                          IsolateHead -> K ]
```

This is the result in `HoldForm`.

```
Out[5]= K[11]
```

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

```
In[6]:= K[11]
```

$$\text{Out[6]=}\ \frac{\text{K[6]}}{2} + \frac{\text{M2 K[4] K[7]}}{\text{K[5]}^2} + \frac{\text{K[1] K[8]}}{\text{2 m2 K[5]}} + \frac{\text{K[3] K[9]}}{\text{2 K[5]}^2} - \frac{\text{K[2] K[10]}}{\text{2 m2 K[5]}^2}$$

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

```
In[7]:= FixedPoint[ReleaseHold[#]&, c12]
```

$$\text{Out[7]=}\ \frac{1}{2\ (4\ m2 - s)} + \frac{(m2 - M2)\ BO[0, m2, M2]}{2\ m2\ (4\ m2 - s)} - \frac{(8\ m2^2 - 10\ m2\ M2 - 2\ m2\ s + M2\ s)\ BO[m2, m2, M2]}{2\ m2\ (4\ m2 - s)^2} + \frac{(4\ m2 - 6\ M2 - s)\ BO[s, m2, m2]}{2\ (4\ m2 - s)^2} + \frac{M2\ (8\ m2 - 3\ M2 - 2\ s)\ CO[m2, m2, s, m2, M2, m2]}{(4\ m2 - s)^2}$$

Take a coefficient 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[8]:= d122 = PaVeReduce[
          PaVe[1, 2, 2, {Small[ME2], MW2, MW2, Small[ME2], S, T},
                        {0, Small[ME2], 0, Small[ME2]}],
          Mandelstam -> {S, T, U, 2 MW2}, IsolateHead -> F ]
Out[8]= F[17]
```

Write the result out into a Fortran file.

```
In[9]:= 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. If you want to have a more optimized Fortran output you may use the settings `MacsymaForm` or `MapleForm` of the option `FormatType` for creating an output file with the syntax of the computer algebra systems *Macsyma* and *Maple*, respectively. Then you can use the Fortran code generators of *Maple* or *Macsyma*. In some versions of *Macsyma* the excellent package *Gentran* by B. Gates and H. van Hulzen is available.

The function `Write2` is explained in Section 4.

Note that the head `Small` is eliminated in the Fortran output automatically.

```
In[10]:= !!d122.for

      F(1) = BO(MW2,ODO,ODO)
      F(2) = BO(T,ODO,ODO)
      F(3) = BO(S,ODO,ODO)
      F(4) = CO(MW2,MW2,T,ME2,ODO,ME2)
      F(5) = CO(MW2,S,ME2,ME2,ODO,ODO)
      F(6) = CO(T,ME2,ME2,ME2,ME2,ODO)
      F(7) = DO(ME2,MW2,MW2,ME2,S,T,ODO,ME2,ODO,ME2)
      F(8) = MW2 + S
      F(9) = 4*MW2 - T
      F(10) = MW2**2 - S*U
      F(11) = F(8)/(F(9)*F(10))
      F(12) = -MW2 + S
      F(13) = 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(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) = MW2**2 - 4*MW2*S + 2*S**2 + S*U
      F(16) = 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(17) = -F(11)/2 + S**2*T**2*F(6)*F(12)/(2*F(10)**3) -
     -  S**3*T**2*F(7)*F(12)/(2*F(10)**3) +
     -  S**2*T*F(5)*F(12)**2/F(10)**3 +
     -  F(1)*F(13)/(2*F(9)**2*F(10)**2*F(12)) +
     -  F(2)*F(8)*F(14)/(2*F(9)**2*F(10)**2) -
     -  S*F(3)*F(15)/(2*F(10)**2*F(12)) -
     -  F(4)*F(8)*F(16)/(2*F(9)**2*F(10)**3)
      d122res = F(17)
```

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.

3.2 A One-Loop Self Energy Diagram

The function `OneLoop` performs the algebraic simplifications of a given amplitude. The result is given in a polymial 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.

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 $A_0 = -i\,\pi^{-2}\,(2\,\pi\mu)^{4-D}\int d^Dq\,(q^2-m^2)^{-1}$, corresponding to a tadpole diagram.

The scaling variable μ is suppressed in *FeynCalc*.

```
In[1]:= -I/Pi^2 FeynAmpDenominator[
            PropagatorDenominator[q, m]]

                 -I
Out[1]= ------------------
                2  2    2
        Pi  (q  - m  )
```

This calculates the tadpole diagram.

```
In[2]:= OneLoop[q, %]

                2    2        2    2
Out[2]= m  + m  BO[0, m , m ]
```

For a most compact result the factoring option of `OneLoop` is set. For a description of all options of `OneLoop` see Section 3.3.4.

```
In[3]:= SetOptions[OneLoop, Factoring -> True,
                                Dimension -> D];
```

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[4]:= 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

                  2
Out[4]= -(el  (k2 + 6 mf2 BO[0, mf2, mf2] -

        3 (k2 + 2 mf2) BO[k2, mf2, mf2])) / (36 Pi )
```

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.

3.3 Generic Diagrams for $W \to f_i\bar{f}_j'$ with OneLoop

As an example for calculating triangle diagrams the result for two generic one-loop diagrams of the decay $W \to f_i\bar{f}_j'$ for massless fermions given in [2] 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.

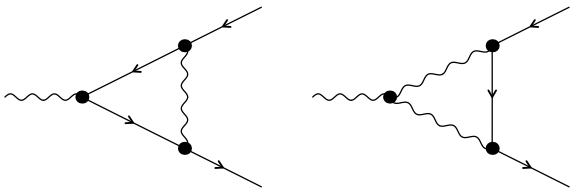

Figure 1: Two generic diagrams for the decay of $W \to f_i \bar{f}_j$, generated by *FeynArts*.

This factors the result.

```
In[1]:= SetOptions[OneLoop, Factoring -> True];
```

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[2]:= (gc[i_] := g[i, "-"] DiracMatrix[7] +
                   g[i, "+"] DiracMatrix[6];
         ScalarProduct[p1, p1] = 0;
         ScalarProduct[p2, p2] = 0;
         ScalarProduct[p1, p2] = k2/2);
```

The analytical expression for the generic diagram is:

$\delta \mathcal{M}_1 = i(2\pi)^{-4} \int (2\pi\mu)^{4-D} d^D q$

$[(q^2 - M^2)\,(q+p_1)^2\,(q-p_2)^2]^{-1}$

$\overline{u}(p_1)\,\gamma^\nu\,(g_1^- \omega_- + g_1^+ \omega_+)\,(\not{q} + \not{p})$

$\not{\epsilon}\,(g_3^- \omega_- + g_3^+ \omega_+)\,(\not{q} - \not{p}_2)\,\gamma^\nu\,(g_2^- \omega_- + g_2^+ \omega_+)\,v(p_2)$

```
In[3]:= 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:

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

The remaining Dirac structure is wrapped with the head `StandardMatrixElement`.

```
Out[3]= ((1 - 2 B0[0, 0, M2] + 2 k2 C0[0, 0, k2, 0, M2, 0] +
      2 k2 PaVe[1, {k2, 0, 0}, {0, 0, M2}] +
      4 PaVe[0, 0, {k2, 0, 0}, {0, 0, M2}])
    (g[1, +] g[2, +] g[3, +]
       StandardMatrixElement[u[p1] gs[ep[k]] ga[6] u[p2]] +
      g[1, -] g[2, -] g[3, -]
       StandardMatrixElement[u[p1] gs[ep[k]] ga[7] u[p2]])) \
  / (16 Pi^2)
```

This reduces the result to scalar integrals.

The default result of `PaVeReduce` is not put over a common denominator. This is achieved by using `Combine`.

```
In[4]:= Wff1 = PaVeReduce[Wff1] // Combine
Out[4]= ((2 k2 - 2 (2 k2 + M2) B0[0, 0, M2] +
      (3 k2 + 2 M2) B0[k2, 0, 0] +
      2 (k2 + M2)^2 C0[0, 0, k2, 0, M2, 0])
    (g[1, +] g[2, +] g[3, +]
       StandardMatrixElement[u[p1] gs[ep[k]] ga[6] u[p2]] +
      g[1, -] g[2, -] g[3, -]
       StandardMatrixElement[u[p1] gs[ep[k]] ga[7] u[p2]])) \
  / (16 k2 Pi^2)
```

With this command you can extract the standard matrix elements.

```
In[5]:= var = Select[Variables[Wff1],
        (Head[#]===StandardMatrixElement)&]
Out[5]= {StandardMatrixElement[u[p1] gs[ep[k]] ga[6] u[p2]],
  StandardMatrixElement[u[p1] gs[ep[k]] ga[7] u[p2]]}
```

Here the `StandardMatrixElement` are set to some abbreviations.

```
In[6]:= Set @@ {var, {MA[1], MA[2]} }
Out[6]= {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[7]:= Write2["Wff1.for", vert =
                Wff1 /. g[i_, "+"] -> gp[i] /. g[i_, "-"] ->
        gm[i],
                FormatType -> FortranForm
              ]
Out[7]= Wff1.for
```

Show the content of the file.

```
In[8]:= !!Wff1.for

      vert = (2*k2 + (3*k2 + 2*M2)*B0(k2,Null,Null) -
     -     2*(2*k2 + M2)*B0(Null,M2,Null) +
     -     2*(k2 + M2)**2*
     -      C0(Null,Null,k2,Null,M2,Null))*
     -   (gp(1)*gp(2)*gp(3)*MA(1) +
     -     gm(1)*gm(2)*gm(3)*MA(2))/(16*k2*Pi**2)
```

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

A head for identifying standard matrix elements; $sm1, sm2$ are the standard matrixelements, $abb1, abb2$ 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 `G`, `FV` and `feynden` stand for metric tensors, four-vectors and denominators of propagators.

```
In[1]:= (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 -> MAT} ];
         G[x_ y_] := MetricTensor[x, y];
         FV[p_, m_] := FourVector[p, m];
         feynden[x:{_, _}..] := FeynAmpDenominator @@
           Map[Apply[PropagatorDenominator, #]&, {x}] );
```

This sets the standard matrix elements:

$\mathcal{M}_1^+ = \overline{u}(p_1) \not{\epsilon}_1 \omega_+ v(p_2)$

$\mathcal{M}_1^- = \overline{u}(p_1) \not{\epsilon}_1 \omega_- v(p_2)$

```
In[2]:= 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[3]:= Wff2 = OneLoop[wff2, 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 ( G[ro mu] FV[p1 + 2 p2 - q, nu] -
                      G[mu nu] FV[2 p1 + p2 + q, ro] +
                      G[nu ro] FV[2 q + p1 - p2, mu] ) *
                 PolarizationVector[k, mu]
                      ] /. (M1^2) -> M12 /. (M2^2) -> M22

Out[3]= -(g3 ((2 k2 + M12) BO[0, 0, M12] +

       (2 k2 + M22) BO[0, 0, M22] -

       (k2 + M12 + M22) BO[k2, M12, M22] +

       2 (k2 M12 + k2 M22 + M12 M22)

        C0[0, 0, k2, M12, 0, M22])

     (gp[1] gp[2] MAT[1] + gm[1] gm[2] MAT[2])) / (16 k2 Pi^2)
```

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

```
In[4]:= !!wff2.for

      wff2 = -(g3*(-((k2 + M1**2 + M2**2)*BO(k2,M1**2,M2**2)) +
     -       (2*k2 + M1**2)*BO(Null,M1**2,Null) +
     -       (2*k2 + M2**2)*BO(Null,M2**2,Null) +
     -       2*(k2*M1**2 + k2*M2**2 + M1**2*M2**2)*
     -        C0(Null,Null,k2,M1**2,Null,M2**2))*
     -     (gp(1)*gp(2)*MAT(1) +
     -       gm(1)*gm(2)*MAT(2)))/(16*k2*Pi**2)
```

The result is also autmatically assigned to `OneLoopResult[`*name*`]`.

```
In[4]:= OneLoopResult[wff2]

Out[4]= -(g3 ((2 k2 + M1^2) BO[0, 0, M1^2] +

       (2 k2 + M2^2) BO[0, 0, M2^2] -

       (k2 + M1^2 + M2^2) BO[k2, M1^2, M2^2] +

       2 (k2 M1^2 + k2 M2^2 + M1^2 M2^2)

        C0[0, 0, k2, M1^2, 0, M2^2])

     (gp[1] gp[2] MAT[1] + gm[1] gm[2] MAT[2])) / (16 k2 Pi^2)
```

`OneLoopResult[`*name*`]`	is set to the result of `OneLoop[`*name*`, ...]`

`OneLoopResult[`*name*`]` gets valus as a side effect of `OneLoop`.

3.4 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 3.3.6 shows the use of some options.

option name	*default value*	
CancelQ2	True	whether to cancel q^2
CancelQP	True	whether to cancel $q \cdot p$
DenominatorOrder	True	order the arguments of FeynAmpDenominator canonically
Dimension	True	whether to change the dimension to D
Factoring	False	whether to factor the result
FinalSubstitutions	{}	substitutions done at the end of the calculation
FormatType	InputForm	in which language to write out the result file
InitialSubstitutions	{}	substitutions done at the beginning of the calculation, *i.e.*, energy momentum conservation
IsolateHead	False	whether to use Isolate on the result
Mandelstam	{}	indicate the Mandelstam relation
Prefactor	1	additional 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 around the head Small
WriteOut	True	write out a result file

Options of OneLoop.

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 3.3.2, 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 IsolateHead is set to K, for example, the result will be given as a K[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 CancelQ2 may be altered to False, inhibiting then the cancellation of the first propagator in the denominator by replacing q^2 in the numerator by $q^2 \to (q^2 - m^2) + m^2$.

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 3.3.6.

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.

3.5 OneLoopSum and its Options

If you want to sum a list of amplitudes two different possibilities for summing amplitudes are provided with 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[FeynAmpList[...][FeynAmp[GraphName[..., N1], q, amp1], FeynAmp[GraphName[..., N2], q, amp2], ...]]	calculate a list of amplitudes
OneLoopSum[*expr*]	sum already calculated amplitudes

A general function for summing amplitudes. FeynAmpList, FeynAmp and GraphName are heads of *FeynArts*.

The input of OneLoopSum is adapted to the output of *FeynArts*. After saving a list of Feynman diagrams created in *FeynArts* with the function CreateFeynAmp, *i.e.* CreateFeynAmp[*ins*]>>"eezhb.amp", you can start a new *Mathematica* session [1], load *FeynCalc* and get the amplitudes by am = <<eezhb.amp [2].

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.

[1] It is currently not possible to load *FeynCalc* and *FeynArts* simultaneously into one *Mathematica* session.

[2] Sometimes you will get an error message when loading a file. This is a bug in the *Mathematica* saving routine and may be fixed by editing the file and enclosing the whole expression in two round brackets (...). Do not put a ";" after the last round bracket.

option name	*default value*	
`CombineGraphs`	`{}`	whether to combine amplitudes
`IsolateHead`	`K`	whether to `Isolate` the result
`Mandelstam`	`{}`	whether to use the Mandelstam relation
`SelectGraphs`	`All`	which graphs to select
`ReduceToScalars`	`True`	whether to reduce to scalar integrals
`WriteOutPaVe`	`False`	whether to write out the reduced `PaVe`

Options of `OneLoopSum`

With the default options `OneLoopSum` calculates each amplitude separately by subsituting `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.

With the setting of 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 emtpy string (which means that the files are written in the current directory), you can specify another directory.

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.

3.6 The 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 one page of process-dependent definitions. The unmodified output of *FeynArts* for the amplitudes is directly used as input for *FeynCalc*. The resulting Fortran file is then obtained without any interactive action.

The six standard matrix elements are:

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

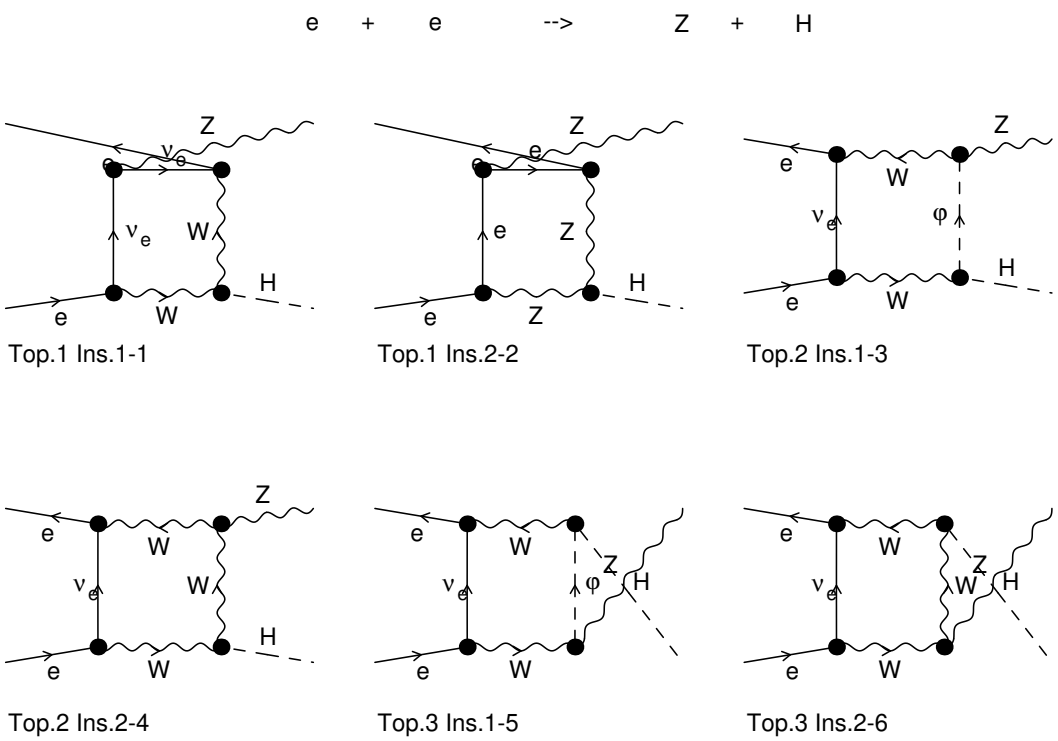

Figure 2: The six box diagrams of $e^+e^- \to ZH$, generated by *FeynArts*. φ denotes the unphysical charged Higgs.

The *FeynArts* convention for the momenta is to assign p_1 to the particle north west and count down on the incoming side. The next momentum is then given to the most north eastern particle, here the Z-boson (k_1), and the Higgs-boson gets momentum k_2.

```
(* Define the Mandelstam variables and put momenta on-shell *)
SetMandelstam[S, T, U, p1, p2, -k1, -k2, Small[ME], Small[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}
          ];
(* 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 matrixelements *)
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 saved and written into a Fortran file *)
Write2["eezhb.s", EEZHBOXES = eezhboxes];
Write2["eezhb.for", EEZHBOXES = eezhboxes, FormatType -> FortranForm];
```

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

```
      K(1) = 2*MH2 - MZ2
      K(2) = MH2 - 5*MZ2
      K(3) = 1 - 2*SW2
      K(4) = MH2 + MZ2
      K(5) = MH2 + 2*MZ2
      K(6) = 4*MH2 + 3*MZ2
      K(7) = MH2 - 3*MZ2
      K(8) = MH2 - MZ2
      K(9) = 2*MH2**2 + 4*MH2*MZ2 + MZ2**2
      K(10) = 2*MH2 + MZ2
      K(11) = MH2 - 7*MZ2
      K(12) = 2*MH2 - 5*MZ2
      K(13) = 3*MH2 - 2*MZ2
      K(14) = MH2**2 - 2*MH2*MZ2 - MZ2**2
      K(15) = 2*MW2 - MZ2
      K(16) = 2*MW2**2 - MZ2**2 + 6*MZ2**2*SW2 -
     -  12*MZ2**2*SW2**2 + 8*MZ2**2*SW2**3
      K(17) = 3*MH2 + 4*MZ2
      K(18) = 5*MH2 + 4*MZ2
      K(19) = MH2 + 5*MZ2
      K(20) = 3*MH2 + 2*MZ2
      K(21) = MH2 + 4*MZ2
      K(22) = 7*MH2 + 4*MZ2
      K(23) = MH - 2*MZ
      K(24) = MH + 2*MZ
      K(25) = 3*MH2 + 8*MZ2
      K(26) = MH2 - 6*MZ2
      K(27) = 5*MH2 + 6*MZ2
      K(28) = MH2 + 6*MZ2
      K(29) = 4*MH2 + MZ2
      K(30) = 2*MH2**2 + 13*MH2*MZ2 + 7*MZ2**2
      K(31) = 8*MH2 - MZ2
      K(32) = 6*MH2**2 + 14*MH2*MZ2 + 3*MZ2**2
      K(33) = 6*MH2 + 5*MZ2
      K(34) = B0(MH2,MZ,MZ)
      K(35) = B0(MZ2,0D0,0D0)
      K(36) = B0(MZ2,MW,MW)
      K(37) = B0(MH2,MW,MW)
      K(38) = D0(MH2,ME2,MZ2,ME2,T,U,MZ2,MZ2,ME2,ME2)
      K(39) = D0(MH2,MZ2,ME2,ME2,S,T,MW2,MW2,MW2,0D0)
      K(40) = D0(MH2,MZ2,ME2,ME2,S,U,MW2,MW2,MW2,0D0)
      K(41) = D0(MH2,ME2,MZ2,ME2,T,U,MW2,MW2,0D0,0D0)
      K(42) = C0(MH2,T,ME2,MZ,MZ,ME)
      K(43) = C0(MH2,U,ME2,MZ,MZ,ME)
      K(44) = C0(MZ2,T,ME2,ME,ME,MZ)
      K(45) = C0(MZ2,U,ME2,ME,ME,MZ)
      K(46) = C0(MH2,MZ2,S,MW,MW,MW)
      K(47) = C0(MH2,T,ME2,MW,MW,0D0)
      K(48) = C0(MH2,U,ME2,MW,MW,0D0)
      K(49) = C0(MZ2,T,ME2,0D0,0D0,MW)
      K(50) = C0(MZ2,T,ME2,MW,MW,0D0)
      K(51) = C0(MZ2,U,ME2,0D0,0D0,MW)
      K(52) = C0(MZ2,U,ME2,MW,MW,0D0)
      K(53) = C0(S,ME2,ME2,MW,MW,0D0)
      K(54) = B0(T,0D0,MZ)
      K(55) = B0(U,0D0,MZ)
      K(56) = B0(T,0D0,MW)
      K(57) = B0(U,0D0,MW)
      K(58) = MH2*MZ2 - T*U
      K(59) = 4*MH2*MZ2**2 - 3*MH2*MZ2*T + MH2*MZ2*U -
     -  3*MZ2*T*U + 2*T**2*U - MZ2*U**2
C --->   ...  356 lines omitted ...
      K(241) = K(235)*MBM(0,1) + K(236)*MBM(0,2) +
     -  K(237)*MBM(1,1) + K(238)*MBM(1,2) +
     -  K(239)*MBM(2,1) + K(240)*MBM(2,2)
      K(242) = 2*ALP4PI*FLUFAC*K(241)
      EEZHBOXES = K(242)
```

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

```
K[1] = 2*MH2 - MZ2
K[2] = MH2 - 5*MZ2
K[3] = 1 - 2*SW2
K[4] = MH2 + MZ2
K[5] = MH2 + 2*MZ2
K[6] = 4*MH2 + 3*MZ2
K[7] = MH2 - 3*MZ2
K[8] = MH2 - MZ2
K[9] = 2*MH2^2 + 4*MH2*MZ2 + MZ2^2
K[10] = 2*MH2 + MZ2
K[11] = MH2 - 7*MZ2
K[12] = 2*MH2 - 5*MZ2
K[13] = 3*MH2 - 2*MZ2
K[14] = MH2^2 - 2*MH2*MZ2 - MZ2^2
K[15] = 2*MW2 - MZ2
K[16] = 2*MW2^2 - MZ2^2 + 6*MZ2^2*SW2 - 12*MZ2^2*SW2^2 + 8*MZ2^2*SW2^3
K[17] = 3*MH2 + 4*MZ2
K[18] = 5*MH2 + 4*MZ2
K[19] = MH2 + 5*MZ2
K[20] = 3*MH2 + 2*MZ2
K[21] = MH2 + 4*MZ2
K[22] = 7*MH2 + 4*MZ2
K[23] = MH - 2*MZ
K[24] = MH + 2*MZ
K[25] = 3*MH2 + 8*MZ2
K[26] = MH2 - 6*MZ2
K[27] = 5*MH2 + 6*MZ2
K[28] = MH2 + 6*MZ2
K[29] = 4*MH2 + MZ2
K[30] = 2*MH2^2 + 13*MH2*MZ2 + 7*MZ2^2
K[31] = 8*MH2 - MZ2
K[32] = 6*MH2^2 + 14*MH2*MZ2 + 3*MZ2^2
K[33] = 6*MH2 + 5*MZ2
K[34] = B0[MH2, MZ2, MZ2]
K[35] = B0[MZ2, 0, 0]
K[36] = B0[MZ2, MW2, MW2]
K[37] = B0[MH2, MW2, MW2]
K[38] = D0[MH2, Small[ME2], MZ2, Small[ME2], T, U, MZ2, MZ2, Small[ME2], Small[ME2]]
K[39] = D0[MH2, MZ2, Small[ME2], Small[ME2], S, T, MW2, MW2, MW2, 0]
K[40] = D0[MH2, MZ2, Small[ME2], Small[ME2], S, U, MW2, MW2, MW2, 0]
K[41] = D0[MH2, Small[ME2], MZ2, Small[ME2], T, U, MW2, MW2, 0, 0]
K[42] = C0[MH2, T, Small[ME2], MZ2, MZ2, Small[ME2]]
K[43] = C0[MH2, U, Small[ME2], MZ2, MZ2, Small[ME2]]
K[44] = C0[MZ2, T, Small[ME2], Small[ME2], Small[ME2], MZ2]
K[45] = C0[MZ2, U, Small[ME2], Small[ME2], Small[ME2], MZ2]
K[46] = C0[MH2, MZ2, S, MW2, MW2, MW2]
K[47] = C0[MH2, T, Small[ME2], MW2, MW2, 0]
K[48] = C0[MH2, U, Small[ME2], MW2, MW2, 0]
K[49] = C0[MZ2, T, Small[ME2], 0, 0, MW2]
K[50] = C0[MZ2, T, Small[ME2], MW2, MW2, 0]
K[51] = C0[MZ2, U, Small[ME2], 0, 0, MW2]
K[52] = C0[MZ2, U, Small[ME2], MW2, MW2, 0]
K[53] = C0[S, Small[ME2], Small[ME2], MW2, MW2, 0]
K[54] = B0[T, 0, MZ2]
K[55] = B0[U, 0, MZ2]
K[56] = B0[T, 0, MW2]
K[57] = B0[U, 0, MW2]
K[58] = MH2*MZ2 - T*U
K[59] = 4*MH2*MZ2^2 - 3*MH2*MZ2*T + MH2*MZ2*U - 3*MZ2*T*U + 2*T^2*U - MZ2*U^2
K[60] = (MZ^5*SW2*HoldForm[K[59]])/(MW2^2*S*HoldForm[K[58]])
  (*  ..............  338 lines omitted  ....................  *)
K[241] = HoldForm[K[235]]*MBM[0, 1] + HoldForm[K[236]]*MBM[0, 2] +
  HoldForm[K[237]]*MBM[1, 1] + HoldForm[K[238]]*MBM[1, 2] +
  HoldForm[K[239]]*MBM[2, 1] + HoldForm[K[240]]*MBM[2, 2]
K[242] = 2*ALP4PI*FLUFAC*HoldForm[K[241]]
EEZHBOXES = HoldForm[K[242]]
```

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

4 Miscellaneous Functions

4.1 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`]`	group together powers of terms containing x
`Collect2[`*expr*`, {`x_1`, `x_2`, ...}]`	group together powers of terms containing $x_1, x_2, \ldots$
`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[1]:= Collect2[ (b - 2) d f[x] + f[x] + c p[y] + p[y], {f, y}]
Out[1]= (1 + (-2 + b) d) f[x] + (1 + c) p[y]
```

With the setting `ProductExpand -> True` the product $3(s+m^2)$ gets expanded.

```
In[2]:= Collect2[3 B0[pp,m1,m2] (m^2 + s) + B0[pp,m1,m2] s,
                 B0, ProductExpand -> True]
                                  2
Out[2]= (3 m  + 4 s) B0[pp, m1, m2]
```

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

```
In[3]:= Combine[(a - b) (c - d)/e + g]
                (a - b) (c - d) + e g
Out[3]= ---------------------------
                      e
```

Consider a generic polynomial.

```
In[4]:= test = (a - b) x + (b - a) y
Out[4]= (a - b) x + (-a + b) y
```

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

```
In[5]:= Map[Factor, test]
Out[5]= (a - b) x + (-a + b) y
```

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

```
In[6]:= Map[Factor2, test]
Out[6]= (a - b) x - (a - b) y
```

This is the overall factorization of `Factor`.

```
In[7]:= Factor[test]
Out[7]= (-a + b) (-x + y)
```

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

```
In[8]:= Factor2[test]
Out[8]= (a - b) (x - y)
```

option name	*default value*	
`ProductExpand`	`False`	whether to expand products

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

4.2 An Isolating Function for Automatically Introducing Abbreviations

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

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[1]:= Isolate[a + b]
Out[1]= K[1]
```

Here *f* gets isolated with `K[1]` and `K[2]` replacing the bracketed subsums.

```
In[2]:= test = Isolate[(a + b) f + (c + d) f + e, f]
Out[2]= e + f K[1] + f K[2]
```

Looking at the `FullForm` reveals that the `K[`*i*`]` are given in `HoldForm`.

```
In[3]:= FullForm[test]
Out[3]//FullForm=
 Plus[e, Times[f, HoldForm[K[1]]], Times[f, HoldForm[K[2]]]]
```

Asking for `K[1]` returns its value, but in `test` `K[1]` is held.

```
In[4]:= { K[1], test, ReleaseHold[test] }
Out[4]= {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[5]:= Isolate[a[z] (b + c (y + z)) + d[z] (y + z), {a, d},
                IsolateHead -> F]
Out[5]= 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[6]:= ( 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[6]= d[z] H[1] + a[z] H[2]
```

Here it is clear that `H[1]` = (y + z) is part of `H[2]`.

```
In[7]:= ReleaseHold[%]
Out[7]= (y + z) d[z] + a[z] (b + c H[1])
```

Decreasing the option `IsolateSplit` significantly forces `Isolate` to use more than one `L` for `a - b - c - d - e`.

```
In[8]:= Isolate[ a - b - c - d - e, IsolateHead -> L,
        IsolateSplit -> 15 ]
Out[8]= L[2]
```

This shows the values of the `L`.

```
In[9]:= {L[2], L[1]}
Out[9]= {-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 4.4.4 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[#]]&.

4.3 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 $f_1, f_2, \ldots$
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[1]:= FreeQ2[M^2 + m^2 B0[pp, m1, m2], {M, B0}]
Out[1]= False
```

This gives the numerical factor of the expression.

```
In[2]:= NumericalFactor[-137 x]
Out[2]= -137
```

The action of PartitHead on a product is to split the product apart.

```
In[3]:= PartitHead[f[m] (s - u), f]
Out[3]= {s - u, f[m]}
```

A sum gets separated into subsums.

```
In[4]:= PartitHead[s^2 + M^2 - f[m], f]
                 2    2
Out[4]= {M  + s , -f[m]}
```

4.4 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
`DOConvention`	0	which convention to use for the scalar integrals

Options for `Write2`.

With the default setting 0 of `DOConvention` 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[1]:= tpol = z + Isolate[Isolate[2 Pi I + f[x] (a - b), f]]
Out[1]= z + K[2]
```

The default writes out in *Mathematica* `InputForm`.

```
In[2]:= Write2["test.m", test = tpol];
```

Show the content of the file.

```
In[3]:= !!test.m

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[3]:= Write2["test.for", test = tpol,
                FormatType -> FortranForm];
```

Show the content of the Fortran file.

```
In[4]:= !!test.for
      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[4]:= Write2["test.mac", test = tpol,
                FormatType -> MacsymaForm];
```

Here the *Macsyma* format can be seen.

```
In[5]:= !!test.mac
test : ( 2*%i*%pi + z + (a - b)*f(x) )$
```

Here we get a file with *Maple* conventions.

```
In[5]:= Write2["test.map", test = tpol,
                FormatType -> MapleForm];
```

The *Maple* conventions are only slightly different.

```
In[6]:= !!test.map
test := ( 2*I*Pi + z + (a - b)*f(x) );
```

■ 4.5 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[1]:= Contract[ LeviCivita[m, n, r, s] FourVector[p + q,s] ]
Out[1]= 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[2]:= EpsEvaluate[ % ]
Out[2]= eps[m, n, r, p] + eps[m, n, r, q]
```

This does not evaluate directly to 0.

```
In[3]:= LeviCivita[a, b, c, d] /. d -> c
Out[3]= eps[a, b, c, c]
```

Like this you can always evaluate ε's.

```
In[4]:= EpsEvaluate[ % ]
Out[4]= 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[5]:= EpsChisholm[ LeviCivita[a, b, c, d] DiracMatrix[mu, a] ]
Out[5]= +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 4.4.9.

This clears `$PrePrint`.

```
In[6]:= $PrePrint=.
```

This shows the internal structure.

```
In[7]:= LeviCivita[m, n, r, s]
Out[7]= 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[8]:= Contract[ % FourVector[p, s]]
Out[8]= Eps[LorentzIndex[m], LorentzIndex[n],
  LorentzIndex[r], Momentum[p]]
```

4.6 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]`	$-g^{\mu\nu} - k_{\mu}k_{\nu}n^2/(k\cdot n)^2 + (n_{\mu}k_{\nu} + n_{\nu}k_{\mu})/(k\cdot n)$

Three 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[1]:= PolarizationSum[mu, nu, k]
```

$$\text{Out[1]= -g[mu, nu] + } \frac{\text{k[mu] k[nu]}}{\text{k.k}}$$

Here the polarization sum for gluons is given; with external momentum $n = p_1 - p_2$.

```
In[2]:= PolarizationSum[mu, nu, k, p1 - p2]
```

$$\text{Out[2]= -g[mu, nu] - } \frac{\text{(p1.p1 - 2 p1.p2 + p2.p2) k[mu] k[nu]}}{\text{(k.p1 - k.p2)}^2} + \frac{\text{k[nu] (p1 - p2)[mu] + k[mu] (p1 - p2)[nu]}}{\text{k.p1 - k.p2}}$$

4.7 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[1]:= TrickMandelstam[(s + t - u) (2 mw2 - t - u),
                        {s, t, u, 2 mw2}]
Out[1]= 2 s (mw2 - u)
```

The result is always given in a factorized form.

```
In[2]:= TrickMandelstam[M^2 s - s^2 + M^2 t - s t + M^2 u - s u,
                        {s, t, u, 2 M^2}]

                  2   2
Out[2]= 2 M^2 (M^2 - s)
```

4.8 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[1]:= PaVeOrder[D0[me2, me2, mw2, mw2, t, s, me2, 0, me2, 0],
                  PaVeOrderList -> {me2, me2, 0, 0}]

Out[1]= D0[me2, s, mw2, t, mw2, me2, me2, 0, 0, me2]
```

This interchanges the *f* and *e*.

```
In[2]:= PaVeOrder[D0[a, b, c, d, e, f, m12, m22, m32, m42],
                  PaVeOrderList -> {f, e}]

Out[2]= D0[a, d, c, b, f, e, m22, m12, m42, m32]
```

This shows how to permute several D_0.

```
In[3]:= 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[3]= D0[a, d, c, b, f, e, m22, m12, m42, m32] +
  D0[me2, s, mw2, t, mw2, me2, me2, 0, 0, me2]
```

4.9 On the Internal Representation

With the default setting of `$PrePrint = FeynCalcForm` the internal representation is hidden. However for special purposes it may be useful to know how the various objects are represented in *FeynCalc*. 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`. The data-type heads usually 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 very useful and reliable for the purpose of calculations in physics.

The exported heads, which often also have simple functional definitions, are listed briefly in this section.

`DiracGamma[`x`]`	head of a four-dimensional γ^μ or $\not{p}$
`DiracGamma[`x`, `d`]`	head of a d-dimensional γ^μ or $\not{p}$
`LorentzIndex[`mu`]`	head of a four-dimensional Lorentz index μ
`LorentzIndex[`mu`, `d`]`	head of a d-dimensional Lorentz index μ
`Momentum[`p`]`	head of a four-dimensional momentum p
`Momentum[`p`, `d`]`	head of a d-dimensional momentum p

Heads of the internal representation.

Clear `$PrePrint`.

```
In[1]:= $PrePrint=.
```

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

```
In[2]:= DiracMatrix[mu]
Out[2]= DiracGamma[LorentzIndex[mu]]
```

Here is a γ^α in D dimensions.

```
In[3]:= DiracMatrix[al, Dimension -> D]
Out[3]= DiracGamma[LorentzIndex[al, D], D]
```

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

```
In[4]:= DiracSlash[q] . DiracSlash[q, Dimension -> D]
Out[4]= DiracGamma[Momentum[q]] .
  DiracGamma[Momentum[q, D], D]
```

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

```
In[5]:= DiracSimplify[%]
Out[5]= Pair[Momentum[q], Momentum[q]]
```

The head `Pair` is also used for metric tensors.

```
In[6]:= MetricTensor[mu, nu]
Out[6]= Pair[LorentzIndex[mu], LorentzIndex[nu]]
```

Clearly a four-vector is also represented as a pair internally.

```
In[7]:= FourVector[l, mu]
Out[7]= Pair[LorentzIndex[mu], Momentum[l]]
```

The dimension is provided as an argument.

```
In[8]:= MetricTensor[a, b, Dimension -> d]
Out[8]= Pair[LorentzIndex[a, d], LorentzIndex[b, d]]
```

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

```
In[9]:= FourVector[p, mu, Dimension -> D - 4]
Out[9]= Pair[LorentzIndex[mu, -4 + D], Momentum[p, -4 + D]]
```

Obviously it vanishes after changing the dimension.

```
In[10]:= % /. D -> 4
Out[10]= 0
```

This shows the internal representations of polarization vectors.

```
In[11]:= PolarizationVector[k, mu]
Out[11]= Pair[LorentzIndex[mu], Momentum[Polarization[k]]]
```

`Pair[`a`, `b`]`	the head of $g^{\mu\nu}$, p^μ and $(p \cdot q)$

The head for a pairing of `LorentzIndex` and `Momentum`.

4.10 FeynCalcForm

`FeynCalcForm[`*expr*`]`	displays *expr* in an easy to read form

The print form of *FeynCalc*.

`FeynCalcForm` changes the internal representation of *FeynCalc* to a shorter and more readable form. The effect of `FeynCalcForm` can be investigated by clearing `$PrePrint` (`$PrePrint=.`) and applying `FeynCalcForm` on the various objects.

The default setting of `$PrePrint` is `FeynCalcForm`.

The effect of `FeynCalcForm` can be seen in the examples.

4.11 Three New Global Variables

option name	*default value*	
`$BreitMaison`	`False`	whether to use the Breitenlohner-Maison scheme
`$VeryVerbose`	`0`	display intermediate messages from *FeynCalc*
`$MemoryAvailable`	`8`	the amount of available main memory in MB

The three global variables of *FeynCalc*.

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: 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. Therefore one should check the results for correctness.

If you want to change the default setting of `BreitMaison` you must set it *before* loading *FeynCalc*.

In the following example the automatic loading of *FeynCalc* from the `init.m` file has been turned off.

Set the variable `$BreitMaison` before loading *FeynCalc*.

```
In[1]:= $BreitMaison = True
Out[1]= True
```

Loading *FeynCalc*.

```
In[2]:= <<FeynCalc.m
```

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

```
In[3]:= DiracMatrix[5] . DiracMatrix[mu, Dimension ->D]
Out[3]= ga[5] ga [mu]
                 D
```

Now only the 4-dimensional part of γ^μ anticommutes, while the $D-4$ dimensional one commutes.

```
In[4]:= DiracSimplify[%]
Out[4]= -ga[mu] ga[5] + ga_{D-4}[mu] ga[5]
```

The chirality projectors ω_+ and ω_- are inserted automatically.

```
In[5]:= ChiralityProjector[+1] . DiracMatrix[mu, Dimension ->D]
```

$$\text{Out[5]=}\ \left(\frac{1}{2} + \frac{\text{ga[5]}}{2}\right) \text{ga}_D\text{[mu]}$$

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

```
In[6]:= DiracSimplify[%]
```

$$\text{Out[6]=}\ \frac{\text{-ga[mu] ga[5]}}{2} + \frac{\text{ga}_{D-4}\text{[mu] ga[5]}}{2} + \frac{\text{ga}_D\text{[mu]}}{2}$$

This is *FeynCalc* does not simplify this sum directly, but you can easily specify which dimension to "eliminate".

```
In[7]:= test = 2 DiracMatrix[mu] +
              DiracMatrix[mu, Dimension -> D] +
              DiracMatrix[mu, Dimension -> D-4]
Out[7]= 2 ga[mu] + ga_{D-4}[mu] + ga_D[mu]
```

Switch to the internal representation.

```
In[8]:= ($PrePrint=. ; test)
Out[8]= 2 DiracGamma[LorentzIndex[mu]] +
  DiracGamma[LorentzIndex[mu, -4 + D], -4 + D] +
  DiracGamma[LorentzIndex[mu, D], D]
```

This eliminates each Dirac matrix in $D-4$ dimension.

```
In[9]:= DiracGamma[x_[y_, d_Symbol - 4], d_Symbol - 4] :=
        DiracGamma[x[y, d], d] - DiracGamma[x[y]]
```

Only D- and 4-dimensional objects are left.

```
In[10]:= test
Out[10]= DiracGamma[LorentzIndex[mu]] +
  2 DiracGamma[LorentzIndex[mu, D], D]
```

■ References

[1] J. Küblbeck, M. Böhm and A. Denner, *Comp. Phys. Comm.* **60** (1990) 165.

[2] 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.

[3] R. Mertig, M. Böhm, and A. Denner, *Comp. Phys. Comm.* **64** (1991) 345.

[4] E. Yehudai and A. C. K. Hsieh, *HIP* — Symbolic High-Energy Physics Calculations, *SLAC-PUB-* **5576** July 1991.

[5] G. J. van Oldenborgh, FF – a package to evaluate one-loop Feynman diagrams, *Comp. Phys. Comm.* **66** (1991) 1.

[6] P.Cvitanovic, *Phys. Rev. D* **14** (1976) 1536.

[7] A.P. Kryukov and A.Ya. Rodionov, *Comp. Phys. Comm.* **48** (1988) 327.

[8] Matthias Jamin and E. Lautenbacher, TRACER, A Mathematica Package for γ-Algebra in Arbitrary Dimensions, preprint Technische Universität München, TUM-T31-20/91.

[9] S. Wolfram, MACSYMA Tools for Feynman Diagram Calculations, *Proceedings of the 1979 MACSYMA Users Conference*.

5 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	whether to replace $A_0(m^2)$ by $m^2(1 + B_0(0, m^2, m^2))$

■ See page 21.

■ A0ToB0

A0ToB0 is an option for A0. If set to True, $A_0(m^2)$ is replaced by $m^2(1 + B_0(0, m^2, m^2))$.

■ See page 23.

■ B0

B0[pp, m1^2, m2^2] is the scalar Passarino-Veltman two-point function.

B0[pp, m1^2, m2^2] 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	whether to reduce B0 in special cases to A0
B0Unique	False	whether to replace $B_0(a, 0, a)$ by $(2 + B_0(0, a, a))$ and $B_0(0, 0, a)$ by $(1 + B_0(0, a, a))$.

■ See page 21.

■ B00

B00[pp, m1^2, m2^2] is the coefficient of $g^{\mu\nu}$ of the tensor integral decomposition of $B^{\mu\nu}$.

The following option can be given:

BReduce	True	whether to reduce B00 to B1 and A0

■ If BReduce is set to True the following simplification holds:
■ $B_{00}(a, b, c) = 1/6(A_0(b) + (B_1(a, b, c)(a - c + b) + a/3B_0(a, b, c) + 1/6(a + b - c/3)))$. ■ See page 23.

■ 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. ■ See page 23.

■ B1

B1[pp, m1^2, m2^2] is the coefficient of p^μ of the tensor integral decomposition of B^μ.

The following option can be given:

BReduce	True	whether to 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.
■ $B_1(a, b, c) = 1/2(A_0(b) - A_0(b) - (a - c + b)B_0(a, b, c))$. ■ $B_1(a, b, b) = -1/2B_0(a, b, b)$.
■ $B_1(a, a, 0) = -1/2B_0(a, a, 0) - 1/2$. ■ $B_1(a, 0, a) = 1/2 - 1/2B_0(a, 0, m)$. ■ $B_1(0, 0, a) = -1/2B_0(0, 0, a) + 1/4$.
■ $B_1(0, a, 0) = -1/2B_0(0, a, 0) - 1/4$. ■ See page 23.

■ B11

B11[pp, m1^2, m2^2] 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	whether to 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.
■ $B_{11}(a, b, c) = 1/(3a)(A_0(b) - 2(a - c + b)B_1(a, b, c) - aB_0(a, b, c) - 1/2(a + b - c/3))$. ■ $B_{11}(0, a, a) = 1/3B_0(0, a, a)$. ■ See page 23.

■ 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. ■ See page 23.

■ 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. ■ See page 30.

■ 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.

■ See page 30.

■ 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 page 9. ■ 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 page 21. ■ 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	whether to expand products in *expr* free of x (x_1, x_2, ...)
IsolateHead	False	whether to isolate with respect to {x_1, x_2, ...}
IsolateSplit	442	the limit before Isolate splits

■ See page 37. ■ 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	whether to expand products

■ See page 37.

■ 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. ■ See page 31.

■ 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	whether to contract products of Levi-Civita (Eps) tensors via the determinant formula
Expanding	True	expand all sums containing Lorentz indices
Factoring	False	whether to 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 page 13. ■ 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 page 21. ■ 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 page 40. ■ 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 page 21. ■ 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. ■ See page 30.

■ 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. ■ See page 30.

■ 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. ■ See page 43.

■ 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 page 9. ■ 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 page 14. ■ 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 page 14. ■ 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 page 9. ■ 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	whether to factor the result
Mandelstam	{}	if set to {s, t, u, m1^2 + m2^2 + m3^2 + m4^2}, TrickMandelstam will be used
PairCollect	True	whether to 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 page 16. ■ 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. ■ See page 17.

■ 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. ■ See page 42.

■ 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

■ See page 41.

■ 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.

■ See page 13.

■ EpsEvaluate

EpsEvaluate[*expr*] applies total antisymmetry and linearity (with respect to Momentum) to all Levi-Civita tensors (Eps) in *expr*.

■ See page 41.

■ **EvaluateDiracTrace**

EvaluateDiracTrace[*expr*] evaluates DiracTrace in *expr*.

■ See page 17.

■ **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.

■ See page 13.

■ **ExpandScalarProduct**

ExpandScalarProduct[*expr*] expands scalar products in *expr*.

Examples: 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]. ■ See page 14.

■ **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. ■ See page 37.

■ **Factoring**

Factoring is an option for Contract, DiracTrace, DiracSimplify and OneLoop. If set to True the result will be factored, using Factor2.

■ See page 30.

■ **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 page 31. ■ 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 page 11.
■ See also: PropagatorDenominator.

■ **FeynAmpList**

FeynAmpList[*info* ...]*FeynAmp*[...], *FeynAmp*[...], ...] is the head of a list of FeynAmp in the result of *FeynArts*.

■ See page 31. ■ 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]. ■ See page 45.

■ 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}. ■ See page 30.

■ 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 page 7. ■ See also: LorentzIndex, Momentum, Pair.

■ FreeQ2

FreeQ2[*expr*, {*form1*, *form2*, ...}] yields True if *expr* does not contain any occurrence of *form1*, *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* || ...]. ■ See page 39.

■ 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 page 10. ■ 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. ■ See page 20.

■ 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.

■ See page 31.

■ 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 }. ■ See page 30.

■ 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 page 38. ■ See also: Write2.

■ IsolateHead

IsolateHead is an option for Isolate.

■ See page 38.

■ IsolateSplit

IsolateSplit is an option for Isolate.

■ See page 38.

■ 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.

■ See page 24.

■ 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 page 10. ■ 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 page 7. ■ 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 $tr(\gamma^a \gamma^b \gamma^c \gamma^d \gamma^5)$.

■ See page 41.

■ 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]. ■ See page 43.

■ MacsymaForm

MacsymaForm is an option for FormatType in Write2.

■ See page 40. ■ 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 $s + t + u = m_1^2 + m_2^2 + m_3^2 + m_4^2$.

■ See page 30.

■ MapleForm

MapleForm is an option for Write2.

■ See page 40. ■ See also: Write2, MacsymaForm.

■ MetricTensor

MetricTensor[*mu*, *nu*] is the input for a metric tensor $g^{\mu\nu}$.

The following option can be given:

Dimension	4	space-time dimension

■ See page 7. ■ 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*].

■ See page 43. ■ Momentum[*p*, *4*] is automatically transformed to Momentum[*p*].

■ NumericalFactor

NumericalFactor[*expr*] gives the numerical factor of *expr*.

■ See page 39.

■ 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	True	order the entries of FeynAmpDenominator
Dimension	True	dimension of integration
FinalSubstitutions	{}	substitutions done at the end of the calculation
Factoring	False	whether to 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	whether to cancel q^2
CancelQP	True	whether to cancel $q \cdot p$
ReduceGamma	False	whether to 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. ■ See page 30.

■ 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.

■ See page 29.

■ 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	whether to Isolate the result
Mandelstam	{}	whether to 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	whether to 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. ■ See page 31.

■ 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 page 44. ■ See also: FourVector, LorentzIndex, MetricTensor, ExpandScalarProduct, ScalarProduct.

■ PairCollect

PairCollect is an option for Contract.

■ See page 17. ■ 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 $i, j, \ldots$ 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. ■ See page 22.

■ PaVeOrder

PaVeOrder[*expr*] brings all arguments of C0 and D0 into a canical order.

The following option can be given:

PaVeOrderList	{}	whether to order according to a list of arguments

■ See page 43.

■ 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. ■ See page 43.

■ 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^μ, $B^{\mu\nu}$, C^μ, $C^{\mu\nu}$, $C^{\mu\nu\rho}$, D^μ, $D^{\mu\nu}$, $D^{\mu\nu\rho}$, $D^{\mu\nu\rho\sigma}$. ■ The following options can be given:

IsolateHead	False	whether to use Isolate
Mandelstam	{}	Mandelstam relation, *e.g.*, {s, t, u, 2 mw^2}

■ See page 24. ■ 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 page 8. ■ 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. ■ See page 42.

■ 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 page 8. ■ 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. ■ See page 30.

■ ProductExpand

ProductExpand is an option for Collect2 and Combine.

■ See page 37.

■ 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. ■ See page 11.

■ 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 page 10. ■ 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. ■ See page 30.

■ 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 page 31. ■ 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 page 7. ■ 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.

■ See page 31.

■ 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] ■ See page 17.

■ 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 page 28. ■ 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.

■ See page 12.

■ 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. ■ See page 30.

■ 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 page 10. ■ 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.

■ See page 28.

SU3Delta

SU3Delta[*a*, *b*] is the Kronecker δ with color indices *a* and *b*.

SU3Delta[i, i] → 8. ■ See page 10. ■ 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	whether to replace f_{abc} by $\frac{i}{4}(tr(\lambda_a\lambda_c\lambda_b) - tr(\lambda_a\lambda_b\lambda_c))$.

■ See page 10. ■ See also: SU3Delta, GellMannMatrix, GellMannTrace.

SU3FToTraces

SU3FToTraces is an option for SU3F.

■ See page 10.

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	whether to factor the result
Mandelstam	{}	if set to {*s*, *t*, *u*, *m1*^2 + *m2*^2 + *m3*^2 + *m4*^2}, TrickMandelstam will be used
PairCollect	True	whether to collect the results with respect to products of Pair

■ See also: DiracTrace. ■ See page 16.

TrickMandelstam

TrickMandelstam[*expr*, {*s*, *t*, *u*, *m1*^2 + *m2*^2 + *m3*^2 + *m4*^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. ■ See page 42.

Write2

Write2[*channel*, *val1* = *expr1*, *val2* = *expr2*, ...] 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. ■ See page 40.

■ 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

■ See page 30.

■ WriteOutPaVe

WriteOutPave is an option for OneLoopSum. If set to a string the reduced PaVe are written into the file indicated.

■ See page 31.

■ $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! ■ See page 45.

■ $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*.

■ See page 45.

■ $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.

■ See page 45.

Index

A0, 21, 47
A0ToB0, 23, 47

B0, 21, 47
B00, 23, 47
B0Unique, 23, 47
B1, 23, 47
B11, 23, 47
BReduce, 23, 47
$BreitMaison, 45, 60

C0, 21, 48
CancelQ2, 30, 48
CancelQP, 30, 48
ChiralityProjector, 9, 48
Collect2, 37, 48
Combine, 37, 48
CombineGraphs, 31, 48
Contract, 13, 48

D0, 21, 49
D0Convention, 40, 49
DB0, 21, 49
DenominatorOrder, 30, 49
Dimension, 7, 9, 30, 49
DiracGamma, 9, 43, 49
DiracMatrix, 9, 50
DiracOrder, 14, 50
DiracSimplify, 14, 50
DiracSlash, 9, 50
DiracTrace, 16, 50
DiracTraceEvaluate, 17, 51

Eps, 42, 51
EpsChisholm, 41, 51
EpsContract, 13, 51
EpsEvaluate, 41, 51
EvaluateDiracTrace, 17, 51
Expanding, 13, 52
ExpandScalarProduct, 14, 52

Factor2, 37, 52
Factoring, 17, 30, 52
FeynAmp, 31, 52
FeynAmpDenominator, 11, 52
FeynAmpList, 31, 52
FeynCalcForm, 45, 52
FinalSubstitutions, 30, 53
FortranForm, 40
FourVector, 7, 53
FreeQ2, 39, 53

GellMannMatrix, 10, 53
GellMannTrace, 20, 53
GraphName, 31, 53

InitialSubstitutions, 30, 53
Isolate, 38, 53
IsolateHead, 24, 30, 38, 54
IsolateSplit, 38, 54

K, 24, 54

LeptonSpinor, 10, 54
LeviCivita, 7, 54
LeviCivitaSign, 17, 41, 54
LorentzIndex, 43, 54

MacsymaForm, 40, 54
Mandelstam, 17, 24, 30, 55
MapleForm, 40, 55
$MemoryAvailable, 45, 60
MetricTensor, 7, 55
Momentum, 43, 55

NumericalFactor, 39, 55

OneLoop, 30, 55

OneLoopResult, 29, 55
OneLoopSum, 31, 55

Pair, 44, 56
PairCollect, 17, 56
PartitHead, 39
PaVe, 22, 56
PaVeOrder, 43, 56
PaVeOrderList, 43, 56
PaVeReduce, 24, 56
Polarization, 8, 56
PolarizationSum, 42, 57
PolarizationVector, 8, 57
Prefactor, 30, 57
ProductExpand, 37, 57
PropagatorDenominator, 11, 57

QuarkSpinor, 10, 57

ReduceGamma, 30, 57
ReduceToScalars, 30, 31, 57

ScalarProduct, 7, 58
SelectGraphs, 31, 58
SetMandelstam, 17, 58
SetStandardMatrixElements, 28, 58
Small, 12, 58
SmallVariables, 30, 58
Spinor, 10, 58
Spinors, entering of , 10
StandardMatrixElement, 28, 58
SU3Delta, 10, 59
SU3F, 10, 59
SU3FToTraces, 10, 59

Tr, 16, 59
TrickMandelstam, 42, 59
$VeryVerbose, 45, 60

Write2, 40, 59
WriteOut, 30, 59
WriteOutPaVe, 31, 60
$BreitMaison, 45, 60
$MemoryAvailable, 45, 60
$VeryVerbose, 45, 60