SLAC–PUB–5576
July 1991
(T/E)

HIP — Symbolic High-Energy Physics Calculations

Alexander Hsieh and Eran Yehudai

Stanford Linear Accelerator Center
Stanford University, Stanford, California/94309

ABSTRACT

We present HIP — a set of computer packages for symbolic calculation of Feynman diagrams. The packages were designed as an aid in calculating tree-level multi-particle electroweak production processes, though they can be used for a much wider class of calculations. We have used the packages to calculate such processes as $e\gamma \to W\nu$ and $\gamma\gamma \to W^+W^-$ at tree level with arbitrary $W\gamma$ couplings. The packages are written in the computer algebra language *Mathematica* which provides a powerful working environment.

Submitted to *Computers in Physics*

1. Introduction

Present day pertubative calculations in the Standard Model (SM) and its extensions often require tedious algebra calculations. While tree level calculations of two-body production processes in the SM can certainly be done manually, it is often helpful to have a check in the form of an automatic calculation method. Processes involving the production of more than two particles (*e.g.* $e^+e^- \to W^+W^-\gamma$) or complicated Feynman rules (*e.g.* $\gamma\gamma \to W^+W^-$ with arbitrary $W\gamma$ couplings) involve much more difficult calculations in which computerized aid is almost indispensable.

Two approaches have been used to automate calculations of this type. Hagiwara *et al.*[1] have written a set of Fortran subroutines that calculate matrix elements numerically. This approach allows the automatic calculation of complicated tree level processes, but it is restricted to numeric results. An alternative approach introduced by Veltman[2] with *Schoonship* is to allow symbolic manipulation of expressions. While *Schoonship* was written as a special purpose program, with all the necessary algorithms 'hard-wired' into its assembly code, *Reduce*[3] followed a more general approach. Much more mathematical knowledge (*e.g.* integration rules) is incorporated into the program as higher-level lisp routines. The user, building on that basis of knowledge, can then expand the program by writing his or her own routines. The price for that flexibility is naturally paid in performance.

We follow this latter approach by writing our packages in *Mathematica*,[4] one of the newer and most promising of the new generation of symbolic manipulation languages which also includes *Maple*.[5] Using *Mathematica*'s high-level programming language greatly simplifies the task of writing programs. Additionally, the physics calculations are embedded within a powerful environment in which results can be simplified, calculated numerically and plotted. With the rapid advance in computer performance, the CPU-time needed for the calculations is usually negligible compared with the time needed to prepare the input and process the output of the programs. This approach is also being used by the Würzburg group in their programs *FeynArts*[6] and *FeynCalc*.[7]

Unlike *Feyn Arts* we do not attempt to automatically generate the necessary Feynman diagrams. Typically, tree-level diagrams can be easily generated manually. HIP takes as input expressions describing the Feynman diagrams. HIP then provides the user with a rich set of operators by which to manipulate the physical objects occurring in these expressions. The user may, for example, substitute explicit four-vectors and particle polarizations, square the matrix elements to give traces that can then be evaluated or convert the expressions to spinor techniques.[8] As an aid in calculating cross-sections and decay widths, phase-space integrals can be automatically constructed and evaluated symbolically, numerically, or converted to a C or Fortran program.

The traditional method of Feynman diagram calculation involves squaring the matrix element symbolically. The number of terms involved increases like the square of the number of Feynman diagrams involved. In contrast, spinor techniques are methods for calculating Feynman diagrams numerically at the matrix element level. The number of terms involved is linear with the number of Feynman diagrams. Photon and fermion polarizations have to be summed explicitly. Spinor techniques are simplest when the fermions involved can be taken to be massless. They are most useful when a large number of Feynman diagrams is involved.

We have used HIP extensively, typically calculating processes with relatively simple topology ($2 \rightarrow 2$ and $2 \rightarrow 3$ tree-level cross-sections) but with complex Feynman rules.

The paper proceeds as follows. In the next section we give a brief overview of HIP. Some of the major objects and functions are mentioned. In sect. 3 we describe in more detail some of HIP's more important functions, presenting the mathematical relations they use and short examples of their application. In sect. 4 we give two complete examples: the calculation of the width for the muon decay $\mu^- \rightarrow \nu_\mu e^- \overline{\nu}_e$ with a finite electron mass, and a calculation of the matrix element for $e^+_R e^-_R \rightarrow Z \rightarrow t\bar{t} \rightarrow W^+W^-b\bar{b}$, including the W coupling to light fermions, which preserves all spin and angular correlations. In sect. 5 we summarize and give an outlook. The complete listing of HIP functions, with their usage messages (available as on-line documentation) is given in the appendix.

2. Overview

The packages in HIP contain functions that can manipulate various mathematical objects occurring in high-energy physics such as four-vectors, spinors and gamma matrices. Rather than follow one strict path from input to output, the packages allow the user to specify how a calculation proceeds (either interactively or in batch mode). A typical calculation might be to construct a matrix element, square it and sum over polarizations, construct the phase-space integral and evaluate this integral to give a symbolic expression for the total cross-section.

The most fundamental component of any high energy calculation is the manipulation of four-vectors. Basic objects such as the dot-product $(p \cdot q)$ (`DotProduct[p, q]`), the metric $g^{\mu\nu}$ (`G[mu, nu]`) and the completely anti-symmetric tensor $\epsilon^{\mu\nu\sigma\tau}$ (`Eps[mu, nu, sig, tau]`) are defined, with some of their elementary properties (*e.g.* the dot-product is symmetric in its two arguments). Four-vectors can be specified in terms of their components. They can then be boosted (using the function `Boost`), represented as sum of other four-vectors (`Decay`), etc. In addition, four-vectors can also be treated without reference to the explicit components. Dot products can be given explicit values (`SetDotProduct`, `SetMass`), Mandelstam variables defined (`SetMandelstam`), Lorentz indices defined (`PrepareIndex`) and contracted (`Contract`).

The second component in HIP is the Dirac algebra. The basic objects involved are the Dirac gamma matrices γ^μ (`DiracGamma[mu]`), γ^5 (`DiracGamma5`), the projection operators $P_\lambda = (1+\lambda\gamma^5)/2$ (`HelicityProjection[lambda]`) and $\not{p} = p_\mu\gamma^\mu$ (`Slash[p]`). The Dirac matrix product is represented by the *Mathematica* built-in function `NonCommutativeMultiply` (aliased to `**`). The trace of a product of Dirac gamma matrices is computed using the operator `GammaTrace`.

Some programs handling Dirac algebra, notably Reduce, can only deal with gamma matrices. HIP can also work with spinors. The basic spinor objects $u(p)$ and $v(p)$ (`SpinorU[p]` and `SpinorV[p]`) and their conjugates $\overline{u}(p)$ and $\overline{v}(p)$ (`SpinorUbar[p]` and `SpinorVbar[p]`) are defined. The function `AbsSquared` is used to square matrix element expressions which may include these spinors.

Expressions involving spinors do not have to be squared before they are calculated numerically. The HIP function `ConvertToST` converts a suitable expression involving spinors to an expression involving the elementary spinor products $s(p,k) = \overline{u}_R(p)u_L(k)$ and $t(p,k) = \overline{u}_L(p)u_R(k)$ (`SpinorS[p, k]` and `SpinorT[p, k]`) defined in reference 8. The expression produced can then be evaluated numerically by giving explicit values to the components of the four-vectors. Alternatively, it can be squared and converted back to an expression involving traces using the function `STToTraces`.

Given an expression for the matrix element squared associated with a process, the calculation of physical observables such as cross sections and decay widths involves integration over the phase-space of the out-going particles. The functions `CrossSection` and `DecayWidth` set up the phase-space integral. The functions return a `PhaseSpaceIntegral` object that can then be evaluated either symbolically using `EvaluatePhaseSpaceIntegral` or numerically using `NEvaluatePhaseSpaceIntegral`. Alternatively, one can write a *Mathematica* program to convert such an object to a C or Fortran program for numeric evaluation. Such a conversion program would be highly specific, depending on the particular programming language, integration routine etc. We have used one such program in our work, but it is not included with HIP.

HIP includes some of the common Feynman rules of the Standard Model which are implemented using the functions `Vertex` and `Propagator`. Constants such as $\sin^2\theta_W$ (`Sin[ThetaW]^2`) and particle masses (*e.g.* `Mass[ZBoson]`) are usually kept as symbolic constants. However, HIP stores a table of their numerical values; these are substituted for the symbolic expression by the *Mathematica* function `N`.

3. HIP functions

Strictly speaking, *Mathematica* does not distinguish between data-structures, functions and procedures. In practice, however, the *Mathematica* objects defined in HIP can be divided into several broad categories. In all cases we try to follow the *Mathematica* convention of beginning each name with a capital letter. Further, as far as is practical we use full, descriptive English names rather than cryptic acronyms. Using *Mathematica* utilities, the user can choose his or her own cryptic abbreviations. The major categories are:

1. Objects such as gamma matrices or dot products. These are characterized by the property that they usually remain unevaluated.

2. Declarations and definitions. These do not typically return anything, but are rather invoked as part of the initialization process.

3. Operations such as contracting indices or taking traces. These typically take their input and convert it to an equivalent expression.

In this section we describe the most important members of each class.

Table 1 lists the major functions representing objects with their equivalent in ordinary physical notation.

The most useful declarative functions are:

— `PrepareIndex`: `PrepareIndex[mu, nu]` declares μ and ν and Lorentz indices.

— `SetMass`: `SetMass[p1, p2,..., m]` sets `p1`, `p2`,.. to be four-vectors with invariant mass `m`.[9]

— `SetMandelstam`: `SetMandelstam[{p1, p2, p3, p4}, {m1, m2, m3, m4}, s, t, u]` sets `p1`, `p2`, `p3` and `p4` to be on-shell with masses `m1`, `m2`, `m3` and `m4` respectively and sets the DotProducts of `p1`, `p2`, `p3` and `p4` in terms of the Mandelstam variables s, t and u and the masses:

$$
\begin{aligned}
(p_1 \cdot p_2) &\to \frac{1}{2}(s - m_1^2 - m_2^2) & (p_3 \cdot p_4) &\to \frac{1}{2}(s - m_3^2 - m_4^2) \\
(p_1 \cdot p_3) &\to \frac{1}{2}(-t + m_1^2 + m_3^2) & (p_2 \cdot p_4) &\to \frac{1}{2}(-t + m_2^2 + m_4^2) \\
(p_1 \cdot p_4) &\to \frac{1}{2}(-u + m_1^2 + m_4^2) & (p_2 \cdot p_3) &\to \frac{1}{2}(-u + m_2^2 + m_3^2),
\end{aligned}
$$

where m_i is the mass of the particle p_i.

Most of HIP's functionality is implemented as operator-type functions. The main ones are:

— `Boost`: `Boost[fv, rap, dir]` gives a four-vector obtained by boosting the four-vector `fv` by rapidity `rap` in the direction specified by `dir`. Example:

```
In[1]:= Boost[{0, 0, 0, m}, r, {cth, 0}]
```

Boost the four-vector (0, 0, 0, m) by rapidity r in the direction $\cos\theta = \mathrm{cth}$, $\phi = 0$.

```
                         2
Out[1]= {Sqrt[1 - cth ] m Sinh[r], 0, cth m Sinh[r], m Cosh[r]}
```

$$(m\sqrt{1-\cos^2\theta}\sinh r,\ 0,\ m\cos\theta\sinh r,\ m\cosh r).$$

— `Decay`: `Decay[v, dir, {m1, m2}]` gives two four-vectors v_1 and v_2 such that $v_1 + v_2 = v$, $v_1^2 = m_1^2$, $v_2^2 = m_2^2$ and the direction of v_1 in the v center-of-mass frame is given by `dir`. Example:

```
In[1]:= Decay[{0, 0, 0, m}, {cth, 0}, {m1, 0}]
```

Decompose the four-vector $p = (0, 0, 0, m)$ into two four-vectors p_1 and p_2 such that $p_1^2 = m_1^2$, $p_2^2 = 0$ and the direction of p_1 in the p center-of-mass frame is given by $\cos\theta = \text{cth}$ and $\phi = 0$. After some rearrangement one gets:

```
                             2     2       2            2       2     2       2
           Sqrt[1 - cth ] (m  - m1 )     cth (m  - m1 )   m  + m1
Out[1]= {{-------------------------, 0, --------------, --------},
                     2 m                     2 m            2 m

                  2     2       2              2       2     2       2
     -(Sqrt[1 - cth ] (m  - m1 ))     -(cth (m  - m1 ))   m  - m1
>    {---------------------------, 0, -----------------, --------}}
                2 m                          2 m            2 m
```

$$p_1 = \left(\frac{\sqrt{1-\cos^2\theta}(m^2 - m_1^2)}{2m}, 0, \frac{\cos\theta(m^2 - m_1^2)}{2m}, \frac{m^2 + m_1^2}{2m} \right),$$
$$p_2 = \left(\frac{-\sqrt{1-\cos^2\theta}(m^2 - m_1^2)}{2m}, 0, \frac{-\cos\theta(m^2 - m_1^2)}{2m}, \frac{m^2 - m_1^2}{2m} \right).$$

— `Contract`: `Contract[expr, index]` contracts `index` in `expr`. `Contract` with respect to a particular index μ invokes a large set of rules. The basic rules for handling arbitrary tensors and vector are:

$$\begin{aligned} g_\mu^\mu &\to D \\ p_\mu q^\mu &\to p \cdot q \\ g_\mu^\tau T^{\nu_1 \cdots \mu \cdots \nu_n} &\to T^{\nu_1 \ldots \tau \ldots \nu_n} \end{aligned} \tag{3.1}$$

where D is the dimension of space-time.[10] For handling the completely anti-symmetric ϵ symbol we use

$$\epsilon_{\mu\nu_1\cdots\nu_n}\epsilon^{\mu\tau_1\cdots\tau_n} \to (D-n) \begin{vmatrix} g_{\nu_1}^{\tau_1} & \cdots & g_{\nu_1}^{\tau_n} \\ \vdots & \ddots & \vdots \\ g_{\nu_n}^{\tau_1} & \cdots & g_{\nu_n}^{\tau_n} \end{vmatrix}. \tag{3.2}$$

The rules associated with γ-matrices are:[11]

$$
\begin{aligned}
p_\mu \gamma^\mu &\to \not p \\
\gamma_\mu \gamma^\mu &\to D \\
\gamma_\mu \gamma_\nu \gamma^\mu &\to (2-D)\gamma_\nu \\
\gamma_\mu \gamma_{\nu_1} \gamma_{\nu_2} \gamma^\mu &\to (D-4)\gamma_{\nu_1}\gamma_{\nu_2} + 4g_{\nu_1\nu_2} \\
\gamma_\mu \gamma_{\nu_1} \gamma_{\nu_2} \gamma_{\nu_3} \gamma^\mu &\to -2\gamma_{\nu_3}\gamma_{\nu_2}\gamma_{\nu_1} - (D-4)\gamma_{\nu_1}\gamma_{\nu_2}\gamma_{\nu_3}.
\end{aligned}
\tag{3.3}
$$

For more complicated cases we use

$$
\gamma_\mu \Gamma^{(n)} \gamma^\mu \to \begin{cases}
2\Gamma^{(n)} + 2\Gamma^{(n)}_R & (n \text{ even}) \quad (D=4) \\
-2\Gamma^{(n)}_R & (n \text{ odd}) \\
(-1)^n \left((D-4)\Gamma^{(n)} + 2\Gamma^{(n)}_R - 2\sum_{i=4}^{n-3} (-1)^i \gamma_{\nu_i} \Gamma^{(n)}_i \right) & (D \neq 4),
\end{cases}
\tag{3.4}
$$

where

$$
\begin{aligned}
\Gamma^{(n)} &= \gamma_{\nu_1} \cdots \gamma_{\nu_n} \\
\Gamma^{(n)}_R &= \gamma_{\nu_n} \cdots \gamma_{\nu_1} \\
\Gamma^{(n)}_i &= \gamma_{\nu_1} \cdots \gamma_{\nu_{i-1}} \gamma_{\nu_{i+1}} \cdots \gamma_{\nu_n}.
\end{aligned}
\tag{3.5}
$$

Example:

```
In[1]:= Contract[G[mu, nu] p[mu], mu]
```

Contract the index μ in $g_{\mu\nu}p^\mu$.

```
Out[1]= p[nu]
```

$$p_\nu$$

```
In[2]:= Contract[DiracGamma[mu]**Slash[p]**DiracGamma[nu]**
        DiracGamma[mu], mu]
```

$$\gamma_\mu \not{p} \gamma_\nu \gamma^\mu$$

```
Out[2]= (-4 + SpaceTimeDimension) Slash[p] ** DiracGamma[nu] +
>     4 p[nu]
```

$(D-4) \not{p}\gamma_\nu + 4p_\nu$ where D is the space-time dimension.

— `GammaTrace`: `GammaTrace[expr]` is the trace (in spinor space) of `expr`. $\mathrm{tr}\{1\}$ can be left unevaluated as the constant `DiracGammaSize`, but is usually set to 4. Whenever possible, `GammaTrace` uses the following simple rules:

$$\begin{aligned}
\mathrm{tr}\{\gamma_{\mu_1}\ldots\gamma_{\mu_{2n+1}}\} &\to 0 & \mathrm{tr}\left\{\gamma^5\right\} &\to 0 \\
\mathrm{tr}\left\{\gamma^5\gamma_\mu\gamma_\nu\right\} &\to 0 & \mathrm{tr}\left\{\gamma^5\gamma_{\mu_1}\ldots\gamma_{\mu_{2n+1}}\right\} &\to 0 \\
\mathrm{tr}\{\gamma_\mu\gamma_\nu\} &\to \mathrm{tr}\{1\}\, g_{\mu\nu} & \mathrm{tr}\left\{\gamma^5\gamma_\mu\gamma_\nu\gamma_\tau\gamma_\sigma\right\} &\to \mathrm{tr}\{1\}\; i\epsilon_{\mu\nu\tau\sigma}. \\
\mathrm{tr}\{\gamma_\mu\gamma_\nu\gamma_\tau\gamma_\sigma\} &\to \mathrm{tr}\{1\}\,(g_{\mu\nu}g_{\tau\sigma} + g_{\mu\sigma}g_{\nu\tau} - g_{\mu\tau}g_{\nu\sigma})
\end{aligned} \tag{3.6}$$

Traces of longer expressions invoke the following recursive rules:

$$\begin{aligned}
\mathrm{tr}\left\{\gamma_\mu\Gamma^{(n)}\right\} &\to \sum_{i=1}^{n}(-1)^{(i+1)} g_{\mu\nu_i}\mathrm{tr}\left\{\Gamma_i^{(n)}\right\} \\
\mathrm{tr}\left\{\gamma^5\Gamma^{(n)}\right\} &\to \sum_{1\le i<j<k<l\le n}(-1)^{(i+j+k+l)} i\epsilon_{\nu_i\nu_j\nu_k\nu_l}\mathrm{tr}\left\{\Gamma_{ijkl}^{(n)}\right\}.
\end{aligned} \tag{3.7}$$

Example:

```
In[1]:= GammaTrace[DiracGamma[mu]**DiracGamma[nu]
```

$$\mathrm{tr}\{\gamma_\mu\gamma_\nu\}$$

```
Out[1]= 4 G[mu, nu]
```

$$4g_{\mu\nu}$$

```
In[2]:= GammaTrace[DiracGamma5**DiracGamma[mu]**DiracGamma[nu]**
        DiracGamma[sig]**DiracGamma[tau]]
```

$$\mathrm{tr}\left\{\gamma^5\gamma_\mu\gamma_\nu\gamma_\sigma\gamma_\tau\right\}$$

```
Out[2]= 4 I Eps[mu, nu, sig, tau]
```

$$4i\epsilon_{\mu\nu\sigma\tau}$$

```
In[3]:= GammaTrace[DiracGamma[mu]**Slash[p1]**DiracGamma[nu]**
        Slash[p2]**DiracGamma[mu]**Slash[p3]**DiracGamma[nu]**
        Slash[p4]]
```

$$\mathrm{tr}\left\{\gamma_\mu \not{p}_1\gamma_\nu \not{p}_2\gamma^\mu \not{p}_3\gamma^\nu \not{p}_4\right\}$$

```
Out[3]= -32 DotProduct[p1, p3] DotProduct[p2, p4]
```

$$-32(p_1\cdot p_3)(p_2\cdot p_4)$$

— `AbsSquared`: `AbsSquared[expr]` is the absolute value of `expr` squared. `AbsSquared` sums over polarization of both external spinors and vectors unless their polarizations are explicitly specified:

$$\begin{aligned}
|Xu(p)|^2 &\to \sum_\lambda X u_\lambda(p)\overline{u}_\lambda(p)X^* \to X(\not{p}+m)X^* \\
|Xv(p)|^2 &\to \sum_\lambda X v_\lambda(p)\overline{v}_\lambda(p)X^* \to X(\not{p}-m)X^* \\
|\epsilon_\mu(p)|^2 &\to \begin{cases} -g_{\mu\mu'} & (m=0) \\ -g_{\mu\mu'}+\dfrac{p_\mu p_{\mu'}}{m^2} & (m\neq 0), \end{cases}
\end{aligned}\tag{3.8}$$

where m is the mass associated with p. Example:

```
In[1]:= AbsSquared[SpinorUbar[p]**SpinorV[q]]
```

$$|\overline{u}(p)v(q)|^2$$

```
Out[1]= 4 DotProduct[p,q] - 4 Mass[p] Mass[q]
```

$$4p \cdot q - 4m_p m_q$$

```
In[2]:= AbsSquared[SpinorUbar[p, Right]**SpinorV[q, Left]]
```

$$|\overline{u}_R(p) v_L(q)|^2$$

```
Out[2]= 2 DotProduct[p,q]
```

$$2p \cdot q$$

```
In[3]:= AbsSquared[SpinorUbar[p]**DiracGamma[mu]**SpinorU[q]]
```

$$|\overline{u}(p)\gamma_\mu u(q)|^2$$

```
Out[3]= -4 DotProduct[p, q] G[mu, Conjugate[mu]] +
>    4 G[mu, Conjugate[mu]] Mass[p] Mass[q] +
>    4 p[Conjugate[mu]] q[mu] + 4 p[mu] q[Conjugate[mu]]
```

$$-4(p \cdot q) g_{\mu\mu'} + 4 g_{\mu\mu'} m_p m_q + 4 p_{\mu'} q_\mu + 4 p_\mu q_{\mu'}$$

`Conjugate[mu]` (μ' is the new index needed on squaring the expression).

— `ConvertToST`: `ConvertToST[expr]` attempts to convert `expr` to SpinorS and SpinorT objects. Unless instructed otherwise, the program assumes every massive vector `q` is the sum of two massless vectors `LightlikeVectorDecayedFrom[q, 1]` and `LightlikeVectorDecayedFrom[q, 2]`. Example:

```
In[1]:= ConvertToST[
        SpinorUbar[p, Right]**Slash[q]**SpinorU[k, Right]]
```

Convert $\overline{u}_R(p)\, \not{q} u_R(k)$ to spinor techniques. p and k have previously been set massless.

```
Out[1]= -(SpinorS[p, LightlikeVectorDecayedFrom[q, 1]] *
>        SpinorT[k, LightlikeVectorDecayedFrom[q, 1]]) -
>     SpinorS[p, LightlikeVectorDecayedFrom[q, 2]] *
>      SpinorT[k, LightlikeVectorDecayedFrom[q, 2]]
```

The result is $s(p, q_1)t(k, q_1) - s(p, q_2)t(k, q_2)$ where $q_1^2 = q_2^2 = 0$ and $q_1 + q_2 = q$ are assumed.

— `Vertex`: `Vertex[line1, line2, ...]` is the Feynman rule for a vertex. Each line is of the form `particle` or `{particle, p}` or `{particle, p, indices}` where `particle` is the particle type, `p` is the particle's momentum, and `indexes` are the appropriate Lorentz or gauge group indexes. Example:

```
In[1]:= Vertex[Electron, Electron, {ZBoson, p, mu}]
```

The eeZ vertex for Z boson carrying momentum p_μ

```
Out[1]= (-2 I Sqrt[Alpha] Sqrt[Pi] DiracGamma[mu] **
          -HelicityProjection[Left]              2
>        (------------------------- + Sin[ThetaW] )) /
                      2
>     (Cos[ThetaW] Sin[ThetaW])
```

$$\frac{-2i\sqrt{\alpha}\sqrt{\pi}}{\cos\theta_W \sin\theta_W}\gamma_\mu \left(-\frac{1}{2}P_L + \sin^2\theta_W\right)$$

where α is the electromagnetic fine structure constant and θ_W is the Weinberg angle.

— `CrossSection` and `DecayWidth`: `CrossSection[me2, q1->{q1x, q1y, q1z, e1}, q2->{q2x, q2y, q2z, e2}, outGoing]` returns an expression for the phase-space integral to be evaluated by `EvaluatePhaseSpaceIntegral`. `me2` is the expression for the matrix element squared, `{p1x, p1y, p1z, e1}` and `{p2x, p2y, p2z, e2}` are the explicit four-vectors of the incoming particles, and `outGoing` specifies the order of phase-space evaluation as explained below. `DecayWidth[me2, p -> {px, py, pz, e}, outGoing]` similarly returns an expression for the phase-space integral resulting in the decay width given matrix element squared `me2` and initial momentum `{px, py, pz, e}`.

The formula used for cross-section calculations is

$$\sigma = \frac{(2\pi)^4}{2s}\frac{2|q_1|}{\sqrt{s}} \int |\mathcal{M}|^2 d\Phi_n(q_1+q_2; p_1, \ldots, p_n), \tag{3.9}$$

and for the decay width it is

$$\Gamma = \frac{(2\pi)^4}{2M} \int |\mathcal{M}|^2 \, d\Phi_n(P; p_1, \ldots p_n). \tag{3.10}$$

Phase-space integration is performed by a recursive use of the relations

$$\begin{aligned} d\Phi_n(P;\, p_1, \ldots, p_n) &= d\Phi_{n-1}\,(P;\, p_{12}, p_3, \ldots, p_n)\, d\Phi_2(p_{12}; p_1, p_2)\,(2\pi)^3 dm_{12}^2 \\ &= d\Phi_{n-1}(P;\, p_{12}, p_3, \ldots, p_n)\,\frac{1}{8(2\pi)^3}\frac{2|p_1|}{m_{12}}\, d\Omega_{12}\, dm_{12}^2, \end{aligned} \tag{3.11}$$

where $p_{12}^2 = m_{12}^2$ and Ω_{12} represents the direction of the 'decay' of the vector p_{12}

to p_1 and p_2 in its center-of-mass frame. The factor $2|p_1|/m_{12}$ is given by

$$\frac{2|p_1|}{m_{12}} = \begin{cases} \frac{\left[\left(m_{12}^2-(m_1+m_2)^2\right)\left(m_{12}^2-(m_1-m_2)^2\right)\right]^{1/2}}{m_{12}^2} & \\ \sqrt{1-\frac{4m_1}{m_{12}}} & (m_1=m_2) \\ 1-\frac{m_1^2}{m_{12}^2} & (m_2=0) \\ 1 & (m_1=m_2=0). \end{cases} \tag{3.12}$$

The argument `outGoing` tells HIP how to build the phase-space element $d\Phi_n$. It specifies both the order that the momenta $p_1 \cdots p_n$ are paired (eq. (3.11)) and, optionally, the symmetry of the individual two-body phase-space elements. By default, the complete angular integral over Ω_{12} is constructed. Often, due to the symmetry of the process, one can reduce the dimension of this integral (in the case of cylindrical symmetry), or eliminate it completely (in the case of spherical symmetry). This is done using the keywords `Cylindrical` and `Spherical`.

For example, let us consider the decay process $\mu^-(p) \to e^-(p_1)\overline{\nu}_e(p_2)\nu_\mu(p_3)$. If the μ is unpolarized, the decay process is spherically symmetric. The direction of ν_μ may be chosen arbitrarily. Once that is done, the direction of the electron with respect to the $(e^-\overline{\nu}_e)$ system has cylindrical symmetry about the ν_μ direction. The `outGoing` argument is given by `Spherical[Cylindrical[p1, p2], p3]`. If the μ is polarized, the spherical symmetry of the decay is reduced to a cylindrical symmetry about the polarization axis. `outGoing` is then given by `Cylindrical[{p1, p2}, p3]`.

4. Examples

In this section we give two examples showing step by step how a HIP calculation is carried out. In the first example we compute the decay width of a muon in the process $\mu^- \to e^- \nu_\mu \overline{\nu}_e$ and with a non-zero electron mass. In the second example we express the matrix element for the process $e_R^+ e_R^- \to Z \to t\overline{t} \to W^+ W^- b\overline{b}$ as a spinor-technique expression.

4.1 $\Gamma(\mu^- \to e^- \nu_\mu \overline{\nu}_e)$

We use a low-energy approximation in which the W-propagator is a constant and is absorbed, along with the coupling constant g into the Fermi constant G_F. figure/1 shows the single Feynman diagram for the process.

```
In[1]:= PrepareIndex[sig]
```

Instruct *Mathematica* to treat `sig` as an index.

```
In[2]:= SetMass[{pnub, 0}, {pe, me}, {pnu, 0}, {pmu, mmu}]
```

Set the masses of the four external particles. The neutrinos (ν_μ and $\overline{\nu}_e$ carrying momenta `pnu` and `pnub` respectively) have zero mass. The electron (`pe`) is set to have mass `me` while the muon (`pmu`) is set to have mass `mmu`. Later on, these masses can be given numerical values.

```
In[3]:= matrixelement = 2 Sqrt[2] FermiGF *
        SpinorUbar[pnu] ** DiracGamma[sig] ** SpinorU[pmu, Left] *
        SpinorUbar[pe] ** DiracGamma[sig] ** SpinorV[pnub, Left];
```

$$\mathcal{M} = 2\sqrt{2} G_F \overline{u}(p_\nu)\gamma_\sigma u_L(p_\mu)\overline{u}(p_e)\gamma^\sigma u_L(p_{\overline{\nu}})$$

```
In[4]:= me2 = AbsSquared[matrixelement]/2;
```

Square the matrix-element using the *Mathematica* function AbsSquared. We suppress the printing of the long intermediate result.

```
In[5]:= me2 = Contract[me2, {sig, Conjugate[sig]}] // Factor
```

Contract over the indices σ and σ'. `// Factor` instructs *Mathematica* to the factor the expression.

```
                      2
Out[5]= 128 FermiGF  DotProduct[pe, pnu] DotProduct[pmu, pnub]
```

$$128 G_F^2 (p_e \cdot p_\nu)(p_\mu \cdot p_{\overline{\nu}})$$

```
In[6]:= width = DecayWidth[me2, pmu -> {0, 0, 0, mmu},
        Spherical[Cylindrical[pe, pnub], pnu]]
```

Ask *Mathematica* to construct the phase-space integral to compute the decay-width. The expression `Spherical[Cylindrical[pe, pnub], pnu]` indicates a cylindrical symmetry in the phase-space integral over the pair $(e^-, \overline{\nu}_e)$ and a spherical symmetry over pair $(\nu_\mu, (e^-, \overline{\nu}_e))$ (here $(e^-, \overline{\nu}_e)$ is the combined system of e^- and $\overline{\nu}_e$.)

```
Out[7]= -PhaseSpaceIntegral-
In[8]:= width = EvaluatePhaseSpaceIntegral[width];
```

Evaluate the phase-space integral symbolically. Again we suppress the long intermediate result.

```
In[9]:= Factor[width /. me->x mmu /. Log[a_ b_] :> Log[a]+Log[b]]
```

Use some *Mathematica* rules to tidy up the expression. We express the mass of the electron in terms of the mass of the muon $m_e = x m_\mu$ and combine logarithms using the rule $\log(ab) \to \log a + \log b$.

```
               2     5         2       6     8        4      2
   -(FermiGF  mmu  (-1 + 8 x  - 8 x  + x  + 12 x  Log[x ]))
Out[10]= --------------------------------------------------------
                                   3
                             192 Pi
```

$$\Gamma(\mu^- \to e^- \nu_\mu \overline{\nu}_e) = \frac{G_F^2 m_\mu^5 (1 - 8x^2 + 8x^6 - x^8 - 12x^4 \log(x^2))}{192\pi^3}$$

4.2 Matrix element for $e_R^+ e_R^- \to Z \to t\overline{t} \to W^+W^-b\overline{b}$ in spinor technique form

We convert the matrix element for the process $e_R^+ e_R^- \to Z \to t\overline{t} \to W^+W^-b\overline{b}$ into the spinor technique form[8]. The advantage of this form is that it can easily be used in a numerical calculation. The spinor technique objects $s(p,q)$ and $t(p,q)$ are defined in table 1. s and t have compact expressions which are easily evaluated numerically.

The relevant diagram is shown in figure/2. The *Mathematica* computation follows. We have suppressed most of the intermediate output.

```
In[1]:= PrepareIndex[tau, mu, nu]
```

Instruct *Mathematica* to treat `tau`, `mu` and `nu` as indices.

```
In[2]:= SetMass[{p1, p2, pb, pbb, 0}, {pwp, pwm, Mass[WBoson]}]
```

Set the masses of the external particles. The e^- and e^+ have momenta `p1` and `p2` respectively, and are massless. The b and $\overline{b}$ have momenta `pb` and `pbb` respectively, and are also massless. The W^+ and W^- have momenta `pwp` and `pwm` respectively, and mass `Mass[WBoson]`.

```
In[3]:= SetDotProduct[{p1, p2, s/2}]
```

We set the value of $(p_1 \cdot p_2)$ to $s/2$.

```
In[4]:= matrixElement =
        HeavyVectorPolarization[pwp, nu] *
        HeavyVectorPolarization[pwm, mu] *
        SpinorVbar[p2, Right] **
        Vertex[{Electron}, {Electron}, {ZBoson, p1+p2, tau}] **
        SpinorU[p1, Right] * Propagator[{ZBoson, p1+p2}] *
        SpinorUbar[pb, Left] **
        Vertex[{BottomQuark}, {TopQuark}, {WBoson, pwp, nu}] **
        FermionPropagator[TopQuark, pwp+pb] **
        Vertex[{TopQuark}, {TopQuark}, {ZBoson, p1+p2, tau}] **
        FermionPropagator[TopQuark, -pwm-pbb] **
        Vertex[{BottomQuark}, {TopQuark}, {WBoson, pwm, mu}] **
        SpinorV[pbb, Left]
```

$$\varepsilon^{\nu}(p_{W^+})\varepsilon^{\mu}(p_{W^-})\left\{\overline{v}_R(p_2)\mathcal{V}^{\tau}(e,e,Z)u_R(p_1)\right\}\Delta(Z,p_1+p_2)$$
$$\times\left\{\overline{u}_L(p_b)\mathcal{V}_{\nu}(b,t,W)\Delta(t,p_b+p_+)\mathcal{V}_{\tau}(t,t,Z)\Delta(t,-p_{\overline{b}}-p_-)\mathcal{V}_{\mu}(b,t,W)v_L(p_{\overline{b}})\right\},$$

where $\Delta(X,p)$ is the propagator for particle X with momentum p and $\mathcal{V}_{\sigma}(A,B,C)$ is the vertex of the particles A, B and C with Lorentz index σ.

```
In[5]:= lightlikeVectorRules =
        {LightlikeVectorDecayedFrom[pwp, 1] -> r1,
        LightlikeVectorDecayedFrom[pwp, 2] -> r2,
        LightlikeVectorDecayedFrom[pwm, 1] -> r3,
        LightlikeVectorDecayedFrom[pwm, 2] -> r4};
```

In the computation the massive four-vector `pwp` is replaced by the sum of two massless four-vectors `r1` and `r2`. `pwm` is similarly replaced by `r3+r4`. In the spinor technique, r_1 and r_2 are the momenta of the massless fermion and antifermion into which the W decays. The final expression, thus, provides us with the angular distribution of the final state fermions.

```
In[6]:= convertedMatrixElement =
        ConvertToST[matrixElement] /. lightlikeVectorRules;
```

Convert the matrix element to spinor technique form and apply the simplifying rules.

After some rearrangement, we get the following expression:

```
                         2                                          2
Out[7]= (-4 I Alpha  Pi KobayashiMaskawa[BottomQuark, TopQuark]
>       SpinorS[pbb, r4] SpinorT[pb, r1]
                             2
>       (4 Mass[TopQuark]  SpinorS[p2, r2] SpinorT[p1, r3] +
                     3
>         (4 - ------------)
                          2
               Sin[ThetaW]
>          (SpinorS[pb, r2] SpinorT[p1, pb] +
>            SpinorS[r1, r2] SpinorT[p1, r1])
>          (SpinorS[p2, pbb] SpinorT[pbb, r3] -
>            SpinorS[p2, r4] SpinorT[r3, r4]))) /
                   2             2
>     (Cos[ThetaW]  Mass[WBoson]
                                                    2                2
>       (2 DotProduct[pb, pwp] - Mass[TopQuark]  + Mass[WBoson] )
                                                     2                2
>       (2 DotProduct[pbb, pwm] - Mass[TopQuark]  + Mass[WBoson] )
                            2
>       (s - Mass[ZBoson] ))
```

$$
\begin{aligned}
-4i\alpha^2\pi V_{tb}^2 s(p_{\overline{b}}, r_4)t(p_b, r_1)\Big[4m_t^2 s(p_2, r_2)t(p_1, r_3)\\
+(4-3/\sin^2\theta_W)\big(s(p_b, r_2)t(p_1, p_b)+s(r_1, r_2)t(p_1, r_1)\big)\\
\times\big(s(p_2, p_{\overline{b}})t(p_{\overline{b}}, r_3)-s(p_2, r_4)t(r_3, r_4)\big)\Big]\Big/\\
\Big[\cos^2\theta_W m_W^2\big(2(p_b\cdot p_+)-m_t^2+m_W^2\big)\big(2(p_{\overline{b}}\cdot p_-)-m_t^2+m_W^2\big)(s-m_Z^2)\Big]
\end{aligned}
$$

5. Conclusion and Outlook

We developed HIP as an aid in the calculation tree-level processes in high energy physics which would otherwise be much more difficult. HIP's main feature is in providing an environment within *Mathematica* in which one can refer to objects and perform operations that occur frequently in this field. One can use HIP interactively to assist with small calculations, or set it up to automatically perform massive 'symbol crunching'.

We have checked HIP against hand calculations of $e^+e^- \to W^+W^-$, $e\gamma \to W\nu$, both with arbitrary (C and P conserving) $W\gamma$ couplings, and of numerous simple electroweak processes. We also checked them against published results for $e^+e^- \to W^+W^-\gamma$, $e^+e^- \to W^+W^-Z$, $e^+e^- \to \gamma\gamma\gamma$ and $e^+e^- \to ZZZ$.

In the future, we hope to extend HIP's capabilities into performing loop integrals, calculating color factors and incorporating other techniques for symbolically calculating Feynman diagrams at the matrix element level. We also hope to translate HIP to other symbolic languages such as *Maple*, so as to maximize the group of its potential users.

HIP is available for distribution. The distribution includes the various component *Mathematica* packages, the online documentation as listed in the appendix, and several files containing sample calculations done by HIP.

ACKNOWLEDGEMENTS

We are grateful to M. Peskin for his encouragement and for critically reading the manuscript. E. Y. would like to thank Wolfram Research Inc. for their support, both financial and technical. We would also like to thank all the people who have used our packages and given us both encouragement and bug reports. Work supported by the Department of Energy, contract DE-AC03-76SF00515.

APPENDIX

In this appendix we include the complete listing of the objects defined in HIP, together with their usage messages. These usage messages are also available as on-line documentation. The functions are listed in alphabetical order.

```
AbsSquared::usage = "AbsSquared[expr] is the absolute value squared
of expr."

Boost::usage = "Boost[fv, rap, dir] gives a four-vector obtained by
boosting fv by rapidity rap in the direction specified by dir.  If
dir is of the form {cth, phi}, cth is the Cosine of the angle
between the direction and the z axis and phi is the angle between
the direction's projection on the x-y plane and the x axis.  If dir
is of the form {x, y, z}, the direction is taken to be the
direction of the vector {x, y, z}. Boost[fv, rap] boosts fv by
amount rap in the z direction."

BoostAmount::usage = "BoostAmount[fv] gives a vector {rap, {x, y,
z}} such that Boost[{0, 0, 0, Mass[fv]}, rap, {x, y, z}] is fv."

CommutativeAllQ::usage = "CommutativeAllQ[expr] is True if expr
does not have any non-commuting sub-expressions, and False
otherwise."

CommutativeQ::usage = "CommutativeQ[x] is True if x is commutative
(the default), and False if x is non-commutative."

Commutator::usage = "Commutator[a, b] is an equivalent expression
to a**b, with a and b interchanged (e.g.  Commutator[a, b] = b**a +
[a, b])."

Commute::usage = "Commute[expr, {x, y}] commutes y from the right
of x to the left of x everywhere in expr."

Contract::usage = "Contract[expr, index] contracts index in expr.
All subexpressions of form G[index, _] or p_[index, _] can
potentially be effected. Contract[expr, {index1, index2, ...}] does
the contraction sequentially."
```

```
ConvertToChiralFermions::usage = "ConvertToChiralFermions[expr]
attempts to convert all occurrences of DiracGamma5 and
HelicityProjection in expr into products of chiral spinors."

ConvertToGamma5::usage = "ConvertToGamma5[expr] attempts to convert
all occurrences of HelicityProjection and spinors of specific
chirality into products involving DiracGamma5."

ConvertToMassless::usage = "ConvertToMassless is an option for
ConvertToST."

ConvertToST::usage = "ConvertToST[expr] attempts to convert
DiracGamma5 and HelicityProjection occurrences in expr to SpinorS
and SpinorT objects."

CrossSection::usage = "CrossSection[me2, p1 -> {p1x, p1y, p1z, e1},
p2 -> {p2x, p2y, p2z, e2}, outGoing] returns a PhaseSpaceIntegral.
me2 is the matrix element squared, {p1x, p1y, p1z, e1} and {p2x,
p2y, p2z, e2} are the explicit four-vectors of the incoming
particles, and outGoing specifies the order of phase-space
evaluation."

Cylindrical::usage = "Cylindrical[p1, p2] indicates a cylindrically
symmetric two body decay into p1 and p2 about the direction of
motion of the decaying particle."

Decay::usage = "Decay[fv, dir, {m1, m2}] gives two four-vectors
{fv1, fv2} such that fv1+fv2 is fv, Mass[fv1] is m1, Mass[fv2] is
m2 and the direction of fv1 in the fv center-of-mass frame is given
by dir.  If dir is of the form {cth, phi}, cth is the Cosine of the
angle between the direction and the z axis and phi is the angle
between the direction's projection on the x-y plane and the x axis.
If dir is of the form {x, y, z}, the direction is taken to be the
direction of the vector {x, y, z}.  Decay[fv, dir] is the same as
Decay[fv, dir, {0, 0}]."

DecayWidth::usage = "DecayWidth[me2, p -> {px, py, pz, e},
outGoing] returns a PhaseSpaceIntegral. me2 is the matrix element
squared, {px, py, pz, e} is explicit four-vector of the decaying
particle, and outGoing specifies the order of phase-space
evaluation."

DiracGamma5::usage = "DiracGamma5 is mainly meaningful for
SpaceTimeDimension of 4. Some of its features will work in any even
spacetime dimension."

DiracGamma::usage = "DiracGamma[mu] is the Dirac Gamma matrix.  It
is set to be non-commutative. Note that DotProduct[DiracGamma, p]
is simplified to Slash[p]."

DiracGammaSize::usage = "DiracGammaSize is the dimension of the
Dirac spinors. Its default value is 4."

DotProduct::usage = "DotProduct[p, q] is the Lorentz invariant dot
product.  It is evaluated explicitly (using the metric) if both p
and q are lists of length SpaceTimeDimension."

DotProductRules::usage = "DotProductRules[{p, q, u}] returns
```

```
{toRule, fromRule}.  toRule takes DotProduct[p, q] to u and
fromRule takes u to DotProduct[p, q].  DotProductRules[{p1, q1,
u1}, {p2, q2, u2},...]  returns {toRules, fromRules} where toRules
are the rules for converting DotProducts to u's and fromRules are
the rules for converting u's to DotProducts.  DotProductRules[{p,
q, u}, Mandelstam -> {s}] will define fromRules in terms of s."

Energy::usage = "Energy[fv] is the energy of fv."

Eps::usage = "Eps[a, b, c, ...] is the completely anti-symmetric
object.  Its arguments are automatically sorted alphabetically.
Eps[..., p, ...] is the same as Eps[..., mu, ...] p[mu]."

EvaluatePhaseSpaceIntegral::usage =
"EvaluatePhaseSpaceIntegral[PhaseSpaceIntegral[...]]  will
evaluate the Phase-space-integral."

ExpandSlash::usage = "ExpandSlash[expr] expands slash of sums to
sums of slashes in expr."

G::usage = "G[mu, nu] is the ordinary spacetime metric.  mu and nu
range from 1 to SpaceTimeDimension."

GammaTrace::usage = "GammaTrace[expr] is the trace (in spinor
space) of expression. Note that no division by DiracGammaSize is
carried out, ie GammaTrace[1] gives DiracGammaSize."

HeavyVectorPolarization::usage = "HeavyVectorPolarization[p, mu,
pol] represents the polarization vector of a massive vector
particle with momentum p, polarization pol and index mu."

HeavyVectorPolarization::usage = "HeavyVectorPolarization[p, mu,
pol] represents the polarization vector of a massive vector
particle with momentum p, polarization pol and index mu."

HelicityProjection::usage = "HelicityProjection[pol] is the
helicity projection operator (1 +/- Gamma5)/2."

KobayashiMaskawa::usage = "KobayashiMaskawa[p1, p2] is the
Kobayashi-Moskawa matrix element connecting particles p1 and p2."

LightlikeVectorDecayedFrom::usage = "LightlikeVectorDecayedFrom[p,
n] represents a lightlike vector used for the polarizations of
massive vector bosons.  It obeys p =
LightlikeVectorDecayedFrom[p,1] + LightlikeVectorDecayedFrom[p,2]
where p is the momentum of the external massive vector boson."

LightlikeVectorNotCollinearWith::usage =
"LightlikeVectorNotCollinearWith[p] represents a lightlike vector
that is not collinear with p."

LongitudinalPolarization::usage = "LongitudinalPolarization[p] is a
four-vector longitudinal polarization of the four-vector p."

Mandelstam::usage = "Mandelstam is an option for specifying a list
of Mandelstam-type variables."

MandelstamRules::usage = "MandelstamRules[{p1, p2, p3, p4}, {m1,
m2, m3, m4}, s, t, u] returns {toRules, fromRules}.  toRules are
```

```
rules for converting DotProducts of p1, p2, p3 and p4 to
expressions containing s, t and u and the masses.  fromRules are
the rules for converting s, t and u to expressions containing the
DotProducts of p1, p2, p3 and p4 and the masses.  It sets p1, p2,
p3, p4 to be on-shell with masses m1, m2, m3, m4 respectively."

Mass::usage = "Mass[p] is the mass of a four-vector p.  It is
evaluated explicitly if p is a list of length SpaceTimeDimension.
"

MasslessVectorPolarization::usage = "MasslessVectorPolarization[ p,
mu, pol] represents the polarization vector of a massless vector
particle with momentum p, polarization pol and index mu."

MasslessVectorPolarization::usage = "MasslessVectorPolarization[ p,
mu, pol] represents the polarization vector of a massless vector
particle with momentum p, polarization pol and index mu."

Momentum::usage = "Momentum[fv] is the three-component momentum of
fv."

NEvaluatePhaseSpaceIntegral::usage =
"NEvaluatePhaseSpaceIntegral[PhaseSpaceIntegral[...]]  will
evaluate the Phase-space-integral numerically."

NonCommutativeExpand::usage = "NonCommutativeExpand[expr] expands
NonCommutativeMultiply of sums."

NonCommutativeFactor::usage = "NonCommutativeFactor[expr] attempts
to factor sums of NonCommutativeMultiply's."

Opposite::usage = "Opposite[p] represents a four-momentum which has
the same energy but opposite three momentum to the four-momentum
p."

PerpendicularMomentum::usage = "PerpendicularMomentum[fv] gives the
component of fv perpendicular to the z-axis."

PhaseSpaceIntegral::usage = "PhaseSpaceIntegral[integrand, {p1,
(p2)}, decays, integrals, extmom] specifies a phase-space integral.
Use EvaluatePhaseSpaceIntegral to evaluate the integral
symbolically, and NEvaluatePhaseSpaceIntegral to evaluate it
numerically."

PolarizationCombinations::usage = "PolarizationCombinations[me]
returns a list of the possible polarization combinations of the
initial and final particles for the matrix-element me over the
polarizations which are not explicitly written in the
matrix-element."

PrepareIndex::usage = "PrepareIndex[a, b, c, ...] makes future
contractions involving a, b, c, ... work faster."

Propagator::usage = "Propagator[line] is the factor associated with
the particle line, where line can be {type, p} or {type, p, mu, nu}
where type is the type of particle, p is the momentum, and mu and
nu are Lorentz indexes.  Propagator[{type, p__}] is the same as
Propagator[type][p]"
```

```
STToTraces::usage = "STToTraces[expr] attempts to convert products
of SpinorS's and SpinorT's in expr to traces using the rule
SpinorS[p1,p2]*SpinorT[p2, p3]...SpinorS[pn, p1] =
GammaTrace[Slash[p1]**Slash[p2]**...
**Slash[pn]**HelicityProjection[Right]]"

SetDotProduct::usage = "SetDotProduct[{p, q, u}] sets DotProduct[p,
q] to be u.  SetDotProduct[{p1, q1, u1}, {p2, q2, u2},...] sets
DotProduct[p1, q1] to be u1, DotProduct[p2, q2] to be u2, etc."

SetMandelstam::usage = "SetMandelstam[{p1, p2, p3, p4}, {m1, m2,
m3, m4}, s, t, u] sets p1, p2, p3, p4 to be on-shell with masses
m1, m2, m3, m4 respectively and sets the DotProducts of p1, p2, p3
and p4 in terms of the Mandelstam variables s, t and u and the
masses."

SetMass::usage = "SetMass[{p, m}] sets p to be a four-vector with
invariant mass m.  SetMass[{p1, p2,..., m}] sets p1, p2,.. to be
four-vector with invariant mass m.  SetMass[{p1, m1}, {p2, m2},...]
applies SetMass to {p1, m1}, {p2, m2},.. in turn."

SetNonCommutative::usage = "SetNonCommutative[a, b, c, ...] sets
the symbols a, b, c, ... to be non-commutative."

SetReal::usage = "SetReal[a] sets Conjugate[a] = a.  SetReal[a,
b,...]  applies SetReal to a, b,... in turn."

Slash::usage = "Slash[p] is the same as
Contract[p[mu]DiracGamma[mu], mu].  Slash is set to be
non-commutative."

SpaceDirection::usage = "SpaceDirection[fv] gives the pair {cth,
phi} describing the direction of the momentum of fv."

SpaceTimeDimension::usage = "SpaceTimeDimension is what you'd
expect."

Spherical::usage = "Spherical[p1, p2] indicates a spherically
symmetric two body decay into p1 and p2."

SpinorS::usage = "SpinorS[p, q] is the same as SpinorUbar[p,
Right]**SpinorU[q, Left]. Both p and q have to be massless."

SpinorT::usage = "SpinorT[p, q] is the same as SpinorUbar[p,
Left]**SpinorU[q, Right]. Both p and q have to be massless."

SpinorU::usage = "SpinorU[p] is a spinor object of a four-vector p.
SpinorU[p, pol] is a spinor object of particular polarization."

SpinorUbar::usage = "SpinorUbar[p] is a spinor object of a
four-vector p.  SpinorUbar[p, pol] is a spinor object of particular
polarization."

SpinorV::usage = "SpinorV[p] is a spinor object of an anti-particle
of four-vector p. SpinorV[p, pol] is a spinor object of particular
polarization."

SpinorVbar::usage = "SpinorVbar[p] is a spinor object of an
anti-particle of four-vector p. SpinorVbar[p, pol] is a spinor
```

```
object of particular polarization."

Vertex::usage = "Vertex[line1, line2, ...] is the Feynman rule for
the vertex.  Each line is of the form {particle} or {particle, p}
or {particle, p, indexes} where particle is the particle type, p is
the particle's momentum, and indexes are the appropriate Lorentz or
gauge group indexes."

ZAxis::usage = "ZAxis is an option used with functions receiving a
direction as an argument. It specifies the axis with respect to
which the direction is given."
```

REFERENCES

1. H. Murayama, I. Watanabe, K. Hagiwara, "HELAS: HELicity Amplitude Subroutines for Feynman diagram evaluations", to appear in KEK-Report.
2. *Schoonship* by M. Veltman, see H. Strubbe, *Comp. Phys. Comm*/**8**(1974), 1.
3. *Reduce* by A. C. Hearn, see Reduce User's Manual, version 3.2, Rand Crop., 1985.
4. *Mathematica*: A System for Doing Mathematics by Computer, S. Wolfram, Addison-Wesley Publishing Company, 1988.
5. *Maple* by Waterloo Maple Corp., see MAPLE Reference Manual, Fifth Edition, Waterloo Maple Publishing, 1988.
6. J. Küblbeck, M. Böhm and A. Denner, *Comp. Phys. Comm.*/**60**(1990), 165.
7. R. Mertig, M. Böhm and A. Denner, *Comp. Phys. Comm.*/**64**(1991), 345.
8. R. Kleiss, W. J. Stirling, *Nucl. Phys.*/**B262**(1985), 235.
9. Note that the p's are used in a dual mode, both as representing momenta and as representing particles. The mass m associated with p is the mass of the particle carrying the momentum p. For off-shell particles, $p^2 \neq m^2$ (`DotProduct[p, p] != Mass[p]^2`).
10. Most of HIP's functions operate well in arbitrary D dimensions. The exceptions are the functions dealing with vectors given in terms of their explicit components (*e.g.* `Boost`), functions associated with phase-space integrals and functions treating γ^5.
11. M. Veltman, *Nucl. Phys.*/**B319**(1989), 253.

Table 1. HIP functions representing *objects*

HIP Function	Example	Physical Equivalent
`{}`	`{px, py, pz, e}`	The four-vector (p_x, p_y, p_z, E)
`DotProduct`	`DotProduct[p, q]`	$p \cdot q$
`G`	`G[mu, nu]`	$g^{\mu\nu}$
`Eps`	`Eps[p, q, mu, nu]`	$\epsilon_{\tau\sigma\mu\nu} p^\tau q^\sigma$
`DiracGamma`	`DiracGamma[mu]`	γ^μ
`Slash`	`Slash[p]`	$\not{p}$
`DiracGamma5`	`DiracGamma5`	γ^5
`**`	`DiracGamma[mu]**Slash[p]`	$\gamma_\mu \not{p}$
`SpinorU`	`SpinorU[p, lambda]`	$u_\lambda(p)$
`SpinorUbar`	`SpinorUbar[p, lambda]`	$\overline{u}_\lambda(p)$
`HelicityProjection`	`HelicityProjection[Left]`	$P_L = (1 - \gamma^5)/2$
`SpinorS`	`SpinorS[p, k]`	$s(p,k) = \overline{u}_R(p) u_L(k)$
`SpinorT`	`SpinorT[p, k]`	$t(p,k) = \overline{u}_L(p) u_R(k)$

FIGURE CAPTIONS

1) Feynman diagram for muon decay $\mu^- \to e^- \overline{\nu}_e \nu_\mu$.

2) Feynman diagram for the process $e^+_R e^-_R \to Z \to t\bar{t} \to W^+ W^- b\bar{b}$.