FORM and Field Theory

J.A.M. Vermaseren

Nikhef

- Why FORM?
- Prehistory: Traces
- Middle ages: Version 1 and 2
- Modern times: Version 3
- The current status
- Planning ahead

Why FORM?.

In particle theory we have categories of calculations that are particularly demanding on hardware and software facilities. So much so that particle theory has stood at the cradle of symbolic computation and also afterwards has made large contributions to it. Yet, as soon as a system becomes bigger and bigger and commercially interesting, it often leaves its origins and it becomes more and more difficult to influence its development.

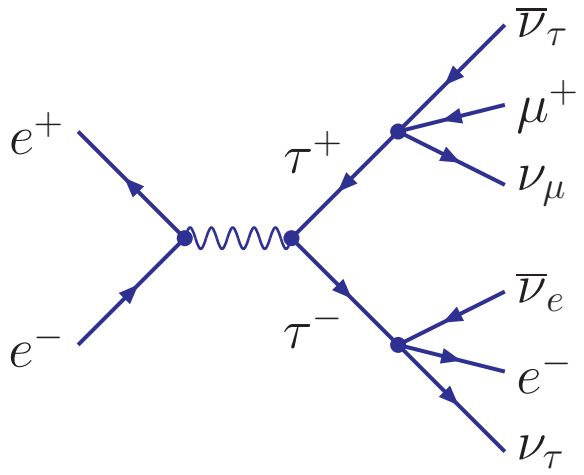
For research it is important to have a system of which the author(s) is/are involved in active research themselves. This way a system will adapt to the developing needs while avoiding a very lengthy cycle of interaction with the organization behind a commercial product.
Additionally, a system should be available to all researchers. Publishing ones results in the language of an expensive system is rather elitist. It means that a large number of people cannot use these results or is forced into illegality.
It is best if a system is open source. With enough documentation it will allow the advanced user the quickest way to satify his/her needs. Currently this aspect of FORM is worked at. Without the documentation of the internals, it doesn't seem a good idea to release the sources. We anticipate their release in about two years (or sooner).

Prehistory: Traces

If one looks at the history of calculations in particle theory one sees a development over the years.

At first the symbolic manipulation was to combine tensors and four vector dotproducts and manipulate gamma matrices. This was what Schoonschip was designed for and also one of the first things that FORM could do. Because there are still people who think in these terms (mainly people who are not in particle phenomenology) FORM is still seen by many as a program that is only suitable for particle physics, just because it is good at traces (and there isn't a big advertizing department).

There are however still some unresolved problems here. Let us take the following reaction at low energy:


e^+
e^-
τ^+
τ^-
$\overline{\nu}_\tau$
μ^+
ν_μ
$\overline{\nu}_e$
e^-
ν_τ

```
    V   p1,p2,Q,q1,q2,p3,p4,p5,p6,p7,p8;
    I   m1,m2,m3;
    I   n1,n2,n3;
    S   emass,tmass,mass4,mass5,mass7,mass8;
    L   F =
        (g_(1,p2)-emass)*g_(1,m1)
        *(g_(1,p1)+emass)*g_(1,n1)
        *g_(2,p3)*g_(2,m2)*g7_(2)
        *(g_(2,q1)+tmass)*g_(2,m1)
        *(-g_(2,q2)+tmass)*g_(2,m3)*g7_(2)*g_(2,p6)
        *g_(2,n3)*g7_(2)*(-g_(2,q2)+tmass)*g_(2,n1)
        *(g_(2,q1)+tmass)*g_(2,n2)*g7_(2)
        *(g_(3,p4)+mass4)*g_(3,m2)*g7_(3)
        *(g_(3,p5)-mass5)*g_(3,n2)*g7_(3)
        *(g_(4,p7)+mass7)*g_(4,m3)*g7_(4)
        *(g_(4,p8)-mass8)*g_(4,n3)*g7_(4)
        ;
    trace4,4;
    trace4,3;
    trace4,1;
    trace4,2;
    print +f +s;
    .end

Time =       0.00 sec    Generated terms =        164
               F         Terms in output =         27
                         Bytes used      =       1384
```

```
F =
    - 524288*p1.p2*q1.q2*p3.p4*p5.p7*p6.p8*tmass^2
    + 524288*p1.p2*q1.p5*q2.p7*p3.p4*p6.p8*tmass^2
    + 524288*p1.p2*q1.p7*q2.p5*p3.p4*p6.p8*tmass^2
    + 1048576*p1.q1*p2.q2*q1.p5*q2.p7*p3.p4*p6.p8
    + 524288*p1.q1*p2.q2*p3.p4*p5.p7*p6.p8*tmass^2
    - 524288*p1.q1*p2.p7*q1.p5*q2.q2*p3.p4*p6.p8
    - 524288*p1.q1*p2.p7*q2.p5*p3.p4*p6.p8*tmass^2
    + 1048576*p1.q2*p2.q1*q1.p5*q2.p7*p3.p4*p6.p8
    + 524288*p1.q2*p2.q1*p3.p4*p5.p7*p6.p8*tmass^2
    - 524288*p1.q2*p2.p5*q1.q1*q2.p7*p3.p4*p6.p8
    - 524288*p1.q2*p2.p5*q1.p7*p3.p4*p6.p8*tmass^2
    - 524288*p1.p5*p2.q2*q1.q1*q2.p7*p3.p4*p6.p8
    - 524288*p1.p5*p2.q2*q1.p7*p3.p4*p6.p8*tmass^2
    + 262144*p1.p5*p2.p7*q1.q1*q2.q2*p3.p4*p6.p8
    + 524288*p1.p5*p2.p7*q1.q2*p3.p4*p6.p8*tmass^2
    + 262144*p1.p5*p2.p7*p3.p4*p6.p8*tmass^4
    - 524288*p1.p7*p2.q1*q1.p5*q2.q2*p3.p4*p6.p8
    - 524288*p1.p7*p2.q1*q2.p5*p3.p4*p6.p8*tmass^2
    + 262144*p1.p7*p2.p5*q1.q1*q2.q2*p3.p4*p6.p8
    + 524288*p1.p7*p2.p5*q1.q2*p3.p4*p6.p8*tmass^2
    + 262144*p1.p7*p2.p5*p3.p4*p6.p8*tmass^4
    + 262144*q1.q1*q2.q2*p3.p4*p5.p7*p6.p8*emass^2
    - 524288*q1.q1*q2.p5*q2.p7*p3.p4*p6.p8*emass^2
    + 1048576*q1.q2*q1.p5*q2.p7*p3.p4*p6.p8*emass^2
    - 524288*q1.p5*q1.p7*q2.q2*p3.p4*p6.p8*emass^2
    + 1048576*q1.p5*q2.p7*p3.p4*p6.p8*emass^2*tmass^2
    + 262144*p3.p4*p5.p7*p6.p8*emass^2*tmass^4
   ;
```

The above still looks rather normal. Problem is that we have used the special trace algorithms of FORM that manage to avoid the worst troubles. We can obtain the 'troubles' by switching off some features:

```
    trace4,nocontract,4;
    trace4,nocontract,3;
    trace4,nocontract,1;
    trace4,nocontract,2;
    contract,0;
    print +f +s;
    .end

Time =       0.43 sec    Generated terms =     275030
               F         Terms in output =       2585
                         Bytes used      =      83032
```

Numerically the answer is still the same, but its formula representation is a disaster. The last terms of the output look like:

```
      - 32768*e_(p5,p6,p7,p8)*p1.p2*p3.p4*tmass^4
      - 16384*e_(p5,p6,p7,p8)*q1.q2*p3.p4*emass^2*tmass^2
      - 32768*e_(p5,p6,p7,p8)*q1.p3*q2.p4*emass^2*tmass^2
      + 49152*e_(p5,p6,p7,p8)*q1.p4*q2.p3*emass^2*tmass^2
      - 32768*e_(p5,p6,p7,p8)*p3.p4*emass^2*tmass^4
    ;
```

On theoretical grounds we know that these terms shouldn't be there!

The problem is with the Schouten identity

$$\begin{aligned}\epsilon^{\mu_1\mu_2\mu_3\mu_4}\delta^{\mu_5\mu_6} \;=\; & \epsilon^{\mu_5\mu_2\mu_3\mu_4}\delta^{\mu_1\mu_6} + \epsilon^{\mu_1\mu_5\mu_3\mu_4}\delta^{\mu_2\mu_6} \\ & +\epsilon^{\mu_1\mu_2\mu_5\mu_4}\delta^{\mu_3\mu_6} + \epsilon^{\mu_1\mu_2\mu_3\mu_5}\delta^{\mu_4\mu_6}\end{aligned}$$

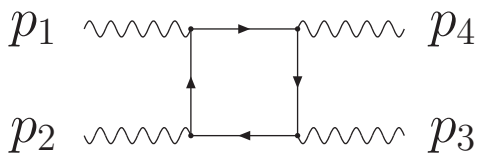
which is another way of saying that an antisymmetric object with 5 indices in 4 dimensions should be zero.
We have no way of applying this identity systematically. The complication is that the indices have been contracted with vectors as in

$$\begin{aligned}\epsilon^{p_1p_2p_3p_4}\; p_5.p_6 \;=\; & \epsilon^{p_5p_2p_3p_4}\; p_1.p_6 + \epsilon^{p_1p_5p_3p_4}\; p_2.p_6 \\ & +\epsilon^{p_1p_2p_5p_4}\; p_3.p_6 + \epsilon^{p_1p_2p_3p_5}\; p_4.p_6\end{aligned}$$

and we may not have all terms of one equation, because the resulting terms need to be treated again, but with a different spectator momentum. Or worse even, the dotproduct with which we have to combine a Levi-Civita tensor may have been in a denominator as well and hence cancelled. It is a mess.

Middle ages: Loop reductions

In field theory we calculate reactions in a perturbation expansion. The expansion parameter is $\hbar$, the Planck constant. In terms of Feynman diagrams this is equivalent to expanding in the number of closed loops inside the diagrams. Hence usually the lowest order has no loops and is called the tree level. Then come the one-loop graphs etc. In exceptional cases a reaction has no tree graphs and the lowest order is at the one-loop level as in the reaction $\gamma\gamma \longrightarrow \gamma\gamma$ in which the lowest order diagrams look like



Loop graphs are much harder to calculate than tree graphs. And each additional loop adds new levels of extreme complexity. The state of the art is that most people are currently worrying about one loop calculations for the LHC. Then there are still a reasonable number of people doing two loop calculations for reactions for which this precision is really needed. There are few people doing three loop calculations and only a limited number has been possible thus far. At the four loop level really few calculations of real reactions exist, requiring astronomical computer resources. Here we will look shortly at what is needed for one loop reactions and then at some higher loop things.

The typical one loop diagram looks like the diagram above: it has a number of external legs and a number of internal lines. The number of lines in the loop determines greatly the complexity of the problem. If there are only two lines we talk about a two-point function, etc.

There is an added complication in that the diagrams/integrals may be divergent. For this an ingenious scheme of regularization has been developed in which the integrals are not over a 4-dimensional space-time but over a space-time that is $n = 4 + \epsilon$ dimensional. This is called n-dimensional regularization. For this it has been shown that the whole theory can be formulated in n dimensions and one has to do the calculation in such a way that the limit $n \to 4$ is stable. Because this regularization also solves a second problem with massless particles, it is the most widely used method. It gives all divergences as powers of $1/\epsilon$.
There are still some outstanding problems here. In supersymmetry the n-dimensional regularization doesn't quite work and another method is needed. One uses something called n-dimensional reduction in which n has to be less than 4 in a special way as some particles still have to behave as 4-dimensional. To my knowledge (which isn't too profound here) it doesn't have the same theoretical basis as the n-dimensional regularization for the standard model.
Mathematicians usually don't want to get involved.....

When we do a one loop calculation we first do the Dirac algebra stuff etc to obtain an expression in terms of 4-vector products. The propagators (internal lines) give us denominators and we have to integrate this expression over the 4-momentum in the loop (or as said n-momentum). One way to do this is to reduce all integrals to a very limited set of integrals in which there are no powers of the loop momentum in the numerator. Such integrals are called scalar-loop integrals because they are all that occurs in theories that contain only scalar (spin 0) particles. These integrals are then processed by different (often numerical) means.

The amplitude for the reaction $\gamma\gamma \to \gamma\gamma$ is given by

$$A = T_b A_b + T_v A_v + T_s A_s + T_u A_u + T_t A_t$$

When the intermediate particle is a spin 0 boson the amplitudes are written as A^0, for the fermions it will be $A^{\frac{1}{2}}$ and for vector bosons it is A^1. The tensors that define the various amplitudes are given by

$$\begin{aligned}
T_b &= b_1^\mu b_2^\nu b_3^\rho b_4^\sigma / \Delta_3^2 \\
T_v &= v^\mu v^\nu v^\rho v^\sigma / \Delta_3^2 \\
T_s &= v^\mu v^\nu b_3^\rho b_4^\sigma / \Delta_3^2 + b_1^\mu b_2^\nu v^\rho v^\sigma / \Delta_3^2 \\
T_t &= v^\mu b_2^\nu b_3^\rho v^\sigma / \Delta_3^2 + b_1^\mu v^\nu v^\rho b_4^\sigma / \Delta_3^2 \\
T_u &= -v^\mu b_2^\nu v^\rho b_4^\sigma / \Delta_3^2 - b_1^\mu v^\nu b_3^\rho v^\sigma / \Delta_3^2 \\
b_1^\mu &= \delta^{p_1 \mu}_{p_2 p_3} \\
b_2^\nu &= \delta^{p_2 \nu}_{p_1 p_3} \\
b_3^\rho &= \delta^{p_3 \rho}_{p_1 p_2} \\
b_4^\sigma &= \delta^{p_4 \sigma}_{p_1 p_2} \\
v^\mu &= \varepsilon^{p_1 p_2 p_3 \mu}
\end{aligned}$$

The terms in these tensors are orthonormal. Therefore the matrix element squared is rather simple:

$$|A|^2 = A_b^2 + A_v^2 + 2A_s^2 + 2A_t^2 + 2A_u^2$$

In addition we may note that $A_t = A_s(s \to t)$ and $A_u = A_s(s \to u)$, so that we have to give only three amplitudes.

A useful set of variables is given by:

$$\begin{aligned} D^{(i)} &= s^i D_s + t^i D_t + u^i D_u \\ C^{(i)} &= s^i C_s + t^i C_t + u^i C_u \\ B^{(i)} &= s^i B_s + t^i B_t + u^i B_u \\ \Delta_3 &= stu/4 \end{aligned}$$

in which the B, C and D are scalar two, three and four point functions. There is one peculiarity: D_u is the four point function in s and t. C_s and B_s are functions of s. The amplitudes are:

$$
\begin{aligned}
A_b^0 &= 12 + 32\Delta_3^2 D^{(-4)} - 4B^{(0)} + B^{(3)}/\Delta_3 \\
&\quad + \frac{1}{4}C^{(7)}/\Delta_3^2 - 3C^{(4)}/\Delta_3 + m^2(16C^{(0)} - 2C^{(3)}/\Delta_3) \\
A_v^0 &= -4 + 32\Delta_3^2 D^{(-4)} + 64m^2\Delta_3 D^{(-2)} + 32m^4 D^{(0)} - 4B^{(0)} + B^{(3)}/\Delta_3 \\
&\quad + \frac{1}{4}C^{(7)}/\Delta_3^2 - 3C^{(4)}/\Delta_3 + m^2(-16C^{(0)} + 6C^{(3)}/\Delta_3) \\
A_s^0 &= 4 + 32\Delta_3^2 D^{(-4)} + 32m^2\Delta_3 D^{(-2)} - 64\Delta_3(\Delta_3/s^4 + m^2/s^2)D_s \\
&\quad - \frac{1}{4}C^{(7)}/\Delta_3^2 + C^{(4)}/\Delta_3 - \frac{1}{2}sC^{(6)}/\Delta_3^2 + 2sC^{(3)}/\Delta_3 \\
&\quad - m^2(2C^{(3)} + 4sC^{(2)})/\Delta_3 + (s^7/\Delta_3 - 6s^4 + 8m^2s^3)C_s/\Delta_3 \\
&\quad - B^{(3)}/\Delta_3 - 2sB^{(2)}/\Delta_3 - 4B_s + 4s^3 B_s/\Delta_3 \\
A_b^{\frac{1}{2}} &= -2A_b^0 + 8\Delta_3 D^{(-1)} - 8C^{(1)} \\
A_v^{\frac{1}{2}} &= -2A_v^0 + 8\Delta_3 D^{(-1)} - 8C^{(1)} \\
A_s^{\frac{1}{2}} &= -2A_s^0 + 8\Delta_3 D^{(-1)} - 8m^2 sD^{(0)} - 8\Delta_3 D_s/s - 4sC_s \\
A_b^1 &= 3A_b^0 - 64\Delta_3 D^{(-1)} + 8D^{(2)} + 32C^{(1)} \\
A_v^1 &= 3A_v^0 - 64\Delta_3 D^{(-1)} + 8D^{(2)} + 32C^{(1)} \\
A_s^1 &= 3A_s^0 - 32\Delta_3 D^{(-1)} + 32m^2 sD^{(0)} + 32\Delta_3 D_s/s + 32sC_s \\
&\quad + 8D^{(2)} + 8sD^{(1)} - 16s^2 D_s
\end{aligned}
$$

It can be quite some work to do this and it is nearly always done by computer. Because the above result had much manual interference it is still presentable. Usually this isn't the case as there are many diagrams and everything needs to be processed 100% automatically.
This is the point where these calculations become kind of an art. No two major players in the field use the same method to do this reduction. Everybody has their own method. This is because there are quite a few problems. One is numerical stability. The resulting formulas are usually evaluated in a FORTRAN (or C) program to perform a Monte Carlo integration.
Because of the gauge invariance of field theory there are many objects that might be written as the contraction of two Levi Civita tensors as in

$$\begin{aligned}\Delta_4(p_1p_2p_3p_4) &= \epsilon^{p_1p_2p_3p_4}\epsilon^{p_1p_2p_3p_4} \\ &= \cdots\end{aligned}$$

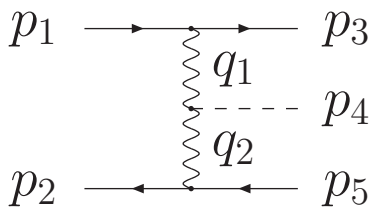
or

```
    Vector  p1,p2,p3,p4;
    Local F = e_(p1,p2,p3,p4)*e_(p1,p2,p3,p4);
    Contract;
    Print +s;
    .end

Time =       0.00 sec    Generated terms =         24
               F         Terms in output =         17
                         Bytes used      =        474

   F =
       + p1.p1*p2.p2*p3.p3*p4.p4
       - p1.p1*p2.p2*p3.p4^2
       + 2*p1.p1*p2.p3*p2.p4*p3.p4
       - p1.p1*p2.p3^2*p4.p4
       - p1.p1*p2.p4^2*p3.p3
       + 2*p1.p2*p1.p3*p2.p3*p4.p4
       - 2*p1.p2*p1.p3*p2.p4*p3.p4
       - 2*p1.p2*p1.p4*p2.p3*p3.p4
       + 2*p1.p2*p1.p4*p2.p4*p3.p3
       - p1.p2^2*p3.p3*p4.p4
       + p1.p2^2*p3.p4^2
       + 2*p1.p3*p1.p4*p2.p2*p3.p4
       - 2*p1.p3*p1.p4*p2.p3*p2.p4
       - p1.p3^2*p2.p2*p4.p4
       + p1.p3^2*p2.p4^2
       - p1.p4^2*p2.p2*p3.p3
       + p1.p4^2*p2.p3^2
      ;
```

in the reaction



The particle that is produced in the center is a pseudo scalar particle. In terms of Levi-Civita tensors the Matrix element (=Probability density) for this reaction can be written as

$$|M|^2 = \left(\frac{8p_1^\mu p_1^\nu - 2q_1^2\delta^{\mu\nu}}{(q_1^2)^2}\right)\left(\frac{8p_2^\rho p_2^\sigma - 2q_2^2\delta^{\rho\sigma}}{(q_2^2)^2}\right)\epsilon^{\mu\rho q_1 q_2}\epsilon^{\nu\sigma q_1 q_2}$$

When it is worked out in dotproducts, as most people will do and all automatic programs will do as well, there are individual terms that behave like E^8 while in regions where most of the reaction takes place (at extremely small angles for the outgoing electron/positron) the value of Δ_4 can be smaller by 20 orders of magnitude. Hence most of the work consists of trying to avoid this problem. In practise most people only calculate this reaction in an approximation because they don't know how to avoid these problems.

It would be a great help if someone would figure out how to rewrite an output in dotproducts into these 'Gram determinants' in a way that would make sense. Unfortunately that is not trivial.

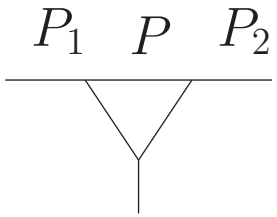
Related to this is the problem of trying to represent the (sometimes very large) output in as short a way as possible. We call this simplification. If the reaction gets rather complicated there may be many thousands of diagrams and each giving a large formula. In one case we had 50 Gbytes of FORTRAN code (divided over 250000 subroutines). And already this was reduced by about a factor three.

```
 xre=xre+cc8*(xw(31)*xu(2)*xz(94)+xw(31)*xu(8)*xz(94)*x3-xw(31)*
&    xz(9)*xz(21)*x3-xw(31)*xz(33)*amel2*x3-xw(31)*xz(54)+xw(33)*
&    xu(2)*xz(40)*x1**2-xw(33)*xz(9)*xz(56)*x1+xw(34)*xz(5)*xz(70)
&    *x3-xw(34)*xz(85)+xw(38)*xz(5)*xz(70)*x1*x3-xw(39)*xz(54)*x1-
&    xw(39)*xz(100)-xw(43)*xz(85)*x3+xw(44)*xz(85)*x1-xw(46)*xz(91
&    )*x3-xw(47)*xz(96)*x3-xw(48)*xz(55)*amel2*x3-xw(48)*xz(90)+
&    xw(63)*xu(2)*xz(100)*x1-xw(74)*xu(2)*xz(41)*x1-xu(5)*xz(106)-
&    xz(9)*xy(11)*x3-xz(31)*xy(10)*x3+xz(39)*xy(14)*x3-xz(41)*xy(
&    15)*x3-xz(53)*xy(46)*x1*x3+xz(58)*xy(34)*x3-xz(64)*xy(36)*x3+
&    xz(65)*xy(39)*x3+xz(81)*xy(29)*x1*x3-xz(82)*xy(25)*x1**2+xz(
&    93)*xy(12)*x3**2-xz(95)*xy(24)*x3-xz(97)*xy(19)*x3**2-xz(99)*
&    xy(35)*x3+xz(100)*xy(22)*x1**2-xz(103)*xy(41)*x3-xz(112)*x3*
&    xlevi-(1-2*xnlaa2-3*xnla)*(xz(98))*xcp1+(1-xnlaa2)*(xw(32)*
&    xz(48))*x3+(3+4*xnlaa2+6*xnla)*(xz(101))*xcp2+(4+xnlaa2+5*
&    xnla)*(xz(106))*x1-(5+2*xnlaa2+9*xnla)*(xz(108))*x1+(7-xnlaa2
&    )*(xz(111))*x1)
 xre=xre+cc8*((7-xnlaa2+8*xnla)*(xz(64)*e3e2)*x3*xcp2+(7+xnla)*(
&    xz(108))+(11+2*xnlaa2+19*xnla)*(xz(105))*x1-(17+3*xnla)*(xz(
&    105))-(xcp2+xw(18))*(xz(30)*e4e2)*x1-(xcp2+xw(22))*(xz(48))+(
&    xcp1-xw(8))*(xu(2)*xz(95))*x1-(xcp1+xw(8))*(xz(43))-(xcp1+xw(
&    12))*(xu(2)*xz(44))*x1+(xw(2)+xw(16))*(xz(102))-(xw(2)+xw(19)
&    )*(xz(40))*x3-(xw(2)+xw(28))*(xz(47))*x3+(xw(2)+xw(29))*(xz(
&    92))*x3+(xw(3)+xw(23))*(xz(100))*x1+(xw(4)-xw(16))*(xz(9)*xz(
&    56))-(xw(4)+xw(27))*(xz(88))*x1+(xw(6)-xw(18))*(xz(51))*x1*x3
&    +(xw(7)+xw(30))*(xz(50))*x3+(xw(9)+xw(17))*(xu(2)*xz(99))*x1-
&    (xw(12)+xw(19))*(xz(40))*x1**2+(xw(13)-xw(19))*(xz(44))*x3-(
&    xw(14)-xw(19))*(xz(43))*x1**2-(2*xb(1)*xz(2)*e3e2-2*xb(5)*xu(
&    1)*xz(62)+xb(10)*xz(27)-2*xb(17)*xz(35)+xb(27)*xu(4)*xz(24)+
&    xz(2)*xz(70)-6*xz(3)*xz(21)+xz(3)*xz(25)-2*xz(3)*xy(9)+2*xz(
&    21)*xy(44)-2*xz(58)*xy(52))*x3**2*xcp1-(4*xb(7)*xz(11)+xb(10)
&    *xz(27)-2*xb(13)*xu(4)*e4e2-2*xb(15)*xz(2)-2*xb(16)*xz(3)-2*
&    xb(17)*xz(35)+2*xb(24)*xu(1)*xz(61)+2*xu(1)*xz(21)*e3e1+2*xu(
&    2)*xz(99)+2*xu(4)*xz(14)*e3e2+xu(4)*xz(23)*amh2+4*xu(6)*xz(74
&    )-4*xw(10)*xw(55)+xw(28)*xw(43)-8*xw(30)*e4e2)*x1*x3*xcp1)
```

Recently some techniques based on superstrings have entered the field and people have been able to take some one loop reactions to extremes without looking at the individual diagrams. Unfortunately this works thus far only in purely massless cases.

Also, very little here is connected to computer algebra. Most is done either by hand or numerically.

The other topic here is the reduction of multi-loop integrals to a limited set of 'master integrals'. There are several techniques for this, mostly based on integration by parts. To illustrate this, take the diagram



The triangle subgraph

We define the integral:

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

The we consider this total derivative:

$$\int d^DP\,\left\{\frac{\partial}{\partial P_\mu}\frac{P_\mu \qquad \mathcal{P}_n(P)}{(P^2)^{\alpha_0}((P+p_1)^2)^{\beta_1}(p_1^2)^{\alpha_1}((P+p_2)^2)^{\beta_2}(p_2^2)^{\alpha_2}}\right\} \;=\; 0$$

Working out the derivations gives

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

This can then be as a recursion relation to work away one of the denominators and obtain a simpler diagram.

This recursion can be solved (Tkachov):

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

There are several of these equations and together they may allow a complete reduction in a giant recursion scheme. Intermediate results may become rather large, especially when one makes power series expansions in one of the variables.

There are much more complicated recursions and it would be very nice if those could be solved in the same way. Unfortunately.....

The remaining master integrals have in some cases been worked out to sufficient powers in ϵ and they can be substituted.

One successful package here is the Mincer package. It has been used for a number of three-loop calculations.

```
#define POW "6"
#define PROJ "0"
#define CURRENT "F2"
#include- mincer.h
.global
#include- diagram.h
        ;
.sort
#call treatqaqa(‘POW’)
.sort
#call integral(‘TOPO’)
.sort
#call trim(‘TOPO’)
print;
.end
   d9c =
      12520988335127/893025000 - 1507544/10125*epˆ-3
       - 33324538/50625*epˆ-2 - 9676515073/31893750*epˆ-1
       - 46848/175*epˆ-1*z3 - 440320/7*z5 - 70272/175*z4
       + 21504323128/496125*z3;

  24.78 sec out of 24.79 sec
```

The holy grail at the moment is to try to make a similar package for 4-loop calculations. Several people have been looking into this already for several years, but unsuccessfully. The best is a related expansion method by P. Baikov which is extremely costly in computer time. His program is called Baicer, but alas, it isn't publicly available.

Modern times: Sums

Since the late nineties new trends have emerged. Not only new techniques for the rewriting of the diagrams in terms of master integrals were developed, but also new methods for the treatment of the integrals themselves saw the daylight. Most notoriously methods with nested sums. And the rewriting of integrals started to need the solving of large sets of equations. This occurs in several ways: Some methods of recursion lead to difference equations which one may solve with a trial function that may contain ten thousands of elements. The second method is that a system of recursions gives relations inside a large class of integrals and by solving these equations one can express them all in terms of a number of 'master integrals'. In the last case the coefficients are usually rational polynomials in a limited number of variables.

The sums have already been part of QCD since its beginning, but nobody really made an attempt to make a real method out of it. Then, almost simultaneously two methods needed them.

- The calculation of all Mellin moments of the deep inelastic structure functions. These were needed for a NNLO determination of the quark and gluon distributions inside the proton. Here finite sums were needed.
- The use of the Mellin Barnes transformation to attack the master integrals in the two loop scattering of two electrons. This approach needed infinite sums.

As it turns out, mathematicians have been looking at these sums, but only very partially. They look almost exclusively at the infinite sums and then only at the non-alternating sums. In that case they are called multiple zeta values (MZV's). Also the name Euler-Zagier sums is used. A notation that is computer friendly had to be invented and the appropriate programs had to be made. This became the summer library. A set of related functions has been defined as well. These are called harmonic polylogarithms (HPL's). Also for them several libraries have been constructed by now.

Harmonic sums are defined by:

$$
\begin{aligned}
S_m(N) &= \sum_{i=1}^{N} \frac{1}{i^m} \\
S_{-m}(N) &= \sum_{i=1}^{N} \frac{(-1)^i}{i^m} \\
S_{m,m_2,\cdots,m_p}(N) &= \sum_{i=1}^{N} \frac{1}{i^m} S_{m_2,\cdots,m_p}(i) \\
S_{-m,m_2,\cdots,m_p}(N) &= \sum_{i=1}^{N} \frac{(-1)^m}{i^m} S_{m_2,\cdots,m_p}(i)
\end{aligned}
$$

This is a notation that is also suitable for computers. There is a difference here between various definitions as there are also people using $i-1$ for the argument of the S in the recursive formula. Those sums we call Z-sums.

The harmonic polylogarithms are defined by:

$$\begin{aligned} H(0;x) &= \ln x \\ H(1;x) &= \int_0^x \frac{dx'}{1-x'} = -\ln(1-x) \\ H(-1;x) &= \int_0^x \frac{dx'}{1+x'} = \ln(1+x) \end{aligned}$$

and the functions

$f(0;x) = \frac{1}{x}, \quad f(1;x) = \frac{1}{1-x}, \quad f(-1;x) = \frac{1}{1+x}$

If $\vec{a}_w$ is an array with w elements, all with value a, then:

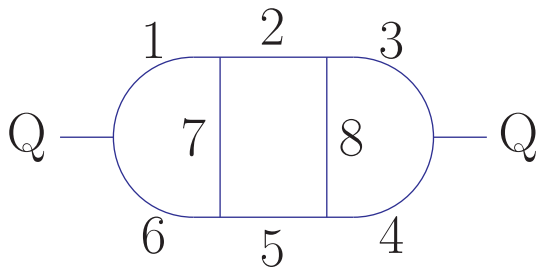
$$\begin{aligned} H(\vec{0}_w;x) &= \frac{1}{w!}\ln^w x \\ H(a,\vec{m}_w;x) &= \int_0^x dx'\ f(a;x')\ H(\vec{m}_w;x') \end{aligned}$$

```
 #define SIZE "6"
 #include- harmpol.h
 Off statistics;
 .global
 Local F = S(R(-1,3,-2),N);
 #call invmel(S,N,H,x)
 Print +f +s;
 .end

F =
    - 51/32*[1-x]^-1*z5
    + 3/4*[1-x]^-1*z2*z3
    - 7/2*s6
    + 51/32*z5*ln2
    - 33/64*z3^2
    + 9/4*z2*z3*ln2
    + 121/840*z2^3
    - 51/32*sign_(N)*[1+x]^-1*z5
    + 3/4*sign_(N)*[1+x]^-1*z2*z3
    - 1/2*sign_(N)*H(R(1,0,0),x)*[1+x]^-1*z2
    + 21/20*H(R(-1),x)*[1-x]^-1*z2^2
    + H(R(-1,-3,0),x)*[1-x]^-1
    + 3/2*H(R(-1,0),x)*[1-x]^-1*z3
    + 1/2*H(R(-1,0,0),x)*[1-x]^-1*z2
   ;

0.28 sec out of 0.33 sec
```

Example:



$$
\begin{aligned}
LA_{22}^{(131)} &= \int d^D p_1 d_2^D d^D p_3 \frac{(2P\cdot p_2)^8 (2p_7\cdot p_8)^3}{p_1^2 (p_2^2)^9 p_3^2 p_4^2 p_5^2 p_6^2 p_7^2 p_8^2} \\
&= (Q^2)^{1-6\epsilon} (\frac{2P\cdot Q}{Q^2})^8 \times \\
&\quad (\frac{436990847}{326592000} + \frac{1}{56}\epsilon^{-2} + \frac{107}{3360}\epsilon^{-1} - \frac{39}{35}\zeta_3)
\end{aligned}
$$

This took 127 sec. with FORM on a P1000 computer.

The general N form of the above diagram:

$$LA_{22}^{(131)} = \int d^D p_1 d_2^D d^D p_3 \frac{(2P\cdot p_2)^N (2p_7\cdot p_8)^3}{p_1^2 (p_2^2)^{N+1} p_3^2 p_4^2 p_5^2 p_6^2 p_7^2 p_8^2} =$$

```
+theta(-2+N)*(+den(-1+N)*(23/8+ep^-2+1/3*ep^-1+4/3*z3)
              +den(-1+N)^2*(-7/2-ep^-1)
              +den(-1+N)^3*(3)
              +den(-1+N)^3*S(R(1),-1+N)*(2/3)
              +den(-1+N)^2*S(R(1),-1+N)*(-31/18)
              +den(-1+N)*S(R(1),-1+N)*(37/36+ep^-1)
              +den(-1+N)*S(R(1,1),-1+N)*(1)
              +den(-1+N)*S(R(2),-1+N)*(-29/18)
              +den(-1+N)*S(R(3),-1+N)*(-2/3) )
+theta(-1+N)*(+den(N)*(-17/18-ep^-2-1/3*ep^-1+16/3*z3)
              +den(N)^2*(-73/18+ep^-1)
              +den(N)^3*(11/3)
              +den(N)^3*S(R(1),N)*(8/3)
              +den(N)^2*S(R(1),N)*(-29/9)
              +den(N)*S(R(1),N)*(23/9-ep^-1)
              +den(N)*S(R(1,1),N)*(-1)
              +den(N)*S(R(2),N)*(5/9)
              +den(N)*S(R(3),N)*(-8/3) )
+theta(N)*(   +den(1+N)*(4*z3)
              +den(1+N)^2*(8*z3)
              +den(1+N)^4*S(R(1),1+N)*(4)
              +den(1+N)^3*S(R(1),1+N)*(2/3)
              +den(1+N)^2*S(R(2),1+N)*(-2/3)
              +den(1+N)^2*S(R(3),1+N)*(-4)
              +den(1+N)*S(R(1),1+N)*(-8*z3)
              +den(1+N)*S(R(1,2),1+N)*(2/3)
              +den(1+N)*S(R(1,3),1+N)*(4)
```

This kind of work has also led to the need to solve difference equations. For equations that have a solution space that consists of harmonic sums we have written some programs, but they are currently not in a shape ready for release.

They involve substituting an Ansatz which may contain tens of thousands of elements. Then we have to solve the linear system of equations. Usually the Mincer package was used to get the boundary values. The technique for solving these equations was based on Gaussian elimination. Select a number of equations with the fewest number of terms and use these to eliminate variables. Then again look for the equations with the fewest number of terms. Etc.

This works of course only well when the system is rather sparse.

We had up to 4-th order equations.

It would be nice to have the sigma package available in FORM.

Actually FORM has been used to solve other aspects of physics and mathematics as well. One thing one has to deal with in the computation of Feynman diagrams is what we call color factors. When there is a symmetry group the vertices in the diagrams may have objects connected to them that belong to the representation of a (usually) Lie group. These have to be combined into the so-called color trace. In principle the answer for a given representation of a given group is a number which can be different for each diagram.

Here we have an example for a diagram in SU(3). We have 14 vertices and the girth of the diagram is 6. This diagram is unique. We assume we are in the adjoint representation:

```
  CFunction Tr(cyclic);
  CFunction T, Tp, f(antisymmetric);
  Symbols a,nf,NF,NA,cF,cA,[cF-cA/6];
  Dimension NA;
  AutoDeclare Index j,k,n;
  Dimension NF;
  AutoDeclare Index i;
  Off Statistics;
  .global
  *
  L g14 = f(j1,j2,j3)*f(j1,j4,j5)*f(j2,j6,j7)*f(j3,j8,j9)
   *f(j4,j10,j11)*f(j5,j12,j13)*f(j6,j14,j15)*f(j7,j16,j17)
   *f(j8,j18,j19)*f(j9,j20,j21)
   *f(j10,j21,j15)*f(j13,j19,j14)*f(j17,j11,j18)*f(j12,j16,j20);

  #call SUn
  id NF = 3;
  id a = 1/2;
  Print;
  .end

 g14 =
    297;

0.88 sec out of 0.93 sec
```

It becomes more interesting if we want to keep some parameters in as the N in SU(N). It becomes even more interesting if we want to express the answer in terms of group invariants without specifying the group or the representation in advance. The program for this works for up to 14 vertices. After that the topological problems become messy and there was no need to go beyond this in the work this was planned for.
An example of the color program:

```
  #include- color.h
  #endif
  Tensor f(antisymmetric);
  I   i1,...,i21;
  Off Statistics;
  .global
  Local girth6 = f(i1,i2,i3)*f(i1,i4,i5)*f(i2,i6,i7)
         *f(i3,i8,i9)*f(i4,i10,i11)*f(i5,i12,i13)
         *f(i6,i14,i15)*f(i7,i16,i17)*f(i8,i18,i19)
         *f(i9,i20,i21)*f(i10,i21,i15)*f(i13,i19,i14)
         *f(i17,i11,i18)*f(i12,i16,i20);
  sum i1,...,i21;
  .sort
  #call docolor
  Print +f +s;
  .end

 girth6 =
     + 1/648*NA*cA^7
     - 8/15*d444(pA1,pA2,pA3)*cA
     + 16/9*d644(pA1,pA2,pA3)
    ;
0.18 sec out of 0.20 sec
```

At the moment we developed the color package (and wrote this in the paper as a 'small' example), this result wasn't known to mathematicians. There are some rather topological statements in the color library:

```
repeat;
  if ( count(ff,1) == 0 );
    ReplaceLoop,f,a=3,l=all,outfun=ff;
    id  ff(i1?,i2?) = -cA*d_(i1,i2);
    id  ff(i1?,i2?,i3?) = cA/2*f(i1,i2,i3);
  endif;
endrepeat;
```

The ReplaceLoop statement hunts for index loops as in

$$f^{a\mu\nu}\, f^{b\nu\rho}\, f^{c\rho\mu}$$

Planning ahead.

The systems of equations that need to be solved are asking often for capabilities with rational polynomials. This is something that FORM doesn't have currently. Hence it has rather high priority to build this in. And to build this in in a rather efficient way as should be in FORM. There exist libraries for the manipulation of polynomials in a single variable, some of them claiming great efficiency, but there are no equivalent libraries for polynomials in many variables. In addition there is the problem of notation. Too much time spent on conversion will not be beneficial. Currently the problem is under study. Most univariate algorithms (in particular the GCD) have been implemented in various methods. This is by now reasonably fast. Factorization is less urgent, but can come in handy when constructing a system for simplification.

It is important to deal with multivariate rational polynomials efficiently when one likes to create a system for computing Gröbner bases. There are however several ways to deal with polynomials and each way needs its own solution:

- Small polynomials: when they take a small amount of space they can be kept inside the argument of a function. There may be billions of such polynomials. They should be treated inside the regular workspace. Univariate polynomials will usually be in this category.
- Intermediate polynomials: these could be handled by means of memory allocations as is done with the dollar variables. One could have hundreds or even thousands of them. Typically not billions.
- Large polynomials: These are complete expressions that could have billions of terms. Calculating their GCD would have to use the same mechanisms by which expressions are treated. There should be only very few of these.

```
 Symbols x,y;
 CFunction pacc;
 PolyRatFun pacc;
 L   F = pacc(x^2+x-3,(x+1)*(x+2))*y
        +pacc(x^2+3*x+1,(x+3)*(x+2))*y^2;
 Print +s;
 .sort

F =
    + y*pacc(x^2 + x - 3,x^2 + 3*x + 2)
    + y^2*pacc(x^2 + 3*x + 1,x^2 + 5*x + 6)
   ;

 id  y = 1;
 Print;
 .end

F =
   pacc(2*x^2 + 4*x - 4,x^2 + 4*x + 3);
```

Sometimes one would like to have quick private additions for things that are extremely hard to program at the FORM level. Such things are often either of combinatoric nature or special patterns. It is of course impossible to forsee what some people will need. Hence FORM should be structured in such a way that it is possible to make such additions, even though this won't be for beginners. The first requirement for this is a good documentation of the inner workings, including a number of examples. The second requirement is code that can be understood and is structured properly. Due to these two requirements FORM hasn't been released yet as open source. We hope to be this far in about two years time.

Maybe it is a good idea to make a library version of FORM as with GiNaC. And maybe even putting the two together (having routines that can convert from one to the other). It would however require much documentation. It might also be nice to connect it together with other packages, like a Gröbner basis package. And probably the converse will be nice as well: if a Gröbner basis package can use the facilities of FORM, it might greatly benefit. It may need some fundamental changes though.