# FIVE

In this lecture we are going to study integration by part identities (IBP's) for Feynman diagrams. We are going to restrict ourselves only to some "simple" graph, because in general this is a whole industry and one cannot get very far in two hours. The principles should however become very clear. At the same time we will study a number of ways of automating such manipulations. The principle of IBP's is based on the paper:

"Integration by Parts: The Algorithm to Calculate beta Functions in 4 Loops",
K.G. Chetyrkin, F.V. Tkachov, Nucl.Phys. B192 (1981) 159-204
DOI: 10.1016/0550-3213(81)90199-1.

We will study the following diagram in which all lines are massless. It is called the T1 topology:
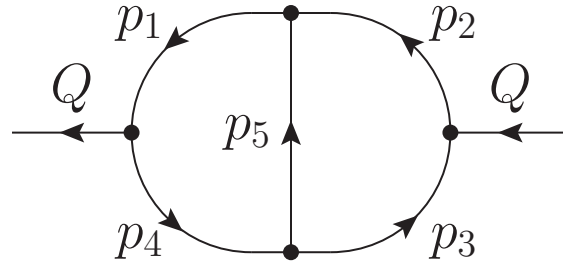
In this topology there are three independent momenta. We take them to be Q, p_1 and p_2. All dotproducts can be expressed in terms of the squares of the propagator momenta and $Q.Q$.

The basic IBP's are of the form

$$0 = \frac{\partial}{\partial p_i^\mu} p_j^\mu \frac{1}{(p_1^2)^{n_1} \cdots (p_5^2)^{n_5}}$$

For the p_i we can choose any loop momentum and hence there are two possible choices (the others being linearly dependent). For the p_j we have three choices: Q, p_1 and p_2. Therefore we can construct 6 equations. In addition the n_1 to n_5 are five independent parameters. And of course the dimension $D = 4 - 2\epsilon$ will enter the equations via

$$\frac{\partial}{\partial p_i^\mu} p_j^\mu = \delta_{ij} D$$

The next step is to decide on a notation. We will call our integrals Z and give this function the arguments $n_1 \cdots n_5$. In addition we will have the first argument to indicate the negative of the complexity of the integral which will be defined as the deviation from $n_1 \cdots n_5$. Hence the integral

$$Z(-3, n1, n2 - 1, n3, n4, n5 + 2)$$

because it has complexity three. This negative is to make sure that FORM will order the integrals in such a way that the highest complexity comes first.

Our goal will be to reduce any integral either to integrals that we can do, or a minimal set of integrals that have to be obtained by other means (master integrals). If, for instance, one of the n_i can be reduced to zero or a negative integer, the integral can be done by simple means, because it is either a product or a convolution of two one-loop integrals. Here we are more interested in cases for which such simplicity may not occur.

Let us start with the construction of the IBP's. We do this in a procedure which we call IBPforT1

```
#procedure IBPforT1
*
*    We use Q, p1 and p2 as independent momenta. Hence
*         p4 = p1-Q;
*         p3 = p2-Q;
*         p5 = p1-p2;
*
#define DER(i) "2*n'~i'*p.p'~i'/p'~i'.p'~i'"
G    A1 = der( Q,p1)*Z(n1,...,n5);
G    A2 = der(p1,p1)*Z(n1,...,n5);
G    A3 = der(p2,p1)*Z(n1,...,n5);
G    A4 = der( Q,p2)*Z(n1,...,n5);
G    A5 = der(p1,p2)*Z(n1,...,n5);
G    A6 = der(p2,p2)*Z(n1,...,n5);
*
id  der(p?,p1)*Z(n1?,...,n5?) = Z(n1,...,n5)*(
```

```
                +D*(del(p,p1)+del(p,p4)+del(p,p5))
                -'DER(1)'
                -'DER(4)'
                -'DER(5)'
                );
id  der(p?,p2)*Z(n1?,...,n5?) = Z(n1,...,n5)*(
                +D*(del(p,p2)+del(p,p3)-del(p,p5))
                -'DER(2)'
                -'DER(3)'
                +'DER(5)'
                );
id  der(p?,Q)*Z(n1?,...,n5?) = Z(n1,...,n5)*(
                +2*(n1+...+n5-D)
                -'DER(1)'
                -'DER(2)'
                );
*
id  del(p?,p?) = 1;
```

```
id  del(?a) = 0;
*
id  p4 = p1-Q;
id  p3 = p2-Q;
id  p5 = p1-p2;
*
id  Q.p1 = (Q.Q+p1.p1-p4.p4)/2;
id  Q.p2 = (Q.Q+p2.p2-p3.p3)/2;
id  p1.p2 = (p1.p1+p2.p2-p5.p5)/2;
*
id  <p1.p1^x1?>*...*<p5.p5^x5?>*Z(n1?,...,n5?) =
                        Z(<n1-x1>,...,<n5-x5>);
*
id  Q.Q = 1;
.sort
#endprocedure
```

The first thing that is new here is the use of the macro DER. It looks similar to regular define instructions, but it has an argument. As in procedures such parameters are used as preprocessor variables.

The function $\mathrm{der}(p_j,p_i)$ is the combination $\frac{\partial}{\partial p_i^\mu} p_j^\mu$ we have seen in the previous pages.

We use the DER function to take the derivatives of the propagators. It takes the direction of the momenta into account.

To try this out, we also need a main program of course. This gives:

```
Symbols D,ep,n1,n2,n3,n4,n5;
Dimension D;
Symbols x,x1,x2,x3,x4,x5;
Vector Q,p,p1,p2,p3,p4,p5;
CFunction der,del,Z,Y;
Off Statistics;
Format nospaces;
.global
#call IBPforT1
Bracket Z;
Print +f;
.end

A1=
    +Z(-1+n1,n2,n3,1+n4,n5)*(-n4)
    +Z(-1+n1,n2,n3,n4,1+n5)*(-n5)
    +Z(1+n1,n2,n3,-1+n4,n5)*(n1)
    +Z(1+n1,n2,n3,n4,n5)*(-n1)
```

```
    +Z(n1,-1+n2,n3,n4,1+n5)*(n5)
    +Z(n1,n2,-1+n3,n4,1+n5)*(-n5)
    +Z(n1,n2,n3,-1+n4,1+n5)*(n5)
    +Z(n1,n2,n3,1+n4,n5)*(n4)
    +Z(n1,n2,n3,n4,n5)*(n4-n1);

A2=
    +Z(-1+n1,n2,n3,1+n4,n5)*(-n4)
    +Z(-1+n1,n2,n3,n4,1+n5)*(-n5)
    +Z(n1,-1+n2,n3,n4,1+n5)*(n5)
    +Z(n1,n2,n3,1+n4,n5)*(n4)
    +Z(n1,n2,n3,n4,n5)*(-n5-n4-2*n1+D);

A3=
    +Z(-1+n1,n2,n3,1+n4,n5)*(-n4)
    +Z(-1+n1,n2,n3,n4,1+n5)*(-n5)
    +Z(1+n1,-1+n2,n3,n4,n5)*(-n1)
    +Z(1+n1,n2,n3,n4,-1+n5)*(n1)
```

```
    +Z(n1,-1+n2,n3,n4,1+n5)*(n5)
    +Z(n1,n2,-1+n3,1+n4,n5)*(-n4)
    +Z(n1,n2,n3,1+n4,-1+n5)*(n4)
    +Z(n1,n2,n3,1+n4,n5)*(n4)
    +Z(n1,n2,n3,n4,n5)*(n5-n1);

A4=
    +Z(-1+n1,n2,n3,n4,1+n5)*(n5)
    +Z(n1,-1+n2,1+n3,n4,n5)*(-n3)
    +Z(n1,-1+n2,n3,n4,1+n5)*(-n5)
    +Z(n1,1+n2,-1+n3,n4,n5)*(n2)
    +Z(n1,1+n2,n3,n4,n5)*(-n2)
    +Z(n1,n2,-1+n3,n4,1+n5)*(n5)
    +Z(n1,n2,1+n3,n4,n5)*(n3)
    +Z(n1,n2,n3,-1+n4,1+n5)*(-n5)
    +Z(n1,n2,n3,n4,n5)*(n3-n2);

A5=
```

```
   +Z(-1+n1,1+n2,n3,n4,n5)*(-n2)
   +Z(-1+n1,n2,n3,n4,1+n5)*(n5)
   +Z(n1,-1+n2,1+n3,n4,n5)*(-n3)
   +Z(n1,-1+n2,n3,n4,1+n5)*(-n5)
   +Z(n1,1+n2,n3,n4,-1+n5)*(n2)
   +Z(n1,n2,1+n3,-1+n4,n5)*(-n3)
   +Z(n1,n2,1+n3,n4,-1+n5)*(n3)
   +Z(n1,n2,1+n3,n4,n5)*(n3)
   +Z(n1,n2,n3,n4,n5)*(n5-n2);

A6=
   +Z(-1+n1,n2,n3,n4,1+n5)*(n5)
   +Z(n1,-1+n2,1+n3,n4,n5)*(-n3)
   +Z(n1,-1+n2,n3,n4,1+n5)*(-n5)
   +Z(n1,n2,1+n3,n4,n5)*(n3)
   +Z(n1,n2,n3,n4,n5)*(-n5-n3-2*n2+D);
```

This is the 'raw' form of the equations. They do not look like the equation that is derived in the paper by Chetyrkin and Tkachov, because they used p_5 as one of the independent

variables. We can obtain those by taking linear combinations. We see this when we continue the program with:

```
Local A2 = A2-A3;
Local A6 = A6-A5;
Bracket Z;
Print +f A2,A6;
.end

A2=
   +Z(1+n1,-1+n2,n3,n4,n5)*(n1)
   +Z(1+n1,n2,n3,n4,-1+n5)*(-n1)
   +Z(n1,n2,-1+n3,1+n4,n5)*(n4)
   +Z(n1,n2,n3,1+n4,-1+n5)*(-n4)
   +Z(n1,n2,n3,n4,n5)*(-2*n5-n4-n1+D);

A6=
   +Z(-1+n1,1+n2,n3,n4,n5)*(n2)
   +Z(n1,1+n2,n3,n4,-1+n5)*(-n2)
```

```
+Z(n1,n2,1+n3,-1+n4,n5)*(n3)
+Z(n1,n2,1+n3,n4,-1+n5)*(-n3)
+Z(n1,n2,n3,n4,n5)*(-2*n5-n3-n2+D);
```

Now we do have the form of the paper and we can see that if either n_1, n_4, n_5 are integer or n_2, n_3, n_5 are integer, we can always kill one of those lines by repeated application of one of the above equations. In the paper they are explaining also what to do, when n_5 is not integer, because it contained an extra loop that was integrated out and hence now the power of p_5^2 contains an ϵ. In that case there will be one integral remaining: $Z(1, 1, 1, 1, 1 + ep)$. It is a master integral and it can be evaluated by a variety of means. One is in x-space, using Gegenbauer polynomials, but a better way is described in
"The Massless two loop two point function" Isabella Bierenbaum, Stefan Weinzierl, Eur.Phys.J. C32 (2003) 67-78 DOI: 10.1140/epjc/s2003-01389-7
They show how to do such master integrals, even when all lines have extra multiples of ϵ in their power. Here we will assume that all such master integrals 'are known'.

We will be interested in the case in which potentially all lines have a multiple of ϵ in their power, and hence that we cannot bring any of the powers to zero. The reduction to a minimal number of integrals is far from trivial in that case. And of course there is the question: "how many of such integrals are there?"

Let us start by taking our original set of equations and let us add the complexity argument. This we do in a dedicated procedure:

```
#procedure ResetComplexity(N,par)
*
#if ( 'par' == 0 )
 id  Z(x?,x1?,...,x'N'?) = Z(x1,...,x'N');
#elseif ( 'par' == 1 )
 id  Z(x?,x1?,...,x'N'?) = Z(x1,...,x'N');
 id  Z(x1?,...,x'N'?) =
     Z(-(<abs_(n1-x1)>+...+<abs_(n'N'-x'N')>),x1,...,x'N');
#endif
*
#endprocedure
```

If we couple this to our original program we obtain:

```
    A1=
        +Z(-2,-1+n1,n2,n3,1+n4,n5)*(-n4)
        +Z(-2,-1+n1,n2,n3,n4,1+n5)*(-n5)
        +Z(-2,1+n1,n2,n3,-1+n4,n5)*(n1)
        +Z(-2,n1,-1+n2,n3,n4,1+n5)*(n5)
        +Z(-2,n1,n2,-1+n3,n4,1+n5)*(-n5)
        +Z(-2,n1,n2,n3,-1+n4,1+n5)*(n5)
        +Z(-1,1+n1,n2,n3,n4,n5)*(-n1)
        +Z(-1,n1,n2,n3,1+n4,n5)*(n4)
        +Z(0,n1,n2,n3,n4,n5)*(n4-n1);
```

and similar for the other expressions.

If we want to eliminate all parameters that differ by one from one of the n_i, we have a problem, because there are 10 possible deviations in our equations and there are only 6 equations. HOWEVER! We can create any number of new equations by shifting the n_i to $n_i \pm x_i$. Of course this generates also more "bad parameters", but, as you will see, the number of equations increases faster than the number of integrals that we do not like. But before we do this, let us change our system by combining some equations, as we did when we wanted to show the equations of the paper. This time the objective is to keep the equations as short as possible.

```
Local A1 = A1-A2;
Local A3 = A3-A2;
Local A4 = A4-A6;
Local A5 = A5-A6;
Print +f;
Bracket Z;
.end
```

```
A2=
   +Z(-2,-1+n1,n2,n3,1+n4,n5)*(-n4)
```

```
    +Z(-2,-1+n1,n2,n3,n4,1+n5)*(-n5)
    +Z(-2,n1,-1+n2,n3,n4,1+n5)*(n5)
    +Z(-1,n1,n2,n3,1+n4,n5)*(n4)
    +Z(0,n1,n2,n3,n4,n5)*(-n5-n4-2*n1+D);

A6=
    +Z(-2,-1+n1,n2,n3,n4,1+n5)*(n5)
    +Z(-2,n1,-1+n2,1+n3,n4,n5)*(-n3)
    +Z(-2,n1,-1+n2,n3,n4,1+n5)*(-n5)
    +Z(-1,n1,n2,1+n3,n4,n5)*(n3)
    +Z(0,n1,n2,n3,n4,n5)*(-n5-n3-2*n2+D);

A1=
    +Z(-2,1+n1,n2,n3,-1+n4,n5)*(n1)
    +Z(-2,n1,n2,-1+n3,n4,1+n5)*(-n5)
    +Z(-2,n1,n2,n3,-1+n4,1+n5)*(n5)
    +Z(-1,1+n1,n2,n3,n4,n5)*(-n1)
    +Z(0,n1,n2,n3,n4,n5)*(n5+2*n4+n1-D);
```

```
A3=
    +Z(-2,1+n1,-1+n2,n3,n4,n5)*(-n1)
    +Z(-2,1+n1,n2,n3,n4,-1+n5)*(n1)
    +Z(-2,n1,n2,-1+n3,1+n4,n5)*(-n4)
    +Z(-2,n1,n2,n3,1+n4,-1+n5)*(n4)
    +Z(0,n1,n2,n3,n4,n5)*(2*n5+n4+n1-D);

A4=
    +Z(-2,n1,1+n2,-1+n3,n4,n5)*(n2)
    +Z(-2,n1,n2,-1+n3,n4,1+n5)*(n5)
    +Z(-2,n1,n2,n3,-1+n4,1+n5)*(-n5)
    +Z(-1,n1,1+n2,n3,n4,n5)*(-n2)
    +Z(0,n1,n2,n3,n4,n5)*(n5+2*n3+n2-D);

A5=
    +Z(-2,-1+n1,1+n2,n3,n4,n5)*(-n2)
    +Z(-2,n1,1+n2,n3,n4,-1+n5)*(n2)
```

```
+Z(-2,n1,n2,1+n3,-1+n4,n5)*(-n3)
+Z(-2,n1,n2,1+n3,n4,-1+n5)*(n3)
+Z(0,n1,n2,n3,n4,n5)*(2*n5+n3+n2-D);
```

This way, each equation has only 5 terms. This step can be automated, but for now we will not do this, because we would run out of time.

Another 'improvement' in which we lower the number of terms is a feature that is called the PolyRatFun. In our manipulations that will follow we try to concentrate on the integrals, and we consider the contents of the brackets more or less as a fancy coefficient. These coefficients are essentially multivariate polynomials, and because we will eventually also have such polynomials in the denominator, we will have to allow for rational multivariate polynomials as coefficients. This is done by declaring a function (I usually call it rat) as polyratfun as in:

```
Symbol x,y,n;
CFunction rat;
Off Statistics;
Format nospaces;
Local F = (x+y+2)^3+y*(x+2*y+3)^2-1;
Bracket y;
Print;
.sort

F=
   +y*(21+18*x+4*x^2)
   +y^2*(18+7*x)
```

```
      +y^3*(5)
      +7+12*x+6*x^2+x^3;
```

We begin with some typical polynomial.

```
    id  x^n? = rat(x^n,1);
    Bracket y;
    Print;
    .sort
```

```
  F=
      +y*(4*rat(x^2,1)+18*rat(x,1)+21*rat(1,1))
      +y^2*(7*rat(x,1)+18*rat(1,1))
      +y^3*(5*rat(1,1))
      +6*rat(x^2,1)+rat(x^3,1)+12*rat(x,1)+7*rat(1,1);
```

After introducing the function rat, there is still nothing special.

```
    PolyRatFun rat;
    Print +s;
    .sort
```

```
F=
    +y*rat(4*x^2+18*x+21,1)
    +y^2*rat(7*x+18,1)
    +y^3*rat(5,1)
    +rat(x^3+6*x^2+12*x+7,1)
    ;
```

Once we declare rat to be the polyratfun, we see that all the terms are combined, ie. terms that are otherwise identical have their coefficients added. In the function rat, the first argument is supposed to be the numerator and the second the denominator of a rational multivariate polynomial. At all times this polynomial is to be kept normalized.

```
    id  y^n? = y^n*rat(1,x+1+n);
    Print +s;
    .sort

F=
    +y*rat(4*x^2+18*x+21,x+2)
    +y^2*rat(7*x+18,x+3)
```

```
       +y^3*rat(5,x+4)
       +rat(x^2+5*x+7,1)
       ;
```

Here we see that in the last term the program saw that the $x + 1$ in the denominator could be divided out.

```
    id  y = 3;
    Print +s;
    .end
```

```
   F=
      +rat(x^5+26*x^4+279*x^3+1477*x^2+3542*x+3030,x^3+9*x^2+26*x+24);
```

When possible the rational polynomials are added, just as with numerical coefficients.

This looks almost too good to be true. And indeed it is. There are restrictions on the size of information inside a single term. This places also restrictions on how big the polynomials inside a function like rat can be. In addition, calculus with multivariate rational polynomials can become very slow when these polynomials have many terms and many variables. The big culprit here is the gcd calculation: it is even worse than for numbers. But we have no choice.

Just remember: there are limitations and hence we should make special efforts to keep the fractions as simple as possible. This holds for all computer algebra systems.

So now we add the code to put our coefficients together. We also omit the complexity for now. We obtain

```
CFunction acc, rat;
PolyRatFun rat;
Collect acc;
id  acc(x?) = rat(x,1);
Print +s;
.end

 A2=
   +Z(-1+n1,n2,n3,1+n4,n5)*rat(-n4,1)
   +Z(-1+n1,n2,n3,n4,1+n5)*rat(-n5,1)
   +Z(n1,-1+n2,n3,n4,1+n5)*rat(n5,1)
   +Z(n1,n2,n3,1+n4,n5)*rat(n4,1)
   +Z(n1,n2,n3,n4,n5)*rat(D-2*n1-n4-n5,1)
   ;
```

```
A6=
    +Z(-1+n1,n2,n3,n4,1+n5)*rat(n5,1)
    +Z(n1,-1+n2,1+n3,n4,n5)*rat(-n3,1)
    +Z(n1,-1+n2,n3,n4,1+n5)*rat(-n5,1)
    +Z(n1,n2,1+n3,n4,n5)*rat(n3,1)
    +Z(n1,n2,n3,n4,n5)*rat(D-2*n2-n3-n5,1)
    ;

A1=
    +Z(1+n1,n2,n3,-1+n4,n5)*rat(n1,1)
    +Z(1+n1,n2,n3,n4,n5)*rat(-n1,1)
    +Z(n1,n2,-1+n3,n4,1+n5)*rat(-n5,1)
    +Z(n1,n2,n3,-1+n4,1+n5)*rat(n5,1)
    +Z(n1,n2,n3,n4,n5)*rat(-D+n1+2*n4+n5,1)
    ;

A3=
```

```
   +Z(1+n1,-1+n2,n3,n4,n5)*rat(-n1,1)
   +Z(1+n1,n2,n3,n4,-1+n5)*rat(n1,1)
   +Z(n1,n2,-1+n3,1+n4,n5)*rat(-n4,1)
   +Z(n1,n2,n3,1+n4,-1+n5)*rat(n4,1)
   +Z(n1,n2,n3,n4,n5)*rat(-D+n1+n4+2*n5,1)
   ;

A4=
   +Z(n1,1+n2,-1+n3,n4,n5)*rat(n2,1)
   +Z(n1,1+n2,n3,n4,n5)*rat(-n2,1)
   +Z(n1,n2,-1+n3,n4,1+n5)*rat(n5,1)
   +Z(n1,n2,n3,-1+n4,1+n5)*rat(-n5,1)
   +Z(n1,n2,n3,n4,n5)*rat(-D+n2+2*n3+n5,1)
   ;

A5=
   +Z(-1+n1,1+n2,n3,n4,n5)*rat(-n2,1)
   +Z(n1,1+n2,n3,n4,-1+n5)*rat(n2,1)
```

```
      +Z(n1,n2,1+n3,-1+n4,n5)*rat(-n3,1)
      +Z(n1,n2,1+n3,n4,-1+n5)*rat(n3,1)
      +Z(n1,n2,n3,n4,n5)*rat(-D+n2+n3+2*n5,1)
      ;
```

At this point we are ready to generate more equations. How are we going to do this? And how many are we going to generate. Let us start with defining a parameter that says how much extra complexity we are allowed to add. After this we have to divide this over the five arguments and we might either raise or lower such a parameter. Here is a very simple program to do this:

```
    #define INCCOMPLEXITY "4"
    CF  inc,inc1;
    Symbol x1,...,x5;
    Local FC =
    #do i1 = 0,'INCCOMPLEXITY'
    #do i2 = 0,'INCCOMPLEXITY'-'i1'
    #do i3 = 0,'INCCOMPLEXITY'-'i1'-'i2'
    #do i4 = 0,'INCCOMPLEXITY'-'i1'-'i2'-'i3'
    #do i5 = 0,'INCCOMPLEXITY'-'i1'-'i2'-'i3'-'i4'
```

```
            +inc1('i1','i2','i3','i4','i5')
    #enddo
    #enddo
    #enddo
    #enddo
    #enddo
        ;
    Multiply inc;
    repeat id inc(?a)*inc1(x1?,?b) = (inc(?a,x1)+inc(?a,-x1))*inc1(?b);
    id  inc1 = 1;
    .sort

Time =         0.00 sec     Generated terms =        4032
            FC                 Terms in output =         681
                               Bytes used      =       20360

    DropCoefficient;
    Print +f +s;
    .end
```

```
Time =           0.01 sec     Generated terms =          681
          FC                  Terms in output =          681
                             Bytes used       =        20360

   FC =
       + inc(-4,0,0,0,0)
       + inc(-3,-1,0,0,0)
       + inc(-3,0,-1,0,0)
              etc.
```

Note that some terms will be generated twice because when there is a zero the 'doubling' is not really necessary. Each of the terms in this expression will generate us a new system of six equations. Hence if we allow the complexity to be increased by up to 4 we will end up with 4086 equations! The table shows this increment even better.

INCCOMPLEXITY	systems	equations	reduced equations
1	11	66	66
2	61	366	186
3	231	1386	966
4	681	4086	1626
5	1683	10098	5646
6	3653	21918	7686

When we run our programs (this is hindsight, but it will save us much time running our examples), we do not consider cases for which more than half the increased complexity (rounded up) comes from positive contributions, or more than half (rounded up) comes from negative contributions. This is indicated in the last column.

Let us however start a bit more modest with INCCOMPLEXITY at 2. How do we combine this with our program? We put it together as follows:

```
#define INCCOMPLEXITY "2"
Symbols D,ep,n1,n2,n3,n4,n5;
Dimension D;
Symbols x,x1,x2,x3,x4,x5;
Vector Q,p,p1,p2,p3,p4,p5;
```

```
CFunction der,del,Z,Y,inc,inc1;
Off Statistics;
Format nospaces;
.global
#define N "5"
#call IBPforT1
.sort
Local A1 = A1-A2;
Local A3 = A3-A2;
Local A4 = A4-A6;
Local A5 = A5-A6;
Bracket Z;
.sort
CFunction acc, rat;
PolyRatFun rat;
Collect acc;
id    acc(x?) = rat(x,1);
.sort
```

```
Hide;        * <-------------------
Local FC =
#do i1 = 0,'INCCOMPLEXITY'
#do i2 = 0,'INCCOMPLEXITY'-'i1'
#do i3 = 0,'INCCOMPLEXITY'-'i1'-'i2'
#do i4 = 0,'INCCOMPLEXITY'-'i1'-'i2'-'i3'
#do i5 = 0,'INCCOMPLEXITY'-'i1'-'i2'-'i3'-'i4'
 +inc1('i1','i2','i3','i4','i5')
#enddo
#enddo
#enddo
#enddo
#enddo
 ;
Multiply inc;
repeat id inc(?a)*inc1(x1?,?b) = (inc(?a,x1)+inc(?a,-x1))*inc1(?b);
id   inc1 = 1;
if ( match(inc(?a,x?{2,-2},?b)) ) Discard;
```

```
if ( match(inc(?a,1,?b,1,?c)) ) Discard;
if ( match(inc(?a,-1,?b,-1,?c)) ) Discard;
.sort
DropCoefficient;
.sort
Hide;          * <--------------------
#$numeq = 0;
#do oneterm = FC
Global G{'$numeq'+1} = A1*'oneterm';
Global G{'$numeq'+2} = A2*'oneterm';
Global G{'$numeq'+3} = A3*'oneterm';
Global G{'$numeq'+4} = A4*'oneterm';
Global G{'$numeq'+5} = A5*'oneterm';
Global G{'$numeq'+6} = A6*'oneterm';
#$numeq = $numeq+6;
#enddo
id   inc(x1?,...,x5?) = replace_(<n1,n1+x1>,...,<n5,n5+x5>);
#call ResetComplexity('N',1)
```

```
   Multiply replace_(D,4-2*ep);
   Print +s;
   .end
```

We may notice the Hide statements and the funny new element here is the do loop which now goes over the terms in the expression FC. This means that the do loop parameter oneterm will contain a textual representation of one of the terms of FC, each time we go through the loop. The variable $numeq is a counter that labels the equations and in the end tells us the number of equations.

We will keep these G equations and remove the others. Our task is to eliminate the undesirable integrals and see what remains at complexity one or zero. We want to do this with a gaussian elimination scheme, but one possible complicating factor may be that the rational polynomial coefficients may become very complicated. In addition it may well be that there will be coefficients that are zero for particular values of the parameters n_i. Let us just start the naive way. The highest complexity integrals in the system have complexity INCCOMPLEXITY+2. We will make a procedure that eliminates all terms of a given complexity. Once we have such a procedure we can work our way down and see whether that gives us a 'proper solution'.

Let us start with making a simple gaussian elimination routine:

```
#procedure gauss0(NUM,C)
*
*     Procedure takes a system of NUM equations named G1,...,G'NUM'.
*     Eliminates all integrals of complexity C.
*     We use the most trivial variety of a Gaussian elimination
*
#do i = 1,'NUM'
  #if ( termsin(G'i') > 0 )
    #$lhs = firstterm_(G'i');
    #$match = 0;
    #inside $lhs
        if ( match(Z(-'C',?a)) );
            id  rat(x1?$x1,x2?$x2) = 1;
            $match = 1;
        endif;
    #endinside
    #if ( '$match' > 0 )
        #$rhs = -(G'i'*rat($x2,$x1)-$lhs);
```

```
            id  '$lhs' = $rhs;
            .sort
        #endif
      #endif
    #enddo
    #endprocedure
```

We run through the equations, and if there is one with an integral of the proper complexity (this is why we count the complexity negative. It makes the highest complexity to end up in the first term) we replace this integral in all equations.

We add the following code to our program to make the restriction on the various complexity combinations (the three if statements):

```
repeat id inc(?a)*inc1(x1?,?b) = (inc(?a,x1)+inc(?a,-x1))*inc1(?b);
id   inc1 = 1;
if ( match(inc(?a,x?{2,-2},?b)) ) Discard;
if ( match(inc(?a,1,?b,1,?c)) ) Discard;
if ( match(inc(?a,-1,?b,-1,?c)) ) Discard;
.sort
DropCoefficient;
```

and the elimination code

```
Multiply replace_(D,4-2*ep);
.sort
#do c = 'INCCOMPLEXITY'+2,2,-1
  #call gauss0('$numeq','c')
#enddo
#do i = 1,'$numeq'
#if ( termsin(G'i') > 0 )
```

```
    Print +s G'i';
  #endif
  #enddo
  .end
```

When we run this program we run into a crash. The program ends with a rather annoying message:

```
ERROR: PolyRatFun doesn't fit in a term
(1) num size = 35956, den size = 7336,  MaxTer = 160000
Program terminating at gauss1 Line 25 -->
  3.02 sec out of 3.05 sec
```

This message indicates that the maximum size for a term is not large enough to contain the rational polynomial that should be in that term. We can try the easiest solution, which is to increase the maximum term size, although that will put a heavy demand on the memory management of the computer. We do this by adding the following line to the program as its first line:

```
  #: MaxTermSize 1M
```

Such instructions that start with #: have to be at the start of the program and are read during the startup of the program, before FORM makes its memory allocations. With this term size the program can indeed finish in a bit more than 5 sec. The output consists of 33 equations, each with integrals of at most complexity one. At first sight this looks great, because there are only 10 integrals of complexity one (and one of complexity zero). Let us study this a bit better though and add a few lines to our Gauss procedure to see what it is doing:

```
#if ( '$match' > 0 )
    #$rhs = -(G'i'*rat($x2,$x1)-$lhs);
    id  '$lhs' = $rhs;
    #$factors = acc($x1);
    #inside $factors
        FactArg,acc;
        ChainOut acc;
        id  acc(x?number_) = 1;
        id  acc(x?symbol_) = x;
    #endinside
    #write "Taking out %$, factors = %$",$lhs,$factors
```

```
        .sort
    #endif
```

The extra lines make the program show the numerator of the term we are replacing. We can use this to see whether we may be accidentally dividing by zero. This could occur with a factor like $n_i - n_j$.

The output is now much more verbose with lines like:

```
    Taking out Z(-2,-1+n1,n2,n3,1+n4,n5), factors = acc(-3+2*n4+n1+2*ep)
    Taking out Z(-2,-1+n1,1+n2,n3,n4,n5), factors = acc(-1+n1)
        .....
```

The factors here are safe: we assume that n_1 contains an integer plus a positive integer times ϵ. In the case n_1 is an integer the second factor could be zero, but that would be for an integral with an argument that is zero, and those integrals can be solved directly (see the paper by Chetyrkin and Tkachov). What is a problem however is that only 22 integrals of complexity two are removed, while counting reveals that there exist 50 of such integrals. And indeed, if we run the program down to integrals of complexity one we obtain that only 7 integrals of complexity one are removed, and hence there are four integrals that are undetermined.

The next attempt is to remove the three if-statements that we added to restrict the number of equations and use all 366 equations. This makes the program much slower (nearly 30 sec)

and now 42 integrals of complexity two are replaced. When we reduce also the complexity one integrals we get a crash again. To resolve this crash we have to know a bit more about FORM.

We saw in lecture one that FORM uses a hierarchy of buffers to sort expressions. But what happens when it has to sort function arguments? For this it has similar buffers, but those are much smaller than the ones for complete expressions. Typically they are no bigger than 100 Kbytes. Fortunately we can also configure these buffers. This is much needed here because the arithmetic of the arguments creates very many terms, even though afterwards they fit inside a single term (inside the rat function). Hence the first lines of the program become now

```
#: MaxTermSize 1M
#: SubTermsInSmall 1M
#: SubSmallSize 100M
#: SubLargeSize 1000M
```

Again, this puts some more demands on the memory of the computer, which is a laptop with 16 Gbytes. But now the program runs. Unfortunately again there are only 7 reductions of complexity one integrals.

Why is this bad? Well, if we cannot reduce all complexity one integrals, we would at least like to reduce all complexity two integrals. Hence we need to raise INCCOMPLEXITY to get

more equations and see whether that will satisfy us. For this we decide to raise MaxTermSize to 10 Mbytes after which the system complains that its WorkSpace is not big enough. This workspace is a stack onto which terms are placed during expansion. Hence if MaxTermSize is very big its regular size may not be enough. Hence we also raise that one. This leaves one more complication: internally FORM has a maximum size for numbers. When we do not use a polyratfun and have only numerical fractions, this is automatically set to about half the maximum size for terms. In the case that we have a polyratfun with lots of terms inside the polyratfun, this is particularly devastating, because each of these terms gets a buffer during factorizations, and FORM would start allocating enormous amounts of memory. Hence we force the size of such numbers to a more realistic value.

```
#: MaxTermSize 10M
#: WorkSpace 1000M
#: SubTermsInSmall 1M
#: SubSmallSize 100M
#: SubLargeSize 1000M
#: MaxNumberSize 50K
```

All these parameters and more are explained in the manual in the chapter about the setup. Now things proceed better, but the program becomes very slow. And we see some horrible

factors coming by on the screen with up to 30 lines. This may not be the way to proceed.

We have to become a bit more sophisticated, trying to make substitutions based on the simplest term with a given complexity, rather than the first we see. This we do with the next procedure:

```
#procedure gauss2(NUM,C)
*
*     Procedure takes a system of NUM equations named G1,...,G'NUM'.
*     Eliminates all integrals of complexity C.
*     Second attempt in the Gauss elimination: we take the term
*     with weight C but with the simplest coefficient.
*
#do i = 1,'NUM'
#if ( termsin(G'i') > 0 )
 #$expr = G'i';
 #inside $expr
     if ( match(Z(-'C',?a)) );
         id  Z(-'C',?a)*rat(x1?,x2?) = Z(-'C',
             nterms_(x1)+nterms_(x2),nterms_(x1),?a)*rat(x1,x2);
     endif;
 #endinside
*
*     Having sorted $expr the 'simplest' term is now first
```

```
*
 #$lhs = firstterm_($expr);
 #$match = 0;
 #inside $lhs
     if ( match(Z(-'C',?a)) );
         id  Z(x1?,x2?,x3?,n1?,...,n5?) = Z(x1,n1,...,n5);
         id  rat(x1?$x1,x2?$x2) = 1;
         $match = 1;
     endif;
 #endinside
 #if ( '$match' > 0 )
     #$rhs = -($expr*rat($x2,$x1)-$lhs);
     #inside $rhs
         id  Z(x1?,x2?,x3?,n1?,...,n5?) = Z(x1,n1,...,n5);
     #endinside
     id  '$lhs' = $rhs;
*     here goes the inspection code.........
     .sort
```

```
  #endif
 #endif
 #enddo
 #endprocedure
```

Even though this improves things, it is still not sufficient. There are some really bad terms here. Hence we have to think again.

The next thing we realize is that we take the first equation we run into with an integral of the proper complexity. This can be very bad, because there are very simple equations, and there are very complicated equations. We should inspect each time which equation is the simplest, and which integral in that equation is the one that should be substituted. Such a program will however be much more complicated. A second improvement would be to check whether an equation has an overal factor (this is called the content of an expression) and divide this out (provided it cannot be zero). This will not only help in selecting which equation is the 'simplest', but also make the substitutions in the other equations easier. Hence we have to construct two procedures: gauss3.prc and takefactor.prc which checks whether we can divide out a factor (and then divides the equation by it).

Let us start with the takefactor procedure. It reads:

```
#procedure takefactor(G)
*
*     We try to divide out overal factors.
*     The set of equations is given in the macro TODO
*     The equations are marked as 'G'<number>
*
PolyRatFun;
id    rat(x1?,x2?) = num(x1)*den(x2);
FactArg,num;
FactArg,den;
ChainOut num;
ChainOut den;
id    num(x?number_) = x;
id    den(x?number_) = 1/x;
*
.sort
*
*     The content_ function can handle only symbols. Hence:
```

```
*
ToPolynomial;
.sort
#do i = 'TODO'
 #if ( '$exists'i'' > 0 )
 #if ( termsin('G''i') > 0 )
  #$content'i' = content_('G''i');
*
*  Exchange num and den. First get rid of the extra symbols.
*
  #inside $content'i'
    FromPolynomial;
    $ccc = 1/coeff_;
    Multiply replace_(num,den,den,num)*$ccc^2;
  #endinside
  #write <> " content'i' = %$",$content'i'
  if ( expression('G''i') ) Multiply '$content'i'';
 #endif
```

```
  #endif
#enddo
.sort
PolyRatFun rat,RAT;
*
*  Back to normal. Get rid of the extra symbols in the expression.
*
FromPolynomial;
id   num(x?)*den(x?) = 1;
id   num(x?) = rat(x,1);
id   den(x?) = rat(1,x);
.sort
#endprocedure
```

We see here a number of new things. FactArg and ChainOut we have seen in lecture 5. To refresh your mind: FactArg,num; factorizes the argument(s) of the function num. Each factor will now be a separate argument. ChainOut, num; then takes the arguments of num and puts each of them in a separate occurrence of function num. This way we can handle them better. The PolyRatFun,rat,RAT; is a variety of the polyratfun statement in which we

declare a second function to be 1/rat. This is handy when manipulating formula's by hand as when changing

```
Z(....)*rat(xx1,xx2)+Z(...)*rat(yy1,yy2)+....;
```

into

```
id Z(....) = -RAT(xx1,xx2)*(+Z(...)*rat(yy1,yy2)+....);
```

This way one does not have to touch the arguments of the rat function, which could be really messy when they are rather lengthy.

The '$exists'i" is explained a little further in the text.

Next we want to obtain the content of each equation. This is basically the GCD of all terms in the equation, but the problem is that at the moment of creating these programs the content_ function could not do this for functions (has been repaired as of 6-mar-2018). There is however a solution: we can replace all nontrivial objects by program-generated symbols, called extra symbols. Their names will end in an underscore character. Their default names are Z1_, Z2_, etc. The start string of these names can be changed by the user (see manual). We invoked this change with the ToPolynomial statement. There is one extra complication: because FactArg also works only over the symbols, it uses the extra symbol system and hence

the two cannot be in the same module (would generate an error message). At a later point we can return to the original notation with the FromPolynomial statement.

```
Symbols a,b,c;
CFunction f1,f2,f3;
Off Statistics;
Format nospaces;
Local F = f1(a)+f1(a+b)*f2(c)*(b+c)+f3(f1(a),f2(b),f3(c));
ToPolynomial;
Print;
.sort

F=
   Z1_+Z4_+c*Z3_*Z2_+b*Z3_*Z2_;
#write "\n   %X"

Z1_=f1(a);
Z2_=f1(b+a);
Z3_=f2(c);
```

```
Z4_=f3(f1(a),f2(b),f3(c));

.sort
ExtraSymbols,array,Y;
#write "\n    %X"

Y(1)=f1(a);
Y(2)=f1(b+a);
Y(3)=f2(c);
Y(4)=f3(f1(a),f2(b),f3(c));

Print;
.sort

F=
   Y(1)+Y(4)+c*Y(3)*Y(2)+b*Y(3)*Y(2);
FromPolynomial;
Print;
```

```
    .end

  F=
    f1(b+a)*f2(c)*c+f1(b+a)*f2(c)*b+f1(a)+f3(f1(a),f2(b),f3(c));
```

The extra symbols are also very useful when trying to optimize output formulas, to make them contain fewer multiplications and additions. For this you should see the chapter on optimizations in the manual. It is a rather advanced and unique feature of FORM which led to several publications.

We can now pick up the content of the various expressions and convert it into 1/content. This is rather simple. After that we multiply each expression by their inverted content and we are done. We write the various contents to the log file to allow inspection for potential division by zero.

Next is a new gauss procedure that at each step searches for the expression that is simplest. For this we will of course need to define what we mean by simplest. Before we study the procedure, we make some changes in the program. We define an 'array' of dollar variables $exists1 to $exists'$numeq' that tells whether an equation still exists. This way we can quickly check whether we have to consider that equation and we can eliminate the equations that have become zero. After that a print statement will not print endless lines with equations that are

relevant no longer. It needs a supporting procedure:

```
#procedure Drop(m)
#$exists'm' = 0;
Drop G'm';
#endprocedure
```

In addition we make the gaussian procedure slightly more general by adding a parameter that indicates the 'array' of expressions (which is G in our case). This is to make the procedure more general for later. Finally we use a preprocessor variable TODO to indicate which equations have to be summed over in a #do loop. In our current program that variable is initialized as

```
#define TODO "1,'$numeq'"
```

but when we are experimenting we could replace it by a list of numbers as in

```
#define TODO "{5,11,56,965}"
```

```
#procedure gauss3(G,c)
*
*    This procedure does a Gaussian elimination for terms
*    with complexity 'c'.
*    The equations containing the elements that are
*    eliminated are lost.
*    We assume that there are $numeq equations,
*    named 'G'1,...,'G''$numeq'.
*    There is also an array $exists1,...,$exists'$numeq'
*    that tells whether an equation still exists.
*
.sort
#message Procedure gauss3 dealing with complexity 'c'
*
*    Suborder Z by the number of terms in rat.
*
if ( match(Z(-'c',?a)) );
    id  Z(-'c',?a)*rat(x1?,x2?) = Z(-'c',
```

```
                       nterms_(x1)+nterms_(x2),?a)*rat(x1,x2);
endif;
B   Z;
.sort
#$numc'c' = 0;
#do j = 1,1
  #if ( 'c' <= 2 )
    #$numc'c' = $numc'c'+1;
    #message $numc'c' = '$numc'c''
  #endif
*
*    Start with counting:
*    The number of terms with leading complexity in $nn
*    The number of terms with subleading complexity in $ns
*    The number of terms in rat in the whole expression in $nc
*    The number of terms in rat in the first term in $nc
*    The total number of terms of the expression in $nt
*
```

```
#do i = 'TODO'
  #if ( '$exists'i'' )
    #$nn'i' = 0;
    #$ns'i' = 0;
    #$nx'i' = 0;
    #$nt'i' = termsin_('G''i');
    #$nc'i' = 0;
  #endif
#enddo
if ( match(Z(-'c',?a)) );
  #do i = 'TODO'
    #if ( '$exists'i'' )
      if ( expression('G''i') );
          $nn'i' = $nn'i'+1;
          id  rat(x1?$x1,x2?$x2) = rat(x1,x2);
          if ( $nn'i' == 1 );
              id  rat(x1?$x1,x2?$x2) = rat(x1,x2);
              $nx'i' = termsin_($x1)+termsin_($x2);
```

```
          endif;
        endif;
      #endif
    #enddo
elseif ( match(Z(-‘c’+1,?a)) );
  #do i = ‘TODO’
    #if ( ‘$exists‘i’’ )
      if ( expression(‘G’‘i’) ) $ns‘i’ = $ns‘i’+1;
    #endif
  #enddo
endif;
#do i = ‘TODO’
  #if ( ‘$exists‘i’’ )
    if ( expression(‘G’‘i’) );
      id  rat(x1?$x1,x2?$x2) = rat(x1,x2);
      $nc‘i’ = $nc‘i’+ termsin_($x1)+termsin_($x2);
    endif;
  #endif
```

```
      #enddo
      B    Z;
      .sort
*
*    Get now the totals of all our variables.
*
      #$nnsum = 1;
      #$nssum = 1;
      #$ntsum = 1;
      #$nxsum = 1;
      #do i = 'TODO'
        #if ( '$exists'i'' )
          #$nnsum = $nnsum+'$nn'i'';
          #$nssum = $nssum+'$ns'i'';
          #$ntsum = $ntsum+'$nt'i'';
          #$nxsum = $nxsum+'$nx'i'';
        #endif
      #enddo
```

```
  #if ( '$nnsum' > 1 )
    #$imin = 0;
*
*   Starting value that is bigger than any weight
*
    #$nmin = 1000*'$nnsum'+40*'$nssum'+'$ntsum'\
                          +40*'$nxsum'+1000000;
    #do i = 'TODO'
     #if ( '$exists'i'' )
      #if ( '$nn'i'' > 0 )
*
*        Finally the weight function
*
        #$n = 100*'$nn'i''+4*'$ns'i''+'$nt'i''\
                          +4*'$nx'i''+'$nc'i'';
        #if ( '$n' < '$nmin' )
            #$imin = 'i';
            #$nmin = $n;
```

```
       #endif
     #endif
    #endif
   #enddo
*
*   At this point we see $imin as optimal.
*

   #if ( 'c' ==  2 )
      Print +f +s 'G''$imin';
      B   Z;
       .sort
   #endif
   #$eq = 'G''$imin';
*

   #$lhs = FirstBracket_('G''$imin'); * object to replace
   #$lhsc = 'G''$imin'['$lhs'];       * coefficient
   #$xxx = $lhs*$lhsc-('G''$imin');
   #inside $lhsc;
```

```
        id     rat(x1?$x1,x2?$x2) = rat(x2,x1);
    #endinside;
    #write <> "Eliminating '$imin' (('$nn'$imin'','$ns'$imin''\
                     ,'$nt'$imin'','$nx'$imin'','$nc'$imin'')))"
    #$xxx = $xxx*$lhsc;        * the right hand side
*
*   PolyRatFun does not work inside dollars.
*   Manually simplify the rat functions a bit.
*
    #inside $xxx
        id  Z(-'c',x?,n1?,...,n'N'?) = Z(-'c',n1,...,n'N');
        id  rat(x1?,x2?)*rat(x3?,x4?) = rat1(x1,x4)*rat2(x3,x2);
        id  rat1(x1?,x2?) = rat1(x1,x2,gcd_(x1,x2));
        id  rat2(x1?,x2?) = rat2(x1,x2,gcd_(x1,x2));
        id  rat1(x1?,x2?,x3?) = rat(div_(x1,x3),div_(x2,x3));
        id  rat2(x1?,x2?,x3?) = rat(div_(x1,x3),div_(x2,x3));
        id  rat(x1?,x2?)*rat(x3?,x4?) = rat(x1*x3,x2*x4);
    #endinside
```

```
      #call Drop('$imin')
*
*     Back to the regular form of the Z.
*     Then we write to the .log file what we will replace.
*

      id  Z(-'c',x?,n1?,...,n'N'?) = Z(-'c',n1,...,n'N');
      #write "id  '$lhs' ="
      #inside $lhs;
        id  Z(-'c',x?,n1?,...,n'N'?) = Z(-'c',n1,...,n'N');
      #endinside;
*
*     And this is the statement that eliminates this integral.
*

      id  '$lhs' = '$xxx';
      .sort
*
*     Now knock out expressions that become zero accidentally.
*
```

```
    #do ijj = 'TODO'
      #if ( '$exists'ijj'' != 0 )
        #if ( termsin('G''ijj') == 0 )
           #call Drop('ijj')
        #endif
      #endif
    #enddo
    #if ( 'c' <= 2 )
      #call takefactor('G')
    #endif
    id  Z(-'c',?a)*rat(x1?,x2?) = Z(-'c',
                 nterms_(x1)+nterms_(x2),?a)*rat(x1,x2);
    B   Z;
    .sort
    #redefine j "0"
  #endif
#enddo
#endprocedure
```

These procedures allow us to go to larger values of INCCOMPLEXITY. When we run this with INCCOMPLEXITY $= 3$ we find reduction equations for all 50 integrals of complexity 2. This means that any integral of complexity 2 or higher can be broken down to integrals of complexity one and zero. Unfortunately there are only 8 reduction equations for complexity one integrals. This means that there are 3 objects that we cannot determine with IBP equations. Hence in the case that all powers are noninteger we must conclude that there must be three master integrals.

Let us assume that the three remaining integrals are

```
Z(-1,-1+n1,n2,n3,n4,n5)
Z(-1,n1,n2,n3,-1+n4,n5)
Z(0,n1,n2,n3,n4,n5)
```

which means that eventually all integrals can be reduced to

```
Z(-1,0+m1*ep,1+m2*ep,1+m3*ep,1+m4*ep,1+m5*ep)
Z(-1,1+m1*ep,1+m2*ep,1+m3*ep,0+m4*ep,1+m5*ep)
Z( 0,1+m1*ep,1+m2*ep,1+m3*ep,1+m4*ep,1+m5*ep)
```

and in the case that some of the m_i are equal the symmetry of the diagram could make that there are at most two master integrals.

This means that for a given set of m_i there are at most three master integrals and the set of 50 equations for complexity two integrals and the 8 equations for complexity one are sufficient to reduce any integral of this type to its master integrals.

Actually we can do a bit better. With the above master integrals we only need a subset of the complexity two reduction equations: Those that contain $n_1 - 1, n_4 - 1, n_1 - 2$ or $n_4 - 2$, which makes a total of 17 equations. This means that any complete reduction scheme needs $17 + 8 = 25$ equations.

For the calculation of the master integrals one could use the methods in the paper by Bierenbaum and Weinzierl.

The above reduction method is called parametric reduction. We can use it to create a procedure as in

```
#procedure reduceT1
*
*   Procedure automatically created Tue Mar  6 02:11:58 2018
*
#$numloop = 0;
#do inumloop = 1,1
#$numloop = $numloop+1;
#$loopaction = 0;
```

```
id,ifmatch->looplabel,Z(n1?,m1?,n2?,m2?,n3?{>1},m3?,n4?neg0_,m4?,n5?,m5?)
  =Z(-1+n1,m1,n2,m2,-1+n3,m3,1+n4,m4,n5,m5)*num(n4+ep*m4)*num(-6+n5+n4+n3+n2+n1+
  3*ep+ep*m5+ep*m4+ep*m3+ep*m2+ep*m1)*den(-1+n3+ep*m3)*den(-2+n3+ep+ep*m3)-Z(n1,
  m1,n2,m2,-1+n3,m3,1+n4,m4,n5,m5)*num(n4+ep*m4)*num(-5+n5+n3+n2+n1+2*ep+ep*m5+
  ep*m3+ep*m2+ep*m1)*den(-1+n3+ep*m3)*den(-2+n3+ep+ep*m3)+Z(n1,m1,n2,m2,-1+n3,
  m3,n4,m4,n5,m5)*num(-5+n5+n3+n2+n1+2*ep+ep*m5+ep*m3+ep*m2+ep*m1)*num(-6+n5+n4+
  n3+n2+n1+3*ep+ep*m5+ep*m4+ep*m3+ep*m2+ep*m1)*den(-1+n3+ep*m3)*den(-2+n3+ep+ep*
  m3);
id,ifmatch->looplabel,Z(n1?neg0_,m1?,n2?,m2?,n3?,m3?,n4?{>1},m4?,n5?,m5?)
              .....24 more statements (most a bit longer) ....
  ep*m3);
goto looplabel2;
label looplabel;
id  num(x?)*den(x?) = 1;
id  num(x?) = rat(x,1);
id  den(x?) = rat(1,x);
$loopaction = 1;
label looplabel2;
ModuleOption Maximum,$loopaction;
.sort:reduceT1-loop '$numloop';
#if ( '$loopaction' == 1 )
```

```
        #redefine inumloop "0"
  #endif
  #enddo
  *
  #endprocedure
```

We have taken out the complexity parameter and each argument n_i has been written now as two arguments n_i, m_i with n_i integer and m_i a positive integer. Let us see how this works:

```
        Symbols ep,x,n,n1,...,n5,m1,...,m5;
        CFunctions Z,Y,rat,RAT,num,den;
        PolyRatFun rat;
        Format nospaces;
        .global
        Local F = Z(2,1,2,2,2,3,2,4,2,5);
        #call oldreduceT1(0)

  Time =          0.04 sec     Generated terms =          3
               F               Terms in output =          3
          reduceT1-loop 1 Bytes used          =       1744
                                 .

                                 .
```

```
                              .
   Time =          0.24 sec      Generated terms =          3
                   F             Terms in output =          3
         reduceT1-loop 6 Bytes used       =        3868
     id     num(x?) = rat(x,1);
     id     den(x?) = rat(1,x);
     Print +f +s;
     .end

   Time =          0.24 sec      Generated terms =          3
                   F             Terms in output =          3
                                 Bytes used       =        3868


     F=
        +Z(0,1,1,2,1,3,1,4,1,5)*rat(2605522570858994688*ep^15+
        5733124412457058560*ep^14+5434061766013820736*ep^13+
        2933159127648654960*ep^12+996429020871290448*ep^11+
        21970099369597072*ep^10+30474403711289040*ep^9+2187759395822333*
        ep^8-34031776903998*ep^7-23713496692257*ep^6-2182471844556*ep^5-
        68940490489*ep^4+2676851682*ep^3+312393745*ep^2+10480200*ep+
        127116,32961195540000*ep^15+129399858315000*ep^14+22442151745700
```

```
*ep^13+230148131079510*ep^12+156992002169205*ep^11+75753182355795
*ep^10+26785824281580*ep^9+7081231070340*ep^8+1413094602990*ep^7+
213064001010*ep^6+24090692640*ep^5+2006803890*ep^4+119252805*ep^3
+4773195*ep^2+115080*ep+1260)
+Z(1,1,1,2,1,3,0,4,1,5)*rat(7752013764359546880*ep^15+
15950087750181139200*ep^14+14377035448750008960*ep^13+
7474636263030531408*ep^12+2470132996699900656*ep^11+
534012149452956832*ep^10+73086003770458548*ep^9+5199933433250555*
ep^8-82580941868658*ep^7-56616880594755*ep^6-5243831418792*ep^5-
169480601095*ep^4+6223006662*ep^3+755399563*ep^2+25839024*ep+
319572,131844782160000*ep^15+517599433260000*ep^14+
897684606982800*ep^13+920592524318040*ep^12+627968008676820*ep^11
+303012729423180*ep^10+107143297126320*ep^9+28324924281360*ep^8+
5652378411960*ep^7+852256004040*ep^6+96362770560*ep^5+8027215560*
ep^4+477011220*ep^3+19092780*ep^2+460320*ep+5040)
+Z(1,1,1,2,1,3,1,4,1,5)*rat(-1018482066734444928*ep^15-
2213760426861637152*ep^14-2151247814777837088*ep^13-
1239313907404952520*ep^12-472671287088809324*ep^11-
126101083932262932*ep^10-24215192894038369*ep^9-3388926888965814*
ep^8-345637150383747*ep^7-25379551373304*ep^6-1306144656547*ep^5-
44872918806*ep^4-942184241*ep^3-10114752*ep^2-34236*ep,
```

```
32961195540000*ep^15+129399858315000*ep^14+224421151745700*ep^13+
23014813107951O*ep^12+156992002169205*ep^11+75753182355795*ep^10+
26785824281580*ep^9+7081231070340*ep^8+1413094602990*ep^7+
213064001010*ep^6+24090692640*ep^5+2006803890*ep^4+119252805*ep^3
+4773195*ep^2+115080*ep+1260)
;
```

This integral might come about in a 17-loop calculation.......

The polyratfun has an interesting option. Often in the 4-loop calculations of moments of splitting functions and coefficient functions we ran into the problem that the powers of ϵ in the rat function would become very high, but the actual calculation would produce only a limited number of poles in ϵ. In addition, the master integrals were only available in a limited power series expansion in ϵ. Hence the calculation might be much faster if we can expand the contents of the rat function. This can be done with

```
PolyRatFun rat(expand,ep,10);
```

for 10 powers. In a 17-loop calculation we would need up to 17 powers but for demonstration purposes we will use only 10.

```
PolyRatFun rat(expand,ep,10);
Print +f +s;
```

```
      .end

 Time =           0.25 sec      Generated terms =              3
                    F          Terms in output =              3
                               Bytes used      =       1548

    F=
        +Z(0,1,1,2,1,3,1,4,1,5)*rat(3531/35-94144/105*ep-32985809/630*
        ep^2+2853141427/3780*ep^3-5113646773/4536*ep^4-461581633221/
        136080*ep^5+95520335002859/163296*ep^6-5505333372797891/699840*
        ep^7+443278782772060729/4199040*ep^8-51508666473478911961/
        35271936*ep^9+2181334288935140623279/1058158080*ep^10)
        +Z(1,1,1,2,1,3,0,4,1,5)*rat(8877/140-139523/210*ep-14938465/504*
        ep^2+1383622943/3024*ep^3-112521249101/90720*ep^4-5082603133133/
        544320*ep^5+658515315781339/3265920*ep^6-1588086750856687/559872*
        ep^7+130557893406403961/3359232*ep^8-383612041150078109501/
        705438720*ep^9+32692137006874616833147/4232632320*ep^10)
        +Z(1,1,1,2,1,3,1,4,1,5)*rat(-951/35*ep-582322/105*ep^2-87132401/
        630*ep^3+323704849/540*ep^4+3324883475/4536*ep^5-1031319405421/
        27216*ep^6+88728924538811/163296*ep^7-35281794395723201/4898880*
        ep^8+82633996073507453/839808*ep^9-243807353102501443601/
```

```
        176359680*ep^10+20832364564601418947143/1058158080*ep^11)
        ;
```

When you inspect the last term, you will see an ϵ^{11}. FORM expands to ten terms from the leading term. In this case the last integral has a leading term in ϵ^1 and hence the ϵ^{11}.

The above looks very workable, but there is still one significant weakness. How do we generate the reduceT1 procedure without making mistakes, or in other words automatically? Also this can be done by FORM with some nice facilities. Let us first study this for a single weight 2 equation. Let us take the first of those equations as generated by the program that worked with INCCOMPLEXITY=2. This is rather fast for experimentation. The first equation that we select (number 159) looks like:

```
+Z(-2,2,1+n1,n2,n3,1+n4,n5)*(
   +rat(2*n1*n4,n5)
   )
+Z(-1,1+n1,n2,n3,n4,n5)*(
   +rat(-2*ep*n1-2*n1^2-2*n1*n4+2*n1,n5)
   )
+Z(-1,n1,n2,n3,1+n4,n5)*(
   +rat(-2*ep*n4-2*n1*n4-2*n4^2+2*n4,n5)
   )
+Z(-1,n1,n2,n3,n4,1+n5)*(
   +rat(2*ep+2*n5-2,1)
   );
```

It generates the id statement

```
id Z(-2,1+n1,n2,n3,1+n4,n5) =
   Z(-1,1+n1,n2,n3,n4,n5)*rat(ep+n1+n4-1,n4)+
   Z(-1,n1,n2,n3,1+n4,n5)*rat(ep+n1+n4-1,n1)+
   Z(-1,n1,n2,n3,n4,1+n5)*rat(-ep*n5-n5^2+n5,n1*n4);
```

but we would like the statement

```
id,ifmatch->looplabel,Z(n1?{>1},m1?,n2?,m2?,n3?,m3?,n4?{>1}
                                           ,m4?,n5?,m5?) =
   Z(n1,m1,n2,m2,n3,m3,n4-1,m4,n5,m5)*
        rat(ep+n1-1+m1*ep+n4-1+m4*ep-1,n4-1+m4*ep)+
   Z(n1-1,m1,n2,m2,n3,m3,n4,m4,n5,m5)*
        rat(ep+n1-1+m1*ep+n4-1+m4*ep-1,n1-1+m1*ep)+
   Z(n1-1,m1,n2,m2,n3,m3,n4-1,m4,1+n5,m5)*
        rat(-ep*(n5+m5*ep)-(n5+m5*ep)^2+n5+m5*ep,
              (n1-1+m1*ep)*(n4-1+m4*ep));
```

and if possible we would also like the polynomials in rat to be factorized to make them simpler, because eventually the n_i and m_i will be numbers and it is better to combine those in short

function arguments before multiplying them out.

The code in the gauss procedure that writes the statement to file is

```
#write <subs> "id '$lhs' = %2$;",$xxx
```

Unfortunately, if we keep this statement in this form, it rewrites into other complexity two integrals and all lower integrals. In that case we may need all 50 complexity 2 relations. We want to rewrite the necessary integrals to the three basis integrals, and hence we should not eliminate them from the system. This way the gauss elimination gets fully diagonalized. And of course we only keep the ones we need. For this we change the gauss procedure once more:

```
#call Drop('$imin')
#$match = 0;
#if ( 'c' <= 1 )
    #$numout = $numout+1;
    G  H'$numout' = '$lhs'-('$xxx');
    #$match = 1;
#elseif ( 'c' == 2 )
    #inside $lhs
        if ( match(Z(-2,?a,-1+n1,?b)) || match(Z(-2,?a,-1+n4,?b))
```

```
              || match(Z(-2,?a,-2+n1,?b))
              || match(Z(-2,?a,-2+n4,?b)) ) $match = 1;
         #endinside
         #if ( '$match' > 0 )
              #$numout = $numout+1;
              G  H'$numout' = '$lhs'-('$xxx');
         #endif
    #endif
*
*    Back to the regular form of the Z.
*    Then we write to the .log file what we will replace.
*

    id  Z(-'c',x?,n1?,...,n'N'?) = Z(-'c',n1,...,n'N');
    #write "id  '$lhs' ="
    #inside $lhs;
      id  Z(-'c',x?,n1?,...,n'N'?) = Z(-'c',n1,...,n'N');
    #endinside;

*
```

```
*     And this is the statement that eliminates this integral.
*
      #if ( 'c' <= 2 )
          if ( ( $match == 1 ) && ( expression(H'$numout') == 0 ) )
      #endif;
            id  '$lhs' = '$xxx';
      .sort
```

This way we create a new series of equations H that contain all relevant equations and no more. In addition the relevant integrals all have coefficient one. This means that they will need no further work. They will be available at the end of the program, where we can use them to write the reduceT1 procedure.

The procedure that can do this contains a new feature again: a dictionary. A dictionary has a name and it allows us to give us very flexible control over the output. If we need for instance LaTeX output we can let FORM print n1 as n_1, and we can let FORM print a blank space instead of a multiplication sign. And many more things. In this case we tell FORM that when the dictionary is active some variables have to be printed as wildcards. We assume that the expressions have already been expressed in terms of factorized num and den functions.

```
   #procedure CreatePRC(name,H,num)
```

```
#opendictionary LHS
  #do j = 1,5
      #add m'j': "m'j'?"
      #add n'j'p0: "n'j'?"
      #add n'j'm0: "n'j'?"
      #add n'j'p1: "n'j'?{>1}"
      #add n'j'm1: "n'j'?neg0_"
      #add n'j'p2: "n'j'?{>2}"
      #add n'j'm2: "n'j'?neg_"
  #enddo
#closedictionary
#write <'name'.prc> "#procedure 'name'"
#write <'name'.prc> "*\n"
#write <'name'.prc> "*  Procedure automatically created 'date_'\n*"
#write <'name'.prc> "#$numloop = 0;"
#write <'name'.prc> "#do inumloop = 1,1"
#write <'name'.prc> "#$numloop = $numloop+1;"
#write <'name'.prc> "#$loopaction = 0;"
```

```
*B    Z;
.sort
#do i = 1,'num'
    #$lhs = firstterm_(H'i');
    #$xxx = -H'i'+$lhs;
    #do j = 1,5
        #$change'j' = 0;
    #enddo
    #inside $lhs
      id  Z(x1?,x?,n1?,...,n'N'?) = Z(n1,...,n'N');
        #do j = 1,5
          #do k = 0,2
            if ( match(Z(?a,n'j'+'k',?b)) );
                $change'j' = 'k';
                id  Z(?a,n'j'+'k',?b) = Z(?a,n'j'p'k',?b);
            elseif ( match(Z(?a,n'j'-'k',?b)) );
                $change'j' = -'k';
                id  Z(?a,n'j'-'k',?b) = Z(?a,n'j'm'k',?b);
```

```
          endif;
        #enddo
      #enddo
      id  Z(n1?,...,n5?) = Z(<n1,m1>,...,<n5,m5>);
    id  rat(1,1) = 1;
#endinside
#inside $xxx
  id  Z(x1?,x?,n1?,...,n'N'?) = Z(n1,...,n'N');
  id  Y(?a) = Z(?a);
    Argument Z,rat,num,den;
        #do j = 1,5;
            id  n'j' = n'j'+m'j'*ep-'$change'j'';
        #enddo
    EndArgument;
    Argument Z;
        id  ep = 1;
    EndArgument;
    #do j = 1,5
```

```
            SplitArg,((m'j')),Z;
         #enddo
      #endinside
      #usedictionary LHS($)
        #write <'name'.prc> "id,ifmatch->looplabel,%2$",$lhs
      #closedictionary
      #write <'name'.prc> "  =%2$;",$xxx
#enddo
#write <'name'.prc> "goto looplabel2;"
#write <'name'.prc> "label looplabel;"
#write <'name'.prc> "id  num(x?)*den(x?) = 1;"
#write <'name'.prc> "id  num(x?) = rat(x,1);"
#write <'name'.prc> "id  den(x?) = rat(1,x);"
#write <'name'.prc> "*id num(x?number_) = x;"
#write <'name'.prc> "*id den(x?number_) = 1/x;"
#write <'name'.prc> "$loopaction = 1;"
#write <'name'.prc> "label looplabel2;"
#write <'name'.prc> "ModuleOption Maximum,$loopaction;"
```

```
    #write <'name'.prc> ".sort:'name'-loop \'$numloop\';"
    #write <'name'.prc> "#if ( \'$loopaction\' == 1 )"
    #write <'name'.prc> "    #redefine inumloop \"0\""
    #write <'name'.prc> "#endif"
    #write <'name'.prc> "#enddo"
    #write <'name'.prc> "*"
    #write <'name'.prc> "#endprocedure"
#endprocedure
```

The major work in the procedure is to change something like

```
    id Z(n1,n2+1,n3,n4,n5) = f(n1,n2,n3,n4,n5)
```

for some function f, into

```
    id Z(n1,n2?{>1},n3,n4,n5) = f(n1,n2-1,n3,n4,n5)
```

or

```
    id Z(n1,n2-1,n3,n4,n5) = f(n1,n2,n3,n4,n5)
```

into

```
    id Z(n1,n2?neg0_,n3,n4,n5) = f(n1,n2+1,n3,n4,n5)
```

The needed modifications are detected in the #inside $lhs part of the code, the shifts are executed and the left hand side of the id statement is written while the dictionary is active. Then there is one more complication: The n_i contain also a multiple m_i of ϵ. We have to take the m_i separately, because otherwise we cannot check the n_i for the conditions we want to impose on them.

At this point we can run the complete program for INCCOMPLEXITY=3 again. The value 4 was also tried but gives nothing new, except for that it takes almost 20000 sec on a big computer while the value 3 takes 2422 sec.

How does this compare with how other people reduce their integrals to master integrals? The most popular other method has been introduced in "High precision calculation of multiloop Feynman integrals by difference equations", S. Laporta, Int.J.Mod.Phys. A15 (2000) 5087-5159 DOI: 10.1142/S0217751X00002157. Whereas the parametric reduction starts with the most complex integrals and gradually work things down to the lowest complexities, the Laporta method works the other way. One starts with the simplest equations and solves what can be solved. The results are stored. Next one goes to the next level in complexity and solves what can be solved, storing the results. Etc. Eventually all integrals that are needed have been stored, ready for use.

Both methods have advantages and disadvantages. In the method of Laporta one may need very large storage facilities. Also, one has to compute many integrals that are never needed. In the parametric reduction one calculates only the integrals that are actually needed, without the need for storage. On the other hand, the derivation of a parametric reduction scheme requires much more care and usually more manual interference. Because the Laporta method can be fully automated (as done by A. von Manteufel with the Reduze program and later also by others), nowadays most people use this method. The parametric reduction however gives by far the fastest programs.

Both methods suffer from problems with rational polynomials. It gets worse when masses

and more external lines are included. This gives extra variables in the problem and they become part of the rational coefficient. One popular solution for the Laporta method is to use an external program (Fermat) for the rational polynomials. The whole becomes very slow and programs can run sometimes for months on powerful computers with many cores, just to obtain tables for the topologies of a reaction, and the tables may take Terabytes. In contrast the whole Forcer program for four-loop massless integrals takes about 10 Mbytes in total.

Homework: Try to modify the ending of the reduceT1 procedure such that it can handle also the cases in which some of the m_i can be zero. This involves getting the integrals to such a basis that there are as few Z integrals as possible remaining. In the case that both the n_i and m_i are zero, the integral is renamed as a Y integral. Those can be expressed in terms of Γ functions. In particular, when there are three m_i that are zero, there is at most one master integral (of the Z type). The equations (all are zero) that are relevant here are:

```
H18=
    +Z(n1,-1+n2,n3,n4,n5)
    -Z(-1+n1,n2,n3,n4,n5)*num(n4-n2)*num(-4+n5+n4+n1+2*ep)*
       den(-3+n5+n3+n2+ep)*den(-4+n5+n3+n2+2*ep)
    -Z(n1,n2,n3,-1+n4,n5)*num(-3+n5+n2+n1+ep)*
       num(-4+n5+n4+n1+2*ep)*den(-3+n5+n3+n2+ep)*
```

```
        den(-4+n5+n3+n2+2*ep)
    +Z(n1,n2,n3,n4,n5)*num(n4-n2)*num(-3+n5+n2+n1+ep)*
        den(-3+n5+n3+n2+ep)*den(-4+n5+n3+n2+2*ep)
    ;
H19=
    +Z(n1,n2,-1+n3,n4,n5)
    -Z(-1+n1,n2,n3,n4,n5)*num(-3+n5+n4+n3+ep)*
        num(-4+n5+n4+n1+2*ep)*den(-3+n5+n3+n2+ep)*
        den(-4+n5+n3+n2+2*ep)
    +Z(n1,n2,n3,-1+n4,n5)*num(n3-n1)*num(-4+n5+n4+n1+2*ep)*
        den(-3+n5+n3+n2+ep)*den(-4+n5+n3+n2+2*ep)
    -Z(n1,n2,n3,n4,n5)*num(n3-n1)*num(-3+n5+n4+n3+ep)*
        den(-3+n5+n3+n2+ep)*den(-4+n5+n3+n2+2*ep)
    ;
H20=
    +Z(n1,n2,n3,n4,-1+n5)
    -Z(-1+n1,n2,n3,n4,n5)*num(-3+n5+n4+n3+ep)*
        num(-7+2*n5+n4+n3+n2+n1+3*ep)*den(-3+n5+n3+n2+ep)*
```

```
    den(-4+n5+n3+n2+2*ep)
-Z(n1,n2,n3,-1+n4,n5)*num(-3+n5+n2+n1+ep)*
    num(-7+2*n5+n4+n3+n2+n1+3*ep)*den(-3+n5+n3+n2+ep)*
    den(-4+n5+n3+n2+2*ep)
+Z(n1,n2,n3,n4,n5)*num(-3+n5+n2+n1+ep)*num(-3+n5+n4+n3+ep)*
    den(-3+n5+n3+n2+ep)*den(-4+n5+n3+n2+2*ep)
;
```