FIVE

In this lecture we will have a look at the construction of the mincer program. The mincer program is a program that calculates 3-loop massless propagator diagrams in n-dimensional regularization. We saw already an example of it in the second lecture:

```
    #include- minceex.h
    Off Statistics;
    Format nospaces;
    .global
    L   F = Q.Q^4/p1.p1^2/p2.p2^2/p3.p3/p4.p4^2/p5.p5;
    #call integral(t1,0)
    #call subvalues
~~~Answer in the Gscheme
    #call expansion(1)
~~~Answer in the Gscheme
```

```
  Print +f;
  .end

 F=
    -16+2*ep^-2-3*ep^-1+73/2*ep;
0.01 sec out of 0.01 sec
```

First some history.

The algorithms mincer uses for calculating such diagrams come from the following papers:

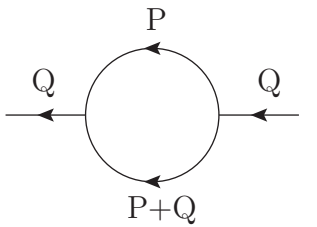
K.G. Chetyrkin, A.L. Kataev and F.V. Tkachov, Nucl. Phys. B174 (1980) 345.
K.G. Chetyrkin and F.V. Tkachov, Nucl. Phys. B192 (1981) 159.
F.V.Tkachov, Phys. Lett. 100B (1981).
F.V. Tkachov, Teor. Mat. Fiz. 56 (1983) 350.

A first program was already constructed in 1989 for the Schoonschip system. (S.G. Gorishny, S.A. Larin, L.R. Surguladze and F.V. Tkachov,Comp. Phys. Comm. 55 (1989) 381.) This was quite an achievement considering many of the limitations of Schoonschip. It was used then for the calculation $e^-e^+ \to$ hadrons to $\mathcal{O}(\alpha_S^3)$ which is actually a 4-loop calculation but because only the divergent parts were needed some theorems could be applied and it could be converted to the calculation of 3-loop massless propagator diagrams. S.G.Gorishny, A.L.Kataev and S.A.Larin, Phys. Lett. B212 (1988) 238, ibid. B259 (1991) 144;

Shortly after FORM became available and the mincer algorithms were reprogrammed in terms of FORM. It was then used for a number of calculations, each more demanding than the previous one. This was possible due to the steady increase in computer power and the continuous expansion of the capabilities of FORM. For several years mincer was the great inspiration for FORM improvements. This role was later taken over by other programs, most recently by forcer which is the 4-loop variety of mincer. This program is however much more complex and hence not really suitable for a class of two hours. The FORM version of the mincer program has been used by others as well and is still very relevant.

There are basically two versions of mincer: mincer.h and minceex.h. The first works with expansions in $\epsilon = 2 - D/2$ (D is the dimension of space-time) and the second is exact but slower.

Before starting off with 3-loop diagrams, we better have a look at 1-loop diagrams. The basic one loop diagram is given by:



The integral is given by

$$\int \frac{d^D P}{(2\pi)^D} \frac{\mathcal{P}_n(P)}{P^{2\alpha}(P+Q)^{2\beta}} =$$

$$\frac{1}{(4\pi)^2}(Q^2)^{D/2-\alpha-\beta} \sum_{\sigma\geq 0}^{[n/2]} G(\alpha,\beta,n,\sigma) Q^{2\sigma} \left\{ \frac{1}{\sigma!} \left(\frac{\Box}{4}\right)^{\sigma} \mathcal{P}_n(P) \right\}_{P=Q},$$

in which

$$\mathcal{P}_n(P) \;=\; P_{\mu_1} P_{\mu_2} \cdots P_{\mu_n}.$$

D is the dimension of space-time and is also given by $D = 4 - 2\epsilon$, $\Box = \partial^2/\partial P_\mu \partial P_\mu$ and G can be expressed in terms of Γ-functions:

$$G(\alpha,\beta,n,\sigma) \;=\; (4\pi)^{\epsilon} \frac{\Gamma(\alpha+\beta-\sigma-D/2)\Gamma(D/2-\alpha+n-\sigma)\Gamma(D/2-\beta+\sigma)}{\Gamma(\alpha)\Gamma(\beta)\Gamma(D-\alpha-\beta+n)}.$$

If we only solve a one-loop problem, we will have integer values for α and β, but when we go to higher loops we will see that this is not the case. Yet we can normalize these functions to just a few functions of the type $G(1+m\epsilon,1,0,0)$ multiplied by Pochhammer symbols:

$$P(x,n) \;=\; \frac{\Gamma(x+n)}{\Gamma(x)}$$

which we can make into

$$Poch(n,m) = \frac{\Gamma(n+m\epsilon)}{\Gamma(1+m\epsilon)}$$

in which case it becomes clear that these Pochhammer symbols are rational polynomials in ϵ (or in D if one prefers it that way).

How to deal with rational polynomials in FORM?

For this we have a special system. Sometimes we would like the coefficient of a term to be more general than just a number. In that case we have two options. The first is the 'polyfun':

```
CFunction acc;
PolyFun acc;
```

In this case the argument of acc is supposed to be a polynomial and the whole function is treated as the coefficient of the term:

```
Symbols x,ep;
CFunction acc;
Off Statistics;
Format nospaces;
L   F = x*acc(1+ep+ep^2)+(1+x)^2*acc(2+3*ep);
Print +s;
Bracket x;
.sort

F=
```

```
   +x*(
      +acc(1+ep+ep^2)
      +2*acc(2+3*ep)
      )
   +x^2*(
      +acc(2+3*ep)
      )
   +acc(2+3*ep)
      ;
 PolyFun,acc;
 Print +s;
 Bracket x;
 .end

F=

   +x*(
```

```
    +acc(5+7*ep+ep^2)
    )
+x^2*(
    +acc(2+3*ep)
    )
+acc(2+3*ep)
    ;
```

As you can see, in the x term the contents of the acc functions have been combined. The advantage of the polyfun is that there are effectively fewer terms and hence the pattern matching on the 'relevant' parts of the terms will be faster. The action FORM takes is

- When there are multiple polyfuns in a term their contents are multiplied and the whole is replaced by a single polyfun.
- If a term does not have a polyfun yet, a polyfun with argument 1 is created.
- A possible numerical coefficient of the term is taken inside the polyfun.
- When two terms need to be added the contents of their polyfuns will be added.
- When a term is printed, the polyfun is always printed last.

```
 Symbols x,ep;
 CFunction acc;
 PolyFun acc;
 Off Statistics;
 Format nospaces;
 L   F = x*acc(1+ep+ep^2)+(1+x)^2*acc(2+3*ep);
 Print +s;
 .sort

F=
   +x*acc(5+7*ep+ep^2)
   +x^2*acc(2+3*ep)
   +acc(2+3*ep)
   ;
 Drop;
 L   H = F^2;
 Print +s;
 .end
```

```
H=
    +x*acc(20+58*ep+46*ep^2+6*ep^3)
    +x^2*acc(33+94*ep+77*ep^2+14*ep^3+ep^4)
    +x^3*acc(20+58*ep+46*ep^2+6*ep^3)
    +x^4*acc(4+12*ep+9*ep^2)
    +acc(4+12*ep+9*ep^2)
    ;
```

The second option is far more expensive: the polyratfun. This function needs two arguments: the first is interpreted as the numerator and the second as the denominator of a fraction. It is always normalized in such a way that the symbols inside (other objects are not allowed) have positive powers and the numerical coefficients are integers.

```
Symbols x,ep;
Cfunction rat;
PolyRatFun rat;
L   F = x*rat(2,1+ep)+x*rat(2,1-ep);
Print;
.end

F =
   x*rat(-4,ep^2 - 1);
```

The algorithms for dealing with rational polynomials are rather efficient when only a single variable is involved. In the case of multivariate polynomials there are faster dedicated programs. People have used a system called Fermat as external program to deal with multivariate rational polynomials before FORM had this capability, but the overhead of the communication is rather big.

OK, now we can program a Pochhammer symbol:

```
repeat id Pochhammer(x?pos_,m?) = Pochhammer(x-1,m)*rat(x-1+m*ep,1);
repeat id Pochhammer(x?neg_,m?) = Pochhammer(x+1,m)*rat(1,x+m*ep);
id  Pochhammer(0,m?) = 1;
```

It is convenient to also define the function

```
PochhammerINV(n?,m?) = 1/Pochhammer(n,m)
```

because of the fact that FORM does not really like denominators.

We are now ready to build our first routine, which is the routine that can work out the G function in terms of a few basic integrals and rational polynomials in ϵ. The G function here has 6 arguments to make it easier to work with the multiples of ϵ. Effectively

```
G(n1,x1,n2,x2,n3,n4) = G(n1+x1*ep,n2+x2*ep,n3,n4)
```

```
#procedure DoG
*
*    The only objects left are the G(1,x1,1,x2,0,0)
*    which have been written as GschemeConstants(x1,x2)
```

```
*
id G(n1?,x1?,n2?,x2?,n3?,n4?) = GschemeConstants(x1,x2)/(1+x1+x2)*
         Pochhammer(n1+n2-n4-2,ep+x1*ep+x2*ep)*
         Pochhammer(1-n1+n3-n4,1-ep-x1*ep)*
         Pochhammer(1-n2+n4,1-ep-x2*ep)*
         PochhammerINV(n1-1,1+x1*ep)*
         PochhammerINV(n2-1,1+x2*ep)*
         PochhammerINV(2-n1-n2+n3,2-2*ep-x1*ep-x2*ep);
repeat id Pochhammer(n?pos_,x?) = Pochhammer(n-1,x)*num(n-1+x);
repeat id Pochhammer(n?neg_,x?) = Pochhammer(n+1,x)*den(n+x);
repeat id PochhammerINV(n?pos_,x?) = PochhammerINV(n-1,x)*den(n-1+x);
repeat id PochhammerINV(n?neg_,x?) = PochhammerINV(n+1,x)*num(n+x);
id   GschemeConstants(0,n?pos_) = GschemeConstants(n,0);
id   GschemeConstants(1,1)*GschemeConstants(0,0) =
      GschemeConstants(1,0)*GschemeConstants(2,0)*rat(1-3*ep,1-2*ep);
id   Pochhammer(0,x?) = 1;
id   PochhammerINV(0,x?) = 1;
id   num(x?)*den(x?) = 1;
```

```
id   den(x?number_) = 1/x;
id   num(x?number_) = x;
id   num(x?) = rat(x,1);
id   den(x?) = rat(1,x);
*
#endprocedure
```

In principle we have two lines that can have a noninteger power but by studying different ways of solving so-called watermelon diagrams we can always reduce this to G functions with only a single line with a noninteger power.

The GschemeConstants function is the G function of a diagram with no numerators, one line with power one and one line with power $1 + m\epsilon$. We have to work out the ratio $G(1 + m\epsilon, 1, 0, 0)/G(1, 1, 0, 0)$ which is rather simple, and we keep the powers of $G(1, 1, 0, 0)$ as overal constants. Exclusion of these factors is called the G-scheme. Conversion to MS-bar involves working out the powers of $G(1, 1, 0, 0)$ which means mostly expanding a number of gamma functions.

Note that as long as we keep the formulas in terms of the GschemeConstants function the routine is exact when expressed in the polyratfun rat.

Now we can program the routine for doing the one-loop diagrams. Note that we have a sum over d'Alembertians and that this is exactly what one of the homework problems was about. Actually, the distrib_ and dd_ functions were invented for this one-loop problem. In practical calculations one may encounter rather high powers of the integration momentum in the numerator (the record is bigger than 30).

```
#procedure IntOne(p3,p4,Q,in,out)
*
*   Loop momenta are p3 and p4. The 'propagator' is Q.
*   We integrate over p3
*   epp3 = 1/p3.p3^ep, epp4 = 1/p4.p4^ep, epQ = 1/Q.Q^ep
*   in and out are markers to tell whether a term should be integrated
*   over, in for here and out for potentially next routines.
*
if ( count(int`in',1) );
  if ( ( count(ep`p3',1) == 0 ) && ( count(`p3'.`p3',1) >= 0 ) ) Discard;
  if ( ( count(ep`p4',1) == 0 ) && ( count(`p4'.`p4',1) >= 0 ) ) Discard;
  ToTensor,nosquare,ftensor,`p3';
  if ( count(ftensor,1) == 0 );
    id  int`in'*ep`p3'^x3?*ep`p4'^x4?/`p3'.`p3'^n3?/`p4'.`p4'^n4? =
            int`out'*G(n3,x3,n4,x4,0,0)*
               `Q'.`Q'^2/`Q'.`Q'^n3/`Q'.`Q'^n4*ep`Q'^x3*ep`Q'^x4*ep`Q';
  elseif ( match(ftensor(i1?)) );
    id  int`in'*ep`p3'^x3?*ep`p4'^x4?/`p3'.`p3'^n3?/`p4'.`p4'^n4?*
        ftensor(i1?) = int`out'*`Q'(i1)*G(n3,x3,n4,x4,1,0)*
```

```
                ‘Q’.‘Q’^2/‘Q’.‘Q’^n3/‘Q’.‘Q’^n4*ep‘Q’^x3*ep‘Q’^x4*ep‘Q’;
  elseif ( match(ftensor(i1?,i2?)) );
    id  int‘in’*ep‘p3’^x3?*ep‘p4’^x4?/‘p3’.‘p3’^n3?/‘p4’.‘p4’^n4?*
        ftensor(i1?,i2?) = int‘out’*‘Q’.‘Q’^2*ep‘Q’*ep‘Q’^x3*
            ep‘Q’^x4/‘Q’.‘Q’^n3/‘Q’.‘Q’^n4*(
             +G(n3,x3,n4,x4,2,0)*‘Q’(i1)*‘Q’(i2)
             +G(n3,x3,n4,x4,2,1)*d_(i1,i2)*‘Q’.‘Q’/2);
  else;
    id  int‘in’*ep‘p3’^x3?*ep‘p4’^x4?/‘p3’.‘p3’^n3?/‘p4’.‘p4’^n4?*
        ftensor(?a) = int‘out’*ftensor(?a)
            *sum_(isum2,0,integer_(nargs_(?a)/2),
              G(n3,x3,n4,x4,nargs_(?a),isum2)
                *y^isum2*‘Q’.‘Q’^isum2/2^isum2)*‘Q’.‘Q’^2
                *ep‘Q’*ep‘Q’^x3*ep‘Q’^x4/‘Q’.‘Q’^n3/‘Q’.‘Q’^n4;
    id  y^isum2?*ftensor(?a) = distrib_(1,2*isum2,del,ftensor,?a);
    tovector,ftensor,‘Q’;
    id  del(?a) = dd_(?a);
  endif;
  id  P.P = 0;
endif;
*
.sort:IntOne-‘in’-1;
*
#call DoG
*
```

```
.sort:IntOne-‘in’-2;
*
#endprocedure
```

As in the answers file, we introduce a tensor and in the general case we use the distrib_ and dd_ functions to work out the d'Alembertians, but we keep the cases of zero, one or two powers of the loop momentum separate, because the use of the sum_ function does represent a certain overhead that we try to avoid when things are very simple.

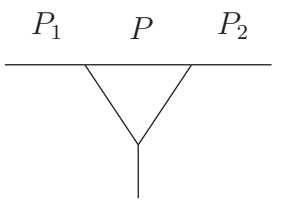
Time to try things out:

```
  #include- minceex.h
  Off Statistics;
  Format nospaces;
  .global
  L   F = epp1/p1.p1^5*epp2/p2.p2^3*Q.Q^6;
  #call integral(l1,0)
  Print +f;
  .end

 F=
    GschemeConstants(1,1)*epQ^2*rat(62208*ep^7+295488*ep^6+521424*
    ep^5+415260*ep^4+132672*ep^3-4068*ep^2-10344*ep-1440,4*ep^7+56*
    ep^6+319*ep^5+956*ep^4+1621*ep^3+1544*ep^2+756*ep+144);
0.01 sec out of 0.01 sec
```

Before going to the two-loop diagrams we need a whole new type of relations. These are called Integration By Parts (=IBP) relations. This is currently the most powerful tool for dealing with multi-loop integrals and a number of programs has been developed for working with them: Reduze, Fire, LiteRed,.... They all have one thing in common: they are extremely demanding w.r.t. computer resources. Yet, in many cases we do not have anything better. However, in the case of massless propagator diagrams we do.

Let us start with the triangle graph:



We define the integral

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

This integral can be studied by first considering the identity:

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

which is an integral over a total derivative.

Working out the derivative one obtains the recursion relation (called the rule of the triangle):

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

If this is part of a diagram in which the α parameters are all positive integers, repeated application of this equation will eventually make one of these parameters zero. It is however possible to improve upon this by 'solving' the recursion. This gives the formula (Tkachov):

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

The complete triangle routine looks like

```
#procedure ntriangle(p,pa,pb,p1,p2)
*
*   Routine solves the triangle recursion
*                     p
*      n1 ------------------------- n2
*          p1    \    n0    /    p2
*                 \        /
*               n3 \      / n4
*                  pa \    / pb
*                      \  /
*                       \/
*
*   n3,n4,n0,n1,n2 are the powers of the denominators
*   eppa and eppb are 1/pa.pa^ep and 1/pb.pb^ep
*   p is the momentum in n0. We need it here to determine the extra
*   momenta in the numerator.
*
id  `p' = xpower*`p';
if ( count(`p1'.`p1',1,`p2'.`p2',1,`p'.`p',1) >= -4 );
*
*   Here we can just use the recursion
*
    repeat;
        id  xpower^n8?/`p1'.`p1'^n1?pos_/`p2'.`p2'^n2?pos_/`p'.`p'^n?pos_/`pa'.`pa'^n3?/`pb'.`pb'^n4?
            *ep`pa'^x1?*ep`pb'^x2? = xpower^n8/`p1'.`p1'^n1/`p2'.`p2'^n2/`p'.`p'^n/`pa'.`pa'^n3/`pb'.`pb'^n4
                *ep`pa'^x1*ep`pb'^x2*(
                    +num(n3+x1*ep)*(`p'.`p'/`pa'.`pa'-`p1'.`p1'/`pa'.`pa')
                    +num(n4+x2*ep)*(`p'.`p'/`pb'.`pb'-`p2'.`p2'/`pb'.`pb')
                )*den(4-2*ep+n8-2*n-n3-n4-x1*ep-x2*ep);
    endrepeat;
    id  num(x?number_) = x;
    id  den(x?number_) = 1/x;
```

```
    id  num(x?)*den(x?) = 1;
    id  num(x?) = rat(x,1);
    id  den(x?) = rat(1,x);
else;
*
*   Here we have to use the general formula
*
    id  xpower^n8?/`p1'.`p1'^n1?pos_/`p2'.`p2'^n2?pos_/`p'.`p'^n?pos_/`pa'.`pa'^n3?/`pb'.`pb'^n4?
            *ep`pa'^x1?*ep`pb'^x2? = ftriangle(n8,n,n3,x1,n4,x2,n1,n2);
    id  ftriangle(n?,n0?,n3?,x1?,n4?,x2?,n1?,n2?) =
        +sum_(isum1,0,n2-1,sum_(isum2,0,n0-1,sum_(isum3,0,n0-1-isum2,
            ftriangle(n,n0-isum2-isum3,n3+n1+isum2,x1,n4+isum1+isum3,x2,0,n2-isum1)*sign_(n1+isum1)
                *fac_(n1+isum1+isum2+isum3-1)
                *Pochhammer(n1+isum2,n3+x1*ep)
                *Pochhammer(isum3+isum1,n4+x2*ep)
                *Pochhammer(-n1-isum1-isum2-isum3,isum2+isum3+5-2*ep+n-2*n0-n3-n4-x1*ep-x2*ep)
                *invfac_(isum1)*invfac_(isum2)*invfac_(isum3)*invfac_(n1-1)
        )))
        +sum_(isum1,0,n1-1,sum_(isum2,0,n2-1,sum_(isum3,0,n0,
            ftriangle(n,0,n3+isum1+isum3,x1,n4+n0+isum2-isum3,x2,n1-isum1,n2-isum2)*sign_(isum1+isum2)
                *fac_(n0+isum1+isum2-1)
                *Pochhammer(isum3+isum1,n3+x1*ep)
                *Pochhammer(n0-isum3+isum2,n4+x2*ep)
                *Pochhammer(-isum1-isum2-n0,4-2*ep+n-n0-n3-n4-x1*ep-x2*ep)
                *n0*invfac_(n0-isum3)
                *invfac_(isum1)*invfac_(isum2)*invfac_(isum3)
        )))
        +sum_(isum1,0,n1-1,sum_(isum2,0,n0-1,sum_(isum3,0,n0-1-isum2,
            ftriangle(n,n0-isum2-isum3,n3+isum1+isum3,x1,n4+n2+isum2,x2,n1-isum1,0)*sign_(n2+isum1)
                *fac_(n2+isum1+isum2+isum3-1)
                *Pochhammer(n2+isum2,n4+x2*ep)
                *Pochhammer(isum3+isum1,n3++x1*ep)
                *Pochhammer(-n2-isum1-isum2-isum3,isum2+isum3+5-2*ep+n-2*n0-n3-n4-x1*ep-x2*ep)
```

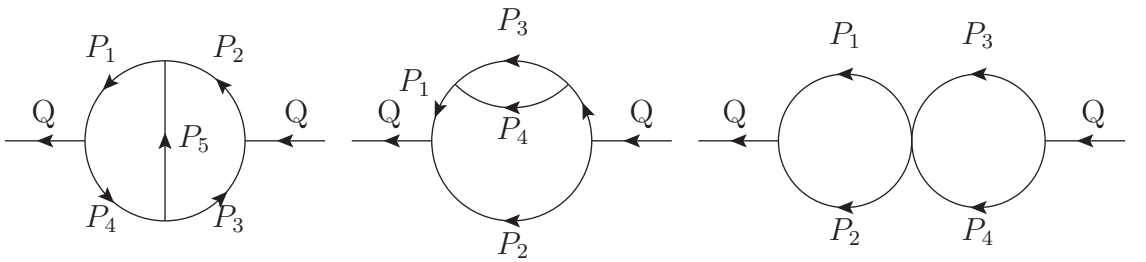
```
                *invfac_(isum1)*invfac_(isum2)*invfac_(isum3)*invfac_(n2-1)
        )));
    repeat id Pochhammer(n?pos_,x?) = Pochhammer(n-1,x)*num(n-1+x);
    repeat id Pochhammer(n?neg_,x?) = Pochhammer(n+1,x)*den(n+x);
    id  Pochhammer(0,x?) = 1;
    id  num(x?number_) = x;
    id  den(x?number_) = 1/x;
    id  num(x?)*den(x?) = 1;
    id  num(x?) = rat(x,1);
    id  den(x?) = rat(1,x);
    id  ftriangle(n?,n0?,n3?,x1?,n4?,x2?,n1?,n2?) =
        ep‘pa’^x1*ep‘pb’^x2/‘p’.‘p’^n0/‘p1’.‘p1’^n1/‘p2’.‘p2’^n2/‘pa’.‘pa’^n3/‘pb’.‘pb’^n4;
endif;
id  xpower = 1;
*
#endprocedure
```

Again, the simple cases are treated separately.

Recently a multi-loop generatization of this formula was formulated, called the diamond rule. It starts becoming relevant for 4- and more-loop propagator diagrams.

How does the triangle formula affect us?

The two loop topologies are given by



The topologies T1, T2, T3

The topologies T2 and T3 can be solved by applying the one-loop equation twice. The topology T1 is treated by applying the rule of the triangle. This will eliminate the denominator belonging to one of the lines and the resulting integrals are either of the type T2 or of the type T3. There is one complication that arises when the power of the center line in T1 is not an integer. In that case the application of the rule of the triangle is useless as it cannot reduce the power of the center line (α_0) to zero. Hence we have to use different recursion relations which are derived by using other momenta than the center line for the derivative in the IBP

relation(s):

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

and

$$
\begin{aligned}
&(\alpha_0 - 1 + \epsilon) I(0, \alpha_0, 1, 1, 1, 1) = \\
&\quad - (\alpha_0 - 2 + 2\epsilon) I(0, \alpha_0 - 1, 1, 1, 1, 1) Q^{-2} \\
&\quad - 2(3 - 3\epsilon - 2\alpha_0) G(1, \alpha_0, 0, 0) G(1, \alpha_0 + \epsilon, 0, 0) Q^{-2(2+\alpha_0)}.
\end{aligned}
$$

In both cases we have one integral left that cannot be solved by these means. We call this a master integral. It has been computed to order ϵ^6 with the use of symmetry considerations and more recently by other powerful techniques:

$$
I(0, 1 + \epsilon, 1, 1, 1, 1) \; = \; (6\zeta_3 + 9\zeta_4\epsilon + 102\zeta_5\epsilon^2 + 240\zeta_6\epsilon^3 - 186\zeta_3^2\epsilon^3 + \cdots \;)(1 - 2\epsilon)
$$

In the case that more lines have a noninteger power things can become rather messy. One has to write down all possible IBP identities for the system and then keep combining them in

such a way that one gets relations that can bring each line down to either zero or $1+m\epsilon$. Then either there is still an equation left that can still eliminate one line, or we have to stop and call the result a master integral. Solving such master integrals is an active field of research. For the T1 topology all integrals can be expanded in ϵ to rather high powers (Bierenbaum and Weinzierl).

Test:

```
 #include- minceex.h
 Off Statistics;
 Format nospaces;
 .global
 L   F = Q.Q^11/p1.p1/p2.p2^2/p3.p3^3/p4.p4^4/p5.p5^5;
 #call integral(t1,0)
 *#call subvalues
 *#call expansion(1)
 Print +f +s;
 .end

F=
    +GschemeConstants(0,0)^2*rat(-512*ep^8-9472*ep^7-71680*ep^6-
    281344*ep^5-595744*ep^4-615952*ep^3-156176*ep^2+171720*ep+77400,3
    *ep^2+27*ep+60)
    +GschemeConstants(0,0)*GschemeConstants(1,0)*rat(314928*ep^18+
    12807072*ep^17+234621360*ep^16+2562787836*ep^15+18618955245*ep^14
    +95025037491*ep^13+350726473557*ep^12+948624569001*ep^11+
```

```
    1879549568667*ep^10+2678165031051*ep^9+2599404875847*ep^8+
    1465738694559*ep^7+182381717340*ep^6-217188534474*ep^5+
    85111370496*ep^4+21478597034 4*ep^3+54977266080*ep^2-23618822400*
    ep-7838208000,2*ep^10+102*ep^9+2284*ep^8+29508*ep^7+242922*ep^6+
    1326798*ep^5+4845736*ep^4+11607432*ep^3+17286656*ep^2+14252160*ep
    +4838400)
    ;
0.08 sec out of 0.08 sec
```

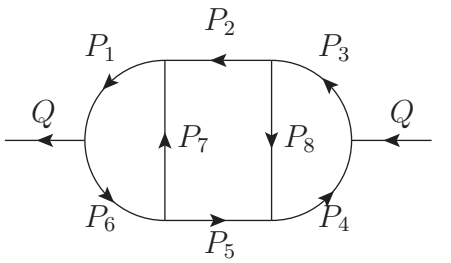
```
    #define MSBAR "1"
    #include- minceex.h
    Off Statistics;
    Format nospaces;
    .global
    L   F = Q.Q^11/p1.p1/p2.p2^2/p3.p3^3/p4.p4^4/p5.p5^5;
    #call integral(t1,0)
    #call subvalues
~~~Answer in MS-bar
    #call expansion(1)
~~~Answer in MS-bar
    Print +f +s;
    .end

   F=
      +41497259/5880
      -330*ep^-2
      -21507/28*ep^-1
```

```
    +220765958387/9878400*ep
    +11260*ep*z3
    ;
0.12 sec out of 0.12 sec
```

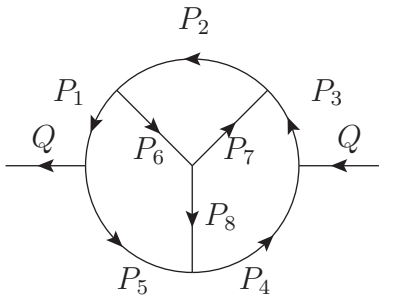
For three-loop integrals there are 14 topologies that have to be considered. Some of these are rather easy but there are 5 topologies that need special attention. One of them is outright complicated.

The first topology to consider is the ladder topology, also called LA:



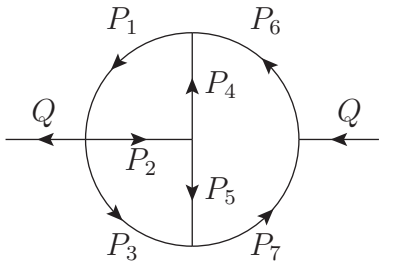
Diagrams of this type can be simplified by applying the rule of the triangle to either the loop with P_7, P_1 and P_6, or to the loop with P_8, P_4 and P_3. Assuming that we use the first triangle we will get then a diagram that misses either P_7 in the denominator (topology O3) or it misses P_2 or P_5 in the denominator (topology FA). The resulting topologies are treated later in this section.

The second basic topology is the Benz topology, also called BE:



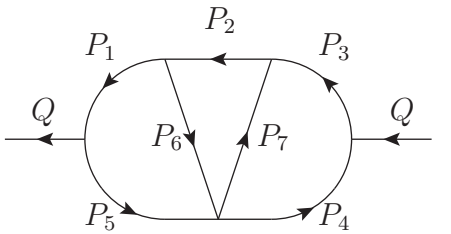
Again we can apply the rule of the triangle to simplify such a diagram. The triangle contains the momenta P_2, P_6 and P_7. After this we have diagrams that miss either P_2 (topology O1) or they miss P_1 or P_3 (topology BU). These topologies are treated later in this section.

The topology BU is a derivative topology. This means that it is a topology that can be obtained by removing one or more denominators in a more complicated topology with the same number of loops. It is given by:



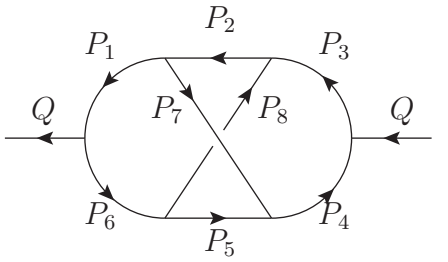
Also this topology can be simplified with an application of the rule of the triangle. In this case the triangle is either the triangle with the momenta P_4, P_1 and P_2 or the triangle with the momenta P_5, P_3 and P_2. The choice of the triangle depends on which triangle will give the fewest number of terms. Assuming that we used the rule for the first triangle we have diagrams that miss either P_4 in the denominator (topology O2), or they miss P_6 in the denominator (topology O4) or they miss P_5 in the denominator (topology O2). These topologies are treated below.

Another derivative topology is the FA topology. It is given by



and it is basically the Benz topology with momentum P_8 missing. We apply again the rule of the triangle and have then either topology O1 left or topology O2.

By far the most complicated topology is the non-planar topology, also called the NO topology:



In this diagram there are no triangles, so we need additional relations. The first step in a reduction is to reduce the dotproducts in the numerator to squares of the 8 momenta P_1 to P_8, the external $Q.Q$ and then either $Q.P_2$ or $P_2.P_5$. The last dotproduct causes problems, but it is inevitable, as there are 10 independent variables before integration and only 9 squares. An important step in the reduction removes this mixed dotproduct (or denominators which effectively simplifies the topology). There are two ways to do this. The first method was introduced in the original mincer paper. We define the integral

$$N'(\alpha_1, \cdots, \alpha_8, \alpha_9) \;=\; \int \frac{d^D p_1\, d^D p_2\, d^D p_3\; (Q.p_2)^{\alpha_9}}{(p_1.p_1)^{\alpha_1} \cdots (p_8.p_8)^{\alpha_8}}.$$

and the recursion is given by

$$\begin{aligned} N'(\alpha_1,\cdots,\alpha_9) \;=\; \Big(& +\frac{1}{2}(D-1-\alpha_1-2\alpha_2-\alpha_3+\alpha_9)\mathbf{9}^- \\ & +\frac{1}{2}\alpha_1\mathbf{1}^+\mathbf{7}^-\mathbf{9}^- - \frac{1}{2}\alpha_1\mathbf{1}^+\mathbf{2}^-\mathbf{9}^- \\ & +\frac{1}{2}\alpha_3\mathbf{3}^+\mathbf{8}^-\mathbf{9}^- - \frac{1}{2}\alpha_3\mathbf{3}^+\mathbf{2}^-\mathbf{9}^- \\ & +\frac{\alpha_9-1}{2}\mathbf{2}^-\mathbf{9}^-\mathbf{9}^-\Big) \Big/ \left(2D-1+\alpha_9-\sum_{i=1}^{8}\alpha_i\right). \end{aligned}$$

The notation here is somewhat schematic: $\mathbf{1}^+\mathbf{7}^-\mathbf{9}^-$ means that in N the α_1 should be incremented by one, the α_7 should be decremented by one and the α_9 should also be lowered by one. This equation can be applied as many times as needed to eliminate all powers of $Q.p_2$. Note that there will never occur negative powers of $Q.p_2$ because the term that could generate such a negative power will have a zero coefficient.

The second scheme comes from Chetyrkin and Tkachov. For this we define the integral

$$N(\alpha_1,\cdots,\alpha_8,\alpha_9) \;=\; \int \frac{d^Dp_1\,d^Dp_2\,d^Dp_3\;(p_2.p_5)^{\alpha_9}}{(p_1.p_1)^{\alpha_1}\cdots(p_8.p_8)^{\alpha_8}}.$$

First we apply the recursion

$$\begin{aligned} N(\alpha_1,\cdots,\alpha_9) \;=\; & +\frac{1}{2}(\mathbf{3^-9^-}-\mathbf{2^-9^-}-\mathbf{8^-9^-}) \\ & +\frac{1}{2(\alpha_6-1)}(D-1-\alpha_1-2\alpha_2-\alpha_8+\alpha_9)\,\mathbf{6^-9^-} \\ & +\frac{\alpha_1}{2(\alpha_6-1)}(\mathbf{1^+6^-7^-9^-}-\mathbf{1^+2^-6^-9^-}) \\ & +\frac{\alpha_8}{2(\alpha_6-1)}(\mathbf{3^-6^-8^+9^-}-\mathbf{2^-6^-8^+9^-}). \end{aligned}$$

till either $\alpha_9 = 0$, or $\alpha_6 = 1$, or one of the other denominators is completely removed. In the first case we have reached our objective of removing the mixed dotproduct and in the last case the topology has been simplified. Hence we can concentrate on the second case. By applying a symmetry transformation we can use it to make α_1, α_3 and α_4 equal to 1 (or remove α_9 or remove one of the other denominators). Hence when we are done with this recursion four of the denominators have only a single power. The next relation is

$$\begin{aligned} & N(1,\alpha_2,1,1,\alpha_5,1,\alpha_7,\alpha_8,\alpha_9) = \\ & \qquad -\tfrac{D-1+\alpha_9-2\alpha_2-\alpha_7-\alpha_8}{2(\alpha_5-1)}\mathbf{5^-9^-}+\tfrac{\alpha_9-1}{2(\alpha_5-1)}\mathbf{2^-5^-9^-9^-} \\ & +\tfrac{\alpha_7}{2(\alpha_5-1)}(\mathbf{2^-5^-7^+9^-}-\mathbf{1^-5^-7^+9^-})+\tfrac{\alpha_8}{2(\alpha_5-1)}(\mathbf{2^-5^-8^+9^-}-\mathbf{3^-5^-8^+9^-}). \end{aligned}$$

This recursion will make $\alpha_5 = 1$ (or again remove α_9 or another denominator) and a symmetric version of it will do the same for α_2. After that we need two more recursions:

$$
\begin{aligned}
&N(1,1,1,1,1,1,\alpha_7,\alpha_8,\alpha_9) = \\
&\quad +\frac{1}{2}(2\,\mathbf{1}^-\mathbf{9}^- + 2\,\mathbf{3}^-\mathbf{9}^- - 2\,\mathbf{2}^-\mathbf{9}^- - \mathbf{7}^-\mathbf{9}^- - Q^2\,\mathbf{9}^-) \\
&\quad +\frac{1}{2}\frac{\alpha_9-1}{\alpha_8-1}(\mathbf{1}^-\mathbf{8}^-\mathbf{9}^-\mathbf{9}^- - \mathbf{2}^-\mathbf{8}^-\mathbf{9}^-\mathbf{9}^- - \mathbf{7}^-\mathbf{8}^-\mathbf{9}^-\mathbf{9}^-) \\
&\quad +\frac{D-1-2\alpha_7-\alpha_8}{2(\alpha_8-1)}\mathbf{8}^-\mathbf{9}^- + \frac{1}{\alpha_8-1}(\mathbf{1}^-\mathbf{2}^+\mathbf{8}^-\mathbf{9}^- - \mathbf{2}^+\mathbf{7}^-\mathbf{8}^-\mathbf{9}^-), \\
&N(1,1,1,1,1,1,\alpha_7,1,\alpha_9) = \\
&\quad +\frac{1}{2}(2\,\mathbf{4}^-\mathbf{9}^- + 2\,\mathbf{6}^-\mathbf{9}^- - 2\,\mathbf{5}^-\mathbf{9}^- - \mathbf{8}^-\mathbf{9}^- - Q^2\,\mathbf{9}^-) \\
&\quad +\frac{1}{2}\frac{\alpha_9-1}{\alpha_7-1}(\mathbf{6}^-\mathbf{7}^-\mathbf{9}^-\mathbf{9}^- - \mathbf{5}^-\mathbf{7}^-\mathbf{9}^-\mathbf{9}^- - \mathbf{7}^-\mathbf{8}^-\mathbf{9}^-\mathbf{9}^-) \\
&\quad +\frac{D-3-\alpha_7}{2(\alpha_7-1)}\mathbf{7}^-\mathbf{9}^- + \frac{1}{\alpha_7-1}(\mathbf{5}^+\mathbf{6}^-\mathbf{7}^-\mathbf{9}^- - \mathbf{5}^+\mathbf{7}^-\mathbf{8}^-\mathbf{9}^-).
\end{aligned}
$$

They will make α_8 and α_7 equal to one. This brings us to the last recursion

$$N(1,1,1,1,1,1,1,1,\alpha_9) = \mathbf{3}^-\mathbf{9}^- - \mathbf{8}^-\mathbf{9}^- + Q^2\frac{1+2\epsilon-\alpha_9}{2(\alpha_9-1-4\epsilon)}\mathbf{9}^- + \frac{Q^2}{\alpha_9-1-4\epsilon}(\mathbf{2}^-\mathbf{3}^+\mathbf{9}^- - \mathbf{3}^+\mathbf{8}^-\mathbf{9}^-).$$

After this all integrals that are left have either a simpler topology or have $\alpha_9 = 0$.

The advantage of the first reduction scheme is its simplicity, while the second scheme is faster because at each step (except for the last) it does not only reduce the value of α_9, but the sum of the other α's is also reduced and hence it leaves simpler integrals behind. The simplicity of the programming made that the first method was used in the original Mincer program. In the FORM version of Mincer we have selected the second method.

After eliminating α_9 we still have to try to remove one of the other α's. First we apply:

$$Q^2N(\alpha_1,\cdots,\alpha_8) = \mathbf{1}^- + \frac{-2D-1+2\alpha_1+2\alpha_2+\alpha_5+\alpha_6+2\alpha_7+\alpha_8}{\alpha_6-1}\mathbf{6}^- + \frac{\alpha_8}{\alpha_6-1}\mathbf{6}^-\mathbf{8}^+(\mathbf{2}^- - \mathbf{3}^-) + \frac{\alpha_5}{\alpha_6-1}\mathbf{6}^-\mathbf{5}^+(\mathbf{7}^- - \mathbf{4}^-).$$

or a symmetric version of it to make α_6, α_1, α_3 and α_4 equal to one (or remove one of the denominators). Note that when α_9 is not present the N' and the N integrals are identical.

Then we use

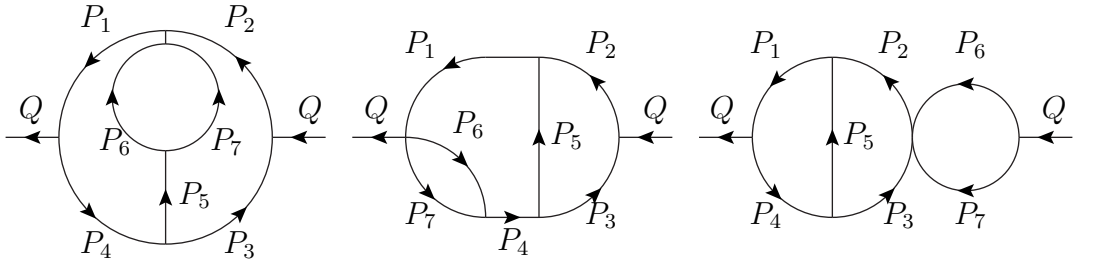
$$\begin{aligned}(\alpha_5-1)(2-D+2\alpha_5)N(1,\alpha_2,1,1,\alpha_5,1,\alpha_7,\alpha_8) = \\ (-2D+4+\beta)Q^2\mathbf{5}^-(+4-3D+2\beta+\mathbf{3}^-\mathbf{4}^+ + \mathbf{1}^-\mathbf{6}^+) \\ +(\alpha_5-1)(\mathbf{7}^-\mathbf{4}^+ + \mathbf{8}^-\mathbf{6}^+) \\ -(3-2D+\beta+\alpha_5)(\mathbf{5}^-\mathbf{4}^+ + \mathbf{5}^-\mathbf{6}^+),\end{aligned}$$

with $\beta = \alpha_2 + \alpha_5 + \alpha_7 + \alpha_8$ to make α_5, α_2, α_8 and α_7 equal to one. At this point we have either killed off at least one denominator in which case the most complicated topologies that are left are of the type FA or BU, or we are faced with the integral

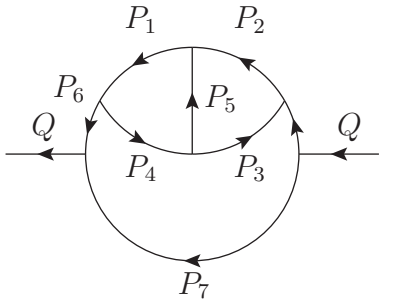
$$N(1,1,1,1,1,1,1,1) \;=\; 20\zeta_5 + \mathcal{O}(\epsilon).$$

This integral was first evaluated by Chetyrkin and Tkachov. Currently many more terms are known in this expansion, but strictly speaking we do not need them in the mincer program.

The leftover topologies have all a one loop two-point function subgraph that can be integrated directly. We still distinguish two groups. Diagrams that still need work at the two loop level (diagrams of type O) and diagrams that do not even need that work anymore (diagrams of type Y). The topologies of type O are



and



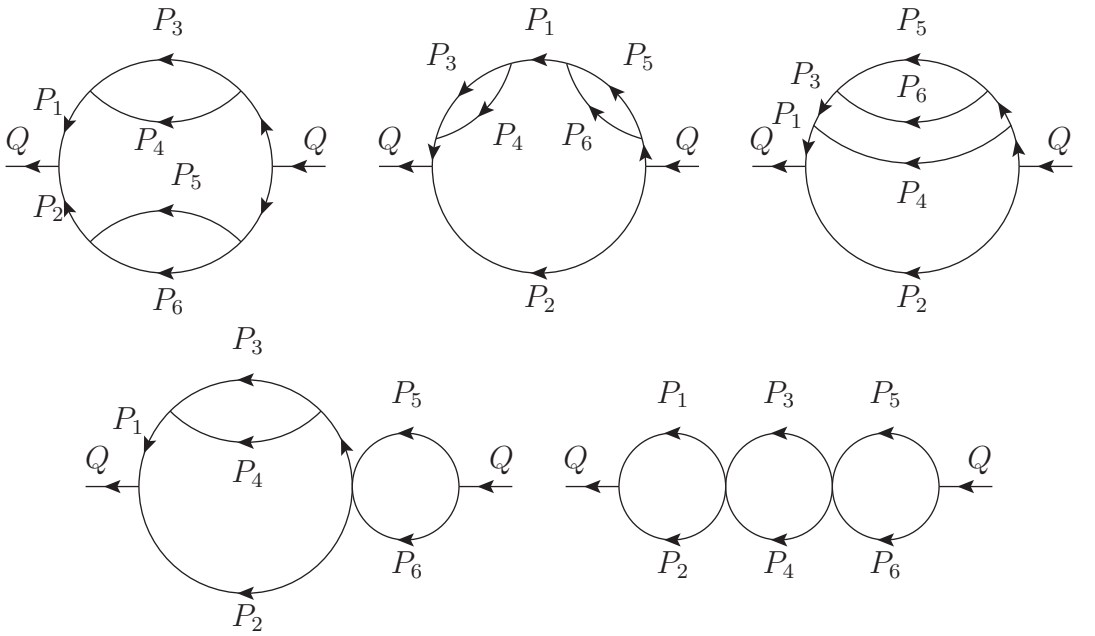
The topologies O1-O3 have a trivial two point subintegral (with momenta P_6 and P_7). The topology O4 requires more care. Based on scaling and Lorentz invariance arguments one can treat the integral over P_6 first. This is particularly simple when there are no dotproducts in the numerator. If there are such dotproducts, dotproducts between momenta in the two loop subgraph are also trivial, dotproducts involving P_7 can be rewritten with $P_7 = Q - P_6$, and $P_6 = P_1 - P_4$. This leaves as only complicated dotproducts the ones involving Q and P_{1-5}.

Let us replace $Q.P_i$ by $xR(P_i)$. Then we can use:

$$\frac{x^n d^D p_6}{(p_6^2)^{x_1}(p_7^2)^{x_2}} = \frac{1}{(Q^2)^{x_1+x_2-D/2}} \sum_{s=0}^{n/2} G(x_1 - s, x_2, n-2s, 0) \sum_{j=0}^{n/2-s} \frac{\Gamma(n+2-\epsilon-2s)\Gamma(n+1-\epsilon-2s-j)(-1)^j}{\Gamma(n+2-\epsilon-s)\Gamma(n+1-\epsilon-2s)\ s!\ j!} \times \left(\frac{Q^2 \Box_R}{4}\right)^{s+j}$$

The d'Alembertian $\Box_R$ acts only on the function R. If it is defined by $\Box_R = \partial^\mu \partial_\mu$ then $\partial^\mu R(p_i) = p_i^\mu$. The evaluation of these derivatives is identical to what is done in the one-loop procedure.

The Y topologies are given by



In the case of these topologies one can directly apply the one-loop equation leaving a diagram of topology T2 or T3.

An example of one of the Y routines:

```
#procedure integy2
*
if ( count(inty2,1) );
  id  p6.p?!{p6} = p1.p-p5.p;
  id  p4.p?!{p4} = p1.p-p3.p;
  id  p2.p?!{p2} = Q.p-p1.p;
endif;
.sort:integy2-1;
if ( count(inty2,1) );
  id  p1.p5 = p5.p5/2+p1.p1/2-p6.p6/2;
  id  p1.p3 = p3.p3/2+p1.p1/2-p4.p4/2;
  id  Q.p1 = p1.p1/2+Q.Q/2-p2.p2/2;
endif;
.sort:integy2-2;
if ( count(inty2,1) );
  if ( ( count(epp2,1) == 0 ) && ( count(p2.p2,1) >= 0 ) ) Discard;
  if ( ( count(epp3,1) == 0 ) && ( count(p3.p3,1) >= 0 ) ) Discard;
```

```
  if ( ( count(epp4,1) == 0 ) && ( count(p4.p4,1) >= 0 ) ) Discard;
endif;
#call IntOne(p5,p6,p1,y2,t2)
*
#endprocedure
```

The final call to IntOne tells to integrate over p5 in a loop that is made up of momenta p5 and p6, while the line outside this loop is p1. Only terms with the marker inty2 will be integrated and the resulting topology is t2 and hence the integrated terms are provided with the marker intt2.

Unfortunately the more complicated topologies require much more code.

At the toplevel the integration routines look like

```
#procedure integral(TOP,par)
*
#switch ‘TOP’
*   ## no :
*   ## be :
*  #[ la :
#case la
    Multiply intla;
    #call integla
    #call integfa
    #call intego1
    #call intego2
    #call intego3
    #call integy4
    #call integy3
    #call integt1
    #call integt2
```

```
    #call integt3
    #call integl1
    Multiply 1/epQ^3/int0;
#break
*  #] la :
*   ## fa :
*   ## bu :
*   ## o1 :
*   ## o2 :
*   ## o3 :
*   ## o4 :
*   ## y1 :
*   ## y2 :
*   ## y3 :
*   ## y4 :
*   ## y5 :
*   ## t1 :
*   ## t2 :
```

```
*   ## t3 :
*   ## l1 :
*   ## tr :
#endswitch
*
#call propagators
#if ( ‘par’ == 1 )
id  GschemeConstants(0,0) = GC0;
id  GschemeConstants(1,0) = GC1;
id  GschemeConstants(2,0) = GC2;
id  BasicNOIntegral = BNO;
id  BasicT1Integral = BT1;
#endif
*
#endprocedure
```

The call to the propagators routine provides some shortcuts for when a diagram contains a higher order propagator. It can give big savings, both in the number of diagrams to be computed, the complexity of the integrals to be computed and the number of terms that are

coming from the substitution of the Feynman rules.

Let us test it a bit again with something bad:

```
#define MSBAR "1"
#include- minceex.h
Off Statistics;
Format nospaces;
.global
L   F = Q.Q^8/p1.p1^2/p2.p2^2/p3.p3^2/p4.p4^2/p5.p5^2
        /p6.p6^2/p7.p7^2/p8.p8^2*Q.p2^2;
#call integral(no,0)
*#call subvalues
*#call expansion(1)
Print +f +s;
.end

F=
   +BasicNOIntegral*rat(4096*ep^10+93184*ep^9+938240*ep^8+5462848*
   ep^7+20209376*ep^6+49173312*ep^5+78744136*ep^4+80631340*ep^3+
   49372908*ep^2+15733320*ep+1843200,ep^2+3*ep+2)
   +GschemeConstants(0,0)*BasicT1Integral*rat(786432*ep^18+24182784*
```

```
ep^17+341596392*ep^16+2928167964*ep^15+16954921542*ep^14+
69845836809*ep^13+209901452148*ep^12+463647927096*ep^11+
746395834995*ep^10+849522664377*ep^9+632771403654*ep^8+
236737319010*ep^7-42491736879*ep^6-93675541320*ep^5-41389099884*
ep^4-6548629680*ep^3,16*ep^8+216*ep^7+1236*ep^6+3906*ep^5+7434*
ep^4+8694*ep^3+6074*ep^2+2304*ep+360)
+GschemeConstants(0,0)^2*GschemeConstants(1,0)*rat(-589824*ep^19-
22069248*ep^18-384110592*ep^17-4118947008*ep^16-30384614016*ep^15
-163007347344*ep^14-655231813872*ep^13-2001476709348*ep^12-
4654835099040*ep^11-8154156766935*ep^10-10449299704155*ep^9-
9147309946377*ep^8-4434802783065*ep^7+243896023899*ep^6+
1873714328823*ep^5+1051949773077*ep^4+104710382361*ep^3-
100963232748*ep^2-28713178188*ep+136080,8*ep^11+156*ep^10+1354*
ep^9+6897*ep^8+22881*ep^7+51840*ep^6+81724*ep^5+89493*ep^4+66581*
ep^3+31974*ep^2+8892*ep+1080)
+GschemeConstants(0,0)^2*GschemeConstants(2,0)*rat(4194304*ep^19+
159383552*ep^18+2824339456*ep^17+30951342080*ep^16+234252156928*
ep^15+1295409152000*ep^14+5400777891840*ep^13+17255349485568*
```

```
ep^12+4245874162380 8*ep^11+7997871030732 8*ep^10+1130554123755 52*
ep^9+1147532579530 24*ep^8+7514365594227 2*ep^7+2023913529577 6*ep^6
-1236328809792 0*ep^5-1414736659091 2*ep^4-4869122329472*ep^3-
6239389065 6*ep^2+3325027173 12*ep+5239327104 0,24*ep^11+468*ep^10+
4062*ep^9+20691*ep^8+68643*ep^7+155520*ep^6+245172*ep^5+268479*
ep^4+199743*ep^3+95922*ep^2+26676*ep+3240)
+GschemeConstants(0,0)*GschemeConstants(1,0)*GschemeConstants(2,0
)*rat(-5347737 6*ep^24-2432696320*ep^23-5243742169 6*ep^22-
7118486758 72*ep^21-6810303750480*ep^20-4887078066212 8*ep^19-
2733373266730 88*ep^18-1220205166546754*ep^17-4411362001445873*
ep^16-1302916008723 2780*ep^15-3158143565989 5828*ep^14-
6285480460868 8977*ep^13-1022835358763 60332*ep^12-
1347077224996 8162*ep^11-1409468017311 99762*ep^10-
1134659951749 10478*ep^9-6610015545025 6875*ep^8-2380142403289 6466*
ep^7-1628478603696 138*ep^6+3360332178325 841*ep^5+1811204047215096
*ep^4+3317369568468 96*ep^3-3891216299572 8*ep^2-2599526180232 0*ep-
3143708107200,384*ep^16+10368*ep^15+129312*ep^14+988416*ep^13+
5179464*ep^12+19717992*ep^11+56375790*ep^10+123395508*ep^9+
```

```
      208805490*ep^8+273845232*ep^7+277173954*ep^6+214024356*ep^5+
      123461862*ep^4+51373008*ep^3+14522544*ep^2+2488320*ep+194400)
      ;
  1.75 sec out of 1.76 sec
```

If you look at this output, you will see that we have 5 terms. They represent the 5 integrals that we have to evaluate by different means and currently those means all involve expansions in ϵ. Let us do that here (same program but now expanded):

```
    #define MSBAR "1"
    #include- minceex.h
    Off Statistics;
    Format nospaces;
    .global
    L   F = Q.Q^8/p1.p1^2/p2.p2^2/p3.p3^2/p4.p4^2/p5.p5^2
            /p6.p6^2/p7.p7^2/p8.p8^2*Q.p2^2;
    #call integral(no,0)
    #call subvalues
~~~Answer in MS-bar
    #call expansion(1)
~~~Answer in MS-bar
    Print +f +s;
    .end

   F=
      -368808805411/81000
      -1348/3*ep^-3
```

```
      -42409/9*ep^-2
      -13820447/900*ep^-1
      +18432000*z5
      -36312616/3*z3
      +18131921333431/810000*ep
      +46080000*ep*z6
      +277141200*ep*z5
      -18156645*ep*z4
      -11700571136/45*ep*z3
      +62668800*ep*z3^2
      ;
  3.23 sec out of 3.23 sec
```

As you can see, these expansions also take time, because they have not been programmed in the most efficient way. That is not really needed, because this is always at the end of the program, when the number of terms is limited to 5, or even only after we have added all diagrams (there could be thousands). This is not timing critical.

The constants z3,z4,z5,z6 in the output are so-called harmonic sums:

$$\zeta_m = \sum_{i=1}^{\infty} \frac{1}{i^m}$$

These occur in the power series expansion of the gamma function. The expansions of the master integrals can contain more complicated nested sums, called multiple zeta values. And when the number of loops becomes large enough there can be even more complicated objects. The full class of functions that can occur in such master integrals is not yet known. Especially when masses are involved each year new categories of functions are found. Currently the fashion word is 'elliptic integrals'.

The regular mincer routines work with a fixed power series expansion in ϵ. If one studies the algorithms carefully one sees that one will never have more than 6 powers of $1/\epsilon$. Therefore one never needs more than 6 powers of ϵ in the expansions. And by shifting a bit with powers of ϵ at the moment new poles can be introduced, one can avoid carrying more powers around than necessary. The result is a much faster program:

```
    #define MSBAR "1"
    #include- minceex.h
    Off Statistics;
    Format nospaces;
    .global
    L   F = Q.Q^15/p1.p1^3/p2.p2^3/p3.p3^3/p4.p4^3/p5.p5^3
            /p6.p6^3/p7.p7^3/p8.p8^3*Q.p2^3;
    #call integral(no,0)
    #call subvalues
~~~Answer in MS-bar
    #call expansion(0)
~~~Answer in MS-bar
    Print +f +s;
```

```
  .end

 F=
    -24973420135522816669757627/5334336000
    +48128763/2*ep^-3
    +97098455533/336*ep^-2
    +5386541947602953/4233600*ep^-1
    +1903058507750400*z5
    -12521623875310875*z3
    ;
92.18 sec out of 92.24 sec
```

versus

```
  #include- mincer.h
  Off Statistics;
  Format nospaces;
  .global
  L   F = Q.Q^15/p1.p1^3/p2.p2^3/p3.p3^3/p4.p4^3/p5.p5^3
          /p6.p6^3/p7.p7^3/p8.p8^3*Q.p2^3;
```

```
    Multiply ep^3;
    #call integral(no)
~~~Answer in MS-bar
    Print +f +s;
    .end

   F=
      -24973420135522816669757627/5334336000
      +48128763/2*ep^-3
      +97098455533/336*ep^-2
      +5386541947602953/4233600*ep^-1
      +1903058507750400*z5
      -12521623875310875*z3
      ;
  8.07 sec out of 8.07 sec
```

Note that the extra ϵ^3 is part of the shifting. It is a little weakness in the system that this is needed.

Because mincer has been used for some rather barbaric calculations the tables had to be big. This is why the mincer.h file contains more than 182000 lines. The programs that were used to generate the tables are included. Hence if further extensions are needed it is a relatively small task to extend them.

The minceex.h file consists of more than 6000 lines, although there are quite a few lines that involve manipulations to get physical objects in a form that the library can handle.

The 4-loop program forcer consists of more than 200000 lines. Of course those have not been programmed by hand. Also the recursion schemes like the one for the NO topology in mincer (but now far more complicated) have been derived by computer, albeit hand guided. In principle there are no fixed expansions (there is an option in the polyratfun statement that forces fixed expansions if needed) because the number of poles in ϵ that can develop during the calculation is currently not known in advance, hence the regular mode is to work with rational polynomials in ϵ.

This is the end of the course. I hope that it has given you an idea of what FORM can do and how you go about solving problems with it. If you pay enough attention to details of efficiency you can obtain very fast programs. Once you have reached some proficiency, it might be useful to go through the manual to see roughly what kind of commands/statements/features there are that were not treated in this course. It might give you ideas.

If you run into problems, you can bring up an issue in the github pages of FORM. There are always people who are willing to help you.