Answers

This file contains the answers to the problems for as far as they are not treated in the course itself.

Problem 1: We have to start with a nice notation in which we define the components of the graph. Hence we define a function for the propagators and two functions for the different vertices. These functions should be non-commuting and they should carry the spinor index.

The definitions are

```
Vectors p;
Functions prop,vert1,vert2;
Index mu,i;
Symbols m;

id vert1(i?,mu?) = g_(i,mu);
id vert2(i?,mu?) = g_(i,mu,7_);
id prop(i?,p?,m?) = g_(i,p)+gi_(i)*m;
```

You see that the spinor index is declared indeed as an index, even though it is a number. We use the unit gamma-matrix in the term with the mass to make sure that we never end up with terms that do not contain any gamma-matrices. This is shown here:

```
Index mu,mu1,mu2,mu3,mu4,mu5,mu6;
Local F1 = gi_(1);
```

```
 Local F2 = 1;
 Local F3 = g_(1,mu1,mu2);
 Local F4 = g_(1,mu1,mu2,mu3,mu4);
 Local F5 = g_(1,5_,mu1,mu2,mu3,mu4);
 Local F6 = g_(1,5_,mu1,mu2,mu3,mu4,mu5,mu6);
 Trace4,1;
 Print +s;
 .end

F1 =
    + 4
   ;

F2 =
    + 1
   ;

F3 =
```

```
     + 4*d_(mu1,mu2)
    ;

F4 =
     + 4*d_(mu1,mu2)*d_(mu3,mu4)
     - 4*d_(mu1,mu3)*d_(mu2,mu4)
     + 4*d_(mu1,mu4)*d_(mu2,mu3)
    ;

F5 =
     + 4*e_(mu1,mu2,mu3,mu4)
    ;

F6 =
     + 4*e_(mu1,mu2,mu3,mu4)*d_(mu5,mu6)
     - 4*e_(mu1,mu2,mu3,mu5)*d_(mu4,mu6)
     + 4*e_(mu1,mu2,mu3,mu6)*d_(mu4,mu5)
     + 4*e_(mu1,mu4,mu5,mu6)*d_(mu2,mu3)
```

```
 - 4*e_(mu2,mu4,mu5,mu6)*d_(mu1,mu3)
 + 4*e_(mu3,mu4,mu5,mu6)*d_(mu1,mu2)
;
```

You see that F2 is just one. There was no gamma matrix. As an added exercise, you may try to figure out how it can be that the last trace has only 6 terms, while most other programs have 15 terms there.

With the above building blocks, it is easy to make the program, taking into account that one has to work against the direction of the arrows in the diagram.

```
Vectors p,p1,p2,p3,p4,p5,p6,p7,p8,q1,q2,q3,q4,q5,q6,q7,q8;
Functions prop,vert1,vert2;
Index mu,mu1,mu2,mu3,mu4,mu5,mu6,mu7,mu8,i;
Symbols m,m1,m2,m3,m4,m5,m6,m7,m8;
Local F =
 prop(1,q1,m1)*vert1(1,mu1)*prop(1,q2,m1)*vert2(1,mu2)*
 prop(1,q3,m2)*vert1(1,mu3)*prop(1,q4,m2)*vert2(1,mu4)*
 prop(1,q5,m3)*vert1(1,mu5)*prop(1,q6,m3)*vert2(1,mu6)*
 prop(1,q7,m4)*vert1(1,mu7)*prop(1,q8,m4)*vert2(1,mu8)*
 prop(2,p1,m5)*vert1(2,mu1)*prop(2,p2,m5)*vert2(2,mu2)*
 prop(2,p3,m6)*vert1(2,mu3)*prop(2,p4,m6)*vert2(2,mu4)*
 prop(2,p5,m7)*vert1(2,mu5)*prop(2,p6,m7)*vert2(2,mu6)*
 prop(2,p7,m8)*vert1(2,mu7)*prop(2,p8,m8)*vert2(2,mu8);
id vert1(i?,mu?) = g_(i,mu);
id vert2(i?,mu?) = g_(i,mu,7_);
id prop(i?,p?,m?) = g_(i,p)+gi_(i)*m;
```

```
    .sort

Time =       0.13 sec    Generated terms =        1024
               F         Terms in output =        1024
                         Bytes used      =      122684
    Trace4,1;
    Trace4,2;
    .end

Time =       0.14 sec    Generated terms =         256
               F         Terms in output =         256
                         Bytes used      =       26548
```

As you can see, even though the diagram is extremely complicated, the answer is actually rather compact. Things are different if we reverse the direction of the arrow on one of the spin lines:

```
Vectors p,p1,p2,p3,p4,p5,p6,p7,p8,q1,q2,q3,q4,q5,q6,q7,q8;
Functions prop,vert1,vert2;
Index mu,mu1,mu2,mu3,mu4,mu5,mu6,mu7,mu8,i;
Symbols m,m1,m2,m3,m4,m5,m6,m7,m8;
Local F =
 prop(1,q1,m1)*vert1(1,mu1)*prop(1,q2,m1)*vert2(1,mu2)*
 prop(1,q3,m2)*vert1(1,mu3)*prop(1,q4,m2)*vert2(1,mu4)*
 prop(1,q5,m3)*vert1(1,mu5)*prop(1,q6,m3)*vert2(1,mu6)*
 prop(1,q7,m4)*vert1(1,mu7)*prop(1,q8,m4)*vert2(1,mu8)*
 prop(2,p1,m5)*vert2(2,mu8)*prop(2,p2,m5)*vert1(2,mu7)*
 prop(2,p3,m6)*vert2(2,mu6)*prop(2,p4,m6)*vert1(2,mu5)*
 prop(2,p5,m7)*vert2(2,mu4)*prop(2,p6,m7)*vert1(2,mu3)*
 prop(2,p7,m8)*vert2(2,mu2)*prop(2,p8,m8)*vert1(2,mu1);
id vert1(i?,mu?) = g_(i,mu);
id vert2(i?,mu?) = g_(i,mu,7_);
id prop(i?,p?,m?) = g_(i,p)+gi_(i)*m;
.sort
```

```
Time =       0.13 sec    Generated terms =        1024
               F         Terms in output =        1024
                         Bytes used      =      139068
    Trace4,1;
    Trace4,2;
    .end

Time =       0.25 sec    Generated terms =       82872
               F       1 Terms left      =       71286
                         Bytes used      =     3922984

Time =       0.37 sec    Generated terms =      165733
               F       1 Terms left      =      141535
                         Bytes used      =     7827280

Time =       0.49 sec    Generated terms =      248605
               F       1 Terms left      =      213115
                         Bytes used      =    11745148
```

```
Time =       0.61 sec    Generated terms =     331470
               F       1 Terms left      =     289500
                         Bytes used      =   16038032

Time =       0.73 sec    Generated terms =     414343
               F       1 Terms left      =     358057
                         Bytes used      =   19800208

Time =       0.85 sec    Generated terms =     497214
               F       1 Terms left      =     430128
                         Bytes used      =   23823344

Time =       0.97 sec    Generated terms =     580075
               F       1 Terms left      =     503449
                         Bytes used      =   27884668

Time =       1.09 sec    Generated terms =     662946
```

```
                F       1 Terms left      =     575880
                          Bytes used      =   31864628

Time =       1.21 sec     Generated terms =     745819
                F       1 Terms left      =     648853
                          Bytes used      =   36055464

Time =       1.33 sec     Generated terms =     828682
                F       1 Terms left      =     719668
                          Bytes used      =   39944908

Time =       1.45 sec     Generated terms =     911556
                F       1 Terms left      =     791370
                          Bytes used      =   43825476

Time =       1.57 sec     Generated terms =     994422
                F       1 Terms left      =     865380
                          Bytes used      =   48131036
```

```
Time =       1.70 sec    Generated terms =    1077290
               F       1 Terms left      =     935492
                         Bytes used      =   52049152

Time =       1.82 sec    Generated terms =    1160160
               F       1 Terms left      =    1007784
                         Bytes used      =   55874636

Time =       1.94 sec    Generated terms =    1243025
               F       1 Terms left      =    1081473
                         Bytes used      =   60165548

Time =       2.06 sec    Generated terms =    1325895
               F       1 Terms left      =    1150738
                         Bytes used      =   63994320

Time =       2.18 sec    Generated terms =    1408768
```

```
                F       1 Terms left      =    1222751
                          Bytes used      =   67982988

Time =       2.29 sec     Generated terms =    1485632
                F       5 Terms left      =    1289511
                          Bytes used      =   72950760

Time =       2.40 sec     Generated terms =    1560687
                F      18 Terms left      =    1351093
                          Bytes used      =   77792804

Time =       2.50 sec     Generated terms =    1637673
                F      33 Terms left      =    1413753
                          Bytes used      =   82465260

Time =       2.61 sec     Generated terms =    1716163
                F      65 Terms left      =    1455429
                          Bytes used      =   85749640
```

```
Time =       2.72 sec    Generated terms =    1795070
               F     129 Terms left      =    1507292
                         Bytes used      =   89830280

Time =       2.82 sec    Generated terms =    1873601
               F     133 Terms left      =    1542432
                         Bytes used      =   92632312

Time =       2.92 sec    Generated terms =    1953565
               F     261 Terms left      =    1602084
                         Bytes used      =   97345212

Time =       3.02 sec    Generated terms =    2025644
               F    1024 Terms left      =    1636316
                         Bytes used      =  100433048

Time =       3.50 sec
```

```
                F          Terms active    =    1076943
                           Bytes used      =   76903428

Time =       3.82 sec      Generated terms =    2025644
                F          Terms in output =    1074170
                           Bytes used      =   80583532
```

The problem is to evaluate

$$\Box_P^{10} P.p_1^{10} P.p_2^{10} P.p_3^{10}$$

Of course it may be better to start with lower powers.
We will try various approaches, each time reaching a higher level of sophistication.

One way to take derivatives of products is to work with non-commuting objects. Let us have a look at the following program:

```
Functions f,f1,f2,f3,der;
Symbols x, n;
Local F = f1(0)*f2(0)*f3(0);
Multiply,left,der*der;
repeat id der*f?(n?) = f(n+1)+f(n)*der;
id der = 0;
Print +s;
.end
```

The notation here is that the argument of the functions tell us how many derivatives we took of that function.

This program makes sure that each derivative acts only once on each function. If the functions f_i would be commuting, the loop would become infinite. Once the derivative is all the way on the right it can only act on the coefficient or whatever other constants we have and hence it will become zero.

The result of the program is:

```
    Functions f,f1,f2,f3,der;
    Symbols x, n;
    Local F = f1(0)*f2(0)*f3(0);
    Multiply,left,der*der;
    repeat id der*f?(n?) = f(n+1)+f(n)*der;
    id der = 0;
    Print +s;
    .end

Time =       0.00 sec    Generated terms =          9
               F         Terms in output =          6
                         Bytes used      =        340

   F =
       + f1(0)*f2(0)*f3(2)
       + 2*f1(0)*f2(1)*f3(1)
       + f1(0)*f2(2)*f3(0)
       + 2*f1(1)*f2(0)*f3(1)
       + 2*f1(1)*f2(1)*f3(0)
       + f1(2)*f2(0)*f3(0)
      ;
```

We notice already that some terms get generated twice. This becomes worse with more functions and/or higher derivatives:

```
Functions f,f1,f2,f3,der;
Symbols x, n;
Local F = f1(0)*f1(0)*f2(0)*f2(0)*f3(0)*f3(0);
Multiply,left,der^5;
repeat id der*f?(n?) = f(n+1)+f(n)*der;
id der = 0;
.end
```

```
Time =       0.07 sec    Generated terms =       7776
               F         Terms in output =        252
                         Bytes used      =      15284
```

Considering that each d'Alembertian stands for two derivatives, it becomes clear that we have a rather formidable task to keep things manageable.

Let us first have a look at a single d'Alembertian, because already there we have to worry about the Lorenz indices.

```
    Vector P,p,p1,p2,p3;
    Function dal,D,dot;
    Index mu,nu;
    L   F = dot(P,p1)*dot(P,p2)*dot(P,p3);
    multiply,left,dal(P);
    id  dal(P) = D(P,mu)*D(P,mu);
    repeat;
        id  D(P,nu?)*dot(P,p?) = p(nu)+dot(P,p)*D(P,nu);
    endrepeat;
    id  D(P,mu?) = 0;
    id  dot(mu?,nu?) = d_(mu,nu);
    Print +f +s;
    .end

Time =       0.00 sec    Generated terms =          6
               F         Terms in output =          3
```

```
                Bytes used      =         132
```

```
F =
      + 2*P.p1*p2.p3
      + 2*P.p2*p1.p3
      + 2*P.p3*p1.p2
     ;
```

Here we have done something similar, but the d'Alembertian is now two derivatives with a contracted index. We represent the dotproducts by a non-commuting function named dot to make life easier.

The substitution with wildcard indices will also work when there is a vector in that location, because then FORM assumes that there must have been an index which was contracted with the index of that vector (Schoonschip notation).

With higher derivatives we have to be more careful. It would be very tempting to write the following program:

```
    Vector P,p,p1,p2,p3;
    Function dal,D,dot;
    Index mu,nu;
    L   F = dot(P,p1)^2*dot(P,p2)^2*dot(P,p3)^2;
    multiply,left,dal(P)^2;
    id  dal(P) = D(P,mu)*D(P,mu);
    repeat;
        id  D(P,nu?)*dot(P,p?) = p(nu)+dot(P,p)*D(P,nu);
    endrepeat;
    id  D(P,mu?) = 0;
    id  dot(mu?,nu?) = d_(mu,nu);
    Print +f +s;
    .end

Time =       0.00 sec    Generated terms =         212
               F         Terms in output =          12
```

```
                    Bytes used      =         644

F =
     + 53*P.p1*P.p2*p1.p2*p3.p3
     + 43*P.p1*P.p2*p1.p3*p2.p3
     + 76*P.p1*P.p3*p1.p2*p2.p3
     + 20*P.p1*P.p3*p1.p3*p2.p2
     + 13*P.p1^2*p2.p2*p3.p3
     + 11*P.p1^2*p2.p3^2
     + 58*P.p2*P.p3*p1.p1*p2.p3
     + 38*P.p2*P.p3*p1.p2*p1.p3
     + 16*P.p2^2*p1.p1*p3.p3
     + 8*P.p2^2*p1.p3^2
     + 13*P.p3^2*p1.p1*p2.p2
     + 11*P.p3^2*p1.p2^2
    ;
```

but unfortunately this answer is wrong. The reason is that suddenly we have 4 indices μ and the contractions are not always what they should be. Hence a proper program would be;

```
    Vector P,p,p1,p2,p3;
    Function dal,D,dot;
    Index mu,nu;
    L   F = P.p1^2*P.p2^2*P.p3^2;
    #do i = 1,2
    id  P.p? = dot(P,p);
    multiply,left,dal(P);
    id  dal(P) = D(P,mu)*D(P,mu);
    repeat;
        id  D(P,nu?)*dot(P,p?) = p(nu)+dot(P,p)*D(P,nu);
    endrepeat;
    id  D(P,mu?) = 0;
    id  dot(mu?,nu?) = d_(mu,nu);
    #enddo
    Print +f +s;
    .end

Time =       0.00 sec    Generated terms =        258
```

```
               F          Terms in output =          12
                          Bytes used      =         644

F =

     + 32*P.p1*P.p2*p1.p2*p3.p3
     + 64*P.p1*P.p2*p1.p3*p2.p3
     + 64*P.p1*P.p3*p1.p2*p2.p3
     + 32*P.p1*P.p3*p1.p3*p2.p2
     + 8*P.p1^2*p2.p2*p3.p3
     + 16*P.p1^2*p2.p3^2
     + 32*P.p2*P.p3*p1.p1*p2.p3
     + 64*P.p2*P.p3*p1.p2*p1.p3
     + 8*P.p2^2*p1.p1*p3.p3
     + 16*P.p2^2*p1.p3^2
     + 8*P.p3^2*p1.p1*p2.p2
     + 16*P.p3^2*p1.p2^2
    ;
```

Clearly this is already much more complicated. Note that we have to get rid of the contracted indices, before we introduce the next d'Alembertian.

We can make the program at least a bit more economical by putting a .sort inside the loop:

```
#define MAX "2"
Vector P,p,p1,p2,p3;
Function dal,D,dot;
Index mu,nu;
L   F = P.p1^‘MAX’*P.p2^‘MAX’*P.p3^‘MAX’;
#do i = 1,‘MAX’
id  P.p? = dot(P,p);
multiply,left,dal(P);
id  dal(P) = D(P,mu)*D(P,mu);
repeat;
    id  D(P,nu?)*dot(P,p?) = p(nu)+dot(P,p)*D(P,nu);
endrepeat;
id  D(P,mu?) = 0;
id  dot(mu?,nu?) = d_(mu,nu);
.sort: pass ‘i’;

Time =       0.00 sec    Generated terms =          24
```

```
                F         Terms in output =          6
                   pass 1 Bytes used      =        336
    #enddo

Time =       0.00 sec    Generated terms =         63
                F         Terms in output =         12
                   pass 2 Bytes used      =        644
    .end

Time =       0.00 sec    Generated terms =         12
                F         Terms in output =         12
                          Bytes used      =        644

   F =
       + 32*P.p1*P.p2*p1.p2*p3.p3
       + 64*P.p1*P.p2*p1.p3*p2.p3
       + 64*P.p1*P.p3*p1.p2*p2.p3
       + 32*P.p1*P.p3*p1.p3*p2.p2
```

```
      + 8*P.p1^2*p2.p2*p3.p3
      + 16*P.p1^2*p2.p3^2
      + 32*P.p2*P.p3*p1.p1*p2.p3
      + 64*P.p2*P.p3*p1.p2*p1.p3
      + 8*P.p2^2*p1.p1*p3.p3
      + 16*P.p2^2*p1.p3^2
      + 8*P.p3^2*p1.p1*p2.p2
      + 16*P.p3^2*p1.p2^2
     ;
  0.00 sec out of 0.00 sec
```

You can see that we get the same answer, but the number of terms generated is less. This way we might try to go to the 10 d'Alembertians of the original problem:

```
Time =       0.02 sec    Generated terms =         492
               F         Terms in output =           6
                  pass 1 Bytes used      =         340
    #enddo

Time =       0.11 sec    Generated terms =        2586
               F         Terms in output =          21
                  pass 2 Bytes used      =        1408

Time =       0.35 sec    Generated terms =        7854
               F         Terms in output =          56
                  pass 3 Bytes used      =        4120

Time =       0.86 sec    Generated terms =       17976
               F         Terms in output =         126
                  pass 4 Bytes used      =        9648

Time =       1.72 sec    Generated terms =       34272
```

```
                F         Terms in output =         252
                   pass 5 Bytes used      =       21556

Time =       2.99 sec    Generated terms =       57177
                F         Terms in output =         453
                   pass 6 Bytes used      =       40056

Time =       4.65 sec    Generated terms =       84114
                F         Terms in output =         735
                   pass 7 Bytes used      =       66744

Time =       6.11 sec    Generated terms =       85216
                F     569 Terms left      =         937
                   pass 8 Bytes used      =       86896

Time =       6.52 sec    Generated terms =      109095
                F     735 Terms left      =        1320
                   pass 8 Bytes used      =      121536
```

```
Time =       6.52 sec    Generated terms =     109095
               F         Terms in output =       1080
                  pass 8 Bytes used      =      99152

Time =       7.77 sec    Generated terms =      83120
               F     708 Terms left       =       1167
                  pass 9 Bytes used      =     115144

Time =       8.38 sec    Generated terms =     124380
               F    1080 Terms left       =       1877
                  pass 9 Bytes used      =     182304

Time =       8.38 sec    Generated terms =     124380
               F         Terms in output =       1435
                  pass 9 Bytes used      =     139336

Time =       9.39 sec    Generated terms =      78668
```

```
                F     892 Terms left      =       1379
                  pass 10 Bytes used      =     140392

Time =       9.96 sec    Generated terms =     123270
                F    1435 Terms left      =       2304
                  pass 10 Bytes used      =     230412

Time =       9.97 sec    Generated terms =     123270
                F         Terms in output =       1701
                  pass 10 Bytes used      =     170044
    .end

Time =       9.97 sec    Generated terms =       1701
                F         Terms in output =       1701
                          Bytes used      =     170044
  9.97 sec out of 9.99 sec
```

It works, but somehow one gets the feeling that this can be done more efficiently.

Let us think a bit. If we have a set of dotproducts, each with one occurrence of P, we could write the other vectors as a string of arguments of a tensor. A program for that would be:

```
#define MAX "4"
Vector P,p1,p2,p3;
Tensor T;
L F = P.p1^`MAX'*P.p2^`MAX'*P.p3^`MAX';
L G = P.P^2*P.p1^`MAX'*P.p2^`MAX'*P.p3^`MAX';
ToTensor,T,P;
Print;
.end
```

```
Time =       0.00 sec    Generated terms =          1
               F         Terms in output =          1
                         Bytes used      =         80

Time =       0.00 sec    Generated terms =          1
               G         Terms in output =          1
                         Bytes used      =         96
```

```
F =
   T(p1,p1,p1,p1,p2,p2,p2,p2,p3,p3,p3,p3);

G =
   T(p1,p1,p1,p1,p2,p2,p2,p2,p3,p3,p3,p3,N1_?,N1_?,N2_?,N2_?);
```

The ToTensor statement needs a vector and a tensor for its arguments and replaces dotproducts that involve the vector by the tensor with the spectator vectors for its arguments. In the case that the vector occurs as a square, we obtain a pair of computer generated indices. In some cases that is not desirable. Hence we have the option nosquare:

```
#define MAX "4"
Vector P,p1,p2,p3;
Tensor T;
L F = P.p1^‘MAX’*P.p2^‘MAX’*P.p3^‘MAX’;
L G = P.P^2*P.p1^‘MAX’*P.p2^‘MAX’*P.p3^‘MAX’;
ToTensor,nosquare,T,P;
Print;
.end
```

```
Time =       0.00 sec    Generated terms =          1
               F         Terms in output =          1
                         Bytes used      =         80

Time =       0.00 sec    Generated terms =          1
               G         Terms in output =          1
                         Bytes used      =        100

   F =
      T(p1,p1,p1,p1,p2,p2,p2,p2,p3,p3,p3,p3);

   G =
      T(p1,p1,p1,p1,p2,p2,p2,p2,p3,p3,p3,p3)*P.P^2;
```

In our example we do not need this.

Next we realize that, due to the fact that each dotproduct had only one power of P, N d'Alembertians take out $2N$ of the arguments of the tensor, but then in all possible ways. This is a matter of combinatorics and FORM has a special function for this, the distrib_ function:

```
#define MAX "4"
Vector P,p1,p2,p3;
Tensor T,dd;
L   F = P.p1^`MAX'*P.p2^`MAX'*P.p3^`MAX';
ToTensor,nosquare,T,P;
id  T(?a) = 2^`MAX'*distrib_(1,2*`MAX',dd,T,?a);
Print +f +s;
.end
```

```
Time =       0.00 sec    Generated terms =          15
               F         Terms in output =          15
                         Bytes used      =        1108

   F =
```

```
 + 16*T(p1,p1,p1,p1)*dd(p2,p2,p2,p2,p3,p3,p3,p3)
 + 256*T(p1,p1,p1,p2)*dd(p1,p2,p2,p2,p3,p3,p3,p3)
 + 256*T(p1,p1,p1,p3)*dd(p1,p2,p2,p2,p2,p3,p3,p3)
 + 576*T(p1,p1,p2,p2)*dd(p1,p1,p2,p2,p3,p3,p3,p3)
 + 1536*T(p1,p1,p2,p3)*dd(p1,p1,p2,p2,p2,p3,p3,p3)
 + 576*T(p1,p1,p3,p3)*dd(p1,p1,p2,p2,p2,p2,p3,p3)
 + 256*T(p1,p2,p2,p2)*dd(p1,p1,p1,p2,p3,p3,p3,p3)
 + 1536*T(p1,p2,p2,p3)*dd(p1,p1,p1,p2,p2,p3,p3,p3)
 + 1536*T(p1,p2,p3,p3)*dd(p1,p1,p1,p2,p2,p2,p3,p3)
 + 256*T(p1,p3,p3,p3)*dd(p1,p1,p1,p2,p2,p2,p2,p3)
 + 16*T(p2,p2,p2,p2)*dd(p1,p1,p1,p1,p3,p3,p3,p3)
 + 256*T(p2,p2,p2,p3)*dd(p1,p1,p1,p1,p2,p3,p3,p3)
 + 576*T(p2,p2,p3,p3)*dd(p1,p1,p1,p1,p2,p2,p3,p3)
 + 256*T(p2,p3,p3,p3)*dd(p1,p1,p1,p1,p2,p2,p2,p3)
 + 16*T(p3,p3,p3,p3)*dd(p1,p1,p1,p1,p2,p2,p2,p2)
;
```

The first argument of this function tells that we want to put something in the function in the third argument (the first function) as opposed to the function in the fourth argument which

will get the remainder (the second function). The second argument tells how many arguments we want to take out. After the four arguments come the arguments we are referring to. Notice that FORM generates exactly the right number of terms and gets the combinatorics right. This helps a lot.

The factor 2^{MAX} is a combinatorics factor that comes from the fact that each d'Alembertian represents two identical derivatives.

The next problem is what to do with this. First the tensor T can now be written back to dotproducts with the ToVector statement that is the opposite of the ToTensor statement. And then we have to decide what to do with the dd tensor. Its arguments should be divided over dotproducts in all possible ways. This can be done with the dd_ function which is the generalized Kronecker delta as in

```
 Indices m1,m2,m3,m4;
 Local F = dd_(m1,m2,m3,m4);
 Print;
 .end
F =
   d_(m1,m2)*d_(m3,m4) + d_(m1,m3)*d_(m2,m4) + d_(m1,m4)*d_(m2,m3);
```

Hence the full program looks now like:

```
#define MAX "4"
Vector P,p1,p2,p3;
Tensor T,dd;
L   F = P.p1^‘MAX’*P.p2^‘MAX’*P.p3^‘MAX’;
ToTensor,nosquare,T,P;
id  T(?a) = 2^‘MAX’*distrib_(1,2*‘MAX’,dd,T,?a);
ToVector,T,P;
id  dd(?a) = dd_(?a);
.end
```

```
Time =       0.00 sec    Generated terms =         69
               F         Terms in output =         69
                         Bytes used      =       5044
```

We see that we have no terms generated twice!

Hence now we can safely set MAX to 10:

```
#define MAX "10"
Vector P,p1,p2,p3;
Tensor T,dd;
L   F = P.p1^`MAX'*P.p2^`MAX'*P.p3^`MAX';
ToTensor,nosquare,T,P;
id  T(?a) = 2^`MAX'*distrib_(1,2*`MAX',dd,T,?a);
ToVector,T,P;
id  dd(?a) = dd_(?a);
.end
```

```
Time =       0.00 sec    Generated terms =       1701
               F         Terms in output =       1701
                         Bytes used      =     157108
```

And indeed we get the same number of terms in the answer as with the previous program that took almost 10 sec on the same computer.

The combinatorics here is astounding. This is seen by looking at some of the output terms:

```
+ 32920473600000*P.p1*P.p2*P.p3^8*p1.p1^4*p1.p2*p2.p2^3*p2.p3^2
+ 4115059200000*P.p1*P.p2*P.p3^8*p1.p1^4*p1.p2*p2.p2^4*p3.p3
+ 8230118400000*P.p1*P.p2*P.p3^8*p1.p1^4*p1.p3*p2.p2^4*p2.p3
+ 60197437440000*P.p1*P.p2*P.p3^8*p1.p2^7*p1.p3^2*p2.p2
+ 30098718720000*P.p1*P.p2*P.p3^8*p1.p2^8*p1.p3*p2.p3
+ 1672151040000*P.p1*P.p2*P.p3^8*p1.p2^9*p3.p3
```

Conclusion: it takes some thinking (= user unfriendly?), but in the end one can program the whole operation in 4 lines and with absolute efficiency.