Potential of FORM 4.0

J.A.M. Vermaseren

Nikhef (and UAM Madrid)

- Introduction
- Factorization
- Rational Polynomials
- New Functions
- Miscellaneous
- What brings the future?

Pre-introduction: Why FORM?

One may wonder why FORM is so important when there are programs like Mathematica and Maple.

We were faced recently with a request for help in programming a gravity problem occurring in Type IIb superstrings. Because this is still unpublished work by others, I will not be very specific. The problem was in a 10-dimensional theory. It needed the evaluation of the Weyl tensor (W) and a tensor (T) with 6 indices. Then it had to evaluate 20 quartic invariants of the types WWWW, WWWT, WWTT, WTTT and TTTT. People had been trying this with the RGTENSOR package in Mathematica. Especially the last category of invariants was very hard on Mathematica. There are 12 indices in 10 dimensions that needed to be contracted.

Most TTTT invariants took 2-3 days each and the last one was never completed. The program had to be halted after a week. The whole effort with Mathematica took the better part of a year. We took three days programming this from scratch in FORM after which the whole project ran in one hour on my laptop. With a few extra optimizations it now runs in 10 minutes. Currently the results are being analyzed.

The above shows a combination of the strengths and weaknesses of FORM. Its strength is that it can be very flexible and very fast. Its weakness is that there are few packages and people like to use packages so that they do not have to think much. With FORM you do have to think. You have to build up expertise, which takes a while, but after that you can do calculations that were impossible before.

```
L   Finv20 = T000000(i1,i2,i3,i4,i5,i6)*
             T100100(i1,i7,i8,i4,i9,i10)*
             T110110(i2,i7,i11,i5,i9,i12)*
             T111111(i3,i8,i11,i6,i10,i12);
Sum i1,1,...,10;
Sum i2,1,...,10;
Sum i3,1,...,10;
.sort
Sum i4,1,...,10;
Sum i5,1,...,10;
Sum i6,1,...,10;
.sort
Sum i7,1,...,10;
Sum i8,1,...,10;
Sum i9,1,...,10;
Sum i10,1,...,10;
.sort
Sum i11,1,...,10;
Sum i12,1,...,10;
.sort
```

This is the whole calculation of that object. The zeroes and ones after the T indicate which indices are upper (1) or lower (0). This takes about 200 seconds on my laptop.

Introduction

Over the past 12 years FORM was available as version 3.0-3.3. This version was significantly more powerful than version 2. But also version 3 needed extensions. During the past few years there were a few opportunities to hire extra people for special contributions and development picked up speed. A number of much needed but very labour intensive projects were undertaken and completed. This marked a good point to clean up the whole program, make it open source and bring it out as a new version.

On March 29 version 4.0 was released. It took much more time than anticipated to prepare this release, because there are quite a few new features and the debugging was a slow procedure. In addition making the source code available and easy installable took much work.

Most work for version 4.0 has been put in by Jan Kuipers, Takahiro Ueda, Jens Vollinga and me. Other people (and also Jens) who worked on FORM in the past have left the field and the current team consists of Jan Kuipers, Takahiro Ueda and me. Jan will leave in the autumn. This means that the speed of advances will be slowed down until there are new opportunities to hire good people.

Jens Vollinga has added a number of very nice features to FORM, including systems independent .sav files, checkpoints (points from which a program can be restarted if it crashes), much documentation and the forum. He also designed much of the open source infrastructure.

Jan Kuipers has made the factorization and polynomial libraries and is currently working on a completely new method for output simplification.

Takahiro Ueda has been hired by Karlsruhe on DFG money with the task to improve the parallelization of FORM. The project is to combine TFORM and ParFORM to make use of clusters of multicore computers. This project is still in its infancy, because his first task was to make ParFORM complete. This has been finished now. He has also taken over the task to manage the open source infrastructure.

This talk will take us through a number of new features and in the end give a few hints about what the future might bring.

The first thing the user will notice is the new header.

```
FORM 4.0 (Jun 25 2012) 64-bits                    Run: Wed Jul  4 14:23:50 2012
```

The mentioning of the 64/32-bits version is for version 4, because we are in a period that many people still use 32-bits computers or operating systems. Other headers are

```
TFORM 4.0 (Jun 25 2012) 64-bits 8 workers         Run: Wed Jul  4 14:24:22 2012
```

or

```
ParFORM 4.0 (Jun 25 2012) 64-bits 8 workers       Run: Wed Jul  4 14:32:45 2012
```

Factorization

The first feature we are going to look at is one that many people have asked for in the past. It should be realized that factorization is a subject that many mathematicians have given attention to. In addition big commercial programs have spent much effort on making good packages for this. Hence one should not expect to outperform other packages. The best model would be to look whether there are library-styled packages under the GNU license that have been created and are maintained by good mathematicians. Unfortunately we could not find any that deal with more than a single variable. The better packages are all closed source and part of a commercial system. This means we had to make our own. But then we could optimize it for what WE anticipate that the use will be (which is not artificial benchmarking).

```
Symbols x,y,z;
CFunction f;
Off Statistics;
Format nospaces;
Local F = f((x+y)*(x*y+4*z^2+3*y*z^4-7*x^2*y)*(x+z));
Print;
.sort

F=
   f(4*y*z^3+3*y^2*z^5+4*x*z^3+4*x*y*z^2+3*x*y*z^5+x*y^2*z+3*x*y^2*
   z^4+4*x^2*z^2+x^2*y*z+3*x^2*y*z^4+x^2*y^2-7*x^2*y^2*z+x^3*y-7*x^3
   *y*z-7*x^3*y^2-7*x^4*y);
FactArg,f;
Print;
.end

F=
   f(y+x,z+x,4*z^2+3*y*z^4+x*y-7*x^2*y);
```

The first example shows the factorization of function arguments. This is probably the most important use in complicated calculations. The `FactArg` statement is natural for this.

Factorization of expressions is a bit more complicated. How to (re)present the results?

```
#define MAX "5"
#define TERMS "6"
#define POW "3"
Symbols a1,...,a`MAX',j;
Off Statistics;
Format NoSpaces;
#do i = 1,`MAX'
Local F`i' = sum_(j,1,`TERMS',random_(`TERMS')*
        <a1^random_(`POW')/a1>*...*<a`MAX'^random_(`POW')/a`MAX'>);
#enddo
Print;
.sort
```

```
F1=
   5*a2*a4+3*a2*a3^2*a4^2*a5+6*a1*a2*a5^2+5*a1*a2*a3^2*a4^2+6*a1*
   a2^2*a3*a4^2+a1^2*a4^2*a5;

F2=
   2*a3^2*a4^2*a5+2*a1*a5+2*a1^2*a3^2*a4^2+2*a1^2*a2^2*a5^2+4*a1^2*
   a2^2*a3*a4^2*a5^2+a1^2*a2^2*a3^2*a4*a5^2;
```

```
  F3=
    6*a3^2*a4*a5^2+5*a2*a3*a4^2+a1*a2*a3^2*a5+a1^2*a3^2*a4^2*a5+4*
    a1^2*a2*a3*a4^2+a1^2*a2^2*a3^2*a4^2;

  F4=
    6*a2^2*a4^2*a5+4*a2^2*a3^2*a4*a5^2+4*a1*a2*a3^2*a4^2+a1^2*a2*a3+6
    *a1^2*a2*a3^2*a5^2+a1^2*a2^2*a5^2;

  F5=
    5*a2*a3^2*a4^2*a5+3*a1*a3^2*a4^2+2*a1*a2*a3*a4^2*a5+a1^2*a3+5*
    a1^2*a2^2*a3^2*a4*a5+a1^2*a2^2*a3^2*a4*a5^2;
   On Statistics;
   Drop;
   Local F = F1*...*F`MAX';
   .sort

Time =       0.02 sec    Generated terms =       7776
               F         Terms in output =       5540
                         Bytes used      =     158532
   Factorize;
   Print;
   .end
```

```
Time =       0.02 sec    Generated terms =       5540
               F         Terms in output =       5540
                         Bytes used      =     158532

Time =       1.65 sec    Generated terms =         33
               F         Terms in output =         33
               factorize Bytes used      =       1684

   F=
       (a3)
      *(a3)
      *(6*a3*a4*a5^2+5*a2*a4^2+a1*a2*a3*a5+a1^2*a3*a4^2*a5+4*a1^2*a2*
        a4^2+a1^2*a2^2*a3*a4^2)
      *(2*a3^2*a4^2*a5+2*a1*a5+2*a1^2*a3^2*a4^2+2*a1^2*a2^2*a5^2+4*a1^2
        *a2^2*a3*a4^2*a5^2+a1^2*a2^2*a3^2*a4*a5^2)
      *(a2)
      *(5*a2*a4+3*a2*a3^2*a4^2*a5+6*a1*a2*a5^2+5*a1*a2*a3^2*a4^2+6*a1*
        a2^2*a3*a4^2+a1^2*a4^2*a5)
      *(6*a2*a4^2*a5+4*a2*a3^2*a4*a5^2+4*a1*a3^2*a4^2+a1^2*a3+6*a1^2*
        a3^2*a5^2+a1^2*a2*a5^2)
      *(5*a2*a3*a4^2*a5+3*a1*a3*a4^2+2*a1*a2*a4^2*a5+a1^2+5*a1^2*a2^2*
        a3*a4*a5+a1^2*a2^2*a3*a4*a5^2);
```

Factorization is considered a 'state' in which the expression exists. It is either factorized or unfactorized. Conversion takes place at the end of the module after the expression has been processed and sorted. Hence we have two output statistics. The second one refers to the factorization procedure. To store the factorized expression we use the FORM bracket system with the built in symbol `factor_`. This allows also a way to refer to the brackets.

The execution time depends critically on how complicated the expression is. If we raise the powers of the variables we can see the effect:

```
#define MAX "5"
#define TERMS "6"
#define POW "4"
Symbols a1,...,a‘MAX’,j;
Off Statistics;
Format NoSpaces;
#do i = 1,‘MAX’
Local F‘i’ = sum_(j,1,‘TERMS’,random_(‘TERMS’)*
        <a1^random_(‘POW’)/a1>*...*<a‘MAX’^random_(‘POW’)/a‘MAX’>);
#enddo
Print;
.sort
```

```
F1=
   6*a1^2*a2*a3^2*a5+6*a1^2*a2*a3^3*a4^2*a5+3*a1^2*a2^2*a3^3*a5^2+
   a1^2*a2^3*a3*a4^3*a5^3+5*a1^3*a2^3*a3*a4^3*a5+5*a1^3*a2^3*a3^2*a4
   *a5^2;

F2=
   a4^2*a5+2*a1*a2^3*a3^3*a5+4*a1^2*a2*a3*a5^2+2*a1^2*a2*a3^3*a4^3*
   a5^3+2*a1^2*a2^2*a4*a5+2*a1^3*a2^2*a3*a4^3*a5;
```

```
  F3=
     5*a2+6*a2^3*a4*a5^3+4*a1*a3*a4^3*a5+a1*a2^2*a4*a5+a1*a2^3*a3^3*
     a4^3*a5+a1^3*a3^3*a4^3;

  F4=
     6*a2*a3^2*a4^2*a5^2+a2^3*a3^3*a4^3*a5+4*a1*a2*a3^3*a4^2*a5^2+4*a1
     *a2^2*a3^3*a4^3*a5+a1^2*a2^2*a3*a5^2+6*a1^3*a2^2*a3*a5;

  F5=
     3*a1*a2^3*a3^3*a4^2+a1^2*a3*a4^2*a5^3+2*a1^2*a2*a4+a1^2*a2*a3*a4*
     a5^2+5*a1^2*a2^2*a4^2+5*a1^3*a3^2*a4^3;
   On Statistics;
   Drop;
   Local F = F1*...*F`MAX';
   .sort

Time =       0.01 sec    Generated terms =       7776
               F         Terms in output =       7125
                         Bytes used      =     213980
   Factorize;
   Print;
   .end
```

```
Time =       0.01 sec    Generated terms =       7125
               F         Terms in output =       7125
                         Bytes used      =     213980

Time =      77.67 sec    Generated terms =         41
               F         Terms in output =         41
               factorize Bytes used      =       2116

   F=
       (a5)
      *(a5)
      *(a5)
      *(a4)
      *(a4^2+2*a1*a2^3*a3^3+4*a1^2*a2*a3*a5+2*a1^2*a2*a3^3*a4^3*a5^2+2*
         a1^2*a2^2*a4+2*a1^3*a2^2*a3*a4^3)
      *(a3)
      *(a3)
      *(6*a3+6*a3^2*a4^2+3*a2*a3^2*a5+a2^2*a4^3*a5^2+5*a1*a2^2*a4^3+5*
         a1*a2^2*a3*a4*a5)
      *(6*a3*a4^2*a5+a2^2*a3^2*a4^3+4*a1*a3^2*a4^2*a5+4*a1*a2*a3^2*a4^3
         +a1^2*a2*a5+6*a1^3*a2)
```

```
*(a2)
*(a2)
*(5*a2+6*a2^3*a4*a5^3+4*a1*a3*a4^3*a5+a1*a2^2*a4*a5+a1*a2^3*a3^3*
   a4^3*a5+a1^3*a3^3*a4^3)
*(3*a2^3*a3^3*a4+a1*a3*a4*a5^3+2*a1*a2+a1*a2*a3*a5^2+5*a1*a2^2*a4
   +5*a1^2*a3^2*a4^2)
*(a1)
*(a1)
*(a1);
```

It is also possible to put expressions in the input in factorized form:

```
    Symbols x,y,z;
    LocalFactor F = (x+1)*(x+y)*(z+2)^2*((x+2)*(y+2));
    Print;
    .sort

Time =       0.00 sec    Generated terms =          12
               F         Terms in output =          12
                         Bytes used      =         448

   F =
         ( 1 + x )
       * ( y + x )
       * ( 2 + z )
       * ( 2 + z )
       * ( 4 + 2*y + 2*x + x*y );

    id x = -y;
    Print;
    .sort

Time =       0.00 sec    Generated terms =          12
               F         Terms in output =           8
                         Bytes used      =         300
```

```
   F =
         ( 1 - y )
       * ( 0 )
       * ( 2 + z )
       * ( 2 + z )
       * ( 4 - y^2 );

    UnFactorize F;
    Print;
    .end

Time =       0.00 sec    Generated terms =          8
               F         Terms in output =          8
                         Bytes used      =        300

Time =       0.00 sec    Generated terms =          2
               F         Terms in output =          2
             unfactorize Bytes used      =         84

Time =       0.00 sec    Generated terms =          0
               F         Terms in output =          0
             unfactorize Bytes used      =          4

   F = 0;
```

This example shows also that if during further processing a factor becomes zero, we still keep the expression and the other factors. If, on the other hand, we unfactorize the expression, we end up with zero of course.

Factorization of $-expressions is yet another case. Here we do not have the bracket system. Neither do we have the possibility to store the factors as arguments. On the other hand, we are not limited by the maximum size of terms.

```
    #define MAX "5"
    #define TERMS "6"
    #define POW "3"
    Symbols a1,...,a`MAX',j;
    Off Statistics;
    Format NoSpaces;
    #do i = 1,`MAX'
    #$v`i' = sum_(j,1,`TERMS',random_(`TERMS')*\
            <a1^random_(`POW')/a1>*...*<a`MAX'^random_(`POW')/a`MAX'>);
    #enddo
    #$V = <$v1>*...*<$v`MAX'>;
    .sort
    #factdollar $V
    #write <> "Factors in $V: `$V[0]'";
Factors in $V: 8
    #do i = 1,`$V[0]'
    #write <> "  Factor `i': %$",$V[`i'];
  Factor 1: a3
    #enddo
```

```
  Factor 2: a3
  Factor 3: 6*a3*a4*a5^2+5*a2*a4^2+a1*a2*a3*a5+a1^2*a3*a4^2*a5+4*a1^2*
a2*a4^2+a1^2*a2^2*a3*a4^2
  Factor 4: 2*a3^2*a4^2*a5+2*a1*a5+2*a1^2*a3^2*a4^2+2*a1^2*a2^2*a5^2+4*
a1^2*a2^2*a3*a4^2*a5^2+a1^2*a2^2*a3^2*a4*a5^2
  Factor 5: a2
  Factor 6: 5*a2*a4+3*a2*a3^2*a4^2*a5+6*a1*a2*a5^2+5*a1*a2*a3^2*a4^2+6*
a1*a2^2*a3*a4^2+a1^2*a4^2*a5
  Factor 7: 6*a2*a4^2*a5+4*a2*a3^2*a4*a5^2+4*a1*a3^2*a4^2+a1^2*a3+6*a1^
2*a3^2*a5^2+a1^2*a2*a5^2
  Factor 8: 5*a2*a3*a4^2*a5+3*a1*a3*a4^2+2*a1*a2*a4^2*a5+a1^2+5*a1^2*a2^
2*a3*a4*a5+a1^2*a2^2*a3*a4*a5^2
    .end
  3.25 sec out of 3.25 sec
```

We refer to the factors as if they are array elements. The zero element tells the number of factors.

Of course \$-variables can be used in two ways: during compilation as shown above, and during execution:

```
Symbols x,y,z;
CFunction f;
Off Statistics;
Format nospaces;
Local F = f((x+y)*(x*y+4*z^2+3*y*z^4-7*x^2*y)*(x+z));
Print;
.sort
```

```
F=
   f(4*y*z^3+3*y^2*z^5+4*x*z^3+4*x*y*z^2+3*x*y*z^5+x*y^2*z+3*x*y^2*
   z^4+4*x^2*z^2+x^2*y*z+3*x^2*y*z^4+x^2*y^2-7*x^2*y^2*z+x^3*y-7*x^3
   *y*z-7*x^3*y^2-7*x^4*y);
 id f(x?$v) = 1;
 FactDollar,$v;
 do $i = 1,$v[0];
  Print "        Factor %$ in $v = %$",$i,$v[$i];
  $t = nterms_($v[$i]);
  Print "          There are %$ terms in factor %$",$t,$i;
 enddo;
 .end
     Factor 1 in $v = z+x
       There are 2 terms in factor 1
     Factor 2 in $v = 4*z^2+3*y*z^4+x*y-7*x^2*y
       There are 4 terms in factor 2
     Factor 3 in $v = y+x
       There are 2 terms in factor 3
```

Here we need an extra supporting facility: the do loop during execution. Its variable is a $-variable.

Internally the factorization algorithms work only with symbols and numbers. Yet we may use other objects as well. Form will replace them temporarily by an internal set of symbols, called the "extra symbols". Then, after factorization these are replaced back. Hence the following example works properly.

```
Symbols x,y,z;
CFunction f,g;
Off Statistics;
Format nospaces;
Local F = f((x+y)*(x+g(z))*(x*y+4*z^2+3*g(y)*z^4-7*x^2*y));
Print;
.sort

F=
   f(4*x*y*z^2+4*x^2*z^2+x^2*y^2+x^3*y-7*x^3*y^2-7*x^4*y+3*g(y)*x*y*
   z^4+3*g(y)*x^2*z^4+3*g(y)*g(z)*y*z^4+3*g(y)*g(z)*x*z^4+4*g(z)*y*
   z^2+4*g(z)*x*z^2+g(z)*x*y^2+g(z)*x^2*y-7*g(z)*x^2*y^2-7*g(z)*x^3*
   y);
```

```
id f(x?$v) = 1;
FactDollar,$v;
do $i = 1,$v[0];
 Print "        Factor %$ in $v = %$",$i,$v[$i];
 $t = nterms_($v[$i]);
 Print "          There are %$ terms in factor %$",$t,$i;
enddo;
.end
    Factor 1 in $v = 4*z^2+x*y-7*x^2*y+3*g(y)*z^4
      There are 4 terms in factor 1
    Factor 2 in $v = y+x
      There are 2 terms in factor 2
    Factor 3 in $v = x+g(z)
      There are 2 terms in factor 3
```

There are more things that can be said about the factorization, but the talks is supposed to be finite in time.

Rational Polynomials

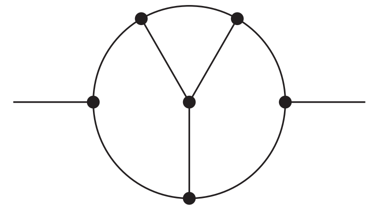
Another important thing that was missing, was the capability to deal with rational polynomials. This has even led to the introduction of the external channels to use other programs like **FERMAT** for this purpose. It would have been nice to have **FERMAT** in the form of a library, like zlib (compression) or the GMP (for multiprecision calculations), but that was not to be. Now we have our own capabilities.

```
    Symbols x,y,z,a,b;
    CFunction rat;
    Format Nospaces;
    Local F = a*rat(x+1,y+1)+a*rat(x+z,y+z)+a*rat(y+z,y-z);
    Print;
    .sort

Time =       0.00 sec    Generated terms =          3
              F          Terms in output =          3
                         Bytes used      =        476
   F=
      rat(1+x,1+y)*a+rat(z+y,-z+y)*a+rat(z+x,z+y)*a;
    PolyRatFun rat;
    Print;
    .end

Time =       0.00 sec    Generated terms =          3
              F          Terms in output =          1
                         Bytes used      =        584
   F=
      a*rat(2*x*y^2-x*y*z+x*y-x*z^2-x*z+y^3+3*y^2*z+2*y^2+3*y*z-z^2,y^3
      +y^2-y*z^2-z^2);
```

$$a\frac{x+1}{y+1}+a\frac{x+z}{y+z}+a\frac{y+z}{y-z} \;\rightarrow\; a\frac{2xy^2-xyz+xy-xz^2-xz+y^3+3y^2z+2y^2+3yz-z^2}{y^3+y^2-yz^2-z^2}$$

Like many things in FORM this is of course limited to the maximum size of the terms. If this turns out to be a limitation, there are usually other ways to attack the problem. In that case the numerators and denominators are very big expressions and it is better to store them in separate expressions or $-variables. These then can be used in the new functions **gcd_**, **div_**, **rem_** to obtain results.

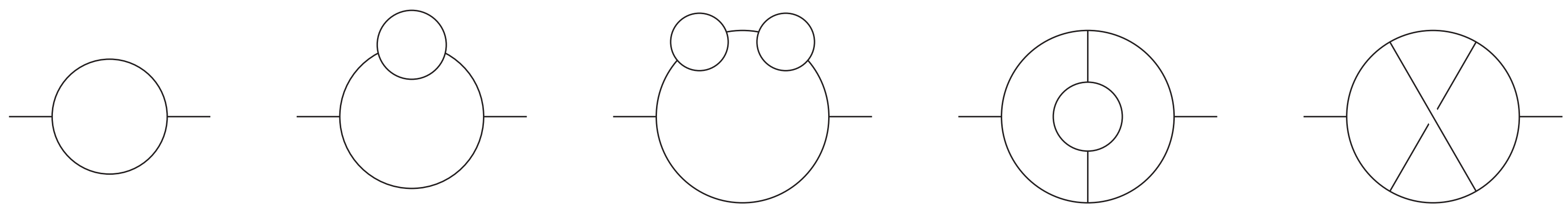
The first application of the rational polynomials was to make a new version of the Mincer library. This version works exact. This means that it does not use expansions in ϵ. All ϵ dependence is put inside the rational polynomial.



```
#include- minceex.h
Off Statistics;
Format nospaces;
.global
L F = Q.Q^2/p1.p1/p2.p2/p3.p3/p4.p4/p5.p5/p6.p6/p7.p7/p8.p8;
#call integral(be,0)
Print +f +s;
.sort

F=
   +GschemeConstants(0,0)*BasicT1Integral*rat(6*ep^3-3*ep^2,2*ep+1)
   +GschemeConstants(0,0)^2*GschemeConstants(1,0)*rat(18*ep^2-15*ep+
   3,2*ep^2+ep)
   +GschemeConstants(0,0)^2*GschemeConstants(2,0)*rat(-128*ep^2+96*
   ep-16,6*ep^2+3*ep)
   +GschemeConstants(0,0)*GschemeConstants(1,0)*GschemeConstants(2,0
   )*rat(84*ep^2-49*ep+7,6*ep^2+3*ep)
   ;
```

```
    #call subvalues
~~~Answer in the Gscheme
    #call expansion(1)
~~~Answer in the Gscheme
    Print +f;
    .end

   F=
      -2*ep^-1*z3-3*z4+12*z3+46*ep*z5+18*ep*z4-32*ep*z3;
```

As is shown above, there are some constants, which are basic one loop integrals with zero, one or two insertions and there is a two loop integral of type T1 with one insertion. There is one more constant which is the basic non-planar integral in three loops.



The first three integrals are known in terms of Γ-functions and can be expanded as far as wanted.

The fourth(T1) integral can be expanded to any precision but that takes more and more time and runs eventually into the limitation that there are relations between the Multiple Zeta Values and these are known only to a certain weight. Enough precision is built in for any practical calculations.

The fifth(NO) integral is more of a problem, but is known to sufficient precision for even 4 loop calculations.

The above program shows that this exact treatment is quite good because we do not have to worry about cancellations of powers of ϵ.

```
    L F1 = GschemeConstants(0,0)^2*GschemeConstants(1,0)*rat(18*ep^2-15*ep+
          3,2*ep^2+ep);
    L F2 = GschemeConstants(0,0)^2*GschemeConstants(2,0)*rat(-128*ep^2+96*
          ep-16,6*ep^2+3*ep);
    L F3 = GschemeConstants(0,0)*GschemeConstants(1,0)*GschemeConstants(2,0)*
          rat(84*ep^2-49*ep+7,6*ep^2+3*ep);
    L F4 = GschemeConstants(0,0)*BasicT1Integral*rat(6*ep^3-3*ep^2,2*ep+1);

    #call subvalues
~~~Answer in the Gscheme
    #call expansion(1)
~~~Answer in the Gscheme
    Print +f;
    .end
```

```
F=
   -2*ep^-1*z3-3*z4+12*z3+46*ep*z5+18*ep*z4-32*ep*z3;

F1=
   192+3*ep^-4-18*ep^-3+48*ep^-2-96*ep^-1-18*ep^-1*z3-27*z4+108*z3-
   384*ep-126*ep*z5+162*ep*z4-288*ep*z3;

F2=
   -1024/3-16/3*ep^-4+32*ep^-3-256/3*ep^-2+512/3*ep^-1+256/3*ep^-1*
   z3+128*z4-512*z3+2048/3*ep+1024*ep*z5-768*ep*z4+4096/3*ep*z3;

F3=
   448/3+7/3*ep^-4-14*ep^-3+112/3*ep^-2-224/3*ep^-1-154/3*ep^-1*z3-
   77*z4+308*z3-896/3*ep-546*ep*z5+462*ep*z4-2464/3*ep*z3;

F4=
   -18*ep^-1*z3-27*z4+108*z3-306*ep*z5+162*ep*z4-288*ep*z3;
```

As one can see, there are quite a few terms cancelling between the terms with only one loop constants.

```
#include- minceex.h
Off Statistics;
Format nospaces;
.global
L F = Q.Q^3*Q.p2^2/p1.p1^2/p2.p2^2/p3.p3^2/p4.p4/p5.p5/p6.p6/p7.p7/p8.p8;
#call integral(be,0)
Print +f +s;
.sort
F=
   +GschemeConstants(0,0)*BasicT1Integral*rat(162*ep^8+729*ep^7+1008
   *ep^6+405*ep^5-60*ep^4-72*ep^3-12*ep^2,8*ep^4+44*ep^3+88*ep^2+76*
   ep+24)
   +GschemeConstants(0,0)^2*GschemeConstants(1,0)*rat(144*ep^9+960*
   ep^8+2418*ep^7+3051*ep^6+1620*ep^5-450*ep^4-690*ep^3-99*ep^2+54*
   ep+12,4*ep^7+34*ep^6+118*ep^5+214*ep^4+214*ep^3+112*ep^2+24*ep)
   +GschemeConstants(0,0)^2*GschemeConstants(2,0)*rat(-288*ep^9-2640
   *ep^8-7486*ep^7-9899*ep^6-5723*ep^5+821*ep^4+2179*ep^3+500*ep^2-
   112*ep-32,6*ep^7+51*ep^6+177*ep^5+321*ep^4+321*ep^3+168*ep^2+36*
   ep)
   +GschemeConstants(0,0)*GschemeConstants(1,0)*GschemeConstants(2,0
   )*rat(-1296*ep^10+16308*ep^9+47592*ep^8+43275*ep^7+2601*ep^6-
   20189*ep^5-9321*ep^4+2300*ep^3+1490*ep^2-84*ep-56,96*ep^8+720*
   ep^7+2088*ep^6+2844*ep^5+1584*ep^4-180*ep^3-528*ep^2-144*ep)
   ;
```

```
    #call subvalues
~~~Answer in the Gscheme
    #call expansion(1)
~~~Answer in the Gscheme
    Print +f;
    .end

   F=
      -2903/1296-1/18*ep^-2+125/216*ep^-1-1/3*ep^-1*z3-1/2*z4-5/18*z3+
      28541/7776*ep+23/3*ep*z5-5/12*ep*z4+467/108*ep*z3;
  0.41 sec out of 0.44 sec
```

The “Mincer Exact” package has been added to the FORM distribution.

New Functions

FORM has obtained many new functions. We name them here. Some names are selfevident:

`Random_`	`RanPerm_`	`Div_`	`Rem_`
`Gcd_`	`Inverse_`	`FirstTerm_`	`Prime_`
`ExtEuclidean_`	`MakeRational_`	`NumFactors_`	`Content_`
`ExtraSymbol_`			

A number of these functions are designed for use in future packages, like a package for Gröbner bases. Such bases can often be calculated faster when calculus is over a prime number and in the end the results over several prime number calculations are combined into a result modulus the product of these numbers.

Making a decent Gröbner basis package is a major undertaking. Again the better packages are not available for inclusion as a library and are part of a commercial product. There is much heuristics involved to take shortcuts and all of that is kept secret. This means that one has to develop ones own heuristics. For this reason we have only been experimenting a little bit with Gröbner bases.

```
#-
#include- groebner.h
Off Statistics;
ON HighFirst;
.global
Local Poly1 = x1^2 + x2*x3 - 2;
Local Poly2 = x1^2*x3 + x2^3 - 3;
Local Poly3 = x1*x2 + x3^2 - 5;
#$n = 3;
#write <> "The input polynomials are:"
Print +f;
.sort
#call groebner(Poly,n)
#write <> "The Groebner basis is:"
Print +f;
.end
```

The above shows what this should look like from the users perspective. The result of this program is:

```
    #-
The input polynomials are:

   Poly1 =
      x1^2 + x2*x3 - 2;

   Poly2 =
      x1^2*x3 + x2^3 - 3;

   Poly3 =
      x1*x2 + x3^2 - 5;
```

```
The Groebner basis is:

  Poly1 =
    4093136817253*x1 - 999056107380*x3^9 + 3162784725684*x3^8 +
    15617604960159*x3^7 - 52374677099676*x3^6 - 78004955176188*x3^5 +
    303405612909504*x3^4 + 117232980911431*x3^3 - 685255923260685*x3^2 -
    3397254300818*x3 + 498469518662424;

  Poly2 =
    30*x3^10 - 9*x3^9 - 570*x3^8 + 222*x3^7 + 4173*x3^6 - 1782*x3^5 - 14569*
    x3^4 + 5523*x3^3 + 24357*x3^2 - 5721*x3 - 15553;

  Poly3 =
    372103347023*x2 + 159558175470*x3^9 - 307637902881*x3^8 - 2539999354128*
    x3^7 + 5261771049639*x3^6 + 13722480161265*x3^5 - 31064912209032*x3^4 -
    27045529790992*x3^3 + 69969321110925*x3^2 + 14416898137155*x3 -
    48858412479378;

 0.10 sec out of 0.11 sec
```

One can see here that this can run out of hand rather quickly. Of course the secret is in what is in the library groebner.h. It uses a large number of the new functions.

This is for example a routine that defines an S-polynomial:

```
#procedure Spoly(A,B,S);
*
*   Procedure defines the S-polynomial of the two polynomials A and B.
*   We work with $'s for the intermediate variables because that is faster
*   and that way we have to compute the gcd only once and do the divisions
*   only once.
*
    #$firstA = firstterm_(`A');
    #$firstB = firstterm_(`B');
    #$gcdfirst = gcd_($firstA, $firstB);
    #if ( isnumerical($gcdfirst) )
        Local `S' = 0;
    #else
        #$nfirstA = $firstA/$gcdfirst;
        #$nfirstB = $firstB/$gcdfirst;
        Skip;
        Local `S' = `A'*$nfirstB - `B'*$nfirstA;
    #endif
    .sort
#endprocedure
```

Note the use of the functions **firstterm_** and **gcd_**. Also the new option **isnumerical** in the preprocessor if statement is being used. Another routine:

```
#procedure MakeInteger(IN,OUT)
*
*   Defines the polynomial `OUT' as a multiple of `IN' so that all its
*   coefficients are integer.
*
    #$MkI = content_(`IN');
    #inside $MkI
        $cMkI = coeff_;
    #endinside
    Skip;
    Drop `IN';
    Local `OUT' = `IN'/(`$cMkI');
    .sort
*
#endprocedure
```

Here we use the function **content_** to eventually obtain the GCD of the numerators and the LCM of the denominators.
Now we hope for volunteers to make a good package.

Miscellaneous

The parallel versions TFORM and ParFORM are both fully functional now. Till about a year ago ParFORM was still missing much of the functionality and was also not very portable. This has all been rectified. It is part of the open source distribution. It does however need a proper MPI installation.

One of the complaints in the past was that .sav files of different executables of FORM were incompatible. This meant that a .sav file generated on one computer might not be usable on another. Also new versions would often need extra variables in the .sav file and hence the old files would be useless. Starting with version 4.0 we have made an attempt to solve these problems. The files should now be uniform, even between 32-bits and 64-bits versions. In addition we have left much spare space in the headers to allow for future extensions that would otherwise invalidate old files.

Of course, some files cannot be carried from a 64-bits computer to a 32-bits computer. If we use x^{123456}, the power is more than the maximum power allowed for symbols on 32-bits systems. Similarly one can exceed the total number of different objects used in all expressions together. But those are rather natural limitations and they would be very rare.

Checkpoints are selected points in a FORM program from which the program can be restarted. One can define many such checkpoints but the program will remember only the last one it has encountered. This facility allows the user to restart the program when external causes have halted execution, like a power outage. The user has to define these checkpoints. It is not done automatically.

There is now a forum for FORM users to communicate with each other. To post on the forum one needs to register. This involves answering an easy question and then waiting till one of the moderators approves of the registration. This procedure is needed because of SPAM attacks and organizations having automatic programs for trying to register on forums like this.
The forum is the proper way to report bugs, to ask questions about installation and versions, or to ask help with certain features or techniques.

For all features in Form, TForm and ParForm holds that we have tried our best to make them running flawlessly on systems that are available to us. Yet it is not excluded that on other systems strange things happen. This may be due to errors in the other systems, unanticipated behaviour, or just insufficiently careful programming on our side. Whenever such a thing occurs we ask the user to report the problems by means of the forum.

What brings the future?

The one line answer to this question:

Hopefully something spectacular.

Currently we (mainly Jan) are trying to make a system for rewriting outputs for numerical programs in a way that takes as few operations as possible. For this we are applying a completely new method based on game theory. At the moment we are obtaining exciting results and a paper will be written before Jan leaves.

At the moment I have an ERC grantrequest for a project in which I want to use game theoretical concepts to solve the systems of recursion relations that are encountered in multi-loop calculations of all Mellin moments in either DIS or Drell Yan processes. The idea is to use something called Monte Carlo Tree Search (MCTS) to work ones way through the enormous number of possibilities to combine equations in order to find either one acceptable solution, or to find the best amoung a large number of solutions.

Indications are that the chance of success for this method is quite good.

The automatic derivation of the formulas would eliminate the most time consuming part of this kind of calculations, even if the derivation by means of MCTS costs a significant amount of CPU time.

If one can derive solutions automatically and if there is more than one solution and if the derivation is reasonably fast, one can envision to make solutions that are optimal for a given diagram. This would make the application of the solution much faster.

More should be known later in the year.

THE END

Graphics done directly into PDF with Axodraw version 2.