About FORM

FORM is a symbolic manipulation program that has been designed for large scale formula manipulation. The formulae that can be handled by FORM are in principle only limited in size by the available disk space. In addition FORM is optimized for speed, enabling it to deal with these giant formulae in relatively little time.

Programs like Macsyma, Maple, Mathematica and Reduce contain many ready made algorithms for a variety of operations. The nature of those operations is often of a 'non local' nature. An operation is called non local if it has to combine the input of more than one term at a time. A good example of a non local operation is factorisation. The disadvantage of most non local operations is that they can only work efficiently if all the terms of the expression they are acting upon are inside the physical memory. This puts a natural restriction on the size of such expressions.

Local operations take the terms one after each other and process them to obtain an output expression. Only at some specific moments are these output expressions normalised (sorted, same terms are added, etc.). This sorting operation is currently the only non local operation in FORM and sorting can be executed very efficiently, even when disks are involved.

The lack of some of the popular non local expressions isn't quite as bad as may seem. Large research problems require usually special algorithms before it is possible to obtain an answer. Typical integration problems require fewer than 20 different integrals which can be programmed easily. The one or two integrals that the computer algebra program cannot solve would have to be fed in anyway. The much higher execution speed pays back

handsomely on the programmers investment. One popular mode of operation is to let a computer algebra program do a small version of the problem or provide the integrals, and then let FORM deal with the entire problem, using the partial results of the other program. This shows that FORM is actually complimentary to the computer algebra programs.

Another feature of FORM is its ability to deal with vectors, tensors and indices. There are many built in properties that have been developed over the past 25 years in programs like Ashmedai, Reduce and Schoonschip. In addition there are extra properties that were either missing or considered user unfriendly in these programs.

The core of the FORM operations is the matching of patterns and making substitutions based on such patterns. This system is very versatile and contains types of wildcarding that are entirely new. In addition there are some built in operations and there is a number of built in objects (like theta functions, summation functions, etc.).

The language of FORM has been designed for user friendlyness. It is rather simple to use, especially if the user has studied the tutorial part of the manual first.

FORM runs on most systems as a batch program. Nowadays the advantage of an interactive program isn't as great anymore as it used to be. Small programs can get a decent turn around, and big programs shouldn't be run interactively anyway. This makes it possible to use ones favorite editor for preparing the input. In addition FORM has become much simpler and can easily be ported to a large variety of computers, acting the same on each of them. Currently FORM is available for SUN 4, Apollo (DN3000 and DN10000), Atari ST, VAX (VMS and ultrix), MacIntosh, Gould (NP1 and 9080) and Alliant. More systems will be added to this list. There is an extensive manual.

1.1 Some examples

The first example shows some very trivial use:

```
    Symbols   a,b;
    Local     F = (a+b)^10;
    print;
    .end
```

```
Time =       0.13 sec    Generated terms =         11
                F        Terms left      =         11
                         Bytes used      =        214
   F =
      a^10 + 10*a^9*b + 45*a^8*b^2 + 120*a^7*b^3
      + 210*a^6*b^4 + 252*a^5*b^5 + 210*a^4*b^6 + 120*
      a^3*b^7 + 45*a^2*b^8 + 10*a*b^9 + b^10;
```

The variables in FORM have to be declared. In the above program a and b are declared as regular objects (called symbols). F is a local expression. FORM tells then what it did and prints the answer as we asked for it.

Vectors are used as they are written. If a function has an index which is contracted with the index of a vector we don't write the indices but we write the vector at the position of the index:

```
    Vectors p1,p2,p3,p4,p5,p6;
    Indices m1,m2,m3,m4,m5,m6;
    Local F = e_(m1,m2,m3)*p1(m1)*p2(m2)*p3(m3)
             *e_(m4,m5,m6)*p4(m4)*p5(m5)*p6(m6);
    print;
    .sort
```

```
Time =       0.05 sec    Generated terms =          1
                F        Terms left      =          1
                         Bytes used      =         30
```

```
   F =
      e_(p1,p2,p3)*e_(p4,p5,p6);

    contract;
    print;
    .end

Time =       0.08 sec    Generated terms =          6
                F        Terms left      =          6
                         Bytes used      =        182
   F =
      p1.p4*p2.p5*p3.p6 - p1.p4*p2.p6*p3.p5 - p1.p5*
      p2.p4*p3.p6 + p1.p5*p2.p6*p3.p4 + p1.p6*p2.p4*
      p3.p5 - p1.p6*p2.p5*p3.p4;
```

We see here that FORM does indeed rewrite the index contractions. The tensor `e_` is the built in Levi-Civita tensor which is used for external products. —FORM knows how to contract them into scalar products if there is more than one. This is only done on request.

Finally we have an example of a medium sized expression. It is an evaluation of a trace in a Dirac algebra.

```
    Indices m1,m2,m3,m4,m5,m6,m7,m8,m9,ma,mb,mc,md,me;
    Local F =
      g_(1,m1,m2,m3,m4,m5,m6,m7,m8,m9,ma,mb,mc,md,me);
    Trace4,1;
    .end

Time =      57.33 sec    Generated terms =      31599
                F        Terms in output =      26931
                         Bytes used      =    1077242
```

This program was run on an Apollo DN3500 (68030).