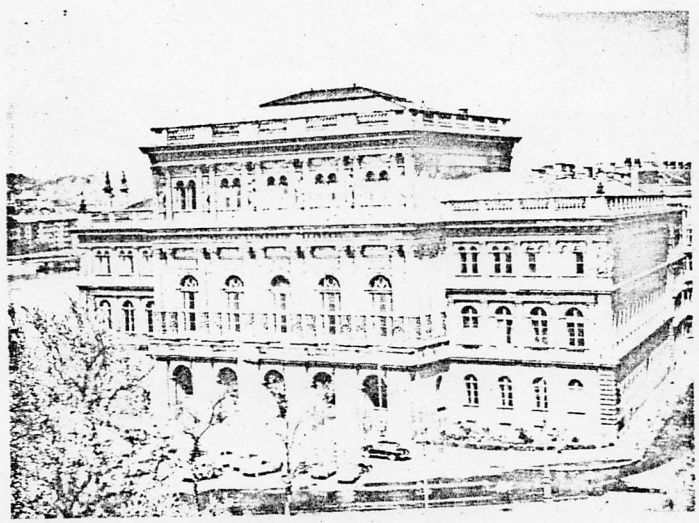
incl. DOMOR

ECODU 20

BUDAPEST
23-25 September, 1975

PROCEEDINGS



Host : HUNGARIAN ACADEMY of SCIENCES

ANALYTICAL COMPUTATIONS AND THE SCHOONSCHIP PROGRAM

H. Strubbe
European Organization for Nuclear Research
Geneva, Switzerland

Session: 31C

I. Introduction

For many mathematics and physics problems one is interested in knowing the general analytical solution, rather than having the numerical answer in a few points. Often the use of analytical methods simplifies numerical calculations. For instance the calculation of

```
R=(N+b)**10 - N**10
```

can become very difficult when done purely numerically if N and b are numbers and N>>b. However, applying first algebra gives

```
R=10*N**9*b + smaller terms
```

and now the numerical calculation is easy.

Unfortunately, many computer users seem to consider their computer only as a "big pocket calculator". They keep on doing their analytical calculations by hand, using a pile of sheets of paper, while making at least one error per page !

The aim of this article is to convince the reader that in many cases he can do his analytical calculations by computer as conveniently as he is used to do his numerical computation by computer.

II. Difference between numerical and analytical calculations

An "algebraic manipulation system" is a computer program that allows one to do calculations with formulae in the same way that the FORTRAN compiler allows one to do calculations with numbers. (In this context the terms : analytical computations, symbolic computations, symbolic manipulations, algebraic manipulations are all assumed to have the same meaning). Let us compare a FORTRAN program with a symbolic program.

FORTRAN program :

```
A=2
B=3
C=1
A1=A+B
A2=(A1+C)*(A1-C)
```

The intermediate results, line after line, are :

```
A=2
B=3
C=1
A1=5
A2=24
```

Note the following points :

- all variables at the right-hand side of a FORTRAN formula have a numerical value before that formula is evaluated.
- each FORTRAN formula, however complicated it is, will reduce to one number in the end.
- interchanging the order of the statements of this program leads to an error message of the type : undefined quantity encountered.

Symbolic program :

```
Z   A2=(A1+C)*(A1-C)
ID, A1=A+B
ID, C=1
ID, B=3
ID, A=2
```

The intermediate results, line after line, are :

```
A2=A1**2-C**2
A2=A**2+2*A*B+B**2-C**2
A2=A**2+2*A*B+B**2-1
A2=A**2+6*A+8
A2=24
```

Note the following points :

- the calculation goes exactly in the reversed order.
- the expression to be evaluated is preceded by a "Z".
- expressions preceded by "ID" are substitutions to be applied on the Z-expression.
- the final result of a calculation is not necessarily one number. The calculation could have terminated after only one substitution.
- the length of the evaluated expression is very hard to predict. This leads to special storage problems.

III. SCHOONSCHIP

The notation of the previous paragraph referred to the SCHOONSCHIP program. It is the major algebra system in use at CERN. It was designed, more than 10 years ago, by M. Veltman. It is set up to do long - but in principle straight forward - analytic calculations. It is very fast in execution and very economical in storage. It can easily deal with expressions of the size of 10^4 to 10^5 terms.

This is achieved by writing it almost entirely in CDC 6000/7000 machine code. The Input/Output is done in FORTRAN. The same program (without any adaptations) is known to run under CERNSCOPE, CDC SCOPE 3.3, 3.4 and 2.0. It uses 20K to 25K (decimal) of memory.

SCHOONSCHIP runs at present at 35 computer installations. Its major use is in the field of high energy physics, but it is sufficiently general to be used for other applications.

IV. Basic statements in SCHOONSCHIP

Most symbolic operations can be thought of as substitutions. For instance, differentiation of powers of Y is done by replacing

```
Y**N  by  N*Y**(N-1)
```

SCHOONSCHIP provides a large set of different types of substitutions, some of which will be described hereafter.

A question in the style of : "Can SCHOONSCHIP integrate ?" gets the same answer as "Can FORTRAN integrate ?". It is : "Yes, if the programmer knows how to do so and gives the appropriate sequence of elementary operations to be performed".

The basic operations, out of which every SCHOONSCHIP calculation has to be built up, are :

- Multiplication and addition of polynomials

```
Input
Z Z=(A+B)**2
Output
Z=A**2+2*A*B+B**2
```

The quantities involved can be algebraic symbols, indices, non-commuting functions or vectors.

- Substitutions

```
Input
Z Z=F(A,B)-A**2+F(X,Y)
ID,F(A,B)=A**2+B**2
Output
Z=B**2+F(X,Y)
```

The replacement is made when the arguments match.

In contrast to this, arguments can be dummies. They are denoted with a + sign behind their names. U+ means : for all values of U.

```
Input
Z Z=F((A+B),(A-B))+F(X,Y)
ID, F(U+,V+)=U*V
Output
Z=A**2-B**2+X*Y
```

- Pattern matching

```
Input
Z Z=A**4*C*B**2
ID,A**2*C**2*B=A2C2B
ID, A**4*C*B=A4CB
ID, C*B**2=CB2
Output
Z=A**4*CB2
```

A more complicated example :

```
Input
Z Z=A**4*F(2,4)*C**3+A*F(1,3)*C+A**3*F(3,3)*C**4
ID, A**N+*F(N+,M+)*C=AFC(N,M)
Output
Z=A**4*F(2,4)*C**3+AFC(1,3)+A**3*F(3,3)*C**4
```

- Special commands in order to :

Calculate traces
Perform complex conjugation
Make partial fraction decomposition
Compare coefficients of terms
Give a weight to terms
Rearrange function arguments, etc.

- Finally, there exists a set of built-in functions :

Gamma matrices
Spin $\frac{1}{2}$ and spin 3/2 spinors
Summation function ($=\Sigma_{j=k}^{n} f_j$)
Kronecker delta, binomial function, etc.

Example :
Expansion in power series up to tenth order in y of

$$\sin(3y^2) \cdot \left(\sin\left(\frac{4}{3}y\right)\right)^2 \cdot \cos(2y)/y^4$$

could be done as follows :

```
Z EXPAN=FS((3*Y**2))*(FS((4*Y/3)))**2
       *FC((2*Y))/Y**4*A*B
ID, FS(Z+)=Z-Z**3/6+Z**5/120-Z**7/5040
       +Z**9/362880-Z**11/39916800
ID, FC(Z+)=1-Z**2/2+Z**4/24-Z**6/720+Z**8/40320
       -Z**10/3628800
ID, A=Y**(-10)
ID, Y=0
ID, B=Y**10
* BEGIN
```

Remarks :

- The factors A and B are added to make the truncation of the series possible. The statement Y=0 sets all POSITIVE powers of Y to zero.
  N.B. There are more efficient ways to do the truncation of a series, but this is a nicer illustration.

- Output is always given in the format :

```
Name = factors * (term + term + ...)
     + factors * (term + term + ...)
     + ...,
```

where "factors" and "terms" are purely multiplicative; "factors" can be empty.

V. Representation of formulae in a computer

Assume we want to represent the polynomial :

```
2.1*ALFA**3*BETA*CCC**4
+7.3*ALFA**4*BETA**4*CCC**4*Y**2*U*V
```

A possible convention could be to write in memory the following string :

```
2.1,ALFA,3,BETA,1,CCC,4,0,7.3,ALFA,4,BETA,4,CCC,4,Y,2,
U,1,V,1,0,0
```

Rather than using the actual name of each variable, it is sufficient to make a namelist and refer to a variable by its number in the list :

1=ALFA ; 2=BETA ; 3=CCC ; 4=Y ; 5=U ; 6=V

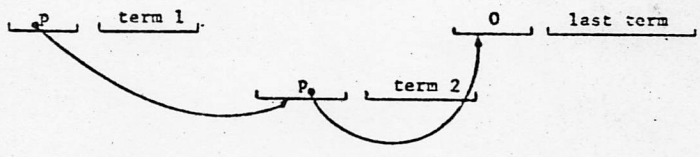
Then the expression becomes :

```
2.1,1,3,2,1,3,4,0,7.3,1,4,2,4,3,4,4,2,5,1,6,1,0,0
```

Typically, the numerical coefficients are double precision floating point numbers. The other quantities can be packed in order to save space. The zero was put in as separator between terms. A double zero indicates the end of the expression. Algorithms to add or multiply 2 such expressions can now easily be written. They involve mainly comparisons of numbers.

In fact, the use of a separator inbetween terms is not very efficient. In order to find the next term, the entire previous term has to be fetched from storage and inspected.

A much more appropriate way is the use of a "POINTER" (symbolized by an arrow). A pointer is nothing but the address of the next term. In other words, we can go from one term to the next, simply by following pointers.

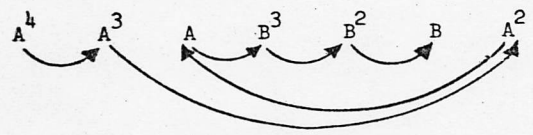


By convention, a pointer with value zero indicates the last term of the expression.

Another advantage of the use of pointers is that it makes the ordering of terms very easy. This is e.g. necessary for the terms in the output space.

Assume we have the string :

Now the term A^2 comes in. Rather than moving all terms in order to allow A^2 to sit inbetween A^3 and A, it is sufficient to change one pointer :



VI. Typical difficulties in algebraic calculations

All the problems inherent of numerical calculations are present in this field as well.

A FORTRAN user will normally not be bothered if 3*(1/3)-1 leads to 10^{-50} as result, but an algebra user will be unhappy if (A+B)*(A-B) leads to A**2-B**2+10^{-50}*A*B. SCHOONSCHIP uses double precision floating point arithmetic. Some systems use infinite precision integer arithmetic. Critical numerical calculations can sometimes be made stable by introducing a few symbolic variables, which only after the major cancellations occured, are set to their numerical value.

We are often not sure whether a long result can be further simplified. Imagine for instance a complicated expression full of powers of trigionometrical functions. To see whether this expression is in fact equal to zero is very hard. To recognize that it is a full square is even a worse problem. This kind of problems is essentially unsolved.

Many calculations generate (intermediate) results that are so long that they overflow the working space in memory. It is an art to transform the calculation to avoid this happening. Such transformations speed up the calculation, sometimes by as much as a factor of 100. Whenever the computer memory is full - after garbage collection is performed - an overflow mechanism could automatically come into action, which uses the disk for further storage. Surprisingly enough, not many algebra systems have this feature built in ! Consequently, an algebra system should be as small as possible in order to allow for a maximal workspace. The ever growing size of CDC system and I/O routines is a major hinderance in this goal.

If the final result is extremely long anyway, it will usually be processed further numerically. Unfortunately CDC FORTRAN compilers do not like expressions with a length of several thousands of cards.

There are many more problems in algebraic calculations. Let me summarize by pointing out that in algebra computations, one has to be more careful about what one is doing than in numerical cases. For instance, requiring the computer to calculate Z=(A+B)**100 by multiplication is hopeless. To ask for the Newton expansion Z=A**100+100*A**99*B+....goes in a fraction of a second.

VII. Why not try it ?

More information on what SCHOONSCHIP can do is found in its manual (1). A description of some applications is given in (2), and details about its internal structure can be found in (3). SCHOONSCHIP can be obtained from the CERN Program Library, if a small tape (400 ft) is mailed to them. On the tape comes then the SCHOONSCHIP program (in source code), its manual and its updating program DOMOR, which is described in the appendix.

Your local mathematics and physics department might wel not be aware of the existence of algebra programs. Availability of SCHOONSCHIP at your computer centre might lead to the solution of problems, previously classified by them as "too difficult" or "too tedious".

REFERENCES

1. H. Strubbe. Manual for SCHOONSCHIP. A CDC 6000/7000 Program for symbolic evaluation of algebraic expressions. Compt. Phys. Commun. 8 (1974) 1.

2. H. Strubbe. Calculations with SCHOONSCHIP. Proceedings of the "Third Colloquium on Advanced Computing Methods in Theoretical Physics", Marseille, June 1973.

3. H. Strubbe. Internal Mechanism of a SCHOONSCHIP calculation. Cern-DD/73/10.

4. H. Strubbe. Conversion of Jobs from 6000 RIOS to 7000 RIOS. Cern Computer Newsletter. 79 (1973) 8.

APPENDIX. The updating program DOMOR

A big program like SCHOONSCHIP (50000 COMPASS statements) requires for its development and debugging a powerful updating program. DOMOR was written for this purpose by M. Veltman. It is very fast, very powerful and easy to use. This will be demonstrated by considering the following (incomplete) example.

```
1  0        PROGRAM LONG
1  1  DEF       CBLANK
1  2        COMMON A(80),B,IBUF1(10),IBUF2(10)
1  3  ENDM
1  4  DEF       INTEG
1  5  MACRO     CBLANK
1  6        INTEGER A,B
1  7  ENDM
1  8  MACRO     INTEG
1  9        -
1 10        -   FORTRAN program
1 11        -
1 12        END

2  0        SUBROUTINE LONGER
2  1  MACRO     INTEG
2  2        -
2  3        -   FORTRAN subroutine
2  4        -
2  5        END

3  0            IDENT SCHOON
3  1            USE //
3  2  MACRO     CBLANK
3  3            USE *
3  4  COPY      BSS 1
3  5            SA1 IBUF1 $ SB1 -9
3  6            ZR X1,COPY $ BX7  X1 $ NO
3  7            SB6 1 $ SA7 IBUF2
3  8  +         SA1 A1+B6 $ SB1 B1+B6 $ BX7 X1 $ SA7 A7+B6
3  9            NG B1,*-1 $ ZR B0,COPY
3 10            END
```

Remarks

- The use of (nested) macro's makes it possible to define only once all common variables. DOMOR will expand the COMMON cards into

  ```
  A    BSS    80
  B    BSS    1
  etc.
  ```

- Several COMPASS instructions can be written on one line. It is very convenient to use a line per instruction word, in particular for the use of "*-1" references and for the identification of dumped code.

- Modifications can be done on parts of cards. For instance to add the variable IBUF3(10) to the COMMON block,

  one uses $ 1 2 /,IBUF3(10)

  to add a comment to a COMPASS line

  one uses $ 3 6 / RETURN IF EMPTY BUFFER

  to copy one word less to IBUF2

  one uses $ 3 5 =-9=-8=

- Obviously, all facilities of a normal updating program (to insert, delete, move, duplicate, ... parts) are available as well. The full description of the DOMOR program is given in (4).