Presentation of Schoonschip

ASSOCIATION FOR COMPUTING MACHINERY · FOUNDED 1947 ·

ACM

SIGSAM Bulletin

A Quarterly Publication of the
Special Interest Group on Symbolic & Algebraic Manipulation

Volume 8, Number 3, August 1974 (Issue Number 31)

PROCEEDINGS OF

EUROSAM '74

ROYAL INSTITUTE OF TECHNOLOGY

STOCKHOLM, SWEDEN

AUGUST 1-2, 1974

Presentation of the SCHOONSCHIP* System

H. Strubbe
CERN, Geneva, Switzerland

1. Introduction

SCHOONSCHIP is the major algebra system in use at CERN. It was designed ten years ago by M. Veltman. It is set up to do long -- but in principle straightforward -- 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.

SCHOONSCHIP runs at present at 25 computer installations. Its major applications are in the field of high-energy physics, but it is sufficiently general to be used for other calculations. At CERN it has been heavily used in high-energy physics: computation of traces of Dirac matrices, renormalization of Yang-Mills fields, multi-dimensional integrations in sixth-order quantum electrodynamics. Moreover, problems in statistics and general relativity were solved with it.

2. General structure of a SCHOONSCHIP program

A SCHOONSCHIP program consists basically of a main expression, and a set of substitutions to be applied on it.

Example:

Computation of

$$\int_0^{A-B} (A+By)*(By+2*y^2)\,dy$$

could be obtained by giving

```
Z INT=(A+B*Y)*(B*Y+2*Y**2)
ID, Y**3=Y**4/4
ID, Y**2=Y**3/3
ID, Y**1=Y**2/2
ID, Y=A-B
* BEGIN
```

Remarks:

- INT is the integrand. The integration itself is done by replacing each $y^n \rightarrow y^{n+1}/(n+1)$. The resulting function is then taken at the point y = A-B. In other words, every operation has to be formulated in terms of elementary substitutions.
- The mechanism of the calculation goes via a scanning and substitution procedure:
  - Z in the first column indicates that the expression which follows has to be examined for substitutions;
  - ID in the first two columns defines the following expression as a substitution;
  - * in the first column is the instruction to start the calculation.
- The intermediate results after each substitution are:

```
INT=A*B*Y+2*A*Y**2+B**2*Y**2+2*B*Y**3
INT=A*B*Y+2*A*Y**2+B**2*Y**2+1/2*B*Y**4
INT=A*B*Y+2/3*A*Y**3+1/3*B**2*Y**3+1/2*B*Y**4
INT=1/2*A*B*Y**2+2/3*A*Y**3+1/3*B**2*Y**3
    +1/2*B*Y**4
INT=-1/6*A*B**3-A*B**4+A**2*B**2+2*A**2*B**3
    -3/2*A**3*B-5/3*A**3*B**2+2/3*A**4
    +1/2*A**4*B+1/6*B**5
```

- Each term of the intermediate results is inspected for the next substitution. If its left-hand side matches, the replacement is made.
- It is important to consider whether the specified substitutions should be applied simultaneously or consecutively.
  - ID as keyword for a substitution implies that the substitution will be applied only after the previous substitution is performed (and its associated expressions in brackets worked out).
  - AL as keyword for a substitution implies that the substitution will be applied simultaneously with the previous substitution. Efficiency imposes the use of AL as often as possible.

Examples:

```
Z E1=Y+Y**2
ID, Y**2=Y**3/3
      intermediate result:  E1=Y+Y**3/3
ID, Y**1=Y**2/2
* BEGIN
      final result       :  E1=Y**2/2+Y**3/3
```

*) SCHOONSCHIP: A CDC 6600 program for symbolic evaluation of algebraic expressions. CERN preprint by M. Veltman, 1967. CERN Program Library R 201.

or

```
Z E1=Y+Y**2
ID, Y**2=Y**3/3
      there is no intermediate result
AL, Y**1=Y**2/2
* BEGIN
      final result       : E1=Y**2/2+Y**3/3
```

or

```
Z E1=Y+Y**2
ID, Y**1=Y**2/2
      there is no intermediate result
AL, Y**2=Y**3/3
* BEGIN
      final result       : E1=Y**2/2+Y**3/3
```

or

```
Z E1=Y+Y**2
ID, Y**1=Y**2/2
      intermediate result: E1=3*Y**2/2
ID, Y**2=Y**3/3
* BEGIN
      final result       : E1=Y**3/2
```

3. Basic statements in SCHOONSCHIP

The elementary 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 ½ 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
AL, FC(Z+)=1-Z**2/2+Z**4/24-Z**6/720+Z**8/40320
      -Z**10/3628800
AL, A=Y**(-10)
ID, Y=0
AL, 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.

- Output is always given in the format:

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

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

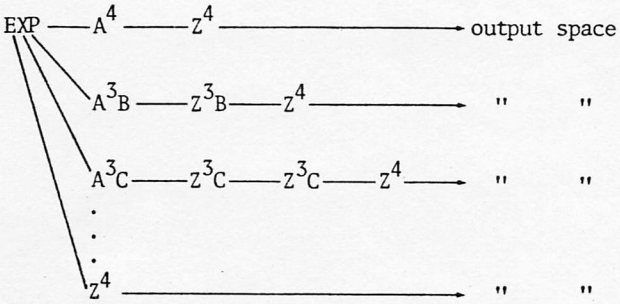
4. The calculation mechanism

Consider the following example:

```
Z EXP=(A+B+....+Y+Z)**4
ID, A=Z
ID, B=Z
---
ID, Y=Z
* BEGIN
```

It would be very inefficient to calculate (A+B+....+Y+Z)**4, then to do the first substitution on this result, then the second substitution on the new result, etc. The reason is that each of those intermediate results is so long that they will have to reside on disk.

The way SCHOONSCHIP performs such a calculation is the following: EXP is generated term by term, by specifying pointers to the appropriate factors. Each term is inspected for substitutions. All substitutions are applied and worked out before the next term is generated. The example goes as follows:



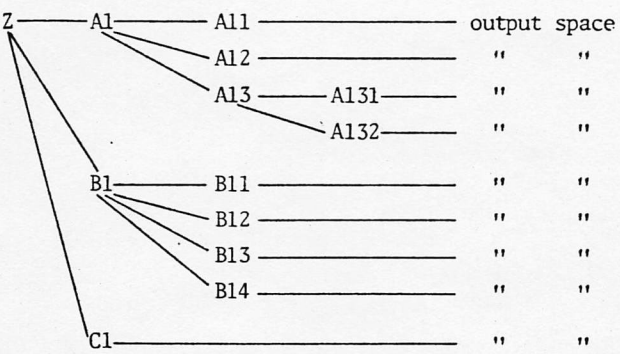
At the moment a term arrives in the output space, a check is made to see whether it adds up with any of the previous terms.

The important point is that we have only one term at the time in memory, rather than long intermediate expressions. The same philosophy applies to new expressions, arising from substitutions. They are also dealt with term by term. This leads roughly to a tree structure.

Example:

```
Z Z=A1+B1+C1
ID, A1=A11+A12+A13
ID, A13=A131+A132
ID, B1=B11+B12+B13+B14
* BEGIN
```

will produce terms for the output in the following order:



All the intermediate terms overwrite each other as soon as they are no longer needed. This calculation mechanism can become very inefficient in certain cases. Consider the following example:

```
Z EXP=(A+B)**10
ID, A=B
* BEGIN
```

The substitution A=B is applied $2^{10} \approx 1000$ times, while the intermediate expression would involve only 11 terms. In such cases, the calculation should be split into two parts, so as to force the intermediate expression to be constructed. The substitution will then be applied 11 times.

```
Z EXP=(A+B)**10
* YEP
ID, A=B
* BEGIN
```

Any card with a * in the first column initiates the evaluation of the Z expression. The keyword on that * card determines what will be done with the result afterwards:

```
* BEGIN:  The result is thrown away.
* NEXT :  The result is stored for later use.
* YEP  :  The result is considered as the Z expres-
          sion for the next set of substitutions.
```

For our example, this means the following:

During the first part A^{10}, A^9B, A^9B, A^8B^2, ..., AB^9, B^{10} (in that order) will be generated and brought to the output space for addition or cancellation. The resulting expression has 11 terms and is transferred from the output space back to the input via the disk. Now the substitution A=B is applied to each term of the (intermediate) expression.

The appropriate use of * YEP statements is crucial for the execution time of a program. The number of substitutions performed depends much on it. However, the amount of additional disk reading and writing has to be considered as well.

5. Memory overflow

No direct limits on the size of input or output expressions are imposed. An overflow mechanism to use the disk comes automatically into action.

How the output overflow works can be seen from the following example.

Assume the output is nearly full (after doing Garbage Collection) and contains A**3 as the only term depending on A. As the calculation proceeds, the following terms are produced:

```
A**4:  can still fit in memory, but occupies the
       last free words;
A**2:  cannot be kept in memory. Goes to the over-
       flow tape (tape 4, usually a disk file);
A**3:  adds up with the A**3 in core;
A**5:  goes to tape 4;
A**2:  goes to tape 4;
----
```

Each term is compared with all terms in the output space. If it does not add up, it is sent to the overflow tape. However, no test is made whether that term adds up with a term already stored on that

tape. In other words, tape 4 will contain in this example:

```
A**2, A**5, A**2
```

All terms on the overflow tape are different from those in the output space. When all terms are generated, the output space can be emptied (e.g. by printing). Now terms are read in from tape 4 and put in the output space, while addition and cancellation can occur. New overflow is written on tape 5. The output space is again emptied, and by repeatedly interchanging the function of tape 4 and tape 5 this mechanism can go on until all terms are printed.

Long input expressions reside on disk and have only one record in core. It is perfectly well possible to multiply two such expressions, the only risk being ... disk overflow!

6. Linking of SCHOONSCHIP programs

Rather than printing and discarding the content of the output space, it is also possible to save it for the next part of the calculation or for a later run.

At the end of a calculation step, one can transfer results from the output back to the input.

Example:

```
Z EXP(U,V)=U*V
KEEP EXP
* NEXT
Z PR=(EXP((A+B),(A-B)))**2
```

Results to be used in a later run can be written onto disk and then catalogued as permanent file. This allows one to split up a long calculation into smaller pieces or to restart it at an earlier point. Doing so pays off in turnaround time, at least in the CERN computer centre. (*N.B.* CERN XJOB = express job: 30K memory and 64 sec 6600 CP time. Turnaround ≈ 15 min.)

This facility has proved to be essential in many calculations, e.g. in multidimensional integrations where the integration scheme is in fact found by trying different substitutions over and over again (typically 100 times).

All the above-mentioned results are kept in the internal SCHOONSCHIP notation, rather than in character code. This eliminates repeated input scan and conversion.

SCHOONSCHIP subprograms -- possibly involving dummy variable names -- can be stored on disk (in character code), and then form a kind of library.

7. Comments on some SCHOONSCHIP facilities

a) SCHOONSCHIP always attempts numerical evaluation of an expression before doing algebra on it. For example, (1+2)**10 is not computed in the same way as (A+B)**10. DX is a built-in function which generates a call to subroutine EXTRA, to be given by the user in FORTRAN.

b) The user can add his own "built-in functions" (called X expressions).

c) DO LOOPS are available. Their format can be seen from the example:

```
DO J=10,1,-1
ID, A**J=A**J/A
* YEP
ENDDO J
```

d) There is no branching possible in a SCHOONSCHIP program. However, this can easily be overcome by using selective substitutions, e.g. IFSMA, IFGRE, IFEQU tests on the coefficient of each term, and appends a symbol to each successful term. COUNT computes the weight N of each term and appends to it, e.g. F(N) or A**N ... . Very complicated pattern matching can then be used to pick out a specific term

```
A**N+*B**N+*C**M+*F(A,B,N+,M+)=...
```

e) The "freezing" of the expression

```
Z=V11*V12*....*(A11+A12+...)
 +V21*V22*....*(A21+A22+...)
 +...
```

leads to

```
Z=V11*V12*....*DF(Z,1)
 +V21*V22*....*DF(Z,2)
 +...
```

Now substitutions on V_{ij} can be done (which may involve DF). DF remains unchanged. At the end DF is again replaced by the A_{ij} values (with the command EXPAND). Appropriate use of this facility can save much execution time.

8. Example of a realistic calculation

The following eigenvalue problem was encountered by B. Schorr at CERN. Mathematical formulation of the problem: to determine the distribution of the statistic of a certain two-dimensional goodness-of-fit test of the von Mises type, the eigenvalues λ of the following integral equations are of great importance:

$$\phi(t_1,t_2) = \lambda \int_0^1\int_0^1 K_1(s_1,s_2;\, t_1,t_2)\phi(s_1,s_2)\, ds_1\, ds_2$$

where

$$K_1(s_1,s_2;\, t_1,t_2) = \min(s_1,t_1)\, \min(s_2,t_2) - s_1 s_2 t_1 t_2 \qquad (1)$$

If

$$K_{n+1}(s_1,s_2;\, t_1,t_2) = \int_0^1\int_0^1 K_n(s_1,s_2;\, u_1,u_2)K_1(u_1,u_2;\, t_1,t_2)\, du_1\, du_2 \qquad (2)$$

for n = 1, 2, ..., it can be shown that

$$L_n = \int_0^1\int_0^1 K_n(s_1,s_2;\ s_1,s_2)\ ds_1\ ds_2$$

$$= \sum_{j=1}^{\infty} \frac{1}{\lambda_j^n}\ , \qquad n = 1, 2, \ldots . \qquad (3)$$

Suppose L_n is known, then λ_j can be calculated.

N.B. min (a,b) = aH(b,a) + bH(a,b), with H being a theta function:

$$H(a,b) = \begin{cases} 1 & \text{when } a > b \\ \frac{1}{2} & \text{when } a = b \\ 0 & \text{when } a < b \end{cases}$$

Remarks:

* This calculation can be performed by repetitively applying the following formulae:

$$\int_0^1 u_1^n H(s_1,u_1) * H(t_1,u_1)\ du_1$$

$$= \frac{s_1^{n+1}}{n+1} + \frac{t_1^{n+1} - s_1^{n+1}}{n+1} * H(s_1,t_1)$$

$$\int_0^1 u_1^n H(s_1,u_1)\ du_1 = \frac{s^{n+1}}{n+1}$$

$$\int_0^1 u_1^n\ du_1 = \frac{1}{n+1}$$

which can be given as substitutions.

* The important point to note is that the H-functions do not commute in SCHOONSCHIP. In order to calculate $K_2(s_1,s_2;\ t_1,t_2)$ we have to integrate terms of the type:

$u_1^2 * u_2^2 * H(s_1,u_1) * H(s_2,u_2) * H(t_1,u_1) * H(t_2,u_2)$

Applying on this the pattern matching substitution:

ID, U1**N+**H(S1,U1)*H(T1,U1)=...

will not lead to a replacement, as the H-functions do not commute. This non-commutativity can be (partly) cancelled by specifying the keywords:

AINBE (allow in between) or
ADISO (allow disorder).

However, in such cases the scanning mechanism becomes much more complicated and time-consuming. Rewriting the program in terms of algebraic symbols -- which automatically commute -- like HS1U1, HT1U1,... speeded up the calculation with a factor of 2.

* The calculation mechanism involved is very easy and straightforward. However, the fact that the final result is a number and not a formula suggests that it could be worth while to investigate the possibility of tackling this problem completely numerically. The algebraic calculation blows up very fast:

K_5 has 700 terms and no longer fits in the input space (3500 words);
K_6 has 1200 terms and no longer fits in the output space (5000 words);
K_{11} has 6500 terms. Owing to overflow of input and output space, 2000 records were written on disk by that time.

The whole calculation (up to order 11) required $\approx$500,000 multiplications of terms and took 288 seconds on a CDC 7600 ($\approx$ 14 times faster than a 6400). Clearly, with this method it is impossible to get much further.

* Fortunately, a completely numerical solution was found by B. Schorr and 100 eigenvalues were calculated in FORTRAN in a few minutes.

* The listing of the SCHOONSCHIP program that reads the last calculated K_n from disk and replaces it by K_{n+1} is given in the appendix.

9. References

Additional information can be found in:

1) Manual of SCHOONSCHIP, H. Strubbe, CERN-DD/74/5.

2) Internal Mechanism of a SCHOONSCHIP calculation, H. Strubbe, CERN-DD/73/10.

APPENDIX

```
C  EXAMPLE. AN EIGENVALUE PROBLEM.

C  THIS PROGRAM COMPUTES A NEW ITERATION FOR  K .

ENTER COMMON

C  THE RESULTS OF THE PREVIOUS RUN ARE ENTERED.
* BEGIN
INDEX N
SYMBOLS U1,U2,S1,S2,T1,T2
FUNCTIONS H,L

C  DEFINITION OF  K1 . SEE FORMULA 1.
Z K1(S1,S2,T1,T2)=S1*S2-S1*S2*T1*T2+S2*T1*H(S1,T1)-S1*S2*H(S1,T1)+
 S1*T2*H(S2,T2)-S1*S2*H(S2,T2)+T1*T2*H(S1,T1)*H(S2,T2)+S1*S2*H(S1,T1)*
 H(S2,T2)-S1*T2*H(S1,T1)*H(S2,T2)-S2*T1*H(S1,T1)*H(S2,T2)

KEEP K1
* NEXT
C  CONSTRUCTION OF THE NEXT  K . SEE FORMULA 2.

NAMES K
Z K(S1,S2,T1,T2)=K(S1,S2,U1,U2)*K1(T1,T2,U1,U2)

ID, AINBE, U1**N+*H(S1,U1)*H(T1,U1)=S1*S1**N/(N+1)+(T1*T1**N-S1*S1**N)*
           H(S1,T1)/(N+1)
AL, AINBE, U2**N+*H(S2,U2)*H(T2,U2)=S2*S2**N/(N+1)+(T2*T2**N-S2*S2**N)*
           H(S2,T2)/(N+1)
AL, AINBE, H(S1,U1)*H(T1,U1)=S1+(T1-S1)*H(S1,T1)
AL, AINBE, H(S2,U2)*H(T2,U2)=S2+(T2-S2)*H(S2,T2)
AL,        U1**N+*H(S1+,U1)=S1*S1**N/(N+1)
AL,        U2**N+*H(S2+,U2)=S2*S2**N/(N+1)
AL,        U1**N+=1/(N+1)
AL,        U2**N+=1/(N+1)
AL,        H(S1+,S2+)=S1

KEEP K1
NPRINT K
* NEXT
C  INTEGRATION OF  K . SEE FORMULA 3.

NAMES K
Z ITER=ITER+1
Z L(S1,S2)=K(S1,S2,S1,S2)

ID,        H(S1+,S2+)=0
AL,        S1**N+=1/(N+1)
AL,        S2**N+=1/(N+1)

* BEGIN
WRITE COMMON

C  THE NEW RESULTS ARE WRITTEN ON DISK TO BE CATALOGUED.
* END
```