STICHTING
MATHEMATISCH CENTRUM
2e BOERHAAVESTRAAT 49
AMSTERDAM

AFDELING TOEGEPASTE WISKUNDE

Report TW 101

FORMULA MANIPULATION in ALGOL 60

(preliminary report)

by

R.P. van de Riet

August 1966

Table of contents

Summary

This report describes two systems, consisting of ALGOL 60 procedures, to be used for formula manipulation.
The first system is able to handle a rather general set of formulae, built up by means of numbers (including complex numbers), algebraic variables, polynomials, the operators: +, -, *, /, ↑ and the function symbols: exp, ln, sin, cos, arctan, sqrt.
The main capabilities of this system are:

1. Simplification, which is illustrated best by the examples:

$(x^2 - 1)/(x^2 + 2x + 1)$ is simplified to $(-\frac{1}{2}x + \frac{1}{2})/(-\frac{1}{2}x - \frac{1}{2})$ and

$\sin^2(x + y) + \cos^2(y + x)$ is simplified to 1.

2. Polynomial manipulation; the polynomials are treated as truncated power series with arbitrary formulae as coefficients.

Besides these, there are less important capabilities such as:

1. Differentiation,
2. Complex conjugation,
3. Substitution,
4. Finding the coefficients of certain products of algebraic variables occurring in a formula,
5. Outputting a formula,
6. Inputting a formula, with some simple instructions, from input tape.

The other system can handle formulae which are built up by means of algebraic variables and the operators + and *.
Its capabilities are only differentiation and outputting.

1. Introduction

The electronic digital computer, already intensively used in numerical mathematics, will be used more and more in other branches of mathematics. One of these branches is elementary algebra, used throughout mathematics and physics.
It occurs frequently that one has to do quite an amount of long and tedious, but elementary algebraic calculations, to solve some problem in, say, mathematical physics.
It is for these calculations, henceforth called "formula manipulation", that the computer may become usefull.
Two possible systems to instruct the computer to manipulate formulae are described in this report.

Before these systems are discussed it is necessary to elucidate the term "formula manipulation".
From Jean Sammet [2] the following rather rough definition is borrowed:

"Formula manipulation means the use of a computer to operate on mathematical expressions in which not all the variables are replaced by numbers and in which some meaningfull mathematical operation is to be done...".

In this report the mathematical expressions are particular arithmetic expressions (see the ALGOL 60 report [1]), which, in the sequel, are called formulae; the form of the formulae is described in sections 2 and 3. The meaningfull mathematical operations are for example: differentiation, simplification, complex conjugation, finding a common divisor of two formulae, substitution.

The two systems, with which one can manipulate formulae on a computer, consist of sets of ALGOL 60 procedures.
One system, described in section 2, manipulates formulae of a very simple form and is therefore called the simple system.
The formulae of the simple system may be built up with algebraic variables and the operators + and *.
The capabilities of this system are differentiation and outputting.

The other system, described in the rest of the report, manipulates formulae which may be built up with numbers (including complex numbers), algebraic variables, polynomials (i.e. truncated power series with arbitrary formulae as coefficients), trigonometric and logarithmic functions and the operators +, -, ×, /, ↑.
This system, called the general system, has the following capabilities: simplification, polynomial manipulation, differentiation, complex conjugation, substitution, finding the coefficients of certain products of algebraic variables in a formula, outputting and inputting formulae from input tape.

The reason to describe both systems is mainly that the simple system serves as an introduction to the general system; a program using the simple system is completely explained.
Moreover, due to the fact that the simple system is much shorter and considerably less time consuming than the general system, it may be more usefull than the general system in all those formula manipulations, where the formulae have the simple form as defined above. Even if the formulae are more complicated, e.g. as constituants also numbers and as operators also the integral power is allowed, but in which the division operator does not occur, then the simple system can be very usefull since it can easily be extended into a system which handles these formulae and which has e.g. the capability of simplification.
In fact, the combination of division and simplification in the general system is one of the reasons why it is so lengthy and time consuming.

Section 4 gives the user information necessary for using the general system as a "black box".
The way in which formulae are represented in the computer is described in sections 5 and 6.
Section 7 describes the procedures for the basic operations, such as storing a formula, extracting information of stored formulae and erasing formulae.
The procedures for the basic arithmetical operation and for the trigonometric and logarithmic functions are described in sections 8 and 9.

Simplification algorithms are treated in section 10, while algorithms for the quotient and the greatest common divisor are described in section 11. Some remarks on storage allocation are contained in section 12. Section 13 contains supplementary equipment in the form of ALGOL 60 procedures for differentiation, complex conjugation, substitution, and finding the coefficients of certain products of algebraic variables in a formula.
The last sections 14 and 15 are devoted to outputting formulae and inputting formulae from input tape.
A program is reproduced which reads input tape on which formulae in polish notation and instructions may be put, as e.g.
f := * + a, b - a, b; g := SIMPLIFY(f); OUTPUT(g);

There exists already quite a number of publications on the subject of formula manipulation, almost all reviewed in the already quoted paper [2] and in the papers [3,4] of the same author.
Several systems for formula manipulation are developed, such as FORMAC [5] or Formula ALGOL [6], to mention two important and general systems. All these systems have, however, the disadvantage that they require special translators.
Another disadvantage is that a user, who has access to a computer for which one of these systems is written, has to know machine code if the system has to be somewhat modified.

Since both systems described in this report are completely written in ALGOL 60, everyone who has access to a computer with an ALGOL compiler can not only use them, but can also write some modified system which can solve his special problem.
Furthermore, a consequence of writing the systems in ALGOL 60 is their unambiguous definitions.

The disadvantages of in particular the general system should also be mentioned: the system needs much storage space, it needs much time and it needs much space for storing formulae.

If one does not need the full generality of the system one may modify the system so as to gain storage space and time. Another way of obtaining more efficiency is to write a machine-code procedure for storing formulae in a compact form.

For applications of preliminary versions of the formula manipulating system, the reader is referred to [7,8,9] in which programs are discussed for obtaining respectively:

1. the second order solution of the (non-linear) Navier Stokes equations coupled with the (non-linear) diffusion equation, together with complicated boundary conditions at a free surface of a liquid;
2. the Taylor expansion of functions defined by the so-called Cauchy problem, i.e. a set of partial differential equations (implicit in the highest derivatives), with initial values;
3. the asymptotic expansion of rather arbitrary integrals by means of the method of steepest descent (the saddle point method).

The procedures constituting the simple system are reproduced compactly, whereas the reproductions of the procedures constituting the general system are scattered throughout the text; they are preceded by the sentence: "This is a part of the general system".

Some critical remarks leading to a considerable improvement of the text of, in particular, sections 1, 2, and 3, are due to professor A. van Wijngaarden.

The programs reproduced in this report were all run and tested for ALGOL correctness on the Electrologica X8 computer, and some of them on the Electrologica X1 computer, both of the Mathematical Centre. Valuable assistence was given by P.J.J. van der Laarschot and his staff. For the X8 the ALGOL compiler written by F.E.J. Kruseman Aretz was used, while the two ALGOL compilers respectively written by E.W. Dijkstra and J.A. Zonneveld and by P.J.J. van der Laarschot and J. Nederkoorn were used for the X1.

2. A simple formula manipulation system

Prior to the sections which describe the general system, this section contains a completely worked out program, which does some very simple formula manipulations.
The procedures declared in this program, together with the heading of the program and some initializing statements form the simple system.
By means of the reproduced program it will be shown how the formula manipulation is internally executed by the computer. Since the internal representation of formulae is essentially the same for the simple and for the general system, it is not necessary to explain in the next sections the detailed mechanism by which the computer treats formulae in the general system.

The formulae to be manipulated occur in the ALGOL 60 program as particular expressions of type integer.
They are syntactically defined as follows:

```
<formula> ::= <algebraic variable>|<formula designator>|
              <sum>|<product>|<derivative>
<algebraic variable> ::= <variable>
<formula designator> ::= <variable>
<sum> ::= S(<lhs>, <rhs>)
<product> ::= P(<lhs>, <rhs>)
<derivative> ::= DER(<formula>, <algebraic variable>)
<lhs> ::= <formula>
<rhs> ::= <formula>
```

In this definition, which is only valid for this section, certain meta linguistic variables were used but not defined; these are defined in the ALGOL 60 report [1].

Some examples of formulae are:	which read in ordinary notation:
x	x
one	1
S(DER(a,x), DER(b,x))	∂a/∂x + ∂b/∂x
P(x,y)	x × y

A typical assignment statement in which formulae occur is for example:

```
derivative := if f = x then one else
              if type = sum then S(DER(a,x), DER(b,x)) else
              if type = product then
                 S(P(a,DER(b,x)), P(DER(a,x),b)) else
              zero
```

Here, derivative, f, a, and b are formula designators; x, one, and zero are algebraic variables.

```
begin comment SIMPLE FORMULA MANIPULATION SYSTEM
  R 1050 RPR 180766/01;
  integer one,zero,sum,product,algebraic variable,k;
  integer array F[1:1000,1:3];
  integer procedure STORE(lhs,type,rhs); value lhs,type,rhs;
  integer lhs,type,rhs;
  begin STORE:= k:= k + 1; F[k,1]:= lhs; F[k,2]:= type; F[k,3]:= rhs end;
  integer procedure TYPE(f,lhs,rhs); value f; integer f,lhs,rhs;
  begin lhs:= F[f,1]; TYPE:= F[f,2]; rhs:= F[f,3] end;
  integer procedure S(a,b); value a,b; integer a,b;
  S:= if a = zero then b else if b = zero then a else STORE(a,sum,b);
  integer procedure P(a,b); value a,b; integer a,b;
  P:= if a = zero ∨ b = zero then zero else
     if a = one then b else if b = one then a else
      STORE(a,product,b);
```

```
  integer procedure DER(f,x); value f,x; integer f,x;
  begin integer a,type,b; type:= TYPE(f,a,b);
    DER:= if f = x then one else
          if type = sum then S(DER(a,x),DER(b,x)) else
          if type = product then S(P(a,DER(b,x)),P(DER(a,x),b)) else
          zero
  end DER;
INITIALIZE: sum:= 1; product:= 2; algebraic variable:= 3; k:= 0;
  one:= STORE(0,algebraic variable,0);
  zero:= STORE(0,algebraic variable,0);
ACTUAL PROGRAM:
  begin integer f,x,y;
    procedure PR(s); string s;
    comment PR prints the string s without the string quotes ‹ and › ;
    PRINTTEXT(s);
    procedure OUTPUT(f); value f; integer f;
    begin integer a,type,b; type:= TYPE(f,a,b);
      if f = one then PR(‹1›) else
      if f = zero then PR(‹0›) else
      if f = x then PR(‹x›) else
      if f = y then PR(‹y›) else
      begin PR(‹(›); OUTPUT(a);
        if type = sum then PR(‹+›) else
        if type = product then PR(‹×›);
        OUTPUT(b); PR(‹)›)
    end end OUTPUT;
    x:= STORE(0,algebraic variable,0);
    y:= STORE(0,algebraic variable,0); NLCR;
    f:= S(x,y); OUTPUT(f); NLCR;
    f:= P(x,y); OUTPUT(f); NLCR;
```

```
    f:= P(S(x,y),S(x,y)); OUTPUT(f); NLCR;
    f:= DER(f,x); OUTPUT(f); NLCR;
    f:= DER(f,y); OUTPUT(f); NLCR
  end
end
```

The program consists of two blocks.
The outer block, containing a number of procedure declarations and some statements labelled INITIALIZE, is standard and forms the simple system. The inner block, labelled ACTUAL PROGRAM, is ad hoc and defines the specific formula manipulations to be performed.

The variables of type integer one, zero, sum, product, algebraic variable and k, declared in the heading of the program, get values after the label INITIALIZE.

The variable k is used as a pointer in the array F[1:1000, 1:3], in which the formulae are internally represented.

As will be seen in the sequel, all algebraic variables and all formula designators occur in the program as variables of type integer.
In executing the program, they become equal to integers which define the location in F where the internal representation of the corresponding formulae are stored.

The effect of executing the following statements will now be examined:

```
      one := STORE (0, algebraic variable, 0);
     zero := STORE (0, algebraic variable, 0)
```

The procedure STORE augments k by 1 (k was originally 0) and stores the three values of its three parameters: lhs, type and rhs into F[k,1], F[k,2] and F[k,3] respectively; moreover STORE itself becomes equal to k.

Thus, after executing the above statements, one = 1 and zero = 2.
Furthermore, F[1,1] = 0, F[1,2] = 3, F[1,3] = 0,
F[2,1] = 0, F[2,2] = 3, F[2,3] = 0.

In the sequel this is abbreviated to:

one = 1(0,3,0),
zero = 2(0,3,0).

The first statements of the actual program are:

x := STORE(0, algebraic variable, 0);
y := STORE(0, algebraic variable, 0),

whereupon x and y get the values 3(0,3,0) and 4(0,3,0) respectively.

The effect of the statement NLCR is to give the printer a New Line Carriage Return command.

The next statement is f := S(x,y) and f gets the value 5(3,1,4), since neither x nor y are equal to the variable zero. The effect of the statement OUTPUT(f) is the printing of the character string "(x+y)".
This can be seen as follows:
By means of the procedure TYPE, which is the counterpart of the procedure STORE, the variables a, type, and b of OUTPUT get the values 3, 1 and 4 respectively.
Since f is not equal to the variables one, zero, x or y, the character "(" is printed.
A call for OUTPUT(3) has the effect of printing "x".
Since type = sum the character "+" is printed.
The character "y" is printed after a call for OUTPUT(4), and finally the character ")" is printed.

The effect of the statements f := P(x,y) and OUTPUT(f) is: f = 6(3,2,4) and the characterlist "(x×y)" is printed.

A more complicated statement is f := P(S(x,y), S(x,y)).
The parameters a and b of the procedure P are called by value, thus they are calculated beforehand;

a gets the value 7(3,1,4) and b the value 8(3,1,4).
Since neither a nor b are equal to one or zero, P and thus f gets the value 9(7,2,8).
The procedure OUTPUT will now print the character string "((x+y) × (x+y))".

The statement f := DER(f,x) is discussed now and the roles of the algebraic variables one and zero will become apparent.

The variables a, type and b of DER get the values 7, 2 and 8 respectively. DER becomes equal to S(P(7,DER(8,x)),P(DER(7,x),8)).
First P(7,DER(8,x)) will be calculated and this in turn activates the calculation of DER(8,x).
In the calculation of DER(8,x),a, type and b of DER become equal to 3, sum and 4 respectively. DER becomes equal to S(DER(3,x),DER(4,x)).
DER(3,x) gets the value of one and DER(4,x) the value of zero.
S(one,zero) becomes equal to the value of one, and thus DER(8,x) = one.
P(7,DER(8,x)) becomes equal to 7, since the parameter b of P is equal to the value of one.
In the same way P(DER(7,x),8) gets the value 8.
Finally DER(f,x) and thus f getsthe value 10(7,sum,8).
Next, OUTPUT prints the character string "((x+y)+(x+y))".

It is left to the reader to verify that the effect of the last statements:

f := DER(f,y) and OUTPUT(f)

is that f gets the value 11(1,1,1) and that the following character string is printed: "(1 + 1)".

This section is closed with the following remarks:

1. As shown above, the computer handles a formula by means of a number which defines the location of the internal representation of this formula in the array F.
   The actions of the procedures STORE, S, P and DER consist primarily of side-effects,namely storing formulae, while the calculated values of the procedure identifiers are only important within the computer.

The programmer is interested in the form of a formula such as it is stored in F, but not in the actual value of the corresponding formula designator.
In the discussion of the next sections, therefore, reference is made to the formula which belongs to a certain formula designator, instead of the value of this formula designator.
In notation this has the consequence that, for example, the following sentence:

"The value of the formula designator corresponding to the formula f becomes equal to the value of the formula designator corresponding to the formula g"

is abbreviated to:

"The formula f becomes equal to the formula g"

or more simply: "f becomes equal to g".

2. For the internal representation of algebraic variables, three array elements of F were used, whereas one array element, F[k,2] was effectively used.
The other array elements may be used, however, to store more information of the algebraic variables, which will be done in the general system.
If there occur a large number of algebraic variables then one may also specify an algebraic variable as having, in the computer, a negative value in contrast to all other formulae.
The procedure TYPE has to be changed in such a way that it takes care of this situation.

3. The array F was used to store formulae; evidently there will be difficulties if there are more than 1000 formulae to be stored.
In the general system this problem is solved by using two devices: one is to use an own array F declared in the procedure body of INT REPR, which can grow if necessary until the program runs out of all available storage space, the other is to provide the user the means to erase uninteresting formulae.

4. If the user wants a system which applies the distributive law to formulae, he may use instead of the procedure P the following procedure:

```
integer procedure P(a,b); value a, b; integer a, b;
begin integer ta, tb, la, lb, ra, rb;
     ta := TYPE(a, la, ra); tb := TYPE(b, lb, rb);
     P := if a = zero ∨ b = zero then zero else
          if a = one then b else
          if b = one then a else
          if ta = sum then S(P(la,b), P(ra,b)) else
          if tb = sum then S(P(a,lb), P(a,rb)) else
          STORE(a, product, b)
end
```

Using this procedure,all formulae are stored internally in expanded form.

3. The form of formulae in the general system

The formulae used in the general system which occur in an ALGOL 60 program should have the form syntactically defined below, in which meta-linguistic variables used but not defined are defined in the ALGOL 60 report [1].

<formula> ::= one|zero|<algebraic variable>|<formula designator>|
<sum>|<difference>|<product>|<quotient>|
<power>|<integral power>|<number>|<polynomial>|
<function>|<extended sum>|<derivative>|<simplified formula>|
<complex conjugate>|<result of substitution>|
<integral quotient>|<common divisor>

<algebraic variable> ::= <variable>

<formula designator> ::= <variable>

<sum> ::= S(<lhs>, <rhs>)

<difference> ::= D(<lhs>,<rhs>)

<product> ::= P(<lhs>,<rhs>)

<quotient> ::= Q(<lhs>,<rhs>)

<power> ::= POWER(<lhs>,<rhs>)

<lhs> ::= <formula>

<rhs> ::= <formula>

<integral power> ::= INT POW (<formula>,<integral arithmetic expression>)

<integral arithmetic expression> ::=<arithmetic expression>

<number> ::= <integer number>|<real number>|<complex number>

<integer number> ::= IN(<integral arithmetic expression>)

<real number> ::= RN(<arithmetic expression>)

<complex number> ::= CN(<arithmetic expression>, <arithmetic expression>)

<polynomial> ::= POL(<integer variable>, <degree>, <formula>,
<coefficient depending on integer variable>)

<integer variable> ::= <variable>

<degree> ::= <integral arithmetic expression>

<coefficient depending on integer variable> ::= <formula>

<function> ::= <special function designator>

<special function designator> ::= <special function identifier>(<formula>)

<special function identifier> ::= EXP|LN|SIN|COS|ARCTAN|SQRT

<extended sum> ::= Sum(<integer variable>, <integral arithmetic expression>, <integral arithmetic expression>, <formula depending on integer variable>)

<formula depending on integer variable> ::= <formula>

<derivative> ::= DER(<formula>, <algebraic variable>)

<simplified formula> ::= SIMPLIFY(<formula>)

<complex conjugate> ::= CC(<formula>)

<result of substitution> ::= SUBSTITUTE(<formula>, <integer variable>, <integral arithmetic expression>, <integral arithmetic expression>, <formula depending on integer variable>, <formula depending on integer variable>)

<integral quotient> ::= QUOTIENT(<formula>, <formula>, <rest>)

<rest> ::= <formula designator>

<common divisor> ::= COMMON DIVISOR(<formula>, <formula>)

The following remarks are made:

1. An integral arithmetic expression is an arithmetic expression of type integer.

2. degree is an arithmetic expression taking non-negative integral values only.

3. Examples of coefficient depending on integer variable and formula depending on integer variable are RN(1/i) or a[i], where a[i] is a formula designator.

4. In the sequel formulae will not only be written in the text as defined above, but also in ordinary notation.

5. The symbols S, D, P, etc., occurring at the left hand sides of the above definition, are in fact identifiers of procedures of type integer (except for one and zero, which are integer variables) in the general system.
   This means that a formula occurring in an ALGOL 60 program is an integral arithmetic expression.

Examples of formulae are:

```
S(one,x)
S(INT POW(SIN(x),2), INT POW(COS(y),2))
Q(CN(1,1), CN(1,-1))
POL(i,n,x, if i = 0 then zero else RN(1/i))
SUBSTITUTE(f, i, 1, 2, a[i], RN(1/i))
COMMON DIVISOR(Sum(i, 0, 3, INT POW(x, 3-i)), D(P(x,x),one))
```

which have in ordinary notation the meaning:

$1+x$

$\sin^2 x + \cos^2 y$

$(1+i)/(1-i)$ (i is the imaginary unit)

$x + x^2/2 + \ldots + x^n/n$

if the variables a[1] and a[2] occur in f, then they are replaced by 1 and $\frac{1}{2}$ respectively.

A common divisor of $x^3 + x^2 + x + 1$ and $x^2 - 1$ is $x + 1$.

4. How to use the general system

There are two ways to use the general system.
If the tools of the system are sufficient, one may put the formulae on input tape and use the program as reproduced in section 15.

The other way of using the general system is described in the rest of this section, it amounts to:

1. Copy the general system as reproduced in sections 5-13.
2. Write a program which defines the operations on formulae to be done, this program will henceforth be called "actual program".
3. Combine the general system and the actual program, preceded by the statement: INITIALIZE and closed by an extra end, to one program.

In most cases it is desirable to output formulae, one may then copy the declaration of the procedure OUTPUT (section 14), in the actual program.
In the procedure OUTPUT a call for OUTPUT VARIABLE is made; the procedure OUTPUT VARIABLE should therefore also be declared in the actual program.
OUTPUT VARIABLE defines the output of a specific algebraic variable.

The algebraic variables and formula designators, used in the actual program, should be declared in the heading of the actual program as variables of type integer or as integer array elements. Before the algebraic variables are used, they should have got a value by means of a statement of the form:

<algebraic variable> := STORE(<integral arithmetic expression>, algebraic variable, <integral arithmetic expression>)

The easiest way to perform these actions is to write the procedure OUTPUT VARIABLE in such a way that it combines both the outputting and the initializing instructions.
An example of a possible declaration of OUTPUT VARIABLE may be found in section 14.

The formula designators should get values in the program by means of statements of the form:

<formula designator> := <integral arithmetic expression>

In most cases the integral arithmetic expression is a formula, it may, however, be built up by means of if clauses and formulae.

5. The internal representation of formulae in the system

The general system is constructed in such a way that one and zero are represented as formulae of the form number (in contrast to their representation in the simple system), moreover an extended sum, a derivative, a simplified formula, a complex conjugate, a result of a substitution, an integral quotient and a common divisor is worked out, i.e. the "operators" Sum, DER, SIMPLIFY, CC, SUBSTITUTE, QUOTIENT and COMMON DIVISOR are applied to the formulae occurring as parameters. This means that each formula f can be characterized by three quantities which will be called: lhs, type, and rhs;
type may be: sum, difference, product, quotient, power, integral power, number, function, polynomial or algebraic variable.
In the first five cases, lhs and rhs denote the left hand side and the right hand side formulae with which f is built up.
The meaning of lhs and rhs in the other cases is summarized in the following table:

type	lhs	rhs
integral power	exponent of f	base of f
number	"type" of number i.e. integer, real or complex	location where the value of the number can be found
polynomial	degree of polynomial	location where the argument and the coefficients can be found
function	"type" of function i.e. exp, ln, sin, cos, arctan or sqrt	argument of function
algebraic variable	may be used to define extra information	

6. The heading and the procedure INT REPR

The heading of the general system runs as follows:

```
begin comment GENERAL SYSTEM for FORMULA MANIPULATION
  R1050 RPR 290466/03/05/06;
  integer sum,difference,product,quotient,power,integral power,
  number,polynomial,algebraic variable,function,
  integer,real,complex,expf,lnf,sinf,cosf,arctanf,sqrtf,
  one,zero,minone,im unit,di;
  real dr,null; Boolean expand; integer array da[0:0];
```

The values of the declared integer variables, except for the integer variables one, zero, min one, im unit and di, are used in defining the characterizing quantities lhs, type and rhs of the internal representation of formulae. The integer variables one, zero, minone and im unit will correspond to formulae of the form number, which in ordinary notation read: 1, 0, -1 and i respectively.

As "dummy" variables the variables di, dr and the array da are declared, to be used as uninteresting actual parameters in some procedure calls.

The general system has its own arithmetic.
Complex numbers with imaginary part smaller than the real variable null, are transformed into real numbers; real numbers lying near an integer number, within the accuracy defined by null, are transformed into integer numbers.
Formulae in the general system can be treated in two ways:
First, the formulae may be stored as they stand, in this case the Boolean variable expand = false.
The second way is that the formulae are stored in expanded form, i.e. the distributive law and other laws are applied; in this case expand = true.

The declaration of the procedure INT REPR is now reproduced. Although INT REPR is, as far as the storage of formulae is concerned, the keystone of the general system, the actual form of it is not very interesting.

```
comment This is a part of the general system;
procedure INT REPR(case,formula,lhs,type,rhs,rnum,inum,coeff); value case;
integer case,formula,lhs,type,rhs; real rnum,inum; integer array coeff;
begin own integer k,krn,kcn,kpol,index,kmax,krnmax,kcnmax,kpolmax,
  degree max,indexmax; integer i; if case ≠ 1 then goto A;
  k:= krn:= kcn:= kpol:= index:= 0;
  kmax:= krnmax:= kcnmax:= kpolmax:= indexmax:= 1; degree max:= lhs;
  null:= rnum; di:= 0; dr:= 0; da[0]:= 0; expand:= formula = 1;
  sum:= 1; difference:= 2; product:= 3; quotient:= 4; power:= 5;
  integral power:= 6; number:= 7; polynomial:= 8; function:= 9;
  algebraic variable:= 10; integer:= 1; real:= 2; complex:= 3;
  expf:= 1; lnf:= 2; sinf:= 3; cosf:= 4; arctanf:= 5; sqrtf:= 6;
A: begin own integer array F[1:kmax,1:3],FPOL[1:kpolmax,-1:degree max],
    INDEX[1:indexmax,1:4];
    own real array FRN[1:krnmax],FCN[1:kcnmax,1:2];
    switch CASE:= END,STORE FORM,STORE REAL NUMBER,STORE
    COMPLEX NUMBER,STORE POLYNOMIAL,TAKE FORM,TAKE
    REAL NUMBER,TAKE COMPLEX NUMBER,TAKE POLYNOMIAL,
    FIX,ERASE,LOWER INDEX,FIXED,REPLACE,MAKE SPACE;
    goto CASE[case];
  STORE FORM: if k < kmax then k:= k + 1 else
    begin kmax:= kmax + 25; goto A end;
    F[k,1]:= lhs; F[k,2]:= type; F[k,3]:= rhs; formula:= k;
    case:= if type = number then (if lhs = real then 3 else if lhs =
      complex then 4 else 1) else if type = polynomial then 5 else 1;
    goto CASE[case];
  STORE REAL NUMBER: if krn < krnmax then krn:= krn + 1 else
    begin krnmax:= krnmax + 25; goto A end;
```

```
    FRN[krn]:= rnum; F[k,3]:= krn; goto END;
  STORE COMPLEX NUMBER: if kcn < kcnmax then kcn:= kcn + 1 else
    begin kcnmax:= kcnmax + 25; goto A end;
    FCN[kcn,1]:= rnum; FCN[kcn,2]:= inum; F[k,3]:= kcn; goto END;
  STORE POLYNOMIAL: if kpol < kpolmax then kpol:= kpol + 1 else
    begin kpolmax:= kpolmax + 25; goto A end;
    FPOL[kpol,-1]:= rhs; for i:= 0 step 1 until lhs do
    FPOL[kpol,i]:= coeff[i]; F[k,3]:= kpol; goto END;
  TAKE FORM: lhs:= F[formula,1]; type:= F[formula,2]; rhs:= F[formula,3];
    goto END;
  TAKE REAL NUMBER: rnum:= FRN[F[formula,3]]; goto END;
  TAKE COMPLEX NUMBER: rnum:= FCN[F[formula,3],1];
    inum:= FCN[F[formula,3],2]; goto END;
  TAKE POLYNOMIAL: for i:= 0 step 1 until lhs do
    coeff[i]:= FPOL[F[formula,3],i]; rhs:= FPOL[F[formula,3],-1]; goto END;
  FIX: if index < indexmax then index:= index + 1 else
    begin indexmax:= indexmax + 10; goto A end; INDEX[index,1]:= k;
    INDEX[index,2]:= krn; INDEX[index,3]:= kcn; INDEX[index,4]:= kpol;
    goto END;
  ERASE: k:= INDEX[index,1]; krn:= INDEX[index,2]; kcn:= INDEX[index,3];
    kpol:= INDEX[index,4]; index:= index - 1; goto END;
  LOWER INDEX: index:= index - 1; goto END;
  FIXED: type:= if formula ≤ INDEX[index,1] then 1 else 0; goto END;
  REPLACE: F[formula,1]:= F[lhs,1]; F[formula,2]:= F[lhs,2];
    F[formula,3]:= F[lhs,3]; goto END;
  MAKE SPACE: kmax:= k; krnmax:= krn; kcnmax:= kcn; kpolmax:= kpol;
    indexmax:= index; case:= 1; goto A;
  END: end
end INT REPR;
```

By means of the procedure INT REPR, formulae are stored, information about stored formulae is defined, formulae are erased, storage space is made available and the necessary initializing statements are executed.

The effect of a call for INT REPR with its parameter case equal to 1, 2, 6, 7, 8, 9, 10, 11, 12, 13, 14 and 15 will now be discussed.

It is remarked that the next section contains 15 procedure declarations; each one corresponds to a label of the switch CASE.

case = 1: The variables declared in the heading get values (except for one, zero, min one and im unit);
the own declared variables k, krn, kcn, kpol, index (used as pointers in the arrays F, FRN, FCN, FPOL and INDEX), kmax, krn max, kcn max, kpol max and index max (used as bounds of the same arrays) get values;
the variable degree max gets a value defining the maximal degree of the polynomials.
The second block of INT REPR contains the declaration of the above mentioned arrays.
F is used to store all formulae, in FRN and FCN the values of real respectively complex numbers are stored; the coefficients of polynomials and the argument are stored in FPOL.
In INDEX fixed values of the pointers k, krn, kcn and kpol are stored.

case = 2: The three characterizing quantities: lhs, type and rhs are stored in F, possible values of numbers or polynomials are stored in FRN, FCN and FPOL; moreover the parameter "formula" gets the value of k.

case = 6, 7, 8, 9: the process as described for case = 2 is reversed: the contents of the array elements defined by the value of "formula" is set in the parameters of INT REPR.

case = 10: The values, at the moment INT REPR is called with case = 10, of the pointers k, krn, kcn and kpol, called fixed pointers, are stored in INDEX.

case = 11: The pointers k, krn, kcn and kpol are set equal to the lastly stored fixed pointers, moreover these fixed pointers are removed from INDEX.

case = 12: The lastly stored fixed pointers are removed from INDEX.

case = 13: INT REPR investigates whether the parameter "formula" has a value less than or equal to the lastly stored fixed pointer k. (In this case the corresponding formula was stored before this pointer was fixed.)

case = 14: The contents of a given formula is replaced by the contents of another formula.

case = 15: The lengths of the arrays are (in general) diminished, so as to obtain more storage space in the computer for other work which has to be done.

Remark: If one does not have an ALGOL 60 compiler which accepts own arrays then the arrays F, FRN, FCN, FPOL and INDEX should be removed from INT REPR and they should be declared in the heading of the general system.
The arraybounds should then be given beforehand, e.g. by means of numbers on input tape, or by means of some number which defines the total amount of storage space still available in the computer.

7. The basic procedures

The procedures to be discussed in this section form the connection between the rest of the general system and the procedure INT REPR. They are used throughout the program, and are called basic therefore. A more efficient system would be obtained by replacing INT REPR by a machine code subroutine, which stores the formulae in a more compact form.
The basic procedures should then be connected with this subroutine, their meaning and their declaration should,however,be maintained.

The declaration of the basic procedures now follows:

```
comment This is a part of the general system;
procedure INITIALIZE;
begin real null; integer i,j; j:= read; null:= abs(read);
  i:= read; INT REPR(1,i,j,di,di,null,dr,da);
  one:= STORE(integer,number,1); zero:= STORE(integer,number,0);
  minone:= IN(-1); im unit:= CN(0,1); FIX
end INITIALIZE;
integer procedure STORE(lhs,type,rhs); value lhs,type,rhs;
integer lhs,type,rhs;
begin integer f; INT REPR(2,f,lhs,type,rhs,dr,dr,da); STORE:= f end;
integer procedure IN(i); value i; integer i;
if i = 1 then IN:= one else if i = 0 then IN:= zero else
begin integer f; INT REPR(2,f,integer,number,i,dr,dr,da); IN:= f end;
integer procedure RN(r); value r; real r;
if r - entier(r) ≤ null then RN:= IN(entier(r)) else
if - r - entier(- r) ≤ null then RN:= IN(- entier(- r)) else
begin integer f; INT REPR(2,f,real,number,di,r,dr,da); RN:= f end;
integer procedure CN(r,i); value r,i; real r,i;
if abs(i) ≤ null then CN:= RN(r) else
begin integer f; INT REPR(2,f,complex,number,di,r,i,da); CN:= f end;
```

```
integer procedure POL(i,degree,x,coeffi); value degree,x;
integer i,degree,x,coeffi;
begin integer f; integer array coeff[0:degree];
  for i:= 0 step 1 until degree do coeff[i]:= coeffi;
  INT REPR(2,f,degree,polynomial,x,dr,dr,coeff); POL:= f
end POL;
integer procedure TYPE(f,lhs,rhs); value f; integer f,lhs,rhs;
begin integer t; INT REPR(6,f,lhs,t,rhs,dr,dr,da); TYPE:= t end;
procedure VALUE OF INT NUM(f,i); value f; integer f,i;
INT REPR(6,f,di,di,i,dr,dr,da);
procedure VALUE OF REAL NUM(f,r); value f; integer f; real r;
INT REPR(7,f,di,di,di,r,dr,da);
procedure VALUE OF COMPLEX NUM(f,r,i); value f; integer f;
real r,i; INT REPR(8,f,di,di,di,r,i,da);
procedure COEFFICIENT(f,degree,x,coeff); value f,degree;
integer f,degree,x; integer array coeff;
INT REPR(9,f,degree,di,x,dr,dr,coeff);
procedure FIX; INT REPR(10,di,di,di,di,dr,dr,da);
procedure ERASE; INT REPR(11,di,di,di,di,dr,dr,da);
procedure LOWER INDEX; INT REPR(12,di,di,di,di,dr,dr,da);
Boolean procedure FIXED(f); value f; integer f;
begin integer t; INT REPR(13,f,di,t,di,dr,dr,da); FIXED:= t = 1 end;
procedure REPLACE(f,g); value f,g; integer f,g;
INT REPR(14,f,g,di,di,dr,dr,da);
procedure MAKE SPACE; INT REPR(15,di,di,di,di,dr,dr,da);
```

A description of these procedures and some comment is given below:

INITIALIZE: A call for INITIALIZE is necessary before any action of the program.

All variables used in the general system get values.

From input tape INITIALIZE reads by means of the not-declared procedure read the values of:

1. the maximal degree of the polynomials,
2. the real variable null,
3. the Boolean variable expand, which becomes true if the read number is equal to 1.

Finally, the integer variables one, zero, min one and im unit get values; by a call for FIX, the formulae corresponding to these variables are protected against (possible) erasing.

STORE, IN, RN, CN, POL: store a formula. To the procedure identifiers numbers are assigned, defining the location of the internally represented formula.

IN: stores an integer number with value i. If i is possibly 1 or 0 then IN becomes equal to one or zero.
It is evident now why the formulae one and zero were not stored in INITIALIZE by the statements one := IN(1) and zero := IN(0).

RN: stores a real number, with value r. The real number is stored as an integer number if r lies close to an integer number (within the precision defined by null).

CN: stores a complex number with value r + $\sqrt{-1}$ i. If i = 0 (within the precision defined by null), the complex number is transformed into a real number.

Remark: a consequence of the structure of IN, RN and CN is that e.g.

EXP(P(RN(2 * 3.149265359), im unit))

becomes equal to one (if null is not too small of course).

POL: stores the polynomial $\sum_{i=0}^{\text{degree}}$ coeff i * x^i (use is made of the Jensen device).

The next five procedure declarations are the counterparts of the five, discussed a moment ago; they deliver information of a stored formula f.

TYPE: becomes equal to the type of f; the parameters lhs and rhs become equal to the lhs and rhs quantities of f.

VALUE OF INT NUM: to the parameter i the value of the integral number f is assigned.

VALUE OF REAL NUM: to the parameter r the value of the real number f is assigned.

VALUE OF COMPLEX NUM: to the parameters r and i, the real and imaginary parts of the value of the complex number f are assigned.

COEFFICIENT: to the parameters x and coeff, the argument and the coefficients, with index running from 0 to the value of degree, of a polynomial f are assigned.
The value of degree should be smaller than or equal to the lhs of f (the degree of the polynomial).

FIX, ERASE, LOWER INDEX: During the course of the program certain formulae are built up which may be erased later on. This is only possible if these formulae are stored consecutively. A call for FIX has the effect that formulae, which were stored before, are protected against the erasing effect of a next call for ERASE, later on. Moreover, a call for ERASE or LOWER INDEX has the effect of cancelling the effect of the last call for FIX.
Example: Suppose the formulae f1, f2 and f3 are stored as an effect of the statements S1, S2 and S3 respectively, then the effect of the procedure statements FIX, ERASE and LOWER INDEX can be seen from the following sequences of statements; the formulae, which are still available after these statements are executed, are placed between brackets.

S1; FIX; S2; ERASE; S3	(f1,f3)
S1; FIX; S2; ERASE; ERASE; S3	(f3)
S1; FIX; S2; ERASE; S3; ERASE	( )
S1; FIX; S2; ERASE; FIX; S3; ERASE	(f1)
S1; FIX; S2; LOWER INDEX; S3	(f1,f2,f3)
S1; FIX; S2; LOWER INDEX; S3; ERASE	( )

It should be remarked that the storage space, used for storing f2 in the first sequence of statements, is used, after the call for ERASE and S3, for storing f3.

Each call for ERASE or LOWER INDEX should be preceded by a call for FIX. A problem occurs with the sequence of statements S1; S2; S3, if after execution of these statements, f1 and f3 may be erased but f2 may not. In this case one can use the procedure ERASE BUT RETAIN, defined in section 12 in the following way:

FIX; FIX; S1; S2; S3; ERASE BUT RETAIN (i, 1, 1, f2)

After execution of these statements, f1 and f3 are erased, the effect of the statements FIX; FIX is cancelled and f2 is still available for further use.

A better method of erasing formulae would be a garbage collection method.
The reasons for not building in this method are described in section 12 on storage allocation.

FIXED: becomes equal to true or false depending on whether the formula f was stored before or after the last, not cancelled, call for FIX.

REPLACE: replaces the contents of the stored formula f by the contents of the stored formula g, without altering the number defining the location of the internal representation of f.
The procedure REPLACE should be used very carefully (it is not used in the general system).
Possible erroneous use of REPLACE is illustrated by the following example:
Suppose expand = true, then the formulae are stored in expanded form, the formula g, defined by:

f := S(a,b); g := P(f,c)

is stored therefore as a * c + b * c.
The statement REPLACE(f,S(x,y)) has no effect on g.
However, if expand = false then also g would be altered by the statement REPLACE(f,S(x,y)); g is then internally represented as (x+y) * c.

MAKE SPACE: storage space, used for already erased formulae, is made available for other purposes, e.g. for storage space needed by recursively called procedures.

MAKE SPACE can only effectively be used if one has the possibility to ask for the total amount of storage space still available in the computer; one may then build in at some appropriate places (e.g. in the procedures S, P and Q) a check for the available storage space, which possibly leads to a call for MAKE SPACE.
The general system does not use MAKE SPACE.

8. The procedures S, D, P, Q, POWER and INT POW

The procedures of the title of this section are used to store a sum, a difference, a product, a quotient, a power and an integral power. In the declaration of these procedures use is made of the following procedures reproduced first.

```
comment This is a part of the general system;
integer procedure CONST POL(degree,c); value degree,c; integer degree,c;
begin integer i; CONST POL:= POL(i,degree,one,if i = 0 then c else zero)
end CONST POL;
integer procedure OPER ON POL(oper,a,ta,da,b,tb,db);
value oper,a,ta,da,b,tb,db; integer oper,a,ta,da,b,tb,db;
begin integer x,y,i,degree,j; Boolean B; B:= true;
  if ta ≠ polynomial then
  begin degree:= db; a:= CONST POL(degree,a) end else
  if tb ≠ polynomial then
  begin degree:= da; b:= CONST POL(degree,b) end else
  degree:= if da < db then da else db;
  begin integer array coeff,coeffa,coeffb[0:degree];
    COEFFICIENT(a,degree,x,coeffa); COEFFICIENT(b,degree,y,coeffb);
    if oper = sum then
    begin for i:= 0 step 1 until degree do
      coeff[i]:= SIMPLIFY(S(coeffa[i],coeffb[i]))
    end else if oper = product then
    begin for i:= 0 step 1 until degree do
      coeff[i]:= SIMPLIFY(Sum(j,0,i,P(coeffa[j],coeffb[i-j])))
    end else if oper = quotient then
    begin for i:= 0 step 1 until degree do coeff[i]:=
      SIMPLIFY(Q(D(coeffa[i],Sum(j,0,i-1,P(coeff[j],coeffb[i-j]))),coeffb[0]))
    end; for i:= 1 step 1 until degree do B:= B ∧ coeff[i] = zero;
    OPER ON POL:= if B then coeff[0] else POL(i,degree,
      if x = one then y else x,coeff[i])
end end OPER ON POL;
```

```
procedure VALUE OF NUM(f,r,i); value f; integer f; real r,i;
begin integer n,type; TYPE(f,type,n);
  if type = integer then begin VALUE OF INT NUM(f,n); r:= n; i:= 0 end
  else if type = real then begin VALUE OF REAL NUM(f,r); i:= 0 end
  else VALUE OF COMPLEX NUM(f,r,i)
end VALUE OF NUM;
integer procedure ARITHMETIC(oper,a,b); value oper,a,b; integer oper,a,b;
begin real r,i,ra,ia,rb,ib;
  VALUE OF NUM(a,ra,ia); VALUE OF NUM(b,rb,ib);
  if oper = sum then begin r:= ra + rb; i:= ia + ib end else
  if oper = product then
  begin r:= ra × rb - ia × ib; i:= ra × ib + ia × rb end else
  begin r:= rb × rb + ib × ib; if abs(r) ≤ null × null then
    r:= null × null; i:= (ia × rb - ra × ib)/r; r:= (ra × rb + ia × ib)/r
  end; ARITHMETIC:= CN(r,i)
end ARITHMETIC;
integer procedure Sum(i,lb,ub,fi); value lb,ub; integer i,lb,ub,fi;
begin integer s; s:= zero; for i:= lb step 1 until ub do s:= S(s,fi); Sum:= s
end Sum;
integer procedure COMB(n,m); value n,m; integer n,m;
COMB:= if m = 0 then 1 else (COMB(n,m-1) × (n + 1 - m)) : m;
integer procedure REPEATED PRODUCT(f,n); value f,n; integer f,n;
begin integer a; a:= INT POW(f,n:2); REPEATED PRODUCT:= P(P(a,a),
  if (n:2)×2=n then one else f)
end REPEATED PRODUCT;
integer procedure comm div(a,b,f); integer a,b,f;
begin integer r,cd; cd:= COMMON DIVISOR(a,b);
  if cd ≠ one then
  begin a:= QUOTIENT(a,cd,r); b:= QUOTIENT(b,cd,r) end;
  comm div:= f
end comm div;
```

A short discussion of these procedure declarations follows:

CONST POL: has an obvious meaning.

OPER ON POL: becomes a polynomial of degree ≥ 0 or a formula as the result of some calculation.
At least one of the parameters a and b has to be a polynomial.
The parameters ta and tb are the types of a and b, while the parameters da and db are the lhs of a and b; they define in the case that a or b are polynomials the degree of the polynomials.
The parameter oper defines the sort of calculation: summation, multiplication or division, which should be done.
If one of the formulae a and b is not a polynomial, it is transformed into a constant polynomial.
The following recursive relations are used for the calculation:
assume the coefficients of a and b are respectively $coeffa_i$ and $coeffb_i$

a + b : $coeff_i = coeffa_i + coeffb_i$

a * b : $coeff_i = \sum_{j=0}^{i} coeffa_j \; coeffb_{i-j}$

a / b: $coeff_i = (coeffa_i - \sum_{j=0}^{i-1} coeff_j \; coeffb_{i-j})/coeffb_0$

The resulting coefficients $coeff_i$ are stored in simplified form.
It should be remarked that the intermediate results e.g.
S(coeffa[i], coeffb[i]), are not erased. It is therefore worth while to use the procedure ERASE BUT RETAIN (section 10) for restoring the final result, in order to save space.

If the $coeff_i$ = zero for all $i > 0$ then the result of the calculation is not a polynomial but a formula defined by $coeff_0$. It is remarked that lowering the degree of the polynomial if the $coeff_i$ = zero for all $i \geq n > 0$, would lead to erroneous results, since then, for example,

$(a_0 + a_1x + a_2x^2 + a_3x^3) + (b_0 + b_1x + b_2x^2 + b_3x^3) - (c_0 + c_1x +$
$+ c_2x^2 + c_3x^3)$, with $b_2 = c_2$ and $b_3 = c_3$,

would have the erroneous result:
$a_0 + b_0 - c_0 + (a_1 + b_1 - c_1)x$.

Notice also that it would be meaningless to get from the calculation:

$(a_0 + a_1x + a_2x^2 + a_3x^3) + (d_0 + d_1x)$.

the result: $a_0 + d_0 + (a_1 + d_1)x + a_2x^2 + a_3x^3$,

since the polynomials are treated as truncated power series, thus d_2 and d_3 will in general not be zero, they are unknown however. In the above case $d_2 = b_2 - c_2$ and $d_3 = b_3 - c_3$, thus they represent relevant coefficients, which just happen to be zero.

VALUE OF NUM: the value of a number f is assigned to the real variables r and i (for the real and imaginary parts). No difference is made between integer, real or complex numbers.

ARITHMETIC: becomes a number as the result of some calculation.

Sum: becomes the formula $\sum_{i=lb}^{ub} fi$.

COMB: becomes equal to the value of the combinatorial coefficient:

$$\binom{n}{m} = \frac{n!}{(n-m)!\,m!}$$

REPEATED PRODUCT: becomes equal to the formula $\prod_{i=1}^{n} f$. This procedure can only be used if f is a number or a polynomial.*

comm div: if there exists a common divisor of a and b, not equal to unity, then a and b are divided by this common divisor and they actually get other values. comm div becomes equal to f (which may depend on a and b).

Next the procedures S, D, P, Q, POWER and INT POW are reproduced:

```
comment This is a part of the general system;
integer procedure S(a,b); value a,b; integer a,b;
begin integer ta,la,ra,tb,lb,rb;
  ta:= TYPE(a,la,ra); tb:= TYPE(b,lb,rb);
  S:= if a = zero then b else if b = zero then a else
    if ta = number ∧ tb = number then ARITHMETIC(sum,a,b) else
```

* The integer division symbol ÷ is written as : .

```
    if expand then
      (if ta = polynomial ∨ tb = polynomial then
        OPER ON POL(sum,a,ta,la,b,tb,lb) else
      if ta = quotient ∧ tb = quotient then
        P(Q(one,ra),comm div(ra,rb,Q(S(P(la,rb),P(lb,ra)),rb))) else
      if ta = quotient then Q(S(la,P(b,ra)),ra) else
      if tb = quotient then Q(S(P(a,rb),lb),rb) else
      STORE(a,sum,b))
    else STORE(a,sum,b)
end S;
integer procedure D(a,b); value a,b; integer a,b;
D:= if (TYPE(a,di,di) = number ∧ TYPE(b,di,di) = number) ∨
  expand then S(a,P(minone,b)) else STORE(a,difference,b);
integer procedure P(a,b); value a,b; integer a,b;
begin integer ta,la,ra,tb,lb,rb;
  ta:= TYPE(a,la,ra); tb:= TYPE(b,lb,rb);
  P:= if a = zero ∨ b = zero then zero else if a = one
    then b else if b = one then a else
    if ta = number ∧ tb = number then ARITHMETIC(product,a,b) else
    if expand then
      (if ta = polynomial ∨ tb = polynomial then
        OPER ON POL(product,a,ta,la,b,tb,lb) else
      if ta = quotient ∧ tb = quotient then
        comm div(la,rb,comm div(lb,ra,Q(P(la,lb),P(ra,rb)))) else
      if ta = quotient then comm div(ra,b,Q(P(la,b),ra)) else
      if tb = quotient then comm div(a,rb,Q(P(a,lb),rb)) else
      if ta = sum then S(P(la,b),P(ra,b)) else
      if tb = sum then S(P(a,lb),P(a,rb)) else
      STORE(a,product,b))
    else STORE(a,product,b)
end P;
```

```
integer procedure Q(a,b); value a,b; integer a,b;
begin integer ta,la,ra,tb,lb,rb,a1; a:= a1:= SIMPLIFY(a);
  b:= SIMPLIFY(b); ta:= TYPE(a,la,ra); tb:= TYPE(b,lb,rb);
  if ta = number ∧ tb = number then
  begin Q:= ARITHMETIC(quotient,a,b); goto END end;
  Q:= if a = zero then zero else if b = one then a else
    if tb = number then P(Q(one,b),a) else
    if expand then
      (if ta = polynomial ∨ tb = polynomial then
        OPER ON POL(quotient,a,ta,la,b,tb,lb) else
      if ta = quotient ∧ tb = quotient then
        comm div(la,lb,comm div(ra,rb,Q(P(la,rb),P(lb,ra)))) else
      if ta = quotient then comm div(la,b,Q(la,P(ra,b))) else
      if tb = quotient then comm div(a,lb,Q(P(a,rb),lb)) else
      comm div(a,b,if a = a1 then STORE(a,quotient,b) else Q(a,b)))
    else STORE(a,quotient,b);
END: end Q;
integer procedure POWER(a,b); value a,b; integer a,b;
POWER:= if (TYPE(a,di,di) = number ∧ TYPE(b,di,di) = number) ∨
  expand then EXP(P(b,LN(a))) else STORE(a,power,b);
integer procedure INT POW(a,n); value a,n; integer a,n;
begin integer i,t,la,ra; t:= TYPE(a,la,ra);
  INT POW:= if n = 0 then one else if n = 1 then a else
    if n < 0 then Q(one,INT POW(a,-n)) else
    if t = number then REPEATED PRODUCT(a,n) else
    if expand then
      (if t = polynomial then REPEATED PRODUCT(a,n) else
      if t = sum then Sum(i,0,n,P(IN(COMB(n,i)),
      P(INT POW(la,n-i),INT POW(ra,i)))) else
      if t = product then P(INT POW(la,n),INT POW(ra,n)) else
      if t = quotient then Q(INT POW(la,n),INT POW(ra,n)) else
      STORE(n,integral power,a))
    else STORE(n,integral power,a)
end INT POW;
```

The construction of these procedures is in principle the same as the construction of the procedures S and P of the simple system described in section 2.
A complicating element in these procedures is the division, to which most of the following discussion is devoted.

First, some trivial remarks:

1. If the non-local Boolean variable expand = false, the formulae are stored as they are written, except for a trivial simplification with the unit- and the zero element one and zero (this is done in view of the differentiation procedure).

2. If it is possible (in a trivial way) to carry out some numerical calculation, then it is done.
   Thus e.g. a + (1 + 2) is stored as a + 3, whereas (a + 1) + 2 is stored as it stands.
   (A call for SIMPLIFY will deliver the result a + 3 if expand = true).

3. Due to the construction of IN, RN and CN and due to the treatment of numbers of the basic procedures as illustrated in the above remark and due to the construction of the procedures for storing functions, the result of storing, for example, the formula
   $(i * (1-i)/(1+i)) * a + \ln(1) * b$, where i is the imaginary unit and a and b algebraic variables, is simply a .
   It should be noticed, however, that the above shown simplification is not exact, since the precision null could be chosen wrong.
   The simplification could be made exact by introducing also Gaussian integers.
   Then, however, there still remains the problem that e.g. $\exp(2\pi i)$ or $\ln(e)$ or $(\sqrt[3]{1})^3$ should be recognized by the system equaling the unit element one.
   Thus the symbols π, e, $\sqrt[3]{1}$, etc. should be adjungated to the field of complex numbers, and this would complicate the treatment of numbers severely.

4. In the following remarks it is assumed that expand = true.

5. A difference is stored as a + (-1) * b;
   a power is stored as exp(b * ln(a)).

6. If one of the parameters a and b (or both) of the procedures S, D, P or Q is a polynomial then the result of the calculation will also be a polynomial.

7. The above remark does not apply to the procedure POWER.

8. The procedure INT POW delivers in case a is a number or a polynomial an efficiently formed repeated product. Thus, if a is a polynomial then the result of INT POW is a polynomial.
   If a is a sum, a product or a quotient then INT POW expands a.

The consequences of introducing the division will now be investigated. Difficulties arise from the combination of division and simplification.

Without division and without functions, simplification can easily be defined in the following way:

Two formulae f and g are called equal if
1. f and g are the same numbers, or
2. f and g are the same algebraic variables, or
3. the simplification of f-g equals 0.

Let n_1 and n_2 be numbers, let f and g be equal formulae, then the simplification of $n_1 * f + n_2 * g$ is $(n_1 + n_2) * f$.

Since the ultimate test for equality is a test on the elementary constituants of a formula, i.e. the algebraic variables and the numbers, simplification should be carried out on the expanded formulae, i.e. a sum of products of algebraic variables and numbers.
For this expansion the procedure P defined at the end of section 2 (point 4 of the remarks) can be very helpfull.

It should be noticed that there exists no common agreement on the definition of simplification.

This is not surprisingly since simplification strongly depends on the specific formula manipulations one wants to do.
A definition of simplification in a general system, with formulae of which nothing is known about the algebraic variables constituting the formula, does not leave much choice however.
Factorization, for example, would be a nice facility and in some cases this can be done in an elementary way, e.g. $ax + bx = (a + b)x$, in other cases, however, it can not be done in an elementary way e.g. $a^2 - b^2$.
At the other hand, the user who knows the form of the formulae beforehand, can build his own simplification procedure, which e.g. also factorizes.

Extension of the field of formulae with exponential functions, offers no serious difficulties; it can easily be checked whether exp(f) equals exp(g), since this amounts in testing whether f equals g apart from a multiple of $2\pi i$.

Logarithmic functions, however, are troublesome, since it does not follow from f = g that ln(f) = ln(g).
Logarithmic functions l_1 and l_2 will only be recognized as to be equal if l_1 and l_2 are physically the same (i.e. l_1 and l_2 refer to the same location in the storage space).

The effect of the implementation of division in the system on simplification will be studied next.

Simplification will now be understood in the following sense:

1. The simplification of a formula f is a quotient of a numerator n and a denominator d.
2. n and d are formulae in which no division operator occurs and which are simplified according to the above definition.
3. Moreover, n and d do not contain, apart from unities, common divisors, which can be found in an elementary way.
   The last sentence can be replaced by:
   · COMMON DIVISOR(n,d) = one.

Anticipating the description of the procedure COMMON DIVISOR in section 11, a few remarks are made.
COMMON DIVISOR calculates a common divisor of two formulae f and g. The calculation is never succesfull if f or g contains exponential functions, in the sequel it will be assumed therefore that this is not the case, unless the contrary is stated.
The calculation is always succesfull if f or g is a factor of g or f.
The calculation is not successfull if f and g are of the form $f = a * x + b * x$ and $g = c * x + d * x$, where a, b, c, d and x are algebraic variables.
This is a consequence of the fact that the terms of f and the terms of g can not be divided by each other.
If only one algebraic variable occurs in both f and g then the calculation will be successfull in all those cases where a greatest common divisor exists.
Thus, COMMON DIVISOR will find out that for example $x^2 + 2 * x + 1$ and $x^2 - 1$ have the common divisor $2x + 2$.

The second property of simplification requires procedures which not only apply the distributive law, but also rules of the following kind:

1. $a/b + c = (a + b * c)/b$

2. $a + c/b = (a * b + c)/b$

3. $a/b * c = (a * c)/b$

4. $a * (b/c) = (a * b)/c$

5. $(a/b)/c = a/(b * c)$

6. $a/(b/c) = (a * c)/b$

It is evident, however, that these rules can not be applied without precautions, since this may lead to a simplified formula which is more complicated than the original formula.
Example: let a, b and x be algebraic variables.

The formulae:	would then be stored as:
(a/x) + (b/x)	(a * x + b * x)/(x * x)
(a/(b * x)) * x	(a * x)/(b * x)
((a * x)/b)/x	(a * x)/(b * x)

Simplifying the right hand sides amounts to calculating the common divisor of the numerator and denominator. This calculation will, however, be unsuccessfull.

Thus, within the procedures S, P and Q there is built in a test to prevent these situations.
It is for this reason that the already discussed procedure comm div is introduced, which replaces, in the first example e.g., the second x by 1.
The result of the calculation is

(1/x) * (a + b)

which is transformed by the procedure P into (a + b)/x.

Notice that the distributive law is applied in P only in case the a and/or b are no quotients. Otherwise, P would transform (1/x) * (a + b) into((1/x) * a + (1/x) * b), which is transformed into a/x + b/x, which is transformed by S into (1/x) * (a + b), etc., etc.

It is also for the reason, not to get a more complicated formula, that the parameters a and b of Q are simplified beforehand which is not necessary for the parameters of S and P.

If there occur simultaneously exponential functions and divisions, no common divisors are extracted from the simplified numerators and denominators.

9. The functions

The procedures EXP, LN, SIN, COS, ARCTAN and SQRT are now reproduced:

```
comment This is a part of the general system;
integer procedure EXP(f); value f; integer f;
begin integer t,a,b; t:= TYPE(f,a,b);
  if t = function ∧ a = lnf then EXP:= b else
  if t = number then
  begin real r,i; VALUE OF NUM(f,r,i); r:= exp(r);
    EXP:= CN(r × cos(i),r × sin(i))
  end else EXP:= STORE(expf,function,f)
end EXP;
integer procedure LN(f); value f; integer f;
begin integer t,a,b; t:= TYPE(f,a,b);
  if t = number then
  begin real r,i; VALUE OF NUM(f,r,i);
    LN:= if f = zero then RN(ln(null)) else CN(.5 × ln(r × r + i × i),
    (if abs(r) ≤ null then (2 - sign(i)) × 1.57079632679 else
     arctan(i/r) + (if r < 0 then 3.14159265359 else if i < 0 then
     6.28318530718 else 0)))
  end else LN:= STORE(lnf,function,f)
end LN;
integer procedure SIN(f); value f; integer f;
SIN:= if TYPE(f,di,di) = number ∨ expand then
  P(CN(0,-.5),D(EXP(P(im unit,f)),EXP(P(CN(0,-1),f)))) else
  STORE(sinf,function,f);
integer procedure COS(f); value f; integer f;
COS:= if TYPE(f,di,di) = number ∨ expand then
  P(RN(.5),S(EXP(P(im unit,f)),EXP(P(CN(0,-1),f)))) else
  STORE(cosf,function,f);
```

```
integer procedure ARCTAN(f); value f; integer f;
ARCTAN:= if TYPE(f,di,di) = number ∨ expand then
  P(CN(0,-.5),LN(Q(S(one,P(im unit,f)),D(one,P(im unit,f))))) else
  STORE(arctanf,function,f);
integer procedure SQRT(f); value f; integer f;
SQRT:= if TYPE(f,di,di) = number ∨ expand then
  EXP(P(RN(.5),LN(f))) else
  STORE(sqrtf,function,f);
```

A few remarks are made:

1. If expand = false the procedures store the functions as they are written, if expand = true the functions are transformed into exponential and logarithmic functions.
2. If the parameter f is a number, the result is another number.
3. The standard function arctan(x) delivers a result between $+\pi$ and $-\pi$.
4. The logarithm of a number will get an imaginary part y with $0 \leq y < 6.28318530718$ ($\approx 2\pi$).

10. The simplification procedures

Already in section 8 simplification was extensively discussed. This section is devoted, therefore, only to a discussion of the procedures to be reproduced.

The internally represented formula is, due to the construction of S, P, Q and INT POW, represented as a quotient of a numerator and a denominator, in which no quotients occur anymore.
The numerator and the denominator are represented as a sum of terms, and each term is a product of factors, i.e. integral powers of formulae which are neither a sum nor a product, nor a quotient.
The simplifying process has therefore as objects formulae which have the structure as given above for the numerator and the denominator.

The chosen tree structure for the internal representation of formulae is very convenient for storing and extracting formulae.
It is, however, very inconvenient for the simplification process, since in simplifying it will be used that the terms of a sum or the factors of a product can freely be permutated so as to compare the terms or the factors with each other.

Another way of representing the formulae presents itself. Use is made of an integer array $a[1:L_0, 0:L_1, 1:2]$, in which L_0 is the number of terms and L_1 the maximal number of factors of all terms.
$a[i, j, 1]$ is set equal to the factor, without the integral exponent; $a[i, j, 2]$ is set equal to this exponent.
The numbers occurring in the i-th term are combined to one number which is stored in $a[i, 0, 1]$.

Example: $-.5 \times x^2 \times 2i \times y \times i \times x + i \times y \times i \times x^2$ is represented as:

$(a[i, j, 1], a[i, j, 2])$

i =	j = 0	j = 1	j = 2	j = 3
1	one	(x,2)	(y,1)	(x,1)
2	IN (-1)	(y,1)	(x,2)	

For convenience sake a[i, 0, 2] is set equal to 1.
It is remarked that not all entries of a have to be filled, which is a pity. Use of this feature is, however, made when the procedure SIMPL 2 REPR interchanges the rows of a.

It is the task of the procedure CONVERT, which is now reproduced, to convert a formula from the tree structure representation into the direct representation using the arrays a and L.
If the actual value of the Boolean parameter "bounds only" of CONVERT is true, only the array bounds of a and L are calculated which are stored in L[0] and L[1].
The actual parameter a of CONVERT should then be some three dimensional integer array, say, B[1:1, 1:1, 1:1] (see the procedure bodies of SIMPLIFY and QUOTIENT).
In the case that the actual value of "bounds only" is false the array elements of a are filled in, moreover, the array elements L[1], ..., L[L[0]] are set equal to the lengths of the rows of a (in the above example L[0] = 2, L[1] = 3 and L[2] = 2).

```
comment This is a part of the general system;
procedure CONVERT(f,a,L,bounds only); value f; integer f;
integer array a,L; Boolean bounds only;
begin integer i,Li,Limax,num;
  procedure SUM(f); value f; integer f;
  begin integer f1,f2; if TYPE(f,f1,f2) = sum then
    begin SUM(f1); SUM(f2) end else
    begin Li:= 0; num:= one; i:= i + 1; PROD(f);
      if bounds only then Limax:= if Limax < Li then Li else Limax
      else begin a[i,0,1]:= num; a[i,0,2]:= 1; L[i]:= Li end
  end end;
```

```
procedure PROD(f); value f; integer f;
begin integer type,f1,f2; type:= TYPE(f,f1,f2); if type = product then
  begin PROD(f1); PROD(f2) end else
  if type = number ∧ ¬ bounds only then num:= P(num,f) else
  begin Li:= Li + 1; if ¬ bounds only then
    begin if type = integral power then
      begin a[i,Li,1]:= f2; a[i,Li,2]:= f1 end else
      begin a[i,Li,1]:= f; a[i,Li,2]:= 1 end
  end end
end;
i:= Limax:= 0; SUM(f); L[0]:= i; if bounds only then L[1]:= Limax
end CONVERT;
```

The procedure SIMPL 2 REPR is discussed now.
It transforms a formula given in the arrays a and L.
The result of this transformation, which may be called simplification, is another formula equal to the old one and also represented in the arrays a and L.
The form of the resulting formula is a standard form satisfying the following conditions:

1. In each row a[i, 1, 1], a[i, 2, 1], ..., at most one element may occur which is an exponential function. This element should be the last one, i.e. a[i, L[i], 1].

2. The argument of the possibly occurring exponential function is simplified; no numbers occur as summand in the argument.

   In the conditions 3-6 it will be assumed that the exponential function is temporarily removed by replacing L[i] by L[i] - 1, if a[i, L[i], 1] is an exponential function.

3. The elements of each row a[i, 0, 1], a[i, 1, 1], a[i, 2, 1], ... satisfy the conditions: a[i, 0, 1] ≠ zero and a[i, j, 1] > a[i, j+1, 1] for j = 1, ..., L[i] - 1.

4. Let for each row the integer l_i be defined as the number of elements $a[i, j, 1]$, including their multiplicity $a[i, j, 2]$. Then $l_i \geq l_{i+1}$ for $i = 1, \ldots, L[0] - 1$.

5. If $l_i = l_{i+1}$ then there are two possibilities for the elements $a[i, j, 1]$ and $a[i+1, j, 1]$, with $1 \leq j \leq \min(L[i], L[i+1])$ and $i = 1, \ldots, L[0] - 1$:

   first: all elements are equal,

   second: there exists a $j_0 > 0$ such that $a[i, j, 1] = a[i+1, j, 1]$ for $0 < j < j_0$ and $a[i, j_0, 1] > a[i+1, j_0, 1]$.

6. If $l_i = l_{i+1}$ and for all j $(1 \leq j \leq \min(L[i], L[i+1]))$, $a[i, j, 1] = a[i+1, j, 1]$, then there are two possibilities:

   first: $a[i, j, 2] = a[i+1, j, 2]$ for $j = 1, \ldots, L[i]$
   (Notice that in this case $L[i] = L[i+1]$).

   second: there exists a $j_1 > 0$ such that $a[i, j, 2] = a[i+1, j, 2]$ for $0 < j < j_1$ and $a[i, j_1, 2] > a[i+1, j_1, 2]$.

7. If the first possibility of condition 6 occurs, then the original elements $a[i, L[i], 1]$ and $a[i+1, L[i+1], 1]$ (before replacing $L[i]$ by $L[i] - 1$), are exponential functions whose arguments are unequal, or one of these elements is an exponential function.

Examples of formulae written in standard form are: (let $x > y > z$)

$x \times y \times z \times e^{x+y+z}$

$x \times y^2 \times z^3$

$z^3 + y^2 + x$

$.5 \times x^2 y - .5 \times x^2 z + 7i$ (i is the imaginary unit)

$x \times y^2 \times z + x \times y \times z^2$

$x * y * e^{z} + x * z * e^{y} + y * z * e^{x}$

$x * y * z * e^{x+y+z} + x * y * z * e^{-x-y-z}$

$x * y * z * e^{-x-y-z} + x * y * z * e^{x+y+z}$

$e^{2x} + e^{2y}$

$e^{2y} + e^{2x}$

The last four examples show that the ordering of terms in which exponential functions occur, does not need to be fixed if the non-exponential parts are equal.

The transformations carried out by the procedure SIMPL 2 REPR in order to get a standard form, can easily be traced in the now reproduced procedure declaration:

```
comment This is a part of the general system;
procedure SIMPL2REPR(a,L); integer array a,L;
begin integer k,i,j,i1,p,q,r,b,c,sexp; integer array exp,length[1:L[0]],
  s[1:2×L[0]];
  FIX; FIX; for i:= 1 step 1 until L[0] do
  begin sexp:= zero; for j:= 1 step 1 until L[i] do
    begin if TYPE(a[i,j,1],b,c) = function ∧ b = expf then
      begin sexp:= S(sexp,P(IN(a[i,j,2]),c)); a[i,j,1]:= -100 000
    end end;
    exp[i]:= SIMPLIFY(sexp); if exp[i] ≠ zero then
    begin if TYPE(exp[i],q,r) = number then
      begin a[i,0,1]:= P(a[i,0,1],EXP(exp[i])); exp[i]:= zero end else
      if TYPE(exp[i],q,r) = sum then
      begin if TYPE(r,b,c) = number then
        begin a[i,0,1]:= P(a[i,0,1],EXP(r)); exp[i]:= q end
end end end; for i:= 1 step 1 until L[0] do
begin p:= q:= 1; r:= L[i] - 1;
A: for j:= p step q until r do
```

```
    begin if a[i,j,1] = a[i,j+1,1] then
      begin a[i,j,2]:= a[i,j,2] + a[i,j+1,2]; a[i,j+1,1]:= -100 000;
        if q = -1 then begin i1:= j + 1; goto B end
      end else if a[i,j,1] < a[i,j+1,1] then
      begin k:= a[i,j,1]; a[i,j,1]:= a[i,j+1,1]; a[i,j+1,1]:= k;
        k:= a[i,j,2]; a[i,j,2]:= a[i,j+1,2]; a[i,j+1,2]:= k;
        if q = 1 then begin p:= j - 1; q:= -1; r:= 1; i1:= j + 1; goto A end
      end else if q = -1 then goto B
    end; B: if q = -1 then begin p:= i1; q:= 1; r:= L[i] - 1; goto A end;
    for j:= 1 step 1 until L[i] do
    begin if a[i,j,1] = - 100 000 then goto C end;
    goto D; C: L[i]:= j - 1;
  D: end;
  for i:= 1 step 1 until L[0] - 1 do for j:= i + 1 step 1 until L[0] do
  begin if a[i,0,1] = zero ∨ a[j,0,1] = zero ∨ L[i] ≠ L[j] then goto EXIT;
    for k:= 1 step 1 until L[i] do
    begin if a[i,k,1] ≠ a[j,k,1] ∨ a[i,k,2] ≠ a[j,k,2] then goto EXIT end;
    if ¬ EQUAL(exp[i],exp[j]) then goto EXIT;
    a[i,0,1]:= S(a[i,0,1],a[j,0,1]); a[j,0,1]:= zero;
  EXIT: end;
  for i:= 1 step 1 until L[0] do
  begin s[i]:= i; length[i]:= 0; if a[i,0,1] = zero then L[i]:= 0;
    for j:= 1 step 1 until L[i] do length[i]:= length[i] + a[i,j,2]
  end;
  p:= q:= 1; r:= L[0] - 1;
AA: for i:= p step q until r do
  begin if length[s[i]] > length[s[i+1]] then goto OUT;
    if length[s[i]] < length[s[i+1]] then goto INTERCHANGE;
    j:= 0; for j:= j + 1 while j ≤ L[s[i]] ∧ j ≤ L[s[i+1]] do
```

```
  begin if a[s[i],j,1] > a[s[i+1],j,1] then goto OUT;
    if a[s[i],j,1] < a[s[i+1],j,1] then goto INTERCHANGE;
    if a[s[i],j,2] > a[s[i+1],j,2] then goto OUT;
    if a[s[i],j,2] < a[s[i+1],j,2] then goto INTERCHANGE
  end; if a[s[i],0,1] = zero ∧ a[s[i+1],0,1] ≠ zero then goto
  INTERCHANGE; goto OUT;
INTERCHANGE: k:= s[i]; s[i]:= s[i+1]; s[i+1]:= k; if q = 1 then
  begin p:= i - 1; r:= 1; q:= -1; i1:= i + 1; goto AA end;
  goto CC;
OUT: if q = -1 then goto BB;
 CC: end; BB: if q = -1 then
begin p:= i1; q:= 1; r:= L[0] - 1; goto AA end;
for i:= 1 step 1 until L[0] do
begin if exp[i] ≠ zero ∧ a[i,0,1] ≠ zero then
  begin L[i]:= L[i] + 1; a[i,L[i],1]:= EXP(exp[i]);
    a[i,L[i],2]:= 1
end end;
p:= 0; for i:= 1 step 1 until L[0] do
begin p:= i; if s[i] ≠ i ∧ a[s[i],0,1] ≠ zero then
  begin j:= -1; for j:= j + 1 while j ≤ L[s[i]] ∧ j ≤ L[i] do
    begin i1:= a[i,j,1]; k:= a[i,j,2]; a[i,j,1]:= a[s[i],j,1];
      a[i,j,2]:= a[s[i],j,2]; a[s[i],j,1]:= i1; a[s[i],j,2]:= k
    end; for j:= L[s[i]] + 1 step 1 until L[i] do
    begin a[s[i],j,1]:= a[i,j,1]; a[s[i],j,2]:= a[i,j,2] end;
    for j:= L[i] + 1 step 1 until L[s[i]] do
    begin a[i,j,1]:= a[s[i],j,1]; a[i,j,2]:= a[s[i],j,2] end;
    k:= L[i]; L[i]:= L[s[i]]; L[s[i]]:= k;
    for j:= i + 1 step 1 until L[0] do
    begin if s[j] = i then begin s[j]:= s[i]; goto AB end end;
  AB: s[i]:= i
```

```
    end else if a[s[i],0,1] = zero then begin p:= p - 1; goto END end
  end; END: L[0]:= p;
  for i:= 1 step 1 until p do
  begin s[i]:= a[i,0,1]; s[i+p]:= if L[i] > 0 then a[i,L[i],1] else zero
  end; ERASE BUT RETAIN(i,1,2 × p,s[i]);
  for i:= 1 step 1 until p do
  begin a[i,0,1]:= s[i]; if L[i] > 0 then a[i,L[i],1]:= s[i+p] end
end SIMPL2REPR;
```

Remarks:

1. Two comparison tests are executed by SIMPL 2 REPR. These are constructed in such a way that almost no superfluous tests are carried out.
   An awkward consequence is their complicated structure.

2. The procedure ERASE BUT RETAIN (see sections 7 and 12) is used to store the arguments of the exponentials and the newly found numerical factors a[i, 0, 1], in an efficient way.

3. Use is made of the fact that the procedure SIMPLIFY stores the simplified formula in a special form.
   I.e. if the formula f is of the form g + n, where n is a number, then g does not contain a number as summand.

   This means that the characterizing quantities lhs, type and rhs of f are equal to g, sum and n.
   With this knowledge the exponential functions are treated.
   If f is of the form g + n, then e^f is represented as $e^n \times e^g$. The numerical part e^n is combined with the number a[i, 0, 1].
   A result of simplifying $e^{x+2\pi i}$ is e^x as a consequence of the above illustrated built in facility.

   Next follows the reproduction of the procedure declarations of SIMPLIFY and EQUAL. Due to the construction of the procedure Q, SIMPLIFY does not have to simplify a quotient.

```
comment This is a part of the general system;
integer procedure SIMPLIFY(f); value f; integer f;
begin integer i,j,t,a,b; integer array A[0:1],B[1:1,1:1,1:1];
   t:= TYPE(f,a,b); if t = quotient ∨ ¬ expand ∨ t = number ∨
   t = algebraic variable ∨ t = polynomial ∨ (t = function ∧ a ≠ expf)
   then begin SIMPLIFY:= f; goto END end;
   CONVERT(f,B,A,true);
   begin integer array s[1:A[0],0:A[1],1:2],L[0:A[0]]; CONVERT(f,s,L,false);
      SIMPL2REPR(s,L); t:= zero; for i:= 1 step 1 until L[0] do
      begin b:= s[i,0,1]; for j:= 1 step 1 until L[i] do
         b:= P(b,INT POW(s[i,j,1],s[i,j,2])); t:= S(t,b)
      end; SIMPLIFY:= t
   end;
END: end SIMPLIFY;
Boolean procedure EQUAL(f,g); value f,g; integer f,g;
if f = g then EQUAL:= true else
begin FIX; EQUAL:= SIMPLIFY(D(f,g)) = zero; ERASE
end EQUAL;
```

11. The procedures QUOTIENT and COMMON DIVISOR

In this section the (integral) division and the (greatest) common divisor algorithm are discussed.

The procedure QUOTIENT calculates the integral quotient q and the rest r of two formulae g and f according to the scheme:

$$g = q \times f + r.$$

To the procedure identifier QUOTIENT the value of q is assigned.
If there occur exponential functions in one or both formulae f and g then the division is not carried out.
QUOTIENT gets the value zero, and rest the value g.

The reproduction of the procedure declaration of QUOTIENT now follows:

```
comment This is a part of the general system;
integer procedure QUOTIENT(g,f,rest); value g,f;
integer g,f,rest;
begin integer i,j,k,lf,lg,quotient; Boolean first; integer array A[0:1],
  B[1:1,1:1,1:1]; FIX; FIX; CONVERT(f,B,A,true);
  begin integer array F[1:A[0],0:A[1],1:2],LF[0:A[0]];
    procedure QUOT(g,factor); value g; integer g,factor;
    begin integer f1; CONVERT(g,B,A,true);
      begin integer array G[1:A[0],0:A[1],1:2],LG[0:A[0]],GG[1:A[1]];
        CONVERT(g,G,LG,false); SIMPL2REPR(G,LG);
        if LG[0] = 0 then goto ZERO;
        if first then
        begin first:= false; for i:= 1 step 1 until LG[0] do
          begin if TYPE(G[i,LG[i],1],j,k) = function ∧
            j = expf then goto UNDEFINED
        end end; lg:= 0; for i:= 1 step 1 until LG[1] do
        begin lg:= lg + G[1,i,2]; GG[i]:= G[1,i,2] end;
        if lg < lf then goto ZERO; lg:= lf; k:= 1;
```

```
        for i:= 1 step 1 until LF[1] do
        begin for j:= k step 1 until LG[1] do
          begin if F[1,i,1] = G[1,j,1] then
            begin if F[1,i,2] ≤ G[1,j,2] then
              begin GG[j]:= G[1,j,2] - F[1,i,2]; lg:= lg - F[1,i,2] end
              else goto ZERO; k:= j + 1; goto AA
            end else if F[1,i,1] > G[1,j,1] then goto ZERO
          end;
        AA: end; if lg > 0 then goto ZERO;
        factor:= Q(G[1,0,1],F[1,0,1]); for i:= 1 step 1 until LG[1] do
        factor:= P(factor,INT POW(G[1,i,1],GG[i]));
        goto NEXT STEP;
      ZERO: factor:= zero; rest:= zero;
        for i:= 1 step 1 until LG[0] do
        begin k:= G[i,0,1]; for j:= 1 step 1 until LG[i] do
          k:= P(k,INT POW(G[i,j,1],G[i,j,2])); rest:= S(rest,k)
        end; goto OUT
      end;
    NEXT STEP: QUOT(D(g,P(factor,f)),f1); factor:= S(factor,f1);
    OUT: end QUOT;
    if ¬ expand then goto UNDEFINED;
    CONVERT(f,F,LF,false); SIMPL2REPR(F,LF);
    if LF[0] = 0 then goto UNDEFINED;
    for i:= 1 step 1 until LF[0] do
    begin if TYPE(F[i,LF[i],1],j,k) = function ∧ j = expf then
      goto UNDEFINED
    end; lf:= 0; for i:= 1 step 1 until LF[1] do lf:= lf + F[1,i,2];
    first:= true; QUOT(g,quotient); A[0]:= quotient; A[1]:= rest;
    ERASE BUT RETAIN(i,0,1,A[i]); QUOTIENT:= A[0]; rest:= A[1];
    goto END;
  UNDEFINED: QUOTIENT:= zero; rest:= g; ERASE; LOWER INDEX;
  END: end
end QUOTIENT;
```

In the second block of the procedure body of QUOTIENT, the arrays F and LF are declared. They are used for the second representation of the formula f in simplified form. Occurrence of exponential functions leads automatically to an end and QUOTIENT and rest get the values zero and g. Assuming that no exponential functions occur, first the length lf of the leading term of f is calculated, which is just equal to the number of multiplicants including their multiplicity of the leading term.

The Boolean variable first becomes true.

Next follows a call for the procedure QUOT.

In fact, QUOT calculates in a recursive way the quotient q of g and f, according to the following scheme:

$$r_0 = y = q_1 * f + r_1$$

$$r_1 = q_2 * f + r_2$$

$$\ldots\ldots\ldots\ldots\ldots$$

$$r_{n-1} = q_n * f + r_n$$

in which q_i is the quotient of the leading term of r_{i-1} and the leading term of f.

The process ends as soon as the leading term of f is not contained in the leading term of r_n.

With this is meant: apart from a numerical factor, not all factors of the leading term of f are also factors of the leading term of r_n.

That this process always ends follows from a theorem proved later on.

Evidently:

$$q = \sum_{i=1}^{n} q_i \quad \text{and} \quad r = r_n.$$

Care has been taken of the following points:

1. The original formula g should be examined as to whether there occur exponential functions in it. This is done the first time QUOT is called, then namely the Boolean variable first has the value true, in all subsequent calls first has the value false.

2. The partial results $q_1, q_2, \ldots, q_n$ should be traced.
   This is established by the statement:
   factor := S(factor,f1)
   which in fact means: $q := q + q_i$.

3. The final rest r_n should be extracted, which is done by assigning to the (with respect to QUOT) non-local variable rest the value of the formula r_n.
   This formula is already simplified.

4. By means of a call for ERASE BUT RETAIN, the intermediate formulae are erased and the results q and r_n are stored in an efficient way.

A call for QUOT is now described:
The parameters g and factor refer to the above quantities r_i and q_{i+1}.

1. g is represented in a simplified form in the arrays G and LG.

2. If g turns out to be zero (LG[0] = 0) then QUOT ends via the label ZERO.
   factor gets the value zero and rest gets the value zero.

3. If first = true, then first is changed into false and it is tested whether there occur exponential functions in g; if so,then QUOT terminates via the label UNDEFINED.

4. It is examined whether the leading term of f is contained in the leading term of g, if this is not the case the process ends via the label ZERO, otherwise the process continues via the label NEXT STEP and by means of the procedure statement QUOT(D(g,P(factor, f)),f1) the quotient f1 of g - factor * f is calculated.
   factor becomes equal to factor + f1 and the process terminates.
   It should be noticed that QUOT is called recursively in a block in which the arrays G and LG do not exist any longer, if it was not arranged in this way then the arrays G and LG would exist simultaneously for all r_i, which would be wasting of storage space.

5. Execution of the statement following the label ZERO leads to assigning to the non-local variable rest, the simplified formula g (r_n in the above scheme).
It should be remarked that although QUOT may be recursively called hundreds of times, only the last time it will end via ZERO; all other times it ends directly via the label OUT.

Theorem: The process as defined by the procedure QUOTIENT will ultimately end.

Remark: It can easily be seen that if no precautions were taken against exponential functions, the process does not need to end. One should try to divide $e^{2x} - e^{2y}$ through $e^{y} + e^{x}$.

Proof: Consider the above shown process:

$$r_n = q_{n+1} * f + r_{n+1},\ n = 0, 1, 2, \ldots \text{ and } r_0 = g.$$

Let the number of factors including the multiplicity, of the leading term of r_n be l_n.

If the process would not end, it may be assumed that l_n remains constant from a certain n_0; since it is easily seen that the l_n can not increase.

Let $\psi_1, \psi_2, \ldots, \psi_m$ denote the set of different factors, apart from the integral powers and apart from numbers, which occur in both f and g (i.e. the array elements a[i, j, 1] in section 10).
Let this set be ordered according to $\psi_{i+1} < \psi_i$.
Let the maximal integral power of these multiplicants as they occur in f and g be $p_1, p_2, \ldots, p_m$.
Let the maximum of all p_i be P.
Then the norm of a term t

$$t = \psi_{i_1}^{p_{i_1}} * \ldots * \psi_{i_k}^{p_{i_k}}$$

is defined as:

$$N(t) = \sum_{l=1}^{k} (2^P)^{\psi_{i_l}} * p_{i_l}.$$

Consider:

$$r_{n_0} = q_{n_0+1} * f + r_{n_0+1}.$$

Let

$$f = \alpha_1 * \phi_{1,1} * \ldots * \phi_{1,i_1} + \ldots + \alpha_p * \phi_{p,1} * \ldots * \phi_{p,i_p}$$

$$r_{n_0} = \beta_1 * \rho_{1,1} * \ldots * \rho_{1,j_1} + \ldots + \beta_q * \rho_{q,1} * \ldots * \rho_{q,j_q}$$

in which $\alpha_1, \ldots, \alpha_p, \beta_1, \ldots, \beta_q$ are numerical constants and in which the $\phi_{n,m}$ and $\rho_{n,m}$ are the factors of the terms of f and r_{n_0}, without integral powers, which are written out ($x^3 * y^2$ is represented as $x * x * x * y * y$).

Assume the above representations are the result of the simplification process (apart from the integral powers) as defined by the procedure SIMPL 2 REPR.
Then there are two possibilities for the f terms:

or $i_1 > i_2$,

or $i_1 = i_2$ and $N(\phi_{1,1} * \ldots * \phi_{1,i_1}) > N(\phi_{2,1} * \ldots * \phi_{2,i_2})$

which follows from the definition of standard form in section 10.

In the same way there are two possibilities for the r_{n_0} terms:

or $j_1 > j_2$,

or $j_1 = j_2$ and $N(\rho_{1,1} * \ldots * \rho_{1,j_1}) > N(\rho_{2,1} * \ldots * \rho_{2,j_2})$.

Consider now r_{n_0+1}:

$$r_{n_0+1} = -\left(\frac{\beta_1}{\alpha_1}\right) * \frac{\rho_{1,1} * \ldots * \rho_{1,j_1}}{\phi_{1,1} * \ldots * \phi_{1,i_1}} * (\alpha_2 * \phi_{2,1} * \ldots * \phi_{2,i_2} + \ldots$$

$$\ldots + \alpha_p * \phi_{p,1} * \ldots * \phi_{p,i_p}) + \beta_2 * \rho_{2,1} * \ldots * \rho_{2,j_2} + \ldots +$$

$$+ \beta_q * \rho_{q,1} * \ldots * \rho_{q,j_q}.$$

There are two possibilities:

1. The leading term of r_{n_0+1} is after simplification:

$$\beta_2 * \rho_{2,1} * \ldots * \rho_{2,j_2}.$$

Since $l_{n_0+1} = j_2$ and $l_{n_0} = l_{n_0+1}$, it follows $j_2 = j_1$.
Thus N (leading term of r_{n_0+1}) < N (leading term of r_{n_0}).

2. The leading term of r_{n_0+1} is after simplification:

$$-\left(\frac{\beta_1 * \alpha_2}{\alpha_1}\right) * \frac{\rho_{1,1} * \ldots * \rho_{1,j_1}}{\phi_{1,1} * \ldots * \phi_{1,i_1}} * \phi_{2,1} * \ldots * \phi_{2,i_2}.$$

Now $l_{n_0+1} = j_1 - i_1 + i_2$ and $l_{n_0+1} = l_{n_0}$, thus $i_1 = i_2$.

Moreover:

$$N(\text{leading term of } r_{n_0+1}) = N(\rho_{1,1} * \ldots * \rho_{1,j_1}) - N(\phi_{1,1} * \ldots * \phi_{1,i_1}) + N(\phi_{2,1} * \ldots * \phi_{2,i_2}),$$

and it follows that:

$$N(\text{leading term of } r_{n_0+1}) < N(\text{leading term of } r_{n_0}).$$

Concluding: the norm N of the leading terms of r_n with $n \geq n_0$ diminishes. It is obvious that the process can not continue for ever, since the norm N can not diminish for ever.

Next the procedure declaration of COMMON DIVISOR is reproduced. COMMON DIVISOR calculates the (greatest) common divisor of the formulae f and g.

```
comment This is a part of the general system;
integer procedure COMMON DIVISOR(f,g); value f,g; integer f,g;
begin integer gcd,q,r; Boolean s,s1;
  procedure GCD(f1,f2,f3); value f1,f2; integer f1,f2,f3;
  begin integer f4; q:= QUOTIENT(f1,f2,f3); s:= q ≠ zero;
```

```
    if ¬ s1 ∧ ¬ s then goto UNDEFINED;
    if f3 = zero then
    begin gcd:= f2; goto if TYPE(gcd,di,di) = number then
      UNDEFINED else OK
    end; s1:= s; GCD(f2,f3,f4)
  end;
  FIX; FIX; s1:= true; GCD(f,g,r);
OK: ERASE BUT RETAIN(q,1,1,gcd); COMMON DIVISOR:= gcd; goto END;
UNDEFINED: COMMON DIVISOR:= one; ERASE; ERASE;
END: end COMMON DIVISOR;
```

The process for calculating the (greatest) common divisor of two formulae f and g can be described as follows:

Let f_1 = f and f_2 be g:

$$f_1 = q_1 * f_2 + f_3$$

$$f_2 = q_2 * f_3 + f_4$$

.....

$$f_n = q_n * f_{n+1}$$

and the (greatest) common divisor of f and g is given by f_{n+1}.

The above process is executed by means of the procedure GCD declared in COMMON DIVISOR. Its parameters f1, f2 and f3 have the meaning of f_{i+1}, f_{i+2} and f_{i+3} (i = 0, ..., n-1).

By a call for QUOTIENT the rest of the division of f1 over f2 is assigned to f3.

If this division was successfull i.e. q ≠ 0, then the Boolean variable s gets the value true.

If f3 turns then out to be zero, the (greatest) common divisor is found (i.e. f2) and GCD comes directly to an end via the label OK or the label UNDEFINED depending on whether f2 is unequal or equal to a number.

If the division was successfull, but f3 is not equal to zero then the process is repeated by a recursive call for GCD(f2, f3, f4).

In case the division was unsuccessfull, this does not mean that there does not exist a (greatest) common divisor (one may have e.g. f1 = x+1 and f2 = x^2-1).
However, if two successive divisions are unsuccessfull then the process is terminated via the label UNDEFINED.
In order to test whether two successive divisions are unsuccessfull, the, with respect to GCD, non-local Boolean variables s1 and s are introduced. s1 has the value true or false depending on the success of the foregoing division, s has the value true or false depending on the success of the division just executed.
If both s1 and s have the value false, GCD ends via the label UNDEFINED.
That this test is sufficient in order to get a process which always terminates follows from the theorem to be stated after the following remarks.

1. The result of the procedure COMMON DIVISOR is restored by a call for ERASE BUT RETAIN in order to erase possible intermediately formed but no more interesting formulae.

2. The reason, for putting parentheses around the word "greatest" in "(greatest) common divisor", is that in a lot of cases the procedure COMMON DIVISOR will not find a common divisor, although it trivially exists.
   For example: the (greatest) common divisor of
   $x^2 + 2xy + y^2$ and $x^2 - y^2$
   is not found. It is easily seen why it is not found:
   if $f_1 = x^2 + 2xy + y^2$ and $f_2 = x^2 - y^2$
   then $q_1 = 1$ and $f_3 = 2xy + 2y^2$
   $q_2 = 0$ and $f_4 = f_2 = x^2 - y^2$
   $q_3 = 0$ and $f_5 = f_3 = 2xy + 2y^2$
   and the process stops.

It could only be continued if it was allowed to use as factor $1/y$ in order to find $q_4 = \frac{1}{2}\frac{x}{y} - \frac{1}{2}$.

Then f_6 would be zero and the doubtful (greatest) common divisor would be the formula $f_5 = 2xy + 2y^2$.
This, however, would be not as bad as the fact that, by allowing QUOTIENT to use factors of this kind, the process as defined by QUOTIENT would in general never end.

A consequence of forbidding QUOTIENT to use factors of the kind $1/y$ is that the quotient $(x^2 + 2xy + y^2)/(x^2 - y^2)$ will not be simplified to, say, $(x + y)/(x - y)$.
These situations, however, only occur when the user gives explicitly the numerator and the denominator of the quotient the values of the above formulae.
If the user had given his quotient in the following way

$$(x + y) \times (((x + y)/(x - y))/(x + y))$$

for example, then the procedures S, D and P automatically divide out the common factor x+y.

Next follows the already mentioned

Theorem: The process as defined by COMMON DIVISOR, always terminates.

If the formulae f and g contain exponential functions, the theorem follows trivially, assume therefore that this is not the case.

Proof: Let the lengths l_n of the leading term of f_n be defined as in the proof of the theorem about the procedure QUOTIENT.

Assume the l_n do not decrease for $n \geq m$ (if this is not the case the process has to terminate).

Consider the step:

$$f_{m-1} = q_{m-1} \times f_m + f_{m+1}$$

in which q_{m-1} and f_{m+1} are calculated by QUOTIENT according to the following scheme:

$$f_{m-1} = r_0 = \bar{q}_1 * f_m + r_1$$

$$\cdots\cdots\cdots\cdots\cdots$$

$$r_k = \bar{q}_{k+1} * f_m + r_{k+1}.$$

Dividing r_{k+1} through f_m turned out to be unsuccessfully thus, the leading term of f_m is not contained in the leading term of r_{k+1}. This means that the leading term of f_m contains at least one factor which is no factor in the leading term of r_{k+1}.
Since $r_{k+1} = f_{m+1}$, the same statement is true for the leading terms of f_m and f_{m+1} (*).
The next step of COMMON DIVISOR is to calculate q_m and f_{m+2} from:

$$f_m = q_m * f_{m+1} + f_{m+2}.$$

Dividing f_m through f_{m+1} will obviously turn out to be unsuccessfully; thus $q_m = 0$ and $f_{m+2} = f_m$, which follows from the facts:

$$l_{m+1} \geq l_m \text{ and } (*).$$

The following step is to divide f_{m+1} through f_{m+2} which means dividing f_{m+1} through f_m, which also will be unsuccessfully due to (*) and for a second time $q_{m+1} = 0$.

From this the theorem follows.

12. Storage allocation

This section contains the procedure declarations of the procedures COPY and ERASE BUT RETAIN.
Moreover, the consequences of a garbage collection method are discussed.

First the procedure COPY is reproduced which becomes equal to a copy of the formula f, in a possibly changed form, depending on the actual parameter F SPECIAL.
COPY is used within ERASE BUT RETAIN, within CC the complex conjugating procedure and within SUBSTITUTE.
It is possible to give as actual parameter for F SPECIAL a Boolean procedure which, as a side effect, changes the value of f in the procedure body of COPY (see CC and SUBSTITUTE).
The procedure COPY can also be used in case some formula should be simplified which is not stored in expanded form (i.e. expand = false). One then should change the value of expand into true and use as actual parameter the Boolean procedure TRUE declared by:
Boolean procedure TRUE(f); integer f; TRUE := true;

Next follows the reproduction of COPY and ERASE BUT RETAIN.

```
comment This is a part of the general system;
integer procedure COPY(f,F SPECIAL); value f; integer f;
Boolean procedure F SPECIAL;
begin integer t,a,b; if F SPECIAL(f) then
  begin COPY:= f; goto END end; t:= TYPE(f,a,b);
  if t = sum ∨ t = difference ∨ t = product ∨ t = quotient
  ∨ t = power then
  begin a:= COPY(a,F SPECIAL); b:= COPY(b,F SPECIAL);
    COPY:= if t = sum then S(a,b) else if t = difference
    then D(a,b) else if t = product then P(a,b) else
    if t = quotient then Q(a,b) else POWER(a,b)
  end else if t = function ∨ t = integral power then
  begin b:= COPY(b,F SPECIAL); COPY:= if t = integral power
```

```
    then INT POW(b,a) else if a = expf then EXP(b) else
    if a = lnf then LN(b) else if a = sinf then SIN(b)
    else if a = cosf then COS(b) else if a = arctanf
    then ARCTAN(b) else SQRT(b)
  end else if t = number then
  begin real r,i; VALUE OF NUM(f,r,i); COPY:= CN(r,i) end
  else if t = polynomial then
  begin integer i,x; integer array co[0:a];
    COEFFICIENT(f,a,x,co);
    for i:= 0 step 1 until a do co[i]:= COPY(co[i],F SPECIAL);
    x:= COPY(x,F SPECIAL); COPY:= POL(i,a,x,co[i])
  end else COPY:= f;
END: end COPY;

procedure ERASE BUT RETAIN(i,lb,ub,fi); integer i,lb,ub,fi;
begin for i:= lb step 1 until ub do fi:= COPY(fi,FIXED);
   ERASE; for i:= lb step 1 until ub do fi:= COPY(fi,FIXED);
  LOWER INDEX
end ERASE BUT RETAIN;
```

As was already said in section 7, a call for the procedure ERASE BUT RETAIN should be preceded by two calls for FIX, then the formulae given by fi, i = lb, ..., ub, which may be stored together with other formulae in the program after the two calls for FIX, are restored by COPY in such a way that they occupy the minimum amount of storage space; the other formulae are all erased.

It should be noticed that the Boolean procedure FIXED is used to determine whether the formulae fi or constituants of fi are stored before the two statements FIX were executed; in that case the copying process is terminated and COPY becomes equal to the formula which it should copy.

One should be very carefull in using the procedures ERASE and ERASE BUT RETAIN. It is forbidden to use a formula which was erased by a call for ERASE or ERASE BUT RETAIN, otherwise it is very easy to get a catastrophe by obtaining e.g. the formula f = a + f. Outputting this formula would result in:

a + a + a + a + ...

Since the fi get in general other values by a call for ERASE BUT RETAIN, care should be taken not to get a situation exemplified by:
FIX; FIX; S1; f := g := S(a, P(b, c)); S2; ERASE BUT RETAIN(i, 1, 1, f);
in which S1 and S2 are statements by which other formulae then f and g are stored.
After execution of these statements f corresponds to the formula a + b − c, however, g does not need to correspond any longer to this formula.
In fact, these statements should be followed by the statement g := f.

The above remarks show the dangers in using the erasing procedures.

The consequences of using a garbage collection method, instead of the more easy but dangerous (with respect to erasing) method which is used in the general system, are now discussed.

At a certain (but arbitrary) moment, during the execution of a formula manipulating program, there exist two sets of formulae.
One set, called FIXED, consists of all those formulae which will be used later on; the other set consists of all those formulae which may be erased.

Notice that all formulae which are stored to build up another formula f, e.g. the P(b, c) in f := S(a, P(b, c)), should also occur in FIXED; after their use they should be removed from FIXED, since then they are protected from erasing through the appearance of f in FIXED.

At some moment it may turn out that not enough storage space in the computer is left (this has to be checked at several appropriate places e.g. in the storing procedures and in the procedure CONVERT).
Then all formulae which may be erased should actually be erased, and the formulae in FIXED should be restored in order to get free storage space.

This means that the formulae in FIXED should be kept on a list (called LIST).

One can choose for LIST the integer array LIST[1:1000] (which of course can also be an own integer array).
A formula of FIXED refers to some place i in LIST, while LIST[i] refers to the internally represented formula.

The following procedures can be used for storing and removing a formula on and from LIST:

```
integer procedure SOL(f); value f; integer f;
begin comment pl is used as pointer for LIST;
      SOL := pl := pl + 1; LIST[pl] := f
end;
procedure RFL; pl := pl - 1;
```

By means of SOL and RFL, the procedure S of section 2 may be changed into:

```
integer procedure S(a, b); value a, b; integer a, b;
begin integer a1, b1; a1 := LIST[a]; b1 := LIST[b];
      RFL; RFL;
      S := SOL(if a1 = LIST[zero]then b1 else
               if b1 = LIST[zero]then a1 else
               STORE(a1, sum, b1))
end;
```

A piece of program might be:

```
x := SOL(STORE(0, algebraic variable, 0));
y := SOL(STORE(0, algebraic variable));
f := S(x, y);
```

Execution of this piece of program would deliver a correct formula for f, however x and y are removed from LIST and they cannot be used any longer.
Moreover, if the last statement was followed by: g := S(f,f), then f is removed from LIST and what is worse, pl is automatically decreased too much.
Not automatically decreasing the pointer pl, however, would result in putting all formulae on LIST and the obtained system would be equivalent but more complicated then the general system of this report.

Another way of construction is to let the formula identifiers refer directly to the internal representation of the corresponding formulae. They should however be put on LIST.

In this construction it is no longer possible to let the garbage collection routine restore the formulae. Since then the value of the formula identifiers would no longer correspond to the corresponding formulae and it is impossible to change this value.
The storage places containing formulae which may be erased can be flagged, in such a way that new formulae are stored in these flagged storage places.
One would then have instead of the above piece of program:

x := STORE(0, algebraic variable, 0); SOL(x);

y := STORE(0, algebraic variable, 0); SOL(y);

f := LIST[S(SOL(x), SOL(y))]; RFL; SOL(f);

g := LIST[S(SOL(f), SOL(F))]; RFL;

(Notice that in this construction the expression LIST[zero], occurring in the procedure body of S, has to be replaced by zero).

Obviously this is an awkward construction.

It should be remarked that choosing a method by which the formulae which may be erased appear on a list, is as bad as the above shown method. Since then one should also have a list of formulae which may not be erased.

Example: f := P(zero, a) should be executed, such that a is placed on the list of eraseable formulae, however a may be a formula which still is needed later on.

It seems likely that introducing a garbage collection system in a process to be described in ALGOL 60, will lead to a similar construction as shown above.
Concluding, if one takes for granted the way the formulae are erased as described in the first part of this section and in section 7, then the method by which formulae are treated by the general system is much easier to use than a garbage collection method.

It is remarked that writing the general system in ALGOL 66, using the new concept: record, the above displayed difficulties in storage allocation no longer exist, since the treatment of records themselves involves a garbage collection method.

13. Supplementary equipment

First the procedure DER is reproduced which calculates the derivative of a formula f with respect to the algebraic variable x (for x even an arbitrary formula may be substituted).
The formula f may be given in expanded as well as in not-expanded form. The result of differentiating a polynomial is, when expand = true, a simplified polynomial.

```
comment This is a part of the general system;
integer procedure DER(f,x); value f,x; integer f,x;
begin integer t,a,b; t:= TYPE(f,a,b);
  if f = x then DER:= one else
  if t = sum then DER:= S(DER(a,x),DER(b,x)) else
  if t = difference then DER:= D(DER(a,x),DER(b,x)) else
  if t = product then DER:= S(P(DER(a,x),b),P(a,DER(b,x))) else
  if t = quotient then DER:= Q(D(P(DER(a,x),b),
                                 P(a,DER(b,x))),P(b,b)) else
  if t = power then DER:= P(f,DER(P(b,LN(a)),x)) else
  if t = integral power then DER:=
    P(IN(a),P(INT POW(b,a-1),DER(b,x))) else
  if t = polynomial then
  begin integer i,y; integer array coeff[0:a]; COEFFICIENT(f,a,y,coeff);
    DER:= S(POL(i,0,a,DER(coeff[i],x)),
    POL(i,0,a,if i = a then zero else P(IN(i+1),P(coeff[i+1],DER(y,x)))))
  end else
  if t = function then
  begin integer d; d:= DER(b,x);
    DER:= if a = expf then P(f,d) else
          if a = lnf then P(INT POW(b,-1),d) else
```

```
        if a = sinf then P(COS(b),d) else
        if a = cosf then P(min one,P(SIN(b),d)) else
        if a = arctanf then P(Q(one,S(one,P(b,b))),d) else
        P(Q(RN(.5),f),d)
  end else DER:= zero
end DER;
```

The next procedure to be reproduced is CC which calculates the complex conjugate of a formula f. It is assumed that all algebraic variables are real algebraic variables.
Thus the complex variable z, e.g. should be replaced by x + iy.
If the algebraic variables are not real then the procedure CHANGE should appropriately being changed in such a way that it defines the complex conjugate of the algebraic variables.
This, however, implies that there should exist besides the algebraic variables also their complex conjugates which may be given by the lhs or rhs quantities (see section 5).

```
comment This is a part of the general system;
integer procedure CC(f); value f; integer f;
begin Boolean procedure CHANGE(g); integer g;
  begin integer a,b; if TYPE(g,a,b) = number ∧ a = complex then
    begin real r,i; VALUE OF NUM(g,r,i); g:= CN(r,-i); CHANGE:= true
    end else CHANGE:= false
  end;
  CC:= COPY(f,CHANGE)
end CC;
```

Now the procedure SUBSTITUTE is reproduced. SUBSTITUTE becomes equal to a formula f in which all constituants given by argument i (i = 1, ..., n) are changed into the formulae value i (i = 1, ..., n).

If, in particular, f happens to be a polynomial and the dependent variable x of f occurs as one of the arguments, then f is transformed into an extended sum in which the x is replaced by the value of the corresponding argument.

```
comment This is a part of the general system;
integer procedure SUBSTITUTE(f,i,lb,ub,argument i,value i);
value f,lb,ub; integer f,i,lb,ub,argument i,value i;
begin integer array argument,value[lb:ub];
  Boolean procedure SUBST(g); integer g;
  begin integer a,b,i,j; if TYPE(g,a,b) = polynomial then
    begin integer x; integer array coeff[0:a];
      COEFFICIENT(f,a,x,coeff);
      for i:= lb step 1 until ub do
      begin if x = argument[i] then
        g:= Sum(j,0,a,P(SUBSTITUTE(coeff[j],b,lb,ub,argument[b],
          value[b]),INT POW(value[i],j)));
        SUBST:= true; goto END
    end end; for i:= lb step 1 until ub do
    begin if g = argument[i] then
      begin g:= value[i]; SUBST:= true; goto END end
    end; SUBST:= false;
  END: end;
  for i:= lb step 1 until ub do
  begin argument[i]:= argument i; value[i]:= value i end;
  SUBSTITUTE:= COPY(f,SUBST)
end SUBSTITUTE;
```

Finally the procedure COEFF OF PRODUCTS is reproduced. Given a formula f and an array: product[1:n], by which n products are given, then the result of executing COEFF OF PRODUCTS is that the variables coeff[i], i = 1, ..., n, become equal to the formulae defining the coefficients of the products in f. Thus, for example, if

$$f = x \times a \times b + y \times a \times b + z \times b \times c \times e^{x}$$

and product[1] = a × b, product[2] = c × e^{x} then coeff[1] and coeff[2] will get the values of the formulae x + y and z × b respectively.

It is required that:

1. expand = true,
2. the products do not contain the operators +, - and /.

```
comment This is a part of the general system;
procedure COEFF OF PRODUCTS(f,n,prod,coeff); value f,n;
integer f,n; integer array prod,coeff;
begin integer k,i,j,q,r,p,e; integer array A[0:1],B[1:1,1:1,1:1];
  if ¬ expand then f:= zero; CONVERT(f,B,A,true);
  begin integer array a[1:A[0],0:A[1],1:2],L,exp,TF[0:A[0]];
    FIX; FIX; CONVERT(f,a,L,false); SIMPL2REPR(a,L);
    for i:= 1 step 1 until L[0] do
    begin TF[i]:= a[i,0,1]; if TYPE(a[i,L[i],1],j,k) = function ∧
      j = expf then
      begin exp[i]:= k; L[i]:= L[i] - 1 end else exp[i]:= zero;
      for j:= 1 step 1 until L[i] do
      TF[i]:= P(TF[i],INT POW(a[i,j,1],a[i,j,2]))
    end; for k:= 1 step 1 until n do
    begin p:= SIMPLIFY(prod[k]); e:= zero;
      if TYPE(p,q,r) = function ∧ q = expf then
      begin p:= one; e:= r end else
      if TYPE(p,q,r) = product then
      begin if (TYPE(r,i,j) = function ∧ i = expf) then
```

```
        begin p:= q; e:= j end
      end; coeff[k]:= zero; for i:= 1 step 1 until L[0] do
      begin q:= QUOTIENT(TF[i],p,r);
        if exp[i] ≠ zero ∨ e ≠ zero then
        q:= P(q,EXP(D(exp[i],e)));
        coeff[k]:= S(coeff[k],q)
      end
    end; ERASE BUT RETAIN(i,1,n,coeff[i])
end end COEFF OF PRODUCTS;
```

14. Outputting

By means of a simple actual program the procedures OUTPUT and OUTPUT VARIABLE will be introduced.

A set of not declared procedures is used, these are MC standard procedures; see [11]. The effect of execution of these procedures is first described:

procedure NLCR; gives the printer a New Line Carriage Return command.

procedure ABSFIXT(n, m, x); value n, m, x; integer n, m; real x;

prints the value of x without its sign in fixed point notation: n decimals before and m decimals behind the decimal point. In case m = 0 then the printing of the decimal point is skipped.

procedure FIXT(n, m, x); value n, m, x; integer n, m; real x;

has the same effect as ABSFIXT, now however the sign of x is also printed.

procedure FLOT(n, m, x); value n, m, x; integer n, m; real x;

prints the value of x in floating point notation: n decimals of the mantissa and m decimals of the exponent (with the base 10).

procedure PRINTTEXT(s); string s;

prints the string s without the string quotes ‡ and †.

Now the actual program, which does some particular formula manipulation, in fact, polynomial manipulation, is reproduced, followed by the results.

```
  INITIALIZE;
  comment RPR 290466/06;
ACTUAL PROGRAM:
  begin
    comment The following calculation originated from the
      Handbook of Mathematical Functions [12], page 15,
      formula 3.6.24:
      Let
        s[1] = POL(i,5,x,if i = 0 then one else a[i])
      (in ordinary notation:
        s[1] = 1 + a[1] × x + ... + a[5] × x ↑ 5 ),
      calculate then the c[i] in:
        s[3] = POL(i,5,x,c[i]),
      which is the truncated power series, in the variable x, for
        f(s[1]) = S(one,LN(s[1]))
      (in ordinary notation: f(s[1]) = 1 + ln(s[1]) ).
      The calculation is performed along the following steps:
    1. Calculate the first five derivatives d[1], ... ,d[5] of f with
       respect to s[1].
    2. Substitute s[1] = one in d[i].
    3. Calculate the coefficients b[i] in:
         g(y) = POL(i,5,y,b[i]),
       which is the truncated power series, in the variable y, for
         f(S(one,y)) (in ordinary notation: f(1 + y)).
    4. Calculate the c[i] by substituting y = D(s[1],one)
       (in ordinary notation: y = s[1] - 1), in g(y).;

    integer f,g,x,y,i,j,fact; integer array a[1:5],b,d[0:5],s[1:4];
    procedure PR(s); string s; PRINTTEXT(s);
    procedure OUTPUT(f); value f; integer f;
    begin integer t,a,b; t:= TYPE(f,a,b); if ¬ expand then PR(↓(↓);
```

```
if f = zero then PR(‹0›) else if f = one then PR(‹1›) else
if t = sum ∨ t = difference ∨ t = product ∨ t = quotient ∨
t = power then
begin if t = quotient ∧ expand then PR(‹(›); OUTPUT(a);
  if t = sum then PR(‹ + ›) else if t = difference then PR(‹ - ›)
  else if t = product then PR(‹ × ›) else if t = quotient then
  begin if expand then PR(‹)/(›) else PR(‹ / ›) end else
   PR(‹ ↑ ›); OUTPUT(b); if t = quotient ∧ expand then PR(‹)›)
end else
if t = function then
begin if a = expf then PR(‹exp(›) else if a = lnf then PR(‹ln(›)
  else if a = sinf then PR(‹sin(›) else if a = cosf then PR(‹cos(›)
  else if a = arctanf then PR(‹arctan(›) else PR(‹sqrt(›);
  OUTPUT(b); PR(‹)›)
end else
if t = integral power then
begin PR(‹(›); OUTPUT(b); PR(‹)↑›); ABSFIXT(2,0,a) end else
if t = number then
 begin integer i; real ra,ia; PR(‹(›);
  if a = integer then
  begin VALUE OF INT NUM(f,i); FIXT(entier(ln(abs(i))/
    2.30258509299 + 1.00001),0,i)
  end else
  if a = real then
  begin VALUE OF REAL NUM(f,ra); FLOT(12,3,ra) end else
  begin VALUE OF COMPLEX NUM(f,ra,ia); FLOT(12,3,ra); PR(‹,›);
    FLOT(12,3,ia)
  end; PR(‹)›)
end else
if t = polynomial then
begin integer i,x; integer array coeff[0:a];
```

```
      COEFFICIENT(f,a,x,coeff); for i:= 0 step 1 until a do
      begin PR(‹(›); OUTPUT(coeff[i]); PR(‹) × (›); OUTPUT(x);
        PR(‹)^›); ABSFIXT(2,0,i); if i < a then PR(‹ + ›)
      end
    end else OUTPUT VARIABLE(f);
    if ¬ expand then PR(‹)›)
  end OUTPUT;

  procedure OUTPUT VARIABLE(f); value f; integer f;
    comment The structure of the following procedure body could be
      made more simple, this structure, however, indicates how the
      lhs and rhs quantities of an algebraic variable can be used
      in a case where a large number of algebraic variables occur;
  begin integer i,lhs,rhs; switch CASE:= X,Y,S1;
    procedure A(g,s); integer g; string s;
    if f = 0 then
    begin i:= i + 1; g:= STORE(1,algebraic variable,i) end
    else begin PR(s); goto END end;
    if f = 0 then
    begin for i:= 1,2,3,4,5 do a[i]:= STORE(2,algebraic variable,i);
      i:= 0; goto CASE[1]
    end else
    begin TYPE(f,lhs,rhs); if lhs = 1 then goto CASE[rhs] else
      begin PR(‹a[›); ABSFIXT(1,0,rhs); PR(‹]›); goto END end
    end;
  X: A(x,‹x›);
  Y: A(y,‹y›);
  S1: A(s[1],‹s[1]›);
  END: end OUTPUT VARIABLE;
```

```
BEGIN OF CALCULATION:
  NLCR; PR(≠results of calculation RPR 290466/06≠);
  OUTPUT VARIABLE(0); FIX;
STEP 1: NLCR; PR(≠s[1] = ≠);
  FIX; OUTPUT(POL(i,5,x,if i = 0 then one else a[i])); ERASE;
  FIX; FIX; d[0]:= S(one,LN(s[1]));
  NLCR; PR(≠f(s[1]) = ≠); OUTPUT(d[0]);
  for i:= 1,2,3,4,5 do d[i]:= DER(d[i-1],s[1]);
STEP 2: for i:= 0,1,2,3,4,5 do d[i]:= SUBSTITUTE(d[i],j,1,1,s[1],one);
STEP 3: fact:= 1; b[0]:= d[0]; b[1]:= d[1];
  for i:= 2,3,4,5 do
  begin fact:= fact × i; b[i]:= Q(d[i],IN(fact)) end;
  ERASE BUT RETAIN(i,0,5,b[i]);
  FIX; g:= POL(i,5,y,b[i]); NLCR; PR(≠g(y) = ≠); OUTPUT(g);
  s[1]:= POL(i,5,x,if i = 0 then one else a[i]);
  comment The easiest way to perform step 4 would be to use the
    statement: s[3]:= SUBSTITUTE(g,i,1,1,y,D(s[1],one)),
    the following statements are, however, more efficient with
    respect to storage space. The first statement serves to erase
    the polynomials g and s[1];
  ERASE;
STEP 4: s[2]:= POL(i,5,x,if i = 0 then zero else a[i]);
  s[4]:= one; s[3]:= b[0];
  for i:= 1,2,3,4,5 do
  begin FIX; FIX; s[4]:= P(s[4],s[2]);
    s[3]:= S(s[3],P(b[i],s[4]));
```

```
      ERASE BUT RETAIN(j,3,4,s[j])
    end;
    NLCR; PR(‡s[3] = ‡); OUTPUT(s[3])
  end; comment the next end corresponds to the begin of the
  general system;
end
```

```
'Input tape for RPR 290466/06'
5          10-10      1
```

The lay-out of the following results is slightly modified by hand, in particular, the unsignificant zeros in numbers are removed. A complex number a + ib is printed as (a, b).

```
results of calculation RPR 290466/06
s[1] = (1) × (x)↑  0  + (a[ 1 ]) × (x)↑  1  + (a[ 2 ]) × (x)↑  2  +
  (a[ 3 ]) × (x)↑  3  + (a[ 4 ]) × (x)↑  4  + (a[ 5 ]) × (x)↑  5
f(s[1]) = 1 + ln(s[1])
g(y) = (1) × (y)↑  0  + (1) × (y)↑  1  + ((-.5 )) × (y)↑  2  +
  ((+.333333333333₁₀-  0 )) × (y)↑  3  + ((-.25 )) × (y)↑  4  +
  ((+.2 )) × (y)↑  5
s[3] = (1) × (x)↑  0  + (a[ 1 ]) × (x)↑  1  +
  ((-.5 ) × (a[ 1 ])↑  2  + a[ 2 ]) × (x)↑  2  +
  ((+.333333333333₁₀-  0 ) × (a[ 1 ])↑  3  + (-1 ) × a[ 2 ] × a[ 1 ] +
    a[ 3 ]) × (x)↑  3  +
  ((-.25 ) × (a[ 1 ])↑  4  + a[ 2 ] × (a[ 1 ])↑  2  + (-1 ) × a[ 3 ] ×
    a[ 1 ] + (-.5 ) × (a[ 2 ])↑  2  + a[ 4 ]) × (x)↑  4  +
  ((+.2 ) × (a[ 1 ])↑  5  + (-1 ) × a[ 2 ] × (a[ 1 ])↑  3  +
    a[ 3 ] × (a[ 1 ])↑  2  + (a[ 2 ])↑  2  × a[ 1 ] + (-1 ) × a[ 4 ] ×
      a[ 1 ] + (-1 ) × a[ 3 ] × a[ 2 ] + a[ 5 ]) × (x)↑  5
```

15. Inputting

In this section an actual program is described, which reads a so-called formula program from input tape.
A user who does not want to build his own system and who does not want to go through the details of the foregoing section can prepare an input tape, the form of which is described below, which activates the actual program to do manipulations with formulae.

A disadvantage of using this way of formula manipulation is that it is not possible to combine the formula manipulation with other capabilities provided by ALGOL 60; calculations with ordinary and complex numbers are, however, possible. This combination is of course very well possible if one makes a special actual program as exemplified in the foregoing section.

In the sequel the meta-linguistic variable identifier will not only be used in the strict sense of the ALGOL 60 report, but also in a more general way, defined as follows:

```
<identifier> ::= <letter>|<digit>|<identifier><letter>|
                 <identifier><digit>|
                 <identifier><lay out symbol>|
                 <identifier>[<unsigned integer>]
```

in which a lay-out symbol, as far as the Mathematical Centre equipment is concerned, may be a space, a tabulator symbol, a new line carriage return symbol and a point.

Example:

```
x
x1
1
f[1]
Test program.
INT POW
```

An algebraic variable and a formula designator, already defined in section 3, will now be redefined by:

<algebraic variable> ::= <identifier>
<formula designator> ::= <identifier>

The input tape to be used for the formula manipulation, should be prepared in such a way that the following numbers and instructions occur on it in the indicated ordering.

1. An integer (≥ 0) defining the maximal degree of the to be used polynomials.

2. A real number defining the absolute accuracy with which the computer has to execute the numerical calculations. (For the X8 Computer of the Mathematical Centre this accuracy can e.g. be set equal to 10^{-10}).

3. An integer number which has to be 1 if the system has to expand the formulae in order to simplify them.
   If this number is not equal to 1, the system will handle the formulae as they are written, except for trivial simplification consisting of calculating numbers. (i.e. Q(CN(1,1), CN(1,-1)) is changed into CN(0,1), or in ordering notation: (1+i)/(1-i) is changed into i).

4. An integral number defining the number of to be used formulae designators and algebraic variables.

5. An integral number defining the maximal number of characters by which the identifiers of the formula designators and algebraic variables are built up (including possible lay-out symbols which do not occur at the very beginning).

6. The following special identifiers, separated by a comma and closed by a semicolon:

   OUTPUT, FIX, ERASE, ER B RET, NLCR, COEFF, END,
   one, zero, S, D, P, Q, POWER, INT POW,
   IN, RN, CN, POL, EXP, LN, SIN, COS, ARCTAN, SQRT, Sum,
   DER, SIMPLIFY, CC, SUBST, QUOTIENT, COMM DIV;

7. The formula manipulation instructions.

The formula should be put on tape in the form as defined in section 3, with the following restriction:
the form of an extended sum, a polynomial and a result of substitutions should be as syntactically defined below:

<formula list> ::= <formula>|<formula list><separator><formula>
<separator> ::= , |;
<extended sum>::= Sum(<integer><separator><formula list>)
<polynomial> ::= POL(<integer><separator><algebraic variable><separator>
<formula list>)
<result of substitution> ::= SUBST(<formula><separator><integer>
<separator><formula list><separator><formula list>)*)

For convenience sake, the symbols S, D, P, Q and POWER may be replaced by the symbols +, -, *, / and ↑; moreover the corresponding brackets may be leaved out.
Thus, S(a,b) may also be put on the tape as +a,b.

Although it was necessary to introduce algebraic variables before they were used, in the actual program as described in section 14, it is no longer necessary here (it is even impossible).
If the actual program encounters a new identifier in a formula then this identifier is automatically recognized to be an algebraic variable or a formula designator.

The form of the formula manipulation instructions is the form of a formula program syntactically defined below:

<formula program> ::= END;|<compound formula statement>; END;

<compound formula statement> ::= <formula statement>|
<compound formula statement>; <formula statement>

<formula statement> ::= FIX|ERASE|NLCR |
<formula assignment statement>|
<calculation of coefficients statement>|

*) The first formula list should be replaced by formula designator list.

<erase but retain statement> |
<output statement>

<formula assignment statement> ::= <formula designator> := <formula>

<calculation of coefficients statement> ::= COEFF(<formula><separator>
<integer><separator><formula list><separator>
<formula designator list>)

<erase but retain statement> ::= ER B RET(<integer><separator>
<formula designator list>)

<output statement> ::= OUTPUT(<formula>)

<formula designator list> ::= <formula designator> | <formula designator
list><separator><formula designator>

The form of a formula program, just defined, should satisfy the following conditions:

1. The formula lists and the formula designator lists should contain a number of respectively formulae and formulae designators, equal to the actual value of the integer occurring as a parameter in the formula program just before the occurrence of the lists; except in a polynomial, where the formula list should contain one extra formula than is defined by the integer.

   Example:
   Sum(5; one; x; INT POW(x,2); INT POW(x,3); INT POW(x,4))
   and POL(3, x, c[0], c[1], c[2], c[3])

2. Each formula statement ERASE should be preceded by a formula statement FIX.
   With the exception of the very first one; in the actual program itself an initializing call for the procedure FIX, equivalent with the formula statement FIX, occurs.
   No reference to this exception is made furtheron.

3. Each erase but retain statement should be preceded by two consecutive corresponding calls for the formula statement FIX.

4. The formula statements ERASE and erase but retain statements should occur in the formula program pairwise with corresponding formula statements FIX.

The effect of the different formula statements are now briefly described (for a more detailed description, the reader is referred to the corresponding sections of this report).

A formula assignment statement: The formula designator becomes a formula.

Example:
f := Sum(5; one; x; INT POW(x,2); INT POW(x,3); INT POW(x,4))

FIX: formulae built up (assigned to formula designators) before a call for FIX are protected against the erasing effect of a call for ERASE, or a call for an erase but retain statement, later on.

ERASE: formulae built up after the last time FIX was called, are erased, moreover the effect of FIX is cancelled.
It is remarked that possibly introduced algebraic variables and formula designators after the last call for FIX, remain in the identifier list, to be discussed further on, but loose their significance*). Introducing these algebraic variables and formula designators again after the ERASE formula statement, leads to new significances.

Example of a compound formula statement:
FIX; f := +a,b; ERASE; FIX; f := -a,b; ERASE

An erase but retain statement: The formulae built up after the last two consecutive calls for FIX are erased, except for the formulae with formula designators occurring in the formula designator list. The effect of the two calls for FIX is cancelled.

Example of a compound formula statement:
FIX; FIX; f := +y,x; g := -a,b; h := *a,b; ER B RET(1,g);
FIX; FIX; f := /a,g; p := ↑b,g; h := EXP(p); ER B RET(1,p)

*) except for those formula designators which refer to formulae which are not erased.

after execution of these statements only a, b, g and p have maintained their significance; x, y, f and h have loosed their significance and they do not refer anymore to formulae (see the note on the preceding page).

A calculation of coefficients statement:
given some formula f as the first parameter of COEFF, given the integer n (> 0) as the second parameter of COEFF, given n formulae $p_1, \ldots, p_n$ in the formula list as the third parameter of COEFF, which do not contain the operators +, -, /, given n formula designators $c_1, \ldots, c_n$ in the formula designator list as the fourth parameter of COEFF, then the effect of this formula statement is:
the formula designators $c_i (i = 1, \ldots, n)$ become equal to formulae which are just the coefficients of the formulae (in general products) p_i in f.

Example:

```
COEFF(S(* * x, a, b, S(* * y, a, b, * z * b * c, EXP(x)));
      2;
      * a, b; * c, EXP(x);
      coeff[1]; coeff[2])
```

has the effect that coeff[1] and coeff[2] become equal to the formulae +x,y and *z,b respectively.

NLCR: a New Line Carriage Return command is given to the printer.

An output statement: The formula occurring as the parameter is printed in ordinary notation.

Example of a formula program:

```
NLCR; OUTPUT(Test );OUTPUT(program );NLCR;
f := Sum(3, one, x, INT POW(x,2));
OUTPUT(f); NLCR; OUTPUT(S(one, x)); NLCR;
OUTPUT(end of );OUTPUT(Test );OUTPUT(program.);END;
```

The printed result is:

```
Test program
1 + x + x ↑ 2
1 + x
end of Test program.
```

Care should be taken for the following points:

1. The lay out symbols: space symbol, tabulator symbol, and nlcr symbol, occurring at the very beginning of an identifier are skipped by the program.
   Within an identifier, they have the significance of an ordinary character. Therefore the spaces in ER B RET should not be forgotten in using an erase but retain statement.

2. Using the output statement for printing some text, as above, it is not allowed to place in this text the following characters: +, -, *, /, =, ∧, ,, :, ;, (,), | (the last character is used to form ↑ by putting first | and then ∧ on input tape).

Next follows the reproduction of the actual program:

```
INITIALIZE;
comment Input program for GENERAL SYSTEM for FORMULA
  MANIPULATION RPR 290466/05;
ACTUAL PROGRAM:
begin integer idp,s,symbol,next symbol, numb of id,length of id;
  numb of id:= read; length of id:= read; if length of id < 8 then
  length of id:= 8;
  begin switch CASE:= OUTP,Fix,Erase,Erase but retain,Nlcr,
    Coeff,END;
    integer array identifier list[1:numb of id + 32,-1:length of id];
    procedure PR(s); string s; PRINTTEXT(s);
    procedure OUTPUT(f); value f; integer f;
    comment at this place the procedure body of the procedure
    OUTPUT declared in the before going section should be
    inserted;
    procedure OUTPUT VARIABLE(f); value f; integer f;
    begin integer i,j; TYPE(f,j,i); if j = 1 then
      begin for j:= 1 step 1 until identifier list[i,0] do
        PRSYM(identifier list[i,j])
    end end OUTPUT VARIABLE;
```

```
integer procedure SYMBOL;
begin symbol:= next symbol; next symbol:= RESYM;
  if symbol = 64 then SYMBOL:= 10 else
  if symbol = 65 then SYMBOL:= 11 else
  if symbol = 66 then SYMBOL:= 12 else
  if symbol = 67 then SYMBOL:= 13 else
  if symbol = 127 ∧ next symbol = 80 then
  begin symbol:= next symbol; next symbol:= RESYM; SYMBOL:= 14
  end else
  if symbol = 98 ∨ symbol = 99 ∨ symbol = 87 ∨ symbol = 91 ∨
    symbol = 90 ∨ symbol = 70 ∨ symbol = 93 ∨ symbol = 118 ∨
    symbol = 119 then SYMBOL:= SYMBOL else
  begin integer i,j,length; integer array identifier[1:length of id];
    identifier[1]:= symbol; i:= length:= 1;
    for i:= i + 1 while
    next symbol ≠ 64 ∧ next symbol ≠ 65 ∧ next symbol ≠ 66 ∧
    next symbol ≠ 67 ∧ next symbol ≠ 127 ∧ next symbol ≠ 98 ∧
    next symbol ≠ 99 ∧ next symbol ≠ 87 ∧ next symbol ≠ 91 ∧
    next symbol ≠ 90 do
    begin symbol:= next symbol; next symbol:= RESYM;
      length:= i; identifier[i]:= symbol
    end; for i:= 1 step 1 until idp do
    begin if identifier list[i,0] ≠ length then goto END;
      for j:= 1 step 1 until length do
      begin if identifier[j] ≠ identifier list[i,j] then goto END end;
      goto SUCCESS;
    END: end; goto NEW IDENTIFIER;
  SUCCESS: SYMBOL:= i; goto OUT;
  NEW IDENTIFIER: idp:= idp + 1; for i:= 1 step 1 until length do
    identifier list[idp,i]:= identifier[i]; identifier list[idp,0]:= length;
    identifier list[idp,-1]:= 0; SYMBOL:= idp;
  OUT: end
end SYMBOL;
```

```
integer procedure Read f;
begin integer p,s,d; switch CASE:= One,Zero,SUm,Difference,
   Product,Quotient,Power,Int pow,Integral number,Real number,
   Complex number,Polynomial,Exp,Ln,Sin,Cos,Arctan,Sqrt,
   SUM,Derivative,Simpl,Complex conjugate,Subst,Quot,Comm div;
   s:= SYMBOL; if s > 32 then goto Identifier else goto CASE[s-7];
One: Read f:= one; goto END;
Zero: Read f:= zero; goto END;
SUm: Read f:= S(Read f,Read f); goto END;
Difference: Read f:= D(Read f,Read f); goto END;
Product: Read f:= P(Read f,Read f); goto END;
Quotient: Read f:= Q(Read f,Read f); goto END;
Power: Read f:= POWER(Read f,Read f); goto END;
Int pow: Read f:= INT POW(Read f,read); goto END;
Derivative: Read f:= DER(Read f,Read f); goto END;
Complex conjugate: Read f:= CC(Read f); goto END;
Polynomial: d:= read;
  begin integer x,i; integer array coeff[0:d];
    x:= Read f; for i:= 0 step 1 until d do
    coeff[i]:= Read f; Read f:= POL(i,d,x,coeff[i])
  end; goto END;
Integral number: Read f:= IN(read); goto END;
Real number: Read f:= RN(read); goto END;
Complex number: Read f:= CN(read,read); goto END;
Exp: Read f:= EXP(Read f); goto END;
Ln: Read f:= LN(Read f); goto END;
Sin: Read f:= SIN(Read f); goto END;
Cos: Read f:= COS(Read f); goto END;
Arctan: Read f:= ARCTAN(Read f); goto END;
Sqrt: Read f:= SQRT(Read f); goto END;
SUM: d:= read; Read f:= Sum(s,1,d,Read f); goto END;
Simpl: Read f:= SIMPLIFY(Read f); goto END;
Subst: s:= Read f; p:= read;
```

```
    begin integer array arg,val[1:p];
      for d:= 1 step 1 until p do arg[d]:= Read f;
      for d:= 1 step 1 until p do val[d]:= Read f;
      Read f:= SUBSTITUTE(s,d,1,p,arg[d],val[d])
    end; goto END;
  Quot: p:= Read f; d:= Read f; s:= SYMBOL;
    Read f:= QUOTIENT(p,d,identifier list[s,-1]);
    goto END;
  Comm div: Read f:= COMMON DIVISOR(Read f,Read f); goto END;
  Identifier: if identifier list[s,-1] ≠ 0 then Read f:=
    identifier list[s,-1] else Read f:= identifier list[s,-1]:=
    STORE(1,algebraic variable,s);
  END: end Read f;

BEGIN OF INPUT PROGRAM:
  idp:= 0; next symbol:= RESYM;
A: if next symbol ≠ 91 then begin SYMBOL; goto A end;
NEXT: s:= SYMBOL; if s > 32 then goto Store else goto CASE[s];
Store: identifier list[s,-1]:= Read f; goto NEXT;
OUTP: OUTPUT(Read f); goto NEXT;
Fix: FIX; goto NEXT;
Erase: for s:= 33 step 1 until idp do
  begin if ¬ FIXED(identifier list[s,-1]) then identifier list[s,-1]:= 0
  end; ERASE; goto NEXT;
Erase but retain: s:= read;
  begin integer i,n; integer array g[1:s];
    n:= s; for i:= 1 step 1 until n do g[i]:= SYMBOL;
    s:= IN(2);
    for i:= 1 step 1 until n do identifier list[g[i],-1]:=
      COPY(identifier list[g[i],-1],FIXED);
    for i:= 33 step 1 until idp do
```

```
        begin if ¬ FIXED(identifier list[i,-1]) ∧ identifier list[i,-1] < s
          then identifier list[i,-1]:= 0
        end; ERASE;
        for i:= 1 step 1 until n do identifier list[g[i],-1]:=
          COPY(identifier list[g[i],-1],FIXED);
        LOWER INDEX
      end; goto NEXT;
    Nlcr: NLCR; goto NEXT;
    Coeff:
      begin integer i,f,n; f:= Read f; n:= read;
        begin integer array p,c[1:n];
          for i:= 1 step 1 until n do p[i]:= Read f;
          COEFF OF PRODUCTS(f,n,p,c);
          for i:= 1 step 1 until n do
          begin s:= SYMBOL; identifier list[s,-1]:= c[i] end
      end end; goto NEXT;
    END: end
  end; comment the next end corresponds to the begin of the
  general system;
end
```

The action of this program is briefly described:

1. The program builds up a list of identifiers by means of the integer array identifier list [1 : numb of id + 32, -1 : length of id]. The special identifiers: OUTPUT, ..., COMM DIV occurring on the input tape just prior to the formula program, are read from input tape in order to occupy the first 32 places in the identifier list.

2. The identifiers are read by means of the integer procedure SYMBOL, which becomes equal to the index s indicating where the identifier is stored in identifier list.

If the identifier was a new one, then identifier list [s,-1] is set equal to 0, in order to give the program the information that a new identifier was just read.
In SYMBOL the non-local integer variables symbol and next symbol are used.
The value of next symbol is the value of the internal representation of the just read character.
The value of symbol is the value of the internal representation of the last but one read character.
The characters are read with the aid of the integer procedure RESYM, which is an MC standard procedure for the X8 (for a detailed description see [11]).
RESYM delivers the value of the internal representation of the read character.
In order to facilitate the reading of the actual program, a table is given of the values of the internal representation of some relevant characters:

64	+	91	;
65	-	93	space symbol
66	*	98	(
67	/	99	)
70	=	118	tabulator symbol
80	∧	119	new line carriage return symbol
87	,	127	|
90	:		

(the character | is used to form the character ↑ by putting consecutively | and ∧ on input tape).

A number is read by the MC standard procedure read.

3. The formulae are read by the integer procedure Read f.
Read f interprets a new identifier as an algebraic variable.
The value of Read f becomes the value of the location of the stored formula in the procedure body of INT REPR (see section 6).

4. The procedure OUTPUT prints a given formula.
   Use is made of the procedure OUTPUT VARIABLE, which prints a string of characters defined by the array elements of identifier list, of an algebraic variable.
   OUTPUT VARIABLE uses the MC standard procedure PRSYM(n), where n is the value of the internal representation of a character, which is printed by PRSYM.

5. The effect of the statements of the actual program, following the label BEGIN OF INPUT PROGRAM, is now elucidated. The pointer of identifier list, the integer variable idp, gets its initial value: 0; next symbol becomes the value of the internal representation of the first read character.
   The special identifiers are read in, until next symbol is the internal representation of ; (= 91).
   Depending on the next read identifier, the different formula statements are executed.

Remark:

1. Inspecting the procedure SYMBOL, it can be seen, that other forms of formula programs may also be excepted by the actual program.
   For example: an input tape containing the following characters:

   f) (; S, a) b : OUTPUT : f = END,

   has the effect of printing a+b.
   Construction of such an input tape is, however, not recommended.

2. Notice that the special identifiers OUTPUT, ..., COMM DIV may be changed into other identifiers, if the corresponding special identifiers occurring in the formula program are also appropriately changed.
   The order of these special identifiers, may, however, not be changed; moreover they may not be built up with more than 8 characters.

3. If the formulae are expanded and simplified, then the ordering of the algebraic variables may be changed; for example, the formula program f := SIMPLIFY(+a,b); OUTPUT(f); END; has the effect of printing b+a.
   The reason for the interchange of a and b is that the algebraic variable a was stored earlier than b, and the value of the internal representation of a is therefore smaller than the corresponding value of b.
   The system, then, orders a and b with respect to decreasing values of their internal representation; and the result is b+a.
   A way to preserve the original ordering is exemplified by the following formula program

```
b := b; f := SIMPLIFY(+a,b); OUTPUT(f); END;
```

   which results in printing a+b.

Next some particular input tapes are reproduced, containing formula programs, followed by their results on execution.
It turned out that about 12000 words were needed, the space for arrays and stacks not included, to store the general system and the actual program in the X8 computer of the Mathematical Centre.

'Input tape 1 with formula program RPR 290466/05
(The procedure read skips the text between the symbols'')
The formula program on this input tape contains the following calculation, described in a rather loose form:
a:= cos(pi/2 - x) - sin(x).
f:= i/(x↑2 + (1 + i) × x + i) + (2/(1 + i))/(x↑2 - 1),
  where i is the imaginary unit.
Real part of f.
Imaginary part of f.

g:= 1/f = x↑2 + (-1 + i) × x - i.

h:= gr comm div(g, x↑2 + 1), thus h = c × (x + i), apart for
the numerical constant c, which turns out to be (-1 + i).

k:= integer quotient of g over h rendering the rest r,
thus, k = 1/c × (x - 1) = (-1/2 - 1/2 i) × (x - 1) and r = 0.

Execution of an erase but retain statement;
it is demonstrated that g and h are retained.

l:= integer quotient of g over x + 1 rendering the rest r,
thus, l = x + (-2 + i) and r = (2 - 2i).

f:= POL(3,x,a,b,c,d).

g:= a + b × x + c × x↑2 + d × x↑3.

By means of the auxiliary formula designator k, the coefficients
c3, c2 and c1 of x↑3, x↑2 and x in g are determined.

h:= f - POL(3,x,zero,b,c,d) = a.

k:= the formula g in which x, a, b, c and d are changed into
y, d, c, b and a.

3 $\uparrow$10 -10 1

40 8

```
OUTPUT,FIX,ERASE,ER B RET,NLCR,COEFF,END,
one,zero,S,D,P,Q,POWER,INT POW,
IN,RN,CN,POL,EXP,LN,SIN,COS,ARCTAN,SQRT,Sum,
DER,SIMPLIFY,CC,SUBST,QUOTIENT,COMM DIV;

NLCR; OUTPUT(tape 1);
1:= one; 2:= IN(2); i:= CN(0,1); pi:= RN(3.14159265359);
x2:= × x,x; FIX;
a:= SIMPLIFY(- COS(- / pi,2, x),SIN(x)); NLCR; OUTPUT(a is );
OUTPUT(a); ERASE;
FIX; FIX; f:= S(Q(i,S(x2,S(P(S(1,i),x),i))),Q(Q(2,S(1,i)),D(x2,1)));
NLCR; OUTPUT(f is ); OUTPUT(f); FIX;
NLCR; OUTPUT(real ); OUTPUT(part );
```

```
OUTPUT(of f is ); OUTPUT(+ f,CC(f));
NLCR; OUTPUT(imag ); OUTPUT(part );
OUTPUT(of f is ); OUTPUT(Q(- f,CC(f),i)); ERASE;
g:= Q(1,f); NLCR; OUTPUT(g is ); OUTPUT(g);
h:= COMM DIV(g,+ x2,1); NLCR; OUTPUT(h is ); OUTPUT(h);
k:= QUOTIENT(g,h,r); NLCR; OUTPUT(k is ); OUTPUT(k);
NLCR; OUTPUT(r is); OUTPUT(r);
ER B RET(2,g,h);
NLCR; NLCR; OUTPUT(g is ); OUTPUT(g);
NLCR; OUTPUT(h is ); OUTPUT(h);
l:= QUOTIENT(g,+ x,1,r);
NLCR; OUTPUT(l is ); OUTPUT(l);
NLCR; OUTPUT(r is ); OUTPUT(r);
a:= IN('this statement is inserted in order to get rid of the
  significance of a (= zero), which would otherwise not be done
  by the ERASE statement' 2); ERASE; FIX; NLCR;
f:= POL(3,x,a,b,c,d); NLCR; OUTPUT(f is ); OUTPUT(f);
g:= SUBST(f;1;x;x); NLCR; OUTPUT(g is ); OUTPUT(g);
COEFF(g;1;× x, × x,x; c3); k:= SIMPLIFY(- g, × c3, × x, × x,x);
NLCR; OUTPUT(k);
COEFF(k,1, × x,x; c2); k:= SIMPLIFY(- k, × c2, × x,x);
NLCR; OUTPUT(k);
COEFF(k,1,x,c1); k:= SIMPLIFY(- k, × c1,x);
NLCR; OUTPUT(k);
h:= - f,POL(3,x,zero,c1,c2,c3);
NLCR; OUTPUT(h is ); OUTPUT(h);
k:= SUBST(g;5;x;a;b;c;d; y;d;c;b;a);
NLCR; OUTPUT(k is ); OUTPUT(k);
END;
```

The lay-out of the following results is slightly modified by hand, in particular, the unsignificant zeros in numbers are removed. A complex number a + ib is printed as (a, b).

```
tape 1
a is  0
f is ( 1 )/((x)↑  2  + (-.1₁₀+  1 ,+.1₁₀+  1 ) × x + (-.0 ,-.1₁₀+  1 ))
real part of f is ((-.0 ,-.1₁₀+  1 ) × x)/((-.0 ,-.5 ) × (x)↑  3  +
  (-.0 ,+.5 ) × (x)↑  2  + (-.0 ,-.5 ) × x + (+.0 ,+.5 ))
imag part of f is ((-.0 ,+.1₁₀+  1 ))/((-.0 ,-.5 ) × (x)↑  3  +
  (-.0 ,+.5 ) × (x)↑  2  + (-.0 ,-.5 ) × x + (+.0 ,+.5 ))
g is (x)↑  2  + (-.1₁₀+  1 ,+.1₁₀+  1 ) × x + (-.0 ,-.1₁₀+  1 )
h is (-.1₁₀+  1 ,+.1₁₀+  1 ) × x + (-.1₁₀+  1 ,-.1₁₀+  1 )
k is (-.5 ,-.5 ) × x + (+.5 ,+.5 )
r is 0

g is (x)↑  2  + (-.1₁₀+  1 ,+.1₁₀+  1 ) × x + (-.0 ,-.1₁₀+  1 )
h is (-.1₁₀+  1 ,+.1₁₀+  1 ) × x + (-.1₁₀+  1 ,-.1₁₀+  1 )
l is x + (-.2₁₀+  1 ,+.1₁₀+  1 )
r is (+.2₁₀+  1 ,-.2₁₀+  1 )

f is ( a ) × (x)↑  0  + (b) × (x)↑  1  + (c) × (x)↑  2  + (d) × (x)↑  3
g is a + b × x + c × (x)↑  2  + d × (x)↑  3
c × (x)↑  2  + b × x + a
b × x + a
a
h is a
k is d + c × y + b × (y)↑  2 + a × (y)↑  3
```

```
'Input tape 2 with formula program RPR 290466/05
The following calculations originated from the Handbook for
Mathematical Functions [12] page 72.
The calculation is first performed without simplifying,
i.e. expand = false.
In the next formula program expand = true'
0        10-10     0
15       8
OUTPUT,FIX,ERASE,ER B RET,NLCR,COEFF,END,
one,zero,S,D,P,Q,POWER,INT POW,
IN,RN,CN,POL,EXP,LN,SIN,COS,ARCTAN,SQRT,Sum,
DER,SIMPLIFY,CC,SUBST,QUOTIENT,COMM DIV;

NLCR; OUTPUT(tape 2);
3:= IN(3); 4:= IN(4); 8:= IN(8); FIX;
f:= SIMPLIFY(- SIN(× 3,z), - × 3,SIN(z),
  × 4, INT POW(SIN(z),3));
NLCR; OUTPUT(formula ); OUTPUT(4.3.27 );
OUTPUT(f); ERASE; FIX;
f:= SIMPLIFY(+ COS(× 3,z), - × 3,COS(z), × 4,INT POW(COS(z),3));
NLCR; OUTPUT(formula ); OUTPUT(4.3.28 );
OUTPUT(f); ERASE; FIX;
f:= SIMPLIFY(- SIN(× 4,z), - × 8, × INT POW(COS(z),3),SIN(z),
  × 4, × COS(z),SIN(z));
NLCR; OUTPUT(formula ); OUTPUT(4.3.29 );
OUTPUT(f); ERASE; FIX;
f:= SIMPLIFY(- COS( × 4,z), × 8, - INT POW(COS(z),4),
   INT POW(COS(z),2));
NLCR; OUTPUT(formula ); OUTPUT(4.3.30 );
OUTPUT(f);
END;
```

```
(tape 2)
(formula )(4.3.27 )((sin((((+3 )) × (z)))) - ((((+3 )) × (sin((z))))
  - (((+4 )) × (((sin((z))))↑ 3 ))))
(formula )(4.3.28 )((cos((((+3 )) × (z)))) + ((((+3 )) × (cos((z))))
  - (((+4 )) × (((cos((z))))↑ 3 ))))
(formula )(4.3.29 )((sin((((+4 )) × (z)))) - ((((+8 )) × ((((cos((z))))↑ 3 )
  × (sin((z))))) - (((+4 )) × ((cos((z))) × (sin((z)))))))
(formula )(4.3.30 )((cos((((+4 )) × (z)))) - (((+8 )) × ((((cos((z))))↑ 4 )
  - (((cos((z))))↑ 2 ))))

'Input tape 3 with formula program RPR 290466/05'
0        10-10     1
15       8
OUTPUT,FIX,ERASE,ER B RET,NLCR,COEFF,END,
one,zero,S,D,P,Q,POWER,INT POW,
IN,RN,CN,POL,EXP,LN,SIN,COS,ARCTAN,SQRT,Sum,
DER,SIMPLIFY,CC,SUBST,QUOTIENT,COMM DIV;

NLCR; OUTPUT(tape 3);
The rest of this formula program is identical to
the above formula program, but for the first two
formula statements.

tape 3
 formula 4.3.27  0
 formula 4.3.28  0
formula 4.3.29  0
formula 4.3.30  1
```

References

[1] J.W. Backus et al., Revised report on the algorithmic language ALGOL 60, edited by P. Naur. Regnecentralen, Copenhagen, 1962.

[2] J.E. Sammet, Formula Manipulation by Computer, to appear in Advances in Computers, Volume 8, McGraw Hill, Preliminary version as Technical Report TR00.1363, IBM Systems Development Division, Poughkeepsie, N.Y., November 1965.

[3] J.E. Sammet, Survey of the Use of Computers for Doing Non-Numerical Mathematics, Technical Report TR00.1428, IBM Systems Development Division, Poughkeepsie, N.Y., March 1966. A shortened version is published as: Survey of Formula Manipulation, Comm. of the ACM 9 (1966), 8, 555-569.

[4] J.E. Sammet, An Annotated Description Based Bibliography on the Use of Computers for Non-Numerical Mathematics, Technical Report TR00.1427, IBM Systems Development Division, Poughkeepsie, N.Y., March 1966.

[5] E.R. Bond, History, Features and Commentary on FORMAC, Technical Report TR00.1426, IBM Systems Development Division, Poughkeepsie, N.Y., March 1966.

[6] A.J. Perlis, R. Itturiaga, T.A. Standish, A Definition of Formula ALGOL, Carnegie Institute of Technology, Pittsburgh, P.A., March 1966.

[7] R.P. van de Riet, Algebraic Operations in ALGOL 60. A second order problem,
Report TW 96, Mathematical Centre, Amsterdam, March 1965.

[8] R.P. van de Riet, Algebraic Operations in ALGOL 60. The Cauchy Problem I,
Report TW 97, Mathematical Centre, Amsterdam, December 1965.

[9] S.J. Bijlsma, Algebraic Operations in ALGOL 60. The Saddlepoint method,
Report TN 45, Mathematical Centre, Amsterdam, May 1966.

[10] R.P. van de Riet, An application of a method for algebraic manipulation in ALGOL 60,
Report TW 99, Mathematical Centre, Amsterdam, January 1966.

[11] F.E.J. Kruseman Aretz, Het MC-ALGOL 60-systeem voor de X8. Voorlopige programmeurshandleiding.
Report MR 81, Mathematical Centre, Amsterdam, June 1966.

[12] M. Abramowitz, I.A. Stegun, (editors), Handbook of Mathematical Functions, U.S. Department of Commerce,
National Bureau of Standards,
Applied Mathematics Series 55, Third printing,
March 1965.