RA

stichting
mathematisch
centrum

MC

REKENAFDELING MR 136/72 AUGUST

G. TEN VELDEN

SIMPLIFICATION PROCEDURES FOR ABC ALGOL

RA

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O), by the Municipality of Amsterdam, by the University of Amsterdam, by the Free University at Amsterdam, and by industries.

Acknowledgement

The author is grateful to R.P. van de Riet for his stimulating remarks and his critically reading of the preliminary text. This led to several improvements concerning the clearness of the ALGOL 60 program, and to a more exact definition of the notions, used in the accompanying comments.

Table of contents.

begin comment

Simplification procedures for ABC ALGOL.

Introduction.

Simplification procedures for formula manipulation have been described in section 2.1 of [1] for a system without automatic garbage collection. There, a second internal representation of a formula has been introduced, using two auxiliary array's 'a' and 'L'.

The purpose of this report is to describe simplification procedures for the garbage collection formula manipulation system with a free list technique, to be used as a basis for the new programming language called 'ABC ALGOL' ('ABC' standing for: 'Algebraische Bewerkingen met behulp van de Computer', dutch for: 'Algebraic Operations by means of the Computer').

For storing formulae, this system, described in [2], makes use of one array 'C' (replaced in this report by the linear arrays C1, C2 and Ctype) only, which is under the supervision of the garbage collection process.

For considerations concerning the need and realisation of automatic garbage collection it suffices to refer to [2].

The special purpose of this report is to perform the simplification of arbitrary formulae, using the storage space for formulae efficiently, possibly at the cost of time efficiency. We want to avoid the situation that two different representations of the same formula have been stored at the same time, one before and one after the simplification, because we are interested in the simplified formula only.

It will become evident that we have to follow a simplification strategy, quite different from the strategy of [1], but we want the result to be the same, i.e. the form of the resulting formula has to be a standard form, satisfying several conditions. In chapter 2 the standard form 'simplified sum' is described, satisfying equivalent conditions as described in section 2 of [1]. (Exponential functions are not considered.)

In chapter 3 another standard form (called 'polynomial') is introduced, illustrating the suitability of these simplification procedures in a system like [3].

Chapter 1 Recapitulation of the garbage collection system.

In this chapter the system of [2], using the free list technique, is described, extended with the two formula types 'integral number' and 'integral power', and changed at several places, to be compared with the system of [3]. Next follows a brief description of those procedures of the system, used in the simplification process, which have to be known by the uninitiated reader, who is not interested in the particular garbage collection process. For understanding of the simplification process, described in the chapters 2 and 3, he may confine himself to this description.

Garbage and non - garbage formulae.

Since the garbage collector has to determine which formulae may be considered as 'garbage', for 'non-garbage' formulae a linked list of so-called names is introduced, growing and shrinking according to a stack mechanism, which is consulted by the garbage collector whenever garbage has to be collected. This stack mechanism involves a block structure and a scope of a name 'f', which do not have to coincide with the block structure of the -ALGOL 60- program and the scope of the -ALGOL 60- variable 'f'.

If the formula manipulator wants to know how much free space is available, i.e. the space containing the free list and the garbage formulae, he can call the auxiliary procedure 'free space', delivering the number of available storage cells for formulae.

The procedures DE and ERASE.

Introduction and erasure of names is organised in a block structure. The beginning of each block contains the declaration of an integer 'fnn'. The first statement of the block is: fnn:=gnn, where 'gnn' serves as a formula stack pointer, indicating the number of names introduced. The final statement of the block is: ERASE(fnn), which erases all names introduced in the block.

The introduction of a name 'f' is performed through the statement: DE(f,F,next), where 'F' is the initial value of 'f' and 'next' is either 0 (ZERO) or DE(f1,F1,next1), for introducing another name 'f1' with initial value 'F1', etc.

The procedures ASSIGN and V.

Given a name f, we can (only within the scope of f of course):

1. make f the name of the formula F (a value), performed by procedure ASSIGN: ASSIGN(f,F),
2. ask for the formula F (the value), whose name is f, performed by the procedure V: V(f), delivering F.

The procedures INT NUM, AV, S, P and INT POW.

To construct formulae of the types 'integral number', 'algebraic variable', 'sum', 'product' or 'integral power', we have to write the formulae in Polish prefix, i.e. in functional notation. Here we use the function designators INT NUM, AV, S, P and INT POW, respectively, for those procedures which have to construct the internal representation of the formulae of the above mentioned types.

As the evaluation of an argument of one of these functions could give rise to a garbage collection, this argument is interpreted by the garbage collector as non-garbage, and the arguments which are evaluated already are connected temporarily to the name list. Evaluating a formula expression, this formula is automatically saved during garbage collection.

The procedures LHS, RHS and TYPE.

Calling these procedures with a value as (first) parameter, the components of the formula, i.e. the quantities left-hand-side and right-hand-side as well as its type are retrieved.(For the internal representation of formulae we refer to section 1 of chapter 2).

A call TYPE(F,A,B), moreover, delivers the left and right component of the formula F, assigned to the parameters A and B respectively.

The output procedures.

The output is performed by a line-printer and a tape punch together. The basic procedures are:
PR string(s) for printing and punching a string 's',
PR nlcr for printing and punching a 'new line carriage return symbol',
PR int num(a) for printing and punching an integral number 'a',
PR sym(a) for printing and punching a symbol 'a';
(N.B. for the internal representation of symbols we refer to [4]),
The main output procedure is procedure 'OUTPUT' for printing and punching a formula. The call OUTPUT(f) causes the printing and punching of the value of the name f, without superfluous parentheses.

```
          program:;

integer free cell, last free cell, last name, max of C, snn, gnn,
integral number, algebraic variable, sum, product, integral power,
one, zero, ONE, ZERO;

max of C:= read;

begin integer array C1, C2, Ctype[1:max of C];
  Boolean array traced[1:max of C];

  integer procedure STORE(A,t,B); value A,t,B; integer A,t,B;
  begin integer k; k:=C1[free cell];
    C1[free cell]:=A; C2[free cell]:=B; Ctype[free cell]:=t;
    STORE:=free cell;
    free cell:=if free cell ≠ last free cell then k else
    COLLECT GARBAGE(free cell)
  end STORE;

  integer procedure TYPE(F,A,B); value F; integer F,A,B;
  begin TYPE:=if F > 0 then Ctype[F] else ERROR(true,⟨error in TYPE⟩);
    A:=C1[F]; B:=C2[F]
  end TYPE;

  integer procedure LHS(F); value F; integer F; LHS:=C1[F];

  integer procedure RHS(F); value F; integer F; RHS:=C2[F];

  integer procedure SAVE(F); value F; integer F;
  begin integer k; ERROR(F < 0,⟨error in SAVE⟩);
    k:=C1[free cell]; C1[free cell]:=F; C2[free cell]:=last name;
    SAVE:=last name:=free cell; gnn:=gnn + 1;
    free cell:=if free cell ≠ last free cell then k else
    COLLECT GARBAGE(0)
  end SAVE;

  integer procedure DE(f,F,next); integer f,F,next;
  begin f:= - SAVE(F); DE:=next end;

  procedure ERASE(n); value n; integer n;
  for n:=n while n < gnn do
  begin join to free space(last name); gnn:=gnn - 1;
    last name:=C2[last name]; ERROR(gnn < snn,⟨error in ERASE⟩)
  end ERASE;

  procedure join to free space(k); value k; integer k;
  C1[last free cell]:=last free cell:=k;
```

```
integer procedure ASSIGN(f,F); value f,F; integer f,F;
ASSIGN:=C1[- f]:=if f < 0 ∧ f ≥ - max of C then F else
ERROR(true,⟨error in ASSIGN⟩);

integer procedure V(f); value f; integer f;
V:=if f < 0 then C1[- f] else ERROR(true,⟨error in V⟩);

integer procedure COLLECT GARBAGE(st); value st; integer st;
begin integer i; last free cell:=0;
  if st ≠ 0 then TRACE(st);
  i:=last name; for i:=i while i ≠ 0 do
  begin TRACE(C1[i]); traced[i]:=true; i:=C2[i] end;
  for i:=1 step 1 until max of C do
  if ¬ traced[i] then
  begin if last free cell ≠ 0 then join to free space(i) else
    COLLECT GARBAGE:=last free cell:=i
  end else traced[i]:=false;
  ERROR(last free cell = 0,⟨no space left⟩)
end COLLECT GARBAGE;

procedure TRACE(F); value F; integer F;
if ¬ traced[F] then
begin integer t,A,B; t:=TYPE(F,A,B);
  if t = sum ∨ t = product then
  begin TRACE(A); TRACE(B) end else
  if t = integral power then TRACE(B);
  traced[F]:=true
end TRACE;

integer procedure free space;
begin integer fc,n;
  for free cell:=free cell while free cell ≠ last free cell do AV(0,100);
  comment The next statement causes a garbage-collection;
  AV(0,100); n:=1; fc:=free cell;
  for fc:=C1[fc] while fc ≠ last free cell do n:=n + 1;
  free space:=n + 1
end free space;

integer procedure INT NUM(i); value i; integer i;
INT NUM:=if i = 0 then ZERO else if i = 1 then ONE else
         STORE(0, integral number, i);

integer procedure AV(l,r); value l,r; integer l,r;
AV:=STORE(l, algebraic variable, r);
```

```
integer procedure S(A,B); integer A,B;
begin integer A1,B1,fnn; fnn:=gnn;
  A1:=A; SAVE(A1); B1:=B; ERASE(fnn);
  S:=if A1 = ZERO then B1 else if B1 = ZERO then A1 else
     STORE(A1, sum, B1)
end S;

integer procedure P(A,B); integer A,B;
begin integer A1,B1,fnn; fnn:=gnn;
  A1:=A; SAVE(A1); B1:=B; ERASE(fnn);
  P:=if A1 = ZERO ∨ B1 = ZERO then ZERO else
     if A1 = ONE then B1 else if B1 = ONE then A1 else
     STORE(A1, product, B1)
end P;

integer procedure INT POW(B,exp); value B,exp; integer B,exp;
INT POW:=if B = ZERO then ZERO else if exp = 1 then B else
         if B = ONE ∨ exp = 0 then ONE else
         STORE(exp, integral power, B);

procedure INITIALIZE;
begin integer i; free cell:=1; last free cell:=max of C;
  for i:=free cell step 1 until last free cell do
  begin C1[i]:=i + 1; traced[i]:=false end;
  last name:=0;
  integral number:=1; algebraic variable:=2;
  sum:=3; product:=4; integral power:=5;
  DE(zero,STORE(0,integral number,0),
  DE(one,STORE(0,integral number,1),0));
  ZERO:=V(zero); ONE:=V(one); snn:=gnn
end INITIALIZE;

integer procedure ERROR(b,s); Boolean b; string s;
if b then
begin ERROR:=1; PR nlcr; PR string(s); PR nlcr;
  for last name:=last name while last name ≠ 0 do
  begin PR nlcr; OUTPUT( - last name); last name:=C2[last name] end;
  EXIT
end ERROR;
```

```
procedure OUTPUT(f); value f; integer f;
begin procedure OP(F,type); value F,type; integer F,type;
  begin integer t,A,B;
    procedure LBR; if t < type then PR string(‹(›);
    procedure RBR; if t < type then PR string(‹)›);
    t:=TYPE(F,A,B);
    if t = integral number then
    begin type:=if B < 0 then t + 1 else t; LBR; PR int num(B); RBR end
    else if t = algebraic variable then
    begin integer l, d; l:=A : 10; d:=A - l×10; PR sym(l);
      if d > 0 then PRsym(d)
    end else
    if t = integral power then
    begin OP(B,t); PR string(‹↑›); PR int num(A) end else
    begin LBR; OP(A,t); if t = sum then PR string(‹+›) else
      if t = product then PR string(‹×›) else
      begin PR string(‹error in OUTPUT›); EXIT end;
      OP(B,t); RBR
    end
  end OP;
  OP(V(f),0)
end OUTPUT;

procedure PR string(s); string s;
begin PRINTTEXT(s); PUTEXT(s) end;
procedure PR nlcr; PR string(‹
›);
procedure PR int num(a); value a; integer a;
begin integer b; if a < 0 then begin PR string(‹-›); a:=-a end;
  if a < 9 then PR sym(a) else
  begin b:=a : 10; a:=a - b × 10; PR int num(b); PR sym(a) end
end PR int num;
procedure PR sym(a); value a; integer a;
begin PRSYM(a); PUSYM(a) end;
```

```
comment

Chapter 2       The simplification procedure 'SIMPLIFY'.

algorithm:;

integer procedure SIMPLIFY(formula); value formula; integer formula;
begin comment
```

1. definitions.

An important part of the simplification process consists of changing the tree, which represents the formula to be simplified. To indicate these different tree structures, we shall use syntactic rules with a special notation: (<lhs>,<type>,<rhs>), indicating the internal representation of a formula.

The correspondence between a mathematical formula and its internal representation needs some attention. The left-hand-side and right-hand-side quantities of the dyadic types 'sum' and 'product' denote the left and right operand of the operator of that type. In case of an 'integral power', the base of the integral power is stored as right-hand-side quantity and the integral exponent is stored as left-hand-side quantity (see def. 1.3).

A 'number' and an 'algebraic variable' are stored with their necessary information (i.e. the numerical value of a number or the hierarchy number of an algebraic variable as right-hand-side quantity, and the output code of an algebraic variable as left-hand-side quantity, see also def. 1.11 and def. 1.12).

1.1 <formula>::=<term>|(<formula>,<sum indication>,<formula>)
1.2 <term>::=<factor>|(<formula>,<product indication>,<formula>)
1.3 <factor>::=<basic factor>|(<exponent>,<int pow indication>,<formula>)
1.4 <basic factor>::=<integral number>|<algebraic variable>

1.5 <prepared sum>::=<term>|(<term>,<sum indication>,<formula>)
1.6 <prepared product>::=<factor>|
(<factor>,<product indication>,<formula>)

1.7 <stratified sum>::=<simplified product>|
(<simplified product>,<sum indication>,<stratified sum>)
1.8 <stratified product>::=<proper factor>|
(<proper factor>,<product indication>,<stratified product>)

1.9 <proper term>::=<proper factor>|
(<proper term>,<product indication>,<proper term>)
1.10 <proper factor>::=<basic factor>|
(<exponent>,<int pow indication>,<basic factor>)

1.11 <integral number>::=(0,<int num indication>,<integer>)
1.12 <algebraic variable>::=(<output code>,<alg var indication>,
<hierarchy number>)

1.13 <hierarchy number>::=<integer> ('positive')
1.14 <output code>::=<integer> ('internal representation of symbols')
1.15 <exponent>::=<integer> ('non-negative')

1.16 <int num indication>::=1
1.17 <alg var indication>::=2
1.18 <sum indication>::=3
1.19 <product indication>::=4
1.20 <int pow indication>::=5

The definitions of 'simplified sum' and 'simplified product' are given in the sections 5 and 3.2, respectively.

2. Preparation.

When we have to simplify a formula, which is generally a sum of terms, we are interested at first in the first term of that formula. Moreover, we want to know the sum of the remaining terms explicitly. For that reason we change the formula into the sum of its first term and the remaining terms. Mathematically this means application of the associative law for the adding operator. For instance:

(x1 + y1) + (x2 + y2) = x1 + (y1 + (x2 + y2))

The latter formula is called a 'prepared sum' (def. 1.5).
Besides this associative law also an associative law is available for the multiplying operator. Application of the latter changes a formula into a prepared product (def. 1.6).
'Preparation' can be applied to formulae with an arbitrary tree structure (see def. 1.1).

```
algorithm:;

procedure prepare(type,formula);value type,formula; integer type,formula;
begin integer A,B,LA,RA;
  if TYPE(V(formula),A,B) ≠ type then goto OUT;
  if TYPE(A,LA,RA) ≠ type then goto OUT;
  if type = sum then ASSIGN(formula,S(LA,S(RA,B))) else
  if type = product then ASSIGN(formula,P(LA,P(RA,B))) else
  ERROR(true,{type not appropriate in prepare});
  prepare(type,formula);
OUT:
end prepare;
```

comment

3. Simplification of a term.

The simplification of a term, named 'term', is performed in two stages, namely a so-called 'stratification' followed by a 'proper simplification', executed by the procedures 'stratify product' and 'simplify stratified product', respectively.

During the stratification all numerical factors (factors of type 'integral number') of the term are retained in a global integer 'coefficient'. So a stratified product (def. 1.8) actually has no factors which are integral numbers except when the term is an integral number itself, in which case the stratified product will become ONE.

At the end of the simplification of the stratified term (i.e. simplification of the non-numerical part of the term) this coefficient is connected again to the term, which completes the simplification of the term.

For the stratification as well as the proper simplification we need some auxiliary names for saving formulae which are still of interest.
In the following procedures they have not been declared as names, because they are used also in other parts of the program. The declaration of all auxiliary names can be found in section 7.

The auxiliary names we use in this section are called 'handles'.

algorithm:;

integer handle one,handle two,term,coefficient;

comment

3.1. Stratification of a term.

During the stratification we determine by means of a global integer 'next wanted' how we have to start the proper simplification process. 'next wanted' indicates for which algebraic variable we want to look at first, i.e. the algebraic variable with the lowest hierarchy (see section 4).

The stratification is performed by procedure 'stratify product', which makes use of one auxiliary name 'handle one' only.

After the stratification this 'handle one' has as value the product, just stratified, while 'term' has as value 1 (ONE).

```
algorithm:;

integer next wanted;

procedure stratify product;
begin integer t,A,B; prepare(product,term);
  next wanted:=0; coefficient:=1;
L:for t:=TYPE(V(term),A,B) while t = product do
  take factor(A)in handle one and assign to term the prepared rest:(B);
  comment Due to the construction with a for statement we have to treat
  the last factor explicitely;
  take factor(V(term))in handle one and assign to term:(ONE);
  comment We have to check now whether new factors have been appeared
  as a possible side-effect of procedure 'stratify product', which
  effect is discussed below;
  if V(term) ≠ ONE then goto L
end stratify product;

comment
```

If the term to be stratified is a proper term (def. 1.9), its factors are proper factors (def. 1.10), and when we have taken these proper factors consecutively in handle one, the latter will be a stratified product, which is our purpose.

But if the term to be stratified is not a proper term, i.e. if one of the factors is not a proper factor, one of the following situations occurs:

1: If the considered factor is a sum of terms, the term we are simplifying can be written as two other terms by applying the distributive law.

2: If the factor is an integral power of a sum or of a product or of an other integral power, the factor can be written as two or more factors.

In the first case we have to get rid of one of the two new terms somehow, because we can stratify actually one of them only.

At this point we have to know something about the procedure 'simplify sum' of section 5, which calls procedure 'stratify product'.

In 'simplify sum' the 'formula' to be simplified is modified at first into a prepared sum (def. 1.7), in this case into the sum of the term we are simplifying now, assigned to 'term', and remaining terms. Henceforward the value of 'formula' consists of these remaining terms only.

The solution of how to get rid of one of the two terms, appearing after application of the distributive law, without loosing it during a possible garbage collection will be clear now:

We simply add this term to the value of 'formula' (i.e. We assign to 'formula' the sum of this term and the original value of 'formula').

These actions are performed by procedure 'take factor'.

We give some examples of the possible modifications performed by 'take factor':

1. $x \times (y + z)$ becomes $x \times y + x \times z$, where $x \times z$ will be added to the value of 'formula', and $x \times y$ becomes the value of 'term',
2. $(x + y)\uparrow 5$ becomes $(x + y) \times ((x + y) \times (x + y))\uparrow 2$, where the second factor will be modified later according to example 3,
3. $(x \times y)\uparrow 3$ becomes $x\uparrow 3 \times y\uparrow 3$,
4. $((x + y)\uparrow 3)\uparrow 2$ becomes $(x + y)\uparrow 6$.

Only if the first actual parameter of 'take factor' is a proper factor we shall actually take it in handle one (by means of a jump to label 'L' in procedure 'take factor') or in 'coefficient' when the proper factor is an integral number, otherwise we only modify the considered term in a suitable way without taking any factor in handle one (by means of a jump to label 'OUT' in procedure 'take factor').

```
algorithm:;
procedure take factor(F)in handle one and assign to term
the prepared rest:(R); value F,R; integer F,R;
begin integer t,A,B; t:=TYPE(F,A,B);
  if t = integral number then coefficient:=coefficient × B else
  if t = algebraic variable then goto L else
  if t = sum then
  begin ASSIGN(formula,
        S(P(INT NUM(coefficient),P(V(handle one),P(B,R))),V(formula)));
    R:=P(A,R)
  end else
  if t = integral power then
  begin integer t1,LB,RB; t1:=TYPE(B,LB,RB);
    if t1 = integral number then coefficient:=coefficient × RB ↑ A else
    if t1 = algebraic variable then goto L else
    if t1 = sum then
    R:=P(if A:2×2=A then ONE else B,P(INT POW(P(B,B),A:2),R)) else
    if t1 = product then R:=P(INT POW(LB,A),P(INT POW(RB,A),R)) else
    if t1 = integral power then R:=P(INT POW(RB,LB × A),R)
  end else ERROR(t = product,⟨F not appropriate in 'take factor'⟩);
  goto OUT;
L:ASSIGN(handle one,P(F,V(handle one))); det next variable(F);
OUT:ASSIGN(term,R); prepare(product,term)
end take factor;
```

comment

3.2. Simplification of a stratified product.

Definition:
A 'simplified product' is a stratified product satisfying the following conditions:
1: Only the first factor may be a numerical factor i.e. an 'integral number',
2: Each two other proper factors have a different hierarchy,
3: The order of the proper facors is unique, prescribed by the hierarchy of the algebraic variables (see section 4).

For the proper simplification process, performed by procedure 'simplify stratified product', we need both handles:
'handle one' initially contains the stratified product we want to simplify, and 'handle two' is used to save factors which we decided to join not yet to the value of 'term'. We noticed already that the initial value of 'term' (i.e. the final value of 'term' in 'stratify product') is 1 (ONE).
(Remark: the value of 'term' is a simplified product during the whole process).
The algorithm is as follows:
1: wanted:=next wanted
2: Treat consecutively all proper factors of the stratified product (i.e. join the factor to the value of 'term' or take it in 'handle two' dependent on whether we are looking for the algebraic variable of that proper factor (see section 4) or not) and choose at the same time the 'next wanted' variable (i.e. the algebraic variable with the lowest hierarchy) from the proper factors which are taken in 'handle two'. The determination which algebraic variable will be treated next, is performed by procedure 'det next variable' of section 4.
3: assign to 'handle one' the value of 'handle two'
4: if the value of 'handle one' is unequal to ONE then goto 1 else
5: We are ready.

Remark:

'term' is not directly accessible for each proper factor we want to join to it, but only through a kind of buffer consisting of the integers 'exponent' and 'BASE', which combines the proper factors to be joined to the value of 'term', to one proper factor, which will be joined actually to the value of 'term', after all factors of the stratified product have been treated (between step 3 and 4).
For example: $x\uparrow 3 \times x \times x\uparrow 2$ will be combined to $x\uparrow 6$.
The result is that the value of 'term' remains a stratified product, mathematically equivalent to the original stratified product, but in the first one each two proper factors have a different hierarchy and the order of the factors is prescribed uniquely by the hierarchy of the algebraic variables, determined in procedure 'det next variable'.

```
algorithm:;

integer wanted,exponent,BASE;

procedure simplify stratified product;
begin integer t,A,B;
L:ASSIGN(handle two,ONE);wanted:=next wanted; next wanted:=0;
  exponent:=0; BASE:=ONE;
  for t:=TYPE(V(handle one),A,B) while t = product do
  treat proper factor(A)and assign to handle one:(B);
  treat proper factor(V(handle one))assign to handle one:(V(handle two));
  ASSIGN(term,P(INT POW(BASE,exponent),V(term)));
  if V(handle one) ≠ ONE then goto L;
  ASSIGN(term,P(INT NUM(coefficient),V(term)))
end simplify stratified product;

procedure treat proper factor(F)and assign to handle one:(R);
value F; integer F,R;
begin integer t,A,B; t:=TYPE(F,A,B);
  if t = algebraic variable then
  begin if B = wanted then buffer for term(1,F) else
    take in handle two(F)
  end else
  if t = integral power then
  begin if RHS(B) = wanted then buffer for term(A,B) else
    take in handle two(F)
  end else ERROR( F ≠ ONE,⟨F not appropriate in treat proper factor⟩);
  ASSIGN(handle one,R)
end treat proper factor;

procedure buffer for term(exp,B);value exp,B;integer exp,B;
begin exponent:=exponent + exp; BASE:=B end;

procedure take in handle two(F);value F;integer F;
begin det next variable(F); ASSIGN(handle two,P(F,V(handle two))) end;

comment
```

4. determination of order of proper factors.

As a proper factor of a stratified product actually cannot be an integral number (integral numbers are retained in 'coefficient') this proper factor is either an algebraic variable or an integral power of an algebraic variable (see def. 1.10), and for the determination of order of proper factors we use the integral number which is stored as right-hand-side quantity 'r' of this algebraic variable. The algebraic variables are ordered according to increasing value of 'r'.

In future this might be changed. Perhaps we want to have an alphabetic ordering or we want the user of the system to specify the hierarchy of the algebraic variables himself.

```
algorithm:;

procedure det next variable(F);value F;integer F;
begin integer t,A,B;
    integer procdure max(a,b);value a,b; integer a,b;
    max:= if a < b then b else a;
  t:=TYPE(F,A,B);
  next wanted:= if t = algebraic variable then max(B,next wanted) else
                if t = integral power then max(RHS(B),next wanted) else
                ERROR(true,<F not appropriate in det next variable>)
end det next variable;

comment
```

5. simplification of a sum of terms.

Definition:
A 'simplified sum' is a stratified sum satisfying the following conditions:

1: Each two terms (which are simplified products) are different in their non-numerical parts,
2: The order of these terms is unique.

Actually this order will be determined by procedure 'put before'. At this point we shall not specify this order because we want to have the possibility to change the ordering of the terms by changing the procedure 'put before'.

Application of the same strategy as used in section 3, namely a proper simplification preceded by a stratification, will be much more complicated here because of the complex structure of a simplified product (in comparison with a proper factor in section 3). This complex structure implies a rather difficult determination of order for the simplified products (performed by procedure 'put before').

For the simplification of a sum of terms we shall use two auxiliary names, namely 'simplified part' and 'handle two'.

The strategy is as follows:

1: prepare the considered sum (named 'formula'),
2: assign the first term of this prepared 'formula' to 'term' and assign the remaining terms to 'formula',
3: simplify the first term by means of the procedures 'stratify product' and 'simplify stratified product' of section 3.
4: join the product, just simplified and named 'term', to the 'simplified part' at such a place that the order of the terms remains the correct order (prescribed in procedure 'put before').
5: if the value of 'formula' is unequal to ZERO then goto 1 else
6: we are ready.

```
algorithm:;

integer simplified part;

procedure simplify sum;
begin integer t,FIRST TERM,REST;
L:prepare(sum,formula);
  for t:=TYPE(V(formula),FIRST TERM,REST) while t = sum do
  begin ASSIGN(term,FIRST TERM); ASSIGN(formula,REST);
    stratify product; simplify stratified product;
    join term to simplified part; prepare(sum,formula)
  end;
  ASSIGN(term,V(formula)); ASSIGN(formula,ZERO);
  stratify product; simplify stratified product;
  join term to simplified part;
  if V(formula) ≠ ZERO then goto L
end simplify sum;

comment
```

6. determination of place in the simplified part.

We suppose that 'simplified part' has as value a sum of terms, unequal to ZERO (If the value of 'simplified part' is equal to ZERO , the place of the term, just simplified and named 'term', is trivial).

At first we find out if we can put the simplified term before the first term of the simplified part by means of procedure 'try to put the term before'. When we have been successfull, we assign to 'term' the value ZERO (in procedure 'try to put the term before') in order to recognize our successfull attempt in procedure 'join term to simplified part'. When we did not have success in 'try to put the term before', we shall take the first term from the simplified part and add it temporarily to the value of 'handle two'. Then we try again to put the simplified term before the first term of the simplified part. We shall go on removing terms from the simplified part until we have found the right place for the simplified term or until no terms have been left.

At last we have to connect the terms of the 'handle two' to the simplified part again, in such a way that the order of the terms in 'simplified part' remains the correct order (by applying the principle: Last in,first out).

```
algorithm:;

procedure join term to simplified part;
begin integer t,A,B; ASSIGN(handle two,ZERO);
  for t:=TYPE(V(simplified part),A,B) while t = sum ∧ V(term) ≠ ZERO do
  try to put the term before(A)next terms:(B);
  if V(term) ≠ ZERO then
  try to put the term before(V(simplified part))next terms:(ZERO);
  if V(term) ≠ ZERO then ASSIGN(simplified part,V(term));
  for t:=TYPE(V(handle two),A,B) while t = sum do
  begin ASSIGN(simplified part,S(A,V(simplified part)));
    ASSIGN(handle two,B)
  end;
  ASSIGN(simplified part,S(V(handle two),V(simplified part)))
end join term to simplified part;

procedure try to put the term before(FIRST)and next terms:(REST);
value FIRST,REST; integer FIRST,REST;
if put before(FIRST) then
begin ASSIGN(simplified part,S(V(term),V(simplified part)));
  ASSIGN(term,ZERO)
end else
begin ASSIGN(handle two,S(FIRST,V(handle two)));
  ASSIGN(simplified part,REST)
end try to put the term before;

comment
```

In procedure 'put before' the actual comparison of simplified terms is performed. If these terms happen to be equivalent (apart from numerical factors) we have to change the coefficient of 'term' only.

Procedure 'try to put the term before' actually joins the simplified term to the simplified part.

If the formula to be simplified is a 'simplified sum' already, we have to put each next term after all terms of the simplified part. In this way we have to make much comparisons of terms. The algorithm is more efficient when we invert the order of the terms of the 'simplified part'. Then, simplifying a (partly) simplified sum, we can put (almost) each next term directly before the first term of the simplified part, while the simplification of arbitrary formulae is not affected essentially.

At the end of the simplification process we have to invert the order of the terms of the simplified part again, in order to obtain the right 'simplified sum', (see section 7.).

```
algorithm:;

Boolean procedure put before(F);value F; integer F;
begin integer d,dF,dTERM,F1,TERM,c1,c2; F1:=F; TERM:=V(term);
  d:=dF:=degree of(F1)coefficient:(c1);
  dTERM:=degree of(TERM)coefficient:(c2);
  if dF ≠ dTERM then put before:= dF > dTERM else
  begin integer hF,hTERM,d1; d1:=0;
  L:hF:=hierarchy of(F1)with degree:(dF);
    hTERM:=hierarchy of(TERM)with degree:(dTERM);
    d1:=d1 + dF;
    if hF ≠ hTERM then put before:= hF < hTERM else
    if dF ≠ dTERM then put before:= dF > dTERM else
    if d1 = d then
    begin integer A,B; put before:=true;
      ASSIGN(term,
             P(INT NUM(c1 +c2),if c2 = 1 then V(term) else
             if TYPE(V(term),A,B) = product then B else ONE ));
      ASSIGN(simplified part,
             if TYPE(V(simplified part),A,B) = sum then B else ZERO)
    end else
    begin F1:=RHS(F1); TERM:=RHS(TERM); goto L end
end end put before;

comment
```

Next, the auxiliary procedures of 'put before' follow.
In procedure 'degree of' the first parameter 'F' is supposed to be a stratified product (it will be a simplified product actually).

As a side-effect, a possibly attached numerical factor (necessarily the first factor) is removed from 'F' and retained in the second parameter 'c'.

In procedure 'hierarchy of' the first actual parameter 'F' is also supposed to be a stratified product. In the last two procedures the actual parameters are supposed to be proper factors.

```
algorithm:;

integer procedure degree of(F)coefficient:(c); integer F,c;
begin integer t,A,B,F1,n; n:=0; F1:=F; c:=1;
  for t:=TYPE(F1,A,B) while t = product do
  begin add to(n)the degree of:(A);
    if n = 0 then begin c:=RHS(A) ; F:=B end; F1:=B
  end;
  add to(n)the degree of:(F1);
  if n = 0 then begin c:=RHS(F1) ; F:=ONE end;
  degree of:=n
end degree of;
```

```
integer procedure hierarchy of(F)with degree:(d);value F; integer F,d;
begin integer A,B;
  hierarchy of:=if TYPE(F,A,B) = product then
                hierarchnumb of(A)with degree:(d) else
                hierarchnumb of(F)with degree:(d)
end hierarchy of;

procedure add to(counter)the degree of:(F);value F; integer counter,F;
begin integer t,A,B; t:=TYPE(F,A,B);
  if t = algebraic variable then counter:=counter + 1 else
  if t = integral power then counter:=counter + A else
  ERROR(t ≠ integral number,⟨F not appropriate in add to⟩)
end add to;

integer procedure hierarchnumb of(F)with degree:(d);value F; integer F,d;
begin integer t,A,B; t:=TYPE(F,A,B);
  if t = integral number then hierarchnumb of:=d:=0 else
  if t = algebraic variable then begin hierarchnumb of:=B; d:=1 end else
  if t = integral power then begin hierarchnumb of:=RHS(B);d:=A end else
  ERROR(true,⟨F not appropriate in hierarchnumb of⟩)
end hierarchnumb of;

comment
```

7. the actual simplification of an arbitrary formula.

As a formula is defined as a term or a sum of terms (def. 1.1) the simplification of an arbitrary formula can be performed by procedure 'simplify sum' of section 5.

```
algorithm:;

begin integer t,A,B,fnn; fnn:=gnn;
  DE(simplified part,ZERO, DE(term,ONE,
  DE(handle one,ONE, DE(handle two,ONE, 0))));
  simplify sum;
  comment finally we have to invert the order of the terms of the
  simplified part again.
  Remark: At this point the value of 'formula' is equal to ZERO ;
  for t:=TYPE(V(simplified part),A,B) while t = sum do
  begin ASSIGN(formula,S(A,V(formula))); ASSIGN(simplified part,B) end;
  SIMPLIFY:=ASSIGN(formula,S(V(simplified part),V(formula)));
  ERASE(fnn)
end

end SIMPLIFY;
```

```
comment

8.      Test of procedure 'SIMPLIFY'.

  In the next following block, procedure SIMPLIFY is applied to two
formulae in the algebraic variables: 'a, b, c, d, e, f'. The available
storage space (fixed by 'max of C') is chosen rather small, to demonstrate
that procedure SIMPLIFY uses the storage space efficiently.

algorithm:;

begin integer a,b,c,d,e,f,A,B,C,D,E,F,  f1,f2,  fnn;

  procedure SIMPL(f); value f; integer f;
  begin PR nlcr; PR string(⟨formula = ⟩); OUTPUT(f); PR nlcr;
    PR string(⟨number of available storage cells: ⟩);
    PR int num(free space);
    SIMPLIFY(f);
    PR nlcr; PR string(⟨simplified formula = ⟩); OUTPUT(f); PR nlcr;
    PR string(⟨number of available storage cells: ⟩);
    PR int num(free space);
    PR nlcr
  end SIMPL;

  PR nlcr; PR string(⟨Results test of 'SIMPLIFY'⟩);
L:PR nlcr; PR nlcr; PR string(⟨max of C = ⟩); PR int num(max of C);
  INITIALIZE; fnn:=gnn;
  DE(a,AV(100,1),DE(b,AV(110,2),DE(c,AV(120,3),DE(d,AV(130,4),
  DE(e,AV(140,5),DE(f,AV(150,6),DE(f1,ZERO,DE(f2,ZERO,0)))))))));
  A:=V(a); B:=V(b); C:=V(c); D:=V(d); E:=V(e); F:=V(f);
  ASSIGN(f1,S(INT POW(S(P(P(P(C,A),D),B),P(P(INT NUM(-1),F),E)),4),
              P(P(P(P(P(P(F,E),D),C),B),A),INT NUM(4)),
                S(P(INT POW(F,2),INT POW(E,2)),
                  P(P(P(INT POW(D,2),INT POW(C,2)),INT POW(A,2)),
                    INT POW(B,2)
        )     ) ) ) );
  ASSIGN(f2,P(INT NUM(-1),
              INT POW(S(P(P(P(INT POW(A,2),INT POW(B,2)),INT POW(C,2)),
                          INT POW(D,2)),
                        P(INT POW(E,2),INT POW(F,2))
                       ),
                      2
        )     )       );
  SIMPL(f1); SIMPL(f2);
  ASSIGN(f1,S(V(f1),V(f2))); SIMPL(f1);
          ERASE(fnn); PR nlcr; PR string(⟨end of example⟩); PR nlcr;
  max of C:=max of C - 1; goto L
end

end end
```

143

```
Results test of 'SIMPLIFY'

max of C = 143
formula = (cxaxdxb+(-1)xfxe)/4+fxexdxcxbxax4x(f/2xe/2+d/2xc/2xa/2xb/2)
number of available storage cells: 82
simplified formula = a/4xb/4xc/4xd/4+6xa/2xb/2xc/2xd/2xe/2xf/2+e/4xf/4
number of available storage cells: 85

formula = (-1)x(a/2xb/2xc/2xd/2+e/2xf/2)/2
number of available storage cells: 85
simplified formula = (-1)xa/4xb/4xc/4xd/4+(-2)xa/2xb/2xc/2xd/2xe/2xf/2+
        (-1)xe/4xf/4
number of available storage cells: 70

formula = a/4xb/4xc/4xd/4+6xa/2xb/2xc/2xd/2xe/2xf/2+e/4xf/4+
        (-1)xa/4xb/4xc/4xd/4+(-2)xa/2xb/2xc/2xd/2xe/2xf/2+(-1)xe/4xf/4
number of available storage cells: 69
simplified formula = 4xa/2xb/2xc/2xd/2xe/2xf/2
number of available storage cells: 82

end of example


max of C = 142
formula = (cxaxdxb+(-1)xfxe)/4+fxexdxcxbxax4x(f/2xe/2+d/2xc/2xa/2xb/2)
number of available storage cells: 81
no space left

f
excxaxdxbxcxaxdxbxcxaxdxb
excxaxdxbxcxaxdxbxcxaxdxb
1
(-1)xaxbxcxdxe/3xf/3+3xa/2xb/2xc/2xd/2xe/2xf/2+
        (-3)xa/3xb/3xc/3xd/3xexf+a/4xb/4xc/4xd/4
(-1)x(a/2xb/2xc/2xd/2+e/2xf/2)/2
(-1)xbxdxaxcxbxdxaxcxexfx(-1)xfxe+(-1)xbxdxaxcxexfx(-1)xfxex(cxaxdxb+
        (-1)xfxe)+(-1)xexfx(-1)xfxex(cxaxdxb+(-1)xfxe)/2+fxexdxcxbxax4x
        (f/2xe/2+d/2xc/2xa/2xb/2)
f
e
d
c
b
a
1
0
```

comment

Chapter 3 The simplification procedure 'POLYNOMIAL'.

In chapter 2 we have described a simplification procedure, modifying an arbitrary formula into a certain standard form. There, we have defined such a standard form called 'simplified sum', but we have noticed that this standard form easily could be changed, merely by changing the procedure 'put before'.

In this chapter we give an example of such an other standard form called 'polynomial', illustrating the connection with the system of [3] and the possibility to use these simplification procedures in that system.

1. definition of a 'polynomial'.

A 'polynomial' is defined as a formula of the form (as used in mathematical textbooks) :

a[n] + a[n − 1] × x + a[n − 2] × x↑2 + ... +a[0] × x↑n ,

where the a[i]'s are called the 'coefficients' and x the 'main variable' of the polynomial. The coefficients are defined to be 'numbers' or other 'polynomials' with as main variable an algebraic variable, having a lower hierarchy (see section 5 of chapter 2) than x.

A 'polynomial' is stored as a sum of terms, while the left operand of each sum is the product ' a[i] × x↑i ' (the product of the coefficient 'a[i]' as left operand and the i'th integral power of the main variable 'x↑i' as right operand).

If some coefficient 'a[i]' is equal to ZERO, the term ' a[i] × x↑i ' is interpreted by procedure 'P' as number ZERO, and terms equal to ZERO are not stored by procedure 'S'. These considerations imply that a number may be interpreted as a 'polynomial' in any algebraic variable with a lower hierarchy than all the variables which are used.

2. modification of procedure 'put before'.

Changing procedure 'put before' is not sufficient to modify an arbitrary formula into a 'polynomial', but may deliver an other standard form, henceforward called 'semi-polynomial', which easily can be modified into a 'polynomial' by another simplification procedure (actually procedure 'POLYNOMIAL' of the next section).

A 'semi-polynomial' has the same structure as a 'simplified sum' of chapter 2, but the order of the terms (actually 'simplified products') is determined now by the modified procedure 'put before', reproduced below. The order of the terms is chosen in such a way that terms, belonging to the same coefficient (each of which containing as a factor the same integral power of the main variable) are successive terms in the 'simplified part'. Terms belonging to the coefficient 'ai' of the i'th integral power of the main variable are 'put' before terms, belonging to the coefficient 'aj' of the j'th integral power of the main variable, if and only if i > j.

remark:
The order of the terms of the 'simplified part' is inverted at the end of procedure 'SIMPLIFY' (see chapter 2). As this is the very inefficient way in case we want to modify 'polynomials' into 'polynomials' (possibly changing the hierarchy of the variables), we release this inversion in procedure 'put before', and change the piece of program reproduced in section 7 of chapter 2 accordingly.

```
algorithm:;

Boolean procedure put before(F); value F; integer F;
begin integer dF,dTERM,hF,hTERM,n1,n2,c1,c2,F1,TERM;
  F1:=F; TERM:=V(term);
  dF:=degree of(F1)coefficient:(c1);
  dTERM:=degree of(TERM)coefficient:(c2);
L:hF:=hierarchy of(F1)degree:(n1);
  hTERM:=hierarchy of(TERM)degree:(n2);
  if hF ≠ hTERM then put before:=hF > hTERM ∧ hTERM ≠ 0 ∨ hF = 0 else
  if n1 ≠ n2    then put before:=n1 < n2 else
  begin dF:=dF - n1; dTERM:=dTERM - n2;
    if dF ≠ 0 ∧ dTERM ≠ 0 then
    begin F1:=RHS(F1); TERM:=RHS(TERM); goto L end else
    if dF ≠ 0    then put before:=false  else
    if dTERM ≠ 0 then put before:=true   else
    begin integer A,B; put before:=true;
      ASSIGN(term,
        P(INT NUM(c1 + c2),if c2 = 1 then V(term) else
        if TYPE(V(term),A,B) = product then B else ONE));
      ASSIGN(simplified part,
        if TYPE(V(simplified part),A,B) = sum then B else ZERO)
end end end put before;

ACTUAL SIMPLIFICATION:

begin integer fnn; fnn:=gnn;
  DE(simplified part,ZERO, DE(term,ONE,
  DE(handle one,ONE, DE(handle two,ONE, 0))));
  simplify sum;
  SIMPLIFY:=ASSIGN(formula,V(simplified part));
  ERASE(fnn)
end

end SIMPLIFY;
```

comment

3. simplification procedure 'POLYNOMIAL'.

The simplification procedure 'POLYNOMIAL' modifies a 'semi-polynomial' into a 'polynomial'. The properties of this procedure are discussed now.

The integral power of the main variable 'x↑j', as factor of the terms belonging to the same coefficient 'aj' has to be 'divided out'. For this reason we introduce an auxiliary formula name 'handle', to which we consecutively add terms, consisting of those factors, which remain after omitting the factor 'x↑j' from the terms.

This addition of terms to the 'handle' has inverted the order of the terms. Having re-ordered the terms, the resulting formula is referred to by the formula name 'coefficient', introduced in procedure 'POLYNOMIAL'. As this coefficient 'aj' itself is not yet of the desired standard form 'polynomial' (actually 'semi-polynomial'), we have to modify it into a 'polynomial', by means of a recursive call of procedure 'POLYNOMIAL'.

After that, the product ' aj × x↑j ' has to be formed and added to the 'polynomial part', i.e. a third auxiliary formula name, consisting of those terms ' ak × x↑k ', which are treated already (this means k > j).

Next, the following terms (if any) of the 'semi-polynomial', belonging to the next coefficient 'ai' (i < j and i > 0) are treated in the same way. The last occuring coefficient 'a0' is treated apart, because in this case any integral power of the main variable fails as a factor of these terms, so needs not to be 'divided out'.

Having treated all terms of the 'semi-polynomial', our desired standard form has been left in the 'polynomial part'. At last, this result is assigned to the actual parameter of procedure 'POLYNOMIAL', the formula name 'semi polynomial'.

algorithm:;

```
procedure POLYNOMIAL(semi polynomial); value semi polynomial;
integer semi polynomial;
begin integer VAR,V1,V2,pow1,pow2,c1,c2,R1,R2,REST,
  t,A,B,polynomial part,coefficient,handle, fnn;
  if TYPE(V(semi polynomial),A,B) = integral number then goto OUT;
  fnn:=gnn; DE(polynomial part,ZERO,DE(coefficient,ZERO,0));
  VAR:=VAR of(FIRST TERM of(V(semi polynomial))
        remaining terms:(B))
        power:(pow1)
        numerical factor of term:(c1)
        remaining factors:(R1);
  for V1:=VAR of(FIRST TERM of(V(semi polynomial),B),pow1,c1,R1)
  while V1 = VAR do
  begin ASSIGN(coefficient,P(INT NUM(c1),R1)); ASSIGN(semi polynomial,B);
  next term: V2:=VAR of(FIRST TERM of(B,REST),pow2,c2,R2);
    if V1 = V2 ∧ pow1 = pow2 then
    begin ASSIGN(coefficient,S(P(INT NUM(c2),R2),V(coefficient)));
      B:=ASSIGN(semi polynomial,REST); goto next term
    end else
```

```
    begin integer fnn1; fnn1:=gnn;
      DE(handle,V(coefficient),0); ASSIGN(coefficient,ZERO);
      for t:=TYPE(V(handle),A,B) while t = sum do
      begin ASSIGN(coefficient,S(A,V(coefficient))); ASSIGN(handle,B) end;
      ASSIGN(coefficient,S(V(handle),V(coefficient)));
      ERASE(fnn1);
      POLYNOMIAL(coefficient);
      ASSIGN(polynomial part,
        S(P(V(coefficient),INT POW(V1,pow1)),V(polynomial part)))
  end end;
  POLYNOMIAL(semi polynomial);
  ASSIGN(semi polynomial,S(V(semi polynomial),V(polynomial part)));
  ERASE(fnn);
OUT:
end POLYNOMIAL;

comment

  Next follow the auxiliary procedures of procedure 'POLYNOMIAL'.

algorithm:;

integer procedure FIRST TERM of(F,REST); value F; integer F,REST;
begin integer A,B;
  if TYPE(F,A,B) = sum then
  begin FIRST TERM of:=A; REST:=B end else
  begin FIRST TERM of:=F; REST:=ZERO end
end FIRST TERM of;

integer procedure VAR of(F, pow, num fact, rem fact); value F;
integer F, pow, num fact, rem fact;
begin integer t,A,B,LA,RA,VAR; t:=TYPE(F,A,B);
  if t = product then
  begin if TYPE(A,LA,RA) = integral number then
    begin VAR of:=VAR of(B, pow, num fact, rem fact);
      num fact:=RA; goto OUT
    end else
    begin VAR:=A; rem fact:=B; num fact:=1 end
  end else
  if t = integral number then
  begin VAR:=rem fact:=ONE; num fact:=B end else
  if t ≠ sum then
  begin VAR:=F; rem fact:=ONE; num fact:=1 end else
  ERROR(true,≠error in VAR of.≠);
  if TYPE(VAR,A,B) ≠ integral power then
  begin VAR of:=VAR; pow:=1 end else
  begin VAR of:=B; pow:=A end;
OUT:
end VAR of;
```

```
comment

4.      Test of procedure 'POLYNOMIAL'.

algorithm:;

begin integer a,b,c,d,e,f,A,B,C,D,E,F,i, formula, fnn;

  procedure POL(f,i,n,variable); value f,n; integer f,i,n,variable;
  begin PR nlcr; PR string(⟨hierarchy of variables: ⟩);
    for i:=1 step 1 until n do
    begin C2[V(variable)]:=i; OUTPUT(variable); PR string(⟨, ⟩) end;
    SIMPLIFY(f); POLYNOMIAL(f);
    PR nlcr; PR string(⟨polynomial = ⟩); OUTPUT(f); PR nlcr;
    PR string(⟨number of available storage cells: ⟩);
    PR int num(free space);
    PR nlcr
  end POL;

  PR nlcr; PR string(⟨Results test of 'POLYNOMIAL' ⟩);
L:PR nlcr; PR nlcr; PR string(⟨max of C = ⟩); PR int num(max of C);
  INITIALIZE; fnn:=gnn;
  DE(a,AV(100,0),DE(b,AV(110,0),DE(c,AV(120,0),DE(d,AV(130,0),
  DE(e,AV(140,0),DE(f,AV(150,0),DE(formula,ZERO,0)))))));
  A:=V(a); B:=V(b); C:=V(c); D:=V(d); E:=V(e); F:=V(f);
  ASSIGN(formula,S(INT POW(S(P(P(P(C,A),D),B),P(F,E)),4),
         P(INT NUM( -1 ),INT POW(S(P(P(C,E),A),P(P(D,B),F)),4))
       )          );
  PR nlcr; PR string(⟨formula = ⟩); OUTPUT(formula); PR nlcr;
  PR string(⟨number of available storage cells: ⟩);PR int num(free space);

  POL(formula,i,6, if i = 1 then a else if i = 2 then b else
                   if i = 3 then c else if i = 4 then d else
                   if i = 5 then e else if i = 6 then f else 0);
  PR nlcr; PR string(⟨Next we change the hierarchy of the variables.⟩);
  PR nlcr;
  POL(formula,i,6, if i = 1 then f else if i = 2 then e else
                   if i = 3 then d else if i = 4 then c else
                   if i = 5 then b else if i = 6 then a else 0);
  PR nlcr; PR string(⟨Old hierarchy of variables again.⟩);
  PR nlcr;
  POL(formula,i,6, if i = 1 then a else if i = 2 then b else
                   if i = 3 then c else if i = 4 then d else
                   if i = 5 then e else if i = 6 then f else 0);
  ERASE(fnn); PR nlcr; PR string(⟨end of example⟩); PR nlcr;
  max of C:=max of C - 1; goto L
end

end end
137
```

```
Results test of 'POLYNOMIAL'

max of C = 137
formula = (cxaxdxb+fxe)↑4+(-1)x(cxexa+dxbxf)↑4
number of available storage cells: 104
hierarchy of variables: a, b, c, d, e, f,
polynomial = f↑4xe↑4+(-1)xf↑4xd↑4xb↑4+(4xf↑3xe↑3xdxcxb+(-4)xf↑3xexd↑3x
        cxb↑3)xa+((-4)xfxe↑3xdxc↑3xb+4xfxexd↑3xc↑3xb↑3)xa↑3+((-1)xe↑4x
        c↑4+d↑4xc↑4xb↑4)xa↑4
number of available storage cells: 53

Next we change the hierarchy of the variables.

hierarchy of variables: f, e, d, c, b, a,
polynomial = a↑4xb↑4xc↑4xd↑4+(-1)xa↑4xc↑4xe↑4+(4xa↑3xb↑3xc↑3xd↑3xe+
        (-4)xa↑3xbxc↑3xdxe↑3)xf+((-4)xaxb↑3xcxd↑3xe+4xaxbxcxdxe↑3)xf↑3+
        ((-1)xb↑4xd↑4+e↑4)xf↑4
number of available storage cells: 53

Old hierarchy of variables again.

hierarchy of variables: a, b, c, d, e, f,
polynomial = f↑4xe↑4+(-1)xf↑4xd↑4xb↑4+(4xf↑3xe↑3xdxcxb+(-4)xf↑3xexd↑3x
        cxb↑3)xa+((-4)xfxe↑3xdxc↑3xb+4xfxexd↑3xc↑3xb↑3)xa↑3+((-1)xe↑4x
        c↑4+d↑4xc↑4xb↑4)xa↑4
number of available storage cells: 53

end of example


max of C = 136
formula = (cxaxdxb+fxe)↑4+(-1)x(cxexa+dxbxf)↑4
number of available storage cells: 103
hierarchy of variables: a, b, c, d, e, f,
no space left

f
dxbxcxexaxcxexaxcxexa
dxbxcxexaxcxexaxcxexa
1
a↑4xb↑4xc↑4xd↑4+(-1)xa↑4xc↑4xe↑4+4xa↑3xb↑3xc↑3xd↑3xexf+(-3)xa↑3xbxc↑3x
        dxe↑3xf+3xa↑2xb↑2xc↑2xd↑2xe↑2xf↑2+(-1)xaxb↑3xcxd↑3xexf↑3+4xax
        bxcxdxe↑3xf↑3+e↑4xf↑4
(-1)xaxexcxaxexcxfxbxdxdxbxf+(-1)xaxexcxfxbxdxdxbxfx(cxexa+dxbxf)+(-1)xfx
        bxdxdxbxfx(cxexa+dxbxf)↑2
f
e
d
c
b
a
1
0
```

begin comment

Chapter 4 Application of procedure 'SIMPLIFY'.

As a non-trivial example for procedure 'SIMPLIFY' we have chosen the computation of a formula determinant, i.e. the determinant of a matrix of formulae.

The problem will be, to prove the next equality:

$$\begin{vmatrix} 0 & 0 & 0 & a2+b3 & a1+b2 & -a2-b2 \\ 0 & 0 & a1+b3 & -a3-b3 & 0 & a3+b2 \\ 0 & a1+b3 & 0 & 0 & -a1-b1 & a2+b1 \\ a2+b3 & -a3-b3 & 0 & 0 & a3+b1 & 0 \\ a1+b2 & 0 & -a1-b1 & a3+b1 & 0 & 0 \\ -a2-b2 & a3+b2 & a2+b1 & 0 & 0 & 0 \end{vmatrix} =$$

$$-(a1\times b1\times(a2+b3-a3-b2)+a2\times b2\times(a3+b1-a1-b3)+a3\times b3\times(a1+b2-a2-b1))\uparrow 2 \ .$$

To evaluate a determinant we use the following procedure 'SOLDET', whose first parameter is an integral procedure (with two parameters 'i' and 'j'), delivering the values of the elements of the matrix (A(i,j), $1 \leq i \leq n$ and $1 \leq j \leq n$), and whose second parameter is the size (n) of the (square) matrix.

```
algorithm:;

integer procedure SOLDET(A,n); value n; integer procedure A; integer n;
begin integer column;
  integer procedure cofactor(i0,j,colj); value i0; integer i0,j,colj;
  begin integer fnn,result,j0,j1,auxj; Boolean even;
    integer procedure colj1;
    begin integer col; j:=j1; col:=colj;
      if col ≥ j0 then begin j:=j1+1; col:=colj end;
      colj1:=col
    end colj1;
    fnn:=gnn; DE(result,ZERO,0); even:=true;
    for auxj:=1 step 1 until n - i0 + 1 do
    begin j:=auxj; j0:=colj; even:= ¬ even;
      cofactor:=ASSIGN(result,S(V(result),
                  P(if even then MINONE else ONE,
                    P(A(i0,j0), if i0 = n then ONE else
                      cofactor(i0 + 1, j1, colj1)  ))))
    end;
    ERASE(fnn)
  end cofactor;
  SOLDET:=cofactor(1, column, column)
end SOLDET;
```

```
comment

  Actually the elements of our matrix are delivered by the procedure
'DET', so computation of the determinant of this matrix and assignation
to the formula name  'determinant' is caused by the call:
  DE(determinant, SOLDET(DET,6), 0) .

algorithm:;

integer procedure DET(i,j); value i,j; integer i,j;
DET:=if i > j then DET(j,i) else
     if i = j then ZERO else
     if i = 1 then (if j = 4 then S(V(a2),V(b3)) else
                    if j = 5 then S(V(a1),V(b2)) else
                    if j = 6 then P(MINONE,S(V(a2),V(b2)))
                    else ZERO) else
     if i = 2 then (if j = 3 then S(V(a1),V(b3)) else
                    if j = 4 then P(MINONE,S(V(a3),V(b3))) else
                    if j = 6 then S(V(a3),V(b2))
                    else ZERO) else
     if i = 3 then (if j = 5 then P(MINONE,S(V(a1),V(b1))) else
                    if j = 6 then S(V(a2),V(b1))
                    else ZERO) else
     if i = 4 then (if j = 5 then S(V(a3),V(b1))
                    else ZERO)
     else ZERO;

integer a1,a2,a3,b1,b2,b3, minone, MINONE, determinant,formula,fnn;

  INITIALIZE; fnn:=gnn; DE(minone, INT NUM(-1),0); MINONE:=V(minone);
  DE(a1,AV(101,1), DE(a2,AV(102,2), DE(a3,AV(103,3), 0)));
  DE(b1,AV(111,4), DE(b2,AV(112,5), DE(b3,AV(113,6), 0)));
  DE(determinant, SOLDET(DET,6), DE(formula,
  INT POW(S(S(P(P(V(a1),V(b1)),S(S(V(a2),V(b3)),P(MINONE,S(V(a3),V(b2))))),
           P(P(V(a2),V(b2)),S(S(V(a3),V(b1)),P(MINONE,S(V(a1),V(b3))))),
           P(P(V(a3),V(b3)),S(S(V(a1),V(b2)),P(MINONE,S(V(a2),V(b1))))),
         2)  ,0));
  PR nlcr; PR string(⟨results simplification of determinant ⟩); PR nlcr;
  PR nlcr; PR nlcr; PR string(⟨max of C = ⟩);PR int num(max of C);PR nlcr;
  PR nlcr; PR nlcr; PR string(⟨determinant = ⟩); OUTPUT(determinant);
  PR nlcr; PR nlcr; PR string(⟨determinant (simplified) = ⟩); PR nlcr;
  SIMPLIFY(determinant); OUTPUT(determinant);
  PR nlcr; PR nlcr; PR string(⟨formula = ⟩); OUTPUT(formula);
  PR nlcr; PR nlcr; PR string(⟨formula (simplified) = ⟩); PR nlcr;
  SIMPLIFY(formula); OUTPUT(formula);
  PR nlcr; PR nlcr; PR string(⟨determinant + formula = ⟩);
  ASSIGN(formula,S(V(determinant),V(formula)));
  SIMPLIFY(formula); OUTPUT(formula);
  ERASE(fnn); PR nlcr; PR nlcr; PR string(⟨end of example⟩)
end

end end
8000
```

results simplification of determinant

max of C = 8000

```
determinant = (-1)×(a2+b3)×((a1+b3)×(-1)×(a2+b1)×(a3+b1)×(a1+b2)×(a3+b2)+
        (a3+b2)×((-1)×(a1+b3)×(a3+b1)×((a1+b2)×(a2+b1)+(-1)×(-1)×(a1+b1)×
        (-1)×(a2+b2))+(-1)×(-1)×(a1+b1)×((a2+b3)×(-1)×(-1)×(a1+b1)×(a3+b2)+
        (-1)×(-1)×(a3+b3)×((a1+b2)×(a2+b1)+(-1)×(-1)×(a1+b1)×(-1)×(a2+b2)
        ))))+(a1+b2)×((a1+b3)×(-1)×(a2+b1)×((a2+b3)×(-1)×(a3+b1)×(a3+b2)+
        (-1)×(-1)×(a3+b3)×(-1)×(a3+b1)×(-1)×(a2+b2))+(-1)×(-1)×(a3+b3)×
        (-1)×(a2+b1)×((a2+b3)×(-1)×(-1)×(a1+b1)×(a3+b2)+(-1)×(-1)×(a3+b3)×
        ((a1+b2)×(a2+b1)+(-1)×(-1)×(a1+b1)×(-1)×(a2+b2)))+(a3+b2)×(-1)×
        (a1+b3)×(a2+b3)×(-1)×(a3+b1)×(a2+b1))+
        (-1)×(-1)×(a2+b2)×((a1+b3)×((-1)×(a1+b3)×(a3+b1)×(-1)×(a3+b1)×
        (-1)×(a2+b2)+(-1)×(-1)×(a1+b1)×((a2+b3)×(-1)×(a3+b1)×(a3+b2)+(-1)×
        (-1)×(a3+b3)×(-1)×(a3+b1)×(-1)×(a2+b2)))+(-1)×(-1)×(a3+b3)×((-1)×
        (a1+b3)×(a3+b1)×((a1+b2)×(a2+b1)+(-1)×(-1)×(a1+b1)×(-1)×(a2+b2))+
        (-1)×(-1)×(a1+b1)×((a2+b3)×(-1)×(-1)×(a1+b1)×(a3+b2)+(-1)×(-1)×
        (a3+b3)×((a1+b2)×(a2+b1)+(-1)×(-1)×(a1+b1)×(-1)×(a2+b2)))))

determinant (simplified) =
        (-1)×a1↑2×a2↑2×b1↑2+2×a1↑2×a2↑2×b1×b2+(-1)×a1↑2×a2↑2×b2↑2+
        2×a1↑2×a2×a3×b1↑2+(-2)×a1↑2×a2×a3×b1×b2+(-2)×a1↑2×a2×a3×b1×b3+
        2×a1↑2×a2×a3×b2×b3+2×a1↑2×a2×b1↑2×b2+(-2)×a1↑2×a2×b1↑2×b3+
        (-2)×a1↑2×a2×b1×b2↑2+2×a1↑2×a2×b1×b2×b3+(-1)×a1↑2×a3↑2×b1↑2+
        2×a1↑2×a3↑2×b1×b3+(-1)×a1↑2×a3↑2×b3↑2+(-2)×a1↑2×a3×b1↑2×b2+
        2×a1↑2×a3×b1↑2×b3+2×a1↑2×a3×b1×b2×b3+(-2)×a1↑2×a3×b1×b3↑2+
        (-1)×a1↑2×b1↑2×b2↑2+2×a1↑2×b1↑2×b2×b3+(-1)×a1↑2×b1↑2×b3↑2+
        (-2)×a1×a2↑2×a3×b1×b2+2×a1×a2↑2×a3×b1×b3+2×a1×a2↑2×a3×b2↑2+
        (-2)×a1×a2↑2×a3×b2×b3+(-2)×a1×a2↑2×b1↑2×b2+2×a1×a2↑2×b1×b2↑2+
        2×a1×a2↑2×b1×b2×b3+(-2)×a1×a2↑2×b2↑2×b3+2×a1×a2×a3↑2×b1×b2+
        (-2)×a1×a2×a3↑2×b1×b3+(-2)×a1×a2×a3↑2×b2×b3+2×a1×a2×a3↑2×b3↑2+
        2×a1×a2×a3×b1↑2×b2+2×a1×a2×a3×b1↑2×b3+2×a1×a2×a3×b1×b2↑2+
        (-12)×a1×a2×a3×b1×b2×b3+2×a1×a2×a3×b1×b3↑2+2×a1×a2×a3×b2↑2×b3+
        2×a1×a2×a3×b2×b3↑2+2×a1×a2×b1↑2×b2↑2+(-2)×a1×a2×b1↑2×b2×b3+
        (-2)×a1×a2×b1×b2↑2×b3+2×a1×a2×b1×b2×b3↑2+(-2)×a1×a3↑2×b1↑2×b3+
        2×a1×a3↑2×b1×b2×b3+2×a1×a3↑2×b1×b3↑2+(-2)×a1×a3↑2×b2×b3↑2+
        (-2)×a1×a3×b1↑2×b2×b3+2×a1×a3×b1↑2×b3↑2+2×a1×a3×b1×b2↑2×b3+
        (-2)×a1×a3×b1×b2×b3↑2+(-1)×a2↑2×a3↑2×b2↑2+2×a2↑2×a3↑2×b2×b3+
        (-1)×a2↑2×a3↑2×b3↑2+(-2)×a2↑2×a3×b1×b2↑2+2×a2↑2×a3×b1×b2×b3+
        2×a2↑2×a3×b2↑2×b3+(-2)×a2↑2×a3×b2×b3↑2+(-1)×a2↑2×b1↑2×b2↑2+
        2×a2↑2×b1×b2↑2×b3+(-1)×a2↑2×b2↑2×b3↑2+2×a2×a3↑2×b1×b2×b3+
        (-2)×a2×a3↑2×b1×b3↑2+(-2)×a2×a3↑2×b2↑2×b3+2×a2×a3↑2×b2×b3↑2+
        2×a2×a3×b1↑2×b2×b3+(-2)×a2×a3×b1×b2↑2×b3+(-2)×a2×a3×b1×b2×b3↑2+
        2×a2×a3×b2↑2×b3↑2+(-1)×a3↑2×b1↑2×b3↑2+2×a3↑2×b1×b2×b3↑2+
        (-1)×a3↑2×b2↑2×b3↑2
```

```
formula = (a1×b1×(a2+b3+(−1)×(a3+b2))+a2×b2×(a3+b1+(−1)×(a1+b3))+
          a3×b3×(a1+b2+(−1)×(a2+b1)))↑2

formula (simplified) =
          a1↑2×a2↑2×b1↑2+(−2)×a1↑2×a2↑2×b1×b2+a1↑2×a2↑2×b2↑2+
          (−2)×a1↑2×a2×a3×b1↑2+2×a1↑2×a2×a3×b1×b2+2×a1↑2×a2×a3×b1×b3+
          (−2)×a1↑2×a2×a3×b2×b3+(−2)×a1↑2×a2×b1↑2×b2+2×a1↑2×a2×b1↑2×b3+
          2×a1↑2×a2×b1×b2↑2+(−2)×a1↑2×a2×b1×b2×b3+a1↑2×a3↑2×b1↑2+
          (−2)×a1↑2×a3↑2×b1×b3+a1↑2×a3↑2×b3↑2+2×a1↑2×a3×b1↑2×b2+
          (−2)×a1↑2×a3×b1↑2×b3+(−2)×a1↑2×a3×b1×b2×b3+2×a1↑2×a3×b1×b3↑2+
          a1↑2×b1↑2×b2↑2+(−2)×a1↑2×b1↑2×b2×b3+a1↑2×b1↑2×b3↑2+
          2×a1×a2↑2×a3×b1×b2+(−2)×a1×a2↑2×a3×b1×b3+(−2)×a1×a2↑2×a3×b2↑2+
          2×a1×a2↑2×a3×b2×b3+2×a1×a2↑2×b1↑2×b2+(−2)×a1×a2↑2×b1×b2↑2+
          (−2)×a1×a2↑2×b1×b2×b3+2×a1×a2↑2×b2↑2×b3+(−2)×a1×a2×a3↑2×b1×b2+
          2×a1×a2×a3↑2×b1×b3+2×a1×a2×a3↑2×b2×b3+(−2)×a1×a2×a3↑2×b3↑2+
          (−2)×a1×a2×a3×b1↑2×b2+(−2)×a1×a2×a3×b1↑2×b3+(−2)×a1×a2×a3×b1×b2↑2+
          12×a1×a2×a3×b1×b2×b3+(−2)×a1×a2×a3×b1×b3↑2+(−2)×a1×a2×a3×b2↑2×b3+
          (−2)×a1×a2×a3×b2×b3↑2+(−2)×a1×a2×b1↑2×b2↑2+2×a1×a2×b1↑2×b2×b3+
          2×a1×a2×b1×b2↑2×b3+(−2)×a1×a2×b1×b2×b3↑2+2×a1×a3↑2×b1↑2×b3+
          (−2)×a1×a3↑2×b1×b2×b3+(−2)×a1×a3↑2×b1×b3↑2+2×a1×a3↑2×b2×b3↑2+
          2×a1×a3×b1↑2×b2×b3+(−2)×a1×a3×b1↑2×b3↑2+(−2)×a1×a3×b1×b2↑2×b3+
          2×a1×a3×b1×b2×b3↑2+a2↑2×a3↑2×b2↑2+(−2)×a2↑2×a3↑2×b2×b3+
          a2↑2×a3↑2×b3↑2+2×a2↑2×a3×b1×b2↑2+(−2)×a2↑2×a3×b1×b2×b3+
          (−2)×a2↑2×a3×b2↑2×b3+2×a2↑2×a3×b2×b3↑2+a2↑2×b1↑2×b2↑2+
          (−2)×a2↑2×b1×b2↑2×b3+a2↑2×b2↑2×b3↑2+(−2)×a2×a3↑2×b1×b2×b3+
          2×a2×a3↑2×b1×b3↑2+2×a2×a3↑2×b2↑2×b3+(−2)×a2×a3↑2×b2×b3↑2+
          (−2)×a2×a3×b1↑2×b2×b3+2×a2×a3×b1×b2↑2×b3+2×a2×a3×b1×b2×b3↑2+
          (−2)×a2×a3×b2↑2×b3↑2+a3↑2×b1↑2×b3↑2+(−2)×a3↑2×b1×b2×b3↑2+
          a3↑2×b2↑2×b3↑2

determinant + formula = 0

end of example
```

References

[1] R.P. van de Riet, Formula manipulation in ALGOL 60, part 1, Mathematical Centre Tracts nr. 17, Mathematisch Centrum.

[2] R.P. van de Riet, Garbage collection methods for ABC in ALGOL 60, TW report 110, Mathematisch Centrum.

[3] W.P. de Roever, An exact rational function system with garbage collection in ALGOL 60, MR 119/70 september, Mathematisch Centrum.

[4] D. Grune, Handleiding Milli-systeem voor de EL X8, LR 1.1, april 1971, Mathematisch Centrum.