RA

stichting mathematisch centrum

MC

REKENAFDELING MR 119/70 SEPTEMBER

W.P. DE ROEVER

AN EXACT RATIONAL FUNCTION SYSTEM WITH GARBAGE COLLECTION IN ALGOL 60

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.

Contents

Summary:

The program, contained in and commented on in this paper, originated from a suggestion by R.P. VAN DE RIET to develop an infinite precision rational function system, using the formula manipulation methods for ALGOL 60 described in [12] and garbage collection methods described in [13]. It is part of a future extension of his formula manipulation system, to be named ABC for

"Algebraische Bewerkingen met de Computer".

I.1. Introduction to formula manipulation and garbage collection.

As an introduction to formula manipulation, consider the following ALGOL 60 program.

```
begin integer one,zero,sum,product,algebraic variable,k;
  integer array C[1:3,1:1000];
  integer procedure STORE (lhs,type,rhs); value lhs,type,rhs;
  integer lhs,type,rhs;
  begin STORE:= k:= k + 1; C[1,k]:= lhs;
    C[2,k]:= type; C[3,k]:= rhs
  end STORE;

  integer procedure TYPE(f,lhs,rhs); value f; integer f,lhs,rhs;
  begin lhs:= C[1,f]; TYPE:= C[2,f]; rhs:= C[3,f] 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);

  comment

  Suppose one wishes to calculate:
        f = (x × x + x) × dy/dx + (y × y + y) × dx/dx,
  which is a trivial problem, but illustrates the need for automatic
  garbage collection.

  The calculation is performed by the following actual program;

ACTUAL PROGRAM:
  begin integer x,y,f;
    x:= STORE(0,algebraic variable,0);
    y:= STORE(0,algebraic variable,0);
    f:= S(P(S(P(x,x),x),
            DER(y,x)),
```

```
P(S(P(y,y),y),
  DER(x,x)
));
```

comment

Since declarations of operators are not feasible in ALGOL 60, we have to transform formulas as used in mathematical textbooks into Polish prefix, that is functional notation, before trying to construct representations of these by means of function designators. In the above program we interpret the usual sum, product and derivative operations by the integer procedure S,P and DER. Corresponding to certain formulas we construct function designators availing ourselves of the afore - mentioned interpretation of operations, which during their execution, result in the construction of objects internal to the array C.
This process is comparable to a Goedelnumbering of a finite class of formulas of some formal system, such that natural numbers to which no formula correspond in the used begin segments of natural numbers are avoided during the process of construction.

;

end
end

The result of the calculation is that f = ((y × y) + y). But during the calculation process the expression S(P(x,x),x) has been evaluated, resulting in the storage of the useless formula g = ((x × x) + x) into the array C.
This formula is useless for two reasons:
a) it is not used for building up f,
b) it cannot be used later on, since it is not known where it is stored in C.
Therefore, we may freely consider this formula as garbage. To get rid of it is not a simple matter, since it occupies space in C which is surrounded by space in which still interesting formulas are stored (y and f).

Consider the situation that occurs, when the array C has been filled up completely during execution of some particular program.
Then the question arises, whether unneccessary information has been stored, e.g. the object corresponding to g.

Suppose each subvalue referred to by the name possessed by the slice C[1:3,i],i = 1,2,..,1000, may be "marked" - the manner in which is discussed in the commentary following the declaration of procedure COLLECT GARBAGE at the end of section 1.1..
Then the multiple value referred to by the name possessed by C is marked in the following steps:

step 1: If there exists a subvalue referred to by the name of a slice

C[1:3,i],i= 1,2,..,1000, which is not yet "marked" and which is referred to by a name possessed by a "formula-identifier" (defined in the sequel) then this value is "marked" and step 1 is taken again;
otherwise, step 2 is taken.

step 2: If there exists a subvalue referred to by the name of a slice C[1:3,i],i = 1,2,..,1000, which is not yet "marked" and which is referred to by a component of a "marked" subvalue, then this subvalue is "marked" and step 2 is taken again;
otherwise, the marking of C is complete.

Now the subvalues referred to by names of slices C[1:3,j], j = 1,2,.., 1000,which are not "marked" will not anymore be relevant for computation during the execution of the program and therefore be considered and henceforward be defined as the <u>garbage of C</u>.

So to determine the garbage of C, the computer must be able to distinguish between those identifiers, to which formulas - names of internal objects of C, i.e. values of certain function designators like STORE,S,P and DER in the previous example - have been assigned, we shall call them in the sequel "formula - identifiers", from those for which this is not the case.
In ALGOL 68 the modes of those identifiers provide an indication of this. In our system we have to construct explicitly a list of those identifiers, to be consulted in case a call of COLLECT GARBAGE, the central garbage determining and free space providing procedure, results in a garbage collection. Actually "list of identifiers" is confusing and erroneous, for what really matters is a list containing the values assigned to those formula-identifiers, i.e. the names of to be saved internal objects of C.
Moreover, one cannot handle lists of ALGOL 60 declared identifiers in an ALGOL 60 program.

Therefore we relax the link between formula-identifier and assigned formula, by assigning the formula-identifier the name of the place in the list, where the formula has been stored. That is, realize the ALGOL 68 name concept in an ALGOL 60 program, by assigning the formula-identifier the name, it would possess in ALGOL 68, and store the value, the name would refer to in ALGOL 68, in that list, which is the function of the <u>integer</u> <u>procedure</u> SAVE: a call SAVE(F) results in computing the value of its actual parameter F, after which the obtained value is stored in a list at a place referred to by SAVE's value.

An additional procedure is now neccessary to obtain the stored value, a formula, from a formula-identifier. This is the function of <u>integer</u> <u>procedure</u> V.

In de sequel we shall use "refer to", as applied to our ALGOL 60 program, for

(i) the relation that exists between a formula – as defined above – and the object it specifies in array C and

(ii) the relation that exists between the value of a formula-identifier and the formula stored – as mentioned above – at a place in the list of to-be-saved names specified by this value.

Since we shall store this list in C itself, the two cases of "refer to" will coincide.
Therefore a formula is a name referring to an object internal to C and the value of a formula-identifier is a name referring to a formula.
This terminology has been derived from [15] and is an interpretation of situations,occurring in this kind of formula manipulation in ALGOL 60, in ALGOL 68, as suggested by a remark of VAN WIJNGAARDEN'S.
This is the function of the <u>integer</u> <u>procedure</u> V.

Now suppose one is writing the <u>integer</u> <u>procedure</u> S for storing a sum.
After declaration of, followed by assigning formula's in the above relaxed sense to, the integers i and j, one subsequently tests the execution of expressions S(V(i),S(V(i),V(j))) and S(S(V(j),V(j)), S(V(i),V(i))).
During execution of the latter arithmetical expression, after execution of S(V(j),V(j)), one has to save the object stored from the garbage collection, as garbage collection may occur during computation of S(V(i),V(i)).
This contrasts with the computation of the former expression, as garbage collection occurring during computation of S(V(i),V(j)) does not erase the internal object referred to by V(i), its name being contained in the list of names of internal objects to be saved.
So names referring to internal objects of C have to be distinguished according to their being possible garbage or not.
In ALGOL 68 no problem of this kind occurs, as the use of a global generator in the identity declaration of the identifier provides for this distinction.

As checking on occurrence in this list is too time-consuming a process, we mark names of saved internal objects (by adding 100000, the value of the, in the embracing block, declared <u>integer</u> saved, confusion not arising as 100000 excels any possible upperbound of C).

The introduction of a name upon relaxation of the link between formula-identifier and formula has, amongst other things, as a consequence that this name has to have a scope, which corresponds with the smallest embracing block in whose heading that identifier occurs, i.e.

(i) upon its declaration the link identifier – name has to be constructed, in our case by the integer procedure DE, and

(ii) upon leaving the block, in whose heading that identifier occurs, the name must cease to exist, i.e. the object of which the name is the formula referred to by the value of that identifier, needs not to be marked when garbage collection occurs.

The latter requirement is the reason to store this list in C itself, and we add the space occupied by those names to the space available for formula manipulation in C, the free space of C.

This is the task of the procedure ERASE. When called upon, it is the last statement prior to leaving the relevant block and it functions due to the principle last in – first out, made possible by the ALGOL 60 block structure. The number of saved names, counted by the integer gnn (global number of names), is assigned as first statement of the block to a locally declared counter, fnn, and subsequently raised as new names are created by calls of SAVE. It is a precise standard which names have been added corresponding to declared identifiers. If, during elaboration of a program, a block, in which new names might have been introduced, is not left by elaborating its textually last statement ERASE(fnn), due to elaboration of a goto statement, leading outside this particular block, the explicitly defined successor has to be a statement, which is or contains as first–to–be–elaborated statement ERASE(snn), where integer snn (second number of names) has been assigned exactly the number of names needed for further elaboration of the program. Another consequence of the link between formula–identifier and formula is, that a special procedure, whose call replaces assignments to formula–identifiers, has to be constructed, the integer procedure ASSIGN. By making it a function designator one provides for the ALGOL 68 value of an assignation.

For a description of particular garbage collection methods in ALGOL 60, for this kind of formula manipulation system, I refer to[13]. VAN DE RIET describes in this article two techniques: the relocation method and the free list technique.
The relocation method has as a possible advantage the feature that each saved object is relocated after garbage collection as a whole, that means, slices C[1:3,i] of the array C used for storage of one particular object have succeeding subscripts, so the usual referencing within the array C, to different components of that object, may be avoided e.g., by specifying the number of necessary slices.
The main feature of the free list technique is that after garbage collection no relocation of the saved objects takes place. The garbage has as structure a linked list called the free list (see below). The argument for using the free – list technique have been given in the above mentioned paper.

I.2.1. Modes and linking complexity.

The next ALGOL 68 declaration of mode formula clarifies our use of "formula" as stated in I.1.

```
union formula = (ref short integer, ref algebraic variable, ref triple,
                 ref multilinked structure, ref linked list);

struct short integer = (int value);
struct algebraic variable = (string name);
struct triple = (formula left operand, int operator, formula
                                        right operand);
struct linked list = (ref linked list list, int value);
struct multilinked structure = (ref multilinked structure multilinked
                                        list, formula coefficient);
int sum = 33, product = 34, rational function = 35, rational number = 36,
    quotient = 37;
```

The described marking of non - garbage and making an object of mode linked list of the garbage of C, is the task of the procedure COLLECT GARBAGE.
After marking the list of names of objects (formulas) to - be - saved, the garbage collector proceeds by marking those objects, guided by the names those objects contain, which can be best demonstrated by the mode declarations in the above ALGOL 68 declaration prelude. They reflect exactly the linking complexity, i.e. the complexity of the manner in which names are contained, of the realizations of objects of corresponding modes in the array C.

```
begin integer free cell,last free cell,last name,max of C,algebraic variable,
  sum,product,quotient,one,zero,ONE,ZERO,long integer,short integer,rational
  function,rational number,polynomial,multilinked structure,auxiliary,saved,
  gnn,fnn,snn,minone,MINONE,G,Gmin1,di; real dii;
  max of C:= read; G:= read; comment our choice for G is 10 ↑ 6;
  begin integer array C1[1:max of C]; real array C2[1:max of C];
```

comment
As in the formula manipulation program specified in I.1.1., the mode concept has been realized in the array C, by declaring integer identifiers having the same names as the corresponding modes and assigning them values, analogous to the above identity declaration, by call of the procedure INITIALIZE - see next section.

Here follows a short discussion and classification, according to linking complexity, of those realizations:

(i) short integer, algebraic variable.

None of the components of the subvalue is a name of other subvalues.

Objects of these modes, respectively algebraic variables and short integers, are comparable to algebraic variables and integers whose absolute value is limited by the value of the expression G – 1.

(ii) sum, product, quotient, rational function, rational number.

Both of the components of the subvalue are names, if one neglects the operator field component. The structure of the object corresponds to a general binary tree.
Objects of this mode are comparable with the usual interpretation given to sum, product, quotient, rational number and rational function.

(iii) long integer.

Objects of this mode correspond to objects of mode linked list, with end specified by C1[last element] = 0. Generally a linked list may be realized in C as follows, integers first element and last element having been declared and assigned values:

(a) The value possessed by (C1[first element],C2[first element]) is its first subvalue, referred to (in our interpretation of an ALGOL 60 program in ALGOL 68 terminilogy) by the value of integer first element.

(b) If the value possessed by (C1[i],C2[i]) is a subvalue of the linked list, its successor is referred to by the value of C1[i], if C1[i] ≠ last element.

(c) The value possessed by (C1[last element],C2[last element]) is its last subvalue.

(iv) polynomial, multilinked structure.

An object of the second mode, as realized in C, corresponds to an object of mode multilinked structure.
An object of the first mode, as realized in C, corresponds again to a multilinked structure, however, its first coefficient field is of mode ref algebraic variable, and is comparable to a polynomial in the variable specified by its first coefficient field.
Both of these objects can be realized in C as objects of mode linked list, with, for each of the elements of these lists, the second component (the value of C2[i]) of the subvalue possessed by (C1[i],C2[i]) being a formula and in case of a polynomial the second component of the first subvalue of the list referring to an object of mode algebraic variable.

(v) In [12,section 2.9] and [15,11.11 f] another mode arises, that of a function (possibly specified by C1[i]) with argument (possibly referred to by the value of C2[i]). It does not occur in our system.

While in this section the difference between polynomials and objects of mode long integer or short integer has been stressed, in [4] COLLINS emphasizes their similarity, by considering an object of mode long integer as a polynomial of degree zero, using the concept "list of order n".

A list of order n may be defined recursively as:

(i) a list of order zero is an object of mode linked list,

(ii) a list of order n, n a national number, is an object of mode multilinked structure, whose coefficient fields refer to lists of order n − 1.

The importance of this concept in his system is, that only arithmetical operations between lists of the same order can be performed by its subroutines.
Let the value of Q refer to an object of mode long integer and the value of P refer to a list of order n, $n > 0$, e.g., a polynomial in the variables x[1], ... , x[n] (consult section 3.1.1. for explanation), the value of each referring to an object of mode algebraic variable.
If in COLLINS' system addition of P to Q is required, one has to construct explicitly a list of order n, in our system referred to by the value of (consult section 1.3)

```
STORE ARRAY(i,-1,0,polynomial,if i = -1 then x[n] else
        STORE ARRAY(i,-1,0,polynomial,if i = -1 then x[n - 1] else - ..
              ... STORE ARRAY(i,-1,0,polynomial,
                               if i = -1 then x[1] else Q) ...)),
```

corresponding to "(..(Q × x[1] ⋀ 0) × .. × x[n] ⋀ 0)".
We do not wish to introduce such versions of "the same number" in this system. This point of view has been expounded in section 3.1.1.

In the sequel we shall use the words "linked list", "multilinked structure", "polynomial", "short integer", "algebraic variable" for an object of mode linked list, multilinked structure, multilinked structure with first coefficient field of mode algebraic variable, short integer, algebraic variable, respectively the words "sum", "product", "quotient", "rational number", "rational function" for an object of mode triple with the value of the operator field equalling the value of integers sum, product, quotient, rational number, rational function, respectively, and the word "long integer" for a "linked list".

I.2.2. The free-list garbage collection technique.

The available space for storage of information is structured as a linked list and realized in C, compare I.2.1. (iii), with its first subvalue specified (referred to, in our interpretation) by the value of integer free cell and its last subvalue by the value of integer last free cell.
When for the execution of the program a new subvalue is needed

(i) free cell is made to refer to this object after the value of C1[free cell] has been saved in an auxiliary integer k,

(ii) if free cell = last free cell, the garbage collector comes into operation and a new linked list is formed of the garbage of C,

(iii) if free cell ≠ last free cell, the assignment free cell:= k is performed.

Having now at our disposal the notion of multilinked structure, we are able to characterize the garbage of C more precisely. The list of names of objects to - be - saved, the name - list, constitutes a linked list, if those names are considered separate from the objects they refer to. If the value of integer i refers to one of its subvalues, the value of C1[i] is a name of a to - be - saved object and, if C2[i] ≠ 0, the value of C2[i] is the name of a next subvalue of the list, with its last subvalue referred to by the value of integer last name.
However the object name - list taken together with the objects its contained names refer to, constitute a multilinked structure (this is here a matter of interpretation, in ALGOL 68 it is forced by the mode declaration).

Garbage of C is exactly the complement with respect to C of the space occupied by this object.

;

```
integer procedure ERROR(b,s); boolean b; string s;
if b then
begin PR nlcr; PR string(s); EXIT; ERROR:= 1 end;


procedure INITIALIZE;
begin integer i; for i:= 1 step 1 until max of C do C1[i]:= i + 1;
  free cell:= 1; last free cell:= max of C; Gmin1:= G - 1;
  last name:= 0; saved:= 100 000; gnn:= fnn:= 0;
  algebraic variable:= 1 + 0 × 16;
  short integer:= 2 + 0 × 16;
  sum:= 1 + 2 × 16; product:= 2 + 2 × 16; quotient:= 5 + 2 × 16;
  rational function:= 3 + 2 × 16; rational number:= 4 + 2 × 16;
  polynomial:= 1 + 3 × 16; multilinked structure:= 2 + 3 × 16;
  long integer:= 1 + 4 × 16;
```

```
  DE(one,STORE(0,short integer,1),DE(zero,STORE(0,short integer,0),
  DE(minone,STORE(0,short integer,-1),0)));
  ONE:= V(one); ZERO:= V(zero); MINONE:= V(minone); snn:= gnn;
end INITIALIZE;

integer procedure TYPE(F,A,B); value F; integer F,A; real B;
begin ERROR(F < 0 ∨ F = 100000,‹F not appropriate in TYPE›);
  if F > saved then F:= F - saved; B:= C1[F];
  A:= B : 128; TYPE:= B - A × 128; B:= C2[F]
end TYPE;
```

comment

We specify the above mentioned modes by declaring (in I.2.1) identifiers carrying the names of those modes and assigning them values in INITIALIZE. As a result, when writing one's own particular program, using the library prelude developed in this paper, its first statement has to be a call of INITIALIZE.

In I.1. we quoted from [13] a program for formula manipulation in its simplest form. In that program execution of the assignment f:= STORE(lhs,type,rhs) creates an object in the array C, corresponding to the sum or product, according to the value of type, of objects representing subformulas referred to by lhs and rhs.
This object is the contents of slice C[f,1:3].
As the length of a computer word affords us to store integers much larger than any possible bound of an array, for efficient use of the available memory we store not the multiple value (lhs, type, rhs) but (lhs × 128 + type, rhs), so affording 128 possible modes of objects by coding. This explains our use of a disguished two dimensional array C1, C2[1:max of C] and the decoding by TYPE(F,A,B), which results in TYPE being assigned the mode of F, A:= lhs and B:= rhs.

Structure enquiries to modes are performed in keeping with the above classification according to linking complexity by

;

```
Boolean procedure MONADIC OP(t); value t; integer t;
MONADIC OP:= t : 16 = 1;

Boolean procedure DYADIC OP(t); value t; integer t;
DYADIC OP:= t : 16 = 2;

Boolean procedure MULTILINKED STRUCTURE(t); value t; integer t;
MULTILINKED STRUCTURE:= t : 16 = 3;

Boolean procedure LINKED LIST(t); value t; integer t;
LINKED LIST:= t : 16 = 4;
```

comment

They function mainly in COLLECT GARBAGE and the boolean procedure EQ, which establishes equality of two objects, referred to by the value of X and Y, in respects specified by the algorithm described by its body.

;

```
boolean procedure EQ(X,Y); value X,Y; integer X,Y;
 if X = Y ∨ abs(X - Y) = saved then EQ:= true else
 begin integer tX,tY,XA,YA; real XB,YB;
   tX:= TYPE(X,XA,XB); tY:= TYPE(Y,YA,YB);
   if tX = tY ∧ (tX = short integer ∨ tX = algebraic variable ∨
           MONADIC OP(tX) ∨ DYADIC OP(tX)) then
   EQ:= if tX = short integer ∨ tX = algebraic variable then
           XB = YB else
   if MONADIC OP(tX) ∧ XB = YB then EQ(XA,YA) else
   EQ(XA,YA) ∧ EQ(XB,YB)
   else
   if tX = tY ∧   (MULTILINKED STRUCTURE(tX) ∨ LINKED LIST(tX)) then
   begin if (if   MULTILINKED STRUCTURE(tX) then EQ(XB,YB) else
     XB = YB) then
   L: begin if XA = 0 = YA = 0 then
      begin if XA = 0 ∧ YA = 0 then begin EQ:= true; goto OUT end
      else
      if (if MULTILINKED STRUCTURE(tX) then EQ(C2[XA],C2[YA])
                                      else C2[XA] = C2[YA]) then
      begin XA:= C1[XA]; YA:= C1[YA]; goto L end
     end
   end; EQ:= false;
 OUT:
 end
 else EQ:= false
end;
```

comment

Adding a name to the multilinked structure of non garbage objects of C1, C2 is performed by:

;

```
integer procedure SAVE(F); value F; integer F;
```

begin comment
 SAVE(F) saves F from garbage collection by adding a cell, whose name is the value assigned to SAVE, to the name list. The procedure body of DE contains the statement SAVE(F) for creating a cell of the name-list, the name of which is assigned to a formula-identifier as an extension of the ordinary declaration. This cell may be used for storing future formulas by means of calls of ASSIGN.

```
  The value of gnn, the number of declared names, is needed for
  realizing in the name - list the scope of corresponding formula -
  identifiers;

  integer k;
  ERROR(F < 0 ∨ F = 100000,‹ F not appropriate in SAVE›);
  if F ≥ saved then F:= F - saved; gnn:= gnn + 1;

  k:= C1[free cell]; C1[free cell]:= F; C2[free cell]:= last name;
  SAVE:= last name:= free cell;

  comment a name has been added to the name - list, so the
          question whether there is still space left in C arises;

  COLLECT GARBAGE(0,auxiliary,k)
end SAVE;

comment
As a complement to SAVE, shrinking the name - list, functions the
procedure ERASE:
                                                                  ;

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,‹ERASE not appropriate›)
end ERASE;

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

comment
For use in arithmetical expressions defining the value of a
function designator, as in S, P and Q, the next two procedures
proved to be convenient:
                                                                  ;

integer procedure RS(n,F); integer n,F;
begin ERASE(n); RS:= F end;

integer procedure SR(n,F); integer n,F;
begin SR:= F; ERASE(n) end;

comment

Formula-identifiers are now initialized, after declaration of integers
f1,f2,..,fn, by DE(f1,F1,DE(f2,F2,..,DE(fn,Fn,0)..)), which call
results in adding new objects, referred to by the values of f1,..,fn,
to the linked name-list such that C1[f1] = Fi.
```

If Fi $\neq$ 0, the value of Fi refers to an object to be marked during garbage collection, otherwise in stead of assigning a name of a subvalue of C to C1[fi] – the lowerbound of C is 1 – C1[fi]:= 0 is elaborated.
The declaration of integer f together with the call DE(f,0,0) may be compared with the ALGOL 68 identity declaration
ref formula f = heap formula.

If we try to interpret MAILLOUX's suggestion – see [9] – for the implementation of heap generators versus local generators, the stack would consist only of the name – list, while the rest of C would occupy the heap.

Merely for reasons of syntactic checking, we assign the negative values of the integers corresponding with names.

;

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

comment

Assignment to formula – identifier is performed by

;

```
integer procedure ASSIGN(f,F); value f,F; integer f,F;
begin ERROR(f < – max of C ∨ f > 0,
  ≮name not appropriate in ASSIGN≯); if F ≥ saved then F:= F – saved;
  ASSIGN:= F + saved; C1[–f]:= F
end ASSIGN;
```

comment

To obtain a formula from the value of a formula-identifier that refers to it we use

;

```
integer procedure V(f); value f; integer f;
V:= if f > 0 then ERROR(true,≮name > 0 in V≯) else C1[–f] + saved;
```

comment

Conditional saving without corresponding formula-identifier is performed by

;

```
integer procedure EV(A); value A; integer A;
if A < saved then begin SAVE(A); EV:= A + saved end else EV:= A;
```

comment

The procedure TRACE marks each subvalue of the object referred to by the value of F in a manner that reflects our particular way of constructing objects in C, as discussed in I.3.. This is such that C1[F] ≥ 0, C1[F] = 0 only occurring if F refers to a subvalue of the name - list originating from a call DE(f,0,0), the value of C1[F] being positive in all other cases.
Thus the condition is fulfilled for the assignement C1[F]:= -C1[F] - 1 to mark the subvalue referred to by the value of F. (Labels of sentences below correspond to labels in COLLECT GARBAGE).

1: The condition F ≥ 0 reflects the lower bound of C being 1.

2: The condition C1[f] ≥ 0 reflects the need to mark only unmarked slices.

3: This block reflects exactly our discussion of a partitioning of the set of modes in classes of linking complexity.

4: Garbage collection is necessary if free cell = last free cell, else there is no objection left to the assignment of a free cell:= fc.

5: This statement, auxiliary to the integer procedure STORE, is explained in the discussion of STORE, see I.3..

6: This for - statement marks the multilinked structure which consists of name - list plus referred to objects.

7: Garbage has been determined and transformed into a linked list while undoing the marking of non garbage.

8: As the garbage of C is empty the call for garbage collection has been of no avail.

I.3. Representation of formulas in C1,C2.

In I.2.1. we classified the modes used in terms of linking complexity. Tracing objects, names in other objects refer to, we end up, as circular reference does not occur in this system, in objects containing no names anymore. These are algebraic variables, short integers and, in a certain sense, long integers, taking into account that use of a relocation method of garbage collection would have resulted in an object containing no names (see page 5), although the free - list technique requires a linked list for storing a long integer.

Every object internal to C1 and C2 originates directly or indirectly from a call of integer procedure STORE:

```
integer procedure ES(A1,A); integer A1,A;
begin A1:= A; if A1 < saved then begin SAVE(A1); A1:= A1 + saved
                                 end; ES:= A1
end ES;
```

comment

To obtain the ALGOL 68 feature of value of a closed clause whose constituents are assignations to formula identifiers, for the sake of convenience the next procedure, besides ASSIGN and DE, is used. It functions mainly in OPER ON NUM and OPER ON RAT, the central arithmetic performing procedures.

;

```
integer procedure Multiple ES(A1,A,B); integer A1,A,B;
begin A1:= A; if A1 < saved then begin SAVE(A1); A1:= A1 + saved end;
  Multiple ES:= B
end;

procedure COLLECT GARBAGE(n,aux,fc); value n; integer n,fc,aux;
begin integer i,a;
  procedure TRACE(F); value F; integer F;
1:  if F > 0 then
2:  begin if C1[F] > 0 then
3:    begin integer t,A; real B; t:= TYPE(F,A,B);
       if MONADIC OP(t) then TRACE(A) else
       if DYADIC OP(t) then begin TRACE(A); TRACE(B) end else
       if MULTILINKED STRUCTURE(t) ∨ LINKED LIST(t) then
       begin if MULTILINKED STRUCTURE(t) then TRACE(B); for A:=
                                           A while A ≠ 0 do
         begin if MULTILINKED STRUCTURE(t) then TRACE(C2[A]); a:= A;
           A:= C1[A]; C1[a]:= - C1[a] - 1
       end end; C1[F]:= - C1[F] - 1
  end end TRACE;
4:  if free cell ≠ last free cell then free cell:= fc else
    begin free cell:= 0;
5:    TRACE(aux);
      i:= last name;
6:    for i:= i while i ≠ 0 do
      begin TRACE(C1[i]); C1[i]:= - C1[i] - 1; i:= C2[i] end;
7:    for i:= 1 step 1 until max of C do
      if C1[i] > 0 then
      begin if free cell = 0 then free cell:= last free cell:= i else
        join to free space(i)
      end else C1[i]:= - C1[i] - 1;
8:  ERROR(free cell = 0,⊰no space left⊱);
end end COLLECT GARBAGE;
```

```
integer procedure STORE(A,t,B); value A,t,B; integer A,t; real B;
begin integer k;
  if MONADIC OP(t) ∧ A ≥ saved then A:= A − saved else
  if DYADIC OP(t) then
  begin if A > saved then A:= A − saved;
    if B ≥ saved then B:= B − saved
  end;

  comment

  The preceding conditional statement cancels the marking of
  saved formulas as mentioned in I.1. page 2  before they are
  stored in C;

  STORE:= free cell; k:= C1[free cell];
  C1[free cell]:= A × 128 + t; C2[free cell]:= B;
  auxiliary:= free cell;

  comment

  The subvalue referred to by the value of free cell is now an
  object of mode specified by the value of t. As the value of free
  cell has not been added to the name − list, garbage collection
  might destroy the subvalue it refers to.
  Execution of the statement labelled by 5 in the procedure
  body of COLLECT GARBAGE prevents this;

  COLLECT GARBAGE(1,auxiliary,k)
end STORE;

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

integer procedure S INT(i); value i; integer i;
S INT:= if i < G then STORE(0,short integer,i) else
ERROR(true,‹i ≥ G in S INT›);

comment

Linked lists and multilinked structures are stored by STORE ARRAY.
                                                                 ;

integer procedure STORE ARRAY(i,low,up,type,Ai);
value low,up,type; integer low,up,type,i,Ai;
begin integer p,fnn; boolean linked list; real q;
  real procedure AI;
  if linked list ∧(i < up) then
  begin real m; m:= Ai; i:= i + 1; AI:= G × Ai + m end
  comment the value of G is such that G × G − 1 fits in one word
  else AI:= Ai;
```

```
  ERROR(low>up,‹low>up in STORE ARRAY›); linked list:= LINKED LIST(type);
  fnn:= gnn; i:= low; q:= AI;
  if ¬ linked list ∧ q ≥ saved then q:= q − saved;

L: STORE ARRAY:= p:= STORE(0,type,q); SAVE(p);
  low:= low + (if linked list then 2 else 1);
  for i:= low step 1 until up do
  begin C1[p]:= if i = low then free cell × 128 + type else free cell;
    p:= free cell; q:= C1[p]; C1[p]:= 0; C2[p]:= 0;
    COLLECT GARBAGE(0,auxiliary,q); q:= AI;
    if ¬ linked list ∧ q ≥ saved then q:= q − saved; C2[p]:= q
  end; ERASE(fnn)
end STORE ARRAY;
```

comment

The operation of STORE ARRAY splits up in two:
the object to be constructed is (i) a polynomial or (ii) a long integer.

(i) The value of AI equals the value of Ai.

At L the first coefficient Ai for i = low is stored and saved, by creating a head of the required mode, that is saved dynamically. If garbage collection occurs during construction of the object, TRACE(C2[p]) creates no difficulties on account of the condition in the conditional statement labelled by 1 in the procedure TRACE, declared in COLLECT GARBAGE.
Moreover the partially constructed object is saved by SAVE(p), the statement following the statement labelled by L.

(ii) In principle a long integer is a linked list with names specified by C1[i] and values stored in C2[i], if the value of i refers to one of its subvalues, and |C2[i]| < G. As $G \uparrow 2 - 1$ fits in one computer word (see REMARK preceding the declaration of the multiplication procedure MULT in 2.1.1.) we may encode two of such values in one word. This motivates the use of AI. Garbage collection occurring during construction is commented upon in (i).
;

```
integer procedure LONG INT(i,length,Ii); value length;
integer i,length,Ii;
begin boolean b; b:= true; i:= length + 1;
  for i:= i − 1 while i > 1 ∧ b do if Ii = 0 then length:=
  length − 1 else b:= false; i:= 1;
  LONG INT:= if length > 1 then STORE ARRAY(i,1,length,long integer,Ii)
  else
  if length = 1 then S INT(Ii) else ZERO
end LONG INT;
```

```
boolean procedure int(X); value X; integer X;
begin integer t; t:= TYPE(X,di,dii);
  int:= t = short integer ∨ t = long integer
end;

integer procedure POL(i,degree,X,Ci); value degree,X;
integer i,degree,X,Ci;
begin boolean b; b:= true; i:= degree + 1;
  for i:= i − 1 while i > 0 ∧ b do
  if EQ(Ci,ZERO) then degree:= degree − 1 else b:= false; i:= 0;
  POL:= if degree = 0 ∧ int(Ci) then Ci else
  STORE ARRAY(i,−1,degree,polynomial,if i = −1 then X else Ci)
end POL;
```

comment

For one by one storing arbitrary formulas, e.g., referring to objects not having the same hierarchy of variables(see III.1.1),

(i) the formula F, that has been constructed first, is stored and saved by elaborating ES(L,STORE ARRAY(i,0,0,multilinked structure,F)) and,

(ii) after formula G has been constructed, it is stored and saved by elaborating ADD TO(L,G).

;

```
procedure ADD TO(L,E); value L,E; integer L,E;
begin integer A,q; tp(53);
  ERROR(¬ MULTILINKED STRUCTURE(TYPE(L,A,dii)),
        ≮type of L not appropriate in Add to≯);
  if L > saved then L:= L − saved; if E > saved then E:= E − saved;
  if A ≠ 0 then
  for q:= C1[A] while q ≠ 0 do A:= q;
  q:= C1[free cell]; C1[free cell]:= 0; C2[free cell]:= E;
  if A = 0 then C1[L]:= 128 × free cell + C1[L]
  else C1[A]:= free cell;
  COLLECT GARBAGE(0,auxiliary,q)
end;
```

comment

I.4. Retrieval of and enquiries concerning objects stored in C.

Let the value of F be a formula and (C1[F],C2[F]) be a subvalue of the object referred to by the value of F. (C1[F] : 128,C1[F] − C1[F] : 128, C2[F]) is retrieved by a call TYPE(F,A,B) and TYPE obtains the numerical value C1[F] − C1[F] : 128 of the mode of that object in the sense specified on page(6).

In case the mode of that object, referred to by the value of F, is known, we may differentiate, for partial retrieval of the object, with regard to its class of linking complexity, see I.2.1., as follows:

(i) a short integer is partially retrieved by integer procedure VAL OF S INT.
If F = S INT(a), VAL OF S INT(F) = a.
;

```
integer procedure VAL OF S INT(a); value a; integer a;
VAL OF S INT:= C2[if a ≥ saved then a-saved else a];
```

comment

(i) and (ii) a short integer, algebraic variable, sum, product, quotient, rational number or rational function is retrieved by TYPE(F,A,B), as specified.

(iii) and (iv) a long integer or polynomial is partially retrieved by integer procedure GET ARRAY if one needs the values of all value and coefficient fields (see I.2.1.). If one needs the value of one value or coefficient field specified by the value of s (see below) a call of integer procedure ELEMENT(s,F) results in the concerned value being assigned to ELEMENT.
;

```
procedure GET ARRAY(F,i,low,up,Ai); value F,low,up;
integer F,i,low,up,Ai;
begin integer t,A; real B; boolean linked list;
  procedure AI(p); value p; real p;
  if linked list ∧ i < up then
  begin real m; m:= entier(abs(p)/G) × sign(p); Ai:= p - m × G;
    i:= i + 1; Ai:= m
  end
  else Ai:= p;
  t:= TYPE(F,A,B); i:= low;
  linked list:= LINKED LIST(t); AI(B);
  for i:= i + 1 step 1 until up do
  if A > 0 then begin AI(C2[A]); A:= C1[A] end else
  Ai:= if MULTILINKED STRUCTURE(t) then ZERO else 0
end GET ARRAY;

integer procedure ELEMENT(s,X); value s,X; integer s,X;
begin integer type,t,A; real B; boolean linked list;
  type:= TYPE(X,A,B); linked list:= LINKED LIST(type);
  t:= if linked list then 4 else 2;
  if s > t ∧ type ≠ short integer then
```

```
begin for t:= t + 1 while t < s ∧ A ≠ 0 do
  begin if linked list then t:= t + 1; A:= C1[A] end
end;
ELEMENT:= if linked list then
          (if s = 1 then B - entier(abs(B)/G) × sign(B) × G else
          if s = 2 then entier(abs(B)/G) × sign(B) else
          if A ≠ 0 then (if ¬ even(s) then
          C2[A] - entier(abs(C2[A])/G) × sign(C2[A]) × G
          else entier(abs(C2[A])/G) × sign(C2[A])) else 0)
          else
          if s = 1 then B else
          if MULTILINKED STRUCTURE(type) then (if A ≠ 0
          then C2[A] else ZERO) else 0
end ELEMENT;
```

comment

Two complications arise:

(a) The encoding of each two succesive value fields of a long integer in one integer in STORE ARRAY has to be cancelled by a corresponding decoding in GET ARRAY, ELEMENT and integer procedure length (see below).

(b) Efficient arithmetic requires the number of array elements, specified by up - low, up and low being parameters, only to be sufficient and not to be equal to the number of to be assigned values of value or coefficient fields of linked list or multilinked structures.
So in case of a long integer we may supply extra null's and in case of a polynomial (in general multilinked structure) extra ZERO's as values, as, in contradistinction to truncated power series, a polynomial of degree $n > 0$ with ZERO as n - th coefficient equals the polynomial of degree $n - 1$ obtained by deleting its highest coefficient.

The number of array elements necessary and sufficient for retrieving fields of an object referred to by F, specified by n,

in case of a polynomial, by counting the variable first, the coefficients as succeeding elements in order of corresponding and increasing degree, as results from POL, see I.3.,

in case of a long integer, by up - low, up and low being parameters of STORE ARRAY, when it is called in the procedure body of LONG INT,

is the numerical value of integer procedure length(F) upon call.

;

```
integer procedure length(F); value F; integer F;
begin integer t,A,l; boolean linked list; real B;
  t:= TYPE(F,A,B); linked list:= LINKED LIST(t);
  if linked list ∨ MULTILINKED STRUCTURE(t) then
  begin l:= if linked list then 2 else 1;
    for A:= A while A ≠ 0 do
    begin B:= C1[A];
      l:= l + (if linked list ∧ (B ≠ 0 ∨ abs(C2[A]) ≥ G) then 2 else 1);
      A:= B
    end; length:= l
  end
else if t = short integer then length:= 1
else ERROR(true,«type not appropriate in length»);
end length;
```

comment

integer procedure LC determines the sign of the highest coefficient of a polynomial recursively. ISIGN is specified in 2.1.

;

```
integer procedure LC(A); value A; integer A;
LC:= if int(A) then ISIGN(A) else LC(ELEMENT(length(A),A));
```

comment

II. The number system.

II.1. Long integer and short integer arithmetic.

II.1.1. A brief description of the basic integer procedures.

Let the values of X and Y both refer to a long integer or short integer. IABS and ISIGN are counterparts of the ALGOL 60 standard function designators abs and sign, see [10, page 17].
The value of INVERT(X) refers to an object corresponding with the value referred to by X, with sign inverted.
The value of SIGNDIF(X,Y), a procedure auxiliary to IQR, determines the sign of the difference of the values referred to by X and Y.
The values of IPROD(X,Y) and ISUM(X,Y) refer respectively to the long or short integer, that is an object corresponding with the product or sum of the values referred to by their arguments X and Y.
The value of IDIF(X,Y) refers to an object corresponding with the difference of the values referred to by its arguments.
The value of IQR(X,Y,R) refers to the long or short integer that is an object corresponding with the integral quotient of the values referred to by X and Y with remainder referred to by R, leaving the case length(Y) = 1 to IQRS(X,Y,R).
The value of IGCD(X,Y) refers to the long or short integer, that is an object corresponding with the greatest common divisor of the values referred to by the values of its arguments, leaving the case length(X) = 1 or length(Y) = 1 to IGCDS(X,Y).

Sign - convention.

The values of all value fields of a long integer have the same sign.

II.1.2. Declarations and explanations of the basic procedures.

;

```
integer procedure IABS(I); value I; integer I;
IABS:= if ISIGN(I) < 0 then INVERT(I) else I;

integer procedure ISIGN(I); value I; integer I;
ISIGN:= sign(ELEMENT(length(I),I));

integer procedure INVERT(I); value I; integer I;
begin integer l,i; l:= length(I);
  begin integer array B[1:l];
   GET ARRAY(I,1,1,l,B[i]);
   for i:= 1 step 1 until l do B[i]:= -B[i];
   INVERT:= LONG INT(i,1,B[i])
end end;
```

```
integer procedure SIGNDIF(I,J); integer I,J;
begin integer fnn,I1,J1,li,lj;
  fnn:= gnn;
  li:= length(ES(I1,I)); lj:= length(ES(J1,J)); if li > lj then lj:= li;
AA: li:= sign(ELEMENT(lj,I1) - ELEMENT(lj,J1));
  if li = 0 ∧ lj > 1 then begin lj:= lj - 1; goto AA end;
  SIGNDIF:= SR(fnn,li)
end SIGNDIF;

integer procedure ISUM(I,J); integer I,J;
begin integer fnn,I1,J1,li,lj,k,i;
  fnn:= gnn;
  li:= length(ES(I1,I)); lj:= length(ES(J1,J));
  k:= if li > lj then li + 1 else lj + 1;
  begin integer array B,C[1:k];
    GET ARRAY(I1,i,1,k,B[i]);
    GET ARRAY(J1,i,1,k,C[i]);
    ADD(B,C,k); ISUM:= SR(fnn,LONG INT(i,k,B[i]))
  end
end;
```

comment

Adding algorithm.

We describe a simple form of this algorithm, namely for the addition of two nonnegative integers.
Let the contents of integer arrays B,C[1:k - 1] be the value fields of the long or short integers referred to by the values of X and Y (depending on whether k - 1 > 1 or k - 1 = 1).
After termination of this algorithm the values of the value fields of the long or short integer, which represents their sum, will be specified by the contents of B.
B[k] = 0 ∧ C[k] = 0, j and carry are integers.

i: j:= 1 and carry:= 0,
ii: B[j]:= (B[j] + C[j] + carry) mod G and
carry:= entier((B[j] + C[j] + carry)/G) (At each stroke of the addition holds abs(carry) ≤ 1.
This follows inductively from
carry = 0 for j = 0 and abs(B[j] + C[j] + carry) ≤ abs(2G - 2) + abs(carry))),
iii: j:= j + 1, if j < k goto ii otherwise B[k]:= carry and terminate.

ADD is a slightly improved version of an adding procedure due to KRUSEMAN - ARETZ, for the addition of long and short integers, irrespective of their sign.

;

```
procedure ADD(B,C,k); value k; integer k; integer array B,C;
begin integer s,t,w,carry;
AA: for w:= B[k] + C[k] while w = 0 ∧ k > 1 do begin B[k]:= 0;
  k:= k - 1; goto AA end;

comment

The value of w determines the sign of the sum as follows:
The contents of arrays B and C represent respectively the value fields
of the long or short integers corresponding to I = B[1] × G ↑ 0 + ... +
B[k] × G ↑ (k - 1) and J = C[1] × G ↑ 0 + ... + C[k] × G ↑ (k - 1).
After execution of the preceding for statement the following equalities
hold: I = B[1] × G ↑ 0 + ... + B[k0] × G ↑ (k0 - 1) + Rest I and J =
C[1] × G ↑ 0 + ... + C[k0] × G ↑ (k0 + 1) + Rest J,
where Rest I + Rest J = 0, so I + J = (B[1] + C[1]) × G ↑ 0 + ... +
(B[k0] + C[k0]) × G ↑ (k0 + 1).
As w = B[k0] + C[k0], sign(I + J) = sign w, due to,
   if I and J have the same signs, the sign - convention,
   if I and J have opposite signs, the condition
abs((B[1] + C[1]) × G ↑ 0 + ... + (B[k0 - 1] + C[k0 - 1]) × G ↑ (k0 - 2)) ≤
(G - 1) × G ↑ 0 + ... + (G - 1) × G ↑ (k0 - 2) = G ↑ (k0 - 1) - 1 <
G ↑ (k0 - 1).

One needs the value of sign(I + J) in order that the value fields of the
sum, delivered in B, fulfill the sign - convention.
                                                                   ;

  s:= sign(w); carry:= 0;
  for t:= 1 step 1 until k do
  begin w:= B[t] + C[t] + carry;
    if s × w < 0 then begin B[t]:= w + s × G; carry:= -s end
    else if abs(w) > G then begin B[t]:= w - s × G; carry:= s end
    else begin B[t]:= w; carry:= 0 end
  end; if carry ≠ 0 then B[k + 1]:= carry
end;

integer procedure IDIF(I,J); integer I,J;
begin integer fnn,I1,J1,li,lj,k,i;
  fnn:= gnn; ES(I1,I); ES(J1,J); li:= length(I1); lj:= length(J1);
  k:= if li > lj then li + 1 else lj + 1;
  begin integer array B,C[1:k];
    GET ARRAY(I1,i,1,k,B[i]); GET ARRAY(J1,i,1,k,C[i]);
    for i:= 1 step 1 until k do C[i]:= - C[i];
    ADD(B,C,k); IDIF:= SR(fnn,LONG INT(i,k,B[i]))
  end
end;

integer procedure IPROD(I,J); integer I,J;
begin integer fnn,I1,J1,li,lj,l,i;
```

```
  fnn:= gnn; li:= length(ES(I1,I)); lj:= length(ES(J1,J));
  l:= li + lj;
  begin integer array B[1:li],C[1:lj],D[1:l];
    GET ARRAY(I1,i,1,li,B[i]); GET ARRAY(J1,i,1,lj,C[i]);
    MULT(B,li,C,lj,D,l); IPROD:= SR(fnn,LONG INT(i,l,D[i]))
  end
end IPROD;
```

comment

Multiplication algorithm.

We describe a simple version of such an algorithm, for the multiplication of nonnegative integers:

Let the contents of integer arrays I[1 : ki] and J[1 : kj] be the values of the value fields of two long or short integers.
Upon termination of this algorithm their product will be represented by the contents of B[1 : kb], kb = ki + kj.
Let i,j,carry and u be integers.

i: Assign zero to each array element of B and j:= 1,
ii: i:= 1 and carry:= 0,
iii: u:= I[i] × J[j] + B[i + j − 1] + carry, thereafter
B[i + j − 1]:= u mod G and carry:= entier(u/G),
iv: i:= i + 1, if i < ki goto iii else B[i + j]:= carry,
v: j:= j + 1, if j ≤ kj goto ii else terminate.

MULT is a slight improvement of a multiplication procedure for long or short integers, irrespective of their sign, due to KRUSEMAN–ARETZ.

REMARK:
Note that abs(u) < G ↑ 2 and abs(k) < G. This may be proved by induction from abs(I[i] × J[j] + B[i + j − 1] + carry) ≤ (G − 1) × G − 1 + G − 1 + G − 1 < G ↑ 2.

Consequently the size of G in our system has been restricted, as G ↑ 2 −1 must fit in one computer word.

Due to the special properties of the arithmetic implemented on the EL X8 computer used at the Mathematical Centre, it is more efficient on this computer to perform the above multi–length arithmetic by procedures in which reals in stead of integers, in order to perform the arithmetic proper, have been declared.

```
procedure MULT(I,ki,J,kj,B,kb); value ki,kj,kb;
integer ki,kj,kb; integer array I,J,B;
```

```
begin integer ti,tj,tij,carry,Jtj; real u;
  for u:= 1 step 1 until ki do B[u]:= 0;
  for tj:= 1 step 1 until kj do
  begin carry:= 0; Jtj:= J[tj]; for ti:= 1 step 1 until ki do
    begin tij:= ti + tj - 1; u:= Jtj × I[ti] + B[tij] + carry;
      carry:= entier(abs(u)/G) × sign(u); B[tij]:= u - carry × G
    end; B[tj + ki]:= carry
  end
end;
```

comment

Division algorithm.

To divide a positive (n + m) - place integer X by a positive n - place integer Y, we use a generalization for arbitrary radix-G of the common pencil and paper radix - 10 division. This boils down to the repeated integral division of a (n + 1) - place integer u by an n - place integer v, given $0 < u/v < G$, in other words to the computation of entier(u/v). If we make sure that $v[n] \geq G/2$, the digits modulo G of u being represented by u[n + 1],...,u[1] and of v by v[n],...,v[1] (this is realized by multiplying X and Y by the normalization factor entier((G/2)/Y[n])), the theorems, proven below, state that for

$$Q = \min(\text{enter}((u[n+1] \times G + u[n])/v[n]), G - 1)$$

holds $Q \geq \text{entier}(u/v) \geq Q - 2$.

By checking the conditions

$$u[n+1] \times G \uparrow 2 + u[n] \times G + u[n-1] \geq Q \times (v[n] \times G + v[n-1])$$

$$\text{and } u - Q \times v \geq 0$$

the exact value of entier(u/v) can be calculated.

Theorem IQR1. $Q \geq q = \text{entier}(u/v)$.

Proof: This holds for $Q = G - 1$ as $0 < u/v < G$, so assume $Q < G - 1$, then $Q \times v[n] \geq u[n+1] \times G + u[n] - v[n] + 1$ from Q's definition.

$u - Q \times v < u - Q \times v[n] \times G \uparrow (n - 1) < u[n+1] \times G \uparrow n + .. + u[1] - (u[n+1] \times G + u[n] - v[n] + 1) \times G \uparrow (n-1) = u[n-1] \times G \uparrow (n-2) + .. + u[1] - G \uparrow (n-1) + v[n]\, G \uparrow (n-1) < v[n] \times G \uparrow (n-1) \leq v$.
So $u - Q \times v < v \Rightarrow Q \geq q$, QED.

Theorem IQR2. $v[n] \geq \text{entier}(G/2) \Rightarrow q \geq Q - 2$.
Proof: Assume $Q \geq q + 3 \Rightarrow Q \leq (u[n+1] \times G \uparrow n + u[n] \times G \uparrow (n-1))/ v[n] \times G \uparrow (n-1) \leq u/(v[n] \times G \uparrow (n-1)) < u/(v - G \uparrow (n-1))$ (if $v = G \uparrow (n-1)$ then $q = Q$) $\Rightarrow$
$3 \leq X - q < u/(v - G \uparrow (n-1)) - (u/v) + 1 = (u/v)\,(G \uparrow (n-1)/ (v - G \uparrow (n-1)) + 1 \Rightarrow u/v > 2\,(v[n] - 1) \Rightarrow G - 4 \geq Q - 3 \geq q = \text{entier}(u/v) \geq 2(v[n] - 1)$ and $v[n] < \text{entier}\,(G/2)$, QED.

REMARK:
Analogous to the proof of theorem IQR1, given respectively u[n + 1] × G ⋀ 2 + u[n] × G + u[n − 1] <, ≥ Q × (v[n] × G + v[n − 1]), in [7 , page 510, answers to exercises 19 and 20] it is proven that respectively q = entier(u/v) ≤ Q − 1, q = Q or q = Q − 1.

Finally, returning to the (n + m) − place by n − place division, observe u/v < G <==> entier(u/G) < v <==> u[2] × G ⋀ 0 + .. + u[n + 1] × G ⋀ [n−1] < v[1] × G ⋀ 0 + .. + v[n] × G ⋀ [n − 1].
Thus each time the condition for repeated (n + 1) − place by n − place division has been satisfied, as u − q v < v, and at the preliminary steps of the algorithm the following normalization takes place:
If X is represented by its digits modulo G, X[n + m],...,X[1], and Y by Y[n],...,Y[1], set X equal to the integer represented by X[n + m + 1], ...,X[1] with X[n + m + 1] = 0.
As the previously given value of the normalization factor is less or equal to G/2, the top (n + m + 1) − th − digit of the product of normalization factor and X is smaller than G/2 and positive and the top n − th − digit of the product of normalization factor and Y is greater than or equal to G/2.

;

```
integer procedure IQR(X1,Y1,R); integer X1,Y1,R;
begin integer X,Y,lX,lY,lQ,fnn; fnn:= gnn;
  lX:= length(ES(X,X1)) + 1; lY:= length(ES(Y,Y1)); lQ:= lX − lY;
  if EQ(X,ZERO) ∨ EQ(Y,ZERO) ∨ lX < lY then
  begin comment IQR:= ZERO if X1 or Y1 equals ZERO or length(X1) < length(Y1)
    else IQR:= IQRS(X1,Y1,R) if length(Y1) = 1, else goto Next comment
                                                                    ;
    IQR:= ZERO; R:= X − saved; if EQ(Y,ZERO) then
    begin PR nlcr; PR string(⌈Y equals ZERO in IQR⌉); PR nlcr end
  end else
  if lY = 1 then IQR:= IQRS(X,Y,R) else
  begin integer s,i,j,normfactor,VGmin1,q,Q,q1,Q1,HeadY,
    VYY1Y,YY1Y,lb,dummy;
    integer array XX[1:lX],YY[1:lY],QQ[1:lQ];
    DE(q,0,DE(q1,0,0)); s:= ISIGN(X) × ISIGN(Y);
    YY1Y:= abs(ELEMENT(lY,Y));
 L1: ES(normfactor,S INT(if YY1Y × (G : (2 × YY1Y)) = G : 2 then
    G : (2 × YY1Y) else G : (2 × YY1Y) ∓ 1));
 L2: GET ARRAY(IABS(IPROD(X,normfactor)),i,1,lX,XX[i]);
    GET ARRAY(ES(Y,IABS(IPROD(Y,normfactor))),i,1,lY,YY[i]);
    ES(VGmin1,S INT(Gmin1)); YY1Y:= YY[lY];
    ES(HeadY,LONG INT(i,2,if i = 1 then YY[lY − 1] else YY1Y));
    ES(VYY1Y,S INT(YY1Y)); lb:= lY ∓ 1;
 L3: for j:= lX step −1 until lb do

 L4: begin comment Next comment: (labels correspond to labels in the
     declaration of IQR)
```

L1: The short integer referring to the normalization factor, ceiling – of ((G/2)/YY1Y), is constructed, saved and assigned to normfactor.
L2: X and Y are normalized by multiplication with norm factor and taking absolute values.
L3: The integral division entier((XX[j – lY] × G ↑ (j – lY – 1) + ... + XX[j] × G ↑ (j – 1))/(YY[1] × G ↑ 0 + ... + YY[lY] × G ↑ (lY – 1))), for j:= lX,lX – 1,..,lY + 1, is performed.
;

```
   Q:= if XX[j] > YY1Y then ASSIGN(q,VGmin1) else
      ASSIGN(q,IQRS(LONG INT(i,2,XX[j – 2 + i]),VYY1Y,dummy));
L5: L: if SIGNDIF(LONG INT(i,3,XX[j – 3 + i]),IPROD(Q,HeadY)) = –1 then
      begin Q:= ASSIGN(q,IDIF(Q,ONE)); goto L end;
      Q1:= ASSIGN(q1,IDIF(LONG INT(i,lb,XX[j – lb + i]),
      IPROD(Q,Y)));
L6: if ISIGN(Q1) = –1 then
      begin Q:= ASSIGN(q,IDIF(Q,ONE)); Q1:= ASSIGN(q1,ISUM(Q1,Y)) end;
L7: GET ARRAY(Q1,i,j – lY,j,XX[i]);
L8: QQ[j – lY]:= VAL OF S INT(Q)
   end;
   ES(R,IQRS(LONG INT(i,lY,s × XX[i]),normfactor,dummy));
   IQR:= SR(fnn,LONG INT(i,lQ,s × QQ[i])); R:= R – saved
  end
end;
```

comment

L4: The previously mentioned first approximation to the integral value of the fraction in L3, min(entier(XX[j] × G + XX[j – 1])/YY1Y),G – 1), is calculated.
L5: The exact integral value is determined by first checking whether XX[j] × G ↑ 2 + XX[j – 1] × G + XX[j – 2] – V(Q) × (YY1Y × G + YY [lY – 1]) < 0, if so,the approximation is at least one to large, and
L6: finally checking whether
XX[j] × G ↑ lY + .. + XX[j – lY] – V(Q) × (YY[1] × G ↑ 0 + .. + YY[lY] × G ↑ (lY – 1)) < 0,
if so, the approximation is exactly one to large.
For the sufficiency of these checks see theorems IQR1 and IQR2 and Remark. Their necessity has been shown in [7].
L7: Analogous to the pencil and paper method of division the dividend receives its new value and
L8: the quotient digit is assigned to an array element specifying a value field of the quotient.
;

```
integer procedure IQRS(X1,Y,R); integer X1,Y,R;
begin integer X,fnn; boolean bYZERO,bYMINONE; fnn:= gnn;
   bYZERO:= EQ(Y,ZERO); bYMINONE:= EQ(Y,MINONE);
   if EQ(ES(X,X1),ZERO) ∨ bYZERO ∨ EQ(Y,ONE) ∨ bYMINONE then
```

```
  begin IQRS:= if bYZERO then ZERO else if bYMINONE then
    INVERT(X) else X - saved;
    if bYZERO then r:= X - saved
    else R:= ZERO
  end else
  begin integer lX; lX:= length(X);
    begin integer s,y,i,n; integer array XX[1:lX],r[0:lX]; real m;
      s:= ISIGN(X);
      GET ARRAY(if s > 0 then X else INVERT(X),i,1,lX,XX[i]);
      r[0]:= r[lX]:= 0;
      y:= VAL OF S INT(Y); s:= sign(y) × s; y:= abs(y);
      for i:= lX step -1 until 1 do
      begin m:= r[i] × G + XX[i]; XX[i]:= n:= entier(abs(m/y)) × sign(m/y);
        r[i - 1]:= m - (n × y)
      end;
      R:= EV(S INT(s × r[0])) - saved;
      IQRS:= LONG INT(i,lX,s × XX[i])
    end
  end;
  ERASE(fnn)
end IQRS;
```

comment

IQRS is an auxiliary procedure. It is called upon in IQR, where name replacement of X1 by the name of a possibly not saved object and name replacement of X1 and Y1, names of already in IQR saved objects, occurs, and in IGCDS, where again name replacement by names of saved objects occurs. So only X1 needs to be saved in IQRS.

;

comment

Greatest common divisor algorithm for multiple length integers due to LEHMER.

LEHMER observed [American Mathematical Monthly 45(1938) p 227-233] that in using a multiple precision version of Euclides' famous algorithm, the multiple precision steps to determine Q (such that U = QV + R with abs(R) < V) were often superfluous, in the sense that the same Q might have been determined by single precision arithmetic.

Let in the radix G representation of U and V, lU be the number of digits of U, lV be the number of digits of V, u be the leading digit of U and v be the leading digit of V.

2: A:= 1, B:= 0, C:= 0, D:= 1.

i: $(u + B) \times G \uparrow (lU - 1) \leq U \leq (u + A) \times G \uparrow (lU - 1)$ and

ii: $(v + C) \times G \uparrow (lV - 1) \leq V \leq (v + D) \times G \uparrow (lV - 1)$ obviously hold.

iii: 3: If lU = lV and

iv: 5: If $v + C \neq 0 \wedge v + D \neq 0$,

v: $\text{entier}((u + B)/(v + C)) \leq \text{entier}(T/S) \leq \text{entier}((u + A)/(v + C))$,

for $T = A \times U + B \times V$ and $S = C \times U + D \times V$ by i, ii, iii, iv.
Q:= entier((u + A)/(v + C)).
The single precision calculation of entier(U/V) is possible by iii, if in v

entier((u + B)/(v + C)) = entier((u + A)/(v + C)) as U = T and V = S.

If so, straightforward calculation shows

$$(v + C)/((u - Q \times v) + (A - Q \times C)) \leq V/(U - Q \times V) \leq (v + D)/((u - Q \times v) + (B - Q \times D)),$$

which amounts to v after the following assignements have been performed from left to right:

6: T:= A − Q × C, A:= C, C:= T, T:= S − Q × D, B:= D, D:= T,
T:= u − Q × v, u:= v, v:= T,

7: Else, perform multiple precision calculation to determine Q.

;

<u>comment</u>

This version of LEHMER's algorithm, IGCD, incorporates a trick due to COLLINS (see his Revised SAC − I integer system). He observed, that, if lU − lV = 1, still single precision simulation might be possible, if multiplying both U and V by the same factor, would result in answers of the same length of digits. To avoid multiple precision multiplication he introduces the following simplication (if (lU − lV) ≤ 1):

3: Assign to u2 and u1 the two top digits of U and, if lU = lV, to v2 and v1 the towo top digits of V else, if lU − lV = 1, to v2 zero and to v1 V's top digit,

4: normfactor:= entier((G/2)/u2) and multiply the long integers represented by (u2,u1) and (v2,v1) by normfactor, in order to
5: check if now the lengths of the results of these multiplications are equal.

REMARK:
Arabic numbered labels in the two comments above correspond to labels in IGCD.

;

```
integer procedure IGCD(X,Y); integer X,Y;
begin integer fnn,U,V1,u,v,lU,lV;
  fnn:= DE(u,IABS(X),DE(v,IABS(Y),gnn)); U:= V(u); V1:= V(v);
  lU:= length(U); lV:= length(V1);
  if lU = 1 ∨ lV = 1 then begin IGCD:= IGCDS(U,V1); goto ENDIGCD end;
  begin integer normfactor,Normfactor,sd,u1,u2,v1,v2,s,t,T,i,A,B,C,D,Q;
    DE(s,0,DE(t,0,DE(normfactor,0,0)));
    sd:= SIGNDIF(U,V1); A:= U;
    U:= ASSIGN(u,if sd > 0 then U else V1);
    V1:= ASSIGN(v,if sd > 0 then V1 else A);
    if sd < 0 then begin A:= lU; lU:= lV; lV:= A end;
    if EQ(U,V1) then begin IGCD:= U - saved; goto ENDIGCD end else
  loop: if EQ(V1,ZERO) then begin IGCD:= U - saved; goto ENDIGCD end else
  2: if lV = 1 then begin IGCD:= IGCDS(U,V1); goto ENDIGCD end else
    begin A:= D:= 1; B:= C:= 0;
    3: if lU - lV  < 1 then
      begin u1:= ELEMENT(lU - 1,U); u2:= ELEMENT(lU,U);
        if lU - lV = 1 then
        begin v1:= ELEMENT(lV,V1); v2:= 0 end else
        begin v1:= ELEMENT(lV - 1,V1); v2:= ELEMENT(lV,V1) end;
      4: Normfactor:= ASSIGN(normfactor,S INT(if u2 × (G : (2 × u2)) = G : 2 then
                                          G : (2 × u2) else (G : (2 × u2)) + 1));
        u2:= ELEMENT(2,IPROD(Normfactor,LONG INT(i,2,if i = 1 then u1 else u2)));
        v2:= ELEMENT(2,IPROD(Normfactor,LONG INT(i,2,if i = 1 then v1 else v2)));
      5: if v2 + C = 0 ∨ v2 + D = 0 then goto 7;
        Q:= (u2 + A) : (v2 + C); if Q ≠ (u2 + B) : (v2 + D) then goto 7;
      6: T:= A - (Q × C); A:= C; C:= T; T:= B -(Q × D); B:= D; D:= T;
        T:= u2 - (Q × v2); u2:= v2; v2:= T; goto 5
      end else
    7: if B = 0 ∨ (lU - lV) > 1 then
      begin IQR(U,V1,i); U:= ASSIGN(u,V1); V1:= ASSIGN(v,i) end else
      begin U:= ASSIGN(u,ISUM(IPROD(S INT(A),U),IPROD(S INT(B),V1)));
        V1:= ASSIGN(v,ISUM(IPROD(S INT(C),U),IPROD(S INT(D),V1)));
      end;
      lU:= length(U); lV:= length(V1); goto loop
    end
  end;
  ENDIGCD: ERASE(fnn)
end IGCD;

integer procedure IGCDS(X,Y); value X,Y; integer X,Y;
begin integer procedure gcd(a,b); value a,b; integer a,b;
  gcd:= if b = 0 then abs(a) else gcd(b,a - ((a : b) × b));
  integer fnn,R; fnn:= gnn;
  if SIGNDIF(ES(X,IABS(X)),ES(Y,IABS(Y))) < 0 then
  begin R:= X; X:= Y; Y:= R end; IQRS(X,Y,R);
  IGCDS:= SR(fnn,S INT(gcd(VAL OF S INT(Y),VAL OF S INT(R))))
end;
```

<u>comment</u>

II.2. The rational number system.

Before proceeding with the discussion of representation of and operations on rational numbers, it should be realized that in this rational function system all operations are unified in the integer procedures S, P and Q. A sum, product or quotient respectively of two arbitrary objects A and B is constructed by a call of S(A,B),P(A,B) and Q(A,B), respectively. In the sequel we assume that their functions are known. A full treatment will be found in the sections corresponding to the relevant modes and in chapter IV.

II.2.1. Representation of a rational number.

Given two long or short integers referred to by the values of A and B, store the rational number, represented by the pair(A,B)(thinking in terms of equivalence classes) as:

i: ZERO, if EQ(A,ZERO) or EQ(B,ZERO)(one is noticed by the system that the latter case occurs by the procedure statement PR string(≮B equals ZERO in Q≯) in the procedure body of Q),

ii: A if EQ(B,ONE) or as P(A,MINONE) if EQ(B,MINONE),

iii: STORE(A,rational number,B) if the integers referred to by A and B are relativity prime and ISIGN(B) positive and else, if ISIGN(B) negative, as

iv: STORE(INVERT(A),rational number,INVERT(B)),

else the greatest common divisor of the values referred to by A and B is calculated by means of IGCD(A,B) referring to a nonnegative integer). By dividing by this integer a relatively prime pair(A1,B1) is constructed and stored as a rational number according to ii,iii or iv.

REMARK: The condition that the value of ISIGN(B) is positive, when storing a rational number, is dictated by the use of EQ.

<u>How are objects of mode rational number, built up from long or short integers as above, introduced in the system?</u>

In the first place by integer procedure Q(A,B). Reading Q (see chapter IV), it is clear to take care of i and ii above. Q calls upon OPER ON NUM(quotient,A,B)(see II.2.) however, to treat case iii and iv, so upon RNPROD(A,ES(B,RINV(B))). After elaboration of ES(B,RINV(B)), B represents the inverse of B(so the original pair(ONE,B)), as follows from the declaration of integer procedure RINV:

```
integer procedure RINV(A1); integer A1;
begin integer A,t,l,fnn; real r; fnn:= gnn; t:= TYPE(ES(A,A1),l,r);
  RINV:= SR(fnn,if t = short integer ∨ t = long integer then
                 (if ISIGN(A) > 0 then STORE(ONE,rational number,A) else
                  STORE(MINONE,rational number,INVERT(A))) else
                 if t = rational number then
                 (if EQ(l,ONE) ∨ EQ(l,MINONE) then P(r,l) else
                   if ISIGN(l) > 0 then STORE(r,rational number,l) else
                    STORE(EV(INVERT(r)),rational number, INVERT(l))) else
                 if t = polynomial then
                 (if LC(A) > 0 then STORE(ONE,rational function,A) else
                   STORE(MINONE,rational function,P(MINONE,A))) else
                 if EQ(l,ONE) ∨ EQ(l,MINONE) then P(r,l) else
                 if LC(l) > 0 then STORE(r,rational function,l) else
                  STORE(EV(P(r,MINONE)),rational function,P(l,MINONE)))
end;
```

comment

If the value of A, a parameter of RNPROD, does not refer to a rational number, the values possessed by A and B are interchanged in the labelled conditional statement, resulting in the value of the "original" B equalling the value of rA and the value of the present B equalling the value of A after elaboration of this statement. If upon call of RNPROD the value of A does not refer to a rational number, RNPROD proceeds with elaboration of ¬ EQ(ES(Gcd,IGCD(rA,B)), ONE), which amounts to answering the question:
"Does the value of Gcd, having been assigned the saved value of an integer representing the greatest common divisor of rA and B, equal the value of ONE or not?", so, the question of relative primeness of the original pair(A,B). If so, condition iii or iv has been fulfilled, else ES(rA,IQR(rA,Gcd,di)) and ES(B,IQR(B,Gcd,di)) result in a relatively prime pair(B,rA) in the same equivalence class as the pair(A,B) we started with (lA being ONE).

Finally notice that, by its last assignment RNPROD, so OPER ON NUM2, so Q, receives its value ST rat(V(l),V(r)). This amounts in our case to storing according to ii, iii or iv (see the beginning of this section).
;

```
integer procedure ST rat(A1,B1); integer A1,B1;
begin integer fnn,A,tA,B,tB; fnn:= gnn;
  tA:= TYPE(ES(A,A1),di,dii); tB:= TYPE(ES(B,B1),di,dii);
  ST rat:= SR(fnn,if EQ(B,ONE) ∨ EQ(B,MINONE) then P(A,B) else
  if EQ(A,ZERO) ∨ EQ(B,ZERO) then ZERO else
  if(tA = short integer ∨ tA = long integer) ∧ (tB = short integer ∨ tB
  = long integer) then
  (if ISIGN(B) > 0 then STORE(A,rational number,B) else
  STORE(EV(INVERT(A)),rational number,INVERT(B))) else
  if LC(B) > 0 then STORE(A,rational function,B) else
  STORE(EV(P(MINONE,A)),rational function,P(MINONE,B)))
end;
```

comment

II.2.2. Operations with rational numbers.

As in the previous section, calling S, P and Q, with names referring to short or long integers or rational numbers as arguments, boils down, except for trivial cases, to calling OPER ON NUM2 the appropriate operation being specified in its first argument. As the values of the second and third parameter refer to saved numbers, it is justified to put A and B in OPER ON NUM2's value list.

;

```
integer procedure OPER ON NUM2(oper,A,B,tA,tB); value oper,A,B,tA,tB;
integer oper,A,B,tA,tB;
begin integer fnn; fnn:= gnn;
  OPER ON NUM2:= SR(fnn,if (tA = short integer ∨
                            tA = long integer) ∧
                           (tB = short integer ∨
                            tB = long integer) then
  (if oper = sum then ISUM(A,B) else
  if oper = product then IPROD(A,B) else
  RNPROD(A,EV(RINV(B)))) else
  if oper = sum then RNSUM(A,B) else
  if oper = product then RNPROD(A,B) else
  RNPROD(A,EV(RINV(B))))
end;
```

comment

Of the integer procedures called upon in OPER ON NUM2, RNSUM and RNPROD remain to be discussed.
RNSUM performs addition of two numbers, one of which at least is a rational number, and delivers the name of the result as its value, while RNPROD performs multiplication in an analogous fashion.
Since calculating the greatest common divisor of two integers is a very time consuming process, one needs algorithms, which minimize both the number of times IGCD is called upon and the length of its arguments.
We cite and use a modification of those used by BROWN in the ALPAK system for addition and multiplication as described in COLLINS' SAC-1 rational function system.
People with a preference for making use of the full expressional power of ALGOL 60 and with a tendency to think in ALGOL 68 terms will be shown afterwards how OPER ON NUM2,RNSUM and RNPROD can be compressed in a few, although very lengthy, statements. They will be explained in section III.2. by means of an ALGOL 68 declaration.
A consistent description of RNSUM and RNPROD, which equals the following comments upon RNSUM and RNPROD in clarity of description, is contained in the ALGOL 68 identity declaration of OPER ON RAT in section III.2.

T = RNSUM(A,B). Assume A = lA/rA, B = lB/rB, where gcd(lA,rA) = 1 and gcd(lB,rB) = 1. Gcd:= gcd(rA,rB).
If Gcd = 1 then lT:= lA × rB + rA × lB. rT:= rA × rB.
It follows from Gcd = 1, that gcd(lT,rT) = 1.
If Gcd ≠ 1, rA1:= rA/Gcd, rB1:= rB/Gcd and lT:= lA × rB1 + lB ×rA1.
rT:= rA × rB1. Next, Gcd:= gcd(rT,Gcd).
If Gcd = 1 then T:= lT/rT else lT:= lT/Gcd,rT:= rT/Gcd and T:= lT/rT.
Notice that ISIGN(rT) > 0.

;

```
integer procedure RNSUM(A,B); value A,B; integer A,B;
begin integer tA,tB,lA,lB,l,r,fnn; real rA,rB; fnn:= DE(l,0,DE(r,0,gnn));
  tA:= TYPE(A,lA,rA);
  if tA ≠ rational number then begin tB:= TYPE(B,lA,rA); B:= A end
  else tB:= TYPE(B,lB,rB);
  if tA = rational number ∧ tB = rational number then
  begin integer Gcd; if ¬ EQ(ES(Gcd,IGCD(rA,rB)),ONE) then
    begin ASSIGN(l,ISUM(IPROD(lA,ES(rB,IQR(rB,Gcd,di))),
                   IPROD(lB,IQR(rA,Gcd,di))));
      if ¬ EQ(ES(Gcd,IGCD(V(l),ASSIGN(r,IPROD(rA,rB)))),ONE) then
      begin ASSIGN(l,IQR(V(l),Gcd,di)); ASSIGN(r,IQR(V(r),Gcd,di)) end
    end else
    begin ASSIGN(l,ISUM(IPROD(lA,rB),IPROD(lB,rA)));
      ASSIGN(r,IPROD(rA,rB))
    end
  end else
  begin ASSIGN(l,ISUM(lA,IPROD(rA,B))); ASSIGN(r,rA)
  end;
  RNSUM:= SR(fnn,ST rat(V(l),V(r))); END:
end RNSUM;
```

comment

T = RNPROD(A,B). Assume A = lA/rA, B = lB/rB, where gcd(lA,rA) = 1 and gcd(lB,rB) = 1. Gcd1:= gcd(lA,rB) and Gcd2:= gcd(rA,lB). Then lA:= lA/Gcd1, rB:= rB/Gcd1, rA:= rA/Gcd2,lB:= lB/Gcd2, except if A = ONE and B = ONE.
lT:= lA × lB and rT:= rA × rB. Finally T:= lT/rT.
Notice that ISIGN(rT) > 0.

;

```
integer procedure RNPROD(A,B); value A,B; integer A,B;
begin integer tA,tB,lA,lB,l,r,fnn; real rA,rB; fnn:= DE(l,0,DE(r,0,gnn));
  tA:= TYPE(A,lA,rA);
L: if tA ≠ rational number then begin tB:= TYPE(B,lA,rA); B:= A end
  else tB:= TYPE(B,lB,rB); if tA = rational number
  ∧ tB = rational number then
  begin integer Gcd1,Gcd2; if ¬ EQ(ES(Gcd1,IGCD(rA,lB)),ONE) then
    begin ES(rA,IQR(rA,Gcd1,di)); ES(lB,IQR(lB,Gcd1,di)) end;
    if ¬ EQ(ES(Gcd2,IGCD(rB,lA)),ONE) then
```

```
    begin ES(lA,IQR(lA,Gcd2,di)); ES(rB,IQR(rB,Gcd2,di)) end;
    ASSIGN(l,IPROD(lA,lB)); ASSIGN(r,IPROD(rA,rB))
  end else
  begin integer Gcd; if ¬ EQ(ES(Gcd,IGCD(rA,B)),ONE) then
    begin ES(rA,IQR(rA,Gcd,di)); ES(B,IQR(B,Gcd,di)) end;
    ASSIGN(l,IPROD(lA,B)); ASSIGN(r,rA)
  end;
  RNPROD:= SR(fnn,ST rat(V(l),V(r)))
end RNPROD;

integer procedure IQI(X,Y); integer X,Y; IQI:= IQR(X,Y,dii);

integer procedure OPER ON NUM(oper,A,B,tA,tB);
value oper,A,B,tA,tB; integer oper,A,B,tA,tB;
begin integer fnn,lA,lB,Gcd,l,r; real rA,rB; fnn:= gnn;
    DE(l,O,DE(r,O,O));
  TYPE(A,lA,rA); if oper = quotient then
  begin tB:= TYPE(ES(B,RINV(B)),lB,rB); oper:= product end
  else TYPE(B,lB,rB);
  if(tA = long integer ∨ tA = short integer) ∧
    (tB = long integer ∨ tB = short integer)
  then OPER ON NUM:= SR(fnn, if oper = sum then ISUM(A,B) else
                        IPROD(A,B)) else
  begin if tA ≠ rational number then
  begin lA:= lB; rA:= rB; B:= A end;
    OPER ON NUM:=
    SR(fnn,if tA = rational number ∧ tB = rational number then
           (if oper = sum then
            (if ¬ EQ(ES(Gcd,IGCD(rA,rB)),ONE) then
             (if ¬ EQ(ES(Gcd,
                         IGCD(ASSIGN(l,
                                     ISUM(IPROD(lA,
                                                ES(rB,
                                                   IQI(rB,Gcd)
                                                )  ),
                                          IPROD(lB,IQI(rA,Gcd)
                                     )    )     ),
                              ASSIGN(r,IPROD(rA,rB))
                         )    ),ONE
                    ) then ST rat(IQI(V(l),Gcd),IQI(V(r),Gcd))
                      else ST rat(V(l),V(r))
             ) else ST rat(ISUM(IPROD(lA,rB),IPROD(lB,rA)),IPROD(rA,rB))
           ) else
           ST rat(IPROD(if ¬ EQ(ES(Gcd,IGCD(lA,rB)),ONE) then
                        Multiple ES(rB,IQI(rB,Gcd),
                                    ES(lA,IQI(lA,Gcd))
                                   )
                        else lA,
                        if ¬ EQ(ES(Gcd,IGCD(lB,rA)),ONE) then
```

```
                        Multiple ES(rA,IQI(rA,Gcd),
                                 ES(lB,IQI(lB,Gcd))
                                )
                        else lB
                       ),IPROD(rA,rB)
         )       ) else
         if oper = sum then ST rat(ISUM(lA,IPROD(rA,B)),rA) else
         ST rat(IPROD(if ┐EQ(ES(Gcd,IGCD(rA,B)),ONE) then
                       Multiple ES(rA,IQI(rA,Gcd),IQI(B,Gcd))
                       else B,
                       lA
                      ),
                  rA
      )          )
end end;
```

comment

III. The rational function system.

III.1. Polynomial arithmetic.

III.1.1. Objects having the same hierarchy of variables.

"Take, for instance, the possible fat man in that doorway.
And, again, the possible bald man in that doorway.
Are they the same possible man, or two possible men?"
From a logical point of view, W.V.O.Quine.

$(1 \times x \uparrow 0 + 1 \times x \uparrow 1) \underline{\times} (1 \times y \uparrow 0 + 1 \times y \uparrow 1),$
is undefined in this system.

$(10 \times x \uparrow 0 + 10 \times x \uparrow 1) \underline{/} (10),$
is defined in this system.

$(1 \times x \uparrow 0 + 1 \times x \uparrow 1) \underline{+} ((1 \times y \uparrow 0 + 1 \times y \uparrow 1) \times x \uparrow 0 + (1 \times y \uparrow 0 + 1 \times y \uparrow 1) \overline{\times} x \uparrow 1),$
is defined in this system.

$(1 \times x \uparrow 0 + 1 \times x \uparrow 1) \times y \uparrow 0 + (1 \times x \uparrow 0 + 1 \times x \uparrow 1) \times y \uparrow 1 = (1 \times y \uparrow 0 + 1 \times y \uparrow 1) \times x \uparrow 0 + (1 \times y \uparrow 0 + 1 \times y \uparrow 0) \times x \uparrow 1,$
is defined in this system and false.

These expressions can be transformed into function designators by

a: modifying applications of ×,/,_, and = into Polish prefix notation by prefixing P,PQT,S and EQ,
b: replacing the coefficients 1 and 10 by ONE S INT(10), respectively, and
c: replacing expressions like coef[0] × x ↑ 0 + coef[1] × x ↑ 1 by POL(i,1,AV(24),coef[i]) – x is the 24 – th letter of the alfabet.

All polynomials in this system, on which arithmetical operations are performed, are represented in recursive canonical form. This terminology has been derived from [4] and refers to the fact that a polynomial in n variables is always regarded as a polynomial in one variable(called the main variable), whose coefficients are themselves objects, at least one of which is a polynomial in n–1 variables, having the same hierarchy of variables, a terminology to be defined below.
This implies an assumed ordering of the variables of any polynomial. Whenever we write p(x[1],...,x[n]), displaying the variables of p, the intention is to specify this ordering, x[n] being the main variable, x[n–1] being the main variable of those coefficients of p, that are not long or short integer, etc..

Two objects have the same hierarchy of variables, a terminology derived from [12, page 40], in case they are

- i: polynomials with the same main variable and with coefficients having the same hierarchy of variables(i.e. every pair has the same hierarchy of variables),
- ii: a polynomial and a long or short integer,
- iii: long or short integers.

From ii, iii and the word "variables" in the term defined above, we might regard a long or short integer as a polynomial, provided the latter term is taken in a wider sense than defined in I.2.1.
From the point of view of the ALGOL 60 procedure declarations of the polynomial arithmetic performing procedures, described in this section, there is, however, a substantial difference, as we have to differentiate according to the mode being either polynomial or long integer or short integer.

Let p be a polynomial of degree d, in n variables,
$p(x[1],..,x[n]) = p[0] \times x[n] \uparrow 0 + \ldots + p[d] \times x[n] \uparrow d$,
with p[0],p[1],..,p[d] objects having the same hierarchy of variables. Let q be a long integer or a short integer.
To add p to or multiply p with q, in [4, the description of PORDER page 26,27 and the arithmetic performing procedures] COLLINS constructs an auxiliary version of q, $(\ldots((q \times x[0] \uparrow 0) \times x[1] \uparrow 0) \times \ldots \times x[n] \uparrow 0)$, and then adds or multiplies by adding or multiplying the coefficients of degree zero recursively.
His point of view, that an infinite precision integer is a polynomial of degree zero, explained in section I.2.1., entails this.
In the footsteps of VAN DE RIET[12] we do not wish to introduce in our system such versions of q as $(\ldots(q \times x[0] \uparrow 0) \times \ldots \times x[n] \uparrow 0)$, for, by introducing them, the unique representation of a long integer or short integer is lost and awkward questions concerning the equality of e.g. q, $(\ldots(q \times x[n] \uparrow 0) \times .. x[0] \uparrow 0)$ and $(\ldots(q \times x[0] \uparrow 0) \times \ldots \times x[n] \uparrow 0)$ have to be raised and answered.
Addition, in this system, of a non - polynomial q to a polynomial p is performed in a recursive way by adding q to p[0], until the process ends with the addition of q to a long or short integer, without introducing auxiliary versions, as above.
The importance of the requirement that two objects p1 and p2 have the same hierarchy of variables is, that arithmetical operations, without introducing the afore - mentioned vacuous occurrences of variables, can only be performed between p1 and p2 if they fulfill this requirement.

III.1.2. A brief description of the basic integer procedures, that perform polynomial arithmetic.

Let the values of X and Y refer to objects with the same hierarchy of variables. If at least one of the values of X and Y refers to a polynomial, the value of OPER ON POL(oper,X,Y,tX,xX,tY,yY) refers to the polynomial or integer(long or short), which is an object corresponding with the sum or product of the objects referred to by the values of X and Y, depending on whether oper = sum or oper = product(oper is of type integer).
Let the value of X refer to a multiple of the value, that Y refers to.
The value of PQI(X,Y) refers to the unique object such that the value of EQ(P(PQI(X,Y),Y),X) is true.
The value of PGCD(X,Y) refers to an object corresponding with the greatest common divisor of the objects referred to by the values of X and Y.
The integer procedures PGCDS,PSREM,PCONT and Product are auxiliary to PGCD.

III.1.3. Declaration and description of the basic polynomial arithmetic performing procedures.

Let the value of X refer to an object corresponding with coefX[n] × x ⋀ n + .. + coefX[0] × x ⋀ 0, with coefX[0],..,coefX[n] having the same hierarchy of variables, let Y analogously refer to an instance of coefY[n] × x ⋀ m + .. + coefY[0] × x ⋀ 0, Y ≠ ZERO, and n = max(n,m).
The value of OPER ON POL(sum,X,Y) refers to an object corresponding with coef[k] × x ⋀ k + .. + coef[0] × x ⋀ 0, with k = n if n > m, else, if n = m with k the maximal nonnegative integer j bounded by n, such that coefX[j] + coefY[j] ≠ 0, if such an integer exists, else the value of OPER ON POL is the value of ZERO.
The value of OPER ON POL(product,X,Y) refers to an object corresponding with the Cauchy product of two polynomials, of degree n + m.

;

```
integer procedure OPER ON POL(oper,PP,QQ,tp,xp,tq,xq);
value oper,PP,QQ,tp,tq,xp,xq; integer oper,PP,QQ,tp,tq,xp,xq;
begin integer dp,dq,d,i,j,fnn; fnn:= gnn;
  if tp ≠ polynomial then begin ES(PP,STORE ARRAY(i,0,1,
  polynomial,if i = 0 then xq else PP)); xp:= xq end;
  if tq ≠ polynomial then begin ES(QQ,STORE ARRAY(i,0,1,
  polynomial,if i = 0 then xp else QQ)); xq:= xp end;
  dp:= length(PP) - 2; dq:= length(QQ) - 2;
  d:= if dp < dq then dq else dp;
  d:= if oper = product then dp + dq else if dp < dq then dq else dp;
  begin integer array Cp[-1:dp],Cq[-1:dq],C[-1:d];
    GET ARRAY(PP,i,-1,dp,Cp[i]); GET ARRAY(QQ,i,-1,dq,Cq[i]);
    ERROR(xp ≠ xq,⊀variables not the same in OPER ON POL⊁);
    OPER ON POL:= SR(fnn,POL(i,d,xp,
      if oper = sum then
      (if i > dp then Cq[i] else
        if i > dq then Cp[i] else S(Cp[i],Cq[i]))
      else if oper = product then
      Sum(j,0,i,P(if j < dp then Cp[j] else ZERO,
                 if i-j < dq then Cq[i-j] else ZERO))
```

```
      else ERROR(true,⊀type in OPER ON POL not appropriate⊁)))
end end OPER ON POL;

integer procedure Sum(i,low,up,Fi); value low,up;
integer low,up,i,Fi;
begin integer s,fnn; fnn:= gnn; DE(s,ZERO,0);
  for i:= low step 1 until up do ASSIGN(s,S(V(s),Fi));
  Sum:= SR(fnn,V(s) - saved)
end Sum;

comment

    1:  To obtain the value of PQI(X,Y) if coefY[0] = 0, we divide
        both polynomials by the polynomial corresponding to x ⋀ j,
        with j the least nonnegative integer such that coefY[j] ≠ 0,
        and proceed with applying ii on (X/(x ⋀ j))/(Y/(x ⋀ j)),
        otherwise
    2:  the value of PQI(X,Y) refers to an object corresponding with the
        polynomial with coefficients as described by
        coef[j] = (coefX[j] - (coef[0] × coefY[j - 1] + ... +
        coef[j - 1] × coefY[0]))/coefY[0].
                                                                    ;

integer procedure PQI(X1,Y1); integer X1,Y1;
begin integer fnn,x,X,y,Y,lX,lY,tX,tY;
  fnn:= gnn; DE(x,X1,DE(y,Y1,0)); X:= V(x); Y:= V(y);
  tX:= TYPE(X,di,dii); tY:= TYPE(Y,di,dii);
  if EQ(X,ZERO) ∨ EQ(Y,ZERO) then PQI:= ZERO else
  if EQ(Y,ONE) then PQI:= X else
  if(tX = short integer ∨ tX = long integer) ∧ (tY = short
  integer ∨ tY = long integer) then PQI:= IQR(X,Y,lY) else
  if tX ≠ polynomial then PQI:= ZERO else
  begin lX:= length(X) - 2; lY:= length(Y) - 2;
    begin integer array coefX[-1:lX]; integer i,varX;
      GET ARRAY(X,i,-1,lX,coefX[i]); varX:= coefX[-1];
      if tY = polynomial ∧ lX > lY then
     begin integer array coefY[-1:lY]; integer trivdivX,
        trivdivY; boolean bool;
        GET ARRAY(Y,i,-1,lY,coefY[i]); bool:= true;
        trivdivX:= trivdivY:= 0; i:= -1;
        for i:= i + 1 while bool ∧ i < lX do
        if EQ(coefX[i],ZERO) then trivdivX:= trivdivX + 1
        else bool:= false; bool:= true; i:= -1;
        for i:= i + 1 while bool ∧ i < lY do
        if EQ(coefY[i],ZERO) then trivdivY:= trivdivY + 1
        else bool:= false;
        if trivdivX < trivdivY then PQI:= ZERO else
      1: if trivdivY > 0 then
        begin integer dtrivdiv; dtrivdiv:= trivdivX - trivdivY;
```

```
        PQI:= P(POL(i,dtrivdiv,varX,if i = dtrivdiv then ONE
        else ZERO),
        PQI(POL(i,lX -trivdivX,varX,coefX[i + trivdivX]),
        POL(i,lY - trivdivY,varX,coefY[i + trivdivY])))
      end else
    2: begin integer c,d,j; d:= lX - lY; c:= coefY[0];
        begin integer array coef[0:d];
          ES(coef[0],PQI(coefX[0],c));
          for i:= 1 step 1 until d do
          ES(coef[i],PQI(D(coefX[i],Sum(j,0,i - 1,
          P(if i - j < lY then coefY[i - j] else ZERO,coef[j]))),c));
          PQI:= POL(i,d,varX,coef[i])
        end
      end
    end else
    if lX < lY ∧ tY = polynomial then PQI:= ZERO else
    PQI:= POL(i,lX,varX,PQI(coefX[i],Y))
  end
 end;
 ERASE(fnn)
end PQI;
```

comment

Discussion and declaration of integer procedure PGCD.

The algorithm for computing the greatest common divisor of two polynomials, applied in this system, appeared for the first time in [1] and has been extensively described in [7].
A brief summary of the relevant facts will be given in order to compare its description with this ALGOL 60 version.

A set of elements of a unique factorization domain is said to be relatively prime if no prime(of the unique factorization domain) divides all of them. A polynomial over a unique factorization domain is called primitive if its coefficients are relatively prime. Moreover the set of those polynomials forms a unique factorization domain itself.
Any (nonzero) polynomial u(x) over a unique factorization domain S can be factored in the form $u(x) = c \times v(x)$, where v(x) is primitive and c is in S. Furthermore, this representation is unique, in the sense that if $u = c1 \times v1(x) = c2 \times v2(x)$, then $c1 = a \times c2$ and $v2(x) = a \times v1(x)$, where a is a unit of S.
c is said to be the content of u, cont(u), and is a greatest common divisor of the coefficients of u(x).
Notice that this factorization explicitly requires multiplication between elements of S and polynomials over S.
If we take for S the integers representable in this system, realizing that no infinite algebraic system can be represented in a computer, such a multiplication has been defined. For other choices of S it

has not been defined in this system. In the latter case we have to regard c, when multiplying with v(x), as a polynomial over S. Let the value of U be the name of u, the name of c, when c is regarded as a polynomial over S, is the value of PCONT(U,false). If such an operation is not required the value of PCONT(U,true) refers to c, as an element of S.

```
integer procedure PCONT(X,reduce); value X,reduce; integer X;
boolean reduce;
begin integer fnn,i,lX; fnn:= gnn; lX:= length(X) - 2;
  begin integer array coef[-1:lX];
    GET ARRAY(X,i,-1,lX,coef[i]);
    if lX = 0 then PCONT:= if reduce then coef[0] else X
    else
    begin integer a,A,low; boolean bool; bool:= true;
      low:= 0; i:= -1; DE(a,0,0);
      for i:= i + 1 while bool ∧ i < lX do
      if EQ(coef[i],ZERO) then low:= low + 1 else bool:= false;
      A:= coef[low] + saved; i:= low;
      for i:= i + 1 while i < lX ∧ ¬ EQ(A,ONE) do
      if ¬ EQ(coef[i],ZERO) then A:= ASSIGN(a,PGCD(A,coef[i]));
      PCONT:= if reduce then A - saved else POL(i,0,coef[-1],A)
    end
  end;
  ERASE(fnn)
end PCONT;
```

comment

It can be deduced, that cont(gcd(u,v)) = a × gcd(cont(u), cont(v)) and, if pp(u(x)) is defined as u(x)/cont(u(x)), pp(gcd(u(x),v(x))) = b × gcd(pp(u(x)),pp(v(x))), where a and b are units of S and gcd(u(x),v(x)) denotes any particular polynomial in x, which is a greatest common divisor of u(x) and v(x).
These equations reduce the problem of finding a greatest common divisor of arbritrary polynomials to the problem of finding greatest common divisors of primitive polynomials.

As a preliminary step, we describe an algorithm for the pseudo - division of polynomials and its ALGOL 60 version, the integer procedure PSREM.
Given two polynomials, u(x) = u[m] × x ↑ m + ... + u[0] × x ↑ 0, referred to by the value of U, and v(x) = v[n] × x ↑ n + ... + v[0] × x ↑ 0, referred to by the value of V, where v[n] ≠ 0 and m ≥ n > 0, the value of PSREM(U,V,m,n) refers, if n > 0, to the, except for multiplication by an instance of the value

referred to by MINONE, unique polynomial
r(x) = r[n − 1] × x ⩚ (n − 1) + ... + r[0], such that there exists a polynomial q(x) = q[m − n] × x ⩚ (m − n) + ... + q[0] × x ⩚ 0, satisfying v[n] ⩚ (m − n + 1) × u(x) = q(x) × v(x) + r(x).
If n = 0 the value of PSREM equals the value of ZERO.

R: The description by KNUTH in [7, page 369] of an algorithm for the pseudo − division of polynomials is:

R1: [Iterate on k.] Do step R2 for k:= m − n, m − n − 1,...,0. Then the algorithm terminates with u[n − 1] = r[n − 1],..., u[0] = r[0].

R2: [Multiplication loop.] Elaborate q[k]:= u[n + k] × V [n] × x ⩚ k and u[j]:= v[n] × u[j] − u[n + k] × v[j − k] for j:= n + k − 1, n + k − 2,...,0. (When j < k this means that u[j]:= v[n] × u[j], since we treat v[−1],v[−2],... as zero.)
;

```
integer procedure PSREM(X,Y,lX,lY); value X,Y,lX,lY;
integer X,Y,lX,lY; if lY = 0 then PSREM:= ZERO else
begin integer fnn,j,k,LCY; integer array x,XX[−1 : lX],YY[−1 : lY];
  GET ARRAY(X,j,−1,lX,XX[j]);
  GET ARRAY(Y,j,−1,lY,YY[j]); LCY:= YY[lY];
  for j:= 0 step 1 until lX do DE(x[j],XX[j],0);
  for k:= lX − lY step −1 until 0 do
  for j:= lY + k − 1 step −1 until 0 do
  XX[j]:= ASSIGN(x[j], if j − k ≥ 0 then D(P(LCY,XX[j]),P(XX[lY + k],
                                                              YY[j − k]))
                                      else P(LCY,XX[j]));
  PSREM:= SR(fnn,POL(j,lY − 1,YY[−1],XX[j]))
end;
```

comment

gcd(u(x),v(x)) = gcd(v(x),r(x)), for any common divisor of u(x) and v(x) divides v(x) and r(x). Conversely, any common divisor of v(x) and r(x) divides v[n] ⩚ (m − n + 1) × u(x) and it must be primitive(since v(x) is primitive), so it divides u(x). If r(x) = 0, we therefore have gcd(u(x)) = v(x). If r(x) ≠ 0, we have gcd(v(x),r(x)) = gcd(v(x),pp(r(x))), since v(x) is primitive, so the process can be iterated.

COLLINS's algorithm.

Given nonzero polynomials u(x) and v(x) over a unique factorization domain S, this algorithm calculates a greatest common divisor of u(x) and v(x).
We assume that an auxiliary algorithm exists to calculate greatest common divisors of elements of S. The division of a, referred to

by the value of A, by b, referred to by the value of B, in S, when b ≠ 0 and a is a multiple of b, is performed by a call of PQI(A,B).

C1: [Reduce to primitive.] Elaborate d:= gcd(cont(u),cont(v)), and replace u(x) and v(x) by, respectively, pp(u(x)) and pp(v(x)). This is the task of PGCD. a:= 1.

C2: [Pseudo - division.]
Elaborate b:= (v[length(Y)]) ↑ (length(X) - length(Y) + 1).
Calculate r(x) by means of algorithm R, in this system by PSREM.
If r(x) = 0, goto C4. If deg(r) = 0, replace v(x) by "1" (ONE) and go to C4.

C3: [Adjust remainder.] Replace u(x) by v(x) and v(x) by r(x)/a, (The main observation of COLLINS is, that at this point all coefficients of r(x) are multiples of a.), a:= b and return to C2. Steps C2 and C3 are performed by PGCDS.

C4: [Attach the content.] The algorithm terminates, with d × pp(v(x)) as answer. This is performed by PGCD.

;

```
integer procedure PGCD(X1,Y1); integer X1,Y1;
begin integer fnn,X,Y; fnn:= gnn;
  ES(X,X1); ES(Y,Y1);
  if EQ(X,ONE) ∨ EQ(Y,ONE) ∨ EQ(X,MINONE) ∨ EQ(Y,MINONE) then
     PGCD:= ONE else
  if EQ(X,ZERO) ∇ EQ(Y,ZERO) then PGCD:= ZERO else
  if int(X) then PGCD:= if int(Y) then IGCD(X,Y) else
     PGCD(X,PCONT(Y,true)) else
  if int(Y) then PGCD:= PGCD(PCONT(X,true),Y) else
  begin integer i,C,CX,CY,var,coef0; var:= ELEMENT(1,X);
    coef0:= EV(PGCD(ES(CX,PCONT(X,true)),ES(CY,PCONT(Y,true))));
    ES(C,POL(i,0,var,coef0));
    PGCD:= if MULTILINKED STRUCTURE(TYPE(ES(X,PQI(X,POL
                                        (i,0,var,CX))),di,dii))
    ∧ MULTILINKED STRUCTURE(TYPE(ES(Y,PQI(Y,POL
                             (i,0,var,CY))),di,dii)) then
    P(C,if length(X) ≥ length(Y) then PGCDS(X,Y) else PGCDS(Y,X))
    else C
  end;
  ERASE(fnn)
end;

integer procedure PGCDS(X,Y); value X,Y; integer X,Y;
begin integer fnn,x,y,lX,lY,a,A,b,B,i,j; boolean procede;

  fnn:= gnn; procede:= true; lX:= length(X);
```

```
  A:= DE(x,X,DE(y,Y,DE(a,0,DE(b,0,ONE))));
  for Y:= V(y) while procede do
  begin lY:= length(Y); B:= ELEMENT(lY,Y);
    B:= ASSIGN(b,Product(j,0,lX - lY,B));
    B:= ASSIGN(b,POL(i,0,ELEMENT(1,X),B));
    X:= ASSIGN(x,PSREM(X,Y,lX - 2,lY - 2));
    if EQ(X,ZERO) then
    begin procede:= false; PGCDS:= SR(fnn,PQI(Y,PCONT(Y,false))) end
    else
    if int(X) ∨ length(X) = 2 then
    begin procede:= false; PGCDS:= RS(fnn,ONE) end
    else
    begin ASSIGN(y,PQI(X,A)); X:= ASSIGN(x,Y);
      A:= ASSIGN(a,B); lX:= lY
  end end
end;

integer procedure Product(i,low,up,Fi); value low,up;
integer low,up,i,Fi;
begin integer p,fnn; fnn:= gnn; DE(p,ONE,0);
  for i:= low step 1 until up do ASSIGN(p,P(V(p),Fi));
  Product:= V(p); ERASE(fnn)
end;

comment
```

III.2. Rational function arithmetic.

Except for a few trivial differences, the integer procedures performing rational function arithmetic are entirely similar to the ones used for performing rational number arithmetic, OPER ON NUM2, RNSUM and RNPROD, the functions of which have been combined in the integer procedure OPER ON NUM.
The complexity of the procedure OPER ON NUM forces us to explain its functioning in a language better suited for explanation, ALGOL 68. As integer procedure OPER ON RAT is similar to OPER ON NUM, we present a possible ALGOL 68 version of it, after which the ALGOL 60 procedure declaration follows, and refer for the algorithms used to section II.2.2. The go on symbol has been represented by ;.

```
procedure OPER ON RAT = (int operation,formula A,B) ref triple:
(operation = quotient | OPER ON RAT(product,A,RINV(B))|
 heap formula lA,rA,lB,rB,X,Y; ref triple C;
 ((C::A) ∧ (C::B)| lA:= left operand of A; rA:= right operand of A;
   lB:= left operand of B; rB:= right operand of B;
   (operation = sum | heap formula gcd;
    ((gcd:= PGCD(rA,rB)) ≠ one | ¢
      Note that the constituent formal - PARAMETERS - pack of the
```

```
   identity declaration of PGCD, (formula A; formula B), contains
   a go on symbol, as we have to translate the ALGOL 60 evaluation
   from left to right of the actual parameter list into ALGOL 68. ¢
  ((gcd:= PGCD(X:= lA × (rB:= PQI(rB,gcd)) + lB × PQI(rA,gcd),
    Y:= rB)
   ) ≠ ONE | ST rat(PQI(X,gcd),PQI(Y,gcd)) | ST rat(X,Y)
  ) | ST rat(lA × rB + lB × rA,rA × rB)
 ) | ((X:= PGCD(rB,lA)) ≠ ONE | rB:= PQI(rB,X); lA:= PQI(lA,X)
     );
     (X:= PGCD(rA,lB)) ≠ ONE | rA:= PQI(rA,X); lB:= PQI(lB,X)
     ); ST rat(lA × lB,rA × rB)
) |
(¬(C::A) | lA= left operand of B; rA:= right operand of B;
  Y:= A | lA:= left operand of A; rA:= right operand of A;
  (operation = sum | ST rat(lA + rA × Y,rA) |
   ((X:= PGCD(rA,Y)) ≠ ONE | rA:= PQI(rA,X); Y:= PQI(Y,X)
   ); ST rat(lA × Y,rA)
  )
 )
)
)
                                                           ;

integer procedure OPER ON RAT(oper,A,B,tA,lA,rA,tB,lB,rB);
value oper,A,B,tA,lA,rA,tB,lB,rB; integer oper,A,B,tA,lA,tB,lB; real rA,rB;
begin integer fnn; fnn:= gnn; if oper = quotient then begin
  tB:= TYPE(ES(B,RINV(B)),lB,rB); oper:= product end;
 begin integer Gcd,l,r; if tA ≠ rational number ∧ tA ≠ rational
   function then begin lA:= lB; rA:= rB; B:= A end; DE(l,0,DE(r,0,0));
   OPER ON RAT:=
   SR(fnn,if (tA = rational number ∨ tA = rational function) ∧ (tB =
   rational number ∨ tB = rational function) then
         (if oper = sum then
           (if ¬ EQ(ES(Gcd,PGCD(rA,rB)),ONE) then
            (if ¬ EQ(ES(Gcd,
                     PGCD(ASSIGN(l,
                                 S(P(lA,
                                     ES(rB,
                                        PQI(rB,Gcd)
                                    )  ),
                                   P(lB,PQI(rA,Gcd)
                                 ) ) ),
                          ASSIGN(r,P(rA,rB))
                     )      ),ONE
                 )then ST rat(PQI(V(l),Gcd),PQI(V(r),Gcd))
                  else ST rat(V(l),V(r))
           ) else ST rat(S(P(lA,rB),P(lB,rA)),P(rA,rB))
          ) else
```

```
            ST rat(P(if ¬ EQ(ES(Gcd,PGCD(lA,rB)),ONE) then
                     Multiple ES(rB,PQI(rB,Gcd),
                                 ES(lA,PQI(lA,Gcd))
                                )
                     else lA,
                     if ¬ EQ(ES(Gcd,PGCD(lB,rA)),ONE) then
                     Multiple ES(rA,PQI(rA,Gcd),
                                 ES(lB,PQI(lB,Gcd))
                                )
                     else lB
                    ),P(rA,rB)
           )        ) else
           if oper = sum then ST rat(S(lA,P(rA,B)),rA) else
           ST rat(P(if ¬ EQ(ES(Gcd,PGCD(rA,B),ONE) then
                    Multiple ES(rA,PQI(rA,Gcd),PQI(B,Gcd))
                    else B,
                    lA
                   ),
                  rA
      )        )
  end end;
```

comment

IV. The integer procedures S, P, Q and D.

If the values of X and Y refer to arbitrary objects, the value of S(X,Y),P(X,Y),Q(X,Y) and D(X,Y) refers to an instance of their sum, product, quotient and difference, respectively. It is assumed that their declarations are self – explanatory (after reading chapters I, II and III).

;

```
boolean procedure numbertype(type); value type; integer type;
numbertype:= type = long integer ∨ type = short integer ∨ type =
rational number;

boolean procedure polynomialtype(type); value type; integer type;
polynomialtype:= type = polynomial ∨ type = short integer ∨ type
= long integer;

boolean procedure rationaltype(type); value type; integer type;
rationaltype:= type = rational function ∨ type = rational number
∨ polynomialtype(type);

integer procedure S(A1,B1); integer A1,B1;
begin integer A,B,tA,tB,lA,lB,n; real rA,rB; n:= gnn;
  tA:= TYPE(ES(A,A1),lA,rA); tB:= TYPE(ES(B,B1),lB,rB);
  S:= SR(n,if EQ(A,ZERO) then B-saved else
  if EQ(B,ZERO) then A-saved else
  if numbertype(tA) ∧ numbertype(tB) then
     OPER ON NUM(sum,A,B,tA,tB) else
  if polynomialtype(tA) ∧ polynomialtype(tB) then
     OPER ON POL(sum,A,B,tA,rA,tB,rB) else
  if rationaltype(tA) ∧ rationaltype(tB) then
     OPER ON RAT(sum,A,B,tA,lA,rA,tB,lB,rB) else
  STORE(A,sum,B))
end S;

integer procedure P(A1,B1); integer A1,B1;
begin integer A,B,tA,tB,lA,lB,n; real rA,rB; n:= gnn;
  tA:= TYPE(ES(A,A1),lA,rA); tB:= TYPE(ES(B,B1),lB,rB);
  P:= SR(n,if EQ(A,ZERO) ∨ EQ(B,ZERO) then ZERO else
  if EQ(A,ONE) then B-saved else if EQ(B,ONE) then A-saved else
  if numbertype(tA) ∧ numbertype(tB) then
     OPER ON NUM(product,A,B,tA,tB) else
  if polynomialtype(tA) ∧ polynomialtype(tB) then
     OPER ON POL(product,A,B,tA,rA,tB,rB) else
  if rationaltype(tA) ∧ rationaltype(tB) then
     OPER ON RAT(product,A,B,tA,lA,rA,tB,lB,rB) else
  if tA = sum then S(P(lA+saved,B),P(rA+saved,B)) else
  if tB = sum then S(P(A,lB+saved),P(A,rB+saved)) else
```

```
  STORE(A,product,B))
end P;

integer procedure D(A,B); integer A,B;
D:= S(A,P(MINONE,B));

integer procedure Q(A1,B1); integer A1,B1;
begin integer A,B,tA,tB,lA,lB,Gcd,n; real rA,rB; n:= gnn;
  tA:= TYPE(ES(A,A1),lA,rA); tB:= TYPE(ES(B,B1),lB,rB);
  if EQ(B,ZERO) then PR string(<B ZERO in Q>);
Q:= SR(n,if EQ(A,ZERO) ∨ EQ(B,ZERO) then ZERO else
  if EQ(B,ONE) then A-saved else
  if numbertype(tA) ∧ numbertype(tB) then
     OPER ON NUM(quotient,A,B,tA,tB) else
  if polynomialtype(tA) ∧ polynomialtype(tB) then
     (if EQ(ES(Gcd,PGCD(A,B)),ONE) then
         (if LC(B) > 0 then STORE(A,rational function,B)
          else STORE(EV(P(A,MINONE)),rational function,P(B,MINONE))
         ) else ST rat(PQI(A,Gcd),PQI(B,Gcd))
     ) else
  if rationaltype(tA) ∧ rationaltype(tB) then
     OPER ON RAT(quotient,A,B,tA,lA,rA,tB,lB,rB) else
     STORE(A,quotient,B))
end Q;
```

comment

V. Output and conversion. ;

```
procedure OUTPUT(F); value F; integer F;

begin procedure OP(F,type); value F,type; integer F,type;
  begin integer t,A; real B;
    procedure LBR; if t < type then PR string(⊰(⊱);
    procedure RBR; if t < type then PR string(⊰)⊱);
    t:= TYPE(F,A,B);
    if t = algebraic variable then Ovar(F) else
    if t = short integer ∨ t = long integer then Oint(F) else
    if DYADIC OP(t) then
    begin LBR; OP(A,t); if t = sum then PR string(⊰+⊱) else
      if t = product then PR string(⊰×⊱) else PR string(⊰/⊱);
      OP(B,t); RBR
    end else
    begin integer i,degree,X; degree:= length(F) - 2;
      begin integer array coef[-1:degree];
        GET ARRAY(F,i,-1,degree,coef[i]);
        if t = polynomial then
        begin integer coefi; t:= sum; LBR; X:= coef[-1];
          for i:= 0 step 1 until degree do
          begin coefi:= coef[i]; if EQ(coefi,ZERO) then goto end for i;
            OP(coefi,product); PR string(⊰×⊱); Ovar(X); PR string(⊰↑⊱);
            PR int num(i); if i < degree then PR string(⊰+⊱);
          end for i:
          end; RBR
        end else
        begin PR string(⊰(⊱); for i:= 0 step 1 until degree do
          begin OP(coef[i],0); if i < degree then PR string(⊰,⊱) end;
          PR string(⊰)⊱)
  end end end end OP;

  procedure Oint(X); value X; integer X;
  begin integer fnn,l; boolean b;
    fnn:= gnn; l:= length(X); b:= ISIGN(X) < 0; if b then PR string(⊰(⊱);
    if l = 1 then PR int num(VAL OF S INT(X))
    else
    begin integer elem;
      if b then PR string(⊰-⊱); PR int num(abs(ELEMENT(l,X)));
      for j:= l - 1 step -1 until 1 do
      begin elem:= abs(ELEMENT(j,X));
        if elem < 1000 then
        begin if elem > 100 then PRstring(⊰00⊱) else
              if elem >  10 then PRstring(⊰0000⊱) else
              if elem =   0 then PRstring(⊰000000⊱) else
                                  PRstring(⊰00000⊱)
        end
```

```
        else  if elem < 10 ↑ 4 then PRstring(⊀00⊁) else
              if elem < 10 ↑ 5 then PRstring(⊀0⊁);
            PR int num(elem)
      end
    end; if b then PR string(⊀)⊁)
  end Oint;
  OP(F,0)
end;

procedure Ovar(X); value X; integer X;
PR sym(VAL OF S INT(X) + 9);

procedure PR string(s); string s;
begin PRINTTEXT(s)end;

procedure PR nlcr; PR string(⊀
⊁);

procedure PR num(a); value a;real a;
begin PRINT(a)end;

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;

procedure PR sym(a); value a; integer a;
begin PRSYM(a) end;
```

comment

In the procedure Oint, whose function is the output of long or short integers, it has been assumed that G = 10 ↑ 6.
The function of the procedure Ovar is the output of algebraic variables.
By elaborating the call Ovar(X), the symbol, of which the number in the alfabet has been specified by the value of VAL OF S INT(X), is printed.
The addition of 9 to the value of VAL OF S INT(X) in the procedure body of Ovar reflects the use of a standard-procedure-PRSYM- of the Mathematical Centre.
The standard procedures, that have been used without describing them, are PRINTTEXT and PUTEXT, for printing and punching a text between the Mathematical Centre version of the string quotes "⊀" and "⊁", and PRSYM and PUSYM, for printing and punching a symbol.
They have been described in [8].

```
boolean procedure even(s); vale s; integer s; even:= s = s : 2 × 2;
```

```
integer PL1,PL2,i,X,fnn;
comment now we shall demonstrate a simple example.;
INITIALIZE; fnn:= gnn; ES(X,AV(0,24))                   ;
ES(PL1,POL(i,8,X,if i = 0 then S INT(-5) else
                 if i = 1 then S INT(2) else
                 if i = 2 then S INT(8) else
                 if i = 3 then S INT(-3) else
                 if i = 4 then S INT(-3) else
                 if i = 5 then ZERO else
                 if i = 6 then ONE else
                 if i = 7 then ZERO else ONE));

ES(PL2,POL(i,6,X,if i = 0 then S INT(21) else
                 if i = 1 then S INT(-9) else
                 if i = 2 then S INT(-4) else
                 if i = 3 then ZERO else
                 if i = 4 then S INT(5) else
                 if i = 5 then ZERO else S INT(3)));
PR string(⌈ The greatest common divisor of ⌉); PRnlcr;
OUTPUT(PL1); PR string(⌈    and     ⌉); PRnlcr;
OUTPUT(PL2); PR string(⌈    is:     ⌉);
OUTPUT(EV(PGCD(PL1,PL2))); ERASE(fnn)
end end
```

The input tape consists of   5000   1000000

The output is:

```
The greatest common divisor of
(-5)*x↑0+2*x↑1+8*x↑2+(-3)*x↑3+(-3)*x↑4+1*x↑6+1*x↑8     and
21*x↑0+(-9)*x↑1+(-4)*x↑2+5*x↑4+3*x↑6     is:     1
```

[1] G.E. Collins, Subresultants and reduced polynomial remainder sequences, JACM, vol. 14, nr. 1, Jan. 1967, pp. 128-142.

[2] G.E. Collins, The SAC-1 integer arithmetic system, Technical reference note of the University of Wisconsin Computing Centre.

[3] G.E. Collins, The revised SAC-1 integer arithmetic system, Technical reference note of the University of Wisconsin Computing Centre.

[4] G.E. Collins, The SAC-1 polynomial system, Technical reference note of the University of Wisconsin Computing Centre.

[5] G.E. Collins, The SAC-1 rational function system, Technical reference note of the University of Wisconsin Computing Centre.

[6] D.E. Knuth, The art of computer programming, Volume 1, Fundamental algorithms, Addison Wesley.

[7] D.E. Knuth, The art of computer programming, Volume 2, Semi-numerical algorithms, Addison Wesley.

[8] F.E.J. Kruseman-Aretz, Het MC-ALGOL 60-systeem voor de X8, Voorlopige programmeurshandleiding, MR 81, Mathematisch Centrum.

[9] B. Mailloux, On the implementation of ALGOL 68, Mathematisch Centrum.

[10] P. Naur (Editor), Revised report on the algorithmic language ALGOL 60.

[11] W.V.O. Quine, From a logical point of view, Harper Torchbooks, Harper and Row, New York.

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

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

[14] B.L. van der Waerden, Algebra I, Springer-Verlag, Heidelberg.

[15] A. van Wijngaarden (editor), B.J. Mailloux, J.E.L. Peck and C.H.A. Koster, Report on the algorithmic language ALGOL 68, second printing by the Mathematisch Centrum, Amsterdam, MR 101, October 1969.

List of errata. MR 119/70.

page 1, line 15	else STORE (a,sum,b). →
	else STORE (a,sum,b);
page 1, line 18	if a = one then b else if b = one then a. →
	if a = one then b else if b = one then a
page 14, line 11	1: the condition F ≥ 0 →
	1: The condition F > 0
page 14, last line	from a call of integer procedure STORE: →
	from a call of integer procedure STORE: ;
page 15, last line	the ; symbol must be deleted.

page 14 and 15, modified as indicated above, must be interchanged.

page 38, line 20	,_, → ,+,
page 46, line 21	Product: = V(p); ERASE (fnn) →
	Product: = V(p)-saved; ERASE (fnn)
page 52, last line	vale → value
page 53, line 17	if i = 5 then ZERO else S INT(3)); →
	if i = 5 then ZERO else S INT(3)));