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.
R.P. VAN DE RIET

ABC ALGOL
A PORTABLE LANGUAGE FOR
FORMULA MANIPULATION SYSTEMS

PART 2 : THE COMPILER

MATHEMATISCH CENTRUM AMSTERDAM 1976
Part 1 of this treatise is published as MC TRACT 46 and contains the Table of Contents and the Introduction pertinent to this second part.
begin comment
6. The ABC ALOOL compiler

6.0. Preliminaries of translation process

Algorithm;

integer preceding begin, begin;

comment:
The above integers are global in the translation process. Their values
point to the information cells of the last two treated begin's.

6.1. Prerequisites for translation process

Syntactic definition:

<program> ::= <block>|<compound statement>

A program may be preceded by comment.

Algorithm;

procedure envelope of block(procedure body);
value procedure body; integer procedure body;
begin integer ptr to integer, ptr to formula, nr of array segments,
    ptr to array segment, max dimension in array declarations;
    Boolean formula block, dangerous inner block, unn necessary,
    block contains labels, block contains gotos, block,
    proc id ass stat, interested in proc id;

comment:
"envelope of block" translates a block, a compound statement or a
statement and treats them all as blocks. It is called from the main
program to translate an ABC ALOOL program, from a procedure body (in
which case the value of "procedure body" is # 0 indicating where the
information about the procedure may be found) or for the translation
of a real block.
The procedure contains the declarations of the above block-administrative variables and all the syntax-translation procedures as "block or statement", "declaration" or "statement". The effect is that the above variables are global to these procedures, while they are local within the procedure "envelope of block".

Each block is translated by a call of "envelope of block", thus each block has attached to it a private set of the above variables, which are initialized by a call of "Open new block in information list".

6.2. Translation of a block

Syntactic definition:

\[
\begin{align*}
\text{block} &::= \text{block head} \triangleleft \text{compound tail} \\
\text{block head} &::= \text{begin} \triangleright \text{declaration with semicolon} \\
\text{begin} \triangleright \text{declaration with semicolon} \\
\text{declaration with semicolon} &::= \text{declaration} \\
\text{compound tail} &::= \text{statement} \text{ end } \text{ ? } \text{ statement } \triangleleft \text{compound tail}
\end{align*}
\]

Algorithm;

procedure block or statement;
begin integer lnr; block:= synt unit = begin symbol;
Open new block in information list;
PR string(begin $); lnr:= line number;
if ptr to array segment $ 0 then
begin comment
In this block formula arrays are declared. Therefore, it is surrounded by another block in which the array bounds are evaluated;
SAVE reading ptrs; array bounds; PR nlc;
PR string(begin $); RESET reading ptrs
end;
comment
The next statement serves to determine whether this block is the procedure body of a formula procedure;
if block then
begin RE begin; for synt unit:= synt unit
while is declarator(synt unit) do declaration
end;
interested in proc id:= if procedure body ≠ 0 then
contents of [procedure body - 4] + 32 =
formula symbol x 1024 + procedure symbol
else false;
if second scan then
begin SAVE reading ptrs;
if ptr to array segment ≠ 0 then
begin Declare formula arrays; Declare integers end else
begin Declare integers; PR string(4lru(4));
PR int num(lnr); PR string(4);+4
end;
if interested in proc id then procedure block entry;
Introduce names for formulae;
Introduce names for formula array elements;
Initialize srn;
RESET reading ptrs
end else formula block:= ptr to formula + ptr to array segment ≠ 0;
proc id ass stat:= false;
L1: statement(block v formula block);
comment the actual parameter of "statement" indicates whether the
environment of the statement is such that it can be split into more
than one statements as e.g. in a block, or that it should be sur-
rounded by begin and end when it is split.;
if synt unit ≠ semicolon symbol ^ synt unit ≠ end symbol then
begin ERR(4statement not properly closed+);
SEEK(synt unit = semicolon symbol v synt unit = end symbol)
end;
if block then
begin L2: if synt unit = semicolon symbol then
begin PR and ME semicolon; goto L2 end;
if synt unit ≠ end symbol then
begin proc id ass stat:= false; goto L1 end
end;
Block exit; Close block in information list
end block or statement;
comment:
Each block has a certain amount of administration stored in the
information list. The information is "initialized" upon a call of
"open new block in information list" and is "buried" upon a call of
"close block in information list".

The block: "begin d s end" is translated into:
"begin d; integer fn, srr; fn:= grn1 srr:= grn2 s
i := ERASE(fn) end",
where d and s stand for a declaration and a statement, respectively.
D is the first half of the translation of the declaration d ("formula
x := f" is translated into "integer x").
D2 is the introduction of names for formula variables("DE(X,F)"). The
locally introduced integers "fn" and "srr" are being used for erasure
of names and thus of formulae. The integer "grn" is, in the translated
program, a global integer which counts the number of names introduced.
"ERASE" is a procedure which erases the last "grn - fn" introduced
names. A labelled statement "l: c" is translated into:
"l: ERASE(srr); S",
and again "ERASE" erases the last "grn - srr" introduced names.

6.3. Translation of a declaration

Syntactic definition:

<declaration>::= <formula declaration>|<formula array declaration>|
<formula procedure declaration>|<procedure declaration>|<type declaration>|<array declaration>|<switch declaration>

The first four declarations are defined in sections 6.3.1, 6.3.2,
6.3.3 and 6.3.4 respectively. The other declarations are defined in
the ALGOL 60 report [10] section 5.

Algorithm;
procedure declaration;
if synt unit = formula symbol then
begin if next synt unit = array symbol then formula array declaration
else if next synt unit = procedure symbol then procedure declaration
else formula declaration
end else
if synt unit = real symbol v synt unit = integer symbol v
synt unit = Boolean symbol then
begin if next synt unit = array symbol then array declaration
else if next synt unit = procedure symbol then procedure declaration
else type declaration
end else
if synt unit = array symbol then array declaration else
if synt unit = procedure symbol then procedure declaration else
begin CHECK(switch symbol); PR and RE; PR string(424); PR and RE;
PR and RE;
L; other expression; if synt unit = comma symbol then
begin PR and RE; goto L end; PR and RE semicolon
end declaration;
procedure type declaration;
begin integer su; su:= synt unit;
if su = integer symbol then
begin L1: RE; if first scan then
ptr to integer:= STORE into information list(st(st(st(st(0,
nr of ident),
ptr to first letgit),
integer symbol x t15 + declared as value),
ptr to integer));
RE; if synt unit = comma symbol then goto L1;
RE semicolon
end else
begin L2: PR and RE; if su # Boolean symbol ^ first scan then
STORE into information list(st(st(st(0,
nr of ident),
ptr to first letgit),
su x t15));
PR string(424); PR and RE; if synt unit = comma symbol then goto L2;
PR and RE semicolon
end end type declaration;
procedure array declaration;
begin integer su, nsu; su:= synt unit; nsu:= next synt unit;
if nsu = array symbol then PR and RE;
L1: PR and RE; if su ≠ Boolean symbol "first scan then
STORE into information list(st(st(st(0,
nr of ident),
ptr to first letgit),
(if nsu = array symbol then su else real symbol) × t15 +
32 × array symbol));
PR string(4Z4); PR and RE;
if synt unit = sub symbol then
begin L2: PR and RE; other expression; CHECK(colon symbol);
PR and RE;
other expression; if synt unit = comma symbol then goto L2;
' CHECK(bus symbol); PR and RE
end; if synt unit = comma symbol then goto L1; PR and RE semicolon
end array declaration;

comment:
The other declarations will be treated in the next sections.

6.3.1. Translation of a formula declaration

Syntactic definition:

<formula declaration>::= <formula> <formula list>
<formula list>::= <formula definition> | <formula list>,
<formula definition>
<formula definition>::= <simple variable><formula initialization>
<formula initialization>::= <empty> | = <formula expression>
= <formula expression>
<simple variable>::= <identifier>
Algorithm;

procedure formula declaration;
begin L: RE; if first scan then
ptr to formula= STORE into information list(st(st(st(0,
nr of ident),
ptr to first letgit).

formula symbol \times t15 + (if next synt unit = equal
symbol then declared as value else declared as name)),
ptr to formula));

SKIP text until(synt unit = comma symbol \ v synt unit =
semicolon symbol);

if synt unit = comma symbol then goto L;
RE semicolon
end formula declaration;

comment:
The following procedures declare all integers, introduce all names and
produce the initialization for smn. note that "formula f := expr1,s =
eqr;" is translated into: "integer G,p_r;".
It is necessary that names are introduced for "P" and "G" and that
the initial values are computed immediately after the translated
declarations.

Algorithm;

procedure Declare integers;
begin integer a,a0,a3,mode; Boolean first,reverse;
procedure PR int com;
if first then
begin PR string(\text{integer } \cdot); first := false end
else PR string(\text{integer } \cdot);
first := true; reverse := false; a0 := 0;
for a := ptr to integer, ptr to formula do
begin for a := a while a \neq 0 do

begin SET reading ptrs on(contents of [a + 2]);
a3 := contents of [a + 3]; mode := a3 - a3 \div 32 \times 32;
if mode = with local v mode = specified as value then
begin PR int com; PR string(\text{integer } \cdot); if operator identifier then
PR operator else PR synt unit
end else if mode = without local then
begin PR int com; PR string(\text{integer } \cdot); PR synt unit end
else operator identifier := false;
a3 := contents of [a + 4]; if reverse then
begin contents of \([a + 4]\):= a0; a0:= a \textbf{end}; a1:= a3
end; reverse:= true
end; ptr to formula:= a0;
if formula block then
begin PR int com; PR string(4fnn4);
if ann necessary then
PR string(4,nn4); PR string(4; fnn:= grn4)
end; if \(\neg\) first then PR string(4;4)
end Declare integers;

procedure Introduce names for formulas;
begin integer a,a3,mode; Boolean first:=;print NULL4;
print NULL:= false; first:= true;
for a:= a while a \(\neq\) 0 do
begin a3:= contents of \([a + 3]\); mods:= a3 - a3 \(\pm\) 32 \(\times\) 32;
if first then
begin if mode \(\neq\) specified as value then
begin PR string(4z4); SET reading ptrs on ( contents of \([a + 2]\)); PR and RE; print NULL:= true;
begin PR string(4:= 4)
end end else
begin if mode = specified as value v mode = declared as value then
PR string(4DE(4) else PR string(4DE(4) else
PR string(4Z4);
if mode = specified as value then PR string(4Y4) else
PR string(4Z4);
SET reading ptrs on (contents of \([a + 2]\)); PR and RE;
PR string(4,4); if mode = specified as value then
begin PR string(4Z4); SET reading ptrs on (contents of \([a + 2]\));
PR synt unit
end else
if synt unit = becomes symbol v synt unit = equal symbol then
begin RE; formula expression end else PR string(4NULL4);
PR string(4); 4
end;
a3:= contents of \([a + 4]\); a:= a3
end;
if first then
begin first:= false; if print NULL then PR string(4NULL;4);
      goto L
end
end Introduce names for formulae;

procedure Initialize ann;
if formula block ^ ann necessary then
   PR string(4 ann:= ann; 4);

comment:

6.3.2. Translation of a formula array declaration

Syntactic definition:

<formula array declaration> ::= formula array <array list>
For array list see the ALGOL 60 report [10] section 5.2.

Algorithm;

procedure formula array declaration;
begin integer a,dimension; RE;
ARRAY SEGMENT: RE; if first scan then
   begin PTR to array segment:= a:=
      STORE into information list(st(st(st(st(0,
         nr of ident),
      PTR to first letgit),
      formula symbol x t15 + 32 x array symbol),
      PTR to array segment),0));
      nr of array segments:= nr of array segments + 1
   end;
   dimension:= 1;
NEXT IDENTIFIER: RE;
if synt unit = comma symbol then
   begin RE; if first scan then
      STORE into information list(st(st(st(0,
         nr of ident),
      PTR to first letgit),
formula symbol \times t15 + 32 \times array symbol)); goto NEXT IDENTIFIER

end else

if synt unit = sub symbol then

begin BOUND PAIR: RE; SKIP text until(synt unit = comma symbol v synt unit = bus symbol);

if synt unit = comma symbol then

begin dimension:= dimension + 1; goto BOUND PAIR end;

if first scan then

begin contents of[a + 5]:= dimension; if max dimension in array

declarations < dimension then

max dimension in array declarations:= dimension

end; RE;

if synt unit = comma symbol then goto ARRAY SEGMENT

end; RE semicolon.

end formula array declaration;

comment:
The translation of:

"Formula array a, b[e1:e2], c, d[e3:e4, e5:e6], i."

is:

"Integer array c, d[low1c1:up1c1, low2c1:up2c1], a, b[low2c1:up2c1], i."

It is necessary that the bounds e1, e2, e3, e4, e5 and e6 are evaluated in a surrounding block and assigned to the array elements of "low" and "up". For this example, the next procedure "array bounds" produces the following text:

"Integer low1c1, up1c1, low2c1, up2c1, i1, low1c2, up1c2, low2c2, up2c2, i2",

low1c1:= E3i1, up1c1:= E4i1

low1c2:= E3i2, up1c2:= E4i2

low2c1:= E5i1, up2c1:= E6i1"

In a case that the block is the body of a procedure, it is necessary to treat the procedure parameters first, i.e. declaring integers and producing names for the formal parameters. This is done in a somewhat tricky way in that the list of formula variables is split into two lists: one consisting of the parameters and the other consisting of the ordinary variables declared in the heading of the block. The latter list has to be treated together with the formula array
declaration, i.e. in the proper block. The first list is treated in
this procedure. Note that the line number is treated also.

Algorithm:

procedure array bounds;
begin integer dimension,a,index,m,ptr,pti,lnr; boolean sn;
lnr:= line number; a:= ptf:= ptr to formula;
ptr to formula:= index:= 0;
for a:= a while a ≠ 0 do
begin m:= contents of[a + 3]; m:= m - m ÷ t15 × t15;
if m = declared as value v m = declared as name then
begin index:= a; a:= contents of[a + 4] end else
begin if index ≠ 0 then contents of[index + 4]:= 0;
ptr to formula:= a;a:= 0
end
end;
PR nlc; PR string(integer #);
for a:= 1 step 1 until max dimension in array declarations do
begin for index:= 1 step 1 until nr of array segments do
begin sn:= sn necessary; sn necessary:= false;
ptr:= ptr to integer; ptr to integer:= 0;
declare integers; PR string(integer #);
PR int num(lnr); PR string(integer #);
introduce names for formulae;
ptr to formula:= ptf; ptr to integer:= pti; sn necessary:= sn;
a:= ptr to array segment; index:= 0;
for a:= a while a ≠ 0 do
begin PR nlc; set reading ptrs on(contents of[a + 2]);
dimension:= contents of[a + 5]; index:= index + 1;
FIND SUB SYMBOL:
RE; if synt unit ≠ sub symbol then goto FIND SUB SYMBOL; RE;
for m:= 1 step 1 until dimension do
  begin PR sn(41ow4=index,PR sn(4c4,=m,PR string(4:= 4)));  
  other expression; PR string(4:4); CHECK(colon symbol); RE;  
  PR sn(4up4,index,PR sn(4c4,m,PR string(4:= 4)));  
  other expression; PR string(4:4);  
  if m < dimension then CHECK(comma symbol) else  
  CHECK(bus symbol); RE  
  end; a:= contents of [a + 4]  
end

comment
In the following procedure the actual formula array declaration is  
produced.

Algorithm;

procedure Declare formula arrays;
begin integer dimension,a,n,m,index;
index:= 0; PR nlor; PR string(4 integer array 4);  
a:= ptr to array segment;
for a:= a while a ≠ 0 do  
begin if a ≠ ptr to array segment then PR string(4,4);  
SET reading ptrs on(contents of [a + 2]);  
index:= index + 1;  
L1: PR string(4L4); PR and RE; dimension:= 1;  
if synt unit = comma symbol then  
begin PR and RE; goto L1 end else  
if synt unit = sub symbol then  
begin L2: PR and RE; SKIP text until  
(synt unit = comma symbol v synt unit = bus symbol);  
PR sn(41ow4,index,PR sn(4c4,dimension,  
PR sn(4: up4,index,PR sn(4c4,dimension,1)));  
if synt unit = comma symbol then  
begin dimension:= dimension + 1; goto L2 end;  
CHECK(bus symbol); PR and RE  
end; a:= contents of [a + 4]
end;
FR string(4;4); PR nler
end Declare formula arrays;

comment:

We finally have to construct an algorithm for the declaration of the
array elements and the introduction of names for them. For the above
example, this algorithm generates the following:

for i1 := low1c1 step 1 until up1c1 do
for i2 := low1c2 step 1 until up1c2 do
begin DE(C[i1,i2],NULL); DE(D[i1,i2],NULL) end ;
for i1 := low2c1 step 1 until up2c1 do
begin DE(A[i1],NULL); DE(B[i1],NULL) end ;"

Algorithm:

procedure Introduce names for formula array elements;
begin integer dimension,a,m,index;
a := ptr to array segment; index := 0;
for a := a while a # 0 do
begin SET reading ptrs on contents of [a + 2]);
dimension := contents of [a + 5]; index := index + 1;
for m := 1 step 1 until dimension do
begin PR nler; PR sn(
  for i1,m,PR sn(4 := low4,index,PR sn(
    for i2,m,PR sn(4 step 1 until up4,index,PR sn(
      for i3,m,PR string(4 do4))))))
end;
FR nler; PR string(4begin 4);
L3: PR string(4DE(Z4)); PR and PE; PR string(4});
for m := 1 step 1 until dimension do
PR sn(4i4,m,if m < dimension then PR string(4,4)
else PR string(44.NULL4));
if synt unit = comma symbol then
begin PR string(4;4); PE; goto L3 end;
FR string(4 end;4); a := contents of [a + 4]
end
end Introduce names for formula array elements;
comment:

6.3.3. Opening and closing a block.

During the first scan of the translation process, information cells (IC) in the information list are reserved for each block begin and for each block end.

In these IC's the values of the block-administrative variables declared in section 6.1 and to which values are assigned in sections 6.3.1 and 6.3.2, are stored together with pointers for the linkage of the block IC's. During the second scan the stored values are used for:

a) Outputting the correct "block begin".
b) Searching the IC of an identifier used in a statement.

Let the begin's of the program being enumerated in the order in which they occur in the program: b[i], i = 1,...,n.

Let the corresponding end's be: e[i], i = 1,...,n.

Let the IC of b[i] be: IC(b[i]) and the IC of e[i] be IC(e[i]), i = 1,...,n.

Let the blocks be b[i], i = 1,...,n, such that the begin of b[i] is b[i].

Let us define a surrounding block b[j] of b[i] as a block in which b[i] is contained. Obviously, we have j < i.

The immediately surrounding block b[j] of b[i] is the surrounding block of b[i] for which i-j is minimal. Define s(i) to be this j.

Obviously, for each i > 1, s(i) exists.

We are now able to describe information stored in IC(b[i]) and IC(e[i]).

IC(b[i]) contains, after the first scan is executed:

1. A reference to IC(e[i]).
2. A reference to IC(b[i + 1]), in case i < n.
3. The values of the block-administrative variables.
4. The block-administrative variables as mentioned in section 6.1.

IC(e[i]) contains, after the first scan is executed:

1. A reference to IC(b[s(i)]), provided i > 1.
Between IC(b[i]) and IC(e[j]), the IC's of the identifiers, declared in b[i], are stored in the information list. Upon entry of a new block, during the second scan, we can now find the corresponding IC(b[i]), since we know IC(b[i - 1]), provided i > 1. Hence, the block-administrative variables can be found and, during the translation of a statement, the IC of an identifier can be found by means of the following simple algorithm:

Step 1: Choose IC(b[i]). Take Step 2.
Step 2: Choose the next IC.

If this IC is the IC of a begin, namely IC(b[j]), then choose IC(e[j]) and take Step 2 again.
If this IC is the IC of an end, namely IC(e[j]), then, if j = 1, the process is finished and no IC is found for the identifier, otherwise choose IC(b[f(s(j))]) and take Step 2 again.
If this IC is the IC of an identifier, then check whether this identifier is the identifier we looked for, in which case the process is finished, otherwise take Step 2 again.

As we will see later, another process will be followed for the standard identifiers.

Example: The program:
"begin formula a,b; S1;
begin formula c;
  procedure P(d); formula d ;
  begin formula e; S2 end;
  formula f; S3;
  begin formula g; S4 end;
end; S5;
begin formula h; S6 end
end"
gives rise to the following information list in which the references are made visible by means of arrows:

begin formula a,b; S1;
A search for the IC of an identifier in S2 leads to the inspection of the IC's of the following identifiers: e, d, c, P, f, a, b. A search for the IC of an identifier in S4 leads to the inspection of the IC's of the following identifiers: g, c, P, f, a, b.

We give now the algorithm for searching the IC of an identifier given by the global integers: "hr of ident" and "ptr to first letgit". These integers have obtained values by means of the reading procedures "HR" and "PR and FE".

The global integer "block depth" decreases by one each time the block depth of the block in which the identifier is sought diminishes.

**Algorithm:**

**Boolean procedure** Search for identifier (IC); integer IC;
begin integer p, i;
p:= begin; Search for identifier:= false;
for p:= p while p ≠ 0 do
begin p:= p + contents of [p]; if p ≥ ptr of inf list then goto OUT;
j:= contents of [p + 1];
if j = -end symbol then
begin block depth:= block depth - 1;
p:= contents of [p + 2]
end else
if j = - begin symbol then p:= contents of [p + 2]
else if j = nr of ident
then begin Search for identifier:= true; IC:= p; goto OUT end
end;
OUT: end Search for identifier;

comment:
We give now the block-administrative procedures.

Algorithm:

procedure Open new block in information list;
if first scan then
begin ptr to integer:= ptr to formula:= ptr to array segment:=
nr of array segments:= max dimension in array declarations:= 0;
block contains labels:= block contains gotos:= false;
if procedure body ≠ 0 then
begin ptr to integer:= contents of [procedure body + 4];
ptr to formula:= contents of [procedure body + 5]
end else
begin begin:= contents of [preceeding begin]:= 
STORE into information list(st(st(st(st(st(st(0,
- begin symbol),
begin),
0),
ptr to integer),
ptr to formula),
ptr to array segment),
nr of array segments),
max dimension in array declarations),
procedure Close block in information list;

if first scan then
begin integer prec begin; Boolean array e,b[0:5];
 prec begin:= contents of [begin + 2];
 contents of [begin + 2]:= 
STORE into information list(st(st(0,
  - end symbol),
  prec begin));
 contents of [begin + 4]:= ptr to integer;
 contents of [begin + 5]:= ptr to formula;
 contents of [begin + 6]:= ptr to array segment;
 contents of [begin + 7]:= nr of array segments;
 contents of [begin + 8]:= max dimension in array declarations;
 comment:
The following can be understood in connection with section 4.6 only.
In the Boolean array \( b \) and \( e \) the properties: form, label, goto, dang, 
sum and sum2 are represented for the array elements with subscripts 
0, 1, 2, 3, 4 and 5, respectively. The array \( b \) gives the properties 
for the block under consideration and the array \( e \) gives the properties 
for the enveloping block.

Note that the transfer of properties from an inner block to the 
block under consideration, such as described in section 4.6, is 
implemented as the transfer of properties from the block under 
consideration to its (unique) enveloping block (provided it is not the 
outermost block in which case \( \text{prec begin} = 0 \)). Therefore, \( b[3] \) does 
not obtain a value here, but \( e[3] \) does, if necessary.;

get bits(contents of \([\text{begin + 9}], b\));
b[0]:= formula block;
b[1]:= block contains labels;
if block contains gotos then \( b[2] := \text{true} \);
if \( b[1] \wedge b[3] \) then \( b[4] := \text{true} \);
if \( b[1] \) then \( b[5] := \text{true} \);
store bits(contents of \([\text{begin + 9}], b\));
if \( \text{prec begin} \neq 0 \) then
begin get bits(contents of \([\text{prec begin + 9}], e\));
if \( b[2] \) then \( e[2] := \text{true} \);
if \( b[0] \wedge b[2] \vee b[3] \) then \( e[3] := \text{true} \);
if \( b[4] \wedge \neg b[0] \) then \( e[4] := \text{true} \);
if \( b[1] \wedge \neg b[0] \) then \( e[5] := \text{true} \);
store bits(contents of \([\text{prec begin + 9}], e\))
end;
begin:= \( \text{prec begin} \)
end else begin:= contents of \([\text{contents of }[\text{begin + 2}] + 2]\);

procedure procedure block entry;

comment the procedure is called during the second scan only, while the 
block is the body of a formula procedure;

begin integer a; a:= procedure body - 7;
if contents of \([a + 6] \geq 1 \) then
begin FR string(4
if protect then ERR(4protection error in form proc4);4);
SET reading ptrs on (contents of \([a + 2]\));
PR nlc; PR string(4Y+); if operator identifier then
PR operator else PR synt unit;
PR string(4:= NULL;+)
end end procedure block entry;

procedure Block exit;
if first scan then
begin if block then RE end;
if interested in proc id then
begin if proc id ass stat then
contents of [procedure body - 1]:= 1
else contents of [procedure body - 4]:= formula symbol x t15 + procedure symbol x 32 + without local
end else
begin integer a,i; Boolean b; PR nlc;
if formula block then PR string(4; ERASE(fnn)+);
a:= if interested in proc id then procedure body - 7 else 0;
if a = 0 then
begin if contents of [a + 6] >= 1 then
begin PR string(4; if Y+); SAVE reading ptrs;
for i:= 1,2,3 do
begin SET reading ptrs on(contents of [a + 2]);
b:= operator identifier;
if b then PR operator else PR synt unit;
if i = 1 then
begin PR string(
4:= NULL then ERR(no assignment to proc ident+);+);
if ~b then PR string(42+)
end else if i = 2 then PR string(4:= Y+)
end; RESET reading ptrs;
if contents of[a + 6] = 1 then
PR string(4; protect:= false+)
end end;
PR nlc; PR string(4end +); if block then RE end;
if ptr to array segment $ 0 then
begin PR nlc; PR string(4; ERASE(fnn) end+ end
end Block exit;

6.3.4. Translation of a (formula or type) procedure declaration.

**Syntactic definition 1:**

<formula procedure declaration>::= formula <procedure heading>

<procedure body>

<procedure declaration>::= <procedure type><procedure heading>

<procedure body>

<procedure type>::= <empty> | <type>

<procedure heading>::= <procedure ident><formal parameter part>:

  <value part> <specification part>

<procedure ident>::= <procedure identifier> | <operator identifier>

<operator identifier>::= dyadic × dyadic / dyadic ↑ dyadic ↓

monadic + monadic - monadic ↑

constant + ! constant ×

For <empty>, <type>, <procedure identifier>, <value part> and <formal

parameter part> see the ALGOL 60 report [10] section 5.4.

**First part of algorithm:**

**procedure** procedure declaration;

begin integer nr of param, a, b, su, nsu, i, proc ic, ptr to integer param,

ptr to formula param;

Boolean formula procedure;

Boolean procedure Identifier in paramlist(p); integer p;

if nr of param = 0 then Identifier in paramlist:= false else

begin integer a, i; i:= nr of param; a:= begin;

for a:= a + contents of [a] while

nr of ident # contents of [a + 1] do

begin i:= i - 1; if i = 0 then

begin Identifier in paramlist:= false; goto OUT end

end; Identifier in paramlist:= true; p:= a;

if contents of [p + 3] & 32 ≠ 0 then ERR(

equal identifiers in specification part);
formula procedure:= \text{false}; su:= synt unit; nsu:= next synt unit;
if nsu = procedure symbol then
  begin if su = formula symbol then
    begin FR string(*integer*); RE; formula procedure:= true end
  else FR and RE
  end; FR and RE;
if first scan then
begin nr of param:= 0; proc ic:=
  STORE into information list(st(st(st(st(st(0,
  nr of ident),
  ptr to first letgit),
  (if nsu = procedure symbol then su else 0) × 13 + 32 ×
  procedure symbol +
  (if formula procedure then with local else 0)),
  0),
  nr of param),0));
if formula procedure then ptr to integer param:= proc ic else
ptr to integer param:= 0;
ptr to formula param:= 0;
begins= contents of [preceding begin]:= STEM into information list(st(st(st(st(st(st(st(st(st(st(st(st(st(st(st(st(0,
- begin symbol),
begin),
0),0),0),0),0),0),0),0));
preceding begin:= begin + 3; RE
end first scan else
begin begin:= contents of [preceding begin];
preceding begin:= begin + 3;
proc ic:= begin - 7;
nr of param:= contents of [proc ic + 5];
if operator identifier then
begin if \text{formula procedure}
  then ERR(*operator identifier at wrong place*);
begin
if nr of param ≠ 2 v contents of [proc ic + 20] ÷ 32 ≠
  formula symbol × 1024 v
  contents of [proc ic + 26] ÷ 32 ≠ formula symbol × 1024
  then ERR("operator identifier with wrong parameters")
  end else if (-nr of ident) = monadic symbol × 1024 +
  power symbol then

begin
if nr of param ≠ 2 v contents of [proc ic + 20] ÷ 32 ≠
  formula symbol × 1024 v
  contents of [proc ic + 26] ÷ 32 = formula symbol × 1024
  then ERR("operator identifier with wrong parameters")
end else if (-nr of ident) ÷ 1024 = monadic symbol then

begin
if nr of param ≠ 1 v contents of [proc ic + 20] ÷ 32 ≠
  formula symbol × 1024
  then ERR("operator identifier with wrong parameters")
end else if (-nr of ident) ÷ 1024 = constant symbol then

begin
if nr of param ≠ 1 v contents of [proc ic + 20] ÷ 32 =
  formula symbol × 1024 then ERR(
  "operator identifier with wrong parameters")
end; PR operator; RE

end else begin PR string(42£); PR and RE end
end;

if synt unit ≠ open symbol then
begin
  PR and RE; goto PROCEDURE BODY end;

FORMAL PARAMETER PART:

begin nr of param:= nr of param + 1;
STORE into information list(st(st(st(st(0,
  nr of ident),
  ptr to first letgit),
  specified as name),
  0),0));
end; PR string(42£); PR and RE;
if synt unit = comma symbol then goto FORMAL PARAMETER PART;
CHECK(close symbol); PR and RE; PR and RE semicolon;

begin
if first scan then contents of [proc :c + 5]:= nr of param;
if synt unit = value symbol then
begin
  VALUE PART: RE; if first scan then
begin if Identifier in param list(a) then
    contents of [a + 3]:= specified as value
else ERR(4 value part not O.K.)
end; RE; if synt unit = comma symbol then goto VALUE PART;
RE semicolon;
if second scan then
begin Boolean value symbol is printed;
    value symbol is printed:= false;
a:= begin; SAVE reading ptrs;
for i:= 1 step 1 until nr of param do
begin a:= a + contents of [a]; b:= contents of [a + 3];
    if b < t15 # formula symbol ^b - b < t32 x t32 =
specified as value then
begin if ^ value symbol is printed then
    FR string(t value #) else FR string(t,#);
value symbol is printed:= true;
SET reading ptrs on (contents of [a + 2]);
FR string(t#); FR synt unit
end end; RESET reading ptrs;
if value symbol is printed then FR string(t;#)
end end value part;

comen:  
As can be seen above, all parameters are placed in the information
list and their possible occurrence in the value part has been taken
account of.
During the translation of the specification part, described below, the
types of these parameters are filled in. If it is necessary to inspect
the type of the i-th parameter of the procedure whose identifier
occupies the IC with entry "a", we have to look into:
"contents of [a + t17 + t6 x (t1 - t1) + t3]".

Syntactic definition 2:

<specification part>::= <empty>|<specification list>
<specification list>::= <specification>|<specification list>|<specification>
<specification>::= <ordinary specification>
<formal procedure specification>
<ordinary specification>::= <specifier><identifier list>|
    formula <identifier list>|
    formula array <identifier list>
<formal procedure specification>::= <formal procedure specifier>
    <formal procedure segment list>
<formal procedure specifier>::= procedure|<type> procedure|
    formula procedure
<formal procedure segment list>::= <formal procedure segment>
    <formal procedure segment list>,
    <formal procedure segment>
<formal procedure segment>::= <identifier>
    <specification of parameters>
<specification of parameters>::= <empty>!(<type of parameters list>)
<type of parameters list>::= <type of parameter>
    <type of parameters list>,<type of parameter>
<type of parameter>::= <empty>!<specifier>|formula|formula value|
    formula array|formula procedure

For specifier, <type> and <identifier list> see the ALGOL 60 report [10]
section 5.4, from which a procedure as specifier should be swept out.

Example of a procedure heading:
  formula procedure P(a,b,c,d,e); value a; formula a,b;
  real c; procedure d; formula procedure e(formula value,formula,,,); 
with this procedure heading a call P(x,y,3.14,PRINT,P)" is possible.
The reason for introducing the specification of parameters for a
formal procedure parameter is that the compiler has to know of each
parameter of a procedure whether it is specified as formula and
whether, in this case, it is called by value. A parameter which is not
specified will be treated as to be not of type formula.

Note that a specification of parameters concerns the parameters of
the formal procedure with its identifier given in the formal procedure
segment only. Hence, "procedure a,b(formula);" means that the formal
procedure "a" does not have a parameter of type formula and that the
formal procedure "b" has one parameter of type formula which is called
by name.

Second part of algorithm;
SPECIFICATION PART:

if synt unit = procedure symbol v
next synt unit = procedure symbol v
(synt unit = real symbol v synt unit = integer symbol v
synt unit = Boolean symbol v synt unit = formula symbol)
then
begin integer su,nsu,nr of param; su:= synt unit;
nsu:= next synt unit;
if nsu = procedure symbol then
begin if su = formula symbol then
begin PR string(\text{integer} \, ); RE end else PR and RE
end;
PROCEDURE SEGMENT: PR and RE;
if first scan then
begin nr of param := 0;
if Identifier in paramlist(a) then contents of [a + 3]:= 
(if nsu = procedure symbol then su else 0) * t15 +
32 * procedure symbol + specified as name
else ERR(\text{formal procedure param not in formal param part});
end; PR string(424); PR and RE;
if synt unit = open symbol then
begin if second scan then
begin RE; SKIP text until(synt unit = close symbol); RE end else
begin integer t; TYPE OF PARAM: RE;
if synt unit = comma symbol v synt unit = close symbol then
t:= 0 else
if next synt unit = value symbol then
begin t:= synt unit * t15 + specified as value; RE; RE end
else if next synt unit = array symbol v
next synt unit = procedure symbol then
begin t:= synt unit * t15 + 32 * next synt unit +
specified as name; RE; RE
end else
begin t:= synt unit * t15 + specified as name; RE end;
b:= STORE into information list(st(0, t - 1));
nr of param:= nr of param + 1;
if nr of param = 1 then contents of[a + 4]:= b;
comment The type of the i-th parameter of a formal procedure
is equal to:
-1 - contents of contents of [a + 4] - 1 + i * 2],
where "a" defines the IC of this procedure;
if synt unit ≠ close symbol then goto TYPE OF PARAM;
contents of [a + 5] := nr of param; RE
end end specification parameters of specified formal procedure;
if synt unit = comma symbol then goto PROCEDURE SEGMENT;
FR and RE semicolon
end procedure specification else

if synt unit = array symbol v next synt unit = array symbol then
begin integer a,su,nsu;
su := synt unit; nsu := next synt unit;
if nsu = array symbol then
begin if su = formula symbol then
  begin FR string(4 integer 4); RE end else FR and RE
  end;

ARRAY IDENTIFIER: FR and RE;
if first scan then
begin if Identifier in paramlist(a) then
  contents of [a + 3] := contents of [a + 3] +
  (if nsu = array symbol then su else real symbol) * t15 +
  32 * array symbol
else ERR("Array param not in formal param list")
end; FR string(424); FR and RE;
if synt unit = comma symbol then goto ARRAY IDENTIFIER;
FR and RE semicolon
end array specification else

if is specifier(synt unit) then
begin integer a,b,su; su := synt unit;
if su = formula symbol then
begin FR string(4 integer 4); RE end else FR and RE;
OTHER IDENTIFIER:
if first scan then
begin if Identifier in paramlist(a) then
begin b := contents of [a + 3] := su * t15 + contents of [a + 3];
if b = formula symbol × t15 + specified as value then
begin contents of [a + 4]:= ptr to formula param;
    ptr to formula param:= a
end end
else ERR(4specified param not in form param list*)
end; PR string(4Z4); PR and RE;
if synt unit = comma symbol then
begin PR and RE; goto OTHER IDENTIFIER end;
PR and RE semicolon
end other specification;
if is specifier(synt unit) then goto SPECIFICATION PART;
if first scan then
begin i:= nr of param; a:= begin; su:= 0;
    for a:= a + contents of [a] while i > 0 do
begin i:= i - 1; b:= contents of [a + 3] + 32;
    if b = 0 v b = label symbol × 1G24 v
    b = switch symbol × 1024 then su:= 1
end end;
comment:
Third part of algorithm:
PROCEDURE BODY: in formula procedure body:=
in formula procedure body + 1;
if first scan then
begin contents of [begin + 4]:= ptr to integer param;
    contents of [begin + 5]:= ptr to formula param
end;
envelope of block(begin);
if first scan ∧ su = 1 then
begin if EVEN(contents of [proc 1c + 16] + 8) = -1
    then dangerous procedures:= true
end; in formula procedure body:= in formula procedure body - 1;
PR and RE semicolon
end procedure declaration;
6.4. Translation of a statement

Syntactic definition:

\[
\text{<statement>:} := \text{<optional label sequence><unlabelled statement>}
\]

\[
\text{<optional label sequence>:} := \text{<empty>|<label sequence>}
\]

\[
\text{<label sequence>:} := \text{<label>|	ext{<label sequence><label>}}
\]

\[
\text{<label>:} := \text{<identifier>|<unsigned integer>}
\]

\[
\text{<unlabelled statement>:} := \text{<conditional statement>|<unconditional statement>|<for statement>}
\]

Algorithms:

```
procedure optional label sequence(divisible, begin printed);
value divisible; Boolean divisible, begin printed;
begin Boolean label; label := begin printed := false;
A: if (synt unit = identifier v synt unit = integral number) then
next synt unit = colon symbol then
begin if synt unit = identifier then PR string(424);
if operator identifier then
begin ERR(operator identifier occurs as label); PR operator; RE
end else PR and RE;
PR and RE; block contains labels := label := true;
goto A
end;
if second scan then
begin if label " (dangerous inner block v dangerous procedures) then
begin if ~ divisible then
begin PR string(4begin #); begin printed := true end;
PR string(4 ERASE(snn); #)
end end end optional label sequence;
```

```
procedure statement(divisible); value divisible; Boolean divisible;
begin Boolean beg pr; optional label sequence(divisible, beg pr);
if synt unit = if symbol then
```
begin PR and RE; conditional statement end else
if synt unit = for symbol then
begin PR and RE; for statement end else
unconditional statement(beg pr v divisible);
if beg pr then PR string(\texttt{end })
end statement;

comment:

6.4.1. Translation of a conditional statement

Syntactic definition:

<conditional statement>::= <if statement>|
     <if statement> else <statement>|
     <if clause><optional label sequence><for statement>
<if statement>::= <if clause><optional label sequence>
    <unconditional statement>
<if clause>::= if <Boolean expression> then

Algorithm;

procedure conditional statement;
begin Boolean bp,pios; other expression; CHECK(then symbol);
    PR and RE;
onoptional label sequence(false,bp);
    if synt unit = for symbol then
    begin PR and RE; for statement; if bp then PR string(\texttt{end })
    end else
begin unconditional statement(bp); if bp then PR string(\texttt{end })
pios:= proc id ass stat; proc id ass stat:= false;
    if synt unit = else symbol then
begin PR and RE; statement(false) end;
pios:= proc id ass stat:= pios \wedge proc id ass stat
end end conditional statement;
6.4.2. Translation of a for statement

Syntactic definition:
See the ALGOL 60 report [10] section 4.6. Note that as
for-list-variable one may not have a formula variable, but one may
have an arithmetic variable only.

Algorithm:

procedure for statement;
begin integer a; if second scan then
begin if Search for identifier(a) then
begin a:= contents of[a + 3] ; t15;
if a = formula symbol then
ERR(4 for list element of type formula)
end end; other expression; CHECK(becomes symbol); FR and RE;
for list element: other expression;
if synt unit = step symbol then
begin FR and RE; other expression; CHECK(until symbol);
FR and RE; other expression
end else
if synt unit = while symbol then
begin FR and RE; other expression end;
if synt unit = comma symbol then
begin FR and RE; goto for list element end;
CHECK(do symbol); FR and RE; statement(false);
proc id ass stat:= false
end for statement;

comment:

6.4.3. Translation of an unconditional statement

Syntactic definition:

<unconditional statement>::= <compound statement>|<block>
<goto statement>|<assignment statement>|<procedure statement>
<dummy statement>
<compound statement>::= begin <compound tail>
<compound tail>::= <statement> end | <statement> | <compound tail>
<goto statement>::= goto <designational expression>
<dummy statement>::= <empty>

For designational expression see the ALGOL 60 report [10] section 3.5.

Algorithm;

procedure unconditional statement(divisible);
value divisible; Boolean divisible;
if synt unit = begin symbol then
begin if is declarator(next synt unit) then envelope of block(0) else
begin PR and RE begin; S: statement(true);
if synt unit ≠ semicolon symbol ¬ synt unit ≠ end symbol then
begin ERR("statement not appropriately closed");
SEEK(synt unit = semicolon symbol v
synt unit = end symbol)
end;
L: if synt unit = semicolon symbol then
begin PR and RE semicolon; goto L end;
if synt unit ≠ end symbol then
begin proc id ass stat:= false; goto S end;
PR and RE end
end end else
if synt unit = goto symbol then
begin block contains gotos:= true; PR and RE; other expression
end else
if synt unit = identifier ¬ (next synt unit = becomes symbol v
next synt unit = sub symbol) then
assignment statement(divisible) else
if synt unit = identifier then procedure statement(false) else
begin comment; dummy statement;; end unconditional statement;
comment:

6.4.4. Translation of an assignment statement

Syntactic definition:
<assignment statement>::= <ordinary assignment statement> | 
<formula assignment statement>
<ordinary assignment statement>::= <left part list>
<Bool or arith expression>
<formula assignment statement>::= <left part list><formula expression>
<formula assignment statement>::= <left part list> <left part list>
<left part>::= <left part> <left part list>
<formula assignment statement>::= <variable>: = | <procedure identifier>: = 
<Bool or arith expression>::= <Boolean expression> | 
<arithmic expression>

The left part list in a formula assignment statement may, of course, 
contain variables or procedure identifiers of type formula only.

Let fp1, fp2, and fp3 be formula procedure identifiers and f1, f2, f3 and 
f4 formula variables, then the translations of:


is as follows:

FP1 := FP2 := FP3 := ASSIGN(F1, ASSIGN(F2, ASSIGN(F3, ASSIGN(F4, EXPR))))

possibly followed by "protect := true".

With the protect statement, the whole is, dependent on the value of 
"divisible", surrounded by begin and end.

Note that as identifiers we may now encounter the special operator 
identifiers: dyadic + etc.

Algorithm:

procedure assignment statement(divisible);
value divisible; Boolean divisible;
if first scan then
begin L: if interested in proc id "--" proc id ass stat then 
begin if next synt unit = becomes symbol then 
proc id ass stat := nr of ident = contents of [procedure body-6]
end; operator identifier := false;
begin RE; if synt unit = sub symbol then other expression;
if synt unit = becomes symbol then 
begin RE; if synt unit = identifier ^
(next synt unit = becomes symbol \next synt unit = sub symbol)
then goto L;
other expression
end end first scan else
begin integer a,t,p,m,n,i,b,
ptr to first formula procedure identifier,prec ptr;
Boolean form ass stat,first,protection needed;
protection needed:= false; first:= true;
ptr to first formula procedure identifier:= 0; n:= 0;
L1: n:= n + 1; block depth:= 0;
if Search for identifier(a) then
begin m:= contents of[a + 3]; t:= m ↓ t15;
m:= m - t \times t15; p:= m ↓ 32; m:= m - p \times 32
end else t:= p:= m:= 0;
if first then
begin form ass stat:= t = formula symbol; first:= false;
if form ass stat then
begin SAVE reading ptrs; min block depth:= 0 end
end else if ~ (form ass+stat \geq t = formula symbol) then
ERR(4(left part list of assignment statement not O.K.));
if form ass stat then
begin if p = procedure symbol then
begin protection needed:= protection needed \vee m = with local;
if m = specified as name then ERR(
+assignment to parameter procedure$1$);  
comment In behalf of the procedure "formula expression" we
introduce the following two statements. The block depth is the
block depth of the procedure declaration.;
proc id ass stat:= true;
if min block depth > block depth then
min block depth:= block depth; if block depth = 0 then ERR(
+assignment to proc ident outside body$1$);
p:= begin; if in formula procedure body > 1 then
begin if m = with local \^ block depth < \(-1\) then
begin 111: b:= p \rightarrow 7; t:= contents of [b + 3];
if contents of [b] = 7 \^ t = formula symbol \times t15
+ procedure symbol \times 32 + with local then

block depth := block depth + 1;
if block depth < -1 then
begin p := contents of [contents of[p + 2] + 2]; goto 111
end end end;
if ptr to first formula procedure identifier = 0 then
ptr to first formula procedure identifier := (a × 1024 + n) × 4
else
contents of [prec ptr] :=
contents of [prec ptr] + (a × 1024 + n) × 4;
prec ptr := a + 6;
if contents of [prec ptr] ÷ 4 ≠ 0 then
begin contents of [prec ptr] :=
remainder(contents of [prec ptr], 4);
ERR(
	two equal form proc idents in lhs of ass stat
) end;

comment
The fourth element of the information cell of the formula
procedure identifier is used as link for a list of integers, and can
not be used as a temporary link in the list of left parts now being
formed. In a situation like:
"Formula procedure fp1 :=
begin formula procedure fp2 := fp1 := f :=
integer i ; i := lhs of fp2
end", we see clearly the danger.
Therefore, the sixth element is chosen, with the obligation to
save the two-bits information;
end else if m = declared as value
m = specified as value then ERR(assignment to value variable);
RE; if synt unit = sub symbol then
begin RE; SKIP text until(synt unit = bus symbol); RE end;
CHECK(becomes symbol); RE;
if synt unit = identifier then
begin if next synt unit = becomes symbol then goto L1;
if next synt unit = sub symbol then
begin SAVE reading ptrs; RE; RE;
  SKIP text until(synt unit = bus symbol);
  if next synt unit = becomes symbol then
    begin RESET reading ptrs; goto L1 end
  else RESET reading ptrs
  end end;

  RESET reading ptrs;
  if ptr to first formula procedure identifier ≠ 0 then
    begin if protection needed ^ ~ divisible then
      PR string(4BEGIN 4); SAVE reading ptrs;
      a := ptr to first formula procedure identifier ÷ 4;
      L2: m := a ÷ 1024;
      SET reading ptrs on (contents of[m + 2]);
      a := contents of [m + 3]; a := a ÷ 32 × 32;
      if a = with local then PR string(4Y#);
      if operator identifier then PR operator else
        begin if a = without local then PR string(4Z#);
          PR synt unit
        end;
      PR string(4:= #); a := contents of [m + 6] ÷ 4;
      if a ≠ 0 then goto L2; RESET reading ptrs
    end;
  end;
  a := ptr to first formula procedure identifier ÷ 4; p := 0; i := 1;
  L3: if a ≠ 0 then
    begin m := a ÷ 1024; t := a - m × 1024 -1;
      b := contents of [m + 6]; a := b ÷ 4;
      contents of [m + 6] := b - a × 4
    end else t := n;
  for i := i step 1 until t do
    begin PR string(4ASSIGN(4#)); p := p + 1; PR and RE;
      if synt unit = sub symbol then other expression;
      CHECK(becomes symbol); RE; PR string(4, #)
    end;
  end;
  if t < n then
    begin comment: Skip the procedure identifier assignment; RE; RE;
      i := t + 2; goto L3
    end;
  end;

formula expression;
for i := 1 step 1 until p do PR string(4); if protection needed then
begin PR string(4); protect := true ;
if ~ divisible then PR string(4
end
end form ass stat else
begin PR string(424); PR and RE;
if synt unit = sub symbol then other expression;
CHECK(becomes symbol); PR and RE;
if synt unit = identifier then
begin if next synt unit = becomes symbol then goto L1;
if next synt unit = sub symbol then
begin SAVE reading ptrs; RE; RE;
SKIP text until(synt unit = bus symbol);
if next synt unit = becomes symbol then
begin RESET reading ptrs; goto L1 end else
RESET reading ptrs
end end;
other expression
end end assignment statement;
comment:
6.4.5. Translation of a procedure statement
Syntactic definition:
(procedure statement)::= <procedure identifier>
<actual parameter part>
(actual parameter part)::= <empty>|(<actual parameter list>)
(actual parameter list)::= <actual parameter>|<actual parameter list> <parameter delimiter><actual parameter>
(actual parameter)::= <string>|<formula expression>|<Bool or arith expression>|<switch identifier>|
<array identifier>|<procedure identifier><Bool or arith expression>::= <Boolean expression>|
<arithmetic expression>
<parameter delimiter>::=,1)<letter string>:(
Algorithm;

procedure procedure statement(IRN necessary);
value IRN necessary; Boolean IRN necessary;
if first scan then
  begin if next synt unit ≠ open symbol then RE else
    begin RE; RE; SKIP text until(synt unit = close symbol); RE
  end end else
begin integer a,i,type,p,m,first param,nr of param,mode of proc;
Boolean piasa; i:= 0; block depth:= 3;
if Search for identifier(a) then
begin i:= contents of [a + 3];
  if i - i : 32 × 32 ≠ specified as name then
    begin mode of proc:= 1; first param:= contents of [a + 4] - 1;
      nr of param:= contents of [a + 5]
    end else
    begin mode of proc:= 2; first param:= a + 14;
      nr of param:= contents of [a + 5]
  end end else
begin if Search for standard identifier(nr of param) then
  mode of proc:= 3 else
begin ERR(4proc not declared);
  nr of param:= mode of proc:= 0
end end;

if IRN necessary then
begin i:= i : t15; if i = real symbol v (mode of proc = 3 ~
type of standard identifier = real symbol) then PR strins(4RN(v)
else if i = integer symbol v (mode of proc = 3 ~
type of standard identifier = integer symbol) then
PR strins(4IN(v) else if i = formula symbol then
begin if proc id ass stat then
  begin if block depth > min block depth v mode of proc = 1 then
    PR strins(4abs(v) else IRN necessary:= false
  end else IRN necessary:= false
end else IRN necessary:= false
end; piass:= proc id ass stat; proc id ass stat:= false;

if mode of proc ≠ 3 then PR string(474); PR and RE;

i:= 0; if synt unit ≠ open symbol then goto END;

L:= PR and RE; i:= i + 1; m:= 0;

if mode of proc = 1 then

begin if i ≤ nr of param then

m:= -1 - contents of [first param + i × 2]

end else

if i > nr of param ^ mode of proc ≠ 0 then

ERR(4 too much param in proc call:) else

if mode of proc = 2 then m:= contents of [first param + i × 6];

type:= m ÷ t15; m:= m - type × t15; p:= m ÷ 32;

m:= m - p × 32;

if p ≠ 0 then

begin if synt unit = identifier then

begin if next synt unit = comma symbol v next synt unit =

close symbol then

begin if type ≠ formula symbol then goto L1 else

if Search for identifier(a) then

begin if contents of [a + 3] ÷ 32 = type × 1028 + p then

goto L1

end end end; ERR(4 actual param not 0.K.); L1:= PR string(474); PR and RE

end else if type = formula symbol then

begin if m = specified as name then

begin if synt unit = identifier ^

(next synt unit = comma symbol

v next synt unit = close symbol) then

begin if Search for identifier(a) then

begin m:= contents of [a + 3];

if m ÷ 32 = type × 1028 then

begin if m - m ÷ 32 × 32 = specified as value

then PR string(4Y+) else PR string(4Z+);

PR and RE; goto OUT

end end

end else if synt unit = identifier ^ next synt unit =

sub symbol then
begin SAVE reading ptrs; RE; RE;
SKIP text until(synt unit = bus symbol);
if next synt unit = comma symbol v
next synt unit = close symbol then
begin RESET reading ptrs;
if Search for identifier(a) then
begin m:= contents of [a + 3];
if m > 32 = type × 1024 + array symbol then
begin PR string(424); PR end RE;
other expression; goto OUT
end end end else RESET reading ptrs
end end;
formula expression
end else other expression;
OUT: if synt unit = comma symbol then goto L;
CHECK(close symbol); PR and RE;
END: if mode of proc # 1 then
begin if i ≠ nr of param ^ mode of proc + 0 then
ENR(4nr of param not 0,K,4)
end; proc id ass stat:= plass; if IFN necessary then
PR string(4)4)
end procedure statement;

comment:

6.5. Translation of an expression

Syntactic definition:

<expression> ::= <simple expression> |
if <Boolean expression> then <simple expression> |
else <expression> |

<simple expression> ::= <simple formula expression> |

<simple formula expression> ::= <simple arithmetic expression> |

<simple arithmetic expression> ::= <simple designational expression> |

Boolean-, arithmetic-, designational- and formula expressions are
expressions in which the simple expressions are chosen of the
appropriate type.

Algorithm;

procedure other expression;
if synt unit = if symbol then
begin PR and RE; other expression; CHECK(then symbol); PR and RE;
simple other expression; CHECK(else symbol); PR and RE;
other expression
end else simple other expression;

procedure formula expression;
begin Boolean piass; piass := proc id ass stat;
if synt unit = if symbol then
begin PR and RE; proc id ass stat := false;
other expression; CHECK(then symbol); PR and RE;
proc id ass stat := piass; simple formula expression(false);
CHECK(else symbol); PR and RE; proc id ass stat := piass;
formula expression
end else simple formula expression(false)
end formula expression;

comment:

A simple other expression is syntactically defined in the ALGOL 60
report [10] chapter 3, with the following addition:
as primary in a simple arithmetic expression a formula enquiry, a
length enquiry and a type enquiry are allowed. These enquiries are
syntactically defined as follows:
<formula enquiry>::= <kind of enquiry> of <formula name>
<type enquiry>::= type of <formula name>
<length enquiry>::= length of <formula name>
<kind of enquiry>::= has|has|el <arithmetic expression>
<formula name>::= <formula primary>

Moreover, as primaries in a boolean expression, the following
constructions are allowed:
constant <formula name>,
monadic <formula name>,
dyadic <formula name>,
polyadic <formula name>,
rowadic <formula name>.
These are called type category enquiries.

Algorithm:

procedure enquiry;
begin integer s; s := synt unit; RE;
if s = el symbol then
begin PR string(EL(s)); other expression; PR string(,a) end else
if s = lhs symbol then PR string(LHS(s)) else
if s = rhs symbol then PR string(RHS(s)) else
if s = type symbol then PR string(TYPE(s)) else
if s = length symbol then PR string(LENGTH(s)) else
ERR(error in enquiry); CHECK(of symbol); RE;
if first scan then other expression else
simple formula expression(true);
PR string(a)
end enquiry;

comment:

6.5.1. Translation of a simple other expression

Algorithm:

procedure simple other expression;
begin if synt unit = identifier then
begin if next synt unit = open symbol then procedure statement(false)
else
begin integer a,t,p,m,br; br := 0;
if second scan then
begin if Search for identifier(a) then
begin m := contents of [a + 3]; t := m \ t15;
m := m - t \ t15; p := m \ 32; m := m - p \ 32;
end else if Search for standard identifier(a) then
begin t := 0; p := procedure symbol; m := standard identifier
end else t := p := m := 0;
if t = formula symbol then
begin if p = procedure symbol v (p = 0 ^ (m = declared as value
v m = specified as value v m = specified as name)) then
begin PR string(4abs(4)); br := 1 end;
if p = array symbol v p # procedure symbol then
m = declared as name then
begin PR string(4V(4)); br := br - 1 end else
if m = specified as name then p # procedure symbol then
begin PR string(4VN(4)); br := br + 1 end
end; if t = formula symbol ^ p = 0 ^ m = specified as value
then PR string(4#)
else if m # standard identifier then PR string(4Z)
end else a := t := p := m := 0; PR and RE;

if synt unit = sub symbol then
begin if second scan then
begin if t = formula symbol then
begin if p # array symbol then ERR(4
formula array not declared#)
end end;
L: PR and RE;
other expression; if synt unit = comma symbol then goto L;
CHECK(bus symbol); PR and RE
end; for br := br - 1 while br > 0 do PR string(4#)
end variable; goto begin

if else
if is enquiry then
begin if synt unit = lhs symbol v
synt unit = rhs symbol v synt unit = el symbol
then PR string(4AR(4)); enquiry; goto begin
end else
if is adic symbol(synt unit) then
begin if second scan then
begin PR int num(synt unit - constant symbol);
PR string(4# TYPE CAT(4)); RE; simple formula expression(true);
PR string(4#)
end else RE; goto begin
end else
if synt unit = open symbol v synt unit = sub symbol then
begin Boolean open; open:= synt unit = open symbol;
L: PR and RE; other expression;
if synt unit = comma symbol then goto L;
if open then CHECK(close symbol) else CHECK(bus symbol);
FR and RE; goto beginend else
begin integer a; for a:= semicolon symbol,end symbol,else symbol,
then symbol,comma symbol,close symbol,bus symbol,step symbol,
until symbol,do symbol,while symbol,colon symbol,becomes symbol,
of symbol do if synt unit = a then goto END;
PR and RE; goto begin;
END:
end simple other expression;

procedure translate string;
begin integer m,n,i,p,T; m:= n:= T:= 0;
p:= ptr of text2 - 3; i:= ptr of text1 - 1; PR string(4$STRING($);
L: i:= i + 1; if take from text array(i) $ smaller than symbol then
goto L;
for i:= i + 1 step 1 until p do
begin n:= n + 1; T:= T * 256 + take from text array(i) + 1;
if n = 3 v i = p then
begin PR string(4$STRING($); PR int num(T);
PR string(4$STRING($); T:= n:= 0; m:= m + 1
end end;
PR string(4$STRING($); for i:= 1 step 1 until m do PR string(4$STRING($);
PR string(4$STRING($); PR int num(m); PR string(4$STRING($)
end translate string;

column:
The translation of "$4012345674$" is:
"$STRING(st(66051, st(263430, st(1800,0))),3)"$,
where
66051 = ((0+1)*256+(1+1))*256+(2+1),
263430 = ((3+1)*256+(4+1))*256+(5+1),
1800 = (6+1)*256+(7+1).
6.5.2. Translation of a simple formula expression

**Syntactic definition:**

A simple formula expression resembles highly a simple arithmetic expression. As formula primaries we may have all the arithmetic primaries, a string, and a formula base (see below). As in an arithmetic expression, the formula-, type- and length enquiries are also possible.

As the syntactic definition of a formula expression is, apart for the primary, identical to the definition of an arithmetic expression, we define a formula primary only:

\[
<\text{formula primary}> ::= \text{unsigned number} \\
   \quad <\text{variable}>\langle\text{function designator}\rangle(\langle\text{formula expression}\rangle) \\
   \quad <\text{formula variable}>\langle\text{formula enquiry}\rangle\langle\text{type enquiry}\rangle \\
   \quad <\text{length enquiry}>\langle\text{string}\rangle\langle\text{formula base}\rangle
\]

\[
<\text{formula base}> ::= \text{constant}(\langle\text{type}\rangle,\langle\text{int arith expr}\rangle,\langle\text{int arith expr}\rangle) \\
   \quad \text{monadic}(\langle\text{type}\rangle,\langle\text{formula expression}\rangle,\langle\text{int arith expr}\rangle) \\
   \quad \text{dyadic}(\langle\text{type}\rangle,\langle\text{formula expression}\rangle,\langle\text{formula expression}\rangle) \\
   \quad \text{polyadic}(\langle\text{type}\rangle,\langle\text{int id}\rangle,\langle\text{length}\rangle,\langle\text{formula expression}\rangle) \\
   \quad \text{rowadic}(\langle\text{type}\rangle,\langle\text{int id}\rangle,\langle\text{length}\rangle,\langle\text{int arith expr}\rangle)
\]

\[
<\text{type}> ::= \langle\text{int arith expr}\rangle \\
   \langle\text{int id}\rangle ::= \langle\text{identifier}\rangle \\
   \langle\text{length}\rangle ::= \langle\text{int arith expr}\rangle
\]

Int arith expr is an arithmetic expression delivering an integer.

It is necessary that

1. The value \( v \) of type satisfies: \( 0 \leq v \leq 30. \)

2. The value \( v \) of the right-hand side of the int arith expr in a constant formula satisfies: \( \text{abs}(v) < 2^{17} - 1. \)

3. The value \( v \) of length satisfies: \( v \geq 0. \)

Algorithm:
procedure simple formula expression(primary only);
value primary only; Boolean primary only;
begin integer if,primary symbol,int num,f.nr of brackets;
integer array F1,Pt,Pr[1:100]; Boolean scan1;

comment:
In a first pass (scan1 = true), the expression is scanned by means
of the procedures "elevator" and "primary" during which a syntactical
tree is formed and stored into the arrays "F1", "Pt" and "Pr", which
have a pointer "if".
In a second pass (scan1 = false), output is produced by a call of "PR
form expr", which, by using the procedure "primary" forms the output
climbing through the syntactical tree.
If the expression is the right-hand side of an assignment to a
procedure identifier, in which case "proc id ass stat" has the value
true, a special translation, involving the abs function in front
of some formula variables and some formula function designators, is
performed. As soon as this variable or function designator is shielded
by another procedure, the "abs" is suppressed as a result of the
statement: "proc id ass stat:= false" in "PR form expr".
Concerning the treatment of brackets we remark that "PR form
expr" skips leading opening brackets and trailing closing brackets.
Therefore, "primary" starts reading, during the second pass, a
non-opening bracket symbol, and "RE" in "PR form expr", reads an
operator symbol (except when it skips brackets).
Special attention is given to the primary: "(if ... then ... else ...
... )". The translation is a translation from infix notation to prefix
notation.;

integer procedure elevator(floor); value floor; integer floor;
if floor = 0 then elevator:= primary else
begin integer el,s;
if floor = 3 then
begin s:= symt unit; if s = plus symbol v s = minus symbol then
begin RE; el:= elevator(2); IF:= IF + 1;
if IF > 100 then fatal err("formula expression too long");
F1[IF]:= el; Pt[IF]:= s + 102; el:= IF;
goto again
end; e1 := elevator(floor - 1);
again: s := synt unit;
if floor = 3 \land (s = plus symbol \lor s = minus symbol) \lor
floor = 2 \land (s = times symbol \lor s = over symbol) \lor
s = int div symbol) \lor
floor = 1 \land (s = power symbol) then
begin RE; iP := iP + 1; if iP > 100 then Fatal ERR(
+ formula expression too long +);
FI[iP] := e1; e1 := iP; Pt[e1] := s;
Fr[e1] := elevator(floor - 1); goto again
end; elevator := e1
end; elevator;

procedure PR form expr(f); value f; integer f;
begin procedure skip opens;
for synt unit := synt unit
while synt unit = open symbol do
begin nr of brackets := nr of brackets + 1; RE end;
skip opens;
if f = primary symbol \lor f = int num then primary else
begin integer l, t, r; l := FI[f]; t := Pt[f]; r := Fr[f];
proc id ass stat := false;
if t > 1024 then
begin t := t - 1024;
if t = plus symbol then PR string(+PT(4))
else PR string(+PT(4));
RE; PR form expr(l); PR string(+PT(4))
end else
if t = power symbol \land r = int num then
begin PR string(+PT(4)); PR form expr(l);
PR string(+PT(4)); RE; skip opens; PR and RE; PR string(+PT(4))
end else
begin if t = plus symbol then PR string(+PT(4)) else
if t = minus symbol then PR string(+PT(4)) else
if t = times symbol then PR string(+PT(4)) else
if t = over symbol then PR string(+PT(4)) else
if t = int div symbol then PR string(+PT(4)) else
if t = power symbol then PR string(+PT(4));
PR form expr(1); PR string(4,4); RE; PR form expr(r);
PR string(4,4)
end end; for synt unit:= synt unit
while synt unit = close symbol " nr of brackets > 0 do
begin nr of brackets:= nr of brackets - 1; RE end
end PR form expr;

integer procedure primary;
begin primary:= primary symbol;
if is adic symbol(synt unit) then
begin if scmi then
begin RE; RE; SKIP text until(synt unit = close symbol); RE
end else
begin integer s; s:= synt unit; RE;
CHECK(open symbol); RE; proc id ass stat:= false;
if s = polyadic symbol v s = rowadic symbol then
begin PR string(4STORE ROW(4));
if s = polyadic symbol then PR string(496,4)
else PR string(4128,4); other expression;
CHECK(comma symbol); PR and RE; CHECK(identifier);
PR string(424); PR and RE; CHECK(comma symbol);
PR and RE; other expression; CHECK(comma symbol);
PR and RE;
if s = polyadic symbol then formula expression else
other expression; CHECK(close symbol); PR and RE
end else
begin PR string(4STORE 4);
if s = constant symbol then PR string(4CONST(4)) else
if s = monadic symbol then PR string(4MONADIC(4)) else
if s = dyadic symbol then PR string(4DYADIC(4));
other expression;
CHECK(comma symbol); PR and RE;
if s = constant symbol v s = monadic symbol then
other expression else formula expression;
CHECK(comma symbol); PR and RE;
if s = constant symbol then other expression
else formula expression; CHECK(close symbol); PR and RE
end end end else
if synt unit = integral number v synt unit = real number then
begin if ~ scan1 then
begin if synt unit = integral number
then PR string(4IN(+)) else PR string(4RN(+));
PR and RE; PR string(4+) end else
begin if synt unit = integral number
then primary:= int num; RE
end end else
if synt unit = identifier then
begin if scan1 then
begin if next synt unit = open symbol v next synt unit =
sub symbol then
begin RE; RE; SKIP text until(
synt unit = close symbol v synt unit = bus symbol)
end; RE end else
if next synt unit = open symbol then procedure statement(true)
else
begin integer a,t,p,m,br; br:= 1; block depth:= 0;
if Search for identifier(a) then
begin m:= contents of [a + J]; t:= m \ t15; m:= m - t x t15;
p:= m \ 32; m:= m - p \ 32;
end else
if Search for standard identifier(a) then
begin t:= real symbol; p:= procedure symbol;
m:= standard identifier
end else
begin ERR(4id not declared in form expr+);
t:= p:= m:= 0
end; if t = formula symbol then
begin if proc id ass stat then
begin if block depth > min block depth ^ (p = procedure
symbol v m = declared as value v m = specified as value v
m = specified as name) v (p = procedure symbol ^
m = specified as name) then
begin PR string(4tabs(+); br:= 2 end
end;
if m = declared as name ^ p # procedure symbol v
  p = array symbol then PR string(4V(+)) else
  if m = specified as name ^ p # procedure symbol then
  PR string(4VN(+)) else br:= br - 1
end else if t = real symbol then PR string(4RN(+))
else if t = integer symbol then PR string(4IN(+)) else br:= 0;
if t = formula symbol ^ p = 0 ^ m = specified as value
then PR string(4Y(+)) else
if m # standard identifier then PR string(4Z(+)); PR and RE;
if synt unit = sub symbol then
  begin if p # array symbol then ERR(
    "form array not declared");
    L: PR and RE; other expression;
    if synt unit = comma symbol then goto L;
    CHECK(bus symbol); PR and RE
  end; for br:= br - 1 while br > 0 do PR string(4Y(+))
end end else

if synt unit = string then
begin if ~ scan1 then translate string; RE end else
if is enquiry then
begin if scan1 then
begin if synt unit = el symbol then
  begin RE; SKIP text until(synt unit = of symbol) end
else RE; CHECK(of symbol); RE;
if synt unit = open symbol then
  begin RE; SKIP text until(synt unit = close symbol); RE
end else primary:= primary
end else
if synt unit = type symbol v synt unit = length symbol
then begin PR string(4IN(+); enquiry; PR string(4Y(+)) end
else enquiry
end else
if ~ scan1 ^ synt unit = if symbol then
begin PR string(4+); formula expression; PR string(4+); end else

if synt unit = open symbol then
begin RE; if synt unit = if symbol then
SKIP text until(synt unit = close symbol) else
begin primary:= elevator(3); CHECK(close symbol) end; RE
end else ERR(4+primary in formula expr not O.K.4)
end primary;

comment: Finally the procedure body of simple formula expression;
scan1:= true; iF:= 0; primary symbol:= -1;
int num:= -2; nr of brackets:= 0;
SAVE reading ptrs;
f:= elevator(if primary only then 0 else 3);
scan1:= false; RESET reading ptrs;
PR form expr(f)
end simple formula expression;

comment:
The body of envelope of block;; block or statement
end envelope of block and all the syntar-translating procedures.;

comment:

6.6. Auxiliary equipment

6.6.1. Declaration and initialization of symbols

Algorithm:

integer symbol,underlining symbol,space symbol,bar symbol,tab symbol,
nlor symbol,go to symbol,if symbol,then symbol,else symbol,for symbol,
do symbol,step symbol,until symbol,while symbol,comment symbol,
begin symbol,end symbol,Boolean symbol,integer symbol,real symbol,
array symbol,switch symbol,procedure symbol,string symbol, label
symbol,value symbol,formula symbol,type symbol,lhs symbol, rhs
symbol,el symbol,of symbol,constant symbol,monicic symbol, dyadic
symbol, polyadic symbol, rowadic symbol, length symbol, int div
symbol, unequal symbol, power symbol, becomes symbol, other symbol, plus
symbol, minus symbol, times symbol, over symbol, smaller than
symbol, equal symbol, greater than symbol, and symbol, comma
symbol, point symbol, lower ten symbol, colon symbol, semicolon
symbol, open symbol, close symbol, sub symbol, bus symbol, numb of und
symbols, string, identifier, integral number, real number, prec symbol;
integer array und symbol[1:350], ref1 to und symbol[30:105],
ref2 to und symbol[129:179],adic op[1:5,0:2];

procedure INITIALIZE symbols;
begin integer j,k,p,hash,ptr;
procedure rs(symbol); integer symbol; symbol:= text symbol;
procedure rs(symbol); integer symbol;
begin j:= j + 1; symbol:= j + 128 end;

procedure ru(symbol); integer symbol;

comment
This procedure puts the symbols of the underlined word delimiters,
together with the number of those symbols and the internal
representation (the value of j), in the array "und symbol". For each
underlined word delimiter a hash h is calculated and the index of
the delimiter where it is placed in "und symbol", is put in "ref1 to
und symbol[h]". Moreover, in "ref2 to und symbol[s]", where s is the
internal representation, the same index is put. In the higher syntax
reading procedures the internal representations are used in the form
of "goto symbol", "end symbol" etc. See also "Read synt unit";
begin integer i,s; i:= 0; j:= j + 1;
for s:= text symbol while s # semicolon symbol do
if s = underlining symbol then
begin s:= text symbol;
if s # underlining symbol then
begin i:= i + 1; und symbol[ptr + i + 1]:= s end
end;
symbol:= und symbol[ptr]:= j + 128;
und symbol[ptr + 1]:= i;
hash:= if i = 1 then und symbol[p:r + 2] else
und symbol[p + 2] + und symbol[ptr + i + 1] × 2;
if hash < 30 or hash > 105 then
FATAL ERR("und symbol in string incorrect");
und symbol[ptr + i + 2]:= ref1 to und symbol[hash];
ref1 to und symbol[hash]:= ref2 to und symbol[symbol]:= ptr;
ptr:= ptr + i + 3
end ru;

integer procedure text symbol;
begin p:= p + 1; text symbol:= STRING SYMBOL(p,
	\| \ +x<=->,\[$,\};\]
goto; go to; if; then; else; for; do; step; until; while; comment;
begin; end; Boolean; boolean; integer; real; array; switch;
procedure; formula; string; label; value;
\(<\); \>=; \[\]; true; false;
type; lhs; length; rhs; el; of; constant; monadic; dyadic;
polyadic; rowadic;
\} entail
end text symbol;

for j:= 30 step 1 until 105 do ref1 to und symbol[j]:= 0;
ptr:= 1; j:= 0; p:= -1;
rs(underlining symbol); rs(space symbol); rs(bar symbol);
rs(tab symbol); rs(plus symbol); rs(minus symbol);
rs(times symbol); rs(over symbol); rs(smaller than symbol);
rs(equal symbol); rs(greater than symbol); rs(and symbol);
rs(comma symbol); rs(point symbol); rs(lower ten symbol);
rs(colon symbol); rs(semi colon symbol); rs(open symbol);
rs(close symbol); rs(sub symbol); rs(bus symbol); rs(nlcr symbol);
k:= ptr; ru(goto symbol); ru(goto symbol);
und symbol[k]:= goto symbol; ru(if symbol); ru(then symbol);
ru(else symbol); ru(for symbol); ru(do symbol); ru(step symbol);
ru(unti1 symbol); ru(while symbol); ru(comment symbol);
ru(begin symbol); ru(end symbol); k:= ptr;
ru(Boolean symbol); ru(Boolean symbol);
und symbol[k]:= Boolean symbol; ru(integer symbol);
ru(real symbol); ru(array symbol); ru(switch symbol);
ru(procedure symbol); ru(formula symbol); ru(string symbol);
ru(label symbol); ru(value symbol); ru(int div symbol);

comment the following symbols are read: ≤, ≥, ≡, ⊥,
true, false;
for k:= 1,2,3,4,5,6 do ru(symbol);
ru(type symbol); ru(lhs symbol); ru(length symbol); ru(rhs symbol);
ru(el symbol); ru(of symbol); ru(constant symbol);
ru(monadic symbol); ru(dyadic symbol); ru(polyadic symbol);
ru(rowadic symbol); numb of und symbols:= j;
ra(power symbol); ra(becomes symbol); ra(other symbol); ra(string);
ra(identifier); ra(integral number); ra(real number);
fill alf
end INITIALIZE symbols;

procedure fill alf;
comment
This procedure is executed once, in order to define the array alf.
This array is being used in the process to search an identifier of
a standard procedure. All standard identifiers begin with lower case
letters with internal representation from 10 to 36. One should compare
the procedures: "Search for standard identifier" and "standard symbol"
of the next section. The absolute value of alf[s] determines for a
certain letter, with internal representation s, the entry in the
string of standard identifiers where the identifiers with this letter
as first one are placed.;
begin integer i,k,s; boolean lc; k:= -1;
for i:= 10 step 1 until 36 do alf[i]:= s4;
cyc1: lc:= true;
k:= k + 1; s:= standard symbol(k);
if s = 255 then goto out;
if s = plus symbol then
begin lc:= false; i:= k; k:= k + 1;
s:= standard symbol(k)
end;
if s < 10 v s > 36 then
ERR(4list of standard symbols not correct");
alf[s]:= if lc then k else - i; i:= s;
2053  cyc2: k := k + 1; s := standard symbol(k);
2054  if s = over symbol then goto cyc1;
2055  if s = plus symbol then
2056  begin lo := false; alf[i] := - alf[i] end;
2057  goto cyc2;
2058  out:
2059  end fill alf;
2060
2061  comment:
2062
2063  6.6.2. The text reading equipment
2064
2065  Algorithm:
2066
2067  integer ptr of text, ptr of text1, ptr of text2, synt unit,
2068  next synt unit, line number, line number1, additional synt unit,
2069  line number2, nr of stringquotes, delimiter array ptr,
2070  ptr to first letgit, nr of ident, nr of begins, edic sym;
2071  Boolean from delimiter array;
2072  integer array text array[0:10000], alf[1:36], delimiter array[1:30];
2073
2074  procedure put in text array(s); value s; integer s;
2075  begin integer i, j; ptr of text := ptr of text + 1;
2076  if ptr of text > 29999 then Fatal Err;
2077  \#program text too long$\$;
2078  i := (ptr of text - 1) \div 3;
2079  j := ptr of text - i \times 3;
2080  text array[i] :=
2081  if j = 1 then s \times 160000 else
2082  if j = 2 then text array[i] := 160000 \times 160000 + s \times 400 else
2083  text array[i] := 400 \times 400 + s
2084  end;
2085
2086  integer procedure take from text array(p); value p; integer p;
2087  begin integer i, j, t; i := (p - 1) \div 3; j := p - i \times 3;
2088  t := text array[i];
2089  if j = 1 then i := t := 160000 else
2090  if j = 2 then begin t := t - t + 160000 \times 160000; t := t \times 400 end
else t:= t - 400 x 400;
take from text array:= t
end;

procedure NS;
if first scan ^ reading allowed then
begin prec symbol:= symbol;
  comment
  The symbols are normally read with RESYM. If, however, an erroneous
  parameter delimiter has been read of the form ")abc:1(", then the
  symbols "abc:1" have been put in the array "delimiter array" and the
  boolean "from delimiter array" has been made equal to true. See also
  READ synt unit.;

if from delimiter array then
begin delimiter array ptr:= delimiter array ptr + 1;
symbol:= delimiter array[delimiter array ptr];
if symbol = -1 then
begin from delimiter array:= false; symbol:= RESYM end
end else symbol:= RESYM;
put in text array(symbol);

if symbol = semicolon symbol ^
prec symbol = underlining symbol then
early end of program;
comment
If the symbol "," is read, the procedure "early end of program"
is called to provide for as many string quotes and end's that are
necessary to appropriately close the text of the ABC ALGOL program
under consideration.;
if symbol = nlcr symbol then
begin line number2:= line number2 + 1;
NLCR; ABSPX:(4,0,line number2)
end else FRSLM(symbol)
end else
begin ptr of text:= ptr of text + 1;
symbol:= take from text array(ptr of text);
if symbol = nlcr symbol then line number2:= line number2 + 1
end NS;
procedure early end of program;

begin integer i,p; p:= ptr of text:= ptr of text - 2;

for i:= 1 step 1 until nr of stringquotes do

begin put in text array(bar symbol);

put in text array(greater than symbol)
end;

put in text array(semicolon symbol);

put in text array(semicolon symbol);

for i:= 1 step 1 until nr of begins do

put in text array(end symbol);

reading allowed:= false;

ERR(4program text contains \\); ptr of text:= p

end early end of program;

integer array und symbol read[1:9];

integer procedure READ synt unit;

begin if is lay out(symbol) then

begin L8: if symbol = space symbol then

ptr of text:= ptr of text - 1;

NS; if is lay out(symbol) then goto L8
end;

line number1:= line number2;

if symbol = underlining symbol then

begin integer i,j,k,hash,nr of letters,ptr; i:= 0;

L; ptr of text:= ptr of text - 1; NS;

if ~ reading allowed then

begin READ synt unit:= other symbol; goto endprog end;

if symbol = underlining symbol then goto L;

i:= i + 1; if i > 9 then

begin ptr of text:= ptr of text - 1; NS;

if symbol = underlining symbol then goto L;

i:= other symbol; goto OUT
end;

und symbol read[i]:= symbol;

ptr of text:= ptr of text - 1; NS;

if symbol = underlining symbol then goto L;
hash := \( \text{if } i = 1 \text{ then und symbol read}[1] \text{ else und symbol read}[1] + 2 \times \text{und symbol read}[1]; \)

\( \text{if hash < 30 \lor hash > 105 \text{ then ptr := 0 else ptr := und symbol}[hash];} \)

\begin{itemize}
\item From the hash of the underlined symbol read, the place of this symbol in "und symbol" is determined and it is checked whether the symbol has been written correctly;\end{itemize}

cyc: if ptr = 0 then

\begin{itemize}
\item begin i := other symbol; \text{goto OUT end;}
\item nr of letters := und symbol[ptr + 1];
\item if nr of letters = 1 then
\item begin j := ptr + 1;
\item for k := 1 step 1 until i do
\item \text{if und symbol read}[k] \neq \text{und symbol}[j + k] \text{ then goto next;}
\item i := und symbol[ptr]; \text{goto OUT next;}
\item end;
\item next: ptr := und symbol[ptr + nr of letters + 2]; \text{goto cyc;}
\item OUT: READ synt unit := 1; \text{ptr of text} := \text{ptr of text} - 1;
\item comment
\item Note that the internal representation, determined by "und symbol[ptr]", is put in the text array. Not the symbols. This makes the reading of the text during the second scan much more rapid;\end{itemize}

\text{put in text array(i); put in text array(symbol);}

\text{endprog;}

\begin{itemize}
\item if symbol = colon symbol then
\item begin NS; if symbol = equal symbol then
\item begin READ synt unit := becomes symbol;
\item ptr of text := ptr of text - 2;
\item put in text array(becomes symbol); NS end else
\item HEAD synt unit := colon symbol
\item end else
\item end else
\item end else
\item if symbol = bar symbol then
\item begin NS; if symbol = equal symbol then
\item end NS; if symbol = equal symbol then
\item end else
begin NS; READ synt unit:= unequal symbol end else
if symbol = and symbol then
begin NS; READ synt unit:= power symbol end else
if symbol = smaller than symbol then
begin integer s; s:= symbol; nr of stringquotes:= 1;
L2: NS; if s = bar symbol then
begin if symbol = smaller than symbol then
nr of stringquotes:= nr of stringquotes + 1 else
if symbol = greater than symbol then
begin nr of stringquotes:= nr of stringquotes - 1;
if nr of stringquotes = 0 then goto L3 end
end; s:= symbol; goto L2;
L3: NS; READ synt unit:= string
end else READ synt unit:= other symbol
end else
if is letter(symbol) then
begin integer i,j,k,ptr,full ptr,A,B,nls in ident;
comment
An identifier will be read now. It is compared with existing
identifiers, which are stored as a binary tree in the upper part of
the information list. The text array will contain the identification
number of the identifier plus the new line symbols occurring in the
identifier, but not the letters and digits.
If the identification number is larger than 218, three "symbols"
will be put in the text array: 399 and two derived from this number.;
A:= symbol + 1; i:= 1; j:= ptr to name list;
READ synt unit:= identifier;
ptr of text2:= ptr of text; nls in ident:= 0;
if line number 1 ≥ 2000 then FATAL ERR(4too much lines$);
ptr to first legit:= ptr of text × 2000 +
line number;
L1: NS; if is lay out(symbol) then
begin if symbol = nlcr symbol then
nls in ident:= nls in ident + 1;
goto L1
end;
if is letter(symbol) v is digit(symbol) then
begin i := i + 1; if i = 5 then
begin contents of [j] := A; A := 0;
j := j - 1; if j - 7 <= ptr of inf list then
FATAL ERR('inf list too small');
i := 1
end;
A := A x 64 + symbol + 1; goto L1
end;
contents of [j] := A x 64 ^ (4 - i);
if j - 7 <= ptr of inf list then
FATAL ERR('inf list too small');
ptr := ptr to first ident;
for i := ptr while ptr != 0 do
begin k := ptr to name list;
next: A := contents of [i]; B := contents of [k];
if A = B then
begin if A < 0 then goto FOUND else
begin i := i - 1; k := k - 1; goto next end
end;
k := if abs(A) > abs(B + .5) then 1 else 2;
L2: if A > 0 then
begin i := i - 1; A := contents of [i]; goto L2 end;
fill ptr := i - k;
ptr := contents of [i - k]
end;

comment We treat a new identifier; i := j;
contents of [i - 1] := contents of [i - 2] := 0;
contents of [i - 3] := nr of identifiers := nr of identifiers + 1;
if nr of identifiers > max nr of identifiers then
FATAL ERR('too much identifiers');

code table [nr of identifiers] := ptr to name list;
if ptr to first ident = 0 then ptr to first ident :=
ptr to name list else
contents of [fill ptr] := ptr to name list;
ptr to name list := i - 4;
FOUND: nr of ident := contents of [i - 3];
ptr of text := ptr of text2 - 1;
if nr of ident ≤ 218 then
    put in text array (nr of ident + 180) else
    begin put in text array(399);
    put in text array(nr of ident ≤ 400);
    put in text array(nr of ident ≤ 400 × 400)
    end;
for i:= 1 step 1 until nls in ident do
    put in text array(nlcr symbol);
    put in text array(symbol);
end else

if symbol > 180 then
    begin READ synt unit:= identifier; ptr of text2:=
    ptr to first letgt:= ptr of text;
    if symbol = 399 then
    begin NS; nr of ident:= symbol × 400;
    NS; nr of ident:= nr of ident + symbol
    end else nr of ident:= symbol - 180;
    NS
    end else

if is digit(symbol) v symbol = point symbol v
    symbol = lower ten symbol then
    begin Boolean integer; integer:= true;
    if 5: if symbol = point symbol then integer:= false else
    if symbol = lower ten symbol then
    begin L51: NS; if is layout(symbol) then goto L51;
    integer:= false
    end else if ~ (is layout(symbol) v is digit(symbol))
    then goto L52; NS;
    goto L5; L52:
    READ synt unit:= if integer then
    integral number else real number
    end else

if symbol = close symbol then
    begin integer p,l,nls in delimiter; if first scan then
    begin p:= ptr of text; l:= linenum2;
nls in delimiter:= 0
end;
l6: NS; if is letter(symbol) then
    begin comment
        Tentatively, a parameter delimiter will be read. If it turns out
        that the delimiter is not well formed an error message is given and
        the reading ptrs are set back to the beginning after the insertion of
        a semicolon symbol which most probably has been forgotten.;
        procedure store(s); value s; integer s;
        begin if delimiter array ptr < 29 then
            begin delimiter array ptr:= delimiter array ptr + 1;
                delimiter array[delimiter array ptr]:= s
            end;
            if s = nlcr symbol then
                nls in delimiter:= nls in delimiter + 1
            end;
            delimiter array ptr:= 0;
        end;
        L7: store(symbol); NS;
        if is letter(symbol) v is digit(symbol) v
        is lay out(symbol) then goto L7;
        if symbol = colon symbol then
            begin L8: store(symbol); NS;
                if is lay out(symbol) then goto L8;
                if symbol = open symbol then
                    begin ptr of text:= p - 1;
                        put in text array(comma symbol);
                        for p:= 1 step 1 until nls in delimiter do
                            put in text array(nlcr symbol);
                        READ synt unit:= comma symbol; NS; goto out
                    end else store(symbol)
                end;
        end;
        L9: ptr of text:= p; linernumber2:= 1;
        from delimiter array:= true;
        put in text array(semicolon symbol);
        symbol:= semicolon symbol;
        delimiter array[delimiter array ptr + 1]:= - 1;
        delimiter array ptr:= 0; READ synt unit:= close symbol;
        ERR(4 incorrect parameter delimiter+)
end else if is lay out(s)ymbol then goto L6 else
READ synt unit:= close symbol;
out: end else
begin READ synt unit:= symbol; NS end
end READ synt unit;

Boolean procedure is adic symbol(s); value s; integer s;
is adic symbol:= constant symbol ≤ s ≤ rowadic symbol;

Boolean procedure is declarator(s); value s; integer s;
is declarator:= Boolean symbol ≤ s ≤ formula symbol;

Boolean procedure is specifier(s); value s; integer s;
is specifier:= Boolean symbol ≤ s ≤ label symbol;

Boolean count nr of begins, delay one RE, no semicolon or
begin end allowed, reading allowed;

Boolean procedure is letter(s); value s; integer s;
is letter:= 10 ≤ s ≤ 62;

Boolean procedure is layout(s); value s; integer s;
is layout:= s = space symbol ∨ s = tab symbol ∨ s = ncr symbol;

Boolean procedure is digit(s); value s; integer s;
is digit:= 0 ≤ s ≤ 9;

Boolean procedure is enquiry; is enquiry:=
type symbol ≤ synt unit ≤ synt unit ≤ el symbol;

Boolean procedure is operator(s,n); value s; integer s,n;
begin procedure P(op); value op; integer op;
begin n:= n + 1; if s = op then goto OUT end;
n:= 0; P(plus symbol); P(minus symbol); P(times symbol);
P(div symbol); P(div symbol); P(power symbol);
is operator:= false; goto END;
OUT: is operator:= true;
END: end is operator;

procedure RE;
if delay one RE then delay one RE := false else

if no semicolon or begin end allowed

(synt unit := semicolon symbol v
 synt unit := begin symbol v
 synt unit := end symbol) then else

begin integer n; synt unit := next synt unit;
 operator identifier := false;
 ptr of text1 := ptr of text2; ptr of text2 := ptr of text;
 line number := line number1;
 next synt unit := if nr of begins > 0 then READ synt unit else
 synt unit;

L0: if count nr of begins then

begin if next synt unit = begin symbol then
 nr of begins := nr of begins + 1 else
 if next synt unit = end symbol then nr of begins :=
 nr of begins - 1
 end;

if synt unit = end symbol then

begin L: if next synt unit != end symbol ^ next synt unit !=
 else symbol ^ next synt unit != semicolon symbol then

begin ptr of text2 := ptr of text; next synt unit := READ synt unit;

if count nr of begins ^ next synt unit = end symbol then
 nr of begins := nr of begins - 1; goto L
 end
 end;

if next synt unit = comment symbol then

begin if "(synt unit = semicolon symbol v synt unit = begin symbol)
 then ERR (4-comment not appropriate);
 ptr of text := ptr of text - 2;

L1: if symbol != semicolon symbol then

begin NS; if symbol != ncr symbol then ptr of text :=
 ptr of text - 1; goto L1
 end;

NS; next synt unit := READ synt unit; goto L0
end else

if is adic symbol(synt unit) ^ is operator(next synt unit,n) then

begin adic sym := adic op[n,synt unit - constant symbol];
if adic sym = 0 then ERR(4 wrong combination for operator ident+);
ptr to first letgit := ptr of text1 X 2000 + line number1;
nr of ident := - (synt unit X 1024 + next synt unit);
RE; operator identifier := true; if next synt unit = identifier
then ERR(identifier after operator identifier+);
synt unit := identifier
end
end RE;

procedure RE semicolon;
begin if synt unit # semicolon symbol then
begin ERR(4 no semicolon where required+);
SEEK(synt unit = semicolon symbol v
synt unit = end symbol)
end; no semicolon or begin end allowed := false;
if synt unit # end symbol then RE;
no semicolon or begin end allowed := true
end;

procedure RE end;
begin no semicolon or begin end allowed := false; RE;
no semicolon or begin end allowed := true
end;

procedure RE begin; RE end;

procedure SAVE reading ptrs;
begin integer i,j; j := 0; count nr of begins := false;
nr of begins := nr of begins + 2;
comment The reason to augment "nr of begins" is that under
extraordinary circumstances this variable could otherwise obtain the
value -1. E.g. when "synt unit" denotes the last end, as in "... f:= 5
end";
ptr of old reading ptrs := ptr of old reading ptrs + 1;
for i := synt unit, next synt unit, ptr to first letgit,
nr of ident, ptr of text1, ptr of text2, ptr of text, symbol,
line number, line number1, line number2 do
begin j := j + 1; old reading ptrs[ptr of old reading ptrs, j] := i end;
old reading pptrs[ptr of old reading pptrs,j+1]=
if operator identifier then 1 else 0
end SAVE reading pptrs;

integer array old reading pptrs[1:10,1:12];
integer ptr of old reading pptrs;

procedure RESET reading pptrs;
begin integer i; procedure A(x); integer x;
begin i:= i + 1; x:= old reading pptrs[ptr of old reading pptrs,i]
end; i:= 0; A(synt unit); A(next sym unit);
A(ptr to first letgit); A(nr of iden); A(ptr of text1);
A(ptr of text2); A(ptr of text); A(symbol); A(line number);
A(line number1); A(line number2);
operator identifier:=
old reading pptrs[ptr of old reading pptrs,i + 1] = 1;
if ptr of old reading pptrs = 1 then count nr of begins:= true;
.nr of begins:= nr of begins - 2; delay one RE:= false;
ptr of old reading pptrs:= ptr of old reading pptrs - 1
end RESET reading pptrs;

procedure SET reading pptrs on(t); value t; integer t;
begin ptr of text:= t \ 2000; operator identifier:= false;
line number2:= t - ptr of text \ 2000;
ptr of text:= ptr of text - 1; NS; synt unit:= next synt unit:=
comma symbol; delay one RE:= false; \RE; RE
end SET reading pptrs on;

comment

The following procedure skips text until a certain condition "b" is
satisfied. The condition can be: "synt unit = close symbol" or "synt
unit = bus symbol" or "synt unit = comma symbol v synt unit = close
symbol".
The symbols ), , and , are read on the appropriate level, i.e. if
they occur in a bracket structure [...] or (...), then the complete
text enclosed by the brackets including the brackets is skipped
automatically. The effect is that "a + b[x \times (y + z) + 5] + c)"
is treated as "a + b + c)", so that "skips until)" has the desired
effect.
In order to restore erroneous situations, in which left brackets do not have corresponding right brackets, singly occurring right brackets stop the skipping process, and, if "b" is not true, "delay one RE" gets the value true so that a next call of RE will not result in reading a syntactic unit. Due to this construction, skipping of the subscript of a in: "a[(b + c[d x (e + f)])]", will be successful.

procedure SKIP text until(b); Boolean b;
begin delay one RE:= false;
L: if synt unit = end symbol v synt unit = begin symbol then
begin ERR(4tr of brackets incorrect); goto end end
else if synt unit = semicolon symbol then goto end
else if synt unit = open symbol then
begin RE; SKIP text until (synt unit = close symbol);
RE; goto L
end else
if synt unit = sub symbol then
begin RE; SKIP text until (synt unit = bus symbol);
RE; goto L
end else
if is enquiry then
begin if synt unit = el symbol then
begin RE; SKIP text until(synt unit = of symbol);
RE; goto L
end else
begin RE; CHECK(of symbol); RE; goto L end
end else
if synt unit = close symbol v synt unit = bus symbol v
synt unit = of symbol then
begin if ¬b then
begin delay one RE:= true;
comment For "additional synt unit" see the
procedure "PR and RE" and CHECK;
additional synt unit:= 0
end;
goto end;
end;
if ¬b then begin RE; goto L end;
2547 end:
2548 end SKIP text until;
2549
2550 procedure SEEK(b); Boolean b;
2551 begin delay one RE:= false;
2552 L: if not b ^ synt unit # begin symbol then begin RE; goto L end
2553 end;
2554
2555 integer type of standard identifier;
2556
2557 Boolean procedure Search for standard identifier(nr); integer nr;
2558 comment
2559 A given identifier is compared with the identifiers of the standard
2560 procedures, as given in the string of the next procedure "standard
2561 symbol".
2562 The "^" means: identifier is allowed in lower and upper case.
2563 The integer means the number of parameters.
2564 The symbols ), > and ] mean: the procedure is a real procedure, a
2565 non-type procedure and an integer procedure, respectively.
2566 The symbol / denotes the end of a set of identifiers beginning with
2567 the same letter (it is used in the procedure "fill alf");
2568 begin integer in,i,s,first letter,k,s1,s2,t,constant;
2569 boolean uc; if nr of ident < 0 then
2570 begin ERR(4:operator identifier in wrong place);
2571 goto notfound
2572 end; in:= i:= code table[nr of ident];
2573 s:= contents of [i]; first letter:= abs(s) + ti8 - 1;
2574 uc:= first letter > 36; if uc then first letter:=
2575 first letter - 27; k:= alf[first letter];
2576 if k = "4" v k > 0 ^ uc then goto notfound; k:= abs(k);
2577 newid: s1:= standard symbol(k); k:= k+1;
2578 if s1 = over symbol then goto notfound else
2579 if s1 = plus symbol then
2580 begin constant:= if uc then 28 else 1; s1:= standard symbol(k);
2581 k:= k + 1
2582 end else begin if uc then goto skipid else constant:= 1 end;
2583 newt:= t:= 0; for s:= 1,2,3,4 do
2584 begin s2:= standard symbol(k); t:= tx64+constant+s1;
if $s_2 =$ open symbol then
begin $t := -b\times a_4(4-a)$; goto compare end;
s1 := s2; $k := k + 1$
end;

compare: $s :=$ contents of $[i]$;
if $s = t \land s < 0$ then goto found else if $s = t$ then
begin $i := i + 1$; $s1 := s2$; goto newt end else
if abs(t) > abs(s + .5) then goto notfound else
skiplid:
begin for $k := k + 1$ while standard symbol(k) $\neq$ semicolon symbol,
$k + 1$ do; $i := in$; goto new id
end;

notfound: Search for standard identifier: $=$ false; $nr := 0$; goto out;
found: Search for standard identifier: $=$ true;
k := k + 1; $nr :=$ standard symbol(k);
k := k + 1; $i :=$ standard symbol(k);
type of standard identifier: $=$
if $i =$ close symbol then real symbol else
if $i =$ bus symbol then integer symbol else 0;
out:
end Search for standard identifier;

integer procedure standard symbol(k); integer k;
begin integer $s$;
cyo; $s :=$ STRING SYMBOL(k, 4)
abs(1); absfix(3) $+$ absfixp(3) $+$ absfixt(3) $+$ and(2);

arctan(1); available(0); /

bit(2); bitstring(3); /
+carrige(1); circ shift(2); clear shift(2); col(1);
compose(2); cos(1); cpos(0); cpunch(1); csym(1); /

date(0); /

entier(1); +even(1); +exit(0); exp(1); /
fix(4); fixc(3); +fixexp(3); +fixt(3); floc(4); floc(3);
+flop(3); +flipt(3); +from drum(2); /

head of(1); /

+inprod(5); /
+line number(0); ln(1); /
matmat(6); matvec(6); matvec(5); /
procedure INITIALIZE reading pters;
begin ptr of text:= ptr of text1:= ptr of text2:= 0;
synt unit:= next synt unit:= semicolon symbol; symbol:= 0;
line number:= line number1:= line number2:= 0;
ptr to first letgt:= nr of ident:= 0; nr of begins:= 2;
delay one RE:= false; from delimiter array:= false;
count nr of begins:= true; ptr of old reading pters:= 0;
nr of stringquotes:= 0; NS; HE semicolon; HE semicolon;
nr of begins:= nr of begins - 2
end;

comment:

6.6.3. The printing equipment

Algorithm:

procedure PR synt unit; if second scan then
begin integer i,s,j,k,ptr,bound; Boolean nlc pr;
if operator identifier then
begin operator identifier:= false;
ERR(operator ident in illegal place)
end; nlc pr:= false;
for i:= ptr of text1 step 1 until ptr of text2 - 1 do
\begin{verbatim}
begin s:= take from text array(i);
  if s < 128 then
  begin if s = nlcr symbol then
    begin if nlcr pr ^ synt unit \# string then else
      begin PR sym(s); nlcr pr:= \texttt{true} end
    end else PR sym(s)
  end else
  begin if s = becomes symbol then PR string(4:=4) else
    if s = other symbol then PR string(4=other \#) else
    if s < 180 then
      begin ptr:= ref2 to und symbol[s];
        bound:=ptr + und symbol[ptr + 1] + 1;
        for k:= ptr + 2 step 1 until bound do
          begin PUSYM(underlining symbol);PRSYM(underlining symbol);
            t:= und symbol[k]; PRSYM(t); PUSYM(t)
          end; PR string(4 \#)
      end else
    begin j:= if s < 399 then code table[s - 180]
      else code table[\texttt{take from text array}(i + 1) \times 400
        + \texttt{take from text array}(i + 2)];
      if s = 399 then i:= i + 2; j:= j + 1; k:= 1;
      for j:= j - 1 step 1 until k > 0 do
        begin k:= contents of [j]; t:= abs(k);
          for t:= t while t > 0 do
            begin s:= t \div t18; t:= (t - s \times t18) \times 64;
              PUSYM(s - 1); PRSYM(s - 1)
            end; PR string(4 \#)
        end end end end PR synt unit;
  end end end end PR and RE;

procedure PR and RE;
begin if second scan ^ delay one RE then
  SHOW(additional synt unit,\texttt{true}) else PR synt unit; RE
end;

procedure PR and RE semicolon;
begin if synt unit = semicolon symbol then
  PR synt unit else SHOW(semicolon symbol,\texttt{true});
  RE semicolon
\end{verbatim}
end;

procedure PR and RE end;
begin if synt unit = end symbol then
   PR synt unit else SHOW(end symbol, true);
   RE end
end;

procedure PR and RE begin;
begin if synt unit = begin symbol then
   PR synt unit else SHOW(begin symbol, true);
   RE begin
end;

integer procedure PR string(s); string s;
begin PR string:= 1; if second scan then
   begin PRINTTEXT(s); PUTTEXT(s) end
end;

integer procedure PR nloc; PR nloc:= PR string("
#");

procedure PR sym(i);
if second scan then begin PSYM(i); PSYM(i) end;

procedure SHOW(su,punch); value su; integer su; Boolean punch;
begin procedure p(s); string s;
begin PRINTTEXT(s); if punch then PUTTEXT(s) end;
procedure PR(s); value s; integer s;
begin PSYM(s); if punch then PSYM(s) end;
if su = unequal symbol then p("#") else
if su = power symbol then p("1") else
if su = becomes symbol then p("=") else
if su = string then p("string") else
if su = identifier then p("identifier") else
if su = integral number then p("int num") else
if su = real number then p("real num") else
if su = other symbol then p("unknown") else
if su < 128 then PR(su) else
begin integer j,k,ptr,bound; ptr:=ref2 to und symbol[su];
    bound:=ptr + und symbol[ptr + 1] + 1;
for k:= ptr + 2 step 1 until bound do
    begin PR(underlining symbol); PR(und symbol[k]) end;
PR(space symbol)
end end SHOW;

integer procedure PR int num(a); value a; integer a;
begin integer b; PR int num:= 1;
if a < 0 then begin PR string(4-k); a:= -a end;
if a < 9 then PR sym(a) else
begin b:= a div 10; a:= a - b * 10; PR int num(b); PR sym(a) end
end PR int num;

procedure PR operator;
begin operator identifier:= false;
if adic sym > 256 then PR sym(adic sym div 256);
PR sym(adic sym - adic sym div 256 * 256)
end;

procedure ERR(s); string s;
begin integer i; i:= print pos; NLCR;PRINTTEXT(10
"+-------------------------------------------------------------------+
ABSFIXT(4,0,line number);
PRINTTEXT(s); SPACE(80 - print pos);
SHOW(synt unit,false); SPACE(95 - print pos);
SHOW(next synt unit,false); SPACE(110 - print pos);
NLCR; SPACE(i)
end ERR;

procedure FATAL ERR(s); string s;
begin ERR(s); EXIT end;

procedure CHECK(s); value s; integer a;
if synt unit # s then
begin integer i; i:= print pos; NLCR;
ERR(4synt unit not OK4);
NLCR; PRINTTEXT(4synt unit should be:)

2775    SHOW(s,false); NLCR; SPACE(1); delay one PE:= true;
2776    additional synt unit:= s
2777    end;
2778
2779    integer procedure PR sn(s,n,tail); string s; integer n,tail;
2780    begin PR string(s); PR int num(n); PR sn:= tail end;
2781
2782    comment:
2783
2784    6.6.4. The information-cells equipment
2785
2786    Algorithm;
2787
2788    integer procedure STORE into information list(list); integer list;
2789    begin integer x,p; p:= ptr of inf list;
2790    ptr of inf list:= ptr of inf list + 1;
2791    x:= list; contents of [p]:= ptr of inf list - p;
2792    if contents of [p + 1] > 0 then CHECK(identifier);
2793    STORE into information list:= p
2794    end STORE into information list;
2795
2796    integer procedure st(head,inf); integer head,inf;
2797    begin st:= head;
2798    if ptr of inf list > ptr to name list - 3 then FATAL ERR(  
2799            "inf list too small");
2800    contents of [ptr of inf list]:= inf;
2801    ptr of inf list:= ptr of inf list + 1
2802    end st;
2803
2804    integer ptr of inf list,max of inf list,ptr to first ident,
2805    ptr to name list,nr of identifiers,max nr of identifiers,
2806    t10,t15,t18,declared as value,declared as name, specified as
2807    value,specified as name,standard identifier,with local,
2808    reinitptr,without local,block depth,min block depth,in formula
2809    procedure body;
2810    Boolean normal compilation,print information list;
2811    integer array contents of [0:5000],code table[1:400];
2812    comment
2813    If "normal compilation" has the value true "INITIALIZE inf list ptrs"
begins with filling the information list and the contents of some "inf list ptrs" with the information of the catalogue. This is done via the procedure "reinit", which uses the procedure "catalogue symbol", in which, in the form of a string, this information is stored.

The information of the catalogue is itself produced by the same compiler with "normal compilation" having the value false, while "inf list ptrs" are given their purely initial values. As a result, a string is produced being the procedure "catalogue symbol", by means of the procedure "print catalogue".

The procedure "PR inf cells" does not come into action normally, as the boolean "print information list" has the value false. If one needs the contents of the information list, however, this value can easily be made true in the main program.

procedure INITIALIZE inf list ptrs;
begin integer procedure val(v); value v; integer v;
val:= if normal compilation then reinit else v;
max of inf list:= 5000; max nr of identifiers:= 400;
t10:= 2 ↑ 10; t15:= t10 × 32; t18:= t15 × 8; reinitptr:= -1;
declared as value:= 1; declared as name:= 2; specified as value:= 3;
specified as name:= 4; standard identifier:= 5; with local:= 6;
without local:= 7; block depth:= min block depth:= 0;
in formula procedure body:= 0;
ptr to first ident:= val(0); ptr of inf list:= val(1);
ptr to name list:= val(max of inf list);
nr of identifiers:= val(0); if normal compilation then
begin integer i;
for i:= ptr of inf list - 1 step - 1 until 1,
ptr to name list + 1 step 1 until max of inf list do
contents off[i]:= reinit;
for i:= 1 step 1 until nr of identifiers do
code table[i]:= reinit
end
end;

decrease procedure PR inf cells; if print information list then
begin integer i,p,s,t; p:= 1;
NLCR; NLCR; PRINTTEXT("contents of inf list");
L: NLOR; if p > ptr of inf list then
   begin p:=ptr to first ident; goto L2 end;
for i:= p step 1 until p + contents of [p] - 1 do
   begin ABSFIXT(4,0,i); FIXT(8,0,contents of [i]) end;
   p:= p + contents of [p]; goto L;
L2: if p = ptr to name list then goto out;
NLOR; ABSFIXT(4,0,p); i:=1;
for s:= contents off[p] while i > 0 do
   begin i:= s; s:= abs(s);
   for s:= s while s > 0 do
      begin t:= s; t18; s:= (s - t x t;8) x 64;
         PRSYM(t - 1)
      end;
   p:= p - 1
end;
NLOR; PR string(4)
for i:= p,p - 1,p - 2 do ABSFIXT(4,0,contents of[i]);
p:= p - 3; goto L2;
out: NLOR; NLOR; for i:= 1 step 1 until nr of identifiers do
   begin ABSFIXT(4,0,i); ABSFIXT(4,0,codetable[i]) end
end PR inf cells;

procedure print catalogue;
begin integer i,j;
   procedure PR cat(x); value x; integer x;
   begin j:= j + 1; PR int num(x);
      if j = j + 5 x 5 then PR nlcr else PR string(4)
   end PR cat;
end PR string(4
integer procedure catalogue symbol;
begin reinitptr:= reinitptr + 1;
catalogue symbol:= STRING SYMBOL(reinitptr,4);
PR nlcr; PR sym(bar symbol); PR sym(smaller than symbol);
j:= 0; PR cat(ptr to first ident); PR cat(ptr of inf list);
PR cat(ptr to name list); PR cat(nr of identifiers);
for i:= ptr of inf list - 1 step - 1 until 1,
   ptr to name list + 1 step 1 until max of inf list do
   PR cat(contents of[i]);
for i := 1 step 1 until nr of identifiers do
FR cat(codetable[i]); FR ncr;
FR sym(bar symbol); FR sym(greater than symbol);
FR string($)
end;
end print catalogue;

integer procedure reinit;
begin integer s,k,t;
for s:= catalogue symbol while is lay out(s) do;
k:= if s = minus symbol then 0 else s;
t:= if s = minus symbol then - 1 else 1;
for s:= catalogue symbol while ~ is lay out(s) do
k:= k * 10 + s;
reinit:= k * t
end reinit;

integer procedure catalogue symbol;
begin reinitptr:= reinitptr + 1;
catalogue symbol:= STRING SYMBOL(reinitptr,
45000 49 4983 4 1
-141 3 0 28 4882435
24001 4 6 0 0
4685827 20001 3 6 0
0 4882435 16001 2 6
32 0 0 0 40
0 0 46 -140 10
0 3 0 4736 12001
1 7 0 0 0
0 0 0 18 49
-140 10 4 0 0
-4456448 3 0 4987 -5829662
2 4991 0 -4194304 1
0 4995 -2937792 7403158 5000
4995 4991 4987
$)
end;
procedure store bits(int,b); integer int; Boolean array b;
begin integer i,k; i := 0;
for k:= 0,1,2,3,4,5 do
i:= 2 * i + (if b[k] then 1 else 0); int:= i
end;

procedure set bits(int,b); value int; integer int;
Boolean array b;
begin integer k;
for k:= 0,1,2,3,4,5 do
if int/32 > .99 then
begin b[k]:= true; int:= (int - 32) * 2 end
else
begin b[k]:= false; int:= int * 2 end
end;

comment:
6.6.5. The main program

Algorithm:
Boolean first scan, second scan,dangerous procedures,
operator identifier;
real t1,t2;
procedure EXIT;
begin integer s; if reading allowed then
begin s:= symbol; symbol:= RESYM;
if " (s = underlining symbol ^ symbol = semicolon symbol)
then goto L
end;
t2:= time; PRINTTEXT(4,time is: $); ABSFLX(4,2,(t2 - $)/3.6);
PRINTTEXT(4 mh$); PR sym(nlor symbol); t1:= t2; goto BEGIN
end EXIT;

begin integer i,j;
for i:= 0,1,2 do for j:= 1,2,3,4,5,6 do adic op[i,j]:= 0;
adic op[1,2]:= 55; comment S;
adic op[2,2]:= 40; comment D;
adic op[3,2]:= 52; comment P;
adic op[4,2]:= 53; comment Q;
adic op[5,2]:= 45 × 256 + 53; comment IQ;
adic op[6,2]:= 41; comment E;
adic op[1,1]:= 52 × 256 + 48; comment PL;
adic op[2,1]:= 49 × 256 + 45; comment MI;
adic op[6,1]:= 45 × 256 + 41; comment IE;
adic op[1,0]:= 45 × 256 + 50; comment IN;
adic op[3,0]:= 54 × 256 + 50; comment RN;
end;

initialize symbols; t1:= time;
normal compilation:= true; print information list:= true;
begin dangerous procedures:= false; operator identifier:= false;
first scan:= true; second scan:= false; initialize inf list pointers;
new page; reading allowed:= true;

l:= initialize reading pointers; if synt unit: ≠ begin symbol then
begin error("program does not begin with begin ");
synt unit:= begin symbol; nr of begins:= nr of begins + 1
end;
if nr of begins < 1 then nr of begins:= 1;
begin:= 1; preceding begin:= 4;
envelope of block(0);
if first scan then
begin first scan:= false; second scan:= true;
operator identifier:= false;
if normal compilation then
begin contents of[begin + 2]:=store into information list(st(st(0,
-end symbol),
0))
end else
begin ptr of inf list:= ptr of inf list - 3;
print catalogue; exit
end;
comment One extra symbol is placed after the program;
if symbol = space symbol then ptr of text:= ptr of text - 1;
put in text array(semicolon symbol);
RUNOUT;
PR inf cells; CARRIAGE(3); goto L
end; EXIT
end
end

6-80
7. Examples of compiled programs

From all the test programs a selection has been chosen and is reproduced in this chapter. This selection does not pretend to give exhaustive tests; it serves illustrative reasons only.

The results of the ABC ALGOL compiler have been edited by an ALGOL editor.

7.1. General structure and information list

Example 1.1 is a simple example illustrating the block structure and the information list. Note that "f" occurs three times: as real, as formula and as formal parameter. Note also the three different translations of "f + 2 x f".

The print-out of the information list consists of three parts:
the part for the information cells,
the part containing the identifiers,
the part containing the code table for the identifiers.

Note the appearance of the catalogue in the information list. The first part of the information list is printed showing the information cells; i.e. the contents of each information cell is printed on a new line. Only the information cells for begin are printed on three lines.

The index and the contents of the array elements are printed. The first element of an information cell contains the number of array elements this particular information cell consists of. The information cells can easily be recognized:
with first element containing +10 : begin information cell
with first element containing +3 : end information cell
with first element containing +7 : procedure identifier cell

The second element contains for an identifier its identification number (note that the three "f"s have the same identification number equal to the identification number of the "f" in the library procedure "replace" (see sections 2.6 and 5.2.8)), for a begin the number -140 and for end the number -141.
The third element contains for an identifier its place of occurrence in the text (the last three digits determining the line number), for a begin the address of the corresponding end information cell and for an end the address (if it exists, otherwise it is 0) of the begin information cell of the enclosing block.

For other elements of the information cells we refer to an exhaustive description in section 5.1.2.

The part of the information list really pertaining to the program shown begins with the information cell with address 49 and ends with the information cell with address 112.

The second part of the information list shows the symbols constituting the identifiers preceded by the address of the identifier and followed by the address of the identifier alphabetically preceding, the address of the identifier alphabetically following and the identification number of the identifier.

The third part shows the code table containing for each identification number the address in the information list where the corresponding identifier is stored.

Example 1.2 demonstrates a scala of syntactic structures together with, for the last time, the contents of the information list.

Example 1.1.

1 begin real f;
2 begin formula f,g;
3 procedure P(f); value f; formula f; g:= f + 2 * f;
4 P(f); f:= 2 * f + f
5 end; f:= 2 * f + f
6 end

contents of inf list

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+10</td>
<td>2</td>
<td>-140</td>
<td>3</td>
<td>+115</td>
<td>4</td>
<td>+49</td>
<td>5</td>
<td>+0</td>
<td>6</td>
<td>+3</td>
<td>7</td>
<td>+0</td>
<td>8</td>
<td>+0</td>
<td>9</td>
<td>+0</td>
<td>10</td>
<td>+0</td>
<td>11</td>
<td>+7</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
<td>37</td>
<td>38</td>
<td>39</td>
<td>39</td>
<td>40</td>
<td>41</td>
</tr>
<tr>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
</tbody>
</table>
| 28 | +6 | 29 | +2 | 30 | +16001 | 31 | +4882435 | 32 | +0 | 33 | +3 | 34 | +6 | 35 | +3 | 36 | +20001 | 37 | +4685827 | 38 | +0 | 39 | +0 | 40 | +6 | 41 | +4 | 42 | +24001 | 43 | +4882435 | 44 | +28 | 45 | +0 | 46 | +3 | 47 | -141 | 48 | +1 | 49 | +10 | 50 | -140 | 51 | +112 | 52 | +63 | 53 | +0 | 54 | +0 | 55 | +0 | 56 | +0 | 57 | +0 | 58 | +0 | 59 | +4 | 60 | +2 | 61 | +12001 | 62 | +4751360 | 63 | +10 | 64 | -140 | 65 | +109 | 66 | +90 | 67 | +0 | 68 | +78 | 69 | +0 | 70 | +0 | 71 | +0 | 72 | +32 | 73 | +5 | 74 | +2 | 75 | +26002 | 76 | +4882434 | 77 | +0 | 78 | +5 | 79 | +4 | 80 | +30002 | 81 | +4882434 | 82 | +73 | 83 | +7 | 84 | +5 | 85 | +42003 | 86 | +4736 | 87 | +0 | 88 | +1 | 89 | +0 | 90 | +10 | 91 | -140 | 92 | +106 | 93 | +0 | 94 | +0 | 95 | +100 | 96 | +0 | 97 | +0 | 98 | +0 | 99 | +32 | 100 | +6 | 101 | +2 | 102 | +46003 | 103 | +4882435 | 104 | +0 | 105 | +0 | 106 | +3 | 107 | -141 | 108 | +63 | 109 | +3 | 110 | -141 | 111 | +49 | 112 | +3 | 113 | -141 | 114 | +1 | 115 | +3 | 116 | -141 | 117 | +0

5000 replace 4995 4983 1
4995 l 0 4991 2
4991 left 4987 0 3
4987 g 0 0 4
4983 P 0 0 5
Translation 1.1.

begin real Zf ; lnr(1); begin
procedure ZF (Zf ); integer Zf ,fname; fname:= gnr1lnr(3);

DEVAL(Yf ,Zf ); ASSIGN(Zg ,S(Yf ,F(IN(2 ),Yf )));
; ERASE(fname)
end ;

integer Zg ,Zf ,fname; fname:= gnr1lnr(2);
Zg := Zg := NULL;
DE(Zf ,NULL); DE(Zg ,NULL); ZP (V(Zf )); ASSIGN(Zf ,S(P(IN(2 ),V(Zf ))),V(Zf )));
; ERASE(fname)
end ; Zf :=2 ×Zf +Zf

end

Example 1.2

begin integer k; formula f:= 3.13, g = 3.14; formula array h[1:10];
formula procedure dyadic + (a,b); formula a,b;
dyadic + := 5;
formula procedure dyadic - (a,b); value a,b; formula a,b;
dyadic - := 5;
integer i; real r;
if dyadic f ~ monadic h[7] then
for i:= 1,1 step 1 until 1, while i < 10,11 do i:= i + 5;
f:= len of el 5 of rhs of f;
ri:= len of el 5 of rhs of(f + g × h[3]);
f:= el 7 of 401234567;
i:= length of 401234567;
ri:= date + f + g;
f:= rowadic(10,i,5,i + 5);
f:= polyadic(10,i,5,i + 5);
f:= +(f + g × 3.14 - f × (-r + 314×2/f - f) ↑
(h[5] + f ↓ (((f ↑ 1024 ↑ 2048/10))));
### contents of inf list

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+10</td>
<td>2</td>
<td>-140</td>
<td>3</td>
<td>+156</td>
<td>4</td>
<td>+49</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>+0</td>
<td>6</td>
<td>+0</td>
<td>7</td>
<td>+0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>+0</td>
<td>9</td>
<td>+0</td>
<td>10</td>
<td>+0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>+7</td>
<td>12</td>
<td>+1</td>
<td>13</td>
<td>+1201</td>
<td>14</td>
<td>+4736</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>+0</td>
<td>16</td>
<td>+3</td>
<td>17</td>
<td>+0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>+10</td>
<td>19</td>
<td>-140</td>
<td>20</td>
<td>+16</td>
<td>21</td>
<td>+0</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>+0</td>
<td>23</td>
<td>+40</td>
<td>24</td>
<td>+0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>+0</td>
<td>26</td>
<td>+0</td>
<td>27</td>
<td>+32</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>+6</td>
<td>29</td>
<td>+2</td>
<td>30</td>
<td>+16001</td>
<td>31</td>
<td>+4802435</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>+0</td>
<td>33</td>
<td>+0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>+6</td>
<td>35</td>
<td>+3</td>
<td>36</td>
<td>+20001</td>
<td>37</td>
<td>+4685827</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>+0</td>
<td>39</td>
<td>+0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>+6</td>
<td>41</td>
<td>+4</td>
<td>42</td>
<td>+20001</td>
<td>43</td>
<td>+4882435</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>+28</td>
<td>45</td>
<td>+0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>+3</td>
<td>47</td>
<td>-141</td>
<td>48</td>
<td>+1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>+10</td>
<td>50</td>
<td>-140</td>
<td>51</td>
<td>+153</td>
<td>52</td>
<td>+87</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>+144</td>
<td>54</td>
<td>+69</td>
<td>55</td>
<td>+74</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>+1</td>
<td>57</td>
<td>+1</td>
<td>58</td>
<td>+32</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>+5</td>
<td>60</td>
<td>+5</td>
<td>61</td>
<td>+14002</td>
<td>62</td>
<td>+4718593</td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>+0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>+5</td>
<td>65</td>
<td>+2</td>
<td>66</td>
<td>+20002</td>
<td>67</td>
<td>+4882434</td>
<td></td>
</tr>
<tr>
<td>68</td>
<td>+0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>69</td>
<td>+5</td>
<td>70</td>
<td>+4</td>
<td>71</td>
<td>+34002</td>
<td>72</td>
<td>+4882433</td>
<td></td>
</tr>
<tr>
<td>73</td>
<td>+64</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>+6</td>
<td>75</td>
<td>+6</td>
<td>76</td>
<td>+52002</td>
<td>77</td>
<td>+4887104</td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>+0</td>
<td>79</td>
<td>+1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>+7</td>
<td>81</td>
<td>-172096</td>
<td>82</td>
<td>+78033</td>
<td>83</td>
<td>+4887175</td>
<td></td>
</tr>
<tr>
<td>84</td>
<td>+0</td>
<td>85</td>
<td>+2</td>
<td>86</td>
<td>+0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>87</td>
<td>+10</td>
<td>88</td>
<td>-140</td>
<td>89</td>
<td>+139</td>
<td>90</td>
<td>+119</td>
<td></td>
</tr>
<tr>
<td>91</td>
<td>+80</td>
<td>92</td>
<td>+0</td>
<td>93</td>
<td>+0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>94</td>
<td>+0</td>
<td>95</td>
<td>+0</td>
<td>96</td>
<td>+0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>97</td>
<td>+6</td>
<td>98</td>
<td>+7</td>
<td>99</td>
<td>+84003</td>
<td>100</td>
<td>+4882436</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>+0</td>
<td>102</td>
<td>+0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>+6</td>
<td>104</td>
<td>+8</td>
<td>105</td>
<td>+88003</td>
<td>106</td>
<td>+4882436</td>
<td></td>
</tr>
<tr>
<td>107</td>
<td>+0</td>
<td>108</td>
<td>+0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>109</td>
<td>+3</td>
<td>110</td>
<td>-141</td>
<td>111</td>
<td>+49</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>112</td>
<td>+7</td>
<td>113</td>
<td>-17207</td>
<td>114</td>
<td>+13003</td>
<td>115</td>
<td>+4887175</td>
<td></td>
</tr>
<tr>
<td>116</td>
<td>+0</td>
<td>117</td>
<td>-2</td>
<td>118</td>
<td>+0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>119</td>
<td>+10</td>
<td>120</td>
<td>-140</td>
<td>121</td>
<td>+141</td>
<td>122</td>
<td>+0</td>
<td></td>
</tr>
<tr>
<td>123</td>
<td>+122</td>
<td>124</td>
<td>+135</td>
<td>125</td>
<td>+0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>126</td>
<td>+0</td>
<td>127</td>
<td>-0</td>
<td>128</td>
<td>+32</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>129</td>
<td>+6</td>
<td>130</td>
<td>+7</td>
<td>131</td>
<td>+136005</td>
<td>132</td>
<td>+4882435</td>
<td></td>
</tr>
<tr>
<td>133</td>
<td>+0</td>
<td>134</td>
<td>+0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>135</td>
<td>+6</td>
<td>136</td>
<td>+8</td>
<td>137</td>
<td>+140005</td>
<td>138</td>
<td>+4882435</td>
<td></td>
</tr>
<tr>
<td>139</td>
<td>+129</td>
<td>140</td>
<td>+0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>141</td>
<td>+3</td>
<td>142</td>
<td>-141</td>
<td>143</td>
<td>+49</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>144</td>
<td>+5</td>
<td>145</td>
<td>+9</td>
<td>146</td>
<td>+190007</td>
<td>147</td>
<td>+4718593</td>
<td></td>
</tr>
<tr>
<td>148</td>
<td>+59</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>149</td>
<td>+4</td>
<td>150</td>
<td>+10</td>
<td>151</td>
<td>+196007</td>
<td>152</td>
<td>+4751360</td>
<td></td>
</tr>
<tr>
<td>153</td>
<td>+3</td>
<td>154</td>
<td>-141</td>
<td>155</td>
<td>+1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>156</td>
<td>+3</td>
<td>157</td>
<td>-141</td>
<td>158</td>
<td>+0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5000 replace 4995 0 1
4995 r 4975 4991 2
4991 left 4987 4963 3
4987 g 0 4983 4
4983 k 4979 0 5
4979 h 0 4967 6
4975 a 0 4971 7
4971 b 0 4959 8
4967 i 0 0 9
4963 r 0 0 10
4959 date 0 0 11

Translation 1.2.

begin
integer low1c1,up1c1,i1;
integer fmn; fmn:= gmn;lnr(2);
low1c1:= 1;up1c1:= 10;
begin integer procedure S(Za ,Zb ); integer Za ,Zb ;
begin lnr(4);S:= IN(5)
end ;integer procedure D(Za ,Zb ); integer Zc ,Zb ;
begin integer Yb ,Ya ,fnn; fnn:= gnm;lnr(6);

DEVAL(Ya ,Za );DEVAL(Yb ,Zb );D:= IN(5)
; ERASE(fnn)
end ;real Zr;

integer array Zh [lowc1: upc1];
integer Zi ,Zk ,Zg ,Zf ,fnn; fnn:= gnm;
Zf := Zg := NULL;
DE(Zf ,RN(3.13));DEVAL(Zg ,RN(3.14));
for i1:= lowc1 step 1 until upc1 do
begin DE(Zh [11],NULL) end;
if Zf = TYPE CAT(V(Zf ))^1= TYPE CAT(V(Zh [7]));then
for Zi :=1,1 step 1 until 1,while Zi <10,11 do Zi :=Zi +5;
ASSIGN(Zf ,LHS(EL(5, RHS(V(Zf )))));
Zr :=ARLHS(EL(5, RHS(S(V(Zg ),P(Zg ,V(Zh [3])))}}));
ASSIGN(Zf ,EL(7, STRLEN(st(66051, st(263430, st(1800,0))))),3));
Zl :=LENGHT(STRING(st(66051, st(263430, st(1800,0)))),3));
Zr :=date +V(Zf )+abs(Zg );
ASSIGN(Zf ,STORE ROW(128,10,Zi ,Zl ,Zi +5));
ASSIGN(Zf ,STORE ROW(96,10,Zi ,Zl ,S(IN(Zi ),IN(5))));
ASSIGN(Zf ,D(S(FL(V(Zf )),P(Zg ,RN(3.14 ))),P(V(Zf ))),B(D(S(MI(RN(Zr ))), Q(RN(3.14-2),V(Zf ))),V(Zf ))),S(V(Zh [5]),I(V(Zf ))),Q(IE(IE(V(Zf )),
1024 ),2048),IN(10))));
; ERASE(fnn)
end ; ERASE(fnn) end

7.2. Block-entry, block-exit

Examples 2.1 - 2.8 demonstrate the effect of goto statements, innerblocks with formula declarations and procedures with unspecified parameter on the appearance of the integer ann in the block reading and the introduction of "ERASE(ann)" in the translation of a label.
Example 2.1.

\begin{verbatim}
begin formula f:= 1, g:= 2; f:= f + g end;
\end{verbatim}

Translation 2.1.

\begin{verbatim}
begin integer Zg, Zf, fnn;
  fnn:= gnn; lnr(2); Zf:= Zg:= NULL; DE(Zf, IN(1)); DEVAL(Zg, IN(2));
  ASSIGN(Zf, S(V(Zf), Zg)); ERASE(fnn)
end
\end{verbatim}

Example 2.2.

\begin{verbatim}
begin formula f; l: f:= 5; begin formula g; goto l end end;
\end{verbatim}

Translation 2.2.

\begin{verbatim}
begin integer Zf, fnn, snn;
  fnn:= gnn; lnr(2); Zf:= NULL; DE(Zf, NULL); snn:= gnn;
Zl: ERASE(snn); ASSIGN(Zf, IN(5));
  begin integer Zg, fnn;
    fnn:= gnn; lnr(2); Zg:= NULL; DE(Zg, NULL); go to Zl; ERASE(fnn)
  end;
  ERASE(fnn)
end
\end{verbatim}

Example 2.3.

\begin{verbatim}
begin formula f; l: f:= 5; begin formula g; g:= f end end;
\end{verbatim}

Translation 2.3.

\begin{verbatim}
begin integer Zf, fnn;
  fnn:= gnn; lnr(2); Zf:= NULL; DE(Zf, NULL);
Zl: ASSIGN(Zf, IN(5));
  begin integer Zg, fnn;
\end{verbatim}
fun := gmn; lr(2); zg := NULL; de(zg, NULL); assign(zg, v(zf));
end;
erase(fun)
end

Example 2.4.

begin formula f; 1: f := 5; begin formula g; goto h; h; end end ;

Translation 2.4.

begin integer zf, fun, snn;
    fun := gmn; lr(2); zf := NULL; de(zf, NULL); snn := gmn;
    zl: erase(snn); assign(zf, in(5));
    begin integer zg, fun;
    fun := gmn; lr(2); zg := NULL; de(zg, NULL); goto zg;
    zg: ; erase(fun)
end;
erase(fun)
end

Example 2.5.

begin formula f; 1: f := 5; begin real r; goto l end end ;

Translation 2.5.

begin integer zf, fun;
    fun := gmn; lr(2); zf := NULL; de(zf, NULL);
    zl: assign(zf, in(5));
    begin real zg;
        lr(2); goto zl
    end;
    erase(fun)
end

Example 2.6.
begin formula f; l: f:= 5;
begin real r1;
begin formula s;
begin real r2; begin integer i; i:= 5 end;
begin real r3; goto 1 end
end end end end;

Translation 2.6.

begin integer Zf, fmn, srrn;
fmn:= gnn; lnr(2); Zf:= NULL; DE(Zf, NULL); srrn:= gnn;
Z1: ERASE(srrn); ASSIGN(Zf, IN(5));
begin real Zr1;
lnr(3);
begin integer Zg, fmn;
fmn:= gnn; lnr(4); Zg:= NULL; DE(Zg, NULL);
begin real Zr2;
lnr(5);
begin integer Zi;
lnr(5); Zi:= 5
end;
begin real Zr3;
lnr(6); go to Z1
end
end;
ERASE(fmn)
end
end;
ERASE(fmn)
end

Example 2.7.

begin formula f; procedure p(label); goto label;
l: f:= 5; p(l)
end ;
Translation 2.7.

begin

procedure Zp(Zlabel);
begin lnr(2); go to Zlabel end;

integer Zf, fnn, srrn;
 frrn:= grn; lnr(2); Zf:= NULL; DE(Zf, NULL); srrn:= grn;
Z1: ERASE(srrn); ASSIGN(Zf, IN(5)); Zp(Z1); ERASE(fnn)
end

Example 2.8.

begin formula f; procedure p(label); f:= 5;
  1: f:= 5; p(1)
end i

Translation 2.8.

begin

procedure Zp(Zlabel);
begin lnr(2); ASSIGN(Zf, IN(5)) end;

integer Zf, fnn;
  frrn:= grn; lnr(2); Zf:= NULL; DE(Zf, NULL);
Z1: ASSIGN(Zf, IN(5)); Zp(Z1); ERASE(fnn)
end

7.3. Formula array in a procedure body

Examples 3.1 - 3.2 demonstrate the complications which arise from a formula array declaration in a procedure body. Note that the "f" parameter is not combined with the variables "x" and "y" into one integer declaration and one initialization, such as it is done in example 3.2.
Example 3.1.

\begin{verbatim}
begin formula procedure p1(f,g);
  value f; formula f,g;
  begin formula x,y; formula array fa[1:10];
  end;
end;
\end{verbatim}

Translation 3.1.

\begin{verbatim}
begin

integer procedure Zp1(Zf, Zg); integer Zf, Zg;
begin integer low1c1, up1c1, i1;
  integer Yf, lnr;
  lnr:= gnr; lnr(4); DEVAL(Yf, Zf); low1c1:= 1; up1c1:= 10;
begin integer array Zfa[low1c1:up1c1];
  integer Yp1, Zy, Zx, lnr;
  lnr:= gnr; if protect then ERR(4protection error in form proc); Yp1:= NULL; Zx:= Zy:= NULL; DE(Zx, NULL); DE(Zy, NULL);
  for i1:= low1c1 step 1 until up1c1 do
  begin DE(Zfa[i1], NULL) end;
  ; ; ERASE(lnr); if Yp1 = NULL then ERR(
  no assignment to proc ident); Zp1:= Yp1; protect:= false
end;
ERASE(lnr)
end;

lnr(2);
end
\end{verbatim}

Example 3.2.

\begin{verbatim}
begin procedure p1(f,g); value f; formula f,g;
  begin formula x,y; end;
end;
\end{verbatim}
Translation 3.2.

begin

procedure Zp1(Zf, Zg); integer Zf, Zg;
begin integer Zy, Zx, Yf, fnn;
    fnn := gmn; lnr(3); Zx := Zy := NULL; DEVAL(Yf, Zf); DE(Zx, NULL);
    DE(Zy, NULL); ; ERASE(fnn)
end;
lnr(2);
end

7.4. Translation of procedure parameters

Example 4.1 demonstrates the way formal and actual procedure parameters are compiled. Note the effect of "call-by-value" and "call-by-name" on the translation of actual parameters.

Example 4.1.

begin procedure P(a,b,c,d); value a; formula a,b; formula array c;
    procedure d(formula value,formula,formula array,procedure);
    begin d(a,b,c,d); d(a+b,b,c,a); d(b,b,c,a);
        d(h[1],h[2],h,P); d(h[1],h[1]+h[2],h,P);
        d(h[1]+h[2],h[2],h,P); d(h[1]+h[2],h[1]+h[2],h,P)
    end;
    formula f:= 3.14,g = 3.14; formula array h[1:2];
    P(f,f,h,P); P(g,f,h,P); P(f+g,f,h,P); P(f,g,h,P); P(f,g,h,P);
    P(h[1],h[2],h,P); P(f,h[1]+h[2],h,P);
end i

Translation 4.1.

begin integer low1c1, up1c1, i1;
    integer fnn;
    fnn := gmn; lnr(2); low1c1 := 1; up1c1 := 2;
begin

procedure ZP(Za, Zb, Ze, Zd); integer Za, Zb; integer array Ze; procedure Zd;
begin integer Ya, fmn;
    fmn:= gm; lnr(4); DEVAL(Ya, Za); Zd(Ya, Zb, Ze, Zd);
    Zd(S(Ya, VN(Zb)), Zb, Ze, Zd); Zd(VN(Zb), Zb, Ze, Zd);
    Zd(V(Zh[1]), Zh[2], Zh, ZP);
    Zd(V(Zh[1]), S(V(Zh[1]), V(Zh[2])), Zh, ZP);
    Zd(V(Zh[1]), V(Zh[2])), Zh[2], Zh, ZP);
    Zd(S(V(Zh[1]), V(Zh[2])), S(V(Zh[1]), V(Zh[2])), Zh, ZP); ERASE(fmn)
end;

integer array Zh[low1c1:up1c1];
integer Zg, Zf, fmn;
    fmn:= gm; Zf:= Zg:= NULL; DE(Zf, RN(3.14)); DEVAL(Zg, RN(3.14));
    for i:= low1c1 step 1 until up1c1 do
    begin
    DE(Zh[i], NULL) end;
    ZP(V(Zf), Zf, Zh, ZP); ZP(Zg, Zf, Zh, ZP); ZP(S(V(Zf), Zg), Zf, Zh, ZP);
    ZP(V(Zf), S(V(Zf), Zg), Zh, ZP); ZP(V(Zf), Zg, Zh, ZP);
    ZP(V(Zh[1]), Zh[2], Zh, ZP); ZP(V(Zf), S(V(Zh[1]), V(Zh[2])), Zh, ZP);
    ERASE(fmn)
end;
        ERASE(fmn)
end

7.5. Protection mechanism for formula procedures

Example 5.1 - 5.3 demonstrate the introduction of the protect- mechanism when it is not sure that the last executed statement in a procedure body of a formula procedure is the assignment to the procedure identifier such as, e.g., is the case in example 1.2.
Example 5.4 demonstrates what happens with formula procedures declared within each other. For "p7" the protect-mechanism is necessary, for "p8" it is not and for "p9" it is. This is reflected in the appearance of "if protect then ERR ..." and the declaration of a local "yp ..." in the block begin and in the appearance of "if yp ... = NULL then ERR ...; zp ...:= yp ...; protect:= false" at the block end for p7 and p9. Note, however, that the "protect:= false" part in the block end of p9 is not given as it should, due to the assignment to p7 in the body of p9.

Example 5.1.

begin formula procedure p10;
    begin formula f; f:= 5; p10:= f; f:= 5 end;
end i

Translation 5.1.

begin

    integer procedure 2p10;
    begin integer Yp10, Zf, fmn;
        fmn:= grn; lnr(3); if protect then ERR(
            4protection error in form proc#); Yp10:= NULL; Zf:= NULL; DE(Zf, NULL);
        ASSIGN(Zf, IN(5)); Yp10:= V(Zf); protect:= true; ASSIGN(Zf, IN(5));
        ERASE(fmn); if Yp10 = NULL then ERR(4no assignment to proc ident#);
        2p10:= Yp10; protect:= false
    end;

    lnr(2);
end

Example 5.2.

begin formula procedure dyadic *(a,b); value a,b; formula a,b;
    begin dyadic + := a; l; end;
end i

Translation 5.2.
begin

integer procedure S(Za, Zb); integer Za, Zb;
begin integer Ys, Yb, Ya, fnr;
  fnr := gnr; lnr(3); if protect then ERR("protection error in form proc"); Ys := NULL; DEVAL(Ya, Za);
  DEVAL(Yb, Zb); Ys := abs(Ya); protect := true;
  Zi := ERASE(fnr); if YS = NULL then ERR("no assignment to proc ident"); S := YS; protect := false
end;

lnr(2);
end

Example 5.3.

begin formula f; integer i;
  formula procedure p14;
  if i = 1 then 1: i := 5 else 2: p14 := 5;
  for i := 1 do 3: fi := 5;
  begin formula 8; goto 4 end
end i

Translation 5.3.

begin

integer procedure Zp14;
begin integer Yp14;
  lnr(4); if protect then ERR("protection error in form proc");
  Yp14 := NULL; if Zi = 1 then
  1: ASSIGN(Zf, IN(5)) else
  2:
    begin Yp14 := IN(5); protect := true end;
    if Yp14 = NULL then ERR("no assignment to proc ident"); Zp14 := Yp14;
    protect := false
  end;
end;
integer Zi, Zf, fnn, snn;
fnn := sgn; lnz(2); Zf := NULL; DE(Zf, NULL); snn := sgn;
for Zi := 1 do
3:
begin ERASE(snn); ASSIGN(Zf, IN(5)) end;
begin integer Zg, fnn;
fnn := sgn; lnr(6); Zg := NULL; DE(Zg, NULL); go to 4; ERASE(fnn)
end;
ERASE(fnn)
end

Example 5.4.

begin formula procedure p7;
begin formula procedure p8;
begin formula procedure p9;
    begin f := 5; p9 := f := p8 := h[1 + the of h[3]] :=
    p7 := h[3 + el 1 of h[4] + length of h[5]] + h[4];
    f := 5
    end;
f := 5;
if 1 = 1 then begin f := 5; f := p8 := h[1] := p7 := 5 end
else begin p8 := 5; f := 5; p8 := 5;; end
end;
for i := 1 do p7 := 5
end;
switch S := S[1], S[2], if 0 < 1 then S[3] else S[4], 1, 2, 3;
array a[1:10]; real array b[1:10, 2:12];
Boolean array N[1 + 1 + 2:3 + 4 + 5];
integer array ia, ib, ic[0:0], id[1:1, 1:1];
formula f := 3.14, g = 3.14;
formula array h[1:10];
; end i

Translation 5.4.

begin integer lowci, upci, ci;
integer fn;
fn:= gn; lnrc(2); lowc1:= 1; uplc1:= 10;
begin

integer procedure Zp7;
begin

integer procedure Zp8;
begin

integer procedure Zp9;
begin integer Yp9;
lnrc(5); if protect then ERR(4 protection error in form proc#);
Yp9:= NULL; ASSIGN(Zf, IN(5));
Yp9:= Zp8:= Yp7:= ASSIGN(Zf, ASSIGN(Zh[1 + ARLHS(V(Zh[3]))],
S(V(Zh[3 + ARLHS(1,V(Zh[4]))] + dump(V(Zh[5]))), V(Zh[4])))));
protect:= true; ASSIGN(Zf, IN(5)); if Yp9 = NULL then ERR(4 no assignment to proc ident#); Zp9:= Yp9
end;

lnrc(4); ASSIGN(Zf, IN(5)); if 1 = 1 then
begin ASSIGN(Zf, IN(5));
Zp8:= Yp7:= ASSIGN(Zf, ASSIGN(Zh[1], IN(5))); protect:= true
end
else
begin Zp8:= IN(5); ASSIGN(Zf, IN(5)); Zp8:= IN(5); ; ; end
end;

integer Yp7;
lnrc(3); if protect then ERR(4 protection error in form proc#);
Yp7:= NULL;
for Zi:= 1 do
begin Yp7:= IN(5); protect:= true end;
if Yp7 = NULL then ERR(4 no assignment to proc ident#); Zp7:= Yp7;
protect:= false
end;

switch ZS:= ZS[1], ZS[2], if 0 < 1 then ZS[3] else ZS[4], 1, 2, 3;
array Za[1:10];
real array Zb[1:10,2:12];
boolean array ZN[1 + 1 + 2:3 + 4 + 5];
integer array Zia, Zib, Zic[0:0], Zid[1:1,1:1];
integer array Zn[low1c1:up1c1];
integer Zg, Zf, fnn;
fnn:= gnn; Zf:= Zg:= NULL; DE(Zf, RN(3.14)); DEVAL(Zg, RN(3.14));
for i1:= low1c1 step 1 until up1c1 do
begin DE(Zh[i1], NULL) end;
; ; ERASE(fnn)
end;
ERASE(fnn)
end

7.6. Assignment to formula procedure identifier

Example 6.1 demonstrates the complicated way variables are translated in an assignment statement to a formula procedure identifier. Due regard is given to the fact that a variable is local or global with respect to the procedure body, whether it is a formal parameter or not.

Example 6.1.

begin formula x,y;
formula procedure p1;
begin formula f:= x * y + x/y;
begin formula procedure p2; p2:= lhs of f;
p1:= p2;
p1:= p3(p2)
end
end;

formula procedure p3(p4); formula procedure p4;
begin p3:= p4;
p3:= if p4 = x then p4 else
 if p4 = y then p4 else p4;
begin procedure p5(p6,g);
begin

begin

integer procedure Zp1;
begin integer Yp1, Zf, fnn;
  fnn:= grn; lnr(4); if protect then ERR(
  +protection error in form proc$); Yp1:= NULL; Zf:= NULL;
  DE(Zf, S(F(V(Zx), V(Zy)), Q(V(Zx), V(Zy))));
begin

  integer procedure Zp2;
begin lnr(5); Zp2:= LHS(V(Zf)) end;

  lnr(5); Yp1:= abs(Zp2); protect:= true; Yp1:= Zp3(Zp2);
  protect:= true
end;

ERASE(fnn); if Yp1 = NULL then ERR(4no assignment to proc ident$);
Zp1:= Yp1; protect:= false
end;

integer procedure Zp3(Zp4);
integer procedure Zp4;
begin integer Yp3;
  lnr(12); if protect then ERR(4protection error in form proc$);
  Yp3:= NULL; Yp3:= abs(Zp4); protect:= true;
  Yp3:= if abs(Zp4) = V(Zx) then abs(Zp4) else if abs(Zp4) = V(Zy) then
  abs(Zp4) else abs(Zp4); protect:= true;
begin
procedure \texttt{zp5}(\texttt{zp6}, \texttt{zg});
\texttt{integer} \texttt{zp5}, \texttt{zp6}, \texttt{zg};
\texttt{begin} \texttt{procedure} \texttt{zp5}; \texttt{integer} \texttt{zg};
\texttt{begin} \texttt{lnr}(18);
\texttt{yp3:= if abs}(\texttt{zp6}) = \texttt{v}(\texttt{zx}) \texttt{then abs(vn}(\texttt{zg}) \texttt{else if abs}(\texttt{zp6}) = \texttt{v}(\texttt{zy}) \texttt{then abs}(\texttt{zp6}) \texttt{else s}(\texttt{vn}(\texttt{zg}), \texttt{zp6}); \texttt{protect := true};
\texttt{assign}(\texttt{zg}, \texttt{if abs}(\texttt{zp6}) = \texttt{v}(\texttt{zx}) \texttt{then vn}(\texttt{zg}) \texttt{else if abs}(\texttt{zp6}) = \texttt{v}(\texttt{zy}) \texttt{then zp6 else s}(\texttt{vn}(\texttt{zg}), \texttt{zp6}))
\texttt{end};
\texttt{lnr}(16); \texttt{zp5}(\texttt{zp4}, \texttt{zx})
\texttt{end};
\texttt{if yp3 = null then err(4no assignment to proc ident\#); zp3 := yp3;}
\texttt{protect := false}
\texttt{end};
\texttt{integer} \texttt{zy}, \texttt{zx}, \texttt{frn};
\texttt{frn := gnn; lnr}(2); \texttt{zx := zy := null; de}(\texttt{zx}, \texttt{null}); \texttt{de}(\texttt{zy}, \texttt{null});
\texttt{erase}(\texttt{frn})
\texttt{end}

7.7. Error detection and error recovery

Example 7.1 demonstrates a very bad ABC ALGOL program and the resulting translation, which, with some luck, is indeed what the programmer wanted. The output from the printer is reproduced on the next pages so that one can see the actual error messages.

Examples 7.2 - 7.4 demonstrate the reaction of the compiler on some other erroneous programs. Examples which blow up the compiler are, obviously, until yet not known.
Example 7.1.

begin formula f,g;
  formula array fa,fg[1:2];
  f:= (fx(f+fa[(g+fg fX(g+f)]+fg[1]);
  g:= f; replace(f,1,g);
  g:= f+g)xf; printtext(4abcde14)
end i

Translation 7.1.

begin integer low1c1, up1c1, i1;
  integer fnn;
  fnn:= gnn; lnr(1); low1c1:= 1; up1c1:= 2;
begin integer array Zfa, Zfg[low1c1:up1c1];
  integer Zg,Zf,fn;
  fn:= gnn; Zf:= Zg:= NULL; DE(Zf,NULL); DE(Zg,NULL);
  for i1:= low1c1 step 1 until up1c1 do
    begin DE(Zfa[i1],NULL); DE(Zfg[i1],NULL) end;
    ASSIGN(Zf,P(V(Zf),S(S(V(Zf),V(Zfa[(V(Zg) + V(Zfg[V(Zf)]) + (V(Zg) + V(Zf))]))))),V(Zfg[1]))); ASSIGN(Zg,V(Zf)); Zreplace(V(Zf),1,V(Zg));
    ASSIGN(Zg,S(V(Zf),V(Zg))); printtext(4abcde14); ; ERASE(fnn)
  end;
  ERASE(fnn)
end

begin lnr(0); end
<table>
<thead>
<tr>
<th>CONTENTS OF INF LIST</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>7</td>
<td>20</td>
<td>9</td>
<td>14</td>
<td>1001</td>
<td>44</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>100</td>
</tr>
<tr>
<td>18</td>
<td>10</td>
<td>9</td>
<td>140</td>
<td>5</td>
<td>14</td>
<td>5</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>100</td>
</tr>
<tr>
<td>27</td>
<td>3</td>
<td>29</td>
<td>8</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>34</td>
<td>6</td>
<td>36</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>40</td>
<td>6</td>
<td>41</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>46</td>
<td>3</td>
<td>47</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>49</td>
<td>10</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>58</td>
<td>3</td>
<td>60</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>57</td>
<td>3</td>
<td>69</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>100</td>
</tr>
</tbody>
</table>

---

**ERROR**

0 PROGRAM DOES NOT BEGIN WITH BEGIN

**UNKNOWN**

---

**ERROR**

0 STATEMENT NOT PROPERLY CLOSED

---

**ERROR**

0 PROGRAM DOES NOT BEGIN WITH BEGIN

**UNKNOWN**

---

**ERROR**

0 STATEMENT NOT PROPERLY CLOSED

---

**ERROR**

0 PROGRAM DOES NOT BEGIN WITH BEGIN

**UNKNOWN**

---

**ERROR**

0 STATEMENT NOT PROPERLY CLOSED

---

**ERROR**

0 PROGRAM DOES NOT BEGIN WITH BEGIN

**UNKNOWN**

---

**ERROR**

0 STATEMENT NOT PROPERLY CLOSED

---
Example 7.2.

\begin{verbatim}
begin formula a,b,c;
a := lhs b;
  if true then c := rhs a
end i
\end{verbatim}

Translation 7.2.

\begin{verbatim}
begin integer Zc, Zb, Za, fn;
  fn := gmn; lr(2); Za := Zb := Zc := NULL; DE(Za, NULL); DE(Zb, NULL);
  DE(Zc, NULL); ASSIGN(Za, LHS(V(Zb))); 
  if true then ASSIGN(Zc, RHS(V(Za))); ERASE(fmn)
end
\end{verbatim}

Example 7.3.

\begin{verbatim}
begin integer procedure p(a); p:= 1;
  integer procedure p1(a)p1:=p(b);p1(1)
end i
\end{verbatim}

Translation 7.3.

\begin{verbatim}
begin

  integer procedure Zp(Za);
  begin lr(2); Zp := 1 end;

  integer procedure Zp1(Za);
  begin lr(3); Zp1 := Zp(Zb) end;

  lr(2); Zp1(1)
end
\end{verbatim}
Example 7.4.
begin procedure p(a);
  procedure p1(a):p(a);p1(1)
end;

Translation 7.4.
begin

  procedure Zp(Za);
  begin inr(1); end;

  procedure Zp1(Za);
  begin inr(2);
    Z1: Zp(Za)
  end;

  inr(1); Zp1(1)
end
8. Machine dependencies

The machine-dependent characteristics of the ABC ALGOL system are described in this chapter.

Firstly, the restrictions are described with respect to size of tables as imposed by the memory size and word length of the particular EL-X8 computer used.

Secondly, short descriptions are given of the standard procedures (in particular for input-output) of the MC-ALGOL 60 system.

Thirdly, the dependencies of the ABC ALGOL system are described with respect to the internal representation of characters and to the way basic symbols, in particular word delimiters, are built up.

8.1. Size of tables

The information list, in the form of the integer array "contents of" has 5000 array elements. The number 5000 appears two times in the program at line 2810 and line 2830. The first time as the actual bound of the array, the second time as the value of the variable "max of inf list".

The integer array "code table" has 400 elements. To each different identifier corresponds one array element. The number 400 appears at lines 2810 and 2830 as array bound and as value of "max nr of identifiers".

For formula expressions three integer arrays: "PI", "PT" and "PM", each with 100 elements, are declared on line 1714. They are used for storing the tree structure of formula expressions. For each of the arrays one array element is used per operator. No space is used for operands. The arrays are declared in a recursive procedure with the effect that more than one version of the arrays may simultaneously exist if the formula expression contains as primary a function designator with formula expressions as actual parameters or a subscripted variable with a formula expression somewhere appearing in the arithmetic expression.

The size of the above tables can be enlarged without any repercussions (the maximum bound must fit into an integer, however).

The size of the text array is 10000, a number appearing on line 2072, and implicitly, i.e. multiplied by 3 and then decreased by one, on line 2076.
The text array is capable of containing 30000 symbols. Lay-out (except new lines) and comment is deleted, symbols composed of several characters, as word delimiters and identifiers, are represented by one symbol, in the text array.

There are 400 different codes available for the symbols. If the code is greater than 180 it is the code for an identifier. The code 399 has a special function in that it indicates that the following two codes are used to identify an identifier. This means that there are maximally 399×400 + 399 = 159999 different identifiers possible if the size of code (400) were the only restriction.

The integer capacity of the EL - X8 is $2^{26}$; the code size 400 is therefore, optimal. If the ALOOOL system had byte operations, a code size 256 would have been more economic. In that case the special function of 399 should be taken over by 255.

Normally, letters do not occur in the text array as they are used in word delimiters and in identifiers. One may not use this feature, however, to fill the text array more economically since letters may occur in strings.

A typical case shows how many symbols are needed: The program of chapter 3 section 3.1-5 consists (without comment) of 800 lines and needs 12 500 symbols in the text array.

The size 10000 is bounded not only by the memory size of the computer but also by the word size of an integer. The reason is that a pointer into the text array, pointing to a symbol, having the value 0 - 29999, is combined with the line number of that symbol to be stored in an element of an information cell. The integer value is built up as follows (see line 2235):

\[
\text{ptr to first letgit} = \text{ptr to text} \times 2000 + \text{line number}.
\]

The value of "ptr to first letgit" is used while treating a declaration. This representation automatically restricts the value of "line number" to 1999.

8.2. The NC-ALOOG 60 standard procedures

We now give an alphabetic list of the headings of the standard procedures used in the ABC ALOOOL system together with a short description. Almost all procedures may be used in capital letters as well as in small letters. We use capital letters where both versions may be used and small letters if the small-letter version may be used only.
procedure ABSFIXT(n,m,x); value n,m,x; integer n,m; real x;
Prints in fixed-point notation, n digits before and m digits after the
decimal point, the absolute value of x without a sign.

procedure CARRIAGE(n); value n; integer n;
Has an effect, if n ≥ 0, n times NLCR.

real procedure compose(a,b); value a,b; integer a,b;
Delivers the real number as a result of composing from two X8 machine words,
with values a and b, one real number.

integer procedure EVEN(n); value n; integer n;
EVEN := (-1)^n.

procedure EXIT; Discontinues the execution of the ALOOL program.

procedure FIXT(n,m,x); value n,m,x; integer n,m; real x;
Prints in fixed-point notation, n digits before and m digits after the
decimal point, the value of x, always preceded by the sign of x.

integer procedure head of(x); value x; real x;
Delivers the integer number being the value of the first machine word of
the two machine words in which the value of x is represented.

procedure NEW PAGE;
Produces a new page on the line printer. The first character to be printed
will be printed at the first position of the first line.

procedure NLCR;
Produces a new line on the line printer. The first character to be printed
will be printed at the first position of the line. (60 lines on a page
invokes a NEW PAGE).

integer procedure print pos;
Determines at any moment the position on the line of the first character to
be printed.
procedure PRINTTEXT(s); string s;
Prints the string s without the string quotes ® and ¢.

procedure PRSYM(n); value n; integer n;
Prints a character, according to the value of n. The internal representation table for MC ALOOL is used which is given here in a super condensed form:

<table>
<thead>
<tr>
<th>character</th>
<th>code</th>
<th>other characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>digit 0 - 9</td>
<td>0 - 9</td>
<td>+ - x / ↑ = &lt; &gt; ÷ v ^</td>
</tr>
<tr>
<td>letters a - z</td>
<td>10 - 35</td>
<td>, . ; : ( ) [ ] &quot; ' &quot; _</td>
</tr>
<tr>
<td>letters A - Z</td>
<td>37 - 62</td>
<td>Space New Line</td>
</tr>
<tr>
<td>with codes: 64 - 127</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note that, during the initialization, the ABC ALOOL compiler assigns values to variables like "plus symbol" or "space symbol" by means of characters read (RESYM) and not by assignment of the form: "plus symbol:= 64; space symbol:= 93;".

procedure PUSYM(n); value n; integer n;
Punches the bit pattern in paper tape, according to MC - Flexewriter code, of the character with internal representation given by the value of n.

procedure PUTEXT(s); string s;
Punches bit patterns in paper tape, according to MC - Flexewriter code, of the characters of the string without the string quotes ® and ¢.

integer procedure REMAINDER(a,b); value a,b; real a,b;
REMAINDER:= if b = 0 then a else a - a ÷ b × b.

integer procedure RESYM;
Delivers the internal representation of the next character from input paper tape. The paper tape is advanced one character.

procedure RUNOUT;
Punches a piece of blank paper tape.

procedure SPACE(n); value n; integer n;
"Prints" n spaces on the line printer.

integer procedure STRING SYMBOL(k,s);
   value k; integer k; string s;
Delivers the internal representation of the (k+1) th character of the string s.
This procedure is used to have permanent tables, which do not need to be initialized. (See the procedures text symbol (line 1928), standard symbol (line 2414) and catalogue symbol (line 2873)).
To circumvent the use of this procedure one can introduce an integer array which is filled with information from input paper tape or from a file during the initialization.

integer procedure tail of(x); value x; real r;
Delivers the integral number being the value of the second machine word of the two machine words in which the value of x is represented.

real procedure time;
Delivers the time, measured in seconds, elapsed after the moment the ALGOL 60 program has been subjected to the ALGOL 60 compiler. The accuracy is .01 sec.

8.3. Other machine-dependent features

The treatment of characters is based on the table of internal representations as in use for the UC-ALGOL system. In particular, paper tape is used as input - and as output medium. A card version of the compiler exists also.
To publish the compiler in a most readable form lead us to use paper tape and underlined word delimiters, instead of key words or words between apostrophes.
If one wants to use the apostrophe representation, changes have to made in the following procedures:
. READ synt unit (line 2146), where the reaction to the "underlining symbol" has to be changed.
.INITIALIZE symbols (line 1949), where the word delimiters are read from input tape in order to give initial values to the variables "goto symbol" - "rowadic symbol".

.3R synt unit (line 2654), which prints the word delimiters.
.SHOW (line 2724), which also prints the word delimiters.

The special symbol "," which must close every ABOOOL program is underlined. If one wants to change this into ' or ';', one has to change NS only.

If one wants the string quotes \ and ] to be changed, one has to change "READ synt unit" (line 2203), "early end of program" (line 2132), which inserts right string quotes, "translate string" (line 1651) and some places where run-time error messages are produced (e.g. lines 721 or 749). The procedure "print catalogue" (line 2873) should also be changed.

The word length of the EL - X8 computer imposes restrictions on line and rhs quantities in constant and monadic formula values, and in the choice of the radix of the rational number system of chapter 3.

With respect to the information list cells we, finally, remark that a type value may have the form:

\[
synt unit1 \times 2 \uparrow 15 + synt unit2 \times 2 \uparrow 5 + mode.
\]

This implies that an integer word should be large enough to contain the number \(180 \times 2 \uparrow 15\), 180 being the maximum number of "synt unit1" and "synt unit2".

The definition of the output for the operators +, x, /, etc. in lines 2963 - 2973 is rather machine-dependent as the internal representation of the characters S, P, Q, etc. is used explicitly.
9. References

   Comm ACM 14 nr 8. (August 1971)
   Journal ACM 18 nr 4. (October 1971)

   language.

   Mathematisch Centrum, Amsterdam

   Report LR 2.5
   Mathematisch Centrum, Amsterdam

   Report LR 1.1
   Mathematisch Centrum, Amsterdam

   Comm ACM 14 nr 1. pp 39-45 (Jan 1971)

   Addison-Wesley (1968)

   Addison-Wesley (1969)

   lijststelsels.
   Informatie 10 nr 12, pp 285-291.

    ALGOL 60.
Numerische Mathematik 4, pp 420-453 (1963)

MC TRACTS 17 (2nd edition)
Mathematisch Centrum, Amsterdam (1970)

MC TRACTS 18
Mathematisch Centrum, Amsterdam (1968)

Report TW 110
Mathematisch Centrum, Amsterdam (1969)

[14] Roever, W.F. de: An exact rational function system with
garbage collection in ALGOL 60.
Report NR 119/70
Mathematisch Centrum, Amsterdam (1970)

[15a] Sammet, J.E.: An annotated description based bibliography
on the use of computers for non-numerical Mathematics.
Computing Reviews 7 nr 4. (July-August 1966)

[15b] - - -: Programming languages: History and fundamentals.

[16] Velden, G. ten: Simplification procedures for ABC ALGOL.
Report NR 136/72
Mathematisch Centrum, Amsterdam (1972)

of ALGOL.
Comm ACM 9, pp 413-431 (1966)

[18] Wijngaarden, A. van (Editor), B.J. Mailloux, J.E.L. Peck,
ALGOL 68.
Numerische Mathematik, 14, pp 79-218 (1969)
OTHER TITLES IN THE SERIES MATHEMATICAL CENTRE TRACTS

A leaflet containing an order-form and abstracts of all publications mentioned below is available at the Mathematisch Centrum, Tweede Boerhaavestraat 49, Amsterdam-1005, The Netherlands. Orders should be sent to the same address.

MCT 7 W.R. VAN ZWET, Convex transformations of random variables, 1964. ISBN 90 6196 007 X.
MCT 13 H.A. LAUWERIER, Asymptotic expansions, 1966, out of print; replaced by MCT 54.

MCT 22 T.J. DEKKER, ALGOL 60 procedures in numerical algebra, part 1, 1968. ISBN 90 6196 029 0.


MCT 28 J. OOSTERHOFF, Combination of one-sided statistical tests, 1969. ISBN 90 6196 041 X.


MCT 42 W. VERAAT, Successive epochs in Bernoulli trials (with applications in number theory), 1972. ISBN 90 6196 077 0.


ISBN 90 6196 083 5.


MCT 56 M. Hall Jr. & J.H. van Lint (eds.), Combinatorics, part 2: Graph theory; foundations, partitions and combinatorial geometry, 1974. ISBN 90 6196 100 9.


MCT 65 J. de Vries, Topological transformation groups I A categorical approach, 1975. ISBN 90 6196 113 0.

* MCT 67  H.A. LAUWERIER, Asymptotic analysis. part 2. ISBN 90 6196 119 X.

* MCT 68  P.P.N. DE GROEN, Singulary pertlibed differential operators of second order. ISBN 90 6196 120 3.


* MCT 70  W.P. DE ROEVER JR., Recursive program schemes: semantics and proof theory. ISBN 90 6196 127 0.


* MCT 72  J.K.M. JANSSEN, Simple periodic and nonperiodic lamé functions and their applications in the theory of conical waveguides. ISBN 90 6196 130 0.

* MCT 73  D.M.R. LEIVANT, Absoluteness of intuitionistic logic. ISBN 90 6196 122 x.


An asterisk before the number means "to appear".