

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

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

MATHEMATICAL CENTRE TRACTS 47

R.P. VAN DE RIET

ABC ALGOL
A PORTABLE LANGUAGE FOR
FORMULA MANIPULATION SYSTEMS

PART 2 : THE COMPILER

MATHEMATISCH CENTRUM AMSTERDAM 1976

AMS(MOS) subject classification scheme (1970): 68A15

ACM-Computing Reviews-categories: 4.22

first edition 1973

second edition 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.

1 begin comment

2 6. The ABC ALGOL compiler

3

4 6.0. Preliminaries of translation process

5

6 Algorithm::

7

8 integer preceding begin,begin;

9

10 comment:

11 The above integers are global in the translation process. Their values

12 point to the information cells of the last two treated begin's.

13

14 6.1. Prerequisites for translation process

15

16 Syntactic definition:

17

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

19

20 A program may be preceded by comment.

21

22 Algorithm::

23

24 procedure envelope of block(procedure body);

25 value procedure body; integer procedure body;

26 begin integer ptr to integer, ptr to formula, nr of array segments,

27 ptr to array segment, max dimension in array declarations;

28 Boolean formula block, dangerous inner block, smn necessary,

29 block contains labels, block contains gotos, block,

30 proc id ass stat, interested in proc id;

31

32 comment:

33 "envelope of block" translates a block, a compound statement or a

34 statement and treats them all as blocks. It is called from the main

35 program to translate an ABC ALGOL program, from a procedure body (in

36 which case the value of "procedure body" is # 0 indicating where the

37 information about the procedure may be found) or for the translation

38 of a real block.

39 The procedure contains the declarations of the above
 40 block-administrative variables and all the syntax-translating
 41 procedures as "block or statement", "declaration" or "statement". The
 42 effect is that the above variables are global to these procedures,
 43 while they are local within the procedure "envelope of block".
 44 Each block is translated by a call of "envelope of block", thus each
 45 block has attached to it a private set of the above variables, which
 46 are initialized by a call of "Open new block in information list".

47

48 6.2. Translation of a block

49

50 Syntactic definition:

51

52 <block> ::= <block head> <compound tail>

53 <block head> ::= begin <declaration with semicolon> |

54 <block head> <declaration with semicolon>

55 <declaration with semicolon> ::= <declaration>;

56 <compound tail> ::= <statement> end | <statement>; <compound tail>

57

58 Algorithm:

59

60 procedure block or statement;61 begin integer lnr; block := synt unit = begin symbol;

62 Open new block in information list;

63 PR string(begin †); lnr := line number;64 if ptr to array segment ≠ 0 then65 begin comment66 In this block formula arrays are declared. Therefore, it is
67 surrounded by another block in which the array bounds are evaluated;

68 SAVE reading ptrs; array bounds; PR nlcr;

69 PR string(begin †); RESET reading ptrs70 end;71 comment72 The next statement serves to determine whether this block is the
73 procedure body of a formula procedure;74 if block then75 begin RE begin; for synt unit := synt unit76 while is declarator(synt unit) do declaration


```

77  end;
78  interested in proc id:= if procedure body  $\neq$  0 then
79    contents of [procedure body - 4] : 32 =
80    formula symbol  $\times$  1024 + procedure symbol
81    else false;
82  if second scan then
83  begin SAVE reading ptrs;
84    if ptr to array segment  $\neq$  0 then
85    begin Declare formula arrays; Declare integers end else
86    begin Declare integers; PR string( $\dagger$ lnr( $\dagger$ );
87      PR int num(lnr); PR string( $\dagger$ ); $\dagger$ )
88    end;
89    if interested in proc id then procedure block entry;
90    Introduce names for formulae;
91    Introduce names for formula array elements;
92    Initialize snn;
93    RESET reading ptrs
94  end else formula block:= ptr to formula + ptr to array segment  $\neq$  0;
95
96  proc id ass stat:= false;
97  L1: statement(block v formula block);
98  comment the actual parameter of "statement" indicates whether the
99  environment of the statement is such that it can be split into more
100 than one statements as e.g. in a block, or that it should be sur-
101 rounded by begin and end when it is split.;
102 if synt unit  $\neq$  semicolon symbol ^ synt unit  $\neq$  end symbol then
103 begin ERR( $\dagger$ statement not properly closed $\dagger$ );
104   SEEK(synt unit = semicolon symbol v synt unit = end symbol)
105 end;
106 if block then
107 begin L2: if synt unit = semicolon symbol then
108   begin PR and RE semicolon; goto L2 end;
109   if synt unit  $\neq$  end symbol then
110   begin proc id ass stat:= false; goto L1 end
111 end;
112 Block exit; Close block in information list
113 end block or statement;
114

```

115 comment:

116 Each block has a certain amount of administration stored in the
117 information list. The information is "initialized" upon a call of
118 "Open new block in information list" and is "buried" upon a call of
119 "Close block in information list".

120

121 The block: "begin d; s end" is translated into:

122 "begin D; integer fnn,snn; fnn:= gnn; D2; snn:= gnn; S
123 ; ERASE(fnn) end",

124 where d and s stand for a declaration and a statement, respectively.

125 D is the first half of the translation of the declaration d ("formula
126 x:= f" is translated into "integer X").

127 D2 is the introduction of names for formula variables("DE(X,F)"). The
128 locally introduced integers "fnn" and "snn" are being used for erasure
129 of names and thus of formulae. The integer "gnn" is, in the translated
130 program, a global integer which counts the number of names introduced.
131 "ERASE" is a procedure which erases the last "gnn - fnn" introduced
132 names. A labelled statement "l: s" is translated into:

133 "L: ERASE(snn); S",

134 and again "ERASE" erases the last "gnn - snn" introduced names.

135

136 6.3. Translation of a declaration

137

138 Syntactic definition:

139

140 <declaration> ::= <formula declaration> | <formula array declaration> |

141 <formula procedure declaration> | <procedure declaration> |

142 <type declaration> | <array declaration> | <switch declaration>

143

144 The first four declarations are defined in sections 6.3.1, 6.3.2,
145 6.3.3 and 6.3.4 respectively. The other declarations are defined in
146 the ALGOL 60 report [10] section 5.

147

148 Algorithm:;

149 procedure declaration;

150 if synt unit = formula symbol then

151 begin if next synt unit = array symbol then formula array declaration

152 else if next synt unit = procedure symbol then procedure declaration

```

153 else formula declaration
154 end else
155 if synt unit = real symbol v synt unit = integer symbol v
156 synt unit = Boolean symbol then
157 begin if next synt unit = array symbol then array declaration
158 else if next synt unit = procedure symbol then procedure declaration
159 else type declaration
160 end else
161 if synt unit = array symbol then array declaration else
162 if synt unit = procedure symbol then procedure declaration else
163 begin CHECK(switch symbol); PR and RE; PR string(†Z†); PR and RE;
164 PR and RE;
165 L: other expression; if synt unit = comma symbol then
166 begin PR and RE; goto L end; PR and RE semicolon
167 end declaration;
168
169 procedure type declaration;
170 begin integer su; su:= synt unit;
171 if su = integer symbol then
172 begin L1: RE; if first scan then
173 ptr to integer:= STORE into information list(st(st(st(st(0,
174 nr of ident),
175 ptr to first letgit),
176 integer symbol × t15 + declared as value),
177 ptr to integer));
178 RE; if synt unit = comma symbol then goto L1;
179 RE semicolon
180 end else
181 begin L2: PR and RE; if su ≠ Boolean symbol ^ first scan then
182 STORE into information list(st(st(st(0,
183 nr of ident),
184 ptr to first letgit),
185 su × t15));
186 PR string(†Z†); PR and RE; if synt unit = comma symbol then goto L2;
187 PR and RE semicolon
188 end end type declaration;
189
190 procedure array declaration;

```

```

191 begin integer su,nsu; su:= synt unit; nsu:= next synt unit;
192   if nsu = array symbol then PR and RE;
193 L1: PR and RE; if su ≠ Boolean symbol ^ first scan then
194   STORE into information list(st(st(st(0,
195     nr of ident),
196     ptr to first letgit),
197     (if nsu = array symbol then su else real symbol) × t15 +
198     32 × array symbol));
199   PR string(↓Z↓); PR and RE;
200   if synt unit = sub symbol then
201     begin L2: PR and RE; other expression; CHECK(colon symbol);
202     PR and RE;
203     other expression; if synt unit = comma symbol then goto L2;
204     CHECK(bus symbol); PR and RE
205   end; if synt unit = comma symbol then goto L1; PR and RE semicolon
206 end array declaration;
207
208 comment:
209 The other declarations will be treated in the next sections.
210
211 6.3.1. Translation of a formula declaration
212
213 Syntactic definition:
214
215 <formula declaration> ::= formula <formula list>
216 <formula list> ::= <formula definition>|<formula list>,
217 <formula definition>
218 <formula definition> ::= <simple variable><formula initialization>
219 <formula initialization> ::= <empty>| := <formula expression>|
220                               = <formula expression>
221 <simple variable> ::= <identifier>
222
223 Algorithm:;
224
225 procedure formula declaration;
226 begin L: RE; if first scan then
227   ptr to formula:= STORE into information list(st(st(st(st(0,
228     nr of ident),

```

```

229         ptr to first letgit),
230         formula symbol × t15 + (if next synt unit = equal
231         symbol then declared as value else declared as name)),
232         ptr to formula));
233     SKIP text until(synt unit = comma symbol ∨ synt unit =
234         semicolon symbol);
235     if synt unit = comma symbol then goto L;
236     RE semicolon
237 end formula declaration;
238
239 comment:
240 The following procedures declare all integers, introduce all names and
241 produce the initialization for smn. note that "formula f:= expr1,g =
242 expr2;" is translated into: "integer G,F;".
243 It is necessary that names are introduced for "F" and "G" and that
244 the initial values are computed immediately after the translated
245 declarations.
246
247 Algorithm;
248
249 procedure Declare integers;
250 begin integer a,a0,a3,mode; Boolean first,reverse;
251     procedure PR int com;
252     if first then
253     begin PR string({integer †); first:= false end
254     else PR string(†,†);
255     first:= true; reverse:= false; a0:= 0;
256     for a:= ptr to integer, ptr to formula do
257     begin for a:= a while a ≠ 0 do
258     begin SET reading ptrs on(contents of [a + 2]);
259     a3:= contents of [a + 3]; mode:= a3 - a3 : 32 × 32;
260     if mode = with local ∨ mode = specified as value then
261     begin PR int com; PR string({Y†); if operator identifier then
262     PR operator else PR synt unit
263     end else if mode ≠ without local then
264     begin PR int com; PR string({Z†); PR synt unit end
265     else operator identifier:= false;
266     a3:= contents of [a + 4]; if reverse then

```

```

267     begin contents of [a + 4]:= a0; a0:= a end; a:= a3
268     end; reverse:= true
269     end; ptr to formula:= a0;
270     if formula block then
271     begin PR int com; PR string(†fnn†);
272     if snn necessary then
273     PR string(†,snn†); PR string(†; fnn:= gnn†)
274     end; if ¬ first then PR string(†;†)
275 end Declare integers;
276
277 procedure Introduce names for formulae;
278 begin integer a,a3,mode; Boolean first,print NULL;
279     print NULL:= false; first:= true;
280 L: a:= ptr to formula; if a ≠ 0 then PR nlcr;
281     for a:= a while a ≠ 0 do
282     begin a3:= contents of [a + 3]; mode:= a3 - a3 : 32 × 32;
283     if first then
284     begin if mode ≠ specified as value then
285     begin PR string(†Z†); SET reading ptrs on (
286     contents of [a + 2]); PR and RE; print NULL:= true;
287     PR string(†:= †)
288     end end else
289     begin if mode = specified as value v mode = declared as value then
290     PR string(†DEVAL(†) else PR string(†DE(†);
291     if mode = specified as value then PR string(†Y†) else
292     PR string(†Z†);
293     SET reading ptrs on (contents of [a + 2]); PR and RE;
294     PR string(†,†); if mode = specified as value then
295     begin PR string(†Z†); SET reading ptrs on (contents of [a + 2]);
296     PR synt unit
297     end else
298     if synt unit = becomes symbol v synt unit = equal symbol then
299     begin RE; formula expression end else PR string(†NULL†);
300     PR string(†);†)
301     end;
302     a3:= contents of [a + 4]; a:= a3
303 end;
304 if first then

```

```

305  begin first:= false; if print NULL then PR string(↓NULL;↓);
306      goto L
307  end
308  end Introduce names for formulae;
309
310  procedure Initialize snn;
311  if formula block ^ snn necessary then
312      PR string(↓ snn:= gnn; ↓);
313
314  comment:
315
316  6.3.2. Translation of a formula array declaration
317
318  Syntactic definition:
319
320  <formula array declaration> ::= formula array <array list>
321  For array list see the ALGOL 60 report [10] section 5.2.
322
323  Algorithm::
324
325  procedure formula array declaration;
326  begin integer a,dimension; RE;
327  ARRAY SEGMENT: RE; if first scan then
328      begin ptr to array segment:= a:=
329          STORE into information list(st(st(st(st(st(0,
330              nr of ident),
331              ptr to first letgit),
332              formula symbol × t15 + 32 × array symbol),
333              ptr to array segment),0));
334          nr of array segments:= nr of array segments + 1
335      end;
336      dimension:= 1;
337  NEXT IDENTIFIER: RE;
338  if synt unit = comma symbol then
339      begin RE; if first scan then
340          STORE into information list(st(st(st(0,
341              nr of ident),
342              ptr to first letgit),

```

```

343     formula symbol × t15 + 32 × array symbol)); goto NEXT IDENTIFIER
344 end else
345 if synt unit = sub symbol then
346 begin BOUND PAIR: RE; SKIP text until(synt unit = comma symbol v
347     synt unit = bus symbol);
348     if synt unit = comma symbol then
349     begin dimension:= dimension + 1; goto BOUND PAIR end;
350     if first scan then
351     begin contents of[a + 5]:= dimension; if max dimension in array
352     declarations < dimension then
353     max dimension in array declarations:= dimension
354     end; RE;
355     if synt unit = comma symbol then goto ARRAY SEGMENT
356 end; RE semicolon.
357 end formula array declaration;
358
359 comment:
360 The translation of:
361     "formula array a,b[e1:e2],c,d[e3:e4,e5:e6];"
362 is:
363     "integer array C,D[low1c1:up1c1,low1c2:up1c2]],A,B[low2c1:up2c1];"
364 It is necessary that the bounds e1,e2,e3,e4,e5 and e6 are evaluated in
365 a surrounding block and assigned to the array elements of "low" and
366 "up". For this example, the next procedure "array bounds" produces the
367 following text:
368     "integer low1c1,up1c1,low2c1,up2c1,i1,
369     low1c2,up1c2,low2c2,up2c2,i2;
370     low1c1:= E3; up1c1:= E4;
371     low1c2:= E5; up1c2:= E6;
372     low2c1:= E1; up2c1:= E2;"
373
374 In a case that the block is the body of a procedure, it is necessary
375 to treat the procedure parameters first, i.e. declaring integers and
376 producing names for the formal parameters. This is done in a somewhat
377 tricky way in that the list of formula variables is split into two
378 lists: one consisting of the parameters and the other consisting of
379 the ordinary variables declared in the heading of the block. The
380 latter list has to be treated together with the formula array

```



```

381 declaration, i.e. in the proper block. The first list is treated in
382 this procedure. Note that the line number is treated also.
383
384 Algorithm::
385
386 procedure array bounds;
387 begin integer dimension,a,index,m,ptf,pti,lnr; boolean sn;
388   lnr:= line number; a:= ptf:= ptr to formula;
389   ptr to formula:= index:= 0;
390   for a:= a while a ≠ 0 do
391     begin m:= contents of[a + 3]; m:= m - m : t15 × t15;
392     if m = declared as value v m = declared as name then
393       begin index:= a; a:= contents of [a + 4] end else
394       begin if index ≠ 0 then contents of [index + 4]:= 0;
395         ptr to formula:= a;a:= 0
396       end
397     end;
398   PR nlcr; PR string(†integer †);
399   for a:= 1 step 1 until max dimension in array declarations do
400     begin for index:= 1 step 1 until nr of array segments do
401       PR sn(†low†,index,PR sn(†c†,a,
402         PR sn(†,up†,index,PR sn(†c†,a,PR string(†,†)))));
403       PR sn(†i†,a,if a < max dimension in array declarations then
404         PR string(†,†) else PR string(†;†)); PR nlcr
405     end;
406   sn:= sm necessary; sm necessary:= false;
407   pti:= ptr to integer; ptr to integer:= 0;
408   Declare integers; PR string(†lnr†);
409   PR int num(lnr); PR string(†;†);
410   Introduce names for formulae;
411   ptr to formula:= ptf; ptr to integer:= pti; sm necessary:= sn;
412   a:= ptr to array segment; index:= 0;
413
414   for a:= a while a ≠ 0 do
415     begin PR nlcr; SET reading ptrs on(contents of [a + 2]);
416     dimension:= contents of [a + 5]; index:= index + 1;
417     FIND SUB SYMBOL:
418     RE; if synt unit ≠ sub symbol then goto FIND SUB SYMBOL; RE;

```

```

419   for m:= 1 step 1 until dimension do
420   begin PR sn(↓low↓,index,PR sn(↓c↓,m,PR string(↓:= ↓)));
421   other expression; PR string(↓;↓); CHECK(colon symbol); RE;
422   PR sn(↓up↓,index,PR sn(↓c↓,m,PR string(↓:= ↓)));
423   other expression; PR string(↓;↓);
424   if m < dimension then CHECK(comma symbol) else
425   CHECK(bus symbol); RE
426   end; a:= contents of [a + 4]
427   end
428 end array bounds;
429
430 comment
431 In the following procedure the actual formula array declaration is
432 produced.
433
434 Algorithm::
435
436
437 procedure Declare formula arrays;
438 begin integer dimension,a,n,m,index;
439   index:= 0; PR n1cr; PR string(↓ integer array ↓);
440   a:= ptr to array segment;
441   for a:= a while a ≠ 0 do
442   begin if a ≠ ptr to array segment then PR string(↓,↓);
443   SET reading ptrs on(contents of [a + 2]);
444   index:= index + 1;
445   L1: PR string(↓Z↓); PR and RE; dimension:= 1;
446   if synt unit = comma symbol then
447   begin PR and RE; goto L1 end else
448   if synt unit = sub symbol then
449   begin L2: PR and RE; SKIP text until
450   (synt unit = comma symbol v synt unit = bus symbol);
451   PR sn(↓low↓,index,PR sn(↓c↓,dimension,
452   PR sn(↓: up↓,index,PR sn(↓c↓,dimension,1))));
453   if synt unit = comma symbol then
454   begin dimension:= dimension + 1; goto L2 end;
455   CHECK(bus symbol); PR and RE
456   end; a:= contents of [a + 4]

```

```

457  end;
458  PR string(†;†); PR nlcr
459  end Declare formula arrays;
460
461  comment:
462  We finally have to construct an algorithm for the declaration of the
463  array elements and the introduction of names for them. For the above
464  example, this algorithm gives the following:
465  "for i1:= low1c1 step 1 until up1c1 do
466    for i2:= low1c2 step 1 until up1c2 do
467      begin DE(C[i1,12],NULL); DE(D[i1,i2],NULL) end ;
468      for i1:= low2c1 step 1 until up2c1 do
469        begin DE(A[i1],NULL); DE(B[i1],NULL) end ;"
470
471  Algorithm:;
472
473  procedure Introduce names for formula array elements;
474  begin integer dimension,a,m,index;
475    a:= ptr to array segment; index:= 0;
476    for a:= a while a ≠ 0 do
477      begin SET reading ptrs on(contents of [a + 2]);
478        dimension:= contents of [a + 5]; index:= index + 1;
479        for m:= 1 step 1 until dimension do
480          begin PR nlcr; PR sn(
481            †for i†,m,PR sn(†:= low†,index,PR sn(
482              †c†,m,PR sn(† step 1 until up†,index,PR sn(
483                †c†,m,PR string(† do†))))))
484          end;
485          PR nlcr; PR string(†begin †);
486          L3: PR string(†DE(Z†); PR and RE; PR string(†[†);
487            for m:= 1 step 1 until dimension do
488              PR sn(†i†,m,if m < dimension then PR string(†,†)
489                else PR string(†],NULL)†);
490              if synt unit = comma symbol then
491                begin PR string(†;†); RE; goto L3 end;
492              PR string(† end;†); a:= contents of [a + 4]
493            end
494  end Introduce names for formula array elements;

```

495

496 comment:

497

498 6.3.3. Opening and closing a block.

499

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

503 In these IC's the values of the block-administrative variables
504 declared in section 6.1 and to which values are assigned in sections
505 6.3.1 and 6.3.2, are stored together with pointers for the linkage of
506 the block IC's. During the second scan the stored values are used for:
507 a) Outputting the correct "block begin".

508 b) Searching the IC of an identifier used in a statement.

509

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

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

513 Let the IC of $b[i]$ be: $IC(b[i])$ and the IC of $e[i]$ be $IC(e[i])$, $i =$
514 $1, \dots, n$.

515 Let the blocks be $b[i]$, $i = 1, \dots, n$, such that the begin of $b[i]$ is
516 $b[i]$.

517 Let us define a surrounding block $b[j]$ of $b[i]$ as a block in which
518 $b[i]$ is contained. Obviously, we have $j < i$.

519

520 The immediately surrounding block $b[j]$ of $b[i]$ is the surrounding
521 block of $b[i]$ for which $i-j$ is minimal. Define $s(i)$ to be this j .
522 Obviously, for each $i > 1$, $s(i)$ exists.

523

524 We are now able to describe information stored in $IC(b[i])$ and
525 $IC(e[i])$.

526 $IC(b[i])$ contains, after the first scan is executed:

527 1. A reference to $IC(e[i])$.

528 2. A reference to $IC(b[i + 1])$, in case $i < n$.

529 3. The values of the block-administrative variables.

530 4. The block-administrative variables as mentioned in section 6.1.

531 $IC(e[i])$ contains, after the first scan is executed:

532 1. A reference to $IC(b[s(i)])$, provided $i > 1$.

533

534 Between $IC(b[i])$ and $IC(e[i])$, the IC's of the identifiers, declared
 535 in $b[i]$, are stored in the information list. Upon entry of a new
 536 block, during the second scan, we can now find the corresponding
 537 $IC(b[i])$, since we know $IC(b[i - 1])$, provided $i > 1$. Hence,
 538 the block-administrative variables can be found and, during the
 539 translation of a statement, the IC of an identifier can be found by
 540 means of the following simple algorithm:

541 Step1: Choose $IC(b[i])$. Take Step2.

542 Step2: Choose the next IC.

543 If this IC is the IC of a begin, namely $IC(b[j])$, then choose
 544 $IC(e[j])$ and take Step2 again.

545 If this IC is the IC of an end, namely $IC(e[j])$, then, if
 546 $j = 1$, the process is finished and no IC is found for the
 547 identifier, otherwise choose $IC(b[s(j)])$ and take Step2
 548 again.

549 If this IC is the IC of an identifier, then check whether this
 550 is the identifier we looked for, in which case the process
 551 is finished, otherwise take Step2 again.

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

554

555 Example: The program:556 "begin formula a,b; S1;557 begin formula c;558 procedure P(d); formula d ;559 begin formula e; S2 end ;560 formula f; S3;561 begin formula g; S4 end ;562 end ; S5;563 begin formula h; S6 end564 end"

565 gives rise to the following information list in which the references
 566 are made visible by means of arrows:

567

568

569

570

```

571 begin                                = IC(b[1])
572   a
573   b
574   begin                                = IC(b[2])
575     c
576     P
577     begin                                = IC(b[3])
578       d
579       e
580     end                                = IC(e[3]) s(3) = 2
581     f
582     begin                                = IC(b[4])
583       g
584     end                                = IC(e[4]) s(4) = 2
585   end                                = IC(e[2]) s(2) = 1
586   begin                                = IC(b[5])
587     h
588   end                                = IC(e[5]) s(5) = 1
589 end                                = IC(e[1])
590   information list
591

```

592 A search for the IC of an identifier in S2 leads to the inspection of
593 the IC's of the following identifiers: e,d,c,P,f,a,b. A search for the
594 IC of an identifier in S4 leads to the inspection of the IC's of the
595 following identifiers: g,c,P,f,a,b.
596

597 We give now the algorithm for searching the IC of an identifier given
598 by the global integers: "nr of ident" and "ptr to first letgit". These
599 integers have obtained values by means of the reading procedures "RE"
600 and "PR and RE".
601

602 The global integer "block depth" decreases by one each time the block
603 depth of the block in which the identifier is sought diminishes.
604

605 Algorithm::;

606

607 Boolean procedure Search for identifier (IC); integer IC;

608 begin integer p,j;

```

609  p:= begin; Search for identifier:= false;
610  for p:= p while p ≠ 0 do
611  begin p:= p + contents of [p]; if p ≥ ptr of inf list then goto OUT;
612  j:= contents of [p + 1];
613  if j = -end symbol then
614  begin block depth:= block depth - 1;
615  p:= contents of [p + 2]
616  end else
617  if j = - begin symbol then p:= contents of [p + 2]
618  else if j = nr of ident
619  then begin Search for identifier:= true; IC:= p; goto OUT end
620  end;
621  OUT: end Search for identifier;
622
623  comment:
624  We give now the block-administrative procedures.
625
626  Algorithm::;
627
628  procedure Open new block in information list;
629  if first scan then
630  begin ptr to integer:= ptr to formula:= ptr to array segment:=
631  nr of array segments:= max dimension in array declarations:= 0;
632  block contains labels:= block contains gotos:= false;
633  if procedure body ≠ 0 then
634  begin ptr to integer:= contents of [procedure body + 4];
635  ptr to formula:= contents of [procedure body + 5]
636  end else
637  begin begin:= contents of [preceding begin]:=
638  STORE into information list(st(st(st(st(st(st(st(st(0,
639  - begin symbol),
640  begin),
641  0),
642  ptr to integer),
643  ptr to formula),
644  ptr to array segment),
645  nr of array segments),
646  max dimension in array declarations),

```

```

647         0));
648         preceding begin:= begin + 3
649     end
650 end else
651
652 begin Boolean array b[0:5];
653     if procedure body = 0 then
654         begin begin:= contents of [preceding begin];
655         preceding begin:= begin + 3
656     end;
657     ptr to integer:= contents of [begin + 4];
658     ptr to formula:= contents of [begin + 5];
659     ptr to array segment:= contents of [begin + 6];
660     nr of array segments:= contents of [begin + 7];
661     max dimension in array declarations:= contents of [begin + 8];
662     get bits(contents of [begin + 9],b);
663     formula block:= b[0];
664     block contains labels:= b[1];
665     block contains gotos:= b[2];
666     dangerous inner block:= b[3];
667     sm necessary:= if dangerous procedures then b[5] else b[4]
668 end;
669
670 procedure Close block in information list;
671 if first scan then
672 begin integer prec begin; Boolean array e,b[0:5];
673     prec begin:= contents of [begin + 2];
674     contents of [begin + 2]:=
675     STORE into information list(st(st(0,
676         - end symbol),
677         prec begin));
678     contents of [begin + 4]:= ptr to integer;
679     contents of [begin + 5]:= ptr to formula;
680     contents of [begin + 6]:= ptr to array segment;
681     contents of [begin + 7]:= nr of array segments;
682     contents of [begin + 8]:= max dimension in array declarations;
683
684     comment:

```


685 The following can be understood in connection with section 4.6 only.
 686 In the Boolean array b and e the properties: form, label, goto, dang,
 687 snn1 and snn2 are represented for the array elements with subscripts
 688 0, 1, 2, 3, 4 and 5, respectively. The array b gives the properties
 689 for the block under consideration and the array e gives the properties
 690 for the enveloping block.

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

```

697
698   get bits(contents of [begin + 9],b);
699   b[0]:= formula block;
700   b[1]:= block contains labels;
701   if block contains gotos then b[2]:= true;
702   if b[1] ^ b[3] then b[4]:= true;
703   if b[1] then b[5]:= true;
704   store bits(contents of [begin + 9],b);
705   if prec begin ≠ 0 then
706     begin get bits(contents of [prec begin + 9],e);
707       if b[2] then e[2]:= true;
708       if b[0] ^ b[2] v b[3] then e[3]:= true;
709       if b[4] ^ ¬ b[0] then e[4]:= true;
710       if b[1] ^ ¬ b[0] then e[5]:= true;
711       store bits(contents of [prec begin + 9],e)
712     end; begin:= prec begin
713 end else begin:= contents of [contents of [begin + 2] + 2];
714
715 procedure procedure block entry;
716 comment the procedure is called during the second scan only, while the
717 block is the body of a formula procedure;
718 begin integer a; a:= procedure body - 7;
719   if contents of [a + 6] ≥ 1 then
720     begin PR string(†
721     if protect then ERR(†protection error in form proc†);†);
722     SET reading ptrs on (contents of [a + 2]);

```

```

723     PR nlcr; PR string(↓Y↓); if operator identifier then
724     PR operator else PR synt unit;
725     PR string(↓:= NULL;↓)
726 end end procedure block entry;
727
728 procedure Block exit;
729 if first scan then
730 begin if block then RE end;
731     if interested in proc id then
732     begin if ¬ proc id ass stat then
733         contents of [procedure body - 1]:= 1
734     else contents of [procedure body - 4]:=
735         formula symbol × t15 + procedure symbol × 32 + without local
736 end end else
737 begin integer a,i; Boolean b; PR nlcr;
738     if formula block then PR string(↓; ERASE(fnn)↓);
739     a:= if interested in proc id then procedure body - 7 else 0;
740     if a ≠ 0 then
741     begin if contents of [a + 6] ≥ 1 then
742         begin PR string(↓; if Y↓); SAVE reading ptrs;
743             for i:= 1,2,3 do
744                 begin SET reading ptrs on(contents of [a + 2]);
745                     b:= operator identifier;
746                     if b then PR operator else PR synt unit;
747                     if i = 1 then
748                         begin PR string(
749                             ↓:= NULL then ERR(↓no assignment to proc ident↓);↓);
750                             if ¬ b then PR string(↓Z↓)
751                             end else if i = 2 then PR string(↓:= Y↓)
752                         end; RESET reading ptrs;
753                         if contents of[a + 6] = 1 then
754                             PR string(↓; protect:= false↓)
755                         end end;
756     PR nlcr; PR string(↓end ↓); if block then RE end;
757     if ptr to array segment ≠ 0 then
758     begin PR nlcr; PR string(↓; ERASE(fnn) end) end
759 end Block exit;
760

```

761 comment:
762
763 6.3.4. Translation of a (formula or type) procedure declaration.
764
765 Syntactic definition 1:
766
767 <formula procedure declaration> ::= formula <procedure heading>
768 <procedure body>
769 <procedure declaration> ::= <procedure type><procedure heading>
770 <procedure body>
771 <procedure type> ::= <empty>|<type>
772 <procedure heading> ::= <procedure ident><formal parameter part>;
773 <value part><specification part>
774 <procedure ident> ::= <procedure identifier>|<operator identifier>
775 <operator identifier> ::= dyadic +|dyadic -|
776 dyadic ×|dyadic /|dyadic ↑|dyadic :|
777 monadic +|monadic -|monadic ↑|
778 constant +|constant ×
779
780 For <empty>,<type>,<procedure identifier>,<value part> and <formal
781 parameter part> see the ALGOL 60 report [10] section 5.4.
782
783 First part of algorithm:
784
785 procedure procedure declaration;
786 begin integer nr of param,a,b,su,nsu,i,proc ic,ptr to integer param,
787 ptr to formula param;
788 Boolean formula procedure;
789 Boolean procedure Identifier in paramlist(p); integer p;
790 if nr of param = 0 then Identifier in paramlist:= false else
791 begin integer a,i; i:= nr of param; a:= begin;
792 for a:= a + contents of [a] while
793 nr of ident ≠ contents of [a + 1] do
794 begin i:= i - 1; if i = 0 then
795 begin Identifier in paramlist:= false; goto OUT end
796 end; Identifier in paramlist:= true; p:= a;
797 if contents of [p + 3] : 32 ≠ 0 then ERR(
798 ≠equal identifiers in specification part);

```

799  OUT:
800  end;
801  formula procedure:= false; su:= synt unit; nsu:= next synt unit;
802  if nsu = procedure symbol then
803  begin if su = formula symbol then
804    begin PR string({integer }); RE; formula procedure:= true end
805    else PR and RE
806  end; PR and RE;
807  if first scan then
808  begin nr of param:= 0; proc ic:=
809    STORE into information list(st(st(st(st(st(st(0,
810      nr of ident),
811      ptr to first letgit),
812      (if nsu = procedure symbol then su else 0) × t15 + 32 ×
813      procedure symbol +
814      (if formula procedure then with local else 0)),
815      0),
816      nr of param),0));
817    if formula procedure then ptr to integer param:= proc ic else
818      ptr to integer param:= 0;
819      ptr to formula param:= 0;
820      begin:= contents of [preceding begin]:=
821      STORE into information list(st(st(st(st(st(st(st(st(0,
822        - begin symbol),
823        begin),
824        0),0),0),0),0),0),0));
825      preceding begin:= begin + 3; RE
826  end first scan else
827  begin begin:= contents of [preceding begin];
828    preceding begin:= begin + 3;
829    proc ic:= begin - 7;
830    nr of param:= contents of [proc ic + 5];
831    if operator identifier then
832    begin if ¬ formula procedure
833      then ERR({operator identifier at wrong place});
834      if in formula procedure body > 0
835      then ERR({oper def in formula proc body});
836    if (-nr of ident) : 1024 = dyadic symbol then

```

```

837     begin if nr of param ≠ 2 v contents of [proc ic + 20] : 32 ≠
838         formula symbol × 1024 v
839         contents of [proc ic + 26] : 32 ≠ formula symbol × 1024
840         then ERR(⊥operator identifier with wrong parameters⊥)
841     end else if (-nr of ident) = monadic symbol × 1024 +
842         power symbol then
843     begin if nr of param ≠ 2 v contents of [proc ic + 20] : 32 ≠
844         formula symbol × 1024 v
845         contents of [proc ic + 26] : 32 = formula symbol × 1024
846         then ERR(⊥operator identifier with wrong parameters⊥)
847     end else if (-nr of ident) : 1024 = monadic symbol then
848     begin if nr of param ≠ 1 v contents of [proc ic + 20] : 32 ≠
849         formula symbol × 1024
850         then ERR(⊥operator identifier with wrong parameters⊥)
851     end else if (-nr of ident) : 1024 = constant symbol then
852     begin if nr of param ≠ 1 v contents of [proc ic + 20] : 32 =
853         formula symbol × 1024 then ERR(
854         ⊥operator identifier with wrong parameters⊥)
855     end; PR operator; RE
856     end else begin PR string(⊥Z⊥); PR and RE end
857 end;
858
859     if synt unit ≠ open symbol then
860     begin PR and RE semicolon; goto PROCEDURE BODY end;
861 FORMAL PARAMETER PART:
862     PR and RE; if first scan then
863     begin nr of param := nr of param + 1;
864         STORE into information list(st(st(st(st(st(0,
865             nr of ident),
866             ptr to first letgit),
867             specified as name),
868             0),0));
869     end; PR string(⊥Z⊥); PR and RE;
870     if synt unit = comma symbol then goto FORMAL PARAMETER PART;
871     CHECK(close symbol); PR and RE; PR and RE semicolon;
872     if first scan then contents of [proc ic + 5] := nr of param;
873     if synt unit = value symbol then
874     begin VALUE PART: RE; if first scan then

```

```

875   begin if Identifier in param list(a) then
876       contents of [a + 3]:= specified as value
877       else ERR(↓value part not O.K.↓)
878   end; RE; if synt unit = comma symbol then goto VALUE PART;
879   RE semicolon;
880   if second scan then
881   begin Boolean value symbol is printed;
882       value symbol is printed:= false;
883       a:= begin; SAVE reading ptrs;
884       for i:= 1 step 1 until nr of param do
885       begin a:= a + contents of [a]; b:= contents of [a + 3];
886           if b : t15 † formula symbol ^ b - b : 32 × 32 =
887           specified as value then
888           begin if ¬ value symbol is printed then
889               PR string(↓ value †) else PR string(†,†);
890               value symbol is printed:= true;
891               SET reading ptrs on (contents of [a + 2]);
892               PR string(†Z†); PR synt unit
893           end end; RESET reading ptrs;
894           if value symbol is printed then PR string(†;†)
895   end end value part;
896
897   comment:
898   As can be seen above, all parameters are placed in the information
899   list and their possible occurrence in the value part has been taken
900   account of.
901   During the translation of the specification part, described below, the
902   types of these parameters are filled in. If it is necessary to inspect
903   the type of the i-th parameter of the procedure whose identifier
904   occupies the IC with entry "a", we have to look into:
905   "contents of [a + 17 + 6 × (i - 1) + 3]".
906
907   Syntactic definition 2:
908
909   <specification part>::= <empty>|<specification list>;
910   <specification list>::= <specification>|<specification list>;
911   <specification>
912   <specification>::= <ordinary specification>|

```

913 <formal procedure specification>
 914 <ordinary specification>::= <specifier><identifier list>|
 915 formula <identifier list>|
 916 formula array <identifier list>
 917 <formal procedure specification>::= <formal procedure specifier>
 918 <formal procedure segment list>
 919 <formal procedure specifier>::= procedure|<type> procedure|
 920 formula procedure
 921 <formal procedure segment list>::= <formal procedure segment>|
 922 <formal procedure segment list>,
 923 <formal procedure segment>
 924 <formal procedure segment>::= <identifier>
 925 <specification of parameters>
 926 <specification of parameters>::= <empty>|(<type of parameters list>)
 927 <type of parameters list>::= <type of parameter>|
 928 <type of parameters list>,<type of parameter>
 929 <type of parameter>::= <empty>|<specifier>|formula|formula value|
 930 formula array|formula procedure
 931 For specifier, type and identifier list see the ALGOL 60 report [10]
 932 section 5.4, from which a procedure as specifier should be swept out.
 933
 934 Example of a procedure heading:
 935 formula procedure P(a,b,c,d,e); value a; formula a,b;
 936 real c; procedure d; formula procedure e(formula value,formula,,);
 937 with this procedure heading a call P(x,y,3.14,PRINT,P)" is possible.
 938 The reason for introducing the specification of parameters for a
 939 formal procedure parameter is that the compiler has to know of each
 940 parameter of a procedure whether it is specified as formula and
 941 whether, in this case, it is called by value. A parameter which is not
 942 specified will be treated as to be not of type formula.
 943 Note that a specification of parameters concerns the parameters of
 944 the formal procedure with its identifier given in the formal procedure
 945 segment only. Hence, "procedure a,b(formula);" means that the formal
 946 procedure "a" does not have a parameter of type formula and that the
 947 formal procedure "b" has one parameter of type formula which is called
 948 by name.
 949
 950 Second part of algorithm::;

```

951
952 SPECIFICATION PART:
953   if synt unit = procedure symbol v
954   next synt unit = procedure symbol ^
955   (synt unit = real symbol v synt unit = integer symbol v
956   synt unit = Boolean symbol v synt unit = formula symbol)
957   then
958   begin integer su,nsu,nr of param; su:= synt unit;
959   nsu:= next synt unit;
960   if nsu = procedure symbol then
961   begin if su = formula symbol then
962     begin PR string(† integer †); RE end else PR and RE
963   end;
964 PROCEDURE SEGMENT: PR and RE;
965   if first scan then
966   begin nr of param:= 0;
967     if Identifier in paramlist(a) then contents of [a + 3]:=
968     (if nsu = procedure symbol then su else 0) × t15 +
969     32 × procedure symbol + specified as name
970     else ERR(†formal procedure param not in formal param part†)
971   end; PR string(†Z†); PR and RE;
972   if synt unit = open symbol then
973   begin if second scan then
974     begin RE; SKIP text until(synt unit = close symbol); RE end else
975     begin integer t; TYPE OF PARAM: RE;
976     if synt unit = comma symbol v synt unit = close symbol then
977       t:= 0 else
978       if next synt unit = value symbol then
979         begin t:= synt unit × t15 + specified as value; RE; RE end
980         else if next synt unit = array symbol v
981         next synt unit = procedure symbol then
982         begin t:= synt unit × t15 + 32 × next synt unit +
983         specified as name; RE; RE
984       end else
985       begin t:= synt unit × t15 + specified as name; RE end;
986       b:= STORE into information list(st(0, -t - 1));
987       nr of param:= nr of param + 1;
988       if nr of param = 1 then contents of[a + 4]:= b;

```



```

989      comment The type of the i - th parameter of a formal procedure
990      is equal to:
991      -1 - contents of[contents of[a + 4] - 1 + i × 2],
992      where "a" defines the IC of this procedure.;
993      if synt unit ≠ close symbol then goto TYPE OF PARAM;
994      contents of [a + 5]:= nr of param; RE
995      end end specification parameters of specified formal procedure;
996      if synt unit = comma symbol then goto PROCEDURE SEGMENT;
997      PR and RE semicolon
998      end procedure specification else
999
1000     if synt unit = array symbol v next synt unit = array symbol then
1001     begin integer a,su,nsu;
1002     su:= synt unit; nsu:= next synt unit;
1003     if nsu = array symbol then
1004     begin if su = formula symbol then
1005     begin PR string(↓ integer ↓); RE end else PR and RE
1006     end;
1007     ARRAY IDENTIFIER: PR and RE;
1008     if first scan then
1009     begin if Identifier in paramlist(a) then
1010     contents of [a + 3]:= contents of [a + 3] +
1011     (if nsu = array symbol then su else real symbol) × t15 +
1012     32 × array symbol
1013     else ERR(↓array param not in formal param list↓)
1014     end; PR string(↓Z↓); PR and RE;
1015     if synt unit = comma symbol then goto ARRAY IDENTIFIER;
1016     PR and RE semicolon
1017     end array specification else
1018
1019     if is specifier(synt unit) then
1020     begin integer a,b,su; su:= synt unit;
1021     if su = formula symbol then
1022     begin PR string(↓ integer ↓); RE end else PR and RE;
1023     OTHER IDENTIFIER:
1024     if first scan then
1025     begin if Identifier in paramlist(a) then
1026     begin b:= contents of [a + 3]:= su × t15 + contents of [a + 3];

```

6-28

```
1027     if b = formula symbol × t15 + specified as value then
1028     begin contents of [a + 4]:= ptr to formula param;
1029     ptr to formula param:= a
1030     end end
1031     else ERR(↓specified param not in form param list↓)
1032     end; PR string(↓Z↓); PR and RE;
1033     if synt unit = comma symbol then
1034     begin PR and RE; goto OTHER IDENTIFIER end;
1035     PR and RE semicolon
1036     end other specification;
1037
1038     if is specifier(synt unit) then goto SPECIFICATION PART;
1039
1040     if first scan then
1041     begin i:= nr of param; a:= begin; su:= 0;
1042     for a:= a + contents of [a] while i > 0 do
1043     begin i:= i - 1; b:= contents of [a + 3] : 32;
1044     if b = 0 v b = label symbol × 1024 v
1045     b = switch symbol × 1024 then su:= 1
1046     end end;
1047
1048     comment:
1049     Third part of algorithm::;
1050
1051     PROCEDURE BODY: in formula procedure body:=
1052     in formula procedure body + 1;
1053     if first scan then
1054     begin contents of [begin + 4]:= ptr to integer param;
1055     contents of [begin + 5]:= ptr to formula param
1056     end;
1057     envelope of block(begin);
1058     if first scan ∧ su = 1 then
1059     begin if EVEN(contents of [proc ic + 16] : 8) = -1
1060     then dangerous procedures:= true
1061     end; in formula procedure body:= in formula procedure body - 1;
1062     PR and RE semicolon
1063     end procedure declaration;
1064
```

```

1065 comment:
1066
1067 6.4. Translation of a statement
1068
1069 Syntactic definition:
1070
1071 <statement> ::= <optional label sequence><unlabelled statement>
1072 <optional label sequence> ::= <empty>|<label sequence>
1073 <label sequence> ::= <label>:|<label sequence><label>:
1074 <label> ::= <identifier>|<unsigned integer>
1075 <unlabelled statement> ::= <conditional statement>|
1076                               <unconditional statement>|<for statement>
1077
1078 Algorithms::
1079
1080 procedure optional label sequence(divisible,begin printed);
1081 value divisible; Boolean divisible,begin printed;
1082 begin Boolean label; label:= begin printed:= false;
1083 A: if (synt unit = identifier v synt unit = integral number) ^
1084     next synt unit = colon symbol then
1085     begin if synt unit = identifier then PR string(†Z†);
1086         if operator identifier then
1087         begin ERR(†operator identifier occurs as label†);
1088             PR operator; RE
1089         end else PR and RE;
1090             PR and RE; block contains labels:= label:= true;
1091         goto A
1092     end;
1093     if second scan then
1094     begin if label ^ (dangerous inner block v dangerous procedures) then
1095         begin if ¬ divisible then
1096             begin PR string(†begin †); begin printed:= true end;
1097             PR string(† ERASE(snn);†)
1098         end end end optional label sequence;
1099
1100 procedure statement(divisible); value divisible; Boolean divisible;
1101 begin Boolean beg pr; optional label sequence(divisible,beg pr);
1102     if synt unit = if symbol then

```

```

1103 begin PR and RE; conditional statement end else
1104 if synt unit = for symbol then
1105 begin PR and RE; for statement end else
1106 unconditional statement(beg pr v divisible);
1107 if beg pr then PR string(end †)
1108 end statement;
1109
1110 comment:
1111
1112 6.4.1. Translation of a conditional statement
1113
1114 Syntactic definition:
1115
1116 <conditional statement> ::= <if statement> |
1117     <if statement> else <statement> |
1118     <if clause> <optional label sequence> <for statement>
1119 <if statement> ::= <if clause> <optional label sequence>
1120     <unconditional statement>
1121 <if clause> ::= if <Boolean expression> then
1122
1123 Algorithm:;
1124
1125 procedure conditional statement;
1126 begin Boolean bp,pias; other expression; CHECK(then symbol);
1127 PR and RE;
1128 optional label sequence(false,bp);
1129 if synt unit = for symbol then
1130 begin PR and RE; for statement; if bp then PR string(end †)
1131 end else
1132 begin unconditional statement(bp); if bp then PR string(end †);
1133 pias := proc id ass stat; proc id ass stat := false;
1134 if synt unit = else symbol then
1135 begin PR and RE; statement(false) end;
1136 proc id ass stat := pias ^ proc id ass stat
1137 end end conditional statement;
1138
1139 comment:
1140

```

1141 6.4.2. Translation of a for statement

1142

1143 Syntactic definition:

1144 See the ALGOL 60 report [10] section 4.6. Note that as
 1145 for-list-variable one may not have a formula variable, but one may
 1146 have an arithmetic variable only.

1147

1148 Algorithm:;

1149

1150 procedure for statement;1151 begin integer a; if second scan then1152 begin if Search for identifier(a) then1153 begin a:= contents of[a + 3] ; t15;1154 if a = formula symbol then

1155 ERR(↓for list element of type formula↓)

1156 end end; other expression; CHECK(becomes symbol); PR and RE;

1157 for list element: other expression;

1158 if synt unit = step symbol then1159 begin PR and RE; other expression; CHECK(until symbol);

1160 PR and RE; other expression

1161 end else1162 if synt unit = while symbol then1163 begin PR and RE; other expression end;1164 if synt unit = comma symbol then1165 begin PR and RE; goto for list element end;1166 CHECK(do symbol); PR and RE; statement(false);1167 proc id ass stat:= false1168 end for statement;

1169

1170 comment:

1171

1172 6.4.3. Translation of an unconditional statement

1173

1174 Syntactic definition:

1175

1176 <unconditional statement> ::= <compound statement>|<block>|

1177 <goto statement>|<assignment statement>|<procedure statement>|

1178 <dummy statement>

```

1179 <compound statement> ::= begin <compound tail>
1180 <compound tail> ::= <statement> end | <statement>; <compound tail>
1181 <goto statement> ::= goto <designational expression>
1182 <dummy statement> ::= <empty>
1183 For designational expression see the ALGOL 60 report [10] section 3.5.
1184
1185 Algorithm::;
1186
1187 procedure unconditional statement(divisible);
1188 value divisible; Boolean divisible;
1189 if synt unit = begin symbol then
1190 begin if is declarator(next synt unit) then envelope of block(0) else
1191   begin PR and RE begin; S: statement(true);
1192     if synt unit ≠ semicolon symbol ^ synt unit ≠ end symbol then
1193       begin ERR(↓statement not appropriately closed↓);
1194         SEEK(synt unit = semicolon symbol v
1195             synt unit = end symbol)
1196     end;
1197   L: if synt unit = semicolon symbol then
1198     begin PR and RE semicolon; goto L end;
1199     if synt unit ≠ end symbol then
1200       begin proc id ass stat := false; goto S end;
1201     PR and RE end
1202   end end else
1203   if synt unit = goto symbol then
1204     begin block contains gotos := true; PR and RE; other expression
1205   end else
1206   if synt unit = identifier ^ (next synt unit = becomes symbol v
1207     next synt unit = sub symbol) then
1208     assignment statement(divisible) else
1209     if synt unit = identifier then procedure statement(false) else
1210     begin comment dummy statement;; end unconditional statement;
1211
1212   comment:
1213
1214   6.4.4. Translation of an assignment statement
1215
1216   Syntactic definition:

```

1217
 1218 <assignment statement>::= <ordinary assignment statement>|
 1219 <formula assignment statement>
 1220 <ordinary assignment statement>::= <left part list>
 1221 <Bool or arith expression>
 1222 <formula assignment statement>::= <left part list><formula expression>
 1223 <left part list>::= <left part>|<left part list><left part>
 1224 <left part>::= <variable>:=|<procedure identifier>:=
 1225 <Bool or arith expression>::= <Boolean expression>|
 1226 <arithmetic expression>
 1227
 1228 The left part list in a formula assignment statement may, of course,
 1229 contain variables or procedure identifiers of type formula only.
 1230
 1231 Let fp1,fp2 and fp3 be formula procedure identifiers and f1,f2,f3 and
 1232 f4 formula variables, then the translations of:
 1233 f1:= f2:= fp1:= fp2:= f3:= fp3:= f4:= expr
 1234 is as follows:
 1235 FP1:= FP2:= FP3:= ASSIGN(F1,ASSIGN(F2,ASSIGN(F3,ASSIGN(F4,EXPR))))
 1236 possibly followed by "; protect:= true".
 1237 With the protect statement, the whole is, dependent on the value of
 1238 "divisible", surrounded by begin and end.
 1239
 1240 Note that as identifiers we may now encounter the special operator
 1241 identifiers: dyadic + etc.
 1242
 1243 Algorithm:;
 1244
 1245 procedure assignment statement(divisible);
 1246 value divisible; Boolean divisible;
 1247 if first scan then
 1248 begin L: if interested in proc id $\hat{=} \neg$ proc id ass stat then
 1249 begin if next synt unit = becomes symbol then
 1250 proc id ass stat:= nr of ident = contents of [procedure body-6]
 1251 end; operator identifier:= false;
 1252 RE; if synt unit = sub symbol then other expression;
 1253 if synt unit = becomes symbol then
 1254 begin RE; if synt unit = identifier $\hat{=}$

```

1255     (next synt unit = becomes symbol v next synt unit = sub symbol)
1256     then goto L;
1257     other expression
1258 end end first scan else
1259 begin integer a,t,p,m,n,i,b,
1260     ptr to first formula procedure identifier,prec ptr;
1261     Boolean form ass stat,first,protection needed;
1262     protection needed:= false; first:= true;
1263     ptr to first formula procedure identifier:= 0; n:= 0;
1264     L1: n:= n + 1; block depth:= 0;
1265     if Search for identifier(a) then
1266     begin m:= contents of[a + 3]; t:= m : t15;
1267         m:= m - t × t15; p:= m : 32; m:= m - p × 32
1268     end else t:= p:= m:= 0;
1269
1270     if first then
1271     begin form ass stat:= t = formula symbol; first:= false;
1272         if form ass stat then
1273         begin SAVE reading ptrs; min block depth:= 0 end
1274     end else if ¬ (form ass stat = t = formula symbol) then
1275         ERR(⟨left part list of assignment statement not O.K.⟩);
1276     if form ass stat then
1277     begin if p = procedure symbol then
1278         begin protection needed:= protection needed v m = with local;
1279         if m = specified as name then ERR(
1280             ⟨assignment to parameter procedure⟩);
1281         comment In behalf of the procedure "formula expression" we
1282             introduce the following two statements. The block depth is the
1283             block depth of the procedure declaration.;
1284         proc id ass stat:= true;
1285         if min block depth > block depth then
1286             min block depth:= block depth; if block depth = 0 then ERR(
1287                 ⟨assignment to proc ident outside body⟩);
1288         p:= begin; if in formula procedure body > 1 then
1289             begin if m = with local ^ block depth < -1 then
1290                 begin l11: b:= p - 7; t:= contents of [b + 3];
1291                 if contents of [b] = 7 ^ t = formula symbol × t15
1292                 + procedure symbol × 32 + with local then

```



```

1293         contents of [b + 6]:= contents of [b + 6] : 4 × 4 + 2;
1294         block depth:= block depth + 1;
1295         if block depth < -1 then
1296             begin p:= contents of [contents of [p + 2] + 2]; goto L11
1297         end end end;
1298         if ptr to first formula procedure identifier = 0 then
1299             ptr to first formula procedure identifier:= (a × 1024 + n) × 4
1300         else
1301             contents of [prec ptr]:=
1302             contents of [prec ptr] + (a × 1024 + n) × 4;
1303             prec ptr:= a + 6;
1304             if contents of [prec ptr] : 4 ≠ 0 then
1305                 begin contents of [prec ptr]:=
1306                     remainder(contents of [prec ptr],4);
1307                     ERR(
1308                         †two equal form proc idents in lhs of ass stat†)
1309                 end;
1310
1311         comment
1312             The fourth element of the information cell of the formula
1313 procedure identifier is used as link for a list of integers, and can
1314 not be used as a temporary link in the list of left parts now being
1315 formed. In a situation like:
1316         "formula procedure fp1 ;
1317         begin formula procedure fp2 ; fp1:= fp2:= f ;
1318         integer i ; i:= lhs of fp2
1319         end", we see clearly the danger.
1320         Therefore, the sixth element is chosen, with the obligation to
1321 save the two-bits information;
1322         end else if m = declared as value v
1323         m = specified as value then ERR(†assignment to value variable†);
1324
1325         RE; if synt unit = sub symbol then
1326         begin RE; SKIP text until(synt unit = bus symbol); RE end;
1327         CHECK(becomes symbol); RE;
1328         if synt unit = identifier then
1329         begin if next synt unit = becomes symbol then goto L1;
1330         if next synt unit = sub symbol then

```

```

1331     begin SAVE reading ptrs; RE; RE;
1332     SKIP text until(synt unit = bus symbol);
1333     if next synt unit = becomes symbol then
1334     begin RESET reading ptrs; goto L1 end
1335     else RESET reading ptrs
1336     end end;
1337     RESET reading ptrs;
1338     if ptr to first formula procedure identifier  $\neq$  0 then
1339     begin if protection needed  $\wedge$   $\neg$  divisible then
1340     PR string( $\downarrow$ begin  $\downarrow$ ); SAVE reading ptrs;
1341     a:= ptr to first formula procedure identifier  $\div$  4;
1342     L2: m:= a  $\div$  1024;
1343     SET reading ptrs on (contents of [m + 2]);
1344     a:= contents of [m + 3]; a:= a - a  $\div$  32  $\times$  32;
1345     if a = with local then PR string( $\downarrow$ Y $\downarrow$ );
1346     if operator identifier then PR operator else
1347     begin if a = without local then PR string( $\downarrow$ Z $\downarrow$ );
1348     PR synt unit
1349     end;
1350     PR string( $\downarrow$ :=  $\downarrow$ ); a:= contents of [m + 6]  $\div$  4;
1351     if a  $\neq$  0 then goto L2; RESET reading ptrs
1352     end;
1353     a:= ptr to first formula procedure identifier  $\div$  4; p:= 0; i:= 1;
1354     L3: if a  $\neq$  0 then
1355     begin m:= a  $\div$  1024; t:= a - m  $\times$  1024 -1;
1356     b:= contents of [m + 6]; a:= b  $\div$  4;
1357     contents of [m + 6]:= b - a  $\times$  4
1358     end else t:= n;
1359     for i:= i step 1 until t do
1360     begin PR string( $\downarrow$ ASSIGN(Z $\downarrow$ ); p:= p + 1; PR and RE;
1361     if synt unit = sub symbol then other expression;
1362     CHECK(becomes symbol); RE; PR string( $\downarrow$ , $\downarrow$ )
1363     end;
1364     if t < n then
1365     begin comment Skip the procedure identifier assignment; RE; RE;
1366     i:= t + 2; goto L3
1367     end;
1368     formula expression;

```



```

1407 <parameter delimiter>::=,|<letter string>:(
1408
1409 Algorithm::;
1410
1411 procedure procedure statement(IRN necessary);
1412 value IRN necessary; Boolean IRN necessary;
1413 if first scan then
1414 begin if next synt unit ≠ open symbol then RE else
1415   begin RE; RE; SKIP text until(synt unit = close symbol); RE
1416 end end else
1417 begin integer a,i,type,p,m,first param,nr of param,mode of proc;
1418   Boolean piass; i:= 0; block depth:= 0;
1419   if Search for identifier(a) then
1420     begin i:= contents of [a + 3];
1421     if i - i : 32 × 32 = specified as name then
1422       begin mode of proc:= 1; first param:= contents of [a + 4] - 1;
1423         nr of param:= contents of [a + 5]
1424       end else
1425       begin mode of proc:= 2; first param:= a + 14;
1426         nr of param:= contents of [a + 5]
1427       end end else
1428     begin if Search for standard identifier(nr of param) then
1429       mode of proc:= 3 else
1430       begin ERR(↓proc not declared↓);
1431       nr of param:= mode of proc:= 0
1432     end end;
1433
1434   if IRN necessary then
1435     begin i:= i : t15; if i = real symbol v (mode of proc = 3 ^
1436       type of standard identifier = real symbol) then PR string(↓RN(↓)
1437     else if i = integer symbol v (mode of proc = 3 ^
1438       type of standard identifier = integer symbol) then
1439     PR string(↓IN(↓) else if i = formula symbol then
1440     begin if proc id ass stat then
1441       begin if block depth > min block depth v mode of proc = 1 then
1442         PR string(↓abs(↓) else IRN necessary:= false
1443       end else IRN necessary:= false
1444     end else IRN necessary:= false

```

```

1445   end; piass:= proc id ass stat; proc id ass stat:= false;
1446
1447   if mode of proc  $\neq$  3 then PR string( $\downarrow Z\downarrow$ ); PR and RE;
1448   i:= 0; if synt unit  $\neq$  open symbol then goto END;
1449 L: PR and RE; i:= i + 1; m:= 0;
1450   if mode of proc = 1 then
1451   begin if i  $\leq$  nr of param then
1452     m:= -1 - contents of [first param + i  $\times$  2]
1453   end else
1454   if i > nr of param  $\wedge$  mode of proc  $\neq$  0 then
1455     ERR( $\downarrow$ too much param in proc call $\downarrow$ ) else
1456     if mode of proc = 2 then m:= contents of [first param + i  $\times$  6];
1457     type:= m : t15; m:= m - type  $\times$  t15; p:= m : 32;
1458     m:= m - p  $\times$  32;
1459     if p  $\neq$  0 then
1460     begin if synt unit = identifier then
1461       begin if next synt unit = comma symbol  $\vee$  next synt unit =
1462         close symbol then
1463         begin if type  $\neq$  formula symbol then goto L1 else
1464           if Search for identifier(a) then
1465             begin if contents of [a + 3] : 32 = type  $\times$  1024 + p then
1466               goto L1
1467             end end end; ERR( $\downarrow$ actual param not O.K. $\downarrow$ );
1468 L1: PR string( $\downarrow Z\downarrow$ ); PR and RE
1469     end else if type = formula symbol then
1470     begin if m = specified as name then
1471       begin if synt unit = identifier  $\wedge$ 
1472         (next synt unit = comma symbol
1473          $\vee$  next synt unit = close symbol) then
1474         begin if Search for identifier(a) then
1475           begin m:= contents of [a + 3];
1476           if m : 32 = type  $\times$  1024 then
1477             begin if m - m : 32  $\times$  32 = specified as value
1478               then PR string( $\downarrow Y\downarrow$ ) else PR string( $\downarrow Z\downarrow$ );
1479               PR and RE; goto OUT
1480             end end
1481           end else if synt unit = identifier  $\wedge$  next synt unit =
1482             sub symbol then

```

```

1483   begin SAVE reading ptrs; RE; RE;
1484       SKIP text until(synt unit = bus symbol);
1485       if next synt unit = comma symbol v
1486         next synt unit = close symbol then
1487         begin RESET reading ptrs;
1488           if Search for identifier(a) then
1489             begin m:= contents of [a + 3];
1490               if m : 32 = type × 1024 + array symbol then
1491                 begin PR string(↓Z↓); PR and RE;
1492                 other expression; goto OUT
1493             end end end else RESET reading ptrs
1494         end end;
1495         formula expression
1496     end else other expression;
1497 OUT: if synt unit = comma symbol then goto L;
1498     CHECK(close symbol); PR and RE;
1499 END: if mode of proc ≠ 1 then
1500     begin if i ≠ nr of param ^ mode of proc ≠ 0 then
1501         ERR(↓nr of param not O.K.↓)
1502     end; proc id ass stat:= piass; if IRN necessary then
1503         PR string(↓)↓)
1504 end procedure statement;
1505
1506 comment:
1507
1508 6.5. Translation of an expression
1509
1510 Syntactic definition:
1511
1512 <expression> ::= <simple expression> |
1513     if <Boolean expression> then <simple expression>
1514     else <expression>
1515 <simple expression> ::= <simple formula expression> |
1516     <simple other expression>
1517 <simple other expression> ::= <simple Boolean expression> |
1518     <simple arithmetic expression> |
1519     <simple designational expression>
1520 Boolean- , arithmetic- , designational- and formula expressions are

```

1521 expressions in which the simple expressions are chosen of the
 1522 appropriate type.
 1523
 1524 Algorithm:;
 1525
 1526 procedure other expression;
 1527 if synt unit = if symbol then
 1528 begin PR and RE; other expression; CHECK(then symbol); PR and RE;
 1529 simple other expression; CHECK(else symbol); PR and RE;
 1530 other expression
 1531 end else simple other expression;
 1532
 1533 procedure formula expression;
 1534 begin Boolean piass; piass:= proc id ass stat;
 1535 if synt unit = if symbol then
 1536 begin PR and RE; proc id ass stat:= false;
 1537 other expression; CHECK(then symbol); PR and RE;
 1538 proc id ass stat:= piass; simple formula expression(false);
 1539 CHECK(else symbol); PR and RE; proc id ass stat:= piass;
 1540 formula expression
 1541 end else simple formula expression(false)
 1542 end formula expression;
 1543
 1544 comment:
 1545 A simple other expression is syntactically defined in the ALGOL 60
 1546 report [10] chapter 3, with the following addition:
 1547 as primary in a simple arithmetic expression a formula enquiry, a
 1548 length enquiry and a type enquiry are allowed. These enquiries are
 1549 syntactically defined as follows:
 1550 <formula enquiry> ::= <kind of enquiry> of <formula name>
 1551 <type enquiry> ::= type of <formula name>
 1552 <length enquiry> ::= length of <formula name>
 1553 <kind of enquiry> ::= lhs|rhs|el <arithmetic expression>
 1554 <formula name> ::= <formula primary>
 1555 Moreover, as primaries in a boolean expression, the following
 1556 constructions are allowed:
 1557 constant <formula name>,
 1558 monadic <formula name>,

```

1559 dyadic <formula name>,
1560 polyadic <formula name>,
1561 rowadic <formula name>.
1562 These are called type category enquiries.
1563
1564 Algorithm::
1565
1566 procedure enquiry;
1567 begin integer s; s:= synt unit; RE;
1568   if s = el symbol then
1569     begin PR string( $\downarrow$ EL( $\downarrow$ ); other expression; PR string( $\downarrow$ , $\downarrow$ ) end else
1570     if s = lhs symbol then PR string( $\downarrow$ LHS( $\downarrow$ ) else
1571     if s = rhs symbol then PR string( $\downarrow$ RHS( $\downarrow$ ) else
1572     if s = type symbol then PR string( $\downarrow$ TYPE( $\downarrow$ ) else
1573     if s = length symbol then PR string( $\downarrow$ LENGTH( $\downarrow$ ) else
1574     ERR( $\downarrow$ error in enquiry $\downarrow$ ); CHECK(of symbol); RE;
1575     if first scan then other expression else
1576     simple formula expression(true);
1577     PR string( $\downarrow$ ) $\downarrow$ )
1578 end enquiry;
1579
1580 comment:
1581
1582 6.5.1. Translation of a simple other expression
1583
1584 Algorithm::
1585
1586 procedure simple other expression;
1587 begin: if synt unit = identifier then
1588 begin if next synt unit = open symbol then procedure statement(false)
1589   else
1590     begin integer a,t,p,m,br; br:= 0;
1591     if second scan then
1592       begin if Search for identifier(a) then
1593         begin m:= contents of [a + 3]; t:= m  $\div$  t15;
1594         m:= m - t  $\times$  t15; p:= m  $\div$  32; m:= m - p  $\times$  32;
1595       end else if Search for standard identifier(a) then
1596       begin t:= 0; p:= procedure symbol; m:= standard identifier

```



```

1597     end else t:= p:= m:= 0;
1598     if t = formula symbol then
1599     begin if p = procedure symbol v (p = 0 ^ (m = declared as value
1600     v m = specified as value v m = specified as name)) then
1601     begin PR string(↓abs(↓)); br:= 1 end;
1602     if p = array symbol v p ≠ procedure symbol ^
1603     m = declared as name then
1604     begin PR string(↓V(↓)); br:= br + 1 end else
1605     if m = specified as name ^ p ≠ procedure symbol then
1606     begin PR string(↓VN(↓)); br:= br + 1 end
1607     end; if t = formula symbol ^ p = 0 ^ m = specified as value
1608     then PR string(↓Y↓)
1609     else if m ≠ standard identifier then PR string(↓Z↓)
1610     end else a:= t:= p:= m:= 0; PR and RE;
1611
1612     if synt unit = sub symbol then
1613     begin if second scan then
1614     begin if t = formula symbol then
1615     begin if p ≠ array symbol then ERR(
1616     ↓formula array not declared↓)
1617     end end;
1618     L: PR and RE;
1619     other expression; if synt unit = comma symbol then goto L;
1620     CHECK(bus symbol); PR and RE
1621     end; for br:= br - 1 while br ≥ 0 do PR string(↓)↓)
1622     end variable; goto begin
1623     end else
1624     if is enquiry then
1625     begin if synt unit = lhs symbol v
1626     synt unit = rhs symbol v synt unit = el symbol
1627     then PR string(↓AR↓); enquiry; goto begin
1628     end else
1629     if is adic symbol(synt unit) then
1630     begin if second scan then
1631     begin PR int num(synt unit - constant symbol);
1632     PR string(↓= TYPE CAT(↓)); RE; simple formula expression(true);
1633     PR string(↓)↓)
1634     end else RE; goto begin

```

6-44

```
1635 end else
1636 if synt unit = open symbol v synt unit = sub symbol then
1637 begin Boolean open; open:= synt unit = open symbol;
1638 L: PR and RE; other expression;
1639 if synt unit = comma symbol then goto L;
1640 if open then CHECK(close symbol) else CHECK(bus symbol);
1641 PR and RE; goto begin
1642 end else
1643 begin integer a; for a:= semicolon symbol,end symbol,else symbol,
1644 then symbol,comma symbol,close symbol,bus symbol,step symbol,
1645 until symbol,do symbol,while symbol,colon symbol,becomes symbol,
1646 of symbol do if synt unit = a then goto END;
1647 PR and RE; goto begin;
1648 END:
1649 end simple other expression;
1650
1651 procedure translate string;
1652 begin integer m,n,i,p,T; m:= n:= T:= 0;
1653 p:= ptr of text2 - 3; i:= ptr of text1 - 1; PR string(↑STRING(↑));
1654 L: i:= i + 1; if take from text array(i) ≠ smaller than symbol then
1655 goto L;
1656 for i:= i + 1 step 1 until p do
1657 begin n:= n + 1; T:= T × 256 + take from text array(i) + 1;
1658 if n = 3 v i = p then
1659 begin PR string(↑st(↑)); PR int num(T);
1660 PR string(↑,↑); T:= n:= 0; m:= m + 1
1661 end end;
1662 PR string(↑0↑); for i:= 1 step 1 until m do PR string(↑)↑);
1663 PR string(↑,↑); PR int num(m); PR string(↑)↑)
1664 end translate string;
1665
1666 comment:
1667 The translation of "↑01234567↑" is:
1668 "↑STRING(st(66051,st(263430,st(1800,0))),3)↑",
1669 where
1670 66051 = ((0+1)×256+(1+1))×256+(2+1),
1671 263430 = ((3+1)×256+(4+1))×256+(5+1),
1672 1800 = (6+1)×256+(7+1).
```

1673

1674 6.5.2. Translation of a simple formula expression

1675

1676 Syntactic definition:

1677

1678 A simple formula expression resembles highly a simple arithmetic
 1679 expression. As formula primaries we may have all the arithmetic
 1680 primaries, a string, and a formula base (see below). As in an
 1681 arithmetic expression, the formula-, type- and length enquiries are
 1682 also possible.

1683 As the syntactic definition of a formula expression is, apart for the
 1684 primary, identical to the definition of an arithmetic expression, we
 1685 define a formula primary only:

1686

```
1687 <formula primary> ::= <unsigned number> |
1688     <variable> | <function designator> | (<formula expression>) |
1689     <formula variable> | <formula enquiry> | <type enquiry> |
1690     <length enquiry> | <string> | <formula base>
```

1691

```
1692 <formula base> ::= constant(<type>, <int arith expr>, <int arith expr>) |
1693     monadic(<type>, <formula expression>, <int arith expr>) |
1694     dyadic(<type>, <formula expression>, <formula expression>) |
1695     polyadic(<type>, <int id>, <length>, <formula expression>) |
1696     rowadic(<type>, <int id>, <length>, <int arith expr>)
```

```
1697 <type> ::= <int arith expr>
```

```
1698 <int id> ::= <identifier>
```

```
1699 <length> ::= <int arith expr>
```

1700

1701 int arith expr is an arithmetic expression delivering an integer.

1702

1703 It is necessary that

- 1704 1. The value v of type satisfies: $0 \leq v \leq 30$.
- 1705 2. The value v of the right-hand side of the int arith expr in a
 1706 constant formula satisfies: $\text{abs}(v) < 2 \uparrow 17 - 1$.
- 1707 3. The value v of length satisfies: $v \geq 0$.

1708

1709 Algorithm;

1710

```

1711 procedure simple formula expression(primary only);
1712 value primary only; Boolean primary only;
1713 begin integer iF,primary symbol,int num,f,nr of brackets;
1714   integer array Fl,Ft,Fr[1:100]; Boolean scan1;
1715
1716 comment:
1717 In a first pass (scan1 = true), the expression is scanned by means
1718 of the procedures "elevator" and "primary" during which a syntactical
1719 tree is formed and stored into the arrays "Fl", "Ft" and "Fr",which
1720 have a pointer "iF".
1721 In a second pass (scan1 = false), output is produced by a call of "PR
1722 form expr", which, by using the procedure "primary" forms the output
1723 climbing through the syntactical tree.
1724 If the expression is the right-hand side of an assignment to a
1725 procedure identifier, in which case "proc id ass stat" has the value
1726 true, a special translation, involving the abs function in front
1727 of some formula variables and some formula function designators, is
1728 performed. As soon as this variable or function designator is shielded
1729 by another procedure, the "abs" is suppressed as a result of the
1730 statement: "proc id ass stat:= false" in "PR form expr".
1731 Concerning the treatment of brackets we remark that "PR form
1732 expr" skips leading opening brackets and trailing closing brackets.
1733 Therefore, "primary" starts reading, during the second pass, a
1734 non-opening bracket symbol, and "RE" in "PR form expr", reads an
1735 operator symbol (except when it skips brackets).
1736 Special attention is given to the primary: "(if ... then ... else
1737 ...)". The translation is a translation from infix notation to prefix
1738 notation.;
1739
1740   integer procedure elevator(floor); value floor; integer floor;
1741   if floor = 0 then elevator:= primary else
1742   begin integer e1,s;
1743     if floor = 3 then
1744       begin s:= synt unit; if s = plus symbol v s = minus symbol then
1745         begin RE; e1:= elevator(2); iF:= iF + 1;
1746         if iF > 100 then FATAL ERR(↓formula expression too long↓);
1747         Fl[iF]:= e1; Ft[iF]:= s + 1024; e1:= iF;
1748         goto again

```

```

1749     end end; el:= elevator(floor - 1);
1750  again: s:= synt unit;
1751     if floor = 3 ^ (s = plus symbol v s = minus symbol) v
1752         floor = 2 ^ (s = times symbol v s = over symbol v
1753             s = int div symbol) v
1754         floor = 1 ^ (s = power symbol) then
1755     begin RE; iF:= iF + 1; if iF > 100 then FATAL ERR(
1756         †formula expression too long†);
1757         Fl[iF]:= el; el:= iF; Ft[el]:= s;
1758         Fr[el]:= elevator(floor - 1); goto again
1759     end; elevator:= el
1760 end elevator;
1761
1762 procedure PR form expr(f); value f; integer f;
1763 begin procedure skip opens;
1764     for synt unit:= synt unit
1765     while synt unit = open symbol do
1766     begin nr of brackets:= nr of brackets + 1; RE end;
1767     skip opens;
1768     if f = primary symbol v f = int num then primary else
1769     begin integer l,t,r; l:= Fl[f]; t:= Ft[f]; r:= Fr[f];
1770     proc id ass stat:= false;
1771     if t > 1024 then
1772     begin t:= t - 1024;
1773         if t = plus symbol then PR string(†PL(†)
1774         else PR string(†MI(†);
1775         RE; PR form expr(l); PR string(†)†)
1776     end else
1777     if t = power symbol ^ r = int num then
1778     begin PR string(†IE(†); PR form expr(l);
1779     PR string(†,†); RE; skip opens; PR and RE; PR string(†)†)
1780     end else
1781     begin if t = plus symbol then PR string(†S(†) else
1782         if t = minus symbol then PR string(†D(†) else
1783         if t = times symbol then PR string(†P(†) else
1784         if t = over symbol then PR string(†Q(†) else
1785         if t = int div symbol then PR string(†IQ(†) else
1786         if t = power symbol then PR string(†E(†);

```

```

1787     PR form expr(1); PR string(†,†); RE; PR form expr(r);
1788     PR string(†)†)
1789     end end; for synt unit:= synt unit
1790     while synt unit = close symbol ^ nr of brackets > 0 do
1791     begin nr of brackets:= nr of brackets - 1; RE end
1792 end PR form expr;
1793
1794 integer procedure primary;
1795 begin primary:= primary symbol;
1796     if is adic symbol(synt unit) then
1797     begin if scan1 then
1798     begin RE; RE; SKIP text until(synt unit = close symbol); RE
1799     end else
1800     begin integer s; s:= synt unit; RE;
1801     CHECK(open symbol); RE; proc id ass stat:= false;
1802     if s = polyadic symbol v s = rowadic symbol then
1803     begin PR string(†STORE ROW(†);
1804     if s = polyadic symbol then PR string(†96,†)
1805     else PR string(†128,†); other expression;
1806     CHECK(comma symbol); PR and RE; CHECK(identifier);
1807     PR string(†Z†); PR and RE; CHECK(comma symbol);
1808     PR and RE; other expression; CHECK(comma symbol);
1809     PR and RE;
1810     if s = polyadic symbol then formula expression else
1811     other expression; CHECK(close symbol); PR and RE
1812     end else
1813     begin PR string(†STORE †);
1814     if s = constant symbol then PR string(†CONST(†) else
1815     if s = monadic symbol then PR string(†MONADIC(†) else
1816     if s = dyadic symbol then PR string(†DYADIC(†);
1817     other expression;
1818     CHECK(comma symbol); PR and RE;
1819     if s = constant symbol v s = monadic symbol then
1820     other expression else formula expression;
1821     CHECK(comma symbol); PR and RE;
1822     if s = constant symbol then other expression
1823     else formula expression; CHECK(close symbol); PR and RE
1824     end end end else

```

```

1825
1826 if synt unit = integral number v synt unit = real number then
1827 begin if ¬ scan1 then
1828     begin if synt unit = integral number
1829         then PR string(↓IN(↓) else PR string(↓RN(↓);
1830         PR and RE; PR string(↓)↓)
1831     end else
1832     begin if synt unit = integral number
1833         then primary:= int num; RE
1834 end end else
1835
1836 if synt unit = identifier then
1837 begin if scan1 then
1838     begin if next synt unit = open symbol v next synt unit =
1839     sub symbol then
1840         begin RE; RE; SKIP text until(
1841         synt unit = close symbol v synt unit = bus symbol)
1842     end; RE
1843 end else
1844 if next synt unit = open symbol then procedure statement(true)
1845 else
1846 begin integer a,t,p,m,br; br:= 1; block depth:= 0;
1847     if Search for identifier(a) then
1848         begin m:= contents of [a + 3]; t:= m : t15; m:= m - t × t15;
1849         p:= m : 32; m:= m - p × 32;
1850     end else
1851     if Search for standard identifier(a) then
1852         begin t:= real symbol; p:= procedure symbol;
1853         m:= standard identifier
1854     end else
1855     begin ERR(↓id not declared in form expr↓);
1856     t:= p:= m:= 0
1857 end; if t = formula symbol then
1858 begin if proc id ass stat then
1859     begin if block depth > min block depth ^ (p = procedure
1860     symbol v m = declared as value v m = specified as value v
1861     m = specified as name) v (p = procedure symbol ^
1862     m = specified as name) then

```

```

1863         begin PR string(↓abs(↓)); br:= 2 end
1864     end;
1865     if m = declared as name ^ p ≠ procedure symbol v
1866         p = array symbol then PR string(↓V(↓))else
1867         if m = specified as name ^ p ≠ procedure symbol then
1868         PR string(↓VN(↓)) else br:= br - 1
1869     end else if t = real symbol then PR string(↓RN(↓))
1870     else if t = integer symbol then PR string(↓IN(↓)) else br:= 0;
1871     if t = formula symbol ^ p = 0 ^ m = specified as value
1872     then PR string(↓Y(↓)) else
1873     if m ≠ standard identifier then PR string(↓Z(↓)); PR and RE;
1874     if synt unit = sub symbol then
1875     begin if p ≠ array symbol then ERR(
1876         ↓form array not declared↓);
1877     L: PR and RE; other expression;
1878     if synt unit = comma symbol then goto L;
1879     CHECK(bus symbol); PR and RE
1880     end; for br:= br - 1 while br ≥ 0 do PR string(↓)↓)
1881 end end else
1882
1883     if synt unit = string then
1884     begin if ¬ scan1 then translate string; RE end else
1885
1886     if is enquiry then
1887     begin if scan1 then
1888         begin if synt unit = el symbol then
1889             begin RE; SKIP text until(synt unit = of symbol) end
1890             else RE; CHECK(of symbol); RE;
1891             if synt unit = open symbol then
1892                 begin RE; SKIP text until(synt unit = close symbol); RE
1893                 end else primary:= primary
1894             end else
1895             if synt unit = type symbol v synt unit = length symbol
1896             then begin PR string(↓IN(↓)); enquiry; PR string(↓)↓) end
1897             else enquiry
1898         end else
1899
1900     if ¬ scan1 ^ synt unit = if symbol then

```



```

1901   begin PR string(↓(↓); formula expression; PR string(↓)↓) end else
1902
1903   if synt unit = open symbol then
1904   begin RE; if synt unit = if symbol then
1905       SKIP text until(synt unit = close symbol) else
1906       begin primary:= elevator(3); CHECK(close symbol) end; RE
1907   end else ERR(↓primary in formula expr not O.K.↓)
1908   end primary;
1909
1910   comment: Finally the procedure body of simple formula expression;
1911   scan1:= true; iF:= 0; primary symbol:= -1;
1912   int num:= -2; nr of brackets:= 0;
1913   SAVE reading ptrs;
1914   f:= elevator(if primary only then 0 else 3);
1915   scan1:= false; RESET reading ptrs;
1916   PR form expr(f)
1917 end simple formula expression;
1918
1919 comment:
1920 The body of envelope of block::; block or statement
1921 end envelope of block and all the syntax-translating procedures.;
1922
1923 comment:
1924
1925
1926 6.6. Auxiliary equipment
1927
1928 6.6.1. Declaration and initialization of symbols
1929
1930 Algorithm::;
1931
1932 integer symbol, underlining symbol, space symbol, bar symbol, tab symbol,
1933 nldr symbol, goto symbol, if symbol, then symbol, else symbol, for symbol,
1934 do symbol, step symbol, until symbol, while symbol, comment symbol,
1935 begin symbol, end symbol, Boolean symbol, integer symbol, real symbol,
1936 array symbol, switch symbol, procedure symbol, string symbol, label
1937 symbol, value symbol, formula symbol, type symbol, lhs symbol, rhs
1938 symbol, el symbol, of symbol, constant symbol, monadic symbol, dyadic

```

```

1939 symbol,polyadic symbol,rowadic symbol,length symbol, int div
1940 symbol,unequal symbol,power symbol,becomes symbol, other symbol,plus
1941 symbol,minus symbol,times symbol,over symbol, smaller than
1942 symbol,equal symbol,greater than symbol,and symbol, comma
1943 symbol,point symbol,lower ten symbol,colon symbol, semicolon
1944 symbol,open symbol,close symbol,sub symbol,bus symbol, numb of und
1945 symbols,string,identifier,integral number,real number, prec symbol;
1946 integer array und symbol[1:350],ref1 to und symbol[30:105],
1947 ref2 to und symbol[129:179],adic op[1:6,0:2];
1948
1949 procedure INITIALIZE symbols;
1950 begin integer j,k,p,hash,ptr;
1951 procedure rs(symbol); integer symbol; symbol:= text symbol;
1952 procedure ra(symbol); integer symbol;
1953 begin j:= j + 1; symbol:= j + 128 end;
1954
1955 procedure ru(symbol); integer symbol;
1956 comment
1957 This procedure puts the symbols of the underlined word delimiters,
1958 together with the number of those symbols and the internal
1959 representation (the value of j), in the array "und symbol". For each
1960 underlined word delimiter a hash h is calculated and the index of
1961 the delimiter where it is placed in "und symbol", is put in "ref1 to
1962 und symbol[h]". Moreover, in "ref2 to und symbol[s]", where s is the
1963 internal representation, the same index is put. In the higher syntax
1964 reading procedures the internal representations are used in the form
1965 of "goto symbol", "end symbol" etc. See also "Read synt unit".;
1966 begin integer i,s; i:= 0; j:= j + 1;
1967 for s:= text symbol while s ≠ semicolon symbol do
1968 if s = underlining symbol then
1969 begin s:= text symbol;
1970 if s ≠ underlining symbol then
1971 begin i:= i + 1; und symbol[ptr + i + 1]:= s end
1972 end;
1973 symbol:= und symbol[ptr]:= j + 128;
1974 und symbol[ptr + 1]:= i;
1975 hash:= if i = 1 then und symbol[ptr + 2] else
1976 und symbol[ptr + 2] + und symbol[ptr + i + 1] × 2;

```

```

1977   if hash < 30 v hash > 105 then
1978   FATAL ERR(und symbol in string incorrect);
1979   und symbol[ptr + i + 2]:= ref1 to und symbol[hash];
1980   ref1 to und symbol[hash]:= ref2 to und symbol[symbol]:= ptr;
1981   ptr:= ptr + i + 3
1982   end ru;
1983
1984   integer procedure text symbol;
1985   begin p:= p + 1; text symbol:= STRING SYMBOL(p,
1986   | +-x/<=>^, .10::()[]
1987   goto;go to;if;then;else;for;do;step;until;while;comment;
1988   begin;end;Boolean;boolean;integer;real;array;switch;
1989   procedure;formula;string;label;value;
1990   ;<;>;=;≡;true;false;
1991   type;lhs;length;rhs;el;of;constant;monadic;dyadic;
1992   polyadic;rowadic;
1993   )
1994   end text symbol;
1995
1996   for j:= 30 step 1 until 105 do ref1 to und symbol[j]:= 0;
1997   ptr:= 1; j:= 0; p:= -1;
1998   rs(underlining symbol); rs(space symbol); rs(bar symbol);
1999   rs(tab symbol); rs(plus symbol); rs(minus symbol);
2000   rs(times symbol); rs(over symbol); rs(smaller than symbol);
2001   rs(equal symbol); rs(greater than symbol); rs(and symbol);
2002   rs(comma symbol); rs(point symbol); rs(lower ten symbol);
2003   rs(colon symbol); rs(semi colon symbol); rs(open symbol);
2004   rs(close symbol); rs(sub symbol); rs(bus symbol);rs(nlcr symbol);
2005
2006   k:= ptr; ru(goto symbol); ru(goto symbol);
2007   und symbol[k]:= goto symbol; ru(if symbol);ru(then symbol);
2008   ru(else symbol); ru(for symbol); ru(do symbol); ru(step symbol);
2009   ru(until symbol); ru(while symbol); ru(comment symbol);
2010   ru(begin symbol); ru(end symbol); k:= ptr;
2011   ru(Boolean symbol); ru(Boolean symbol);
2012   und symbol[k]:= Boolean symbol; ru(integer symbol);
2013   ru(real symbol); ru(array symbol); ru(switch symbol);
2014   ru(procedure symbol); ru(formula symbol); ru(string symbol);

```

6-54

```
2015 ru(label symbol); ru(value symbol); ru(int div symbol);
2016
2017 comment the following symbols are read:  $\leq$ ,  $\geq$ ,  $\equiv$ ,  $\sqsubset$ ,
2018 true, false;
2019 for k:= 1,2,3,4,5,6 do ru(symbol);
2020 ru(type symbol); ru(lhs symbol); ru(length symbol); ru(rhs symbol);
2021 ru(el symbol); ru(of symbol); ru(constant symbol);
2022 ru(monadic symbol); ru(dyadic symbol); ru(polyadic symbol);
2023 ru(rowadic symbol); numb of und symbols:= j;
2024
2025 ra(power symbol); ra(becomes symbol); ra(other symbol); ra(string);
2026 ra(identifier); ra(integral number); ra(real number);
2027 fill alf
2028 end INITIALIZE symbols;
2029
2030 procedure fill alf;
2031 comment
2032 This procedure is executed once, in order to define the array alf.
2033 This array is being used in the process to search an identifier of
2034 a standard procedure. All standard identifiers begin with lower case
2035 letters with internal representation from 10 to 36. One should compare
2036 the procedures: "Search for standard identifier" and "standard symbol"
2037 of the next section. The absolute value of alf[s] determines for a
2038 certain letter, with internal representation s, the entry in the
2039 string of standard identifiers where the identifiers with this letter
2040 as first one are placed.;
2041 begin integer i,k,s; boolean lc; k:= -1;
2042 for i:= 10 step 1 until 36 do alf[i]:=  $10^4$ ;
2043 cycle: lc:= true;
2044 k:= k + 1; s:= standard symbol(k);
2045 if s = 255 then goto out;
2046 if s = plus symbol then
2047 begin lc:= false; i:= k; k:= k + 1;
2048 s:= standard symbol(k)
2049 end;
2050 if s < 10 v s > 36 then
2051 ERR({list of standard symbols not correct});
2052 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 ^ lc then
2056     begin lc:= 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,adic sym;
2071 Boolean from delimiter array;
2072 integer array text array[0:10000],alf[10:36],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) : 3;
2079   j:= ptr of text - i × 3;
2080   text array[i]:=
2081     if j = 1 then s × 160000 else
2082     if j = 2 then text array[i] : 160000 × 160000 + s × 400 else
2083     text array[i] : 400 × 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) : 3; j:= p - i × 3;
2088   t:= text array[i];
2089   if j = 1 then t:= t : 160000 else
2090   if j = 2 then begin t:= t - t : 160000 × 160000; t:= t : 400 end

```

```

2091  else t:= t - t : 400 × 400;
2092  take from text array:= t
2093  end;
2094
2095  procedure NS;
2096  if first scan ^ reading allowed then
2097  begin prec symbol:= symbol;
2098  comment
2099  The symbols are normally read with RESYM. If, however, an erroneous
2100  parameter delimiter has been read of the form ")abc:1(", then the
2101  symbols "abc:1" have been put in the array "delimiter array" and the
2102  boolean "from delimiter array" has been made equal to true. See also
2103  READ synt unit.;
2104  if from delimiter array then
2105  begin delimiter array ptr:= delimiter array ptr + 1;
2106  symbol:= delimiter array[delimiter array ptr];
2107  if symbol = -1 then
2108  begin from delimiter array:= false; symbol:= RESYM end
2109  end else symbol:= RESYM;
2110  put in text array(symbol);
2111  if symbol = semicolon symbol ^
2112  prec symbol = underlining symbol then
2113  early end of program;
2114  comment
2115  If the symbol ";;" is read, the procedure "early end of program"
2116  is called to provide for as many string quotes and end's that are
2117  necessary to appropriately close the text of the ABC ALGOL program
2118  under consideration.;
2119  if symbol = nlcr symbol then
2120  begin line number2:= line number2 + 1;
2121  NLCR; ABSFIXT(4,0,line number2)
2122  end else PRSYM(symbol)
2123  end else
2124  begin ptr of text:= ptr of text + 1;
2125  symbol:= take from text array(ptr of text);
2126  if symbol = nlcr symbol then line number2:= line number2 + 1
2127  end NS;
2128

```

```

2129 procedure early end of program;
2130 begin integer i,p; p:= ptr of text:= ptr of text - 2;
2131   for i:= 1 step 1 until nr of stringquotes do
2132     begin put in text array(bar symbol);
2133       put in text array(greater than symbol)
2134     end;
2135     put in text array(semicolon symbol);
2136     put in text array(semicolon symbol);
2137     for i:= 1 step 1 until nr of begins do
2138       put in text array(end symbol);
2139       reading allowed:= false;
2140       ERR(†program text contains ; †);
2141       ptr of text:= p
2142     end early end of program;
2143
2144 integer array und symbol read[1:9];
2145
2146 integer procedure READ synt unit;
2147 begin if is lay out(symbol) then
2148   begin L8: if symbol = space symbol then
2149     ptr of text:= ptr of text - 1;
2150     NS; if is lay out(symbol) then goto L8
2151   end;
2152   line number1:= line number2;
2153   if symbol = underlining symbol then
2154     begin integer i,j,k,hash,nr of letters,ptr; i:= 0;
2155     L: ptr of text:= ptr of text - 1; NS;
2156     if ¬ reading allowed then
2157       begin READ synt unit:= other symbol; goto endprog end;
2158     if symbol = underlining symbol then goto L;
2159     i:= i + 1; if i > 9 then
2160       begin ptr of text:= ptr of text - 1; NS;
2161         if symbol = underlining symbol then goto L;
2162         i:= other symbol; goto OUT
2163       end;
2164     und symbol read[i]:= symbol;
2165     ptr of text:= ptr of text - 1; NS;
2166     if symbol = underlining symbol then goto L;

```

```

2167     hash:= if i = 1 then und symbol read[1] else
2168     und symbol read[1] + 2 × und symbol read[i];
2169     if hash < 30 v hash > 105 then ptr:= 0 else
2170     ptr:= ref1 to und symbol[hash];
2171     comment
2172     From the hash of the underlined symbol read, the place of this
2173     symbol in "und symbol" is determined and it is checked whether the
2174     symbol has been written correctly.;
2175     cyc: if ptr = 0 then
2176     begin i:= other symbol; goto OUT end;
2177     nr of letters:= und symbol[ptr + 1];
2178     if nr of letters = i then
2179     begin j:= ptr + 1;
2180     for k:= 1 step 1 until i do
2181     if und symbol read[k] ≠ und symbol[j + k] then goto next;
2182     i:= und symbol[ptr]; goto OUT
2183     end;
2184     next: ptr:= und symbol[ptr + nr of letters + 2]; goto cyc;
2185     OUT: READ synt unit:= i; ptr of text:= ptr of text - 1;
2186     comment
2187     Note that the internal representation, determined by "und
2188     symbol[ptr]", is put in the text array. Not the symbols. This makes
2189     the reading of the text during the second scan much more rapid.;
2190     put in text array(i); put in text array(symbol);
2191     endprog:
2192     end else
2193
2194     if symbol = colon symbol then
2195     begin NS; if symbol = equal symbol then
2196     begin READ synt unit:= becomes symbol;
2197     ptr of text:= ptr of text - 2;
2198     put in text array(becomes symbol); NS
2199     end else
2200     READ synt unit:= colon symbol
2201     end else
2202
2203     if symbol = bar symbol then
2204     begin NS; if symbol = equal symbol then

```



```

2205     begin NS; READ synt unit:= unequal symbol end else
2206     if symbol = and symbol then
2207     begin NS; READ synt unit:= power symbol end else
2208     if symbol = smaller than symbol then
2209     begin integer s; s:= symbol; nr of stringquotes:= 1;
2210     L2: NS; if s = bar symbol then
2211         begin if symbol = smaller than symbol then
2212             nr of stringquotes:= nr of stringquotes + 1 else
2213                 if symbol = greater than symbol then
2214                     begin nr of stringquotes:= nr of stringquotes - 1;
2215                     if nr of stringquotes = 0 then goto L3 end
2216                 end; s:= symbol; goto L2;
2217     L3: NS; READ synt unit:= string
2218     end else READ synt unit:= other symbol
2219 end else
2220
2221     if is letter(symbol) then
2222     begin integer i,j,k,ptr,fill ptr,A,B,nls in ident;
2223     comment
2224     An identifier will be read now. It is compared with existing
2225     identifiers, which are stored as a binary tree in the upper part of
2226     the information list. The text array will contain the identification
2227     number of the identifier plus the new line symbols occurring in the
2228     identifier, but not the letters and digits.
2229     If the identification number is larger than 218, three "symbols"
2230     will be put in the text array: 399 and two derived from this number.;
2231     A:= symbol + 1; i:= 1; j:= ptr to name list;
2232     READ synt unit:= identifier;
2233     ptr of text2:= ptr of text; nls in ident:= 0;
2234     if line number 1  $\geq$  2000 then FATAL ERR(†too much lines†);
2235     ptr to first letgit:= ptr of text  $\times$  2000 +
2236     line number1;
2237     L1: NS; if is lay out(symbol) then
2238     begin if symbol = nler symbol then
2239         nls in ident:= nls in ident + 1;
2240         goto L1
2241     end;
2242     if is letter(symbol) v is digit(symbol) then

```

```

2243 begin i:= i + 1; if i = 5 then
2244     begin contents of [j]:= A; A:= 0;
2245         j:= j - 1; if j - 7 ≤ ptr of inf list then
2246             FATAL ERR(↓inf list too small↓);
2247             i:= 1
2248         end;
2249         A:= A × 64 + symbol + 1; goto L1
2250     end;
2251     contents of [j]:= - A × 64 ↑ (4 - i);
2252     if j - 7 ≤ ptr of inf list then
2253         FATAL ERR(↓inf list too small↓);
2254     ptr:= ptr to first ident;
2255     for i:= ptr while ptr ≠ 0 do
2256         begin k:= ptr to name list;
2257         next: A:= contents of [i]; B:= contents of [k];
2258             if A = B then
2259                 begin if A < 0 then goto FOUND else
2260                     begin i:= i - 1; k:= k - 1; goto next end
2261                 end;
2262             k:= if abs(A) > abs(B + .5) then 1 else 2;
2263     L2: if A > 0 then
2264         begin i:= i - 1; A:= contents of [i]; goto L2 end;
2265         fill ptr:= i - k;
2266         ptr:= contents of [i - k]
2267     end;
2268
2269     comment We treat a new identifier; i:= j;
2270     contents of [i - 1]:= contents of [i - 2]:= 0;
2271     contents of [i - 3]:= nr of identifiers:= nr of identifiers + 1;
2272     if nr of identifiers > max nr of identifiers then
2273         FATAL ERR(↓too much identifiers↓);
2274     code table [nr of identifiers]:= ptr to name list;
2275     if ptr to first ident = 0 then ptr to first ident:=
2276         ptr to name list else
2277         contents of [fill ptr]:= ptr to name list;
2278     ptr to name list:= i - 4;
2279     FOUND: nr of ident:= contents of [i - 3];
2280     ptr of text:= ptr of text2 - 1;

```

```

2281   if nr of ident < 218 then
2282     put in text array (nr of ident + 180) else
2283     begin put in text array(399);
2284       put in text array(nr of ident : 400);
2285       put in text array(nr of ident - nr of ident : 400 × 400)
2286     end;
2287     for i:= 1 step 1 until nls in ident do
2288       put in text array(nlcr symbol);
2289       put in text array(symbol);
2290     end else
2291
2292   if symbol > 180 then
2293     begin READ synt unit:= identifier; ptr of text2:=
2294       ptr to first letgit:= ptr of text;
2295       if symbol = 399 then
2296         begin NS; nr of ident:= symbol × 400;
2297         NS; nr of ident:= nr of ident + symbol
2298         end else nr of ident:= symbol - 180;
2299       NS
2300     end else
2301
2302   if is digit(symbol) v symbol = point symbol v
2303     symbol = lower ten symbol then
2304     begin Boolean integer; integer:= true;
2305     L5: if symbol = point symbol then integer:= false else
2306       if symbol = lower ten symbol then
2307         begin L51: NS; if is layout(symbol) then goto L51;
2308         integer:= false
2309         end else if ¬ (is layout(symbol) v is digit(symbol))
2310         then goto L52; NS;
2311         goto L5; L52:
2312         READ synt unit:= if integer then
2313         integral number else real number
2314       end else
2315
2316   if symbol = close symbol then
2317     begin integer p,l,nls in delimiter; if first scan then
2318       begin p:= ptr of text; l:= linenum2;

```

```

2319     nls in delimiter:= 0
2320     end;
2321     L6: NS; if is letter(symbol) then
2322         begin comment
2323             Tentatively, a parameter delimiter will be read. If it turns out
2324             that the delimiter is not well formed an error message is given and
2325             the reading ptrs are set back to the beginning after the insertion of
2326             a semicolon symbol which most probably has been forgotten.;
2327             procedure store(s); value s; integer s;
2328             begin if delimiter array ptr < 29 then
2329                 begin delimiter array ptr:= delimiter array ptr + 1;
2330                 delimiter array[delimiter array ptr]:= s
2331             end;
2332             if s = nlcr symbol then
2333                 nls in delimiter:= nls in delimiter + 1
2334             end;
2335             delimiter array ptr:= 0;
2336             L7: store(symbol); NS;
2337             if is letter(symbol) v is digit(symbol) v
2338             is lay out(symbol) then goto L7;
2339             if symbol = colon symbol then
2340             begin L8: store(symbol); NS;
2341                 if is lay out(symbol) then goto L8;
2342                 if symbol = open symbol then
2343                     begin ptr of text:= p - 1;
2344                     put in text array(comma symbol);
2345                     for p:= 1 step 1 until nls in delimiter do
2346                         put in text array(nlcr symbol);
2347                         READ synt unit:= comma symbol; NS; goto out
2348                     end else store(symbol)
2349             end;
2350             L9: ptr of text:= p; linenum2:= 1;
2351             from delimiter array:= true;
2352             put in text array(semicolon symbol);
2353             symbol:= semicolon symbol;
2354             delimiter array[delimiter array ptr + 1]:= - 1;
2355             delimiter array ptr:= 0; READ synt unit:= close symbol;
2356             ERR(↓incorrect parameter delimiter↓)

```

```

2357     end else if is lay out(symbol) then goto L6 else
2358     READ synt unit:= close symbol;
2359 out: end else
2360     begin READ synt unit:= symbol; NS end
2361 end READ synt unit;
2362
2363 Boolean procedure is adic symbol(s); value s; integer s;
2364 is adic symbol:= constant symbol  $\leq s \wedge s \leq$  rowadic symbol;
2365
2366 Boolean procedure is declarator(s); value s; integer s;
2367 is declarator:= Boolean symbol  $\leq s \wedge s \leq$  formula symbol;
2368
2369 Boolean procedure is specifier(s); value s; integer s;
2370 is specifier:= Boolean symbol  $\leq s \wedge s \leq$  label symbol;
2371
2372 Boolean count nr of begins, delay one RE, no semicolon or
2373 begin end allowed, reading allowed;
2374
2375 Boolean procedure is letter(s); value s; integer s;
2376 is letter:=  $10 \leq s \wedge s \leq 62$ ;
2377 Boolean procedure is layout(s); value s; integer s;
2378 is layout:=  $s = \text{space symbol} \vee s = \text{tab symbol} \vee s = \text{nocr symbol}$ ;
2379 Boolean procedure is digit(s); value s; integer s;
2380 is digit:=  $0 \leq s \wedge s \leq 9$ ;
2381
2382 Boolean procedure is enquiry; is enquiry:=
2383 type symbol  $\leq$  synt unit  $\wedge$  synt unit  $\leq$  el symbol;
2384
2385 Boolean procedure is operator(s,n); value s; integer s,n;
2386 begin procedure P(op); value op; integer op;
2387     begin n:= n + 1; if s = op then goto OUT end;
2388     n:= 0; P(plus symbol); P(minus symbol); P(times symbol);
2389     P(over symbol); P(int div symbol); P(power symbol);
2390     is operator:= false; goto END;
2391 OUT: is operator:= true;
2392 END: end is operator;
2393
2394 procedure RE;

```

```

2395 if delay one RE then delay one RE:= false else
2396 if no semicolon or begin end allowed ^
2397     (synt unit = semicolon symbol v
2398     synt unit = begin symbol v
2399     synt unit = end symbol) then else
2400 begin integer n; synt unit:= next synt unit;
2401     operator identifier:= false;
2402     ptr of text1:= ptr of text2; ptr of text2:= ptr of text;
2403     line number:= line number1;
2404     next synt unit:= if nr of begins > 0 then READ synt unit else
2405         synt unit;
2406 L0: if count nr of begins then
2407     begin if next synt unit = begin symbol then
2408         nr of begins:= nr of begins + 1 else
2409         if next synt unit = end symbol then nr of begins:=
2410             nr of begins - 1
2411     end;
2412     if synt unit = end symbol then
2413     begin L: if next synt unit ≠ end symbol ^ next synt unit ≠
2414         else symbol ^ next synt unit ≠ semicolon symbol then
2415
2416         begin ptr of text2:= ptr of text; next synt unit:= READ synt unit;
2417         if count nr of begins ^ next synt unit = end symbol then
2418             nr of begins:= nr of begins - 1; goto L
2419         end
2420     end;
2421     if next synt unit = comment symbol then
2422     begin if ¬ (synt unit = semicolon symbol v synt unit = begin symbol)
2423         then ERR(↓comment not appropriate↓);
2424         ptr of text:= ptr of text -2;
2425     L1: if symbol ≠ semicolon symbol then
2426         begin NS; if symbol ≠ nler symbol then ptr of text:=
2427             ptr of text - 1; goto L1
2428         end;
2429     NS; next synt unit:= READ synt unit; goto L0
2430 end else
2431 if is adic symbol(synt unit) ^ is operator(next synt unit,n) then
2432 begin adic sym:= adic op[n,synt unit - constant symbol];

```

```

2433     if adic sym = 0 then ERR(†wrong combination for operator ident†);
2434     ptr to first letgit:= ptr of text1 × 2000 + line number1;
2435     nr of ident:= - (synt unit × 1024 + next synt unit);
2436     RE; operator identifier:= true; if next synt unit = identifier
2437     then ERR(†identifier after operator identifier†);
2438     synt unit:= identifier
2439     end
2440 end RE;
2441
2442 procedure RE semicolon;
2443 begin if synt unit ≠ semicolon symbol then
2444     begin ERR(†no semicolon where required†);
2445     SEEK(synt unit = semicolon symbol v
2446     synt unit = end symbol)
2447     end; no semicolon or begin end allowed:= false;
2448     if synt unit ≠ end symbol then RE;
2449     no semicolon or begin end allowed:= true
2450 end;
2451
2452 procedure RE end;
2453 begin no semicolon or begin end allowed:= false; RE;
2454     no semicolon or begin end allowed:= true
2455 end;
2456
2457 procedure RE begin; RE end;
2458
2459 procedure SAVE reading ptrs;
2460 begin integer i,j; j:= 0; count nr of begins:= false;
2461     nr of begins:= nr of begins + 2;
2462     comment The reason to augment "nr of begins" is that under
2463     extraordinary circumstances this variable could otherwise obtain the
2464     value -1. E.g. when "synt unit" denotes the last end, as in "... f:= 5
2465     end";
2466     ptr of old reading ptrs:= ptr of old reading ptrs + 1;
2467     for i:= synt unit,next synt unit,ptr to first letgit,
2468     nr of ident,ptr of text1,ptr of text2,ptr of text,symbol,
2469     line number,line number1,line number2 do
2470     begin j:= j + 1; old reading ptrs[ptr of old reading ptrs,j]:=i end;

```

```

2471   old reading ptrs[ptr of old reading ptrs,j+1]:=
2472   if operator identifier then 1 else 0
2473   end SAVE reading ptrs;
2474
2475   integer array old reading ptrs[1:10,1:12];
2476   integer ptr of old reading ptrs;
2477
2478   procedure RESET reading ptrs;
2479   begin integer i; procedure A(x); integer x;
2480     begin i:= i + 1; x:= old reading ptrs[ptr of old reading ptrs,i]
2481     end; i:= 0; A(synt unit); A(next synt unit);
2482     A(ptr to first letgit); A(nr of ident); A(ptr of text1);
2483     A(ptr of text2); A(ptr of text); A(symbol); A(line number);
2484     A(line number1); A(line number2);
2485     operator identifier:=
2486     old reading ptrs[ptr of old reading ptrs,i + 1] = 1;
2487     if ptr of old reading ptrs = 1 then count nr of begins:= true;
2488     nr of begins:= nr of begins - 2; delay one RE:= false;
2489     ptr of old reading ptrs:= ptr of old reading ptrs - 1
2490   end RESET reading ptrs;
2491
2492   procedure SET reading ptrs on(t); value t; integer t;
2493   begin ptr of text:= t : 2000; operator identifier:= false;
2494     line number2:= t - ptr of text × 2000;
2495     ptr of text:= ptr of text - 1; NS; synt unit:= next synt unit:=
2496     comma symbol; delay one RE:= false; RE; RE
2497   end SET reading ptrs on;
2498   comment
2499   The following procedure skips text until a certain condition "b" is
2500   satisfied. The condition can be: "synt unit = close symbol" or "synt
2501   unit = bus symbol" or "synt unit = comma symbol v synt unit = close
2502   symbol".
2503   The symbols ), ] and , are read on the appropriate level, i.e. if
2504   they occur in a bracket structure [...] or (...), then the complete
2505   text enclosed by the brackets including the brackets is skipped
2506   automatically. The effect is that "a + b[x × (y + z) + 5] + c)"
2507   is treated as "a + b + c)", so that "skips until" has the desired
2508   effect.

```


2509 In order to restore erroneous situations, in which left brackets do
 2510 not have corresponding right brackets, singly occurring right brackets
 2511 stop the skipping process, and, if "b" is not true, "delay one RE"
 2512 gets the value true so that a next call of RE will not result in
 2513 reading a syntactic unit. Due to this construction, skipping of the
 2514 subscript of a in: "a[(b + c[d × (e + f)]]", will be successful.;
 2515
 2516 procedure SKIP text until(b); Boolean b;
 2517 begin delay one RE:= false;
 2518 L: if synt unit = end symbol v synt unit = begin symbol then
 2519 begin ERR(4nr of brackets incorrect); goto end end
 2520 else if synt unit = semicolon symbol then goto end
 2521 else if synt unit = open symbol then
 2522 begin RE; SKIP text until (synt unit = close symbol);
 2523 RE; goto L
 2524 end else
 2525 if synt unit = sub symbol then
 2526 begin RE; SKIP text until (synt unit = bus symbol);
 2527 RE; goto L
 2528 end else
 2529 if is enquiry then
 2530 begin if synt unit = el symbol then
 2531 begin RE; SKIP text until(synt unit = of symbol);
 2532 RE; goto L
 2533 end else
 2534 begin RE; CHECK(of symbol); RE; goto L end
 2535 end else
 2536 if synt unit = close symbol v synt unit = bus symbol v
 2537 synt unit = of symbol then
 2538 begin if ¬ b then
 2539 begin delay one RE:= true;
 2540 comment For "additional synt unit" see the
 2541 procedure "PR and RE" and CHECK;
 2542 additional synt unit:= 0
 2543 end;
 2544 goto end
 2545 end;
 2546 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  $\neg b \wedge$  synt unit  $\neq$  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( $\downarrow$ operator identifier in wrong place $\downarrow$ );
2571     goto notfound
2572   end; in:= i:= code table[nr of ident];
2573   s:= contents of [i]; first letter:= abs(s)  $\div$  t18 - 1;
2574   uc:= first letter > 36; if uc then first letter:=
2575     first letter - 27; k:= alf[first letter];
2576   if k =  $_{10}4 \vee k > 0 \wedge$  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;

```

```

2585     if s2 = open symbol then
2586     begin t:= -t×64↑(4-s); goto compare end;
2587     s1:=s2; k:= k+1
2588     end;
2589 compare: s:= contents of [i];
2590     if s = t ^ s < 0 then goto found else if s = t then
2591     begin i:= i-1; s1:= s2; goto newt end else
2592     if abs(t) > abs(s + .5) then goto notfound else
2593 skipid:
2594     begin for k:= k+1 while standard symbol(k) ≠ semicolon symbol,
2595     k+1 do; i:= in; goto new id
2596     end;
2597 notfound: Search for standard identifier:= false; nr:= 0; goto out;
2598 found: Search for standard identifier:= true;
2599     k:= k+1; nr:= standard symbol(k);
2600     k:= k+1; i:= standard symbol(k);
2601     type of standard identifier:=
2602     if i = close symbol then real symbol else
2603     if i = bus symbol then integer symbol else 0;
2604 out:
2605 end Search for standard identifier;
2606
2607 integer procedure standard symbol(k); integer k;
2608 begin integer s;
2609 cyc: s:= STRING SYMBOL(k,↑
2610     abs(1);absfixc(3>;+absfixp(3>;+absfixt(3>;and(2];
2611     arctan(1);available(0]; /
2612     bit(2];bitstring(3]; /
2613     +carriage(1>;circ shift(2];clear shift(2];col(1>;
2614     compose(2);cos(1);cpos(0];cpunch(1>;csym(1>; /
2615     date(0]; /
2616     entier(1];+even(1];+exit(0>;exp(1); /
2617     fix(4>;fixc(3>;+fixp(3>;+fixt(3>;flo(4>;floc(3>;
2618     +flop(3>;+flot(3>;+from drum(2>; /
2619     head of(1]; /
2620     +inprod(5); /
2621     +line number(0];ln(1); /
2622     matmat(6);mattam(6);matvec(5); /

```

```

2623 +new page(0>;+nlcr(0>;norm shift(2]; /
2624 or(2];outflexo(0>;outiso(0>; /
2625 +print(1>;print pos(0];+printtext(1>;+prsym(1>;+puhep(1>;
2626 +punch(1>;+punlcr(0>;+puspace(1>;+pusym(1>;+putext(1>; /
2627 +random(0);+read(0);read1(2);+rehep(0];+remainder(2];
2628 +resym(0];resymbol(0];+runout(0>; /
2629 set(4];+setrandom(1>;sign(1];sin(1);+space(1>;sqrt(1);
2630 +stopcode(0>;+stringsymbol(2];+sum(4); /
2631 +tab(0>;tail of(1];tammat(6);tamvec(5);time(0);+to drum(2>; /
2632 vecvec(5); /
2633 xor(2]; / †); if is lay out(s) then begin k:= k+1; goto cyc end
2634 else standard symbol:= s
2635 end standard symbol;
2636
2637 procedure INITIALIZE reading ptrs;
2638 begin ptr of text:= ptr of text1:= ptr of text2:= 0;
2639 synt unit:= next synt unit:= semicolon symbol; symbol:= 0;
2640 line number:= line number1:= line number2:= 0;
2641 ptr to first letgit:= nr of ident:= 0; nr of begins:= 2;
2642 delay one RE:= false; from delimiter array:= false;
2643 count nr of begins:= true; ptr of old reading ptrs:= 0;
2644 nr of stringquotes:= 0; NS; RE semicolon; RE semicolon;
2645 nr of begins:= nr of begins - 2
2646 end;
2647
2648 comment:
2649
2650 6.6.3. The printing equipment
2651
2652 Algorithm::
2653
2654 procedure PR synt unit; if second scan then
2655 begin integer i,s,j,k,t,ptr,bound; Boolean nlcr pr;
2656 if operator identifier then
2657 begin operator identifier:= false;
2658 ERR(†operator ident in illegal place†)
2659 end; nlcr pr:= false;
2660 for i:= ptr of text1 step 1 until ptr of text2 - 1 do

```

```

2661 begin s:= take from text array(i);
2662   if s < 128 then
2663     begin if s = nlcr symbol then
2664       begin if nlcr pr ^ synt unit ≠ string then else
2665         begin PR sym(s); nlcr pr:= true end
2666       end else PR sym(s)
2667     end else
2668     begin if s = becomes symbol then PR string(↓:=↓) else
2669       if s = other symbol then PR string(↓other ↓) else
2670       if s < 180 then
2671         begin ptr:= ref2 to und symbol[s];
2672         bound:=ptr + und symbol[ptr + 1] + 1;
2673         for k:= ptr + 2 step 1 until bound do
2674           begin PUSYM(underlining symbol); PRSYM(underlining symbol);
2675           t:= und symbol[k]; PRSYM(t); PUSYM(t)
2676         end; PR string(↓ ↓)
2677       end else
2678       begin j:= if s < 399 then code table[s - 180]
2679         else code table[take from text array(i + 1) × 400
2680           + take from text array(i + 2)];
2681       if s = 399 then i:= i + 2; j:= j + 1; k:= 1;
2682       for j:= j - 1 while k > 0 do
2683         begin k:= contents of [j]; t:= abs(k);
2684         for t:= t while t > 0 do
2685           begin s:= t ÷ t18; t:= (t - s × t18) × 64;
2686           PUSYM(s - 1); PRSYM(s - 1)
2687         end end; PR string(↓ ↓)
2688       end end end end PR synt unit;
2689
2690 procedure PR and RE;
2691 begin if second scan ^ delay one RE then
2692 SHOW(additional synt unit,true) else PR synt unit; RE
2693 end;
2694
2695 procedure PR and RE semicolon;
2696 begin if synt unit = semicolon symbol then
2697 PR synt unit else SHOW(semicolon symbol,true);
2698 RE semicolon

```

```

2699 end;
2700
2701 procedure PR and RE end;
2702 begin if synt unit = end symbol then
2703   PR synt unit else SHOW(end symbol,true);
2704   RE end
2705 end;
2706
2707 procedure PR and RE begin;
2708 begin if synt unit = begin symbol then
2709   PR synt unit else SHOW(begin symbol,true);
2710   RE begin
2711 end;
2712
2713 integer procedure PR string(s); string s;
2714 begin PR string:= 1; if second scan then
2715   begin PRINTTEXT(s); PUTTEXT(s) end
2716 end;
2717
2718 integer procedure PR nlcr; PR nlcr:= PR string(†
2719 †);
2720
2721 procedure PR sym(i);
2722 if second scan then begin PRSYM(i); PUSYM(i) end;
2723
2724 procedure SHOW(su,punch); value su; integer su; Boolean punch;
2725 begin procedure p(s); string s;
2726   begin PRINTTEXT(s);if punch then PUTTEXT(s) end;
2727   procedure PR(s); value s; integer s;
2728   begin PRSYM(s); if punch then PUSYM(s) end;
2729   if su = unequal symbol then p(††) else
2730   if su = power symbol then p(††) else
2731   if su = becomes symbol then p(†:=†) else
2732   if su = string then p(†"string"†) else
2733   if su = identifier then p(†"identifier"†) else
2734   if su = integral number then p(†"int num"†) else
2735   if su = real number then p(†"real num"†) else
2736   if su = other symbol then p(†"unknown"†) else

```

```

2737     if su < 128 then PR(su) else
2738     begin integer j,k,ptr,bound; ptr:=ref2 to und symbol[su];
2739         bound:=ptr + und symbol[ptr + 1] + 1;
2740         for k:= ptr + 2 step 1 until bound do
2741             begin PR(underlining symbol); PR(und symbol[k]) end;
2742             PR(space symbol)
2743     end end SHOW;
2744
2745     integer procedure PR int num(a); value a; integer a;
2746     begin integer b; PR int num:= 1;
2747     if a < 0 then begin PR string(↵); a:= -a end;
2748     if a < 9 then PR sym(a) else
2749     begin b:= a : 10; a:= a - b × 10; PR int num(b); PR sym(a) end
2750     end PR int num;
2751
2752     procedure PR operator;
2753     begin operator identifier:= false;
2754     if adic sym ≥ 256 then PR sym(adic sym : 256);
2755     PR sym(adic sym - adic sym : 256 × 256)
2756     end;
2757
2758     procedure ERR(s); string s;
2759     begin integer i; i:= print pos; NLCR;PRINTTEXT(
2760         ↵xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx↵); ABSFIXT(4,0,line number);
2761     PRINTTEXT(s); SPACE(80 - print pos);
2762     SHOW(synt unit,false); SPACE(95 - print pos);
2763     SHOW(next synt unit,false); SPACE(110 - print pos);
2764     NLCR; SPACE(i)
2765     end ERR;
2766
2767     procedure FATAL ERR(s); string s;
2768     begin ERR(s); EXIT end;
2769
2770     procedure CHECK(s); value s; integer s;
2771     if synt unit ≠ s then
2772     begin integer i; i:= print pos; NLCR;
2773     ERR(↵synt unit not OK↵);
2774     NLCR; PRINTTEXT(↵synt unit should be:↵);

```

6-74

```
2775  SHOW(s,false); NLCR; SPACE(i); delay one RE:= 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  integer ptr of inf list,max of inf list,ptr to first ident,
2804  ptr to name list,nr of identifiers,max nr of identifiers,
2805  t10,t15,t18,declared as value,declared as name, specified as
2806  value,specified as name,standard identifier,with local,
2807  reinitptr,without local,block depth,min block depth,in formula
2808  procedure body;
2809  Boolean normal compilation,print information list;
2810  integer array contents of [0:5000],code table[1:400];
2811  comment
2812  If "normal compilation" has the value true "INITIALIZE inf list ptrs"
```


2813 begins with filling the information list and the contents of some "inf
 2814 list ptrs" with the information of the catalogue. This is done via the
 2815 procedure "reinit", which uses the procedure "catalogue symbol", in
 2816 which, in the form of a string, this information is stored.
 2817 The information of the catalogue is itself produced by the same
 2818 compiler with "normal compilation" having the value false, while "inf
 2819 list ptrs" are given their purely initial values. As a result, a
 2820 string is produced being the procedure "catalogue symbol", by means of
 2821 the procedure "print catalogue".
 2822 The procedure "PR inf cells" does not come into action normally, as
 2823 the boolean "print information list" has the value false. If one needs
 2824 the contents of the information list, however, this value can easily
 2825 be made true in the main program.;
 2826
 2827 procedure INITIALIZE inf list ptrs;
 2828 begin integer procedure val(v); value v; integer v;
 2829 val:= if normal compilation then reinit else v;
 2830 max of inf list:= 5000; max nr of identifiers:= 400;
 2831 t10:= 2 ↑ 10; t15:= t10 × 32; t18:=t15 × 8; reinitptr:= -1;
 2832 declared as value:= 1; declared as name:= 2; specified as value:= 3;
 2833 specified as name:= 4; standard identifier:= 5; with local:= 6;
 2834 without local:= 7; block depth:= min block depth:= 0;
 2835 in formula procedure body:= 0;
 2836 ptr to first ident:= val(0); ptr of inf list:= val(1);
 2837 ptr to name list:= val(max of inf list);
 2838 nr of identifiers:= val(0); if normal compilation then
 2839 begin integer i;
 2840 for i:= ptr of inf list - 1 step - 1 until 1,
 2841 ptr to name list + 1 step 1 until max of inf list do
 2842 contents of[i]:= reinit;
 2843 for i:= 1 step 1 until nr of identifiers do
 2844 codetable[i]:= reinit
 2845 end
 2846 end;
 2847
 2848 procedure PR inf cells; if print information list then
 2849 begin integer i,p,s,t; p:= 1;
 2850 NLCR; NLCR; PRINTTEXT(↓contents of inf list↓);

```

2851 L: NLCR; if p ≥ ptr of inf list then
2852   begin p:=ptr to first ident; goto L2 end;
2853   for i:= p step 1 until p + contents of [p] - 1 do
2854     begin ABSFIXT(4,0,i); FIXT(8,0,contents of [i]) end;
2855     p:= p + contents of [p]; goto L;
2856 L2: if p = ptr to name list then goto out;
2857   NLCR; ABSFIXT(4,0,p); i:=1;
2858   for s:= contents of[p] while i > 0 do
2859     begin i:= s; s:= abs(s);
2860       for s:= s while s > 0 do
2861         begin t:= s : t18; s:= (s - t × t18) × 64;
2862           PRSYM(t - 1)
2863         end;
2864         p:= p - 1
2865       end;
2866   NLCR; PR string(† †);
2867   for i:= p,p - 1,p - 2 do ABSFIXT(4,0,contents of[i]);
2868   p:= p - 3; goto L2;
2869 out: NLCR; NLCR; for i:= 1 step 1 until nr of identifiers do
2870   begin ABSFIXT(4,0,i); ABSFIXT(4,0,codetable[i]) end
2871 end PR inf cells;
2872
2873 procedure print catalogue;
2874 begin integer i,j;
2875   procedure PR cat(x); value x; integer x;
2876   begin j:= j + 1; PR int num(x);
2877     if j = j : 5 × 5 then PR nlcr else PR string(† †)
2878   end PR cat;
2879   PR string(†
2880 integer procedure catalogue symbol;
2881 begin reinitptr:= reinitptr + 1;
2882   catalogue symbol:= STRING SYMBOL(reinitptr,†);
2883   PR nlcr; PR sym(bar symbol); PR sym(smaller than symbol);
2884   j:= 0; PR cat(ptr to first ident); PR cat(ptr of inf list);
2885   PR cat(ptr to name list); PR cat(nr of identifiers);
2886   for i:= ptr of inf list - 1 step - 1 until 1,
2887   ptr to name list + 1 step 1 until max of inf list do
2888   PR cat(contents of[i]);

```

```
2889   for i:= 1 step 1 until nr of identifiers do
2890     PR cat(codetable[i]); PR nlcr;
2891     PR sym(bar symbol); PR sym(greater than symbol);
2892     PR string(†)
2893   end;†)
2894   end print catalogue;
2895
2896   integer procedure reinit;
2897   begin integer s,k,t;
2898     for s:= catalogue symbol while is lay out(s) do;
2899     k:= if s = minus symbol then 0 else s;
2900     t:= if s = minus symbol then - 1 else 1;
2901     for s:= catalogue symbol while  $\neg$  is lay out(s) do
2902       k:= k  $\times$  10 + s;
2903       reinit:= k  $\times$  t
2904     end reinit;
2905
2906   integer procedure catalogue symbol;
2907   begin reinitptr:= reinitptr + 1;
2908     catalogue symbol:= STRING SYMBOL(reinitptr,
2909   †5000 49 4983 4 1
2910   -141 3 0 28 4882435
2911   24001 4 6 0 0
2912   4685827 20001 3 6 0
2913   0 4882435 16001 2 6
2914   32 0 0 0 40
2915   0 0 46 -140 10
2916   0 3 0 4736 12001
2917   1 7 0 0 0
2918   0 0 0 18 49
2919   -140 10 4 0 0
2920   -4456448 3 0 4987 -5829662
2921   2 4991 0 -4194304 1
2922   0 4995 -2937792 7403158 5000
2923   4995 4991 4987
2924   †)
2925   end;
2926
```

6-78

```
2927 procedure store bits(int,b); integer int; Boolean array b;
2928 begin integer i,k; i:= 0;
2929   for k:= 0,1,2,3,4,5 do
2930     i:= 2 × i + (if b[k] then 1 else 0); int:= i
2931   end;
2932
2933 procedure get bits(int,b); value int; integer int;
2934 Boolean array b;
2935 begin integer k;
2936   for k:= 0,1,2,3,4,5 do
2937     if int/32 > .99 then
2938       begin b[k]:= true; int:= (int - 32) × 2 end else
2939       begin b[k]:= false; int:= int × 2 end
2940   end;
2941
2942 comment:
2943
2944 6.6.5. The main program
2945
2946 Algorithm::
2947
2948 Boolean first scan, second scan,dangerous procedures,
2949   operator identifier;
2950 real t1,t2;
2951 procedure EXIT;
2952 begin integer s; if reading allowed then
2953   begin L: s:= symbol; symbol:= RESYM;
2954     if ¬ (s = underlining symbol ^ symbol = semicolon symbol)
2955     then goto L
2956   end;
2957   t2:= time; PRINTTEXT(⋄time is: ⋄); ABSFIXT(4,2,(t2 - t1//3.6);
2958   PRINTTEXT(⋄ mh⋄); PR sym(nler symbol); t1:= t2; goto BEGIN
2959 end EXIT;
2960
2961 begin integer i,j;
2962   for i:= 0,1,2 do for j:= 1,2,3,4,5,6 do adic op[j,i]:= 0;
2963   adic op[1,2]:= 55; comment S;
2964   adic op[2,2]:= 40; comment D;
```

```

2965   adic op[3,2]:= 52; comment P;
2966   adic op[4,2]:= 53; comment Q;
2967   adic op[5,2]:= 45 × 256 + 53; comment IQ;
2968   adic op[6,2]:= 41; comment E;
2969   adic op[1,1]:= 52 × 256 + 48; comment PL;
2970   adic op[2,1]:= 49 × 256 + 45; comment MI;
2971   adic op[6,1]:= 45 × 256 + 41; comment IE;
2972   adic op[1,0]:= 45 × 256 + 50; comment IN;
2973   adic op[3,0]:= 54 × 256 + 50; comment RN;
2974 end;
2975
2976 INITIALIZE symbols; t1:= time;
2977 normal compilation:= true; print information list:= true;
2978     BEGIN: dangerous procedures:= false; operator identifier:= false;
2979 first scan:= true; second scan:= false; INITIALIZE inf list ptrs;
2980 NEW PAGE; reading allowed:= true;
2981 L: INITIALIZE reading ptrs; if synt unit ≠ begin symbol then
2982   begin ERR(†program does not begin with begin †);
2983     synt unit:= begin symbol; nr of begins:= nr of begins + 1
2984   end;
2985   if nr of begins < 1 then nr of begins:= 1;
2986   begin:= 1; preceding begin:= 4;
2987   envelope of block(0);
2988   if first scan then
2989     begin first scan:= false; second scan:= true;
2990     operator identifier:= false;
2991     if normal compilation then
2992       begin contents of[begin + 2]:=
2993         STORE into information list(st(st(0,
2994         -end symbol),
2995         0))
2996     end else
2997     begin ptr of inf list:= ptr of inf list - 3;
2998     print catalogue; EXIT
2999   end;
3000   comment One extra symbol is placed after the program.;
3001   if symbol = space symbol then ptr of text:= ptr of text - 1;
3002   put in text array(semicolon symbol);

```

6-80

```
3003     RUNOUT;  
3004     PR inf cells; CARRIAGE(3); goto L  
3005     end; EXIT  
3006 end  
3007
```

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 enveloping 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

1	+10	2	-140	3	+115	4	+49
		5	+0	6	+0	7	+0
		8	+0	9	+0	10	+0
11	+7	12	+1	13	+12001	14	+4736
		15	+0	16	+3	17	+0
18	+10	19	-140	20	+46	21	+0

		22	+0	23	+40	24	+0
		25	+0	26	+0	27	+32
28	+6	29	+2	30	+16001	31	+4882435
		32	+0	33	+0		
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 f	0	4991	2
4991 left	4987	0	3
4987 g	0	0	4
4983 P	0	0	5

7- 84

1 5000 2 4995 3 4991 4 4987 5 4983

Translation 1.1.

```
begin real Zf ;lnr(1);begin
procedure ZP (Zf ) ; integer Zf ;begin integer Yf ,fnn ; fnn:= gnn;lnr(3);

DEVAL(Yf ,Zf ) ;ASSIGN(Zg ,S(Yf ,P(IN(2 ),Yf )))
; ERASE(fnn)
end ;
integer Zg ,Zf ,fnn ; fnn:= gnn;lnr(2);
Zf := Zg := NULL;
DE(Zf ,NULL);DE(Zg ,NULL);ZP (V(Zf )) ;ASSIGN(Zf ,S(P(IN(2 ),V(Zf )),V(Zf )))
; ERASE(fnn)
end ;Zf :=2 xZf +Zf

end
```

Example 1.2.

```
1
2 begin integer k ; formula f:= 3.13,g = 3.14 ; formula array h[1:10];
3 formula procedure dyadic + (a,b) ; formula a,b ;
4 dyadic + := 5 ;
5 formula procedure dyadic - (a,b) ; value a,b ; formula a,b ;
6 dyadic - := 5 ;
7 integer i ; real r ;
8 if dyadic f ^ monadic h[7] then
9 for i:= 1,1 step 1 until 1, while i < 10,11 do i:= i + 5 ;
10 f:= lhs of el 5 of rhs of f ;
11 r:= lhs of el 5 of rhs of (f + g x h[3]) ;
12 f:= el 7 of †01234567‡ ;
13 i:= length of †01234567‡ ;
14 r:= date + f + g ;
15 f:= rowadic(10,i,5,i + 5) ;
16 f:= polyadic(10,i,5,i + 5) ;
17 f:= +f + g x 3.14 - f x (-r + 31410-2/f - f) ↑
18 (h[5] + f ; (((f ↑ 1024 ↑ 2048/10)))) ;
```

19 end

contents of inf list

1	+10	2	-140	3	+156	4	+49
		5	+0	6	+0	7	+0
		8	+0	9	+0	10	+0
11	+7	12	+1	13	+12001	14	+4736
		15	+0	16	+3	17	+0
18	+10	19	-140	20	+46	21	+0
		22	+0	23	+40	24	+0
		25	+0	26	+0	27	+32
28	+6	29	+2	30	+16001	31	+4882435
		32	+0	33	+0		
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	+153	52	+87
		53	+144	54	+69	55	+74
		56	+1	57	+1	58	+32
59	+5	60	+5	61	+14002	62	+4718593
		63	+0				
64	+5	65	+2	66	+20002	67	+4882434
		68	+0				
69	+5	70	+4	71	+34002	72	+4882433
		73	+64				
74	+6	75	+6	76	+52002	77	+4887104
		78	+0	79	+1		
80	+7	81	-172096	82	+78003	83	+4887175
		84	+0	85	+2	86	+0
87	+10	88	-140	89	+109	90	+119
		91	+80	92	+0	93	+0
		94	+0	95	+0	96	+0
97	+6	98	+7	99	+84003	100	+4882436
		101	+0	102	+0		
103	+6	104	+8	105	+88003	106	+4882436
		107	+0	108	+0		

7- 86

109	+3	110	-141	111	+49		
112	+7	113	-172097	114	+130005	115	+4887175
		116	+0	117	+2	118	+0
119	+10	120	-140	121	+141	122	+0
		123	+112	124	+135	125	+0
		126	+0	127	+0	128	+32
129	+6	130	+7	131	+136005	132	+4882435
		133	+0	134	+0		
135	+6	136	+8	137	+140005	138	+4882435
		139	+129	140	+0		
141	+3	142	-141	143	+49		
144	+5	145	+9	146	+190007	147	+4718593
		148	+59				
149	+4	150	+10	151	+196007	152	+4751360
153	+3	154	-141	155	+1		
156	+3	157	-141	158	+0		

5000 replace	4995	0	1
4995 f	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

1	5000	2	4995	3	4991	4	4987	5	4983	6	4979
7	4975	8	4971	9	4967	10	4963	11	4959		

Translation 1.2.

```
begin  
integer low1c1,up1c1,i1;  
integer fmn; fmn:= gm;lmr(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 Za ,Zb ;
begin integer Yb ,Ya ,fnn; fnn:= gmn;lnr(6);

DEVAL(Ya ,Za );DEVAL(Yb ,Zb );D:= IN(5)
; ERASE(fnn)
end ;real Zr ;
integer array Zh [low1c1: up1c1];
integer Zi ,Zk ,Zg ,Zf ,fnn; fnn:= gmn;
Zf := Zg := NULL;
DE(Zf ,RN(3.13));DEVAL(Zg ,RN(3.14));
for i1:= low1c1 step 1 until up1c1 do
begin DE(Zh [i1],NULL) end;
if 2= 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(Zf ),P(Zg ,V(Zh [3]))))););
ASSIGN(Zf ,EL(7 ,STRING(st(66051,st(263430,st(1800,0))),3)));
Zi :=LENGTH(STRING(st(66051,st(263430,st(1800,0))),3));
Zr :=date +V(Zf )+abs(Zg );
ASSIGN(Zf ,STORE ROW(128,10,Zi ,5,Zi +5));
ASSIGN(Zf ,STORE ROW(96,10,Zi ,5,S(IN(Zi ),IN(5))));
ASSIGN(Zf ,D(S(PL(V(Zf ))),P(Zg ,RN(3.14 ))),P(V(Zf ),E(D(S(MI(RN(Zr ))),
Q(RN(3140-2),V(Zf ))),V(Zf ))),S(V(Zh [5]),IQ(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 smn in the block heading and the introduction of "ERASE(snn)" in the translation of a label.

7- 88

Example 2.1.

begin formula f:= 1,g= 2; f:= f + g end ;

Translation 2.1.

```
begin integer Zg, Zf, fmn;  
  fmn:= gmn; lnr(2); Zf:= Zg:= NULL; DE(Zf, IN(1)); DEVAL(Zg, IN(2));  
  ASSIGN(Zf, S(V(Zf), Zg)); ERASE(fmn)  
end
```

Example 2.2.

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

Translation 2.2.

```
begin integer Zf, fmn, smn;  
  fmn:= gmn; lnr(2); Zf:= NULL; DE(Zf, NULL); smn:= gmn;  
Z1: ERASE(smn); ASSIGN(Zf, IN(5));  
  begin integer Zg, fmn;  
    fmn:= gmn; lnr(2); Zg:= NULL; DE(Zg, NULL); go to Z1; ERASE(fmn)  
  end;  
  ERASE(fmn)  
end
```

Example 2.3.

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

Translation 2.3.

```
begin integer Zf, fmn;  
  fmn:= gmn; lnr(2); Zf:= NULL; DE(Zf, NULL);  
Z1: ASSIGN(Zf, IN(5));  
  begin integer Zg, fmn;
```

```

    fnn:= gnn; lnr(2); Zg:= NULL; DE(Zg, NULL); ASSIGN(Zg, V(Zf));
    ERASE(fnn)
  end;
  ERASE(fnn)
end

```

Example 2.4.

```

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

```

Translation 2.4.

```

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 Zh;
    Zh: ; ERASE(fnn)
  end;
  ERASE(fnn)
end

```

Example 2.5.

```

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

```

Translation 2.5.

```

begin integer Zf, fnn;
  fnn:= gnn; lnr(2); Zf:= NULL; DE(Zf, NULL);
  Zl: ASSIGN(Zf, IN(5));
  begin real Zr;
    lnr(2); go to Zl
  end;
  ERASE(fnn)
end

```

Example 2.6.

7- 90

```
begin formula f; 1: f:= 5;  
  begin real r1;  
    begin formula g;  
      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, smn;  
  fmn:= gmn; lnr(2); Zf:= NULL; DE(Zf, NULL); smn:= gmn;  
Z1: ERASE(smn); ASSIGN(Zf, IN(5));  
  begin real Zr1;  
    lnr(3);  
  begin integer Zg, fmn;  
    fmn:= gmn; 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;  
  1: f:= 5; p(1)  
end ;
```


Translation 2.7.begin

```

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

```

```

integer Zf, fnn, snn;
  fnn:= gmn; lnr(2); Zf:= NULL; DE(Zf, NULL); snn:= gmn;
Z1: ERASE(snn); 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 ;

```

Translation 2.8.begin

```

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

```

```

integer Zf, fnn;
  fnn:= gmn; 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.

7- 92

Example 3.1.

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

Translation 3.1.

```
begin  
  
  integer procedure Zp1(Zf, Zg); integer Zf, Zg;  
  begin integer low1c1, up1c1, i1;  
    integer Yf, fnn;  
    fnn:= gn; lnr(4); DEVAL(Yf, Zf); low1c1:= 1; up1c1:= 10;  
    begin integer array Zfa[low1c1:up1c1];  
      integer Yp1, Zy, Zx, fnn;  
      fnn:= gn; if protect then ERR(↓protection 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(fnn); if Yp1 = NULL then ERR(  
        ↓no assignment to proc ident↓); Zp1:= Yp1; protect:= false  
      end;  
      ERASE(fnn)  
    end;  
  
  lnr(2);  
end
```

Example 3.2.

```
begin procedure p1(f,g); value f; formula f,g;  
  begin formula x,y;; end ;  
end ;
```

Translation 3.2.begin

```

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

```

```

  lnr(2);

```

end7.4. 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,d); d(b,b,c,d);
    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,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 ;

```

Translation 4.1.

```

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

```

```

begin

  procedure ZP(Za, Zb, Zc, Zd); integer Za, Zb; integer array Zc;
  procedure Zd;
  begin integer Ya, fmn;
    fmn:= gmn; lnr(4); DEVAL(Ya, Za); Zd(Ya, Zb, Zc, Zd);
    Zd(S(Ya, VN(Zb)), Zb, Zc, Zd); Zd(VN(Zb), Zb, Zc, Zd);
    Zd(V(Zh[1]), Zh[2], Zh, ZP);
    Zd(V(Zh[1]), S(V(Zh[1]), V(Zh[2])), Zh, ZP);
    Zd(S(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:= gmn; 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;
  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 ;
```

Translation 5.1.

```
begin

  integer procedure Zp10;
  begin integer Yp10, Zf, fnn;
    fnn:= gnn; lnr(3); if protect then ERR(
    †protection 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(fnn); if Yp10 = NULL then ERR(†no assignment to proc ident†);
    Zp10:= 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 ;
```

Translation 5.2.

7- 96

begin

```
integer procedure S(Za, Zb); integer Za, Zb;  
begin integer YS, Yb, Ya, fmn;  
  fmn:= gmn; lnr(3); if protect then ERR(  
  †protection error in form proc†); YS:= NULL; DEVAL(Ya, Za);  
  DEVAL(Yb, Zb); YS:= abs(Ya); protect:= true;  
Z1: ; ERASE(fmn); 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: f:= 5 else 2: p14:= 5;  
  for i:= 1 do 3: f:= 5;  
  begin formula g; goto 4 end  
end ;
```

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;
```

```

integer Zi, Zf, fmn, smn;
fmn:= gmn; lnr(2); Zf:= NULL; DE(Zf, NULL); smn:= gmn;
for Zi:= 1 do
3:
begin ERASE(smn); ASSIGN(Zf, IN(5)) end;
begin integer Zg, fmn;
  fmn:= gmn; lnr(6); Zg:= NULL; DE(Zg, NULL); go to 4; ERASE(fmn)
end;
ERASE(fmn)
end

```

Example 5.4.

```

begin formula procedure p7;
  begin formula procedure p8;
    begin formula procedure p9;
      begin f:= 5; p9:= f:= p8:= h[1 + lhs of h[3]]:=
        p7:= h[3 + e1 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 ;

```

Translation 5.4.

```

begin integer low1c1, up1c1, i1;

```

7- 98

```
integer fmn;
fnn:= gnn; lnr(2); lowlc1:= 1; uplc1:= 10;
begin

integer procedure Zp7;
begin

integer procedure Zp8;
begin

integer procedure Zp9;
begin integer Yp9;
  lnr(5); if protect then ERR(↓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 + AREL(1,V(Zh[4])) + LENGTH(V(Zh[5]))]), V(Zh[4]))));
  protect:= true; ASSIGN(Zf, IN(5)); if Yp9 = NULL then ERR(
  ↓no assignment to proc ident↓); Zp9:= Yp9
end;

  lnr(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;
lnr(3); if protect then ERR(↓protection error in form proc↓);
Yp7:= NULL;
for Zi:= 1 do
begin Yp7:= IN(5); protect:= true end;
if Yp7 = NULL then ERR(↓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 Zh[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);

```

7- 100

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

Translation 6.1.

```
begin  
  
integer procedure Zp1;  
begin integer Yp1, Zf, fmn;  
  fmn:= gmn; lnr(4); if protect then ERR(  
  †protection error in form proc†); Yp1:= NULL; Zf:= NULL;  
  DE(Zf, S(P(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(fmn); if Yp1 = NULL then ERR(†no 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(†protection 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 Zp5(Zp6, Zg);
integer procedure Zp6; integer Zg;
begin lnr(18);
    Yp3:= if abs(Zp6) = V(Zx) then abs(VN(Zg)) else if abs(Zp6) =
    V(Zy) then abs(Zp6) else S(VN(Zg), Zp6); protect:= true;
    ASSIGN(Zg, if abs(Zp6) = V(Zx) then VN(Zg) else if abs(Zp6) =
    V(Zy) then Zp6 else S(VN(Zg), Zp6))
end;

    lnr(16); Zp5(Zp4, Zx)
end;
if Yp3 = NULL then ERR(4no assignment to proc ident4); Zp3:= Yp3;
protect:= false
end;

integer Zy, Zx, fnn;
fnn:= gnn; lnr(2); Zx:= Zy:= NULL; DE(Zx, NULL); DE(Zy, NULL); ;
ERASE(fnn)
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(†abc†de†);
end ;

```

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,fnm;
    fnm:= 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(†abc†de†); ; ; ERASE(fnm)
    end;
    ERASE(fnn)
  end

begin lnr(0); end

```

```

1 BEGIN FORMULA F,G)
2 FORMULA ARRAY FA,FG[1:2]
3 F:= (F*(F+FA[(G+FG(F*(G+F)]+
*****ERROR***** 3 SYNT UNIT NOT OK } ]
SYNT UNIT SHOULD BE:)
F
*****ERROR***** 3 SYNT UNIT NOT OK } ]
SYNT UNIT SHOULD BE:)
G(1)
4
*****ERROR***** 3 SYNT UNIT NOT OK } ]
SYNT UNIT SHOULD BE:)
G!
*****ERROR***** 3 SYNT UNIT NOT OK } "IDENTIFIER"
SYNT UNIT SHOULD BE:)
=F) REPLACE(F,1,G)
5 G:= F+G)*F
*****ERROR***** 5 STATEMENT NOT PROPERLY CLOSED } *
)PRINTTEXT({ABCDE
*****ERROR***** 5 PROGRAM TEXT CONTAINS 1 ( (
;

CONTENTS OF INF LIST
1 +10 2 -140 3 +82 4 +49 5 +0 6 +0 7 +0 8 +0 9 +0
10 +0
11 +7 12 +1 13 +12001 14 +4736 15 +0 16 +3 17 +0
18 +10 19 -140 20 +46 21 +0 22 +0 23 +40 24 +0 25 +0 26 +0
27 +32
28 +6 29 +2 30 +16001 31 +4882435 32 +0 33 +0
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 +79 52 +0 53 +0 54 +64 55 +69 56 +1 57 +1
58 +32
59 +5 60 +2 61 +8001 62 +4882434 63 +0
64 +5 65 +4 66 +12001 67 +4882434 68 +59
69 +6 70 +5 71 +22002 72 +4887104 73 +0 74 +1
75 +4 76 +6 77 +26002 78 +4887104
79 +3 80 -141 81 +1
82 +3 83 -141 84 +0

5000 REPLACE
4995 0 1
4995 F 0 4991 2
4991 LEFT
4987 4975 3
4987 G
4983 0 4
4983 FA
0 4979 5

```

```

4979 FG      0 0 6
4975 PRINTTEXT 0 0 7
1 5000      2 4995      3 4991      4 4987      5 4983      6 4979      7 4975

BEGIN
  INTEGER LOWIC1,UPIC1,11;
  INTEGER FNN; FNN:= GNN;LNR(1);
  LOWIC1:= 1;UPIC1:= 2;
*****ERROR*****      2 SYNT UNIT NOT OK      ]      "IDENTIFIER"
SYNT UNIT SHOULD BE:]

  BEGIN
  INTEGER ABBAY RFA ,RFG (LOWIC1,UPIC1
*****ERROR*****      2 SYNT UNIT NOT OK      ]      "IDENTIFIER"
SYNT UNIT SHOULD BE:]

  INTEGER RG ,RF ,FNN; FNN:= GNN;
  RF := RG := NUWU;
  DE(RF ,NUWU);DE(RG ,NUWU);
  EQB 11:= LOWIC1 STEP 1 UNTIL UPIC1 DO
  BEGIN DE(RFA [1],NUWU);DE(RFG [1],NUWU) END;ADD+GN(RF ,
*****ERROR*****      3 SYNT UNIT NOT OK      ]      "IDENTIFIER"
SYNT UNIT SHOULD BE:]

*****ERROR*****      3 SYNT UNIT NOT OK      ]      "IDENTIFIER"
SYNT UNIT SHOULD BE:]
  R(V(RF ),R(R(V(RF ),V(RFA ((V(RG )+V(RFG (V(RF )+(V(RG )+V(RF )
*****ERROR*****      3 SYNT UNIT NOT OK      ]      ]
SYNT UNIT SHOULD BE:]
  ]))
*****ERROR*****      3 SYNT UNIT NOT OK      ]      *
SYNT UNIT SHOULD BE:]
  V(RFG [1
  ]))
*****ERROR*****      3 SYNT UNIT NOT OK      ]      ]
SYNT UNIT SHOULD BE:]
  ]))];
ADD+GN(RG ,V(RF ));REPLACE (V(RF ),1,V(RG ));
ADD+GN(RG ,5(V(RF ),V(RG )))
*****ERROR*****      5 STATEMENT NOT PROPERLY CLOSED      ]      *
  PRINTTEXT ({ABC+DE});

```

*****ERROR***** 5 SYNT UNIT NOT OK

SYNT UNIT SHOULD BE 1

1 ERABE(FNN) 711

END

1 ERABE(FNN) ENDTIME 1.35 MH

}
*****ERROR***** 0 PROGRAM DOES NOT BEGIN WITH BEGIN "UNKNOWN" >

1 -
*****ERROR***** 0 STATEMENT NOT PROPERLY CLOSED >)
END

CONTENTS OF INF LIST

1	+10	2	-140	3	+62	4	+49	5	+0	6	+0	7	+0	8	+0	9	+0
10	+0																
11	+7	12	+1	13	+12001	14	+4736	15	+0	16	+3	17	+0				
18	+10	19	-140	20	+46	21	+0	22	+0	23	+40	24	+0	25	+0	26	+0
27	+32																
28	+6	29	+2	30	+16001	31	+4882435	32	+0	33	+0						
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	+59	52	+0	53	+0	54	+0	55	+0	56	+0	57	+0
58	+0																
59	+3	60	-141	61	+1												
62	+3	63	-141	64	+0												

5000 REPLACE
4995 0 1
4995 F 0 4991 2
4991 LEFT
4987 0 3
4987 G 0 4
1 5000 2 4995 3 4991 4 4987

*****ERROR***** 0 PROGRAM DOES NOT BEGIN WITH BEGIN "UNKNOWN" >
BEGIN LNR(0)
*****ERROR***** 0 STATEMENT NOT PROPERLY CLOSED >)

END TIME IS: .33 MM

Example 7.2.

```
begin formula a,b,c;
  a:= lhs b;
  if true then c:= rhs a
end ;
```

Translation 7.2.

```
begin integer Zc, Zb, Za, fmn;
  fmn:= gmn; lnr(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
```

Example 7.3.

```
begin integer procedure p(a); p:= 1;
  integer procedure p1(a)p1:=p(b);p1(1)
end ;
```

Translation 7.3.

```
begin

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

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

  lnr(2); Zp1(1)
end
```

7- 108

Example 7.4.

```
begin procedure p(a);;  
  procedure p1(a):p(a);p1(1)  
end ;
```

Translation 7.4.

begin

```
  procedure Zp(Za);  
  begin lnr(1); end;
```

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

```
  lnr(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: "Fl", "Ft" and "Fr", 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 $399 \times 400 + 399 = 159999$ different identifiers possible if the size of code (400) were the only restriction.

The integer capacity of the EL - X8 is $2 \uparrow 26$; the code size 400 is therefore, optimal. If the ALGOL 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):

ptr to first letgit := ptr to text \times 2000 + 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 MC-ALGOL 60 standard procedures

We now give an alphabetic list of the headings of the standard procedures used in the ABC ALGOL 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 as effect, if $n \geq 0$, 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:= $(-1)^{\uparrow n}$.

procedure EXIT; Discontinues the execution of the ALGOL program.

procedure FLXT(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 integral 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 ALGOL is used which is given here in a super condensed form:

character	code	other characters
digit 0 - 9	0 - 9	+ - × / ↑ = < > ¬ ↓ √ ^
letters a - z	10 - 35	, . 10 : ; () [] ' " ? _
letters A - Z	37 - 62	Space New Line
		with codes: 64 - 127

Note that, during the initialization, the ABC ALGOL 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 - Flexowriter 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 - Flexowriter 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) st 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 x;

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 MC-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 be 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".

.PR 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 ABC ALGOL 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 lhs 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:

$$\text{synt unit1} \times 2 \uparrow 15 + \text{synt unit2} \times 2 \uparrow 5 + \text{mode}.$$

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

- [1] ACM SIGSAM symposium II, Proceedings.
Comm ACM 14 nr 8. (August 1971)
Journal ACM 18 nr 4. (October 1971)

- [2] Dahl, O., K. Nygaard: SIMULA - An ALGOL based simulation language.
Comm ACM 9 nr 9. pp 671-682 (Sept. 1966)

- [3] Geurts, L., L. Meertens, G. Nogarede, M. Rem, R.P. van de Riet:
The MC ELAN assembler. Report MR 132/72 (April 1972)
Mathematisch Centrum, Amsterdam

- [4] Grune, D.: Handleiding bij de tekstschaaf.
Report LR 2.5
Mathematisch Centrum, Amsterdam

- [5] - - -: Handleiding milli-systeem voor de EL X8.
Report LR 1.1
Mathematisch Centrum, Amsterdam

- [6] Hoare, C.A.R.: Proof of a program: FIND.
Comm ACM 14 nr 1. pp 39-45 (Jan 1971)

- [7] Knuth, D.: Fundamental Algorithms.
Addison-Wesley (1968)

- [8] - - -: Seminumerical Algorithms.
Addison-Wesley (1969)

- [9] Mey, G. van der: Over recursieve processen in circulerende lijststelsels.
Informatie 10 nr 12, pp 285-291.

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

Numerische Mathematik 4, pp 420-453 (1963)

- [11] Riet, R.P. van de: Formula manipulation in ALGOL 60 part 1.
MC TRACTS 17 (2nd edition)
Mathematisch Centrum, Amsterdam (1970)

- [12] - - -: Formula manipulation in ALGOL 60 part 2.
MC TRACTS 18
Mathematisch Centrum, Amsterdam (1968)

- [13] - - -: Garbage-collection methods for ABC in ALGOL 60.
Report TW 110
Mathematisch Centrum, Amsterdam (1969)

- [14] Roever, W.P. de: An exact rational function system with
garbage collection in ALGOL 60.
Report MR 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.
Prentice-Hall, Englewood Cliffs, N.J. (1969)

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

- [17] Wirth, N., C.A.R. Hoare: A contribution to the development
of ALGOL.
Comm ACM 9, pp 413-431 (1966)

- [18] Wijngaarden, A. van (Editor), B.J. Mailloux, J.E.L. Peck,
C.H.A. Koster: Report on the algorithmic language
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 1 T. VAN DER WALT, *Fixed and almost fixed points*, 1963. ISBN 90 6196 002 9.
- MCT 2 A.R. BLOEMENA, *Sampling from a graph*, 1964. ISBN 90 6196 003 7.
- MCT 3 G. DE LEVE, *Generalized Markovian decision processes, part I: Model and method*, 1964. ISBN 90 6196 004 5.
- MCT 4 G. DE LEVE, *Generalized Markovian decision processes, part II: Probabilistic background*, 1964. ISBN 90 6196 006 1.
- MCT 5 G. DE LEVE, H.C. TIJMS & P.J. WEEDA, *Generalized Markovian decision processes, Applications*, 1970. ISBN 90 6196 051 7.
- MCT 6 M.A. MAURICE, *Compact ordered spaces*, 1964. ISBN 90 6196 006 1.
- MCT 7 W.R. VAN ZWET, *Convex transformations of random variables*, 1964. ISBN 90 6196 007 X.
- MCT 8 J.A. ZONNEVELD, *Automatic numerical integration*, 1964. ISBN 90 6196 008 8.
- MCT 9 P.C. BAAYEN, *Universal morphisms*, 1964. ISBN 90 6196 009 6.
- MCT 10 E.M. DE JAGER, *Applications of distributions in mathematical physics*, 1964. ISBN 90 6196 010 X.
- MCT 11 A.B. PAALMAN-DE MIRANDA, *Topological semigroups*, 1964. ISBN 90 6196 011 8.
- MCT 12 J.A.TH.M. VAN BERCKEL, H. BRANDT CORSTIUS, R.J. MOKKEN & A. VAN WIJNGAARDEN, *Formal properties of newspaper Dutch*, 1965. ISBN 90 6196 013 4.
- MCT 13 H.A. LAUWERIER, *Asymptotic expansions*, 1966, out of print; replaced by MCT 54.
- MCT 14 H.A. LAUWERIER, *Calculus of variations in mathematical physics*, 1966. ISBN 90 6196 020 7.
- MCT 15 R. DOORNBOS, *Slippage tests*, 1966. ISBN 90 6196 021 5.
- MCT 16 J.W. DE BAKKER, *Formal definition of programming languages with an application to the definition of ALGOL 60*, 1967. ISBN 90 6196 022 3.
- MCT 17 R.P. VAN DE RIET, *Formula manipulation in ALGOL 60, part 1*, 1968. ISBN 90 6196 025 8.
- MCT 18 R.P. VAN DE RIET, *Formula manipulation in ALGOL 60, part 2*, 1968. ISBN 90 6196 038 X.
- MCT 19 J. VAN DER SLOT, *Some properties related to compactness*, 1968. ISBN 90 6196 026 6.
- MCT 20 P.J. VAN DER HOUWEN, *Finite difference methods for solving partial differential equations*, 1968. ISBN 90 6196 027 4.

- MCT 21 E. WATTEL, *The compactness operator in set theory and topology*, 1968. ISBN 90 6196 028 2.
- MCT 22 T.J. DEKKER, *ALGOL 60 procedures in numerical algebra, part 1*, 1968. ISBN 90 6196 029 0.
- MCT 23 T.J. DEKKER & W. HOFFMANN, *ALGOL 60 procedures in numerical algebra, part 2*, 1968. ISBN 90 6196 030 4.
- MCT 24 J.W. DE BAKKER, *Recursive procedures*, 1971. ISBN 90 6196 060 6.
- MCT 25 E.R. PAERL, *Representations of the Lorentz group and projective geometry*, 1969. ISBN 90 6196 039 8.
- MCT 26 EUROPEAN MEETING 1968, *Selected statistical papers, part I*, 1968. ISBN 90 6196 031 2.
- MCT 27 EUROPEAN MEETING 1968, *Selected statistical papers, part II*, 1969. ISBN 90 6196 040 1.
- MCT 28 J. OOSTERHOFF, *Combination of one-sided statistical tests*, 1969. ISBN 90 6196 041 X.
- MCT 29 J. VERHOEFF, *Error detecting decimal codes*, 1969. ISBN 90 6196 042 8.
- MCT 30 H. BRANDT CORSTIUS, *Excercises in computational linguistics*, 1970. ISBN 90 6196 052 5.
- MCT 31 W. MOLENAAR, *Approximations to the Poisson, binomial and hypergeometric distribution functions*, 1970. ISBN 90 6196 053 3.
- MCT 32 L. DE HAAN, *On regular variation and its application to the weak convergence of sample extremes*, 1970. ISBN 90 6196 054 1.
- MCT 33 F.W. STEUTEL, *Preservation of infinite divisibility under mixing and related topics*, 1970. ISBN 90 6196 061 4.
- MCT 34 I. JUHÁSZ, A. VERBEEK & N.S. KROONENBERG, *Cardinal functions in topology*, 1971. ISBN 90 6196 062 2.
- MCT 35 M.H. VAN EMDEN, *An analysis of complexity*, 1971. ISBN 90 6196 063 0.
- MCT 36 J. GRASMAN, *On the birth of boundary layers*, 1971. ISBN 90 6196 064 9.
- MCT 37 J.W. DE BAKKER, G.A. BLAAUW, A.J.W. DUIJVESTIJN, E.W. DIJKSTRA, P.J. VAN DER HOUWEN, G.A.M. KAMSTEEG-KEMPER, F.E.J. KRUSEMAN ARETZ, W.L. VAN DER POEL, J.P. SCHAAP-KRUSEMAN, M.V. WILKES & G. ZOUTENDIJK, *MC-25 Informatica Symposium*, 1971. ISBN 90 6196 065 7.
- MCT 38 W.A. VERLOREN VAN THEMAAT, *Automatic analysis of Dutch compound words*, 1971. ISBN 90 6196 073 8.
- MCT 39 H. BAVINCK, *Jacobi series and approximation*, 1972. ISBN 90 6196 074 6.
- MCT 40 H.C. TIJMS, *Analysis of (s,S) inventory models*, 1972. ISBN 90 6196 075 4.
- MCT 41 A. VERBEEK, *Superextensions of topological spaces*, 1972. ISBN 90 6196 076 2.
- MCT 42 W. VERVAAT, *Success epochs in Bernoulli trials (with applications in number theory)*, 1972. ISBN 90 6196 077 0.
- MCT 43 F.H. RUYMGAART, *Asymptotic theory of rank tests for independence*, 1973. ISBN 90 6196 081 9.
- MCT 44 H. BART, *Meromorphic operator valued functions*, 1973. ISBN 90 6196 082 7.

- MCT 45 A.A. BALKEMA, *Monotone transformations and limit laws*, 1973. ISBN 90 6196 083 5.
- MCT 46 R.P. VAN DE RIET, *ABC ALGOL, A portable language for formula manipulation systems, part 1: The language*, 1973. ISBN 90 6196 084 3.
- MCT 47 R.P. VAN DE RIET, *ABC ALGOL A portable language for formula manipulation systems part 2: The compiler*, 1973. ISBN 90 6196 085 1.
- MCT 48 F.E.J. KRUSEMAN ARETZ, P.J.W. TEN HAGEN & H.L. OUDSHOORN, *In ALGOL 60 compiler in ALGOL 60, Text of the MC-compiler for the EL-X8*, 1973. ISBN 90 6196 086 X.
- MCT 49 H. KOK, *Connected orderable spaces*, 1974. ISBN 90 6196 088 6.
- MCT 50 A. VAN WIJNGAARDEN, B.J. MAILLOUX, J.E.L. PECK, C.H.A. KOSTER, M. SINTZOFF, C.H. LINDSEY, L.G.L.T. MEERTENS & R.G. FISHER (eds.), *Revised report on the algorithmic language ALGOL 68*. ISBN 90 6196 089 4.
- MCT 51 A. HORDIJK, *Dynamic programming and Markov potential theory*, 1974. ISBN 90 6196 095 9.
- MCT 52 P.C. BAAYEN (ed.), *Topological structures*, 1974. ISBN 90 6196 096 7.
- MCT 53 M.J. FABER, *Metrizability in generalized ordered spaces*, 1974. ISBN 90 6196 097 5.
- MCT 54 H.A. LAUWERIER, *Asymptotic analysis, part 1*, 1974. ISBN 90 6196 098 3.
- MCT 55 M. HALL JR. & J.H. VAN LINT (eds.), *Combinatorics, part 1: Theory of designs finite geometry and coding theory*, 1974. ISBN 90 6196 099 1.
- 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 57 M. HALL JR. & J.H. VAN LINT (eds.), *Combinatorics, part 3: Combinatorial group theory*, 1974. ISBN 90 6196 101 7.
- MCT 58 W. ALBERS, *Asymptotic expansions and the deficiency concept in statistics*, 1975. ISBN 90 6196 102 5.
- MCT 59 J.L. MIJNHEER, *Sample path properties of stable processes*, 1975. ISBN 90 6196 107 6.
- MCT 60 F. GÖBEL, *Queueing models involving buffers*. ISBN 90 6196 108 4.
- * MCT 61 P. VAN EMDE BOAS, *Abstract resource-bound classes, part 1*. ISBN 90 6196 109 2.
- * MCT 62 P. VAN EMDE BOAS, *Abstract resource-bound classes, part 2*. ISBN 90 6196 110 6.
- MCT 63 J.W. DE BAKKER (ed.), *Foundations of computer science*, 1975. ISBN 90 6196 111 4.
- MCT 64 W.J. DE SCHIPPER, *Symmetries closed categories*, 1975. ISBN 90 6196 112 2.
- MCT 65 J. DE VRIES, *Topological transformation groups 1 A categorical approach*, 1975. ISBN 90 6196 113 0.
- * MCT 66 H.G.J. PIJLS, *Locally convex algebras in spectral theory and eigenfunction expansions*. ISBN 90 6196 114 9.

- * MCT 67 H.A. LAUWERIER, *Asymptotic analysis, part 2.*
ISBN 90 6196 119 X.
- * MCT 68 P.P.N. DE GROEN, *Singularly perturbed differential operators of second order.* ISBN 90 6196 120 3.
- * MCT 69 J.K. LENSTRA, *Sequencing by enumerative methods.*
ISBN 90 6196 125 4.
- MCT 70 W.P. DE ROEVER JR., *Recursive program schemes: semantics and proof theory.* ISBN 90 6196 127 0.
- * MCT 71 J.A.E.E. VAN NUNEN, *Contracting Markov decision processes.*
ISBN 90 6196 129 7.
- * MCT 72 J.K.M. JANSEN, *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.
- * MCT 74 H.J.J. TE RIELE, *A theoretical and computational study of generalized aliquot sequences.* ISBN 90 6196 131 9.

An asterisk before the number means "to appear".