

Algol : Main.

Assume Procedures without formal parameters

a) Require or global accepted fully.

Then

Procedures without in- or output or both.

```
procedure A(x) =: (y), B(z), C =: (t), D;  
require (k, l, m)  
begin A: y := x + k; return;  
      B: l := z; l := m; B := m; return;  
      C: t := k + m; return;  
      D:
```

Proposal:

A procedure Q may be used in an expression with the meaning of a function if, and only if, it is not followed by in the defining procedure declaration

a) it is not followed by $=:$
b) an assignment statement $Q := E$ occurs in the procedure program.

A =:
B =: (x)
C =: (x, y)
D

procedure $A = :$, $B(x, y) = :$, $C = : (z, t)$, $D(u, v) = : (s, r)$,
 E , $F(p, q)$;
require (l, m)
begin $A : \Sigma^{(l, m)} ; \text{return} ; \Sigma(l, m) = : (l, m)$;
 $B : \Sigma(x, y, l, m) = : (l, m) ; \text{return} ;$
 $C : \Sigma(l, m) = : (l, m, z, t) ; \text{return} ;$
 $D : \Sigma(u, v, l, m) = : (l, m, s, r) ; \text{return} ;$
 $E : \Sigma(l, m) = : (l, m, E) ; E := ; \text{return} ;$
 $F : \Sigma(p, q, l, m) = : (l, m, F) ; F := ; \text{return} ;$
end

Possible procedure statements:

$A = :$;
 $B(x, y) = :$;
 $C = : (z, t) ;$
 $D(u, v) = : (s, r) ;$

E and $F(p, q)$ may
 only be used in expression

function $G := \underline{E}$

function $H(v, u) := \Sigma(v, u)$

All this will necessitate that no simple
 variable has the same identifier as the
 procedure of type E or function of
 type G . For consistency reasons it would

be natural to suggest that
an identifier identifies completely.

However it should be noted that this
arises incidentally from the circumstance
that ~~we have~~ our notation for
 $1, 2, \dots$ formal parameters ~~cannot~~ does
not point to a symbolism for 0 para-
meters. $(P)(, P(), P(), \dots)??$

Algol, Main 2.

Is Dummy = Formal? No.

Formal variables are distinguishable from the context. They may occur only in function and procedure declarations.

A(,

A(Y, Z[,], X(,

```
procedure A(y, z[k], x(a, b[c], d(e, f) =: (g)) =: (h[, ], i:))  
  integer(  
    array(  
      begin integer(k, l)  
        s := z[k, l] etc  
      end
```

Problem 1. Type declarations to functions (defined in a function declaration.)

Problem 2. $AT[e, d] = A[d, e]$. Renaming, several names for one quantity.

Problem 3. Type declarations in

The prefix idea probably will necessitate that a compound statement may only go to itself, and may not be gone to from outside. This may not be desirable.

Two types of compound statements

A) Time = write up

B) One + write up.

A) 1) Special marking of jump out (return)
2) No jump into.

Algol in Mainz doc. 59.

Problem complex 1.

Identification of objects.

Identifiers

Bracket structure

Dummy variables

Syntactical questions:

Declarations

For - if -

Prefixes

Compound statements.

So far 3 kinds: For, if, procedures.

Dynamic array declarations.

Special case of ~~renaming~~

Questions (using Samelson's notation 1B 7.22):

array (A [1: 10]); begin Σ_1 end;

Σ_2 (now A[] is non-existing)

array (A [7: 10]); begin Σ_3 end;

Even if Σ_2 is empty A [1: 10] is lost the moment the first end is encountered. Thus the first and second A have no relation whatever.

It seems to me:

1) Change of size must imply that the former array disappears — unless specific rules are given.

... array A [1: 5] ...) what is their mutual relation.
~~array~~ A [10: 15]

2) Same holds generally.

integer (a)

complex (a)

Does this mean $\text{Re}(a) := a$
 $\text{Im}(a) := ?$

double length is more obvious.

Ident

Declared identifier is a container. If you change its size what happens to the contents ??

Dynamic declarations continued.

Double length hardware problem.

P: Σ
~~if~~ $(a-b \geq \epsilon)$; goto P;
~~if~~ $(a-b > \epsilon \text{ max})$; begin double length (a, b); goto P;

7.37 Bottenbruch suggests range limiting by time succession (double length example)

7.31 van Wijngaarden, Dijkstra suggest write-up limiting defined by contradictory declaration

7.16 Samuelson & Reithmann suggest write-up limiting by compound statement.

7.22 limiting by compound statement.

Mandag worgm.

Bauer: Compound statements, general disjunction

Internal means: defined and used
introduced
variables $i :=$

function
proc
label

function
procedure
like: Σ

$i = i$
 $i \geq i$
} used
go to

Samelson: New, old introduces new bracketing
structure. Prefers to use the existing
brackets begin end.

Van Wijnen wants new (a) new (b) old (a) old (b)

Samelson, this introduces any number of brackets

Bauer:

Proposal

L; Δ begin Σ end

For compound statement; and procedure

- 1) local (a, b, c)
 - 2) global (a, b, c)
 - 3) "nothing" implies everything global
 - 4) local " " everything local
- } are permissible

To be understood in the write-up sense.

- 2) In connection with a compound statement any parameter which is local (by a "local" - declaration) or implicitly (through a "global" declaration) may be declared.

Accumulation of declarations is compulsory.

- 3) The program is between begin and end. Universal declarations are written in front of begin. Local parameters ~~are~~ ~~outside~~ ~~begin~~ cannot be expected to ~~and~~ survive after the compound has been left.

4. All declarations must appear in the above described way. (in above situation).

The local declaration.

variable (a), label (L), type (),

function f(x)

switch Q := ()

procedure

array

The global declaration.

global ()

global is special.

One particular entity may be mentioned in one normal declaration and any number of global declarations.

The range of ~~meaning~~ (~~existence~~) identity of one particular entity (characterized by an identifier I and possibly a declarator) is normally one statement (ordinarily a compound statement)

governed by the structure of the program as determined

1. Identification, symbolization
Declarations
New types of program section
Procedure headings
Dummy
2. For.. statement
3. If.. and alternative.
4. Types of expressions
Compatibility Arithmetic

A procedure may be constructed as follows:

1) Take any compound statement including the declarations.

2) Add a heading with the following contents:

a) The delimiter "procedure".

b) The procedure name (only one)

c) Input output list.

d) Declarations for input output list.

The compound statement then gives:

e) Global things

f) Declarations for local things.

g) begin Σ end.

ad d): ~~Parameters~~ in Expressions ~~may be~~ appearing

in parameter lists in any declarations may be

- 1) ^{Simple} formal variables (not appearing in
- 2) expressions in formal input parameters
- 3) ——— global ———

procedure Pip (a, b, c, d, e) =: (f, g, h);
integer (b), array (c [1 : b+1]), procedure d(i, j) =: (k, l),
label (f); array (i [p, q]), integer (i), label (k),
array (l)

Functions and procedure functions.

- ~~Delete function completely~~
- ~~Remove the possibility of defining procedures by procedure decl.~~
- Call all functions "functions", even when defined by procedure.

Agreement: a procedure
 procedure A(...) =: gives result
 function A() := "function"

 which defines a function will start with the declarator function.

procedure Simpson (F, a, b, delta, V)

function ~~procedure~~ (F(x)) ;

local integer (k, n)

begin Ibar := V * (b-a) ; n := 1 ;
h := (b-a)/2 ; J := h * (F(a) + F(b)) ;

JL : S := 0

for k := 1(1)n ; S := S + F(a + (2*k-1) * h) ;

I := J + 4 * h * S ;

if (delta < abs(I - Ibar)) ;

begin Ibar := I ; J := (I + J)/4 ;
n := 2 * n ; h := h/2 ; go to JL
end ;

Simps := I/3 ;

return & end Simps ;

Heading ~~Call~~ Call

P(a, b) =: (c, d)
P(a, b) =: (-)
P(-) =: (c, d)
P(-) =: (-)

P
P

~~P(A, B) =: (C, D)~~
~~P(A, B) =: (-)~~
~~P(-) =: (C, D)~~

P(A, B) =: (C, D) ;
P(A, B)
P =: (C, D)
P

procedure C

~~procedure statements~~

begin

- C () =: ()

procedure A () ()
global a, B, b, c
begin Σ end

procedure B () ()
global b
begin Σ end

~~global C~~
begin

a := E

b := E

> A () =: ()

└──────────────────┘

C () =: ()

B () =: ()

end

end

Procedure declarations.

declares a program to be

A procedure declaration

Zürich report.

- pag 41. 1. Nothing to do with universal language (Woodger)
2. Overlooks the fact that the many programming languages is due to different problem groups.
Say algorithmic instead of programming.
42. 3. ACM Ad-Hoc
43. 4. delete larger
5. and statement prefixes
- pag 44. 6. meaningless
7. Notion of program.
8. or delimited
9. begin end
10. Type
or comprising the ^{single} letter A alone
44. 10. a) non negative integer (string of figures)
b) integer
45. 11. wording.
12. 123, change description.
13. Description. the computed value
Intention: expressions are to be mathematically integer expr.
46. 14. No heading omitted.
15. Use Backus' description.
47. 16. Complete according to Copeland's list
possibly add for ~~the~~ ~~the~~

Zürich report.

a/b/

47. 17. a (Move to the beginning of section)
17. b. Leave out parentheses.
17. c. Add ⁰ 1 doubtful.
18. Add Boolean
19. a. Notation label appears for the first time.
- b. Remodel
20. Introduce category of statement prefixes
21. and may be followed by any string not containing end or ; (comment)
22. Delete full stop.
23. Must be ~~also~~ defined more closely with arithmetic expressions.
24. Interpretation 1 and 0 is doubtful. cfr. 17c
25. This remark is basic must come earlier.
26. Round parentheses list.
27. Give better example. cfr. subscripts.
28. Change example.
29. Complete revision. of iv and v
30. Kill it.
31. Reword Change word e.g. HALT)

52. 32. See 37.
33. Probably complete revision.
34. $n \geq k$
35. Designational expr. allowed?
36. * up there footnote
37. ; between 35. Explain use of delimiters.
38. Arc () needed for one I? (Lit)
39. Complete revision.
40. Write function
41. Remember definition of integers.
42. Explain the expression allowed
43. Give an example of switch
44. Delete ;
45. Complete revision.
46. Rewrite (perhaps write proceeds from
47. Delimiter after (P₀);
48. Delete whole line.
49. Indicate general form of P().
50. Inconsistency.
51. Woodger maintains trouble.
52. - sign. not =
53. Display alphabet
54. State clearly what line convention means.

"Types" of arithmetic expressions. Compatibility.

Samelson:

Type
real
integer
complex

Declarator

integer
complex

Arithmetics

16. dec.

1. Expressions whose result belongs, according to the rules available to the compiler (and stated in ALGOL report) to an arithmetic A_1 , may be used in right hand side of assignment statements whose left hand side has been declared to belong to any arithmetic A_2 which includes A_1 .

2. These rules should allow combination of variables belonging to different arithmetics.

3. When ~~as~~ the ^{turns out to be by the rules to belong to} result of an expression ~~is~~ an arithmetic A_1 , but is mathematically in an arithmetic A_2 included in and different from A_1 , and this fact is not determinable by these rules available as above the result must be treated as belonging to A_1 .

Rules are given for the type T of an expression E in terms of the types of its constituents, except when E appears in an assignment statement form $y := E$. In this case a current type declaration governing y may determine the type of E , provided it is compatible with the type T which would be given by the rules for E in isolation.

→ It is not excluded that there rules may be such that the result

E_1 E_2	int int	var int	int var	var var	int compl.	compl int	var compl	compl var	and
$E_1 \pm E_2$	i	v	v	v	c	c	c	c	
$E_1 \times E_2$	i	v	v	v	c	c	c	c	
E_1 / E_2	v	v	v	v	c	c	c	c	
$E_1 \uparrow E_2 \downarrow$	v	v	(not possible) only var > 0	v	c	c	c	c	

E_1 E_2	var doub	doub var
$E_1 \pm E_2$	d	d
$E_1 \times E_2$	d	d
E_1 / E_2	d	d
$E_1 \uparrow E_2 \downarrow$	not possible	(not possible) only var > 0

Purpose

Discuss Algol: More than that:

Arrive at our opinion on Algol 1.

Secondarily Algol 2-3....

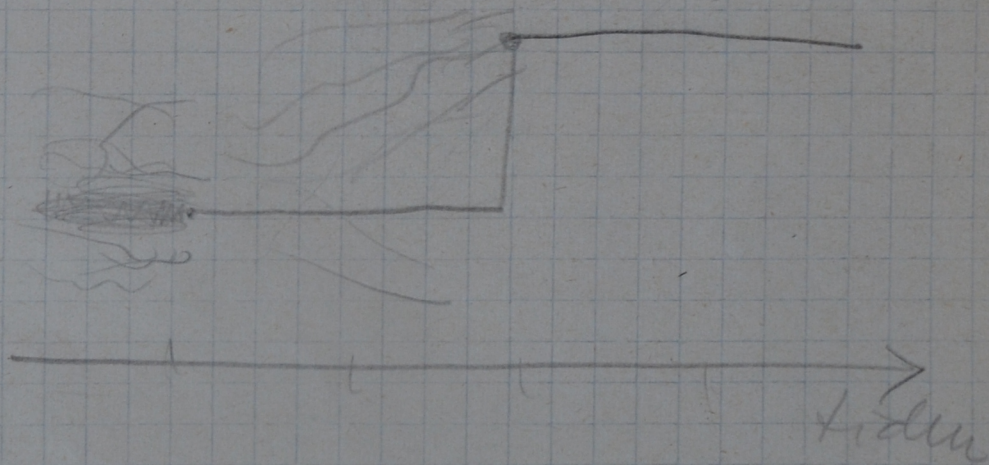
- 1) Complete Zürich report
- 2) Changes
- 3) Extensions.

Thursday afternoon Subcomm. Hec

Friday morning Plenary session

afternoon Subcomm. Hec

Saturday



14-16 Dec in Mainz