

A-6  
2

JUN 20 3 19 PM '79

A B S T R A C T

THE SYNTAX AND SEMANTICS OF THE PROPOSED INTERNATIONAL ALGEBRAIC LANGUAGE OF THE ZURICH ACM-GAMM CONFERENCE

J. W. BACKUS,

International Business Machines Corp., New York, USA

This paper gives a tutorial summary of the syntax and interpretation rules of the proposed international algebraic language put forward by the Zurich ACM-GAMM Conference, followed by a formal, complete presentation of the same information. Notations are presented for numbers, numerical variables, Boolean variables, relations, n-dimensional arrays, functions, operators and algebraic expressions. Means are provided in the language for specifying assignment of values to variables, conditional execution of statements, iterative procedures, formation of compound statements from sequences of statements, definition of new statements for arbitrary procedures, reuse and alteration of program segments.

The proposed language is intended to provide convenient and concise means for expressing virtually all procedures of numerical computation while employing relatively few syntactical rules and statement types.

JUL 05 8 10 AM '50

THE SYNTAX AND SEMANTICS OF THE PROPOSED INTER-  
NATIONAL ALGEBRAIC LANGUAGE OF THE ZURICH ACM-  
GAMM CONFERENCE

J. W. BACKUS,

International Business Machines Corp., New York, USA

General

In May 1958 a conference in Zurich completed a joint project of the ACM (Association for Computing Machinery) and the GAMM (Association for Applied Mathematics and Mechanics) by approving a proposed International Algebraic Language ( IAL ). Other papers to be presented here discuss the history of the project and various aspects of the language. It is the purpose of the present paper to precisely describe both the syntax and semantics of IAL. The description given here will be that of the so-called "reference language" which, for definiteness, employs a specific set of symbols and notational conventions.

It should be kept in mind, however, that the reference language form of IAL exists primarily for the purpose of describing the rules of construction of the language and its meanings. In actual use, a variety of symbolizations and notational conventions are envisaged, each representation being a fairly direct transliteration of the reference language and having the same rules of syntax and semantics. "Hardware representations" will generally use a smaller set of symbols than that employed in the reference language and will be suitable for mechanical translation into machine programs by a given type of machine. On the other hand, the "publication forms" of the language will employ many of the notational conventions of mathematics (e. g., exponents, subscripts, Greek letters) and will be used in human communication of IAL programs.

The description of the reference language is given in two parts. Part I gives a brief informal description of the major elements of the language in terms of examples. Part II gives a formal description of the structure of the language,

## Part I : Informal Description

### General

An IAL program is a sequence of statements which may be interspersed with certain declarations. Each statement describes a rule of computation and, explicitly or implicitly, specifies a successor-statement (the successor being the next statement in the program unless otherwise stated). The computing rule given by a program is, of course, the sequence of computations specified by the statements when taken in the order provided by successor-relations. Declarations state certain facts which are to obtain throughout the computation. Their location in the program is generally unimportant.

The following paragraphs present some of the important properties of expressions and statements, with examples.

### Algebraic Expressions

1. Algebraic expressions are composed of variables, constants, functions, and operators for the usual arithmetic operations.
2. Names of variables, functions and other entities are composed of alphabetic and numeric characters, the first being alphabetic. Any length sequence of characters may be used.
3. It is understood that the arithmetic operations in an expression signify a floating point approximation of real number arithmetic. Some variables may be declared to be integer-valued. The effect of such a declaration is that any value which is to be assigned to such a variable is first rounded to the nearest integer.
4. Array declarations (cf. Declarations below) may state that certain names are the names of arrays of data of various dimensions. These names may appear in expressions as subscripted variables which are followed by a sequence of subscripts enclosed in a pair of square brackets, the number of subscripts corresponding to the dimensionality of the array and each subscript separated from the next by a comma. Since parentheses are used to enclose function arguments (and sub-expressions), it is easy to distinguish between functions and subscripted variables. A subscript may be any expression; the value of the subscript is the integer formed by rounding the value of the expression (if it is not already an integer).

### Examples of Algebraic Expressions

1.  $X^2 + Y^2$

means:  $X^2 + Y^2$

2.  $3.14 \times (\alpha + \sin(x))$

means:  $3.14 \times (\alpha + \sin(x))$

3.  $(b/2 - \sqrt{axc - (b/2)^2})/a$

means:  $(b/2 - \sqrt{axc - (b/2)^2})/a$

4.  $R_{[i+1, j]} \times S_{[i, j+1]}$

means:  $R_{i+1, j} \times S_{i, j+1}$ . In particular, if  $i$  and  $j$  both have the value 1 (or 1.2) when the above expression is encountered, it is then equivalent to:  $R_{2, 1} \times S_{1, 2}$

### Boolean Expressions

Some variables or arrays of quantities may be declared to be Boolean-valued (i. e., having only the value 0, "false", or 1, "true"). These variables, the constants 0 and 1, Boolean-valued functions, and subexpressions of the form:

$$(E \text{ r } E')$$

may be combined with the Boolean operators "and", "or", "not", and "equivalent" to form a Boolean expression. (In the above  $E$  and  $E'$  are algebraic expressions and  $r$  is a relation, e. g.,  $(x/y > 1)$ .)

### Examples of Boolean Expressions

1.  $(X > Y) \vee (Y > Z^2)$

This Boolean expression is true if, and only if, either  $X$  is greater than  $Y$  or  $Y$  is greater than  $Z^2$  or both. The operator is that for "or".

2.  $(\neg A) \wedge B \vee (A \wedge (X = Y + 1))$

Here  $A$  and  $B$  must be Boolean-valued variables. The expression is true when either  $A$  is false and  $B$  is true or when  $A$  is true and  $X$  equals  $Y + 1$ .

## Statements

1. Statements may be either "basic" (described below) or "compound" (formed from a sequence of statements enclosed in "statement parentheses", begin and end ). Statements are separated one from the next, by the separator ";".

### Example of a Compound Statement

begin a [i] := b [i] + c [i] ; x := a [i] end

In general, if  $S_1, S_2, \dots, S_n$  are statements then

begin  $S_1; S_2; \dots; S_n$  end

is a statement.

2. A statement may be labeled, as follows:

L: S

where L is a name or an integer; L thus becomes the label of the statement S.

## Assignment Statements

Assignment statements direct that an expression is to be evaluated and assigned as the value of a variable.

### Example of Assignment Statements

1.  $x := a + b$

means: assign the value of  $a + b$  as the value of  $x$ .

2.  $\text{alpha} := 4.63$

3.  $r [i, j] := s [i] + t [j]$

4.  $B := (x > 0) \wedge A$

here A and B are Boolean variables and B receives the truth-value of the Boolean expression on the right.

## go to Statements

1. A go to statement may specify some statement, other than

the statement which follows it, as its successor in the computing process described by the program. This may be done explicitly by writing the label of the desired successor as follows:

go to A

where A is the label of some statement in the program. Or, the successor of the go to statement may be made to depend upon the value of some expression by the use of a switch variable whose subscript is the desired expression. The value of a switch variable is a label. Thus if "branch" is the name of a switch variable, then

go to branch [i+j]

has as its successor that statement whose label is the value of branch [i+j].

2. Switch variables are defined by switch declarations. Thus the switch variable "branch" might be defined by the following declaration:

switch branch( A1, A2, B1, B2 )

where A1, A2, B1, B2 are labels of statements. The value of branch [i+j] is then the (i+j)th label in the sequence (or the nth label, where n is the integer formed by rounding i+j). For example, if i+j equals 3, then the value of branch [i+j] is B1, and in that instance, "go to branch [i+j]" has the same significance as "go to B1"

3. Switch variables may be used in the definition of other switch variables. Thus the following two switch declarations might appear in a program:

switch branch(A1, A2, B1, B2)

switch fork( branch [i], A1, B1)

In this case if i equals 4 and j equals 1, the value of fork [j] is B2.

4. A switch variable may have no value in some cases. For example, branch [i] as defined above has no value if the integer nearest i is less than 1 or greater than 4. In such an instance the successor of "go to branch [i]" is the next statement in the program.

### if Clauses and if Statements

The if clause, "if B", where B is some Boolean expression, combines with the statement, S, following it to form a compound statement. This compound statement has the same effect as S if B is true and has the effect of "no operation" if B is false.

### Example of the use of if statements

```
if (a > 0) ; x := 1 ; if (a = 0) ; begin x := y : p := 0 end ;  
if (a < 0) ; go to B ; S
```

In the above program segment statement S is encountered unless a is negative, in which case 'go to B' is executed; if a is positive and non-zero, x is 1 and p is unaffected when S is reached, otherwise x equals y and p equals 0.

### for Clauses and for Statements

1. A for clause, like an if clause, combines with the statement, S, following it to form a compound statement. A for statement specifies that a given variable take on a succession of values and that the governed statement, S, be executed once for each value of the variable. Thus,

```
for x := 1, r + s, k, 6.3, -10 ; a := (a + x) / x
```

causes the governed statement to be repeated five times, once for each of the listed values of x. Since the statement governed by a for clause may be an involved compound statement which may include other for statements, complex recursive procedures may be easily specified by their use.

2. Sequences of values may also be given as arithmetic progressions:

```
for i := a ↑ 2 ↓ (b/2) y + 1
```

this clause directs that i should assume the values:  $a^2$ ,  $a^2 + b/2$ ,  $a^2 + b$ , ...,  $a^2 + nb/2$ , ... until  $y + 1$  is reached but not passed. Several such progressions may be included in a single for clause

```
for r := 1(3)10, 12(4)20, 21(-7)1
```

this gives the sequence of r-values: 1, 4, 7, 10, 12, 16, 20, 21, 14, 7.

## converge Statements

1. A converge statement preceding a compound statement S causes those substatements comprising S, which do not explicitly specify a successor, to have the statement following S as their common successor. However, in the case of an if statement, only the successor of the governed statement is altered. Thus the successor of an if statement with a false Boolean expression is the next statement. Thus:

```
converge; begin if B1; S1; if B2; S2; if B3; S3 end; S4
```

causes that single statement S<sub>1</sub> to be executed which corresponds to the first true Boolean expression B<sub>1</sub>; this S<sub>1</sub> is then succeeded by S<sub>4</sub>. Thus if B<sub>1</sub>, B<sub>2</sub> and B<sub>3</sub> are all true, the effect is to execute S<sub>1</sub> and then S<sub>4</sub>.

2. A converge statement may also be used to conveniently select a single statement for execution as follows:

```
converge; begin go to branch [i]; L1:S1; L2:S2; L3:S3 end;  
S4 if "branch [i]" takes on the values L1, L2, L3 for i=1,2,3, then  
the statement following converge has the effect of the single statement  
Si followed by S4.
```

## do Statements

1. A do statement is simply a shorthand way of rewriting a segment of a program which appears elsewhere. Thus,

```
do A; x:=y+z; A: r:=x+r
```

is another way of writing the following:

```
B: r:=x+r; x:=y+z; A: r:=x+r
```

here the copied statement has been given a new label, B, simply to avoid having two statements with the same label. The choice of new labels in the copy is immaterial as long as the reassignment is done consistently throughout the copy.

2. A do statement may specify a sequence of statements to be copied by giving two labels.

```
do R, S
```

thus represents the sequence of statements from the one labeled R through the one labeled S.

3. A do statement may require that certain alterations are to be made in the copy which it represents.

do R, S (  $x \leftarrow x + y$ ,  $a[i] \rightarrow \text{beta}$  )

stands for the segment from R through S but with the name "x" replaced by "x+y" everywhere it appears and with "beta" replaced by "a[i]" .

### stop Statements

stop statements have no successor, hence they signify the operational end of the process described by a program.

### return Statements

A return statement, written "return", is used only in programs which define the meaning of procedure statements. It signifies that the defined procedure has been completed and that the successor of the procedure statement which invoked the defining program is the next statement in the invoking program.

### Procedure Statements

A procedure statement indicates that some particular process is to be performed, employing those entities indicated by input parameters and producing results which are designated by the output parameters. (Some procedure statements may also indicate specific successor statements by a list of labels or switch variables which are associated with various exit conditions). Thus

integrate (F(r, ), a, b, e) =:(int):(L)

might indicate that  $\int_a^b F(r, x) dx$  is to be found with an error less than e (if possible with the given procedure), that the result is to be assigned as the value of int and that if the desired result is not obtained the successor of this statement should be the one labelled L

Some procedure statements may simply be added to IAL

as primitives (and realized by machine language subroutines). Others may refer, by the name which stands in front (e.g., "integrate"), to an IAL program (cf procedure declarations) which represents the process indicated.

Some procedure statements may not have any input parameters, others may have no output parameters. Thus a procedure statement which causes one number to be obtained from an external medium might be,

in=: (a)

indicating that the number should be assigned as the value of a. Another might be

print (r, s, t).

Another procedure statement may specify the replacement of a matrix, A, by its inverse:

invert (A [I,]) =: (A [I,])

In general, it is intended that virtually any process can be expressed by a procedure statement.

### Declarations

Declarations in an IAL program state certain facts which are to obtain throughout the program. They may appear at any point in the program, but their position in it is not significant.

### Type Declarations

A type declaration, boolean or integer, asserts that the value of a variable or function or element of an array will always be of the given type. Thus,

boolean (a, b, c)

asserts that any value assigned to entities having the names a, b, or c will be changed to a zero or a one according to same rule, t, (e.g.,  $t(x) = 1$  when  $x = 1$ , otherwise  $t(x) = 0$ ).

Thus

a:= x

in a program containing the above declaration would be equivalent to:

$$a := t(x)$$

or again the appearance of a function  $b(x)$  in an expression in the program would be equivalent to  $t(b(x))$ .

Similarly,

$$\underline{\text{integer}}(r, s)$$

constrains entities having names  $r$  or  $s$  to have integer values by rounding non-integer values to the nearest integer.

### array Declarations

array declarations specify the upper and lower bounds for the subscripts which may be meaningfully associated with an array. Thus

$$\underline{\text{array}}(a, b, c [1:100], r, s [-10, 1:+10, 50] )$$

indicates that  $a, b, c$  are one-dimensional arrays whose subscripts have meaning only in the range 1 to 100 (inclusive), and that  $r, s$  are two-dimensional arrays whose first subscript should lie between -10 and +10 and the second between 1 and 50 (inclusive).

### switch Declarations

switch declarations have already been described under the heading, "go to statements".

### Function Declarations

Certain functions may be defined for use within a program by a function declaration as follows:

$$g(x, y) := w + x + f(y, x)$$

If, in the program containing the above declaration, the following appears:  $g(t/v, h(v))$ , then its value is given by the expression:

$$w + t/v + f(h(v), t/v)$$

Note, in this case, that  $g(x,y)$  is always a function of  $w$  even though it does not appear as an argument.

### comment Declarations

comment declarations do not affect the behavior of the program, they simply provide the ability to incorporate verbal comments in a program. For example:

```
comment This is the end of Section 1.
```

The comment may be arbitrarily long provided it does not contain a statement separator, semicolon.

### Procedure Declarations

Procedure declarations are unique among declarations: precisely one procedure declaration must precede the program it refers to. Such a declaration indicates that the IAL program following defines one or more procedure statements and/or functions. For example:

```
procedure branch ( a ) : ( L1, L2, L3 ) ;  
begin branch : if ( a < 0 ) ; go to L1 ; if ( a = 0 ) ; go to L2 ;  
                  if ( a > 0 ) ; go to L3 end
```

This comprises a procedure declaration followed by a program of one compound statement. The procedure statement defined thereby is one named "branch" which selects one of three successors according as the value of the input is negative, zero, or positive. Thus the procedure statement,

```
branch ( x - f(y) ) : ( a[i] , BB, MM )
```

used in any program means, by virtue of its defining program: go to the statement indicated by the switch variable  $a[i]$  when  $x - f(y)$  is negative, if it is zero, go to BB, and if it is positive, go to MM.

One more example:

```
procedure root (a,b), ckroot(a,b) =: (c) : (d);  
begin ckroot : if(a - b < 0); go to d ;  
          root : c := sqrt(a - b) ; root := c ; return end
```

The procedure declaration, together with the program, defines a function, root, and a procedure statement, named "ckroot". The calculation of the function begins with the statement labelled root, and its value is that of the variable, root, when return is encountered. The program for ckroot begins at the first statement, labelled ckroot.

The process corresponding to

$$\text{ckroot}(x, y/z) =: (a) : (GL)$$

therefor either gives a the value  $\text{sqrt}(x - y/z)$  or, if  $x - y/z$  is negative, specifies the statement labelled GL as its successor. On the other hand,

$$v := r + \text{root}(m, n)$$

results in  $v$  having the value  $r + \text{sqrt}(m - n)$  regardless of the sign of  $m - n$ .

## Part II : Formal Description

### General

The Zurich ACM-GAMM Conference had two principal motives in proposing the IAL :

- 1) To provide a means of communicating numerical methods and other procedures between people.
- and, 2) To provide a means for realizing a stated process on a variety of machines with the only differences between the realizations of a program on two machines being those imposed by differences in word lengths, overflow conditions and the like.

It appears that IAL, as given by informal descriptions here and

elsewhere (reference 1 ), suffices rather well for goal 1) .  
However, if programs are to be written for a variety of machines to translate from IAL to the machine's language in such a way that goal 2) is satisfied, two requirements must be met:

- I) There must exist a precise description of those sequences of symbols which constitute legal IAL programs. Otherwise it will often be the case that a program which is legal and translatable for one translating program will not be so with respect to another.
- II) For every legal program there must be a precise description of its "meaning", the process or transformation which it describes, if any. Otherwise the machine language programs obtained by two translating programs from a single IAL program may behave differently in one or more crucial respects.

Heretofore there has existed no formal description of a machine-independent language ( other than that provided implicitly by a complete translating program ) which has met either of the two requirements above. Consequently, as anyone who has actually been involved in writing a translating program can testify, a large number of decisions affecting membership in the class of legal programs and the meaning of legal programs must be made during the construction of the translating program.

If, therefore, IAL is left in its present state of incomplete and informal description, it is likely that many man-years may be invested in producing a number of translating programs which will not reliably produce equivalent machine programs. In this case the great potential benefits and savings offered by goal 2) will be lost.

The author had hoped to complete a formal description of the set of legal IAL programs and of their meanings in time to present it here. Only the description of legal programs has been completed however. Therefore the formal treatment of the semantics of legal programs will be included in a subsequent paper. The work which has already been done in this direction indicates a need for minor modifications in IAL to facilitate its completion. These

changes will require the approval of the appropriate groups. Since some of these changes are present in both parts I and II of this paper, no 'official' status for the descriptions contained in it should be assumed, although the changes are few and, for the most part, slight.

### Syntax of IAL

In the description of IAL syntax which follows we shall need some metalinguistic conventions for characterizing various strings of symbols. To begin, we shall need metalinguistic formulas. Their interpretation is best explained by an example:

$$\langle ab \rangle : \equiv ( \bar{0} \bar{r} [ \bar{0} \bar{r} \langle ab \rangle ( \bar{0} \bar{r} \langle ab \rangle \langle d \rangle$$

Sequences of characters enclosed in " $\langle \rangle$ " represent metalinguistic variables whose values are strings of symbols. The marks " $;$ " and " $\bar{0} \bar{r}$ " are metalinguistic connectives. Any mark in a formula, which is not a variable or a connective, denotes itself ( or the class of marks which are similar to it ). Juxtaposition of marks and/or variables in a formula signifies juxtaposition of the strings denoted. Thus the formula above gives a recursive rule for the formation of values of the variable  $\langle ab \rangle$ . It indicates that  $\langle ab \rangle$  may have the value "(" or "[" or that given some legitimate value of  $\langle ab \rangle$ , another may be formed by following it with the character "(" or by following it with some value of the variable  $\langle d \rangle$ . If the values of  $\langle d \rangle$  are the decimal digits, some values of  $\langle ab \rangle$  are:

```

[(((1(37(
(12345(
(((
[86

```

### Integers and Numbers

$$\begin{aligned} \langle \text{digit} \rangle &: \equiv 0 \bar{0} \bar{r} 1 \bar{0} \bar{r} 2 \bar{0} \bar{r} 3 \bar{0} \bar{r} 4 \bar{0} \bar{r} 5 \bar{0} \bar{r} 6 \bar{0} \bar{r} 7 \bar{0} \bar{r} 8 \bar{0} \bar{r} 9 \\ \langle \text{integer} \rangle &: \equiv \langle \text{digit} \rangle \bar{0} \bar{r} \langle \text{integer} \rangle \langle \text{digit} \rangle \\ \langle \text{dn} \rangle &: \equiv \langle \text{integer} \rangle \bar{0} \bar{r} . \langle \text{integer} \rangle \bar{0} \bar{r} \langle \text{integer} \rangle \text{ or } \langle \text{dn} \rangle \langle \text{integer} \rangle \\ \langle \text{si} \rangle &: \equiv + \langle \text{integer} \rangle \bar{0} \bar{r} - \langle \text{integer} \rangle \bar{0} \bar{r} \langle \text{integer} \rangle \\ \langle \text{en} \rangle &: \equiv \langle \text{dn} \rangle_{10} \langle \text{si} \rangle \bar{0} \bar{r} {}_{10} \langle \text{si} \rangle \\ \langle \text{number} \rangle &: \equiv \langle \text{integer} \rangle \bar{0} \bar{r} \langle \text{dn} \rangle \bar{0} \bar{r} \langle \text{en} \rangle \end{aligned}$$

## Identifiers and Variables

$\langle \text{letter} \rangle ::= a \text{ or } b \text{ or } c \text{ or } d \text{ or } e \text{ or } f \text{ or } g \text{ or } h \text{ or } i \text{ or } j \text{ or } k \text{ or } l \text{ or } m \text{ or } n \text{ or } o \text{ or } p \text{ or } q \text{ or } r \text{ or } s \text{ or } t \text{ or } u \text{ or } v \text{ or } w \text{ or } x \text{ or } y \text{ or } z \text{ or } A \text{ or } B \text{ or } C \text{ or } D \text{ or } E \text{ or } F \text{ or } G \text{ or } H \text{ or } I \text{ or } J \text{ or } K \text{ or } L \text{ or } M \text{ or } N \text{ or } O \text{ or } P \text{ or } Q \text{ or } R \text{ or } S \text{ or } T \text{ or } U \text{ or } V \text{ or } W \text{ or } X \text{ or } Y \text{ or } Z$   
 $\langle \text{identifier} \rangle ::= \langle \text{letter} \rangle \text{ or } \langle \text{identifier} \rangle \langle \text{letter} \rangle \text{ or } \langle \text{identifier} \rangle \langle \text{digit} \rangle$   
 $\langle \text{id} \rangle ::= \langle \text{identifier} \rangle$   
 $\langle \text{variable} \rangle ::= \langle \text{id} \rangle$

## Subscripted Variables and Arrays

$\langle \text{el} \rangle ::= \langle \text{ar exp} \rangle \text{ or } \langle \text{el} \rangle , \langle \text{ar exp} \rangle$   
 $\langle \text{subscr var} \rangle ::= \langle \text{id} \rangle [ \langle \text{el} \rangle ]$   
 $\langle \text{blank} \rangle ::= \langle \text{the null string of characters} \rangle$   
 $\langle \text{ppel} \rangle ::= \langle \text{blank} \rangle \text{ or } \langle \text{ppel} \rangle \langle \text{ar exp} \rangle ,$   
 $\langle \text{pel} \rangle ::= \langle \text{ppel} \rangle \text{ or } \langle \text{pel} \rangle , \text{ or } \langle \text{pel} \rangle , \langle \text{ar exp} \rangle$   
 $\langle \text{array} \rangle ::= \langle \text{id} \rangle [ \langle \text{pel} \rangle ]$

(see below for definition of  $\langle \text{ar exp} \rangle$ )

## Parameters, Functions and Pure Functions

$\langle \text{nq} \rangle ::= \langle \text{any character other than a quote mark " } \rangle$   
 $\langle \text{arb} \rangle ::= \langle \text{nq} \rangle \text{ or } \langle \text{arb} \rangle \langle \text{nq} \rangle$   
 $\langle \text{quoted string} \rangle ::= " \langle \text{arb} \rangle "$   
 $\langle \text{param} \rangle ::= \langle \text{exp} \rangle \text{ or } \langle \text{quoted string} \rangle \text{ or } \langle \text{array} \rangle \text{ or } \langle \text{pure function} \rangle \text{ or } \langle \text{pure procedure} \rangle$   
 $\langle \text{param list} \rangle ::= \langle \text{param} \rangle \text{ or } \langle \text{param list} \rangle , \langle \text{param} \rangle$   
 $\langle \text{function} \rangle ::= \langle \text{id} \rangle ( \langle \text{param list} \rangle )$   
 $\langle \text{pl} \rangle ::= \langle \text{blank} \rangle \text{ or } \langle \text{pl} \rangle \langle \text{param} \rangle ,$   
 $\langle \text{ppl} \rangle ::= \langle \text{pl} \rangle \text{ or } \langle \text{ppl} \rangle \text{ or } \langle \text{ppl} \rangle , \langle \text{param} \rangle$   
 $\langle \text{pure function} \rangle ::= \langle \text{id} \rangle ( \langle \text{ppl} \rangle )$

## Arithmetic Expressions, Boolean Expressions, and Expressions

$\langle \text{factor} \rangle ::= \langle \text{number} \rangle \text{ or } \langle \text{function} \rangle \text{ or } \langle \text{variable} \rangle \text{ or } \langle \text{subscr var} \rangle \text{ or } ( \langle \text{ar exp} \rangle ) \text{ or } \langle \text{factor} \rangle \uparrow \langle \text{ar exp} \rangle \downarrow$

$$\begin{aligned} \langle \text{term} \rangle &::= \langle \text{factor} \rangle \bar{\text{or}} \langle \text{term} \rangle \times \langle \text{factor} \rangle \bar{\text{or}} \langle \text{term} \rangle / \langle \text{factor} \rangle \\ \langle \text{ar exp} \rangle &::= \langle \text{term} \rangle \bar{\text{or}} + \langle \text{term} \rangle \bar{\text{or}} - \langle \text{term} \rangle \bar{\text{or}} \langle \text{ar exp} \rangle + \langle \text{term} \rangle \\ &\quad \bar{\text{or}} \langle \text{ar exp} \rangle - \langle \text{term} \rangle \end{aligned}$$

$$\begin{aligned} \langle \text{ar exp A} \rangle &::= \langle \text{ar exp} \rangle \\ \langle \text{relation} \rangle &::= < \bar{\text{or}} > \bar{\text{or}} \leq \bar{\text{or}} \geq \bar{\text{or}} = \bar{\text{or}} \neq \\ \langle \text{rel exp} \rangle &::= ( \langle \text{ar exp} \rangle \langle \text{relation} \rangle \langle \text{ar exp A} \rangle ) \\ \langle \text{bool term} \rangle &::= 0 \bar{\text{or}} 1 \bar{\text{or}} \langle \text{rel exp} \rangle \bar{\text{or}} \langle \text{function} \rangle \bar{\text{or}} \\ &\quad \langle \text{variable} \rangle \bar{\text{or}} \langle \text{subscr var} \rangle \bar{\text{or}} \langle \text{bool exp} \rangle \\ &\quad \bar{\text{or}} \neg \langle \text{bool term} \rangle \\ \langle \text{bool exp} \rangle &::= \langle \text{bool term} \rangle \bar{\text{or}} \langle \text{bool exp} \rangle \vee \langle \text{bool term} \rangle \\ &\quad \bar{\text{or}} \langle \text{bool exp} \rangle \wedge \langle \text{bool term} \rangle \bar{\text{or}} \\ \langle \text{bool exp} \rangle &::= \langle \text{bool term} \rangle \end{aligned}$$

$$\langle \text{exp} \rangle ::= \langle \text{ar exp} \rangle \bar{\text{or}} \langle \text{bool exp} \rangle$$

### Assignment Statements

$$\begin{aligned} \langle \text{left element} \rangle &::= \langle \text{variable} \rangle \bar{\text{or}} \langle \text{subscr var} \rangle \\ \langle \text{assmnt stmt} \rangle &::= \langle \text{left element} \rangle := \langle \text{exp} \rangle \end{aligned}$$

### go to Statements and Designational Expressions

$$\begin{aligned} \langle \text{desig exp} \rangle &::= \langle \text{id} \rangle \bar{\text{or}} \langle \text{integer} \rangle \bar{\text{or}} \langle \text{id} \rangle [ \langle \text{exp} \rangle ] \\ \langle \text{go to stmt} \rangle &::= \underline{\text{go to}} \langle \text{desig exp} \rangle \end{aligned}$$

### if Statements

$$\begin{aligned} \langle \text{if clause} \rangle &::= \underline{\text{if}} \langle \text{bool exp} \rangle \\ \langle \text{if stmt} \rangle &::= \langle \text{if clause} \rangle ; \langle \text{stmt} \rangle \end{aligned}$$

### for Statements

$$\begin{aligned} \langle \text{el} \rangle &::= \langle \text{ar exp} \rangle \bar{\text{or}} \langle \text{el} \rangle , \langle \text{ar exp} \rangle \\ \langle \text{unsigned exp} \rangle &::= \langle \text{term} \rangle \bar{\text{or}} \langle \text{unsigned exp} \rangle + \langle \text{term} \rangle \\ &\quad \bar{\text{or}} \langle \text{unsigned exp} \rangle - \langle \text{term} \rangle \end{aligned}$$

$\langle \text{ap} \rangle ::= \langle \text{ar exp} \rangle (\langle \text{ar exp} \rangle) \langle \text{unsigned exp} \rangle$   
 $\langle \text{apl} \rangle ::= \langle \text{ap} \rangle \text{ or } \langle \text{apl} \rangle, \langle \text{ap} \rangle$   
 $\langle \text{for clause A} \rangle ::= \text{for } \langle \text{left element} \rangle := \langle \text{el} \rangle$   
 $\langle \text{for clause B} \rangle ::= \text{for } \langle \text{left element} \rangle := \langle \text{apl} \rangle$   
 $\langle \text{for stmt} \rangle ::= \langle \text{for clause A} \rangle; \langle \text{stmt} \rangle \text{ or } \langle \text{for clause B} \rangle;$   
 $\langle \text{stmt} \rangle$

### converge Statements

$\langle \text{converge clause} \rangle ::= \text{converge}$   
 $\langle \text{converge stmt} \rangle ::= \langle \text{converge clause} \rangle; \langle \text{stmt} \rangle$

### do Statements

$\langle \text{sc} \rangle ::= \langle \text{any character other than " \rightarrow " } \rangle$   
 $\langle \text{ss} \rangle ::= \langle \text{sc} \rangle \text{ or } \langle \text{ss} \rangle \langle \text{sc} \rangle$   
 $\langle \text{subsl} \rangle ::= \langle \text{ss} \rangle \rightarrow \langle \text{id} \rangle \text{ or } \langle \text{subsl} \rangle, \langle \text{ss} \rangle \rightarrow \langle \text{id} \rangle$   
 $\langle \text{label} \rangle ::= \langle \text{id} \rangle \text{ or } \langle \text{integer} \rangle$   
 $\langle \text{do stmt} \rangle ::= \text{do } \langle \text{label} \rangle \text{ or } \text{do } \langle \text{label} \rangle (\langle \text{subsl} \rangle)$   
 $\text{or } \text{do } \langle \text{label} \rangle, \langle \text{label} \rangle$   
 $\text{or } \text{do } \langle \text{label} \rangle, \langle \text{label} \rangle (\langle \text{subsl} \rangle)$

### stop Statements

$\langle \text{stop stmt} \rangle ::= \text{stop}$

### Procedure statements and Pure Procedures

$\langle \text{oe} \rangle ::= \langle \text{left element} \rangle$   
 $\langle \text{out list} \rangle ::= \langle \text{oe} \rangle \text{ or } \langle \text{outlist} \rangle, \langle \text{oe} \rangle$   
 $\langle \text{suc} \rangle ::= \langle \text{label} \rangle \text{ or } \langle \text{id} \rangle [\langle \text{exp} \rangle]$   
 $\langle \text{succl list} \rangle ::= \langle \text{suc} \rangle \text{ or } \langle \text{succl list} \rangle, \langle \text{suc} \rangle$   
 $\langle A \rangle ::= =: (\langle \text{out list} \rangle) \text{ or } \langle \text{blank} \rangle$   
 $\langle B \rangle ::= : (\langle \text{succl list} \rangle) \text{ or } \langle \text{blank} \rangle$   
 $\langle \text{proc stmt} \rangle ::= \langle \text{Function} \rangle \langle A \rangle \langle B \rangle$   
 $\text{or } \langle \text{id} \rangle =: (\langle \text{outlist} \rangle) \langle B \rangle$   
 $\text{or } \langle \text{id} \rangle : (\langle \text{succl list} \rangle)$   
 $\langle \text{ppol} \rangle ::= \langle \text{blank} \rangle \text{ or } \langle \text{ppol} \rangle \langle \text{oe} \rangle,$   
 $\langle \text{pol} \rangle ::= \langle \text{ppol} \rangle \text{ or } \langle \text{pol} \rangle, \text{ or } \langle \text{pol} \rangle, \langle \text{oe} \rangle$   
 $\langle A' \rangle ::= =: (\langle \text{pol} \rangle)$   
 $\langle \text{ppsl} \rangle ::= \langle \text{blank} \rangle \text{ or } \langle \text{ppsl} \rangle \langle \text{suc} \rangle,$   
 $\langle \text{psl} \rangle ::= \langle \text{ppsl} \rangle \text{ or } \langle \text{psl} \rangle, \text{ or } \langle \text{psl} \rangle, \langle \text{suc} \rangle$   
 $\langle B' \rangle ::= : (\langle \text{psl} \rangle)$   
 $\langle F^* \rangle ::= \langle \text{function} \rangle \text{ or } \langle \text{pure function} \rangle \text{ or } \langle \text{id} \rangle$

$$\begin{aligned}
\langle A^* \rangle &::= \langle A \rangle \text{ or } \langle A' \rangle \\
\langle B^* \rangle &::= \langle B \rangle \text{ or } \langle B' \rangle \\
\langle \text{pure procedure} \rangle &::= \langle \text{pure Function} \rangle \langle A^* \rangle \langle B^* \rangle \\
&\text{ or } \langle F^* \rangle \langle A' \rangle \langle B^* \rangle \\
&\text{ or } \langle F^* \rangle \langle A^* \rangle \langle B' \rangle
\end{aligned}$$

[a pure procedure may have any of the forms of a procedure statement but at least one position of one existing list must be empty: at least one input parameter position or one output position or one successor position.]

### return Statements

$$\langle \text{return stmt} \rangle ::= \text{return}$$

### Type Declarations

$$\begin{aligned}
\langle \text{id list} \rangle &::= \langle \text{id} \rangle \text{ or } \langle \text{id list} \rangle, \langle \text{id} \rangle \\
\langle \text{bool declar} \rangle &::= \text{boolean} (\langle \text{id list} \rangle) \\
\langle \text{integer declar} \rangle &::= \text{integer} (\langle \text{id list} \rangle) \\
\langle \text{type declar} \rangle &::= \langle \text{bool declar} \rangle \text{ or } \langle \text{integer declar} \rangle
\end{aligned}$$

### array Declarations

$$\begin{aligned}
\langle \text{ulb} \rangle &::= \langle \text{ar exp} \rangle \\
\langle \text{ulb A} \rangle &::= \langle \text{ar exp} \rangle \\
\langle \text{ulbl} \rangle &::= \langle \text{ulbA} \rangle : \langle \text{ulb} \rangle \\
&\text{ or } \langle \text{ulbA} \rangle, \langle \text{ulbl} \rangle, \langle \text{ulb} \rangle \\
\langle \text{array seg} \rangle &::= \langle \text{id} \rangle [\langle \text{ulbl} \rangle \text{ or } \langle \text{id} \rangle], \langle \text{array seg} \rangle \\
\langle \text{arl} \rangle &::= \langle \text{array seg} \rangle \text{ or } \langle \text{arl} \rangle, \langle \text{array seg} \rangle \\
\langle \text{array declar} \rangle &::= \text{array} (\langle \text{arl} \rangle)
\end{aligned}$$

### switch Declarations

$$\begin{aligned}
\langle \text{de} \rangle &::= \langle \text{id} \rangle \text{ or } \langle \text{integer} \rangle \text{ or } \langle \text{id} \rangle [\langle \text{exp} \rangle] \\
\langle \text{del} \rangle &::= \langle \text{de} \rangle \text{ or } \langle \text{del} \rangle, \langle \text{de} \rangle \\
\langle \text{switch declar} \rangle &::= \text{switch} \langle \text{id} \rangle := (\langle \text{del} \rangle)
\end{aligned}$$

### Function declarations

$\langle \text{function declar} \rangle ::= \langle \text{id} \rangle ( \langle \text{id list} \rangle ) := \langle \text{exp} \rangle$

### Comment declarations

$\langle \text{not sc} \rangle ::= \langle \text{any character other than " ;"} \rangle$   
 $\langle \text{str} \rangle ::= \langle \text{not sc} \rangle \text{ or } \langle \text{str} \rangle \langle \text{not sc} \rangle$   
 $\langle \text{comment declar} \rangle ::= \underline{\text{comment}} \langle \text{str} \rangle$

### Common declarations

$\langle \text{com entry} \rangle ::= \langle \text{id} \rangle ( \langle \text{id list} \rangle )$   
 $\langle \text{com list} \rangle ::= \langle \text{com entry} \rangle \text{ or } \langle \text{com list} \rangle , \langle \text{com entry} \rangle$   
 $\langle \text{common declar} \rangle ::= \underline{\text{common}} ( \langle \text{com list} \rangle )$

### Statements and Declarations

$\langle \text{b stmt} \rangle ::= \langle \text{assnmt stmt} \rangle$   
 $\text{or } \langle \text{go to stmt} \rangle$   
 $\text{or } \langle \text{do stmt} \rangle$   
 $\text{or } \langle \text{stop stmt} \rangle$   
 $\text{or } \langle \text{proc stmt} \rangle \text{ or } \langle \text{return stmt} \rangle$   
 $\langle \text{basic stmt} \rangle ::= \langle \text{b stmt} \rangle \text{ or } \langle \text{label} \rangle : \langle \text{b stmt} \rangle$   
 $\langle \text{sl} \rangle ::= \langle \text{stmt} \rangle \text{ or } \langle \text{sl} \rangle ; \langle \text{stmt} \rangle$   
 $\text{or } \langle \text{sl} \rangle ; \langle \text{declar} \rangle$   
 $\text{or } \langle \text{declar} \rangle ; \langle \text{sl} \rangle$   
 $\langle \text{u stmt} \rangle ::= \langle \text{b stmt} \rangle \text{ or } \langle \text{if stmt} \rangle$   
 $\text{or } \langle \text{for stmt} \rangle \text{ or } \langle \text{converge stmt} \rangle$   
 $\text{or } \text{begin } \langle \text{sl} \rangle \text{ end}$   
 $\langle \text{stmt} \rangle ::= \underline{\langle \text{u stmt} \rangle} \text{ or } \underline{\langle \text{label} \rangle} : \langle \text{u stmt} \rangle$   
 $\langle \text{declar} \rangle ::= \underline{\langle \text{type declar} \rangle} \text{ or } \underline{\langle \text{array declar} \rangle}$   
 $\text{or } \underline{\langle \text{switch declar} \rangle} \text{ or } \underline{\langle \text{function declar} \rangle}$   
 $\text{or } \underline{\langle \text{comment declar} \rangle} \text{ or } \underline{\langle \text{common declar} \rangle}$

### Syntactic Properties of IAL Statement Lists

a) L is the entry label of a statement S if S has the form:

L:  $\langle \text{u stmt} \rangle$

b) If  $d_2$  is a designational expression in statement list S and there is a switch declaration:

$$\text{switch } x_1 := (\dots, d_2, \dots)$$

then  $d_2$  is said to succeed any designational expression of the form  $x_1 [\langle \text{exp} \rangle]$ . Further, if  $d_3$  succeeds  $d_2$  and  $d_2$  succeeds  $d_1$ , then  $d_3$  succeeds  $d_1$ .

### Definition of an IAL Program

An IAL Program is a statement list ( $\langle \text{sl} \rangle$ ) with the following properties:

- 1) No two statements or substatements have the same prefixed label.
- 2) No designational expression succeeds itself.
- 3) Every designational expression has some label as successor, or is itself a label.
- 4) To every n-place array name or subscripted variable name there corresponds one and only one entry of that name in an n-place array segment ( $n = 1/2$  number of entries in the upper-lower-bound list,  $\langle \text{ulbl} \rangle$ ).

### ACKNOWLEDGMENT

The proposed International Algebraic Language is the direct and indirect product of many people's work. Most directly it is the result of the Zurich Conference at which the participants were: F. L. Bauer, H. Bottenbruch, H. Rutishauser, and K. Samelson ( representing the GAMM ), and J. W. Backus, C. Katz, A. J. Perlis, and J. H. Wegstein ( representing the ACM ). Extensive preparatory work was done by larger ACM and GAMM committees ( see reference 1 ). All of this, in turn, was based on prior work done by many individuals and groups in the field of automatic programming. The earliest work ( to the author's knowledge ) in this area was that of Rutishauser, in Switzerland, and of Laning and Zierler, in the United States.

## REFERENCE

1. Perlis, A. J., and Samelson, K., "Preliminary Report - International Algebraic Language"  
Communications of the ACM vol. 1, no. 12 ( Dec. 1958 )

