PROPOSAL FOR A UNIVERSAL LANGUAGE FOR THE DESCRIPTION OF COMPUTING PROCESSES*

*For technical reasons, this manuscript had to be translated from German in very short time and may contain therefore minor errors and omissions.

PROPOSAL FOR A UNIVERSAL LANGUAGE FOR THE DESCRIPTION OF COMPUTING PROCESSES

by

F. L. Bauer
H. Bottenbruch
H. Rutishauser
K. Samelson

INTRODUCTION

In the following, the first stage of an algorithmic language representing the basis of the formula translation project Zurich-Munchen-Mainz-Darmstadt is developed. The language is intended to permit complete description of any computing process in a compact and easily legible and controllable form, with the important restriction that for the time being only calculations with real numbers are permitted. Hypercomplex quantities of all kinds (such as complex and double precision numbers, vectors, matrices, etc.) are not included at the moment; calculations with such quantities must be written down componentwise.

The symbols necessary for representation of the elements of the language are chosen as far as possible in conformity with usages of printing. Furthermore the language has been adapted as far as possible to the following postulates:

I) The language should coincide wherever possible with standard mathematical formula notation, and should, for a mathematician, be readable without further explanations.

II) It should be possible to use the language in publications, for the description of computing processes.

III) The language should describe any computing process as accurately as to allow automatic translation into machine programs.

With regard to realization of postulate III it must be remarked that in consequence of the restricted number of characters on commercial input-output apparatus multiple use of certain characters or verbal transcriptions cannot be avoided. Possibilities of the international teletype code, and of punch card tabulators are carefully studied.

1. Development of the Formula Language

A first attempt to automatize programming for computers was made by H. Rutishauser (Lecture given at the GAMM-conference 1951; publication 1952 "Automatische Rechenplanfertigung bei programmgesteuerten Rechenmaschinen").

The method proposed required that formulae should be written down in an "appropriate way",

$$\begin{aligned} &((a \times b) + c) \Rightarrow s \\ &(a - b) \times c \Rightarrow t \\ &t + s - (a \times c) + (b \times b) \Rightarrow z \end{aligned} \qquad (1)$$

wherefrom the machine carries out translation into the machine program.

The main principle was to write to the left of the "ergibt"-symbol "=>" the arithmetic expression to be evaluated and to the right the designation of the new quantity defined by the calculation. This "ergibt" symbol corresponds better to the dynamic process of computing than the usual equality symbol. Especially it can be used in situations where the latter may lead to contradictions, e.g., s + 2 => s ("old" s plus 2 yields "new" s).

For description of loops generated by running subscripts, a symbolization of the following form was proposed (example for multiplication of a matrix by a vector):

$$\begin{aligned} &\text{Für } i = 1(1)\ n: \\ &\text{Für } j = o: \qquad o \Rightarrow h_j \\ &\text{Für } j = 1(1)\ n: \quad h_j - 1 + (a_{ij} \times x_j) \Rightarrow h_j \\ &\text{Für } j = n \qquad h_j \Rightarrow y_i \end{aligned} \qquad (2)$$

2. Principles of the New Language

The new language follows essentially the ideas given by Rutishauser in his original paper as stated in Section 1: A formula program, that is description of a computing process in this language, consists in "ergibt" formulae (describing the arithmetic operations to be carried out), and in "guiding symbols" (e.g., for i = 1 ... or similar verbal statements) describing the structure of the computing process. Ergibt-formulae and guiding-symbols are called "statements".

In "ergibt" - formulae, to the left the usual arithmetic expressions appear which may contain, as operands, numbers, simple variables and subscripted variables; furthermore brackets and operations symbols (for restrictions see table of forbidden combinations in Section 3). To the right of the "ergibt" symbol, in any case only a (simple or subscripted) variable (this being the result of the calculation) is permitted.

Following postulate I, the "guiding symbols" explained in Chapters II and III have been formulated verbally.

CHAPTER I

CONSTRUCTION OF "ergibt" - FORMULAE

3. Symbols for Construction of "ergibt" - Formulae

Elementary building blocks for "ergibt" formulae are the following symbols which are only representatives for underlying abstract concepts.

		general abbreviation
a) letters	a b ..... z A B ..... Z and other alphabets	λ
b) figures	o 1 2 9	ξ
c) decimal comma	, [1]	
decimal base for scaling	lo (lowered small lo)	
d) separating symbols		
semicolon	;	τ
for strong separation	| or ;;	τ
colon	:	τ
brackets	() []	τ
half line shift ↓↑	[2] (for index positions)	τ
e) operations symbols	+ - . [1] /	ω
f) definition symbols	=> :=	

1) Alternative proposal: decimal point
and multiplication symbol ×

2) Framing of symbols means that in printed texts the symbols are replaced by actual line shifts. On punching equipment where half line shifts are impossible, symbols corresponding to printed as well as punched characters must be used.

4. Elements of "ergibt" formulae

With the symbols defined so far the following entities can be built.

α) Numbers Z
general form: $\pm\ \xi\xi\ldots\xi,\xi\ldots\xi_{10} \pm \xi\ldots\xi$

where the optional scale factor $_{10} \pm \xi\ldots\xi$ as well as + signs are optional. In integers, no decimal comma is needed.

examples
273
2,73 $_{10}$ 2
3,14159
-0,047
+0,478 $_{10}$ - 2

β) Simple variables V
as designations for numbers as in elementary arithmetics.
general form: letter λ followed by arbitrary letters and/or figures
examples
a
x1 [1]
alpha

No distinctions are made between fixed point and floating point variables.

All those combinations of letters which are used for guiding symbols can not be used as names for variables. They are listed below

FOR	ALWAYS	VARY
FROM	END	STOP
TO	GO TO	AND
IF	ANY	OR
CASE	STEP	

Also forbidden are combinations used generally as function names such as ABS, EXP, COS, SIGN.

γ) Subscripted Variables 20
as designation for components of multidimensional arrays of numbers such as vectors and matrices

general form: V [↓] ; ... ; [↑]

1) The figure 1 in second position is to be interpreted as a distinguishing mark, not as a subscript.

(name of variable, followed by line shift character downwards
arbitrary number of index positions separated by semicolons
and line shift character upwards)

Index positions can be occupied by numbers, simple or subscripted variables as well as by arithmetic expressions $\mathcal{A}$ as defined in Section 7.

examples

a_i
$zeta_{7;i\cdot(i+1)/2}$
$b_{i-j;k+j}$ (3)
$inverse_{i;k}$

Subscripts can only take on integral values. At the moment, no provisions are made for fractional subscripts as they appear in the calculus of finite differences.

5. Functions and Functionals

α) Functions $\mathcal{F}$ are abbreviations for complicated computing processes, especially for the standard functions of analysis.

general form: $\mathcal{FN}[\ ;\ ;\ ]$

Function name (identical to variable with or without subscript) followed by opening function bracket "[", an arbitrary number of argument positions separated by semicolons, and the closing function bracket "]".

The argument positions can be occupied by arbitrary arithmetic expressions as defined in Section 7 (in the simplest case by numbers Z or simple variables V). These expressions may themselves contain functions.

examples

SQRT [x]
SIN [(w·t)+f]
$P_{\ell;m}$ [v;w] (4)
COS [m.ARCCOS [x]]
Q [u_i;v_j]

Certain arguments of many standard functions of analysis usually take on integral values only and are then formally written as subscripts. If such "subscripts" take on non-integral values, they must be written as function arguments.

Therefore not $J_{n+1/2}$ [x] but J [n+1/2;x]

With regard to the last example of (4) it must be remarked that for each argument, a separate argument position must be provided explicitly. The last example of (4) therefore can only mean that the two argument positions of the function Q[x;y] are to be occupied by the components u_i and v_j respectively given by the present values of the subscripts i and j.

β) Functionals $\mathcal{OJ}$ are introduced as abbreviations for computing processes which contain arbitrary functions and/or variable array (e.g., vectors); the result of such a functional can only be a number (otherwise see Section 6).

general form: $\mathcal{W}$[;;... ;; ;; ...]

Name of functional followed by function bracket, an arbitrary number of argument function positions, strong separation symbol, an arbitrary number of argument positions and function bracket. The argument positions are occupied in the same way as in the case of functions.

Argument function positions are occupied either by a function of a prescribed number of arguments or by a subscripted variable with a prescribed number of subscripts.

The argument of subscript positions of the above-mentioned quantities are occupied by simple variables as dummies for defining the number of such positions.

These dummy variables have meaning only for the functional in question and have no connections to synonymous variables occurring somewhere outside this functional.

examples:
SIMPSON [f[x];; a ; b ; n]
RUNGE KUTTA [g[u;v];; x ; y ; h] (5)
SCALAR PRODUCT [a_k ; b_k ;; n]

γ) Definition of Functions.

If functions and functionals appearing in computing processes must be defined - which is always the case for functions and functionals for which no conventions exist - a constructive definition (i.e., a computing procedure) must be given. Especially all functions occurring as argument functions in functionals must be defined in that way (even if they are standard functions) in order to clarify the meaning of the arguments of these argument functions.

general form: $\mathcal{F}$: statements .....
$\mathcal{F}$:= $\mathcal{V}$ ..... => $\mathcal{V}$ 1)

(For functionals write $\mathcal{OJ}$ in place of $\mathcal{F}$).

Function (functional) symbol, followed by colon, then by the statements (formulae and guiding symbols) which define the value of the function (-al). These in turn are followed by the function value assignment which at the same time indicates the end of the function definition.

1) An alternative proposal is the notation $\mathcal{V}$ => $\mathcal{F}$ for the function value assignment.

Logically, a function definition is not a part of the formulae program but rather an auxiliary definition which may be written down at any place, preferably in the form of a footnote or at the end of the formulae program.

In case the function value can be computed by one arithmetic expression $\mathfrak{A}$ only, the latter may be combined with the function value assignment in the general form

$$\mathfrak{F} := \mathfrak{A}$$

Warning: A function definition has "sectional character" and must therefore meet the section postulates I and II, formulated in Chapter II.

6. Pseudoformulae

Technically pseudoformulae serve for calling in library programs which generally have the character of functionals $\mathfrak{F}$, but often yield several results R1,...,Rn which may be arrays of numbers.

Furthermore, because of internal decisions, they usually allow a number of emergency exits, which bear names Y1,...,Yk. For these reasons they cannot be incorporated into arithmetic expressions but must be used in the form of pseudoformulae.

general form:

$\mathfrak{F}$ R1 => $\mathfrak{N}_1$; ... ; Rn => $\mathfrak{N}_n$;
IF Y1 : N1 ; IF Y2 : N2 ; ... IF Yk : Nk ;;

Here $\mathfrak{F}$ is a functional symbol with appropriate arguments. R1 ... Rn are names of pseudovariables each of which without bearing subscripts may represent an array of numbers.

The Y1 ... Yk are names which correspond to internal decisions of the process which is called in by the pseudoformula. The N1 ... Nk are guiding symbols, namely the jumps which have to be carried out if the respective exits Y1 ... Yk occur.

These jumps may be:

a) without return (executed by GO TO)
b) with return through the exit back into the process (so called intermediary exit) and must be executed by section-call (as defined in Section 11).

The names R1 ... Rn and Y1 ... Yk are characteristic for the process called in by the pseudoformula.

Example:

$$\begin{aligned} &\text{GAUSS-JORDAN } [a_{ik} : b_{\ell;m} ;; n ; s ; 10^{-8}] \\ &\text{INVERSE} \Rightarrow c_{j;k} ; \quad \text{SOLUTION} \Rightarrow y_{\ell;m} ; \text{ DET} \Rightarrow z; \\ &\text{IF SING : GO TO 15 ; IF SUM : ALARM ;;} \end{aligned} \tag{6}$$

Warning: Pseudoformulae have sectional character and must therefore fulfill the section-postulates.

7. Rules for the Construction of "ergibt" Formulae

α) Arithmetical expressions $\mathcal{A}$ are composed of

Operation symbols	ω
Brackets	( )
Numbers	Z
Variables	V, $\mathcal{U}$
functions	$\mathcal{F}$
functionals	$\mathcal{F}_J$

according to the usual rules of arithmetic.

It should be noted that

1) . / takes precedence over +-
2) not allowed (since not unique) is a/b.c or a/b/c
3) Intended multiplication (ab instead of a.b) is forbidden.
4) Superfluous brackets are allowed. Numerically they are not meaningless, since expressions in brackets are computed at some time as intermediate results.
5) Besides, the sequence of execution corresponds to the sequence of notation.

Examples:

$$\begin{aligned} &(a+b).(c+d) \\ &a/(c+d) - b.e/f \\ &x.\sin[a.y] + b.\cos[c.y] \\ &a_{i;j} \cdot b_{j;k} \end{aligned} \tag{7}$$

β) The "ergibt" formulae, being the proper computing statements, have the form of equations solved for the unknown variable.

general form: $\mathcal{A} \Rightarrow \mathcal{U}$;

Arithmetic expression as defined above, followed by the ergibt symbol, the variable to be computed, and the separation character.

Names of variables are free as soon as the designated numbers are no longer needed. Reuse of a variable name is permitted even in the same formula.

Example: $s + a_j \Rightarrow s$ (8)

In printed texts only a reuse of a variable name should be indicated by adding a superscript in brackets; example (8) will then read as follows

$$s^{(j)} + a_j \Rightarrow s^{(j+1)} ;$$

CHAPTER II

GUIDING SYMBOLS FOR DESCRIPTION OF THE STRUCTURE OF COMPUTING PROCESSES

In the following it will be shown how to describe structurally complicated computing processes with the aid of this language.

The process is broken up into closed parts which in the sequel are called parts of sectional character. In ordinary programming they correspond to a) loops, b) open and closed subroutines, c) conditional parts of a program.

These "sectional parts" which will be defined in Chapters II and III have to meet certain conditions, namely:

Section postulate I

Any two "sectional parts" $\mathfrak{A}$ and $\mathfrak{L}$ must either be disjoint, or one of them must be totally contained in the other one.

Section postulate II

In general it is not allowed to jump to a place inside a sectional part from the outside (jumps are explained in Section 9).

8. Guiding Symbols for Loop Control

For the description of cyclic processes (for instance recurrence formulae), the values of subscripts for which the calculation is to be carried out, must be defined:

general form:

$\mathfrak{W}$ FROM $\mathfrak{A}1$ TO $\mathfrak{A}2$; STEP $\mathfrak{A}3$:

alternative

$\mathfrak{W}$:= $\mathfrak{A}1$ ... $\mathfrak{A}2$; STEP $\mathfrak{A}3$:

This means that all subsequent statements (up to the corresponding guiding symbol "VARY ", see below) are to be executed for the sequence of values

$$\mathfrak{A}1;\ \mathfrak{A}1 + \mathfrak{A}3;\ \mathfrak{A}1 + 2.\mathfrak{A}3;\ \ldots\ \mathfrak{A}1 + n.\mathfrak{A}3,$$

where $n \leq \frac{\mathfrak{A}2 - \mathfrak{A}1}{\mathfrak{A}3} < n + 1$

if this n turns out to be negative the loop will be skipped completely.

The following rules must be observed:

a) $\mathfrak{A}1$, $\mathfrak{A}2$, $\mathfrak{A}3$ may be arbitrary expressions in the sense of Section 5 α), which, however, must not contain the running variable itself.

b) If $\mathfrak{A}3 = 1$, the part "; STEP 1" may be omitted.

c) The variable $\mathcal{N}$ need not take on integral values only. Therefore, also the values of expressions $\mathcal{A}1$, $\mathcal{A}2$, $\mathcal{A}3$ are not necessarily integers. All values of expressions in subscript positions, however, must be integers.

d) Cyclic iteration processes may, or may not depend explicitly on a running variable. For safety reasons, however, a maximum number of iteration steps should be prescribed in any case by aid of the "FOR...." symbol with an appropriate $\mathcal{A}3$.

The end point of the loop in the formula program initiated by the guiding symbol "FOR ...." is defined by the guiding symbol

VARY $\mathcal{N}$;

This replaces the words "Ende Index $\mathcal{N}$" in Rutishauser's original paper as well as the use of indentions in later proposals and in example (2).

When the running variable $\mathcal{N}$ has attained its final value the loop is left, and the statements written down immediately after the symbol "VARY $\mathcal{N}$" are carried out. If, in these statements, the variable $\mathcal{N}$ appears again, it is given the value it had when leaving the loop (This holds even when the loop is left, because of an internal decision, by a separate exit).

Example:
$$\begin{aligned} &0 \Rightarrow h_0 \\ &\text{FOR } i := k...n; \quad \text{STEP } k : h_{i-k} + i.s_i \Rightarrow h_i \\ &\text{VARY } i \\ &h_i/k \Rightarrow g_k \end{aligned} \tag{9}$$

In the description of multiply cyclic processes each loop, according to postulate I, must be enclosed totally by the next higher one as shown in the following example (Matrix . Vector):

$$\begin{aligned} &\text{FOR} \quad i := 1...n \; : \quad 0 \Rightarrow h_i^{(0)} \\ &\text{FOR} \quad j := 1...n \; : \quad h_i^{(j-1)} + a_{i;j} \cdot x_j \Rightarrow h_i^{(j)} \\ &\text{VARY} \quad j \\ &\qquad\qquad\qquad\qquad\quad h_i^{(n)} \Rightarrow y_i \\ &\text{VARY} \quad i \\ &\text{STOP} \end{aligned}$$

Warning: Loops have sectional character, and must meet section postulates I and II.

9. Jumps and Statement Numbers

Statements may be labeled by statement numbers N in order to be reached by jumps from other places in the formula program.

general form: §§ ... § :
an arbitrary number of figures followed by a colon.

To be precise, statement numbers designate always the one statement following them, and serve only as points of reentry (after a jump) into the formula program.

Jumps to the beginning of labeled statements are effected by the guiding symbol

GO TO N ;

This means that the computation has to proceed from the entry point labeled by N without automatic return to the jump-off place.

Warning: It has to be remembered that jumps from the outside into parts of sectional character according to section postulate II are dangerous and in general not allowed.

It is quite reasonable on the other hand to leave a part of sectional character by a jump. In fact, this is usual procedure for leaving an iteration loop.

10. Sections

Closed parts of the formula program may be defined as sections for the dual purpose of a) clarifying the structure of the process, and b) repeated use at different places (in the sense of open or closed subroutines in ordinary programming).

A properly defined section consists of
section designation symbol
statements of the section
section end symbol

General form of the section designation symbol A:

Name of section (at least one letter followed by an arbitrary number of letters and/or figures) followed by a colon.

example: a :
pivots :
ak2 :

General form of the section end symbol:
"END" followed by the section name and strong separation symbol.

example: END pivots ;;
END ak2 ;;

Naturally a section may contain other parts of sectional character such as section or loops.

example:
```
17 : ... => x ;
MATRIVECT :
FOR i:=1...n:        0 => h;
18:  FOR  j:=1...n: h + a_i;j . x_j => h;
     VARY j;                                    (11)
                     h => y_i;
VARY i;
END MATRIVECT ;;
19:  FOR i:=1...n : y_i ... ;
```

Of course, sections have sectional character, and have to meet the section postulates. Thus, in example (11), no jump is possible from outside of section MATRIVECT to point 18 but jumps are allowed from any point between FOR i... and VARY i to point 18 as well as 17 and 19.

11. Section Calls

Section calls are the means of inserting (in a logical sense) sections already defined somewhere else.

general form: section name followed by semicolon.
(For clarity, it seems advisable to add another word such as DO... in front of the section name.)

example:
```
pivots ;
MATRIVECT ;
```

In practive, it will happen quite often that the set of computing statements contained in a section A is to be used at a different place, X, but that the names of some of the variables in A must be replaced according to the requirements of the process at position X. This is done by a substitution procedure initiated by the section call with substitution

general form:

A[ℋ1:= 𝒞1; ... ;ℋ n:= 𝒞n;; ℱ 1:= 𝒞 1; ... ℱ m:= 𝒞m;;;...];

Here, ℋ 1:=𝒞1 means that the variable symbol ℋ 1 appearing in section A must, for this call-in be replaced by the expression 𝒞 1. All variables contained in expression 𝒞1 must be defined numerically at point X.

example: At place X in the formula program a matrix (a_{ij}) has to be multiplied by a vector (p_i), the resulting vectors being named (q_i). Since the product matrix . vector is already defined as section

MATRIVECT - [see example (11)] the required operation can be performed by calling in section MATRIVECT as follows:

```
                .............=> p_i;
                VARY i ;
        X       MATRIVECT [x_i:= p_i; y_i:= q_i] ;          (12)
                FOR i := 1...n: q_i + ... ;
```

In case the section MATRIVECT is to be used for the multiplication of a matrix (a_{ij}) by a matrix (p_{kh}) it is <u>not allowed</u> to write this as follows:

```
                FOR h : = 1...n:
                MATRIVECT [x_i : = p_i;h ; y_i : = q_i;h]    (13)
                VARY h
```

because the number of subscripts of the variables which appear on both sides of :=, must be identical. To write it correctly, the column vectors of the matrix (p_{ij}) have to be extracted, before entering the section call.

CHAPTER III

CONDITIONAL STATEMENTS

12. Explicitly Conditional Statements

If a computation has to proceed in different ways depending on certain results computed previously an "explicit condition $\mathfrak{L}$" (IF-formula) may be formulated.

general form: IF C1 AND C2 AND ... AND Ck :

"IF", followed by an arbitrary number of criterions Ci (in conjunction), separated by "AND" and terminated by colon.

Each of the criterions has the form

$$\mathfrak{A}_1 \; \mathfrak{K} \; \mathfrak{A}_2$$

where $\mathfrak{A}_1$, $\mathfrak{A}_2$ are "expressions" as defined in Section 7 α), and $\mathfrak{K}$ is one of the "comparing characters" listed below:

$$= \quad \neq \quad > \quad \geq \quad < \quad \leq$$

The statements following an explicit condition $\mathfrak{L}$ up to the symbol ALWAYS which terminates the range of validity of $\mathfrak{L}$, are carried out only if that condition is fulfilled; otherwise these statements are skipped and the computation is continued immediately after ALWAYS.

Moreover, the range of validity of an explicit condition is terminated by a new IF - Formula, as well as after GO TO.

examples:

a) IF ABS [(x+y) . x] > 10^{-5} : statements ... ; (14)
ALWAYS ;
means therefore, that the statements are carried out only if $|(x+y)\,x| > 10^{-5}$.

b) IF a > 0 : a . a + b . b => c ;
IF a ≤ 0 : 2 . a . b => c ; (15)
ALWAYS
FOR i : = ...
Here c is computed as $a^2 + b^2$ or 2ab depending on whether a > 0 or a ≤ 0. After that, FOR i:= ... is carried out unconditionally.

c) IF x > -1 : statement 1
IF x < 1 : statement 2 (16)
IF x = 0 : statement 3

In this case, for x < -1 only statement 2, for x > 1 only statement 1 is carried out, but for $|x| \leq 1$ statements 1 and 2; for x = 0 all 3 statements are carried out in the sequence given.

Example c) shows that IF-formulae are completely independent from each other. On the other hand the rules stated make it impossible to build up a hierarchy of conditions (i.e., to define sub-cases of a condition). For such cases see Section 13.

Warning: The program part lying between IF and the next IF or ALWAYS, i.e., the set of all statements which are carried out under one condition, has sectional character and has to fulfill the section postulates. Therefore the following combination is forbidden:

```
FOR   i : = ...  :  statements ;
IF  k > 0        :  statements ;          (17)
VARY  i ;
ALWAYS  ;
```

since the two segments FOR-VARY and IF-ALWAYS overlap.

According to the section-postulates, the range of validity of an explicit conditionis not terminated by a section call, regardless of possible IF-formulae which may occur in the section.

```
Thus  IF  x > 0 :  statements
                   RIVET  ;                (18)
                   statements
      ALWAYS
```

means the condition x > 0 remains valid through RIVET until the next ALWAYS after it, even if a new IF-formula and a corresponding ALWAYS occurs in the section RIVET.

Important applications of explicit conditions are conditional jumps in the formula program which are effectuated by the combination

IF ... : GO TO N :

Here no closing ALWAYS is needed, since GO TO itself cancels the validity of IF...

As a further example we give here the complete formula program for the computation of ln x (not the best method):

```
1:  (x . x + 1)/2x => p0 ; (x . x - 1)/2x => q0  ;
2:  FOR  k := 1 ... 20 :
        SQRT [1/2 . (1 + pk-1)] => pk          (19)
             qk-1 < pk          => qk
```

```
3:  IF  ABS [p_{k-1}] - 10^-5 < 0 :  GO TO 6:
4:  VARY k ;
5:  NOT CONV ;                                        (19-Cont'd)
6:  3.q_k/(2 + p_k) => ln x ;  STOP;
```

Here the iteration loop is left as soon as $|p_{k-1}| < 10^{-5}$ which yields a 11-digit-accuracy of ln x. Since this condition is for any x with $10^{-200} \leq x \leq 10^{200}$ fulfilled after at most 20 steps, the upper limit 20 in the guiding symbol FOR k:= 1 ...20 is only a safety limit in case of machine error.

It has to be noted, that also criteria of non-arithmetic character are admitted in IF-formulae;

```
e.g., IF SENSE-SWITCH ON : ...
      IF u = Q-Number : ...
```

13. "Discriminations"

For complicated branchings of a formula program, the discriminations can be used. These consist of

a) Definition of cases
b) Call in of cases

a) The Definition of a case has the general form:
Designation of the case (letter λ, followed by one or more figures and closing bracket)
and a criterion (same as in IF-formulae, see Section 12)
terminated by semicolon.

```
example:    a1)  x < 0
            a2)  x = 0                                (20)
            a3)  ELSE
```

Here three cases are defined: case a1) corresponds to $x < 0$, case a2) to $x = 0$ and case a3) to the complement of the sum of the two others, i.e., $x > 0$.

What is done with case a3) is called a remainder - case definition and is preferably used to avoid that in complicated situations for some values of the variables no case at all is defined.

Moreover, as in Section 12, several criterions may be used in conjunction to define cases, e.g.,

```
b1)  x < 0  AND  y < 0  ;
b2)  x < 0  AND  y ≥ 0  ;
b3)  x = 0  AND  y = 0  ;                             (21)
b4)  x > 0  AND  y < 0  ;
b5)  x > 0  AND  y > 0  ;
b6)  ELSE
```

b) The "Call in" of cases.

The cases thus defined can be called in later arbitrarily and several times if needed. Such a call in is effectuated as follows:

general form of "Call in":

CASE, followed by designation of case and colon; later on the statements to be carried out under the condition of the case and terminated by the guiding symbol

ANY CASE

or by another "CASE"

examples:

```
CASE a1) : statements 1 ;
CASE a2) : statements 2 ;
CASE a3) : statements 3 ;
CASE a1) : statements 4 ;
ANY CASE ;
statements 5 ;
```
(22)

Here statements 1 and 4 are carried out if the criterion of case a1 ($x < 0$) is fulfilled, whereas the statements 5 are unconditionally.

It is allowed:

1) After ANYCASE to call in again some of the cases at any time.
2) To call in some of the cases together as shown by the following example:

```
CASE a1) :              statements 6 ;
CASE a2) OR a3) :       statements 7 ;
ANY CASE ;
statements 8 ;
```
(23)

3) To define subcases
   If a case has been called in subcases may be defined and designated in the sense of decimal classification.
   example: (Suppose the cases a1, a2, a3 of example (2o) are already defined here)

```
                    :
                    :
CASE a1) : statements ;
           a11) y < 0 ;
           a12) y ≥ 0 ;
CASE a2) : statements ;
           a21) y = 0 ;
           a22) ELSE  ;
CASE a3) : statements ;
           a31) y < 0 ;
           a32) y > 0 ;
           a33) ELSE  ;
```
(24)

Thus the cases of example (20) are split up into subcases a11, a12 ... a33 which in fact coincide with the cases b1,...b6 of example 21 (b6 = logical sum of cases a22 and a33).
These subcases may be called in themselves and further subcases a111), a112) ... may be defined then.

4) If, after subcases have been defined, for instance

CASE a3)

is called in, this means automatically a re-unition of all subcases of a3) but by a subsequent

CASE a31)

the subcase a31) alone is again called in.

Warnings:

5) The call in of a case, more exactly, the interval from a CASE to the next CASE or ANYCASE has sectional character and has therefore to meet the section postulates. On the other hand it is allowed to insert IF-formulae under a case.

6) A new definition of a case a ξξ ... ξ automatically cancels all cases which have been defined earlier under the same letter. Therefore, if two families of cases have to be used independently at the same time, different guiding letters have to be used, e.g.,

```
a1)      x > 0  ;
a2)      ELSE   ;
b1)      y > 0  ;      Definition of cases
b2)      y = 0  ;
b3)      ELSE   ;
   .
   .                   statements 1
   .                   (these are unconditional)
CASE a1)  OR  b1)  :    statements 2  ;
CASE a2)           :    statements 3  ;
ANYCASE  ;
   .
   .                   unconditional statements 4
   .
CASE b1)  :             statements 5  ;
CASE b2)  OR  a2)  :    statements 6  ;
ANYCASE  ;
   .
   .                   unconditional statements 7
   .
a1)    z = 0       <-- This new definition of a case destroys
                       the previously defined cases a1 and a2.
```

Here only the statements 1, 3, 4, 6 are executed if $x < 0$, $y < 0$, but all statements if $x < 0$, $y > 0$. Zurich, May 9, 1958.