Prof. Dr. H. Stoyan
Universität Erlangen-Nürnberg
Institut für Mathematische Maschinen
und Datenverarbeitung (Informatik VIII)
Am Weichselgarten 9
91058 Erlangen

Proposal For A Programming Language

General This report gives the technical specifications of a programming language proposed by the Ad Hoc Committee on Languages of the Association for Computing Machinery. The membership of this committee is as follows:

J. W. Backus (I.B.M.)
P. H. Desilets (Remington Rand)
D. C. Evans (Bendix Aviation Corp.)
R. Goodman (Westinghouse)
H. Huskey (University of California)
C. Katz (Remington Rand)
J. McCarthy (M.I.T.)
A. Orden (Burroughs Corp.)
A. J. Perlis (Carnegie Institute of Technology)
R. Rich (Johns Hopkins University)
S. Rosen (Burroughs Corp.)
W. Turanski (Remington Rand)
J. Wegstein (U. S. Bureau of Standards)

The objectives of the Ad Hoc Committee in designing the language described herein were to provide a language suitable for:

(1) publication of computing procedures in a concise and widely-understood notation,

and

(2) accurate and convenient programming of computing procedures in a language mechanically translatable into machine programs for a variety of machines.

It is recognized that certain one-for-one substitutions of one character-sequence for another will often be required to put a program written in the proposed language into a form mechanically acceptable by the input equipment of a given machine.

Certain subsidiary properties were taken to be necessary or strongly desirable to satisfy the two main goals above:

(a) The set of rules required to specify the syntax of the language should be kept as brief and uncomplicated as possible.

Prof. Dr. H. Stoyan
Universität Erlangen-Nürnberg
Institut für Mathematische Maschinen
und Datenverarbeitung (Informatik VIII)
Am Weichselgarten 9
91058 Erlangen

(b) The meaning of each statement in a program, in any context and under any circumstances, should either be determined completely and precisely by a simple set of rules for interpretation of statements, or the syntax rules of (a) above should exclude the configuration from the class of legal programs.

(c) Deviations from the conventions of mathematical or other notation in the proposed language are acceptable, even desirable, when such deviations improve properties (a) and (b) above.

The statements of the proposed language described below are written as sequences of characters on a line (no characters appear in a raised or lowered position). It is felt that the ability of the language to be written in this way will greatly simplify the preparation of typescripts, eliminate errors due to misjudgment of character position in hand-written material, eliminate rules for dealing with subscripts of superscripts and the like, and make possible direct transcription for mechanical translation.

Character Set

Character set for publication	Representation with minimum character set	Representation with intermediate character set	Meanings
A - Z	A - Z	A - Z	alphabetic characters (letters)
0 - 9	0 - 9	0 - 9	decimal digits or no meaning
+	'A'	+	unary plus or binary add
-	'S'	-	unary minus or binary subtract
*	'M'	*	multiply
/	'D'	/	divide
↑	'E'	**	exponentiate (a'b means a^b)
(	'L'	(	left parenthesis
)	'R'	)	right parenthesis
.	'P'	.	period or radix point
→	'I'	'I'	"a→b" = "if a is true, then take b or "substitute symbol a for b"
|...|	'AB'	ABS( )	absolute value
∧	'AND'	'AND'	AND
∨	'OR'	'OR'	OR
≠	'EXOR'	'EXOR'	exclusive or
~	'NOT'	'NOT'	not
,	'C'	,	comma
=	'EQ'	=	"equals" or set quantity on left equal to quantity on right
<	'LT'	'LT'	less than
>	'GT'	'GT'	greater than
≤	'LTE'	'LTE'	less than or equals
≥	'GTE'	'GTE'	greater than or equals
'			used only to represent special characters in smaller character
;	'O'	'O'	end of statement

Any non-ambiguous representation of the characters in column 1, beyond those in columns 2 and 3, utilizing the character-set of a given machine, would be permitted.

Use of Spaces in Statements

The deletion of one or more blank spaces between two characters shall not alter the interpretation of any statement.

Symbols

A symbol is a sequence of letters and/or digits beginning with a letter. A symbol may refer to a unit of data, an array of such units, a segment of program, a function, or a subroutine. In the text below the lower case letter "s", with or without numerical subscripts, will be used to indicate a place in a statement occupied by an arbitrary symbol.

Program Structure

A description of a complete, self-contained procedure is called a program and consists of a sequence of statements. A program may be written in line format or in telegraphic format. In the former case each statement is begun at the left hand margin of a line and continuations of a statement on subsequent lines are indicated by generous indenting on these lines. No end-of-statement punctuation is required in a program in line format. In telegraphic format each statement is followed by an exclamation point, "!", and the program is a single sequence of characters.

A program is terminated by the declarative statement:

FINIS

This statement thus serves as a separation mark between two programs.

Statement Structure

A statement is a sequence of characters and may have one or two parts: a content part (always), which may or may not be preceded by a name part. If a name part is present, it has the following form:

(s)

That is, a symbol enclosed in parentheses. The symbol is said to be the name of the statement.

The Procedure Described by a Program

A program will in general contain both declarative statements, which state certain facts which obtain throughout the program, and imperative statements, which are to be executed in a well defined sequence. This sequence is determined in the following way: after a given imperative statement has been executed the next statement to be executed is the next imperative statement occurring in the program unless the given statement directs otherwise. In this case the execution of the given statement will determine which imperative statement is to be executed next.

Thus the process described by a program is the sequence of the processes described by each statement, taken in their sequence of execution.

Symbol Classification

A symbol which is used in a program to refer to a unit of data, an array of data units, or a function is called a data symbol. (The data referred to in the case of a function is the function value.) A data symbol falls in one of the following classes:

a) Integer
b) Boolean
c) General

according as its referent or referents are integers, boolean quantities ("true" = 1, "false" = 0), or neither of these types. The class of a data symbol is determined by its initial segment of characters in accordance with a convention or with certain symbol classification statements occurring in the program.

Symbol Classification Statements

Symbol classification statements are declarative statements. The symbol classification statements are:

INTEGER ($s_1, s_2, \ldots, s_n$)
BOOLEAN ($s_1, s_2, \ldots, s_n$)
GENERAL ($s_1, s_2, \ldots, s_n$)

Thus each statement contains a list of symbols, each separated from the next by a comma, and the list enclosed in parentheses.

Suppose one is given a set of symbol classification statements and wishes to determine the class of a given symbol, s, governed by them. One finds the

longest symbol, s', (greatest number of characters) contained in one of the symbol classification statements which is an initial segment of the given symbol s. The class of s is then determined by the statement containing s'. If none of the symbol classification statements contain a symbol which is an initial segment of s then s belongs to the general class.

Example

Given:

```
INTEGER (IR, RA)
BOOLEAN (IRE, IRK)
GENERAL (RAP, IRKS)
```

Then:

Symbol	Class
IRA	Integer
IRELAND	Boolean
IRKED	Boolean
IRKSOME	General
RA	Integer
RAP	General
RAM	Integer
R	General
REAR	General

Symbol Classification Convention

The symbol classification convention is that the following symbol classification statements are to be understood as accompanying every program whether or not other symbol classification statements are specified:

```
INTEGER (I, J, K, L, M, N)
BOOLEAN (Q)
```

If a specified symbol classification statement lists one of the above symbols, the supplied statement shall take precedence (when there is any conflict) over the two provided by convention. The classification of a symbol in a given program is, therefore, governed by the symbol classification statements included in it plus the two above. Thus for programs which do not contain any symbol classification statements, there are, nevertheless, an unlimited number of symbols in each class.

Arrays

Certain symbols may be used to refer to arrays of data-units. Such a symbol, s, may be followed by subscripts, e_i, separated by commas and the sequence of subscripts enclosed in parentheses:

$$s(e_1, e_2, \ldots, e_n)$$

A subscript is an expression with an integer value (see later paragraph on expressions). The occurrence in a program of an array symbol accompanied by subscripts effects a reference to that data-unit in the array whose coordinates are the sequence of subscript values. Under certain circumstances reference may be made to an entire array by writing its symbol without any subscripts:

s

The class of data-units in an array is given by the class of the array symbol.

ARRAY Statements

ARRAY statements are declarative statements which identify those symbols which refer to arrays and indicate the number of coordinates (dimensionality) of each array and the maximum value that each subscript may assume. (Subscripts range from 1 to this maximum.) Thus an ARRAY statement has the following form:

$$\text{ARRAY } s_1(l_1),\ s_2(l_2), \ldots,\ s_n(l_n)$$

where the s_i are the desired array symbols and the l_i are sequences of integer constant subscripts. The number of such constants is the number of coordinates of the array and each constant is the maximum value a subscript in that coordinate position may assume throughout the execution of the program.

All array symbols must appear in an ARRAY statement.

Example

ARRAY A(10), BETA1 (5, 5, 20)

means: A is a 1-dimensional array of 10 elements; BETA1 is a 3-dimensional array, 5x5x20.

Operators

The following characters denote various operators as indicated in the section: character-set; they may be grouped as follows:

1) Arithmetic operators
   + - * / ↑ |...|

2) Boolean operators
   ∽ ∧ ∨ ≠

3) Relational operators
   = < > ≤ ≥

Arithmetic operators have one or two numerical quantities (floating-point or integer) as operands and give a numerical result which is of the same type as the operand in the case of unary operators and is floating-point in the case of binary operators unless both operands are integers, in which case the result is the nearest integer whose absolute value is less than or equal to the absolute value of the real number resulting from the operation.

Boolean operators have one or two boolean quantities as operands and give a boolean quantity as the result.

Relational operators have two numerical operands and a boolean result.

Constants

A numerical constant is symbolized by a sequence of numeric characters. If there is a decimal point included or at either end, a floating-point constant is indicated, if not the constant is an integer or, depending on context, a boolean constant, 0 or 1.

Variables

A variable is denoted by a symbol or an array symbol with appropriate subscripts. The class of the symbol is the class of the variable quantity. A numerical variable in the general class is understood to refer to a floating-point quantity.

Functions

A function is designated by a symbol (the name of the function) followed by a list of parameters (defined below) which are separated by commas; the list is enclosed in parentheses. All functions are single valued and the class of the function symbol determines the type of the result (integer or floating-point) for numerical functions. The type of parameter which may appear in a given position of the list may be restricted in the case of particular functions.

Expressions

Numerical expressions:

1. an integer or floating-point (numerical) constant
2. a numerical variable
3. a numerical subscripted variable with an integer-valued expression in each subscript position
4. a function which takes on numerical values and which has expressions or other parameters in each argument position which satisfy the requirements of the particular function

In the definitions below, the symbols E and F are used to denote any expressions and G to denote any expression whose first symbol is not " + " nor " - ".

5. +G
6. -G
7. (E)
8. E + G
9. E - G
10. E * G
11. E / G
12. E↑F↓
13. ~~[illegible]~~ ab [E]

Boolean expressions:

0. A boolean-valued function with appropriate arguments
1. A boolean constant
2. A boolean variable
3. (E = F)
4. (E < F)
5. (E > F)
6. (E ≤ F)
7. (E ≥ F)

In the following definitions "P" and "Q" represent boolean expressions.

8. ⌣P
9. (P)
10. P∨Q
11. P∧Q
12. P≠Q

Interpretation of Expressions

The precise sequence in which the operations in an expression are performed may have a considerable effect on the result. Thus it is often important to know what sequence of operations is specified and to be able to dictate this sequence completely. Furthermore, since it is desirable to be able to intermingle floating-point and integer quantities in an expression one must be able to determine whether floating or integer arithmetic will be used for any particular operator.

Sequance of Operations

Sequence is determined by

1. First, parentheses (and absolute sign).
2. Second, order of precedence of operators:
   ~~1) ↑~~ 2) * / 3) + -
3. Third, left to right.

Use of integer or floating arithmetic for an operator in a numeric expression is determined as follows:

~~1. operands of same type give~~ arithmetic and result of that ~~type~~

~~2. operands of different types give~~ floating arithmetic after ~~conversion of integer operand to floating form. Result is~~ floating.

Replacement Statements

Numerical:

"V = E"

where V is a numeric variable or subscripted variable and E is an expression, means "substitute the value of E as the value of V after putting the result in the appropriate form (floating or integer) as determined by V". If E is boolean the value of V will be 1 or 0 according as E is true or false.

Boolean:

"V = E"

means substitute the value of E as the value of V. V and E are both boolean.

GO TO Statements

A GO TO statement may specify some statement other than the one immediately following as the statement to be executed next. They have the form:

GO TO e

where e is a designational expression. A designational expression is defined as follows:

1. The name (a symbol) of an imperative statement in the program containing this designation.
2. An expression s(E) where s is a symbol and E is an integer expression. For each such symbol s there must be a corresponding declarative statement:

   SWITCH $s(e_1, e_2, \ldots, e_n)$

   where each e_i is a designational expression. The statement designated when E has the value k is that one (if any) designated by e_k. If $E \leq 0$ or $E > n$, no statement is designated.
3. An expression of the form

   $(p_1 \longrightarrow e_1,\ p_2 \longrightarrow e_2, \ldots, p_n \longrightarrow e_n)$

   where each p_i is a boolean expression and each e_i is a designational expression. The statement designated is that one (if any) designated by e_k where p_k is the first true expression of $p_1, p_2, \ldots, p_k$. If no p_i is true, no statement is designated.

When e designates no statement, the statement to be executed next is the one following "GO TO e" in the program.

VARY and LOOP Statements

A VARY statement causes a segment of program immediately following it to be executed several times, once for each of a number of values of a variable given in the VARY statement. The segment of program to be repeated is terminated by a matching LOOP statement: namely, the first subsequent LOOP statement which is not the mate of some other VARY statement. Thus VARY and LOOP act like left and right parentheses respectively in their role of designating segments of program. A VARY statement has the following form:

VARY v = r

where v is a variable and r is a list of values. A list of values may have one

of two forms:

$$e_1(e_2)e_3(e_4)e_5\ldots e_n$$

where each e_i is an expression and each alternate expression beginning with the second is enclosed in parentheses. The sequence of values designated by this first form is:

$e_1, e_1+e_2, e_1+2e_2, \ldots, e_1+ne_2$ ($e_1+ne_2 < e_3$ when $e_1 \leq e_3$, or $e_1+ne_2 > e_3$ when $e_1 \geq e_3$), $e_3, e_3+e_4, e_3+2e_4, \ldots, e_3+ne_4$ ($e_3+ne_4 < e_5$ when $e_3 \leq e_5$, or $e_3+ne_4 > e_5$ when $e_3 \geq e_5$), ... and so on.

The second form for a list of values is:

$$e_1, e_2, \ldots, e_n$$

where each e_i is an expression and the sequence designated is:

$$e_1, e_2, \ldots, e_n$$

LOOP statements have the following form:

LOOP

or

LOOP s

A symbol may be written after "LOOP" for mnemonic purposes; it has no effect on the meaning of the statement. When a LOOP statement is encountered the variable given by its matching VARY is assigned its next value in the given list of values and the statement following the matching VARY is executed next (and its successors). If the variable of the matching VARY already has its last value of the list when LOOP is encountered, the statement following LOOP is executed next.

If a GO TO statement in the segment controlled by a VARY statement passes control to a statement not in that segment or a RETURN statement is encountered, the effect of the VARY statement is terminated and its given variable retains the value it had at that moment.

If a statement not in the segment controlled by a VARY statement passes control to one in that segment, the significance of the terminating LOOP statement is not defined.

If a statement in the segment controlled by a VARY statement specifies the assignment of a value to the variable of the VARY, the effect of the VARY and the corresponding LOOP statement is not defined.

Procedure Statements

Call

Procedure statements have the following form:

s(l)

or,

s

where s is a symbol which is a name of a subroutine and l is a list of parameters separated by commas. A subroutine is any program (see below) or other procedure which is so constructed that it may be invoked by a procedure statement.

A parameter may be any of the following (depending on the properties of the particular subroutine):

1. An expression
   (variables, constants or functions are, of course, included)
2. The symbol of an array
   (without subscripts)
3. A sequence of characters preceded by the sequence nC where n is an integer denoting the length of the sequence and "C" is for identification of this configuration. Thus "3 C 1 2." is a parameter which provides the subroutine with the sequence of characters "1", "2", and ".".
4. The name of a statement in the program containing the procedure statement
5. The name of a subroutine
6. The name of a function

When a procedure statement, s(l), is encountered in a program the process indicated by the subroutine for the name s is performed, employing the parameters, l, after which control passes to the statement following s(l) unless the subroutine specifies otherwise (in which case the parameters must specify the possible successors of s(l)).

STOP Statements

When a STOP statement (an imperative statement) is encountered in a program, the procedure described by it is terminated. STOP statements have the form:

STOP

SUBROUTINE Statements

A SUBROUTINE statement is a declarative statement which indicates that the entire program containing it is a subroutine and is to be invoked only by procedure statements in other programs. A SUBROUTINE statement has the

following form:

SUBROUTINE $s(s_1, s_2, \ldots, s_n)$

where s is a name of the subroutine and each s_i is some symbol occurring in the subroutine.

The significance of a subroutine and of the SUBROUTINE statement or statements which it contains can be best described by considering a second program which contains a procedure statement whose identifying symbol is the name given by one of the SUBROUTINE statements in the subroutine. Suppose the procedure statement is:

$S1(p_1, p_2, p_3, p_4)$

Then the process which it signifies is described by the program of the subroutine beginning with the first imperative statement following the SUBROUTINE statement giving the name S1 and ending with the first RETURN statement encountered (see paragraph on RETURN statements) or the first GO TO which designates a statement in the program containing the procedure statement. If the SUBROUTINE statement is:

SUBROUTINE S1(SA, SB, SC, SD)

In the execution of the portion of the subroutine given above the symbols SA, SB, SC, and SD are replaced everywhere in the subroutine by the parameters p_1, p_2, p_3 and p_4. act of replacement

Symbols (other than subroutine and function names) in a subroutine have no relation to any identical symbols which may occur in a program having a procedure statement invoking the subroutine.

The correctness of a procedure statement may be determined by substituting its parameters for the appropriate symbols in the subroutine it refers to and determining whether the subroutine is then a legal program.

FUNCTION Statements

A FUNCTION statement is a declarative statement which indicates that the program containing it is a subroutine which may only be invoked by the appropriate use of the function name (in some expression), or the use of other function or subroutine names referring to this subroutine, in another program.

The form of a FUNCTION statement is:

$$\text{FUNCTION } s(s_1, s_2, \ldots, s_n)$$

where s is the name of the function and each s_i is a symbol occurring in the subroutine. The portion of program executed as a result of the appearance of the function symbol s in some program and the substitution of the parameters given there for the symbols $s_1, s_2, \ldots, s_n$ in the subroutine and the method of terminating the execution of the subroutine are precisely the same as in the case for SUBROUTINE statements. The value the function will have in the expression containing a reference to it will be that of the variable s (the name of the function which has been used also as a variable within the subroutine) when RETURN is encountered.

RETURN Statements

RETURN statements are to be used only in subroutines and when encountered indicate the process invoked by a reference to the subroutine has been completed. If the subroutine was referenced by a procedure statement, control will resume in the referring program at the statement following the procedure statement, when RETURN is encountered in the subroutine. If the reference was initiated by the occurrence of a function, the evaluation of the expression containing the function will continue when RETURN is encountered in the subroutine.

The form of a RETURN statement is:

RETURN

Conditional Statements

A conditional statement is one which has the effect of one of several given statements in accordance with certain conditions which exist when it is encountered. Let any of the following imperative statements be termed a module:

1. A replacement statement
2. A GO TO statement
3. A RETURN statement
4. A STOP statement
5. A procedure statement
6. A substitution statement

Then conditional statements are recursively defined as any statement of the following form:

$$P_1 \rightarrow S_1,\ P_2 \rightarrow S_2,\ \ldots,\ P_m \rightarrow S_m$$

where each P_i is a boolean expression and each S_i is a module or a conditional

statement enclosed in parentheses.

The effect of a conditional statement is that of the single statement S_i following the first true boolean expression, P_i, $i = 1, 2..., m$. More precisely, the effect of the conditional statement given above, where the name of the next statement is NEXT, is the same as that of the following sequence of statements:

```
          GO TO(P1→N1, P2→N2, ..., Pm→Nm)
          GO TO NEXT
N1)       S1
          GO TO NEXT
N2)       S2
          GO TO NEXT
          .
          .
          .
Nm)       Sm
NEXT)
```

LABEL Statements

A LABEL statement is a declarative statement which associates a symbol (label) with an arbitrary sequence of statements occurring in the program containing the LABEL statement. The form of a LABEL statement is:

$$\text{LABEL } s(s_1, s_2)(s_3, s_4)\ldots(s_{2n-1}, s_{2n})$$

where each s_{2i-1} is the name of a statement and either each s_{2i} is the name of a statement following the statement named s_{2i-1}, or $s_{2i} = s_{2i-1}$. The sequence of statements designated is the concatenation of the subsequences: from s_1 through s_2 both inclusive, from s_3 through s_4 both inclusive, etc. s is said to be the label of this sequence. Declarative statements are not included in the labelled sequence.

Substitution Statements

A substitution statement is one which stands for a sequence of statements which has been labeled by a LABEL statement. It permits one to avoid rewriting segments of program with minor changes and to avoid complications in the flow of control in a program which would often be required to avoid rewriting segments, were substitution statements not available. The form of a substitution statement is:

$$s(s'_1 \rightarrow s_1,\ s'_2 \rightarrow s_2, \ldots,\ s'_n \rightarrow s_n)$$

or

$$s$$

where s is the label of a sequence of statements given by a LABEL statement, each s_i is a symbol appearing in the labelled sequence and the corresponding s'_i is a symbol to be substituted for the symbol s_i everywhere in the given sequence.

The appearance of a substitution statement in a program in general means: execute the labelled sequence of statements, first making the specified substitutions of symbols, and thereafter pass control to the statement following the substitution statement. This definition of the effect of a substitution statement may be ambiguous under certain complicated circumstances. The effect of a program containing substitution statements is precisely described in terms of an equivalent program from which substitution statements have been eliminated. Given a program containing substitution statements the equivalent program which defines the behavior of the first is constructed as follows:

1. Replace each conditional statement by the sequence of statements which precisely define it, assigning statement names to the statements so generated so as to be distinct from all other statement names in the program.
2. Replace each substitution statement, which does not itself belong to a labelled sequence, by the appropriate labelled sequence of statements, substituting the indicated symbols for the corresponding ones in the labelled sequence.
3. After each replacement of a substitution statement by the appropriate sequence, substitute for each statement name of the statements in the sequence either the symbol specified by the substitution statement or an arbitrary, unique symbol (one which is used nowhere else in the program) in a one-for-one fashion everywhere the name appears in the sequence. If the substitution statement has a name, that name is to be substituted for that of the first statement of the sequence unless another substitution is specified.

The use of substitution statements must be such that if steps 1, 2 and 3 of the above process are applied a finite number of times, no substitution statements remain. The legitimate use of substitution statements is determined by the legality of the program which results.

It should be noted that a program containing substitution statements merely represents in a shorthand fashion the transformed program obtained as above. Thus the original program may contain GO TO statements which refer to statements existing only in the transformed program.

Summary of Declarative Statements

Symbol Classification	Form
INTEGER	INTEGER $(s_1, s_2, \ldots, s_n)$
BOOLEAN	BOOLEAN $(s_1, s_2, \ldots, s_n)$
GENERAL	GENERAL $(s_1, s_2, \ldots, s_n)$
Other	
SWITCH	SWITCH $s(e_1, e_2, \ldots, e_n)$ where e_i is a designational expression.
ARRAY	ARRAY $s_1(l_1), s_2(l_2), \ldots, s_n(l_n)$ where l_i is a list of constant subscripts.
SUBROUTINE	SUBROUTINE $s(s_1, s_2, \ldots, s_n)$.
FUNCTION	FUNCTION $s(s_1, s_2, \ldots, s_n)$
LABEL	LABEL $s(s_1, s_2)(s_3, s_4)\ldots(s_n, s_r$
FINIS	FINIS

Summary of Imperative Statements

Statement Type	Form
Replacement	V = E where V is a variable and E is an expression.

Statement Type	Form
VARY	VARY V = r where V is a variable and r is a list of values of the forms: $e_1(e_2)e_3(e_4)e_5 \ldots e_r$ or $e_1, e_2, \ldots, e_n$ where e_i is an expression.
LOOP	LOOP
GO TO	GO TO e where e is a designational expression.
STOP	STOP
RETURN	RETURN
Procedure	s or, $s(p_1, p_2, \ldots, p_n)$ where p_i is a parameter and s is a symbol.
Substitution (*)	s or, $s(s'_1 \longrightarrow s_1, s'_2 \longrightarrow s_2, \ldots, s'_n \longrightarrow s_n$ where s and s_i are symbols.
Conditional	$P_1 \longrightarrow S_1, P_2 \longrightarrow S_2, \ldots, P_n \longrightarrow S_n$ where P_i is a boolean expression and S_i is a module or a conditional statement enclosed in parentheses.

(*) Recall that a substitution statement is a shorthand designation for a sequence of imperative statements.