EXAMPLES OF COMPUTER AIDS TO ALGORITHM VERIFICATION

by

W. D. Maurer

Memorandum No. ERL-M319

January 1972

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Research sponsored by the National Science Foundation, Grant GJ-821.

EXAMPLES OF COMPUTER AIDS TO ALGORITHM VERIFICATION

W. D. Maurer

The following descriptions of programs and sample input and output were produced by students in the fall quarter of 1971 as class term projects. Each is an aid to the verification of programs which are written in a specific languages. The languages and the students who wrote the computer aids (all of which have themselves been written in SNOBOL 4) are as follows:

COMPASS (assembly language, CDC 6400) -- Michael Megas
SNOBOL 4 -- Joon Chang
PL/I -- Edward Gould
Extended SYMBOL (assembly language, XDS Sigma 2) --
    Mark Burnside
ALGOL -- Richard Harris
ALGOL -- Pavel Stoffel
ALGOL -- Hideki Nakano
DAP-16 (assembly language, Honeywell 416) --
    Jean-Yves Le Goic
PDP-8 Assembly Language -- Alan Campbell

The minimum amount of work accepted for credit was a program which listed out the control paths, relative to an arbitrary set of control points, of a program written in a well-defined subset of the given language. Specifically, suppose that the input program has assertions given before certain of its statements (the control points). Then a control path is any way that the program can get from one control point to another. Each of these is listed as follows. First, the assertion attached to the initial control point is printed out. Any assignments occurring in the control path are listed as they are. If a conditional transfer appears in the control path, then some condition (either the given condition or its negation) must be satisfied if this particular control path is to be taken, and this condition (and <u>not</u> the entire conditional statement) is printed out at the point in the path where the conditional transfer occurred. Finally, the assertion attached to the final control point is printed. All this is done for all control paths. This is the tedious part of the job of proving a program (partially) correct; the person who is proving the correctness of a program is expected to go through the control paths as listed and prove, for each one, that if it is started with its initial assertion valid, and if it is actually taken, then when it finishes the final assertion given will be valid.

Still another program verifier, not described here, has been written by Pauline Wong, for the handling of FORTRAN input programs. It was written in SNOBOL 4 and later recoded in FORTRAN by a group of six students under Pauline Wong's direction.

PRESENT FUNCTIONS OF THE COMPASS VERIFICATION PROGRAM

Michael C. Megas
Dec. 17, 1971

I. Lists the input program

II. Determines the Control Paths of that program

III. Lists those control paths, with the following information and notations:

1. The specified preconditions

2. The program statements

3. The register assignment(s) made by each statement

   REG B3 SET TO :I$5:
   means register B3 set to the Integer value 5

   REG A6 SET TO :A$J:
   refers to the address of variable J

   REG X1 SET TO :VO$L:
   refers to initial value of variable L

   REG A2 SET TO :S$ARY(VO$L):
   refers to address of the subscripted variable ARY(L) (initial value of L)

   REG X2 set TO :VO$ARY(VO$L):
   refers to initial value of ARY(L)

4. Notes assignments of new values to variables

   NEW VALUE IN VALS :V1$ARY(I$5) .EQ. I$13:
   means ARY(5) has been assigned the integer 13 as its new value (V1)

5. Post Conditions (which it does not yet verify)

6. The final value string showing all of the values each variable has had thruout the control path, including those which were assigned by the preconditions.

7. Error messages whenever combinations are attempted which are not meaningful to the program, e.g., adding the address of one variable to the address of another variable.

SUCCESSFUL COMPILATION

THE INPUT PROGRAM IS:

```
         * J .EQ. 2:K .EQ. 7
               SB3        5
               SX6        B3-2
               SA6        J
         * ARY(5).EQ.13
               SA1        L
               SA2        ARY+X1
               SA1        ARY
               SA3        A1+5
               SA4        A2+5
               SA5        A3+5
               SX6        X2
               SA6        A3
               SA4        A2+L
               SA5        A3+L
         *IFIRST .EQ. 8:ISEC .EQ. 6
               SA1        JOE
               SX6        X1-4
               SA6        IFIRST
               PL         X1,L1
               SX3        X6+ISEC       NO $C EXPT
         L1    SA4        ISEC          DUMMY
         *ERROR TEST
               SA1        II
               SA2        A1+3          NO $C EXPT
               SA3        J             DUMMY
               SA4        K             DUMMY
         * I .EQ. 3:J .EQ. 45:K .EQ. 9 : ARY(3) .EQ. 2
               SA1        I
               SA2        J
               SA3        K
               IX6        X1+X2
               SA6        A1
               BX6        X1
               SA6        A6
               IX6        X2/X3
               SA6        J
               IX6        X1*X3
               SA6        J
               BX7        X6
               SA7        ARY+X1
         *LAST CONDITION
               END
```

```
*********CONTROL PATH 1 ****************************
1      * J .EQ. 2:K .EQ. 7
2           SB3        5
                                  REG B3 SET TO  :I$5:
3           SX6        B3-2
                                  REG X6 SET TO  :I$3:
4           SA6        J
                                  REG A6 SET TO  :A$J:
                                  NEW VALUE IN VALS  :V1$J.EQ.I$3:
5      * ARY(5).EQ.13
*********END OF CONTROL PATH 1 ********************
           FINAL VALUE STRING :V1$J.EQ.I$3:VO$J.EQ.I$2:VO$K.EQ.I$7:

*********CONTROL PATH 2 ****************************
5      * ARY(5).EQ.13
6           SA1        L
                                  REG A1 SET TO  :A$L:
                                  GVV DIAG   VAR  :L:  VAL  :VO$L:
                                  REG X1 SET TO  :VO$L:
7           SA2        ARY+X1
                                  REG A2 SET TO  :S$ARY(VO$L):
                                  GVV DIAG   VAR  :ARY(VO$L):  VAL  :VO$ARY(VO$L):
                                  REG X2 SET TO  :VO$ARY(VO$L):
8           SA1        ARY
                                  REG A1 SET TO  :S$ARY:
                                  GVV DIAG   VAR  :ARY:  VAL  :VO$ARY:
                                  REG X1 SET TO  :VO$ARY:
9           SA3        A1+5
                                  REG A3 SET TO  :S$ARY(I$5):
                                  GVV DIAG   VAR  :ARY(I$5):  VAL  :I$13:
                                  REG X3 SET TO  :I$13:
```

```
 10          SA4         A2+5
                                              TCODE BEFORE SI BRANCH IS  :V:
                                              REG A4 SET TO  :S$ARY(MVO$L+I$5):
                                              GVV DIAG   VAR  :ARY(MVO$L+I$5):  VAL  :VO$ARY(MVO$L+I$5):
                                              REG X4 SET TO  :VO$ARY(MVO$L+I$5):
 11          SA5         A3+5
                                              TCODE BEFORE SI BRANCH IS  :I:
                                              REG A5 SET TO  :S$ARY(I$10):
                                              GVV DIAG   VAR  :ARY(I$10):  VAL  :VO$ARY(I$10):
                                              REG X5 SET TO  :VO$ARY(I$10):
 12          SX6         X3
                                              REG X6 SET TO  :I$13:
 13          SA6         A2
                                              REG A6 SET TO  :S$ARY(I$5):
                                              NEW VALUE IN VALS  :V1$ARY(I$5).EQ.I$13:
 14          SA4         A2+L
 COMBINING SUB.ADDR. W/ ADDR.
 FIR AND SEC  :S$ARY(VO$L):A$L:
 15          SA5         A2+L
 16    *IFIRST .EQ. 8:ISEC .EQ. 6
 ***********END OF CONTROL PATH 2 ********************

          FINAL VALUE STRING :V1$ARY(I$5).EQ.I$13:VO$ARY(I$5).EQ.I$13:



 **********CONTROL PATH 3 ***************************
 16    *IFIRST .EQ. 8:ISEC .EQ. 6
 17          SA1         JOE
                                              REG A1 SET TO  :A$JOE:
                                              GVV DIAG   VAR  :JOE:  VAL  :VO$JOE:
                                              REG X1 SET TO  :VO$JOE:
 18          SX6         X1-4
                                              REG X6 SET TO  :MVO$JOE-I$4:
 19          SA6         IFIRST
                                              REG A6 SET TO  :A$IFIRST:
                                              NEW VALUE IN VALS  :V1$IFIRST.EQ.MVO$JOE-I$4:
 20        (X1 .LT. ZERO)
 21          SX3         X6+ISEC      NO SC EXPT
 COMBINING ADDR. W/M-CODE
 FIR AND SEC  :MVO$JOE-I$4:A$ISEC:
 22    L1    SA4         ISEC         DUMMY
 23    *ERROR TEST
 ***********END OF CONTROL PATH 3 ********************

          FINAL VALUE STRING :V1$IFIRST.EQ.MVO$JOE-I$4:VO$IFIRST.EQ.I$8:VO$ISEC.EQ.I$6:
```

```
**********CONTROL PATH 4 ******************************
16    *IFIRST .EQ. 8:ISEC .EQ. 6
17          SA1       JOE
                                   REG A1 SET TC  :A$JOE:
                                   GVV DIAG   VAR  :JOE:  VAL  :VO$JOE:
                                   REG X1 SET TO  :VO$JOE:
18          SX6       X1-4
                                   REG X6 SET TO  :MVO$JOE-I$4:
19          SA6       IFIRST
                                   REG A6 SET TO  :A$IFIRST:
                                   NEW VALUE IN VALS  :V1$IFIRST.EQ.MVO$JOE-I$4:
20       (X1 .GE. ZERO)
22    L1    SA4       ISEC         DUMMY
                                   REG A4 SET TO  :A$ISEC:
                                   GVV DIAG   VAR  :ISEC:  VAL  :I$6:
                                   REG X4 SET TO  :I$6:
23    *ERROR TEST
**********END OF CONTROL PATH 4 ********************

         FINAL VALUE STRING :V1$IFIRST.EQ.MVO$JOE-I$4:VO$IFIRST.EQ.I$8:VO$ISEC.EQ.I$6:
```

```
**********CONTROL PATH 5 ***************************
23    *ERROR TEST
24          SA1       II
                                   REG A1 SET TC  :A$II:
                                   GVV DIAG   VAR  :II:  VAL  :VO$II:
                                   REG X1 SET TO  :VO$II:
25          SA2       A1+3         NO SC EXPT
COMBINING ADDR. W/I-CODE
FIR AND SEC  :A$II:I$3:
26          SA3       J            DUMMY
27          SA4       K            DUMMY
28    * I .EQ. 3:J .EQ. 45:K .EQ. 9 : ARY(3) .EQ. 2
**********END OF CONTROL PATH 5 ********************

         FINAL VALUE STRING :VO$ERRORTEST:
```

```
********* CONTROL PATH 6 *****************************
28    * I .EQ. 3:J .EQ. 45:K .EQ. 9 : ARY(3) .EQ. 2
29          SA1       I
                                  REG A1 SET TO  :A$I:
                                  GVV DIAG   VAR  :I:  VAL  :I$3:
                                  REG X1 SET TO  :I$3:
30          SA2       J
                                  REG A2 SET TO  :A$J:
                                  GVV DIAG   VAR  :J:  VAL  :I$45:
                                  REG X2 SET TO  :I$45:
31          SA3       K
                                  REG A3 SET TO  :A$K:
                                  GVV DIAG   VAR  :K:  VAL  :I$9:
                                  REG X3 SET TO  :I$9:
32          IX6       X1+X2
                                  REG X6 SET TO  :I$48:
33          SA6       A1
                                  REG A6 SET TO  :A$I:
                                  NEW VALUE IN VALS  :V1$I.EQ.I$48:
34          PX6       X1
                                  REG X6 SET TO  :I$3:
35          SA6       A6
                                  REG A6 SET TO  :A$I:
                                  NEW VALUE IN VALS  :V2$I.EQ.I$3:
36          IX6       X2/X3
                                  REG X6 SET TO  :I$5:
37          SA6       J
                                  REG A6 SET TO  :A$J:
                                  NEW VALUE IN VALS  :V1$J.EQ.I$5:
38          IX6       X1*X3
                                  REG X6 SET TO  :I$27:
39          SA6       J
                                  REG A6 SET TO  :A$J:
                                  NEW VALUE IN VALS  :V2$J.EQ.I$27:
40          BX7       X6
                                  REG X7 SET TO  :I$27:
41          SA7       ARY+X1
                                  REG A7 SET TO  :S$ARY(I$3):
                                  NEW VALUE IN VALS  :V1$ARY(I$3).EQ.I$27:
42    *LAST CONDITION
********* END OF CONTROL PATH 6 ********************

          FINAL VALUE STRING :V1$ARY(I$3).EQ.I$27:V2$J.EQ.I$27:V1$J.EQ.I$5:V2$I.EQ.I$3:V1$I.EQ.I

NO MORE PATHS IN THIS PROGRAM
```

Status Report of Program Verifier for SNOBOL source program

The abilities of the verifier at the present and procedures for input deck preparation are as follows;

1). Verifier first checks the syntax of input program, and provides error-message if any error is detected.

2). Verifier enumerates control paths of the input program. For each control path, verifier tries to verify the final assertions with respect to the initial assertions and statements in the control path itself. Only simple algebraically expressed assignment statements and conditional statements in the final assertions are verified at the present.

If a closed loop is found inside a control path, Verifier prints out the error-message together with the loop.

3). For each SNOBOL statement which has either successful or failure brahch condition, Verifier explicitely prints out which condition the current control path assumes. If additional information about the possible values of indirectly referenced variable is given, branching with indirect referencing is also possible.

4). Preparation of input deck.

a). comments------ start with * in the first column.

b). assertions --- start with */ in the first two columns.

c). Formula(s) defining an assertion inductively --- start with ** in the first two columns.

d). To supply informations about possible values of a indirectly referenced variable -- start with *$.

e). The syntax of assertion statements are the same as specified by the SNOBOL language, using a comma as the delimiter between assertions.

Assertions about logical conditions should be punched following the syntax of FORTRAN.

(S = 0)

SUM(α, β) means α = $\sum_{i=1}^{\beta} i$

```
      ** SUM(S + I,I) = SUM(S,I - 1)
      */ N .GE. 1
1        I = 1
2        S = 0
      */ N .GE. 1, I .LE. N, SUM(S,I - 1)
3   ADD  S = S + I
```

α + i =

```
4        I = I + 1
5        LE(I,N)                                   :S(ADD)
      */ SUM(S,I)
6        OUTPUT = S
```

```
***** ENUMERATION OF CONTROL PATHS WITH PRECONDITIONS.

--------------------------------------------------------------
*CONTROL PATH NO. 1
      ** SUM(S + I,I) = SUM(S,I - 1)
      */ N .GE. 1
1        I = 1
2        S = 0
      */ N .GE. 1, I .LE. N, SUM(S,I - 1)
--------------------------------------------------------------

VERIFICATION OF : N .GE. 1
         CORRESPONDING INITIAL ASSERTION : N .GE. 1

VERIFICATION OF : I .LE. N
         SUBSTITUTION I --- 1
         AFTER SUBSTITUTION --- 1 .LE. N
         CORRESPONDING INITIAL ASSERTION : N .GE. 1

VERIFICATION OF : SUM(S,I - 1)
```

```
--------------------------------------------------------------------
*CONTROL PATH NO.2
        */ N .GT. 1, I .LT. N, SUM(S,I - 1)
[illegible]
4               I = I + 1
5               ( I .LT. N )
        */ N .GT. 1, I .LT. N, SUM(S,I - 1)
--------------------------------------------------------------------

V[illegible]
        CORRESPONDING INITIAL ASSERTION : N .GT. 1

VERIFICATION OF : I .LE. N
        CORRESPONDING INITIAL ASSERTION : LT(I, N)

V[illegible]
```

```
--------------------------------------------------------------------
*CONTROL PATH NO.3
        */ N .GT. 1, I .LT. N, SUM(S,I - 1)
[illegible]
4               I = I + 1
5               ( I .GT. N )
        */ SUM(S,I)
--------------------------------------------------------------------

V[illegible]
```

```
*CONTROL PATH NO.4
[illegible]
6               OUTPUT = S
```

[illegible]

Edward M. Gould
9 December, 1971

USING THE PL/I PROGRAM VERIFICATION AID

The PL/I program verification aid is a SNOBOL 4 program designed to aid in the verification of correctness of a program written in PL/I. It is very simple to use, but a few constraints must be placed on the program to be verified. These constraints are such that existing programs will most likely not be in acceptable format. Even so, if they are kept in mind while writing a PL/I program, there is only a very limited number of features that cannot be either used directly or written in another form, using only allowable statements.

The constraints that must be imposed are as follows:

1. The only comments that are allowed are those to be used as assertions in verification.
2. Statements must be contained on one card, and must be terminated with a semicolon.
3. DO statements, ELSE clauses, BEGIN blocks, and internal procedures are not yet supported.
4. All declarations and similar non-executable statements must precede the first assertion, and the first assertion must precede the first executable statement.
5. The verification aid assumes that the program to be verified is correct and legal PL/I. Statements that are not PL/I statements, but are in the correct format, will be accepted, although their meaning is not defined. Some faults, such as attempting to transfer to an undefined label, will cause error termination of the verifier.

Actual use of the verification aid is quite simple and straightforward. The program to be verified is submitted as data to the verification program. The deck setup for the 6400 follows.

```
Job card
SNOBOL.
7-8-9
  verifier source deck
7-8-9
  program to be verified
6-7-8-9
```

Output from the verifier is also easy to use. The control paths through the data program are scanned from the assertion indicating the first control point until all paths have been indicated. Scanning then begins at the next control point, and so on until all possible paths are indicated. All statements before the first control point are skipped. A listing of the verification aid and a sample output may be fould in the accompanying sample run.

```
 GCD: PROCEDURE;
 /* M>0 N>0    */
 I=M;
 J=N;
 /* M>0 N>0 I>0 J>0 GCD(I,J)=GCD(M,N) */
L: IF I=J THEN GO TO U;
 IF I<J THEN GO TO V;
 I=I-J;
 GO TO L;
V: J=J-I;
 GO TO L;
 /* I=GCD(M,N) */
U: RETURN(I);
 END.
```

```
CONTROL PATH 1, BEGINS AT CONTROL POINT 1
                                 /* M>0 N>0    */
                                I=M
                                J=N
                                 /* M>0 N>0 I>0 J>0 GCD(I,J)=GCD(M,N) */

CONTROL PATH 2, BEGINS AT CONTROL POINT 2
                                 /* M>0 N>0 I>0 J>0 GCD(I,J)=GCD(M,N) */
                                (I=J)
                                 /* I=GCD(M,N) */

CONTROL PATH 3, BEGINS AT CONTROL POINT 2
                                 /* M>0 N>0 I>0 J>0 GCD(I,J)=GCD(M,N) */
                                ¬(I=J)
                                (I<J)
                                J=J-I
                                 /* M>0 N>0 I>0 J>0 GCD(I,J)=GCD(M,N) */

CONTROL PATH 4, BEGINS AT CONTROL POINT 2
                                 /* M>0 N>0 I>0 J>0 GCD(I,J)=GCD(M,N) */
                                ¬(I=J)
                                ¬(I<J)
                                I=I-J
                                 /* M>0 N>0 I>0 J>0 GCD(I,J)=GCD(M,N) */

CONTROL PATH 5, BEGINS AT CONTROL POINT 3
                                 /* I=GCD(M,N) */
                                RETURN(I)
                                *****  CONTROL RETURNS TO CALLING PROGRAM

PROGRAM ENDED   SCANNER EXECUTION COMPLETE
```

This writeup describes the program for determining program correctness of assembly programs written in Xerox Data Systems (XDS), formerly Scientific Data Systems (SDS), Extended Symbol for operation on XDS Sigma 2 and Sigma 3 computers. It defines limitations on the format of the Symbol program as imposed by the correctness program.

The general function of the program is to read in a syntactically correct symbol program and produce complete copies of all possible control paths (like branches of a tree) by selectively permuting the conditional branch instructions.

The symbol program statements must be placed in the following required order: The first card must have an asterisk in column one followed by an initial assertion concerning initial values. This card may be followed by any number of statements having no branch or transfer instructions. Following this is the intermediate assertion (asterisk in column one) concerning the status of the function being processed. The following statements contain labels and both conditional and unconditional branch statements which eventually determine the value of the function along with termination branch. After this is the final assertion on the value of the function followed by card containing the label of the termination branch.

Example

(1) * Initial assertion

(2) Statements contain no branch or transfer

(3) * Intermediate assertion

(4) Evaluation of function

(5) * Final assertion

(6) Termination label as specified in (4).

XDS SIGMA 2 EXTENDED SYMBOL INPUT PROGRAM

```
*  M>0, N>0
         LDA    M
         STA    I
         LDA    N
         STA    J
*  I>0, J>0, GCD(I,J)=GCD(M,N)
ONE      LDA    I
         SUB    J
         BAN    TWO
         BAZ    END
         LDA    I
         SUB    J
         STA    I
         B      ONE
TWO      LDA    J
         SUB    I
         STA    J
         B      ONE
*  I=GCD(M,N)
END
```

ABOVE PROGRAM CONDENSED WITH SEMI-COLONS

```
*  M>0, N>0

 LDA   M;  STA   I;  LDA   N;  STA   J;

*  I>0, J>0, GCD(I,J)=GCD(M,N)

ONE      LDA   I;  SUB   J;  BAN   TWO;  BAZ   END;  LDA   I;  SUB   J;  STA   I;  B     ONE;

TWO      LDA   J.  SUB   I.  STA   J.  B     ONE.

*  I=GCD(M,N)

END;
```

ABOVE PROGRAM BROKEN DOWN INTO ALL POSSIBLE CONTROL PATHS

```
                   *  M>0, N>0
PATH 1              AC = M.  I = AC.  AC = N.  J = AC.
                   *  I>0, J>0, GCD(I,J)=GCD(M,N)

                   *  I>0, J>0, GCD(I,J)=GCD(M,N)
PATH 2             AC = I.  AC = AC - J.  (AC & 0). AC = J.  AC = AC - I.  J = AC.
                   *  I>0, J>0, GCD(I,J)=GCD(M,N)

                   *  I>0, J>0, GCD(I,J)=GCD(M,N)
PATH 3             AC = I.  AC = AC - J.   (AC ≥ 0). (AC = 0).
                   *  I=GCD(M,N)

                   *  I>0, J>0, GCD(I,J)=GCD(M,N)
PATH 4             AC = I.  AC = AC - J.  (AC ≥ 0). (AC ≠ 0).    AC = I.  AC = AC - J.  I = AC;
                   *  I>0, J>0, GCD(I,J)=GCD(M,N)
```

Richard L. Harris
EECS 198-3, Prof. Maurer
Fall, 1971

TERM PROJECT WRITE-UP

This SNOBOL program accepts as input a series of ALGOL statements which must be syntactically correct. Statements may have any of the following forms:

(1) label part 'IF' condition 'THEN' action ;
(2) label part 'IF' condition 'THEN' action
'ELSE' alternative action;
(3) label part 'GOTO' label;
(4) label part variable := expression ;
(5) 'COMMENT' (any string of symbols not containing
;) ;

where

label part is a valid ALGOL label (may be null)
condition is an ALGOL boolean expression not involving
'IF'
action is either (2) or (3) where the label part is null
alternative action is the same form as action
variable is an ALGOL varible
expression is any valid ALGOL arithmetic expression not
involving 'IF'.

Any symbol valid as an equivalent of an ALGOL symbol on the ALGOL compiler at the Berkeley Computer Center (CDC 6400 only) is acceptable to this program. (example: the ALGOL symbol = may be represented as =, or 'EQ', or 'EQUAL' as is consistent with the ALGOL available on the 6400.

Output from the program is a listing of control paths from each assertion to the next one. (It is assumed that every 'COMMENT' is an assertion.) If the program transfers to a statement not in the input deck, either by a 'GOTO' or by "running off the end", the last line in the control path indicates that this is what has happened, rather than being an assertion.

If the ALGOL program part transfers to a statement previously listed in the current control path, without passsng by an assertion, an error message is printed , followed by those lines of the program which might loop back on themselves . At this point all attempts to trace control paths which start at the same assertion are abandoned and normal tracing resumes with the next assertion.

If an ALGOL statement is included which does not begin with 'IF' or "COMMENT' or 'GOTO' (or its equivalent, 'GO TO') and is not a simple variable assignment, it is treated as though it were.

Statements of the form (4) are listed in the control path in their original form, but with all blanks removed.

Statements of the form (1) or (2) cause the listing of the condition or its negation in parentheses. If the action or alternative action is taken, it is listed, without blanks, if it is of the form (4).

Statements of the form (3) cause no output, but serve to direct the SNOBOL program in its tracing of control.

Statements of the form (5) are listed, with blanks, and the ALGOL symbol 'COMMENT' replaced by the word assertion.

Note: a list of ALGOL symbols and acceptable equivalents is available in the 6400 System Messages, dated 6/18/69.

```
***** PROGRAM LISTING *****
≠COMMENT≠ ... ..
  I .= M .. J .= N ..
   ≠COMMENT≠ I ≠GT≠ 0 . J ≠GT≠ 0. GCD(I,J) = GCD(M,N) ..
L1.. ≠IF≠ I-J = 0 ≠THEN≠ ≠GOTO≠ L4 ..
≠IF≠ I-J ≠GT≠ 0 ≠THEN≠ ≠GOTO≠ L3..
J.= J-I.. ≠GOTO≠ L1..
L3.. I.= I - J.. ≠GOTO≠ L1..
    ≠COMMENT≠ I = GCD(M,N) ..   L4..
  THIS IS A DUMMY STATEMENT
```

```
***** PROGRAM CONTROL PATH *****
```

PATH 1

```
        ASSERTION  ≠GT≠ , N ≠GT≠ 0 ..
        I.=M
        J.=N
        ASSERTION  I ≠GT≠ 0. J ≠GT≠ 0. GCD(I.J) = GCD(M,N) ..
```

PATH 2

```
        ASSERTION  I ≠GT≠ 0, J ≠GT≠ 0. GCD(I,J) = GCD(M,N) ..
             (I-J= )
        ASSERTION  I = GCD(M,N) ..
```

PATH 3

```
        ASSERTION  I ≠GT≠ 0, J ≠GT≠ 0. GCD(I.J) = GCD(M,N) ..
             (I-J≠ ≠ )
             (I-J≠GT≠ )
        I.=I-J
        ASSERTION  I ≠GT≠ 0. J ≠GT≠ 0. GCD(I.J) = GCD(M,N) ..
```

PATH 4

```
        ASSERTION  I ≠GT≠ 0, J ≠GT≠ 0. GCD(I,J) = GCD(M,N) ..
             (I-J≠ ≠ )
             (I-J< )
        J.=J-I
        ASSERTION  I ≠GT≠ 0. J ≠GT≠ 0. GCD(I.J) = GCD(M,N) ..
```

PATH 5

```
        ASSERTION I = GCD(M.N) ..
        THISISADUMMYSTATEMENT
        ***THIS CONTROL PATH TERMINATED BY A TRANSFER TO A STATEMENT NOT IN THIS PROGRAM PART
```

EECS 198.3
FALL 1971

PAVEL STOFFEL

VERIFICATION PROGRAM FOR PROGRAMS WRITTEN IN ALGOL (USERS MANUAL)

I. Introduction, purpose, and function.

This program has been designed to be used and has been tested on the CAL CDC 6400 computer. It is a verification program. Its purpose is to aid the user in verifying that a program written by him in a subset of ALGOL is correct, namely that it does what it is supposed to do. This is accomplished in the following way: This program accepts as input a program written in a subset of ALGOL. At various points in the ALGOL program the user might insert COMMENT cards which contain certain assertions. Thse assertions must be valid during the execution of the ALGOL program when control passes through these points if the program under test is correct at all.

The verification program will analyze all possible ways the control can flow in the program under test. (Note that by "all possible ways the control can flow" does not mean an actual control flow when the program is executed given a set of data). The analysis consists of constructing what it is called "contol paths". A control path is a sequence of statements that are executed in the given order under certain conditions. A control path always starts when a COMMENT card is encountered and the given ASSERTION will be stated. Durint the control path if at any time one or more statements are executed because a certain condition is met then the condition is listed in brackets. This condition can be either the true or false case of a Boolean expression encountered in an IF statement. A control path is ended and the program goes into searching for another possible path whenever control flows into a COMMENT statement or there is a transfer to a statement which follows immediately a COMMENT statement. Thus, any control path will begin and end with an ASSERTION.

All what the program does is to list the ALGOL program which is under verification and as an output a numbered list of all possible control paths. It is left to the user to prove that given that the ASSERTION from the begining of any path is true, as well as the fact that the program variables are under the restrains stated by the Boolean expressions enclosed in parentheses, the execution of the statements of the path under consideration will have a total effect on the variables of the program such that the ASSERTION from the end of the path will hold true.

The program can detect if the input program contains an infinite loop. In this case an error message is printed and the program halts.

II. How to use the program, input format and restrictions.

The verification program has been written in SNOBOL, and it can be called either by SNOBOL. or XSNOBOL. control cards.

The input cards contain the ALGOL statements according to a free format (i.e. a statement can be extended on more cards if necessary or even two or more statements can be punched on the same card. Leading, trailing and inserted blanks can be used as desired. Though, no blanks can be inserted in the name of any label.). The last statement of the ALGOL program must be blank (i.e. empty).

The ALGOL subset accepted by this program is restricted by the following rules:

- arithmetic expressions should not contain constructions of the form:
  <if clause><simple arithmetic expression> _else_<arithmetic expression>
  (if they do analysis is done in one path. No different paths are constructed for different conditions of the Boolean expression(s)).

- Boolean expressions should not contain the construction of the form:
  <if clause><simple Boolean>_else_<Boolean expression>.

- no blocks and compound statements are allowed.

- no FOR statements are allowed.

- a statement inside an IF statement cannot have a label.

- a designational expression must be a label.

- no designational expression is allowed in an expression.

PATH.23 :

```
*COND13,COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*              [JA]=0,[JB]<0,
*              ASC(R,[.X.]),[R+[.X.]-1]≤[[IB]],[.X.]>0
34      LA34 LDA*  IB
35           STA*  ADD5
36*          [C] + 1 = 0
38*          [JB] + 1 ≠ 0
*COND13,COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*              [JA]=0,[JB]<0,
*              ASC(R,[.X.]),[R+[.X.]-1]≤[[IB]],[.X.]>0
```

PATH.24 :

```
*COND13,COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*              [JA]=0,[JB]<0,
*              ASC(R,[.X.]),[R+[.X.]-1]≤[[IB]],[.X.]>0
34      LA34 LDA*  IB
35           STA*  ADD5
36*          [C] + 1 ≠ 0
37*          [IB] + 1 = 0
*COND13,COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*              [JA]=0,[JB]<0,
*              ASC(R,[.X.]),[R+[.X.]-1]≤[[IB]],[.X.]>0
```

PATH.25 :

```
*COND13,COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                [JA]=0,[JB]<0,
*                ASC(3,[.X.]),[B+[.X.]-1]≤[[IB]],[.X.]>0
34      LAB4 LDA*  IB
35           STA*  ADC5
36*          [C] + 1 ≠ 0
37*          [IB] + 1 ≠ 0
38*          [JB] + 1 = 0
*COND14,COND PERM(NA,NB),[.X.]=NA+NB,[JA]=0,[JB]=0,
*                ASC(3,[.X.]),MERGE(NA,NB)
```

PATH.26 :

```
*COND13,COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                [JA]=0,[JB]<0,
*                ASC(3,[.X.]),[B+[.X.]-1]≤[[IB]],[.X.]>0
34      LAB4 LDA*  IB
35           STA*  ADC5
36*          [C] + 1 ≠ 0
37*          [IB] + 1 ≠ 0
38*          [JB] + 1 ≠ 0
*COND13,COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                [JA]=0,[JB]<0,
*                ASC(3,[.X.]),[B+[.X.]-1]≤[[IB]],[.X.]>0
```

The following program looks at the PDP-8 assembly language program of a simple version of Euclid's algorithm for finding the greatest common divisor of two integers M and N, and finds the control paths. Determining these paths is essential in the field of programing verification. In this particular program there are only two branching statements considered, SPA- skip on positive ACC , and SNA - skip on negative Acc. However the remaining skip functions can easily be implemented using techniques similar to those used in the SPA and SNA routines. The only restriction being that only one branch instruction is allowed per line. This program also changes DCA n into n = ACC, Acc = 0; TAD n into (XXXXXXXXX) ACC = ACC + n; and CMA into ACC = -ACC. The PDP-8 assembly program is inserted at the end of the SNOBOL program between the 7/8/9 card and the 6/7/8/9 card.

Alan Campbell

```
*ACC = 0, M.GT. 0, N.GT.0
            ACC = ACC +  M;  I= ACC; ACC = 0 ; ACC = ACC +  N;  J= ACC; ACC = 0 ;
* I .GT. 0, J .GT.0, GCD(I,4) = GCD(M,N), ACC = 0
L1,         ACC = ACC +  I; ACC = -ACC,; ACC = ACC +  J; SNA; JMP L4;
            SPA; JMP L3;  J= ACC: ACC = 0 ; JMP L1;
L3,         ACC = -ACC,.  I= ACC. ACC = 0 . JMP L1.
* I = GCD(M,N)
L4,
```

PATH1

```
*ACC = 0, M.GT. 0, N.GT.0
      ACC = ACC +  M;  I= ACC; ACC = 0 ; ACC = ACC +  N;  J= ACC; ACC = 0 ;
* I .GT. 0, J .GT.0, GCD(I,4) = GCD(M,N), ACC = 0
```

PATH2

```
* I .GT. 0, J .GT.0, GCD(I,4) = GCD(M,N), ACC = 0
    ACC = ACC +  I; ACC = -ACC,. ACC = ACC +  J. (ACC.NE.0).
      (ACC.GT.0):  J= ACC; ACC = 0 ;
* I .GT. 0, J .GT.0, GCD(I,4) = GCD(M,N), ACC = 0
```

PATH3

```
* I .GT. 0, J .GT.0, GCD(I,4) = GCD(M,N), ACC = 0
    ACC = ACC +  I; ACC = -ACC,; ACC = ACC +  J. (ACC.NE.0).
      (ACC.LE.0);
    ACC = -ACC,;  I= ACC; ACC = 0 ;
* I .GT. 0, J .GT.0, GCD(I,4) = GCD(M,N), ACC = 0
```

PATH4

```
* I .GT. 0, J .GT.0, GCD(I,4) = GCD(M,N), ACC = 0
    ACC = ACC +  I; ACC = -ACC,; ACC = ACC +  J; (ACC=0);
* I = GCD(M,N)
```

```
PATH.3 :

*COMB11,[.X.],CONC PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                        [JA]<0,[JB]<0,[IA]=A+[JA]+NA,[IB]=B+[JB]+NB,
*                        [.X.]=0 OR (ASC(R,[.X.]),[R+[.X.]-1]≤[[IA]],
*                                   [R+[.X.]-1]≤[[IB]],[.X.]>0)
12     LABEL LEAF  11
13•          [.X.] < [[IA]]
16           STAR  NOS
17•          [C] + ? = 0
19•          [JB] + ? ≠ 0
*COMB11,[.X.],CONC PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                        [JA]<0,[JB]<0,[IA]=A+[JA]+NA,[IB]=B+[JB]+NB,
*                        [.X.]=0 OR (ASC(R,[.X.]),[R+[.X.]-1]≤[[IA]],
*                                   [R+[.X.]-1]≤[[IB]],[.X.]>0)




PATH.4 :

*COMB11,[.X.],CONC PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                        [JA]<0,[JB]<0,[IA]=A+[JA]+NA,[IB]=B+[JB]+NB,
*                        [.X.]=0 OR (ASC(R,[.X.]),[R+[.X.]-1]≤[[IA]],
*                                   [R+[.X.]-1]≤[[IB]],[.X.]>0)
12     LABEL LEAF  12
13•          [.X.] < [[IA]]
16           STAR  NOS
17•          [C] + ? ≠ 0
19•          [IB] + ? = 0
*COMB11,[.X.],CONC PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                        [JA]<0,[JB]<0,[IA]=A+[JA]+NA,[IB]=B+[JB]+NB,
*                        [.X.]=0 OR (ASC(R,[.X.]),[R+[.X.]-1]≤[[IA]],
*                                   [R+[.X.]-1]≤[[IB]],[.X.]>0)
```

PATH.5 :

```
*CCND11,[.X.],COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                       [JA]<0,[JB]<0,[IA]=A+[JA]+NA,[IB]=B+[JB]+NB,
*                       [.X.]=0 OR (ASC(B,[.X.]),[R+[.X.]-1]≤[[IA]],
*                                        [R+[.X.]-1]≤[[IB]],[.X.]>0)
12      LABL LDA*  IB
13*          [.A.] < [[IA]]
16           STA*  ADDS
17*          [C] + 1 ≠ 0
18*          [IB] + 1 ≠ 0
19*          [JB] + 1 = 0
*CCND12,COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                 [JA]<0,[JB]=0,
*                 ASC(R,[.X.]),[R+[.X.]-1]≤[[IA]],[.X.]>0
```

PATH.6 :

```
*CCND11,[.X.],COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                       [JA]<0,[JB]<0,[IA]=A+[JA]+NA,[IB]=B+[JB]+NB,
*                       [.X.]=0 OR (ASC(R,[.X.]),[R+[.X.]-1]≤[[IA]],
*                                        [R+[.X.]-1]≤[[IB]],[.X.]>0)
12      LABL LDA*  IB
13*          [.A.] < [[IA]]
16           STA*  ADDS
17*          [C] + 1 ≠ 0
18*          [IB] + 1 ≠ 0
19*          [JB] + 1 ≠ 0
*CCND11,[.X.],COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                       [JA]<0,[JB]<0,[IA]=A+[JA]+NA,[IB]=B+[JB]+NB,
*                       [.X.]=0 OR (ASC(R,[.X.]),[R+[.X.]-1]≤[[IA]],
*                                        [R+[.X.]-1]≤[[IB]],[.X.]>0)
```

PATH.7 :

```
*CONDN2,[.X.],COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                        [JA]<0,[JB]<0,[IA]=A+[JA]+NA,[IB]=B+[JB]+NB,
*                        [.X.]=0 OR (ASC(B,[.X.]),[B+[.X.]-2]≤[[IA]],
*                                          (B+[.X.]-3]<[[IB]],[.X.]>0)
12      LAB1 LCA*   IB
13*          [.A.] = [[IA]]
28      LAB2 LCA*   IA
29           STA*   ADD5
30*          [0] + 2 = 0
32*          [JA] + 1 = 0
*CONDN3,COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                 [JA]=0,[JB]<0,
*                 ASC(B,[.X.]),[B+[.X.]-3]≤[[IB]],[.X.]>0
```

PATH.8 :

```
*CONDN1,[.X.],COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                        [JA]<0,[JB]<0,[IA]=A+[JA]+NA,[IB]=2+[JB]+NB,
*                        [.X.]=0 OR (ASC(B,[.X.]),[B+[.X.]-2]≤[[IA]],
*                                          [B+[.X.]-3]≤[[IB]],[.X.]≥0)
12      LAB1 LCA*   IB
13*          [.A.] = [[IA]]
28      LAB2 LCA*   IA
29           STA*   ADD5
30*          [C] + 2 ≠ 0
32*          [JA] + 1 ≠ 0
*CONDN1,[.X.],COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                        [JA]<0,[JB]<0,[IA]=A+[JA]+NA,[IB]=B+[JB]+NB,
*                        [.X.]=0 OR (ASC(B,[.X.]),[B+[.X.]-2]≤[[IA]],
*                                          [B+[.X.]-3]≤[[IB]],[.X.]>0)
```

```
PATH.9 :

*CONDL1,[.X.],COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                     [JA]<0,[JB]<0,[IA]=A+[JA]+NA,[IB]=B+[JB]+NB,
*                     [.X.]=0 OR (ASC(B,[.X.]),[B+[.X.]-1]≤[[IA]],
*                                   [B+[.X.]-1]≤[[IB]],[.X.]>0)
12      LAB1 LDA*  IB
13►          [.A.] = [[IA]]
28      LAB3 LDA*  IA
29           STA*  ADD5
30►          [0] + 1 ≠ 0
31►          [IA] + 1 = 0
*CONDL1,[.X.],COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                     [JA]<0,[JB]<0,[IA]=A+[JA]+NA,[IB]=B+[JB]+NB,
*                     [.X.]=0 OR (ASC(B,[.X.]),[B+[.X.]-1]≤[[IA]],
*                                   [B+[.X.]-1]≤[[IB]],[.X.]>0)
```

```
PATH.10 :

*CONDL1,[.X.],COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                     [JA]<0,[JB]<0,[IA]=A+[JA]+NA,[IB]=B+[JB]+NB,
*                     [.X.]=0 OR (ASC(B,[.X.]),[B+[.X.]-1]≤[[IA]],
*                                   [B+[.X.]-1]≤[[IB]],[.X.]>0)
12      LAB1 LDA*  IB
13►          [.A.] = [[IA]]
28      LAB3 LDA*  IA
29           STA*  ADD5
30►          [0] + 1 ≠ 0
31►          [IB] + 1 ≠ 0
32►          [JA] + 1 = 0
*CONDL3,COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                 [JA]=0,[JB]<0,
*                 ASC(B,[.X.]),[B+[.X.]-1]≤[[IB]],[.X.]>0
```

PATH.11 :

```
*CONC11,[.X.],COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                  [JA]<0,[JB]<0,[IA]=A+[JA]+NA,[IB]=B+[JB]+NB,
*                  [.X.]=0 OR (ASC(R,[.X.]),[R+[.X.]-1]≤[[IA]],
*                               [R+[.X.]-1]≤[[IB]],[.X.]>0)
12       LAB1 LDA*   IB
13▸           [.A.] = [[IA]]
28       LAB3 LDA*   IA
29            STA*   ADD5
30▸           [0] + 1 ≠ 0
31▸           [IA] + 1 ≠ 0
32▸           [JA] + 1 ≠ 0
*CONC11,[.X.],COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                  [JA]<0,[JB]<0,[IA]=A+[JA]+NA,[IB]=B+[JB]+NB,
*                  [.X.]=0 OR (ASC(R,[.X.]),[R+[.X.]-1]≤[[IA]],
*                               [R+[.X.]-1]≤[[IB]],[.X.]>0)
```

PATH.12 :

```
*CONC11,[.X.],COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                  [JA]<0,[JB]<0,[IA]=A+[JA]+NA,[IB]=B+[JB]+NB,
*                  [.X.]=0 OR (ASC(R,[.X.]),[R+[.X.]-1]≤[[IA]],
*                               [R+[.X.]-1]≤[[IB]],[.X.]>0)
12       LAB1 LDA*   IB
13▸           [.A.] > [[IA]]
28       LAB3 LDA*   IA
29            STA*   ADD5
30▸           [0] + 1 = 0
32▸           [JA] + 1 = 0
*CONC13,COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*              [JA]=0,[JB]<0,
*              ASC(R,[.X.]),[R+[.X.]-1]≤[[IB]],[.X.]>0
```

PATH.13 :

```
*COND13,[.X.],COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                          [JA]<0,[JB]<0,[IA]=A+[JA]+NA,[IB]=B+[JB]+NB,
*                          [.X.]=0 OR (ASC(R,[.X.]),[R+[.X.]-1]≤[[IA]],
*                                         [R+[.X.]-1]≤[[IB]],[.X.]>0)
12      LAB1 LDA*  IB
13•          [.A.] > [[IA]]
23      LAB3 LDA*  IA
29           STA*  ALD5
30•          [C] + 1 = 0
32•          [JA] + 1 ≠ 0
*COND11,[.X.],COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                          [JA]<0,[JB]<0,[IA]=A+[JA]+NA,[IB]=B+[JB]+NB,
*                          [.X.]=0 OR (ASC(R,[.X.]),[R+[.X.]-1]≤[[IA]],
*                                         [R+[.X.]-1]≤[[IB]],[.X.]>0)
```

PATH.24 :

```
*COND13,[.X.],COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                          [JA]<0,[JB]<0,[IA]=A+[JA]+NA,[IB]=B+[JB]+NB,
*                          [.X.]=0 OR (ASC(R,[.X.]),[R+[.X.]-1]≤[[IA]],
*                                         [R+[.X.]-1]≤[[IB]],[.X.]>0)
12      LAB1 LDA*  IB
13•          [.A.] > [[IA]]
23      LAB3 LDA*  IA
29           STA*  ALD5
30•          [C] + 1 = 0
31•          [IA] + 1 = 0
*COND12,[.X.],COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                          [JA]<0,[JB]<0,[IA]=A+[JA]+NA,[IB]=B+[JB]+NB,
*                          [.X.]=0 OR (ASC(R,[.X.]),[R+[.X.]-1]≤[[IA]],
*                                         [R+[.X.]-1]≤[[IB]],[.X.]>0)
```

PATH.15 :

```
*COND11,[.X.],COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                   [JA]<0,[JB]<0,[IA]=A+[JA]+NA,[IB]=B+[JB]+NB,
*                   [.X.]=0 OR (ASC(R,[.X.]),[R+[.X.]-1]≤[[IA]],
*                              [R+[.X.]-1]≤[[IB]],[.X.]>0)
12      LAB1 LCA*   IB
13•          [.A.] > [[IA]]
28      LAB3 LCA*   IA
29           STA*   /TOS
30•          [C] + 1 ≠ 0
31•          [IA] + 1 ≠ 0
32•          [JA] + 1 = 0
*COND13,COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*             [JA]=0,[JB]<0,
*             ASC(R,[.X.]),[R+[.X.]-1]≤[[IB]],[.X.]>0
```

PATH.16 :

```
*COND11,[.X.],COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                   [JA]<0,[JB]<0,[IA]=A+[JA]+NA,[IB]=B+[JB]+NB,
*                   [.X.]=0 OR (ASC(R,[.X.]),[R+[.X.]-1]≤[[IA]],
*                              [R+[.X.]-1]≤[[IB]],[.X.]>0)
12      LAB1 LCA*   IB
13•          [.A.] > [[IA]]
28      LAB3 LCA*   IA
29           STA*   /TOS
30•          [C] + 1 ≠ 0
31•          [IA] + 1 ≠ 0
32•          [JA] + 1 ≠ 0
*COND11,[.X.],COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                   [JA]<0,[JB]<0,[IA]=A+[JA]+NA,[IB]=B+[JB]+NB,
*                   [.X.]=0 OR (ASC(R,[.X.]),[R+[.X.]-1]≤[[IA]],
*                              [R+[.X.]-1]≤[[IB]],[.X.]>0)
```

```
PATH.17 :

*COND12,COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*            [JA]<0,[JB]=0,
*            ASC(2,[.X.]),[B+[.X.]-1]≤[[IA]],[.X.]>0
21      LAB2 LDA*  IA
22           STA*  ABBS
23>          [0] + 1 = 0
25>          [JA] + 1 = 0
*COND14,COND PERM(NA,NB),[.X.]=NA+NB,[JA]=0,[JB]=0,
*            ASC(2,[.X.]),MERGE(NA,NB)
```

```
PATH.18 :

*COND12,COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*            [JA]<0,[JB]=0,
*            ASC(2,[.X.]),[B+[.X.]-1]≤[[IA]],[.X.]>0
21      LAB2 LDA*  IA
22           STA*  ABDB
23>          [0] + 1 = 0
25>          [JA] + 1 ≠ 0
*COND12,COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*            [JA]<0,[JB]=0,
*            ASC(2,[.X.]),[B+[.X.]-1]≤[[IA]],[.X.]>0
```

PATH.19 :

```
*CCNDL2,COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                [JA]<0,[JB]=0,
*                ASC(P,[.X.]),[P+[.X.]-1]≤[[IA]],[.X.]>0
21      LAB2 LDA*   IA
22           STA*   ADDS
23*          [0] + 1 ≠ 0
24*          [IA] + 1 = 0
*CCNDL2,COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                [JA]<0,[JB]=0,
*                ASC(P,[.X.]),[P+[.X.]-1]≤[[IA]],[.X.]>0
```

PATH.20 :

```
*CCNDL2,COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                [JA]<0,[JB]=0,
*                ASC(P,[.X.]),[P+[.X.]-1]≤[[IA]],[.X.]>0
21      LAB2 LDA*   IA
22           STA*   ADDS
23*          [0] + 1 ≠ 0
24*          [IA] + 1 ≠ 0
25*          [JA] + 1 = 0
*CCNDL4,COND PERM(NA,NB),[.X.]=NA+NB,[JA]=0,[JB]=0,
*                ASC(P,[.X.]),MERGE(NA,NB)
```

PATH.21 :

```
*COND12,COND PERM([IA]+IA,[JB]+JB),[.X.]=[JA]+NA+[JB]+NB,
*             [JA]<0,[JB]=0,
*             ASC(P,[.X.]),[P+[.X.]-1]≤[[IA]],[.X.]>0
21     LAB2 LDA*   IA
22          STA*   ADD5
23▸         [C] + 1 ≠ 0
24▸         [IA] + 1 ≠ 0
25▸         [JA] + 1 ≠ 0
*COND12,COND PERM([IA]+IA,[JB]+JB),[.X.]=[JA]+NA+[JB]+NB,
*             [JA]<0,[JB]=0,
*             ASC(P,[.X.]),[P+[.X.]-1]≤[[IA]],[.X.]>0
```

PATH.22 :

```
*COND13,COND PERM([JA]+JA,[JB]+JB),[.X.]=[JA]+NA+[JB]+NB,
*             [JA]=0,[JB]<0,
*             ASC(P,[.X.]),[P+[.X.]-1]≤[[IB]],[.X.]>0
34     LAB4 LDA*   IB
35          STA*   ADD5
36▸         [C] + 1 = 0
38▸         [JB] + 1 = 0
*COND14,COND PERM(NA,NB),[.X.]=NA+NB,[JA]=0,[JB]=0,
*             ASC(P,[.X.]),MERGE(NA,NB)
```

INPUT DATA:

```
**********

≠COMMENT≠ M ≠GT≠ 0, N ≠GT≠ 0.,
    I.=M.,    J.=N.,
≠COMMENT≠ I ≠GT≠ 0, J ≠GT≠ 0, GCD(I,J)=GCD(M,N).,
L1.. ≠IF≠ I-J ≠EQ≠ 0 ≠THEN≠ ≠GO TO≠ L4.,
     ≠IF≠ I-J ≠LT≠ 0 ≠THEN≠ ≠GO TO≠ L2 ≠ELSE≠ ≠GO TO≠ L3.,
L2.. J.=J-I., ≠GO TO≠ L1.,
L3.. I.=I-J., ≠GO TO≠ L1.,
≠COMMENT≠ I=GCD(M,N)., L4.
```

OUTPUT:

```
******

PATH NO. 1:  ≠ASSERTION≠M ≠GT≠ 0, N ≠GT≠ 0;
                  I.=M;
                  J.=N;
             ≠ASSERTION≠I ≠GT≠ 0, J ≠GT≠ 0, GCD(I,J)=GCD(M,N);

PATH NO. 2:  ≠ASSERTION≠I ≠GT≠ 0, J ≠GT≠ 0, GCD(I,J)=GCD(M,N);
                       (I-J≠EQ≠ 0)
             ≠ASSERTION≠I=GCD(M,N);

PATH NO. 3:  ≠ASSERTION≠I ≠GT≠ 0, J ≠GT≠ 0, GCD(I,J)=GCD(M,N);
                       (I-J≠NE≠ 0)
                       (I-J ≠LT≠ 0)
                  J.=J-I;
             ≠ASSERTION≠I ≠GT≠ 0, J ≠GT≠ 0, GCD(I,J)=GCD(M,N);

PATH NO. 4:  ≠ASSERTION≠I ≠GT≠ 0, J ≠GT≠ 0, GCD(I,J)=GCD(M,N);
                       (I-J≠NE≠ 0)
                       (I-J ≠GE≠ 0)
                  I.=I-J;
             ≠ASSERTION≠I ≠GT≠ 0, J ≠GT≠ 0, GCD(I,J)=GCD(M,N);
```

Hideki Nakano
EECS 198-3

USER'S MANUAL

This program is to check the correctness of an ALGOL subprogram. An assertion (or condition) which will be used to check the program of its correctness is to be inserted as 'COMMENT' in the program. There is to be an initial assertion at the start of the subprogram, a final assertion at the end, and also, an intermediate assertion which is to be inserted whenever there is a value assignment.

The ALGOL subprogram to be run on this program must be runable on a computer. That is, its program must be in accordance with the ALGOL 60 report. There are certain additional limitations also. (1) The subprogram to be input must be within the block of 'BEGIN' and 'END' statements; that is, the statements 'BEGIN' and 'END' may not be in the program. (2) Boolean arguments may not be used. (3) 'THEN' must be followed by a 'GOTO' statement. (4) 'ELSE' is not permitted in the 'IF' statements.

When a subprogram of ALGOL, under these restrictions, is run on this program it will print out all the paths that are to be taken, with their assertions appearing at the begining and at the end of each paths. All of the conditional arguments will appear in parenthesis. Also 'EQ', 'GT', 'GE', 'LT', and 'LE' will be printed in the form of $=, >, \geq, <$, and $\leq$ respectively.

THIS IS THE PROGRAM TO BE TESTED

```
        ≠COMMENT≠ M ≠GT≠ 0, N ≠GT≠ 0.,
        I.=M., J.=N.,
        ≠COMMENT≠ I ≠GT≠ 0, J ≠GT≠ 0, GCD(I,J)=GCD(M,N).,
        L1.. ≠IF≠ I-J=0 ≠THEN≠ ≠GOTO≠L4.,
               ≠IF≠ I-J ≠GT≠ 0 ≠THEN≠ ≠GOTO≠ L3.,
               J.=J-I., ≠GOTO≠ L1.,
        L3.. I.=I-J., ≠GOTO≠ L1.,
        ≠COMMENT≠ I=GCD(M,N)., L4..
```

THESE ARE THE PATHS

```
PATH 1  ≠ASSERTION≠M≠GT≠0,N≠GT≠0.,
               I.=M.,
               J.=N.,
        ≠ASSERTION≠I≠GT≠0,J≠GT≠0,GCD(I,J)=GCD(M,N).,

PATH 2  ≠ASSERTION≠I≠GT≠0,J≠GT≠0,GCD(I,J)=GCD(M,N).,
               (I-J=0)
        ≠ASSERTION≠I=GCD(M,N).,

PATH 3  ≠ASSERTION≠I≠GT≠0,J≠GT≠0,GCD(I,J)=GCD(M,N).,
               (I-J≠NE≠0)
               (I-J>0)
               I.=I-J.,
        ≠ASSERTION≠I≠GT≠0,J≠GT≠0,GCD(I,J)=GCD(M,N).,

PATH 4  ≠ASSERTION≠I≠GT≠0,J≠GT≠0,GCD(I,J)=GCD(M,N).,
               (I-J≠NE≠0)
               (I-J<0)
               J.=J-I.,
        ≠ASSERTION≠I≠GT≠0,J≠GT≠0,GCD(I,J)=GCD(M,N).,
```

Jean-Yves Le Goic December 9 1971

PROGRAM VERIFIER FOR THE DAP-16 LANGUAGE

The Program Verifier is a program written in SNOBOL, which accepts as input DAP-16 programs with preconditions inserted.

Input data for the Program Verifier

The DAP-16 program to be verified may consist of:

1) Condition cards: They contain a star (*) in column 1, followed by COND*jx* ; *j* is an integer denoting the level of subsection and *x* is an optional identifier. The main section being verified is taken as level zero.

   If there is a controlled expression in a given section, this expression appears after COND*jx*, separated from this by a blank. The conditions follow; they are preceded either by ,GLOBAL, if they are global conditions or preconditions for this section, or by ,COND, if they are preconditions for the first executable statement of this section, or by both; they are separated by commas. On any such card, if the last non-blank character is a comma, a continuation card is to follow, which may be any card with a star in column 1 and is scanned from its first non-blank character.

2) Comment cards: They contain a star in column 1, but this star may not be followed by COND.

3) DAP-16 statements: the Program verifier accepts the following subset of the standard Honeywell 316/516 instruction repertoire:

   - Load and store instructions,
   - Arithmetic instructions,
   - Logical instructions,
   - Shift instructions,
   - five Control instructions, which are:

- CAS Compare
- IRS Increment, Replace and skip
- JMP Unconditional Jump
- NOP No Operation
- SZE Skip if [.A.] = 0

A memory reference instruction may use direct addressing, indexing, and indirect addressing.

Note: In the conditions as well as in the listing of the paths, a quantity between brackets represents the contents of the location whose address is the quantity itself : for example, [lab] represents the contents of location lab ; lab may be a Symbolic address, or an absolute address, or a hardware register. The accumulator is represented by .A. and the index register by .X. or location 0.

Output of the Program Verifier

The output of the Program Verifier consists of four parts :

1) Echo of the DAP-16 input program:

   Errors messages are given for syntax errors which prevent further processing of the DAP-16 program by the Program Verifier. These messages follow the statements which are incorrect and are indicated as ↑↑↑↑↑↑ ERROR IN STATEMENT ABOVE.

   If such messages are printed, the output will consist of this first part only.

2) Listing of the DAP-16 statements:

   Each executable statement is printed in this part. It is prefixed with an identification number which will later allow the user to know which statements have been taken in a given path.

3) Listing of the conditions:

   The conditions appearing in the DAP-16 program are listed in this part.

4) <u>Listing of the paths</u>:

All the paths found in the DAP-16 program are listed in this part. They appear in the following form:

- PATH.<u>n</u>
- Assertion at the beginning of the path (condition or precondition)
- Instructions executed in the path; they are prefixed by the identification of part 2.
- Conditions imposed by the program for this path; they are prefixed by the identification number of part 2, followed by an arrow. For example, if, in part 2, the statement IRS BETA appears as 13 IRS BETA

  it will appear as

  13→ [BETA] + 1 = 0

  or 13→ [BETA] + 1 ≠ 0

  according to the path which is taken.
- The assertion at the end of the path.

Diagnostics are given in this part for loops which are not broken by an assertion, and may therefore never terminate. They follow the incorrect statement and have the following form:

↑↑↑↑↑↑ ERROR 3 IN PATH ABOVE: PATH CONTAINS A LOOP.

This indicates that such a statement should be preceded by an assertion. The listing of the other paths is not affected by this error, but an error message will be printed before the listing of the paths for each occurrence of such an error , in the following form: ERROR 3 IN PATH.<u>n</u> AT STATEMENT <u>x</u>.

INPUT PROGRAM IMAGE ***********

```
*COND1,GLOBAL [ADD1]=A,[ADD2]=NA,[ADD3]=B,[ADD4]=NB,[ADD5]=R,
*                    ASC(A,NA),ASC(B,NB),NA>0,NB>0
      LDA   ADD1                 *SET [IA] TO A
      STA   IA                        *
      LDA   ADD2                 *SET [JA] TO -NA
      TCA                             *
      STA   JA                        *
      LDA   ADD3                 *SET [IB] TO B
      STA   IB                        *
      LDA   ADD4                 *SET [JB] TO -NB
      TCA                             *
      STA   JB                        *
      LDX   =0
*COND11,[.X.],COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                   [JA]<0,[JB]<0,[IA]=A+[JA]+NA,[IB]=B+[JB]+NB,
*                   [.X.]=0 OR (ASC(R,[.X.]),[R+[.X.]-1]≤[[IA]],
*                    [R+[.X.]-1]≤[[IB]],[.X.]>0)
LAB1 LDA*  IB                    *COMPARE [[IB]] AND [[IA]]
     CAS*  IA                         *
     JMP   LAB3                  *IF [[IB]]>[[IA]],THEN JUMP TO LAB3
     JMP   LAB3                       *
     STA*  ADD5                  *ELSE,STORE [[IB]] AT LOC. R+[.X.]
     IRS   0                     *INCREMENT [.X.]
     IRS   IB                    *INCREMENT [IB]
     IRS   JB                    *INCREMENT [JB]
     JMP   LAB1                  *IF [JB]≠0, JUMP TO LAB1
*COND12,COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                  [JA]<0,[JB]=0,
*                  ASC(R,[.X.]),[R+[.X.]-1]≤[[IA]],[.X.]>0
LAB2 LDA*  IA                    *STORE [[IA]] AT R+[.X.]
     STA*  ADD5                       *
     IRS   0                     *INCREMENT [.X.]
     IRS   IA                    *INCREMENT [IA]
     IRS   JA                    *INCREMENT [JA]
     JMP   LAB2                  *IF [JA]≠0, JUMP TO LAB2
     JMP   LAB5                  *ELSE, JUMP TO LAB5
LAB3 LDA*  IA                    *STORE [[IA]] AT R+[.X.]
     STA*  ADD5                       *
     IRS   0                     *INCREMENT [.X.]
     IRS   IA                    *INCREMENT [IA]
     IRS   JA                    *INCREMENT [JA]
     JMP   LAB1                  *IF [JA]≠0, JUMP TO LAB1
*COND13,COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                  [JA]=0,[JB]<0,
*                  ASC(R,[.X.]),[R+[.X.]-1]≤[[IB]],[.X.]>0
LAB4 LDA*  IB                    *STORE [[IB]] AT R+[.X.]
     STA*  ADD5                       *
     IRS   0                     *INCREMENT [.X.]
     IRS   IB                    *INCREMENT [IB]
     IRS   JB                    *INCREMENT [JB]
     JMP   LAB4                  *IF [JB]≠0, JUMP TO LAB4
*COND14,COND PERM(NA,NB),[.X.]=NA+NB,[JA]=0,[JB]=0,
```

```
*                       ASC(N,I,X,1),MERGE(KA,IP)
LAB5 NOP
     END
```

PROGRAM [illegible]

```
          LDA   ADD1
          STA   IA
          LDA   ADD2
          TCA
          STA   JA
          LDA   ADD3
          STA   IB
          LDA   ADD4
          TCA
          STA   JB
          LDX   =0
LAB1      LDA*  IB
          CAS*  IA
          JMP   LAB3
          JMP   LAB2
          STA*  ADD5
          IRS   0
          IRS   IB
          IRS   JB
          JMP   LAB1
LAB2      LDA*  IA
          STA*  ADD5
          IRS   0
          IRS   IA
          IRS   JA
          JMP   LAB1
          JMP   [illegible]
LAB3      LDA*  IA
          STA*  ADD5
          IRS   0
          IRS   IA
          IRS   JA
          JMP   [illegible]
LAB4      LDA*  IB
          STA*  ADD5
          IRS   0
          IRS   IB
          IRS   JB
          JMP   LAB4
LAB5      NOP
          END
```

PROGRAM CONDITIONS ************

```
C.1 :
*COND0,GLOBAL [ADD1]=A,[ADD2]=NA,[ADD3]=B,[ADD4]=NB,[ADD5]=R,
*                ASC(A,NA),ASC(B,NB),NA>0,NB>0
C.12 :
*COND11,[.X.],COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                       [JA]<0,[JB]<0,[IA]=A+[JA]+NA,[IB]=B+[JB]+NB,
*                       [.X.]=0 OR (ASC(R,[.X.]),[R+[.X.]-1]≤[[IA]],
*                                   [R+[.X.]-1]≤[[IB]],[.X.]>0)
C.21 :
*COND12,COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                 [JA]<0,[JB]=0,
*                 ASC(R,[.X.]),[R+[.X.]-1]≤[[IA]],[.X.]>0
C.34 :
*COND13,COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                 [JA]=0,[JB]<0,
*                 ASC(R,[.X.]),[R+[.X.]-1]≤[[IB]],[.X.]>0
C.40 :
*COND14,COND PERM(A,B),[.X.]=NA+NB,[JA]=0,[JB]=0,
*                 ASC(R,[.X.]),MERGE(NA,NB)
```

PRINTING OF THE PATHS *********

PATH.1 :

```
*COND0,GLOBAL [ADD1]=A,[ADD2]=NA,[ADD3]=B,[ADD4]=NB,[ADD5]=R,
*                   ASC(A,[A]),ASC(B,NB),NA>0,NB>0
1            LDA    ADD1
2            STA    IA
3            LDA    ADD2
4            TCA
5            STA    JA
6            LDA    ADD3
7            STA    IB
8            LDA    ADD4
9            TCA
10            STA    JB
11            LDX    =0
*COND11,[.X.],COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                             [JA]<0,[JB]<0,[IA]=A+[JA]+NA,[IB]=B+[JB]+NB,
*                             [.X.]=0 OR (ASC(R,[.X.]),[R+[.X.]-1]≤[[IA]],
*                                               [R+[.X.]-1]≤[[IB]],[.X.]>0)
```

PATH.2 :

```
*COND11,[.X.],COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                             [JA]<0,[JB]<0,[IA]=A+[JA]+NA,[IB]=B+[JB]+NB,
*                             [.X.]=0 OR (ASC(R,[.X.]),[R+[.X.]-1]≤[[IA]],
*                                               [R+[.X.]-1]≤[[IB]],[.X.]>0)
12      [AB] LDA*   IB
13*          [.A.] < [[IA]]
16           STA*   ADD5
17*          [0] + 1 = 0
19*          [JB] + 1 = 0
*COND12,COND PERM([JA]+NA,[JB]+NB),[.X.]=[JA]+NA+[JB]+NB,
*                   [JA]<0,[JB]=0,
*                   ASC(R,[.X.]),[R+[.X.]-1]≤[[IA]],[.X.]>0
```