NEW ASSERTION CONCEPTS FOR SELF-METRIC SOFTWARE VALIDATION

LEON G STUCKI
MCDONNELL DOUGLAS ASTRONAUTICS COMPANY
HUNTINGTON BEACH, CALIFORNIA

GARY L FOSHEE
MCDONNELL DOUGLAS AUTOMATION COMPANY
HUNTINGTON BEACH, CALIFORNIA

Abstract

Automated tools and structured programming techniques are in use on a variety of scientific and business application programming projects within the McDonnell Douglas Corporation. An examination of the resulting programs reveals certain development and maintenance characteristics that suggest new and very interesting applications for automated tools.

An extension of PET (a currently operational McDonnell Douglas validation tool for FORTRAN) to include a user embedded assertion capability offers a step in the direction of automatically verifying the dynamic execution of programs. A user-oriented local and global assertion capability is introduced and its implementation is discussed.

Application of these tools within a well-conceived structured programming environment offers a positive step forward in the development of more reliable software systems.

Introduction

A traditional hardware approach to equipment monitoring uses the insertion of "probes" or instrumentation into the device being monitored. These probes gather pertinent statistics about device operation in a typical environment. Recently, this instrumentation approach has been applied to software monitoring.[1] Software probes in the form of source language statements are inserted into the source code to gather statistics during program execution. These probes can provide a wide spectrum of internal measurements. The notion of building self-metric (self-measuring) software has been introduced previously; however, a significant enhancement and refinement of this capability is presented in this paper.[2,3]

A number of systems have been designed to provide automated program analysis. To the best of the authors' knowledge the first attempts at the automatic analysis of dynamic program behavior began at UCLA in 1967 under Estrin.[1,4,5,6] Knuth and Ingnalls at Stanford built and made extensive use of a FORTRAN Execution Time Estimator dubbed FETE in 1970.[7,8] Brown, Carey and Paige have also built a number of automated tools for FORTRAN.[9-14]

All of these previously mentioned tools have taken a philosophy of measuring the thoroughness with which the software has been tested in an after-the-fact sense. These tools have been applied primarily to programs during final developmental testing and validation testing.

Program Evaluator and Tester (PET) System[15]

The McDonnell Douglas PET system was also designed to be used as a testing tool. In addition to the statement execution count statistics common to all of the previously mentioned systems, the PET system provides the following capabilities:

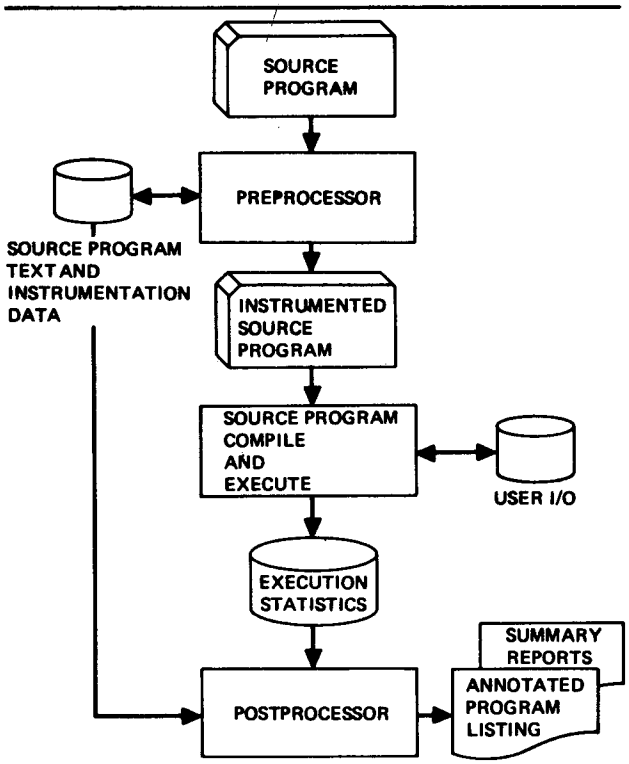
- Detailed branch or transfer of control counts.
- Minimum/maximum range information collected for each assignment and Do-loop control variable.
- An optional first/last value for each assignment statement.

These self-metric techniques, together with a syntactic standard checking capability, have proven extremely valuable for FORTRAN debugging and conversion efforts.

After having used the current PET system for several years, a new series of automated functions is now being designed and implemented. The most significant factor to observe is the conscious attempt to incorporate design verification criteria directly into evolving systems through a powerful assertion capability described in the remainder of this paper. The scope of the assertions presented encompasses the entire life cycle of a programming system from initial program design through operation and maintenance.

Program Instrumentation

A Preprocessor examines a source program and inserts additional source statements to gather pertinent statistics during program execution . A Postprocessor matches statistics generated during program execution with individual source program statements to produce an annotated program listing and summary report.



Three types of automatic instrumentation have been identified for programs:

1. Monitoring source statement execution and branch conditions.
2. Verification of assertions on data characteristics and program behavior.
3. Monitoring range of values assumed by scalar variables, arrays, and subscripts.

Source Statement Execution and Branch Conditions

Statement execution and branch monitoring are functions implemented in PET. [2,3,15] These monitoring functions identify which source statements have been executed and which transfers have been taken, thus showing the degree to which the program has been exercised. If all statements in a program are not executed during a test case or series of test cases, it is then known that correct execution of the program has not been fully demonstrated. This does not mean that the program is incorrect; the correctness has just not been demonstrated.

Verification of Assertions on Data Characteristics and Program Behavior

The ability of the user to establish his own assertions at key points within his algorithms in the languages he is now using, and to obtain dynamic analysis and feedback on the validity of his assertions, offers a practical step toward the design and implementation of more reliable software.

The proposed assertion and monitor capability will provide for two levels of control. Global assertions and monitor commands will be located with the declaratives and have effect over the length of their enclosing module or block. Local assertions will be position dependent and will consist of any legal logical expression of the host language. While it is acknowledged that global assertions could be replaced by numerous local assertions, human factors motivate the inclusion of these additional global assertions.

The assertions are transparent to the normal language compilers and must be preprocessed in order for dynamic execution checking to take place. The preprocessor will instrument the assertions by augmenting the original source program with self-metric (self-measuring) instrumentation.[4]

The use of the assertion capabilities will be emphasized throughout all phases of program development. Designers will be encouraged to include as assertions all critical requirements to be placed upon the key program algorithms. Interface requirements between system modules can be documented and validated through the use of appropriate assertions. Quality control and configuration management can be analyzed and enhanced by means of assertions.

Local assertions

Local assertions may appear as comments at any point within the executable program code. They serve to enhance the documentation of critical algorithms during regular program development while later on providing meaningful checkpoints for the debugging, validation and maintenance of complex systems.

The formats of the local assertions include:

```
ASSERT (extended logical expression)
  [HALT ON n [VIOLATIONS]]

ASSERT ORDER (array cross-section)
  [{ASCENDING | DESCENDING}] [HALT ON n [VIOLATIONS]]
```

These assertions will be placed in the comment field provided by each language for which this capability is to be implemented. The HALT option will stop program execution if n assertion violations occur. The ASSERT ORDER statement checks the table or array cross-section values to verify that they are in the selected (ascending or descending)

sequence. An example from a complete program in the Appendix follows:

```
  .
  .
00040            IF MOVE.NE.10 THEN
00041              BOARD(MOVE) = HUMAN
00042              TURNS = TURNS+1
  .
  .
00045  C         ASSERT (MOVE .LT. 9)  HALT ON 10
  .
  .
```

Structured FORTRAN Example

In this example the Preprocessor will insert instrumentation to count the number of times the assertion test was made and record any violations of the assertion. Thus, on the source listing produced by the Postprocessor, this assertion statement could have the following statistics associated with it:

```
                                                    EXECUTION
                                                      COUNT          SPECIFIC EXECUTION DATA

00045  C            ASSERT (MOVE .LT. 9)  HALT ON 10      9    ASSERTION VIOLATIONS          2
                                                              EXEC NUMBER              VALUE
                                                                        6                  9
                                                                        7                  9
```

Note: If the character ':' (colon) is not available on specific machine, another suitable character can be substituted.

Examples of the use of array cross-sections in extended logical expressions include the following: (assume an array A(10,20) has been defined)

```
(a)    ASSERT    (A(*,3) .LT. 10)  HALT ON 6
(b)    ASSERT    (A(2:6,2:10) . NE . 0)
(c)    ASSERT    (A(*,*) .GT. 0)
```

In (a), the value of each array element whose second subscript=3 is checked and reported as a violation if its value is not less than 10. Ten array values will be checked in all. Any number of assertion violations within an array cause the array to be counted as a single assertion violation. Thus, the HALT ON 6 parameter is concerned only with the number of array assertion violations and not with the number of violations within the array.

In (b), only array values within the specified subscript ranges are checked for an assertion violation. In (c) all array values are checked for an assertion violation. The ASSERT ORDER statement checks the sequence of array values as follows:

```
ASSERT ORDER (A(*,3)) ASCENDING
```

For an array A(10,20), the following assertion violation summary illustrates the type of information traced for a violation:

```
EXECUTION
  COUNT          SPECIFIC EXECUTION DATA

  229    ASSERTION VIOLATIONS              1
           EXEC NUMBER  SEQUENCE  SNAPSHOT VALUE
                 18       A(7,3)              6
                          A(8,3)            100 *
                          A(9,3)              8
```

This means an assertion violation occurred at the sixth and seventh execution of the assertion statement. A snapshot is taken of all pertinent variable values associated with the violation when the trace mode is specified.

Motivated by a need to make assertions about arrays as well as scalars, the following notation has been adopted.

Array Notation for Assertions

Two areas of concern immediately arise when discussing data arrays namely, array indices and array values. Thus, if one is monitoring program behavior, it is not enough to monitor array values alone, since program logic is invariably concerned with where these values are stored within the array.

The approach is to generalize the assertion and monitor capabilities to include data arrays. Array notation is as follows:

Assume an array of the form $A(I_1,I_2,I_3 \ldots I_n)$. References to specific subsets of array values or array indices are indicated by $A(I_1',I_2,I_3' \ldots I_n')$, where I_i' is a subrange of I_i. This notation is position dependent; i.e., if I_2 is not referenced, its position must be indicated by an asterisk (*), as in $A(I_1',*,I_3' \ldots I_n')$. The format of each I_i' is $\ell:\mu$ where $\ell<\mu<$ I_i. (see note below). If $\ell=\mu$ $\ell:\mu$ may be replaced by μ, as $A(\ell_1:\mu_1,\mu_2,\ldots)$. Thus, for A(10,20) we might reference

```
A(5,10:15)
A(*,3)
A(2:5,*)
A(2:6,2:10)
```

A TRACE statement is available to allow the user to control the number of execution snapshots reported for local assertion violations.

The format of the TRACE statement is:

TRACE [{ FIRST / LAST / OFF }] n [VIOLATIONS]

If a TRACE statement is not coded in the source program, execution counts for local assertion violations will not be

reported. If TRACE statements are coded, the first TRACE statement encountered causes the first (or last) n violations of subsequent local assertions to include snapshot information. Any subsequent TRACE statements encountered reset the values specified by the previous TRACE statement. TRACE OFF halts the reporting of execution snapshots for local assertions until the next TRACE statement is encountered.

Global Assertions & Monitor Commands

Global assertions and monitor commands are designed to extend the capacity to inspect and monitor the behavior of entire program modules. Global assertions check on the enforcement of a specific set of logical requirements while monitor commands gather data on actual program behavior.

Global assertions appear in the declarative section of the program module.

Formats include:

ASSERT RANGE (list of variables) (min, max)

ASSERT VALUES (list of variables) (list of legal values)

ASSERT VALUES (list of variables) NOT (list of illegal values)

ASSERT SUBSCRIPT RANGE (list of array specifications)

ASSERT NO SIDE EFFECTS (parameter list)

The RANGE statement examines each specified variable as its value changes and reports those new values which fall outside the specified min-max range.

The VALUES statement inspects each specified variable as its value changes and reports when: (option 1) the new value is not one of the specified legal values, or (option 2) the new value assumes a specified illegal value.

The ASSERT SUBSCRIPT RANGE statement verifies that array subscripts fall within a specified range whenever the array is referenced during program execution. It should be noted that this statement provides a means for checking portions of arrays as well as normal upper and lower bounds. For this reason, it is more powerful than the PL/I type ON SUBSCRIPT RANGE check.

The ASSERT NO SIDE EFFECTS statement reports any parameters of a call statement that change their value as a result of the call.

Instrumentation will be inserted into the source program by the preprocessor to accumulate the following statistics relative to assertion violations:

(1) Identify the statement that caused the assertion violation. For that statement an execution count and violation execution counts identical to those obtained for local assertions are reported.

(2) The actual value that caused the violation. This value is linked to the statistics identified in (1) above.

A GLOBAL TRACE statement will be available to allow the user to control the number of execution counts and associated data values reported for global assertion violations. The format of this statement is:

GLOBAL TRACE [[{ FIRST / LAST / OFF }] n] [VIOLATIONS]

Some FORTRAN examples follow:

```
                .
                .
                .
20              DIMENSION A(10,20)
21      C       GLOBAL TRACE 10 VIOLATIONS
22      C       ASSERT RANGE (I,J,K,L) (0,100)
23      C       ASSERT RANGE (II,LL) (-10,10)
24      C       ASSERT VALUES (KK,NN) (2,4,6,8,10)
25      C       ASSERT SUBSCRIPT RANGE (A(*,3))
26      C       ASSERT NO SIDE EFFECTS (X,Y,Z)
27      C       MONITOR RANGE FIRST LAST (K,NN)
28      C       MONITOR SUBSCRIPT RANGE (A)
                .
                .
                .
102             K = K + 1
103             II = A(L,J) + LL
                .
                .
                .
234             K = A(J,K) + I*100
235             II = II + 2
236             NN = KK*(I-J)
                .
                .
                .
300             CALL ROUTINEX(X,Y)
                .
                .
                .
```

If assertion violations occurred in this example, the following statistics are indicative of what would be reported by the Postprocessor:

```
ANNOTATED PROGRAM LISTING                EXECUTION COUNT   SPECIFIC EXECUTION DATA

102          K = K + 1                        511          ASSERTION VIOLATIONS          1
                                                           ASSERT RANGE (I,J,K,L) (0,100)
                                                              EXEC NUMBER          VALUE
                                                                    10             101

103          II = A(L,J) + LL                 511          ASSERTION VIOLATIONS          3
                                                           ASSERT RANGE (II,LL) (-10,10)
                                                              EXEC NUMBER          VALUE
                                                                    22              20
                                                           ASSERT SUBSCRIPT RANGE (A(*,3))
                                                              EXEC NUMBER          VALUE
                                                                     5           A(12,3)
                                                                   105           A(1,4)

234          K = A(J,K) + I*100               125          ASSERTION VIOLATIONS          4
                                                           ASSERT RANGE (I,J,K,L) (0,100)
                                                              EXEC NUMBER          VALUE
                                                                    52             101
                                                                    53             102
                                                           ASSERT SUBSCRIPT RANGE (A(*,3))
                                                              EXEC NUMBER          VALUE
                                                                    52           A(5,4)
                                                                    53           A(6,4)

235          II = II + 2                      125          ASSERTION VIOLATIONS          1
                                                           ASSERT RANGE (II,LL) (-10,10)
                                                              EXEC NUMBER          VALUE
                                                                    50              12

236          NN = KK*(I-J)                     38          ASSERTION VIOLATIONS          1
                                                           ASSERT VALUES (KK,NN) (2,4,6,8,10)
                                                              EXEC NUMBER          VALUE
                                                                    20              7

300          CALL ROUTINEX(X,Y)                53          ASSERTION VIOLATIONS          1
                                                           ASSERT NO SIDE EFFECTS (X,Y,Z)
                                                                        VALUE OF CALL PARM X
                                                              EXEC NUMBER  BEFORE CALL  AFTER CALL
                                                                    30          -10          -20
```

These statistics are reported in an orderly fashion on the final program listing (see Appendix for sample output).

A HALT statement is available to stop program execution if a specified number of global assertion violations occur. The format of this statement is:

HALT ON n [VIOLATIONS]

The program halts with appropriate messages if any global assertion is violated n times. This statement is placed in the declarative section of the program. The monitor capability is covered in the next section.

Monitoring Range of Values Acquired By Variable Quantities

The format of these monitor statements are:

MONITOR [{NUMERIC / CHARACTER}] [RANGE]

FIRST[n VALUES]

LAST [n VALUES]

[{(list of variables) / ALL}]

MONITOR SUBSCRIPT RANGE

[{(list of array names) / ALL}]

The NUMERIC or CHARACTER option apply principally to FORTRAN programs and tells the Preprocessor the type of variable to be monitored.

For each reference to the variable in a source statement, the RANGE parameter monitors the minimum, maximum values assigned to the specified variables during instrumented program execution.

In a similar manner FIRST and LAST monitor the first and last values obtained by the specified variable.

The ALL parameter for non-array variables causes all variables to be monitored. The ALL parameter for arrays causes all array values to be monitored.

Some examples:

```
(a)   C      MONITOR RANGE FIRST LAST ALL            (FORTRAN)
(b)   C      MONITOR CHARACTER RANGE (XVAR, YVAR)    (FORTRAN)
(c)   /*     MONITOR RANGE (A(*,3))          */      (PL/I)
(d)   /*     MONITOR SUBSCRIPT RANGE (A,B,C) */      (PL/I)
(e)   "MONITOR FIRST 5 VALUES (INDEX)"               (JOVIAL)
(f)   "MONITOR FIRST LAST 2 VALUES (J)"              (JOVIAL)
```

In (a) the minimum, maximum, first, and last values acquired by every non-array variable in the program module are reported at each point where a new value is set into a monitored variable.

In (b) the minimum and maximum character values of variables XVAR and YVAR are reported for each point at which they are set.

In (c) the minimum and maximum values for all array A elements with second subscript = 3 are reported.

In (d) the minimum and maximum values of all subscript references to arrays A,B, and C are reported.

In (e) the first 5 values assigned to INDEX will be saved for each point at which INDEX is set.

In (f) the first value and last 2 values for J will be reported for each point at which it is set.

The incorporation of an assertion capability in existing language support systems appears to offer a sophisticated yet practical approach to the validation task on today's programs while at the same time serving as a powerful capability for future system analysts in designing higher quality software.

Conclusions

Much ground still remains to be covered in the quest for producing more reliable software systems. A set of powerful new assertion concepts was introduced for software verification and validation. The implementation of these assertions is achieved through the synthesis of a new class of self-metric software.

The assertion concepts presented in this paper are intended for incorporation into currently existing languages. They serve to improve program documentation by conveying critical design verification criteria while at the same time imposing no undo penalties to early developmental phases of the software life cycle. They are presented in an attempt to bridge the gap, to some extent, between what today's tool builders and today's program provers are currently doing.

References

1. Estrin, G., et al., "SNUPER COMPUTER - A Computer in Instrumentation Automation," AFIPS Spring Joint Computer Conference 1967.

2. Stucki, L.G., "A Prototype Automatic Program Testing Tool," AFIPS Fall Joint Computer Conference, Anaheim, Calif., December 1972.

3. Stucki, L.G., "Automatic Generation of Self-Metric Software", Proceedings of the 1973 IEEE Symposium on Computer Software Reliability, New York City, April 30-May 2, 1973.

4. Russell, E.C., and Estrin, G., "Measurement Based Automatic Analysis of FORTRAN Programs," AFIPS Spring Joint Computer Conference, 1969.

5. Bussell, B. and Koster, R.A., "Instrumenting Computer Systems and Their Programs," AFIPS Fall Joint Computer Conference, 1970.

6. Cerf, V.G., "Measurement of Recursive Programs," PhD Thesis, School of Engineering and Applied Science, University of California, Los Angeles, Calif. (May 1970), Report No. 70-43.

7. Knuth, D.E., "An Empirical Study of FORTRAN Programs," Stanford Artificial Intelligence Project, Computer Science Department, Stanford University, Report No. STAN-CS-186.

8. Ingnalls, Daniel H.H., "FETE A FORTRAN Execution Time Estimator", Computer Science Department, Stanford University, (February 1971), STAN-CS-71-204.

9. Brown, J.R., et al., "Automated Software Quality Assurance: A Case Study of Three Systems", ACM SIGPLAN Symposium on Computer Program Test Methods, Chapel Hill, N.C., June 21-23, 1972.

10. Brown, J.R. and Hoffman, R.H., "Evaluating the Effectiveness of Software Verification - Practical Experience with an Automated Tool, "AFIPS Fall Joint Computer Conference, Anaheim, Calif., December 1972.

11. Krause, K.W., Smith, R.W., and Goodwin, M.A., "Optimal Software Test Planning Through Automated Network Analysis", 1973 IEEE Symposium on Computer Software Reliability, New York City, April 30-May 2, 1973.

12. Carey, L.J., "Qualifier Users Manual", Computer Systems Analyst, Inc., CSA-DL-0028, April 1974.

13. Paige, M.R., and Balkovich, E.E., "On Testing Programs", 1973 IEEE Symposium on Computer Software Reliability, New York City, April 30-May 2, 1973.

14. Paige,M.R., and Benson, J.P., "The Use of Software Probes in Testing FORTRAN Programs", Computer,IEEE Computer Society, Volume 7, Number 7, July 1974, pp40-47.

15. Boettcher, C.B., "Program Evaluator and Tester", PET CDC Users Manual, McDonnell Douglas Automation Company, Document No. M2085074, 1974.

APPENDIX

SAMPLE OUTPUT FORMATS

```
                                                              DATE 01/31/75          PAGE    1

                                                                     EXECUTION
                    ANNOTATED PROGRAM LISTING                          COUNT         SPECIFIC EXECUTION DATA

                   .
                   .
                   .
20                 DIMENSION A(10,20)
21      C          GLOBAL TRACE 10 VIOLATIONS
22      C          ASSERT RANGE (I,J,K,L) (0,100)
23      C          ASSERT RANGE (II,LL) (-10,10)
24      C          ASSERT VALUES (KK,NN) (2,4,6,8,10)
25      C          ASSERT SUBSCRIPT RANGE (A(*,3))
26      C          ASSERT NO SIDE EFFECTS (X,Y,Z)
27      C          MONITOR RANGE FIRST LAST (K,NN)
28      C          MONITOR SUBSCRIPT RANGE (A)
                   .
                   .
                   .
102                K = K + 1                                            511    MIN=                 0  MAX=             101
                                                                               FIRST=               8  LAST=             29
                                                                               ASSERTION VIOLATIONS       1
                                                                               ASSERT RANGE (I,J,K,L) (0,100)
                                                                                  EXEC NUMBER          VALUE
                                                                                         10             101

103                II = A(L,J) + LL                                     511    SUBSCRIPT 1 IN ARRAY A
                                                                                MIN=      1 MAX=    12
                                                                               SUBSCRIPT 2 IN ARRAY A
                                                                                MIN=      3 MAX=     4
                                                                               ASSERTION VIOLATIONS       3
                                                                               ASSERT RANGE (II,LL) (-10,10)
                                                                                  EXEC NUMBER          VALUE
                                                                                         22              20
                                                                               ASSERT SUBSCRIPT RANGE (A(*,3))
                                                                                  EXEC NUMBER          VALUE
                                                                                          5            A(12,3)
                                                                                        105            A(1,4)
                   .
                   .
                   .
234                K = A(J,K) + I*100                                   125    MIN=                 0  MAX=             102
                                                                               FIRST=              31  LAST=             80
                                                                               SUBSCRIPT 1 IN ARRAY A
                                                                                MIN=      3 MAX=     9
                                                                               SUBSCRIPT 2 IN ARRAY A
                                                                                MIN=      3 MAX=     4
                                                                               ASSERTION VIOLATIONS       4
                                                                               ASSERT RANGE (I,J,K,L) (0,100)
                                                                                  EXEC NUMBER          VALUE
                                                                                         52             101
                                                                                         53             102
                                                                               ASSERT SUBSCRIPT RANGE (A(*,3))
                                                                                  EXEC NUMBER          VALUE
                                                                                         52            A(5,4)
                                                                                         53            A(6,4)
```

ANNOTATED PROGRAM LISTING

EXECUTION COUNT — SPECIFIC EXECUTION DATA

```
235           II = II + 2

236           NN = KK*(I-J)

                .
                .
                .
250     C     ASSERT  (A(2:6,2:10) .EQ. 0)

                .
                .
                .
280     C     ASSERT  ORDER  (A(*,3))  ASCENDING

                .
                .
                .
300           CALL ROUTINEX(X,Y)

                .
                .
                .
```

```
125   ASSERTION VIOLATIONS          1
      ASSERT RANGE (II,LL) (-10,10)
         EXEC NUMBER            VALUE
               50                12

 38   MIN=                 2  MAX=              10
      FIRST=               2  LAST=             8
      ASSERTION VIOLATIONS          1
      ASSERT VALUES (KK,NN) (2,4,6,8,10)
         EXEC NUMBER            VALUE
               20                7

253   ASSERTION VIOLATIONS          2
         EXEC NUMBER     ELEMENT        VALUE
               29        A(3,5)          10
               55        A(2,2)          18

229   ASSERTION VIOLATIONS          1
         EXEC NUMBER  SEQUENCE SNAPSHOT VALUE
               18          A(7,3)          6
                           A(8,3)        100 *
                           A(9,3)          8

 53   ASSERTION VIOLATIONS          1
      ASSERT NO SIDE EFFECTS (X,Y,Z)
                        VALUE OF CALL PARM X
         EXEC NUMBER  BEFORE CALL   AFTER CALL
               30          -10          -20
```

ANNOTATED PROGRAM LISTING

```
00001        PROGRAM  TICTAC (INPUT,OUTPUT)
00002  C
00003  C       INTERACTIVE TIC-TAC-TOE PLAYER
00004  C
00005        INTEGER  FIRST, GAMES, MOVE, PLAYER, TURNS, WINNER
00006        INTEGER  HUMAN, MACHINE, NOBODY, QUIT, BOARD(9), NAME(10)
00007        COMMON  /CONST/  HUMAN, MACHINE, NOBODY, QUIT, NAME
00008  C
00009  C     GLOBAL TRACE 10 VIOLATIONS
00010  C     ASSERT RANGE (TURNS) (0,5)
00011  C     MONITOR RANGE FIRST LAST ( PLAYER,MACHINE,TURNS )
00012  C     MONITOR CHARACTER RANGE FIRST LAST ( NAME(*) )
00013  C
00014        DATA  NOBODY /1/, MACHINE /2/, HUMAN /3/, QUIT /4/
00015        DATA  NAME /  2HTL, 2HTC, 2HTR, 2HML, 2HMC, 2HMR, 2HBL, 2HBC, 2HBR
00016       X, 2HNO/
00017        CALL PRINTRO                                                    1
00018  C       PLAY SOME GAMES
00019        FIRST = HUMAN                                                   1
00020        GAMES = 0                                                       1
00021        LOST = 0                                                        1
00022        MYSCORE = 0                                                     1
00023        WHILE WINNER.NE.QUIT DO                                         1
00024  C          PLAY 1 GAME
00025          GAMES = GAMES+1                                               3
00026          WINNER = NOBODY                                               3
00027          TURNS = 0                                                     3
00028          FOR I=1 STEP 1 UNTIL 9 DO                                     3
00029            BOARD(I) = NOBODY                                          27
00030          ENDDO                                                        27
00031          IF FIRST.EQ.MACHINE THEN                                      3   TRUE          1   FALSE         2
00032            BOARD(5) = MACHINE                                          1
00033            TURNS = 1                                                   1
00034            PRINT 90005, NAME(5)                                        1
00035  90005     FORMAT(*I HAVE SELECTED SQUARE *,A2)
00036          ENDIF                                                         3
00037          WHILE WINNER.EQ.NOBODY .AND. TURNS.LT.9 DO                    3
00038            PLAYER = HUMAN                                             10   MIN=          3   MAX=          3
                                                                                 FIRST=        3   LAST=         3
00039            CALL HUMNMOV (BOARD, MOVE)                                 10
00040            IF MOVE.NE.10 THEN                                         10   TRUE          9   FALSE         1
00041              BOARD(MOVE) = HUMAN                                       9
00042              TURNS = TURNS+1                                           9   MIN=          1   MAX=          9
                                                                                 FIRST=        1   LAST=         3
                                                                                 ASSERTION VIOLATIONS          4
                                                                                 ASSERT RANGE (TURNS) (0,5)
                                                                                    EXEC NUMBER          VALUE
                                                                                            3              6
                                                                                            4              7
                                                                                            5              8
                                                                                            6              9
```

```
                                                              EXECUTION
       ANNOTATED PROGRAM LISTING                                COUNT       SPECIFIC EXECUTION DATA

         CALL CHEKWIN (BOARD, MOVE, TURNS, PLAYER, WINNER)        9
C          TRACE 10 VIOLATIONS
C          ASSERT (MOVE .LT. 9)  HALT ON 10                       9    ASSERTION VIOLATIONS          2
                                                                          EXEC NUMBER            VALUE
                                                                               6                   9
                                                                               7                   9
C         TRACE OFF
         IF WINNER.EQ.NOBODY .AND. TURNS.LT.9 THEN                9    TRUE              8  FALSE          1
           PLAYER = MACHINE                                       8    MIN=              2  MAX=           2
                                                                       FIRST=            2  LAST=          2
           CALL MACHMOV (BOARD, TURNS, MOVE)                      8
           BOARD(MOVE) = MACHINE                                  8
           TURNS = TURNS+1                                        8    MIN=              2  MAX=           8
                                                                       FIRST=            2  LAST=          4
                                                                       ASSERTION VIOLATIONS          3
                                                                       ASSERT RANGE (TURNS) (0,5)
                                                                          EXEC NUMBER            VALUE
                                                                               3                   6
                                                                               4                   7
                                                                               5                   8
           CALL CHEKWIN (BOARD, MOVE, TURNS, PLAYER, WINNER)      8
         ENDIF                                                    9
       ELSE                                                       1
         WINNER = QUIT                                            1
       ENDIF                                                     10
     ENDDO                                                       10
     CALL PRESULT (BOARD, WINNER, GAMES, MYSCORE, LOST)           3
     IF FIRST.EQ.MACHINE THEN                                     3    TRUE              1  FALSE          2
       FIRST = HUMAN                                              1
     ELSE                                                         2
       FIRST = MACHINE                                            2
     ENDIF                                                        3
   ENDDO                                                          3
   END                                                            1
```

```
**************************************
*                                    *
*    AUTOMATED VERIFICATION SYSTEM   *
*         OPERATIONAL PROFILE        *
*          SUBROUTINE TICTAC         *
*                                    *
**************************************
```

TYPE OF STATEMENT	NUMBER	NUMBER EXECUTED	PERCENT EXECUTED
TRANSFER			
IF	4	4	100.0
TRUE CONDITIONS		4	100.0
FALSE CONDITIONS		4	100.0
CALL	6	6	100.0
GO TO	0	0	N/A
CASE	0	0	N/A
ALSO CONDITIONS	0	0	N/A
OTHERWISE CONDITIONS	0	0	N/A
DO	3	3	100.0
WHILE CONDITION	2	2	100.0
ASSIGNMENT	19	19	100.0
I/O			
READ	0	0	N/A
WRITE	1	1	100.0
OTHER EXECUTABLE SOURCE	9	9	100.0

INSTRUMENTATION SUMMARY

LOCAL ASSERTIONS

STMT NO.	ASSERTION	EXEC COUNT	VIOLATIONS
00045	ASSERT (MOVE .LT. 9) HALT ON 10	9	2

GLOBAL ASSERTIONS

ASSERT RANGE (TURNS) (0,5)

VARIABLE	STATEMENTS THAT CHANGE THE VALUE OF VARIABLE	STATEMENTS THAT CAUSED AN ASSERTION VIOLATION
TURNS	00027 00033 00042 00051	00042 00051

MONITOR RANGE FIRST LAST (PLAYER,MACHINE,TURNS)

VARIABLE	STATEMENTS THAT CHANGE THE VALUE OF VARIABLE	VALUES ATTAINED MIN	MAX
PLAYER	00038 00048	2	3
MACHINE	00014	2	2
TURNS	00027 00033 00042 00051	0	9

MONITOR CHARACTER RANGE FIRST LAST (NAME(*))

VARIABLE	STATEMENTS THAT CHANGE THE VALUE OF VARIABLE	VALUES ATTAINED MIN	MAX
NAME(*)	00015	'BC '	'TR