NASA Technical Memorandum 80205

Evaluation of Verification and Testing Tools for FORTRAN Programs

Kathryn A. Smith

JULY 1980

NASA

NASA Technical Memorandum 80205

Evaluation of Verification and Testing Tools for FORTRAN Programs

Kathryn A. Smith
Langley Research Center
Hampton, Virginia

NASA
National Aeronautics
and Space Administration

Scientific and Technical
Information Branch

1980

SUMMARY

The traditional procedure to ensure the provision of quality, reliable software involves extensive testing. In recent years, several automated software verification and testing systems (V&T tools) have been developed to discover errors in computer programs, saving computer and analyst time and improving the quality of programs. Two such tools which may be used in the analysis of FORTRAN programs have been installed and evaluated at the National Aeronautics and Space Administration Langley Research Center. These are DAVE, a static analysis tool, and PET (Program Evaluator and Tester), a dynamic analysis tool. These V&T systems were evaluated using a series of test cases. The results of this evaluation indicate that, despite some limitations, both of these tools are significantly beneficial in the development and testing of FORTRAN programs. Based on this analysis, these tools are recommended for general use.

INTRODUCTION

Computers are becoming larger and faster. As a result, computer programs are becoming more complex, and software tools for the development and testing of programs are becoming increasingly important. A number of tools are emerging for use in the software development and testing process. Included in this number are several software tools which may be used in the verification and testing of FORTRAN programs. Two such tools which have been applied to computer programs at the National Aeronautics and Space Administration (NASA) Langley Research Center (LaRC) are a static analyzer, DAVE (refs. 1 and 2), and a dynamic analyzer, PET (refs. 3 and 4). This paper presents our evaluation of these tools and our experiences with them.

Use of trade names or names of manufacturers in this report does not constitute an official endorsement of such products or manufacturers, either expressed or implied, by the National Aeronautics and Space Administration.

SOFTWARE DEVELOPMENT PROCESS

The software development process consists of several steps. The first is the definition of program requirements. Figure 1 illustrates the basic development phases which follow the establishment of requirements. Although these steps may appear disjointed, they overlap in many instances. For example, documentation should be developed concurrently with the program. Once the program requirements are established, the design of the software can begin. After completion of the design phase, the program is coded and several steps of successive program testing begin. The first step is the use of a compiler to perform a syntax analysis which checks that the program is compatible with the grammatical structure of the programming language. Any syntax errors should be corrected before going to the next phase of testing.

Frequently, a programmer will consider the testing completed at this point. However, careful, disciplined testing includes both static analysis and execution time testing. Static analysis takes a subject program as input and tests the logic flow and data flow through the program. After the logic- and data-flow errors are found and corrected, the execution time testing begins. Part of this testing is a dynamic analysis which identifies the portions of the subject program which have been executed. Static analysis and dynamic analysis are difficult and time consuming without the aid of software tools. At LaRC, the DAVE system is recommended for static analysis and PET is recommended for execution time analysis of FORTRAN programs. When the execution testing is completed, the documentation of the program can be completed, after which the program is ready for productive use. Many of the steps used in software development are also used in the acceptance of programs developed under contract and in the maintenance and revision of existing programs. These steps are indicated by the solid boxes in figure 1.

DAVE

Description

DAVE is a static software verification tool which was developed by the Department of Computer Science, University of Colorado, under a grant from the National Science Foundation. LaRC obtained DAVE from the University of Colorado in 1976. It is a system tool for gathering information about global data flow in syntactically correct ANSI FORTRAN[1] programs and for identifying the anomalous use of data in these programs. DAVE is called a static analysis tool, because the FORTRAN program is neither compiled nor executed; instead, information is gathered about the subject program by analyzing the source code. DAVE uses global data flow analysis to reveal suspicious or erroneous use of data; that is, it examines all possible data paths. Thus, DAVE offers a multimodule analysis capability not available in the Control Data (CDC) FORTRAN Extended (FTN) compiler at LaRC. In general, DAVE does not require any modification of the subject program, nor does it require any intervention by the user during execution. The steps in DAVE system execution are shown in figure 2. The appendix describes the procedure to use DAVE at LaRC.

DAVE issues three types of diagnostics - "errors," "warnings," and "messages." "Error" diagnostics indicate likely execution errors. These errors include such things as variables uninitialized on all paths, a constant subroutine argument assigned a value in the subroutine, and a subprogram FUNCTION name never assigned a value. "Warning" diagnostics are possible execution errors. Examples of warnings include: variables uninitialized on some paths and variables unused on some paths. The "message" diagnostics are used to give information about COMMON blocks and COMMON variables. For example, a message would identify those COMMON variables in a module which are initialized in BLOCK DATA. The user has the option to select the diagnostics he desires.

---

[1] American National Standard FORTRAN, ANSI X3.9-1966.

Resources

The DAVE system consists of four programs (phases) which are executed sequentially. Information is passed between phases by intermediate files. (See fig. 2.) The largest phase of DAVE (PHASE2) requires 207 000_8 words of memory at LaRC. The execution time depends on the length (i.e., number of lines of code) and complexity of the subject program (e.g., number of subroutines and variables). Timing information for several test cases is provided in table I. For these test cases, the average execution time/line of input code was 0.09 sec on the Control Data CYBER 175 (or 0.45 sec/line on a CYBER 173) at LaRC. Execution time/line increases as the size of the subject program increases. There is no direct relationship between DAVE execu-tion time and FORTRAN compilation time. In separate test cases, DAVE time ran from 25 times the compilation time (800 lines of code) to 130 times (7200 lines). A comparison of FORTRAN compilation time and DAVE execution time is shown in figure 3.

Experiences

DAVE was first installed at LaRC in 1976. An improved version, DAVE 8.0, was installed in 1977. Although this second version of DAVE performed well, its execution time was greater than desired. With the help of the University of Colorado, changes were made to DAVE, in the fall of 1977, that decreased execution time by as much as 47 percent. The new version works well on small and medium programs (i.e., less than 2000 lines of code). DAVE was originally written to accept only ANSI FORTRAN, a limitation which was not entirely satisfactory. While improving DAVE execution time, changes were made to handle some non-ANSI FORTRAN. The LaRC version of DAVE will now process the following non-ANSI input/output (I/O) features: NAMELIST, ENCODE/DECODE, and BUFFER IN/BUFFER OUT. In addition, most other non-ANSI statements now are treated as comments. An updated version of DAVE, which analyzes many widely used non-ANSI features of FTN, has become available. It is anticipated that this version will be installed at LaRC.

Approximately 15 people have used DAVE since its installation at LaRC. It has been used on several applications programs, including a cross-assembler, an ozone model program, and IFTRAN, a structured preprocessor. DAVE typically found three to five errors in test programs that were syntactically correct, including some in production use. Generally, these errors were characterized by incorrect use of subroutine parameters. This particular type of error is difficult to detect by ordinary means. DAVE has found errors that are several levels deep. For example, a literal parameter was passed through three subroutine calls and then changed at the lowest level. A simplified example follows:

```
PROGRAM TEST
  ...
CALL SUB1(1.)
  ...
END
SUBROUTINE SUB1(A)
B=10.0
CALL SUB2(A,B,C)
```

```
      RETURN
      END
      SUBROUTINE SUB2 (X,Y,Z)
      Z=Y**2
      X=5.0*Z-Y
      RETURN
      END
```

In this example, the literal parameter (1.) is passed from the main program to subroutine SUB1. SUB1 in turn passes it as the first parameter (A) in its call to subroutine SUB2. In SUB2, the first parameter (X) is not used as an input, but rather X is set to the value of a computed expression. This computed value will replace the present value of X, which is passed back to SUB1 and the main program. Thus, the value of the literal (1.) in the main program is destroyed. This could cause some unknown and untraceable errors in the main program. An error of this type generally is not detected by the LaRC FTN compiler.

Other errors which have been detected include incorrect number of subroutine parameters in calling program, uninitialized variable passed as an input parameter to a subroutine, and disagreement on subroutine parameter type.

Evaluation

Most of our experiences with DAVE have been favorable. DAVE can be a cost effective verification tool whether used as an aid in the development or acceptance of new programs or in the maintenance of existing programs. Our experiences have shown DAVE to be an effective tool when used with small to medium sized programs. Execution time required for programs of this size is not unreasonable. As can be seen in figure 3 however, execution time increases significantly with program size for programs over 2000 lines long. Thus, DAVE is not recommended for use with long programs. One other drawback of DAVE (LaRC) is that some non-ANSI constructs still cause DAVE to halt. Among these are octal constants, variable names with more than six characters, and OVERLAY statements.

From the user's point of view, DAVE is easy to use. It requires only a few control cards and does not require user intervention or interaction during execution. On the surface, DAVE appears to give the same sort of diagnostics as a compiler. Actually, it has a multimodule capability not available on the FTN compiler and, thus, can detect errors not easily found by other means.

The overall evaluation is that DAVE is a useful analysis tool when used in the development and maintenance of small to medium FORTRAN programs. DAVE is easy to use and provides analysis diagnostics not available from other sources, thus saving analyst and computer time.

PET

Description

PET (Program Evaluator and Tester) is a package of programs designed as an automated aid to assist in the debugging, testing, and documentation of FORTRAN programs. PET is a proprietary program which was acquired by LaRC from the McDonnell Douglas Astronautics Company in the spring of 1977 as part of the MUST (Multipurpose User-Oriented Software Technology) program.

PET is a two-step process consisting of a preprocessor and a postprocessor. The preprocessor instruments the user's FORTRAN program based on the options specified. That is, PET translates the user's options into code which is inserted into the program. When the instrumented program is executed with test data, the PET inserted code generates an intermediate file of run-time statistics. These statistics include execution counts and information on assignment variables and DO-loop parameters. After execution, the postprocessor sorts through the run-time statistics and generates the requested reports. Figure 4 is an illustration of the way PET interacts with the user's program. The procedures to use PET at LaRC are described in the appendix.

PET generates several types of reports which can be used to determine the thoroughness of test cases. These reports are generated for each instrumented module and in summary for the whole program. The module reports include a syntactic profile, an operational profile, an assertion summary, and a detailed execution report. The syntactic profile indicates such things as the number of executable statements, the number of nonexecutable statements, and the number of comments. The operational profile gives such information as how many statements were executed and what percentage of the total executables were used. Syntactic and operational profiles are shown in figure 5. The detailed execution report gives execution counts for all lines of code and optionally reports the first/last and minimum/maximum values of assignments and first/last values of DO-loop parameters. An example of the execution report with these options is shown in figure 6. Summary reports include syntactic and operational profiles which are composites of the modular profiles, a module operational profile, a module execution summary, and a module timing summary with an optional relative timing histogram.

Figure 7 is an illustration of the way probes are inserted into the subject programs to keep track of execution counts. The PET options are controlled by PET directives which are inserted into the source code. These directives are implemented to look like FORTRAN comments. For example, selection of the option which reports the minimum/maximum values of assignment variables is accomplished by inserting the following option card into the user's program:

```
COPT=1
```

This can be selected for all or part of a program. To change options, the user would simply insert a new COPT card. A complete description of the PET directives can be found in the PET user's manual (ref. 4).

Among the many features of PET are a modular capability and an assertion capability. The modular feature allows the user to optionally instrument all or any part of his program. This option allows existing instrumented code to be combined with new or modified modules. In the modular mode, only the changed modules are input to the preprocessor, resulting in reduced preprocess and compilation time. This particular feature is useful when testing only the modified parts of large programs.

The assertion capability allows user inserted statements to assert that computed values meet given criteria. Local (at a given line of code) and global (over a module) assertions are available. The assertion checking option is useful in debugging and in verifying that a program is operating correctly. PET will report any violations of these assertions and will give a summary of violations for each module. For example, the user may assert that a variable T is greater than -290.0, whenever it is computed in a module by inserting the following comment card at the beginning of the module:

```
CASSERT VALUE(T) (GT.-290.0)
```

The PET preprocessor will insert a test into the user's program wherever variable T appears on the left side of an assignment statement. Whenever this assertion is violated during execution, a counter will be incremented. The postprocessor will sort these violations and will report the number and location of violations.

Resources

The PET preprocessor uses $145\ 000_8$ words of memory and the postprocessor $62\ 000_8$ words of memory for execution on a CYBER 173 at LaRC. Execution time for test cases ranged from 5 to 40 seconds for the preprocessor and from 3 to 10 seconds for the postprocessor on a CYBER 173. Run-time field length for the subject programs was increased by about 30 percent. This may vary slightly depending on the options selected. The execution time of most test programs was increased by about 20 to 50 percent. Table II is a summary of the resources used by our test cases.

Experiences

PET was installed at LaRC in 1977. The initial version of PET did not recognize certain FORTRAN constructs, such as certain sequences of multiple statements. These problems have since been corrected. The few limitations to PET are offset by its effectiveness as a software tool. These limitations are as follows:

(1) Increased execution time of the subject program

(2) Logical IF statements are rewritten as logical IF/NOT statements in instrumented code

Since its installation at LaRC, PET has been used by about 20 people. PET has provided some outstanding results on a variety of programs. For example, the use of PET output as an aid to pinpoint problem areas enabled a programmer to streamline one program extensively, reducing the cost/run by over 80 percent. In another example, PET was used as an aid to pinpoint those parts of a program executed most frequently. By making appropriate program changes, the programmer was able to cut execution time by a factor of 2.5 from 90 seconds to 36 seconds. The execution counts of PET were used to aid in debugging a program. These counts quickly revealed that an entire section of code was never executed, although it should have been. The assertion capability has been used as an aid in the conversion of an old program to FORTRAN Extended. In this instance, PET was used to pinpoint the range of a variable exponent causing an underflow condition and some extraneous error messages.

PET also can be used to check the effectiveness and completeness of test cases in determining if all possible branches and paths have been tested. This feature can be used during the development of new programs, in the acceptance of programs developed under contract, and during the testing of modifications made to existing programs to provide test coverage results.

Evaluation

Experiences with PET have shown it to be an effective analysis tool when used in the verification and testing of FORTRAN programs. PET is useful in both the development and maintenance of programs, providing an analysis capability not otherwise available. This includes such things as: the assertion capability, the relative timing of modules, and the run-time statistics. The flexibility of the modular mode allows instrumentation of new or changed modules as they are added to a program. The assertion capability is useful in detecting computation of erroneous data based upon user specified criteria. PET also provides testing coverage assessment with its run-time statistics, which allows a user to determine the effectiveness of test cases.

The PET system can be easy to use, depending on the options selected. The PET directives, which look like FORTRAN comments, are inserted into the user's code, requiring a minimal amount of program modification. Since the execution of the PET preprocessor and postprocessor is controlled by NOS procedure files, only a few extra control cards are needed by PET users. PET does have a few drawbacks. These include the increased execution time and storage requirements of the subject program, and the special handling required for OVERLAY programs. The modular mode, however, may be used to ease these problems.

CONCLUDING REMARKS

In this paper we have discussed the automated verification and testing tools, DAVE and PET, used in the analysis of FORTRAN programs at Langley Research Center (LaRC). These tools form an effective, complementary set of tools for testing FORTRAN programs. The overhead cost of using these tools may seem high, but this cost is offset by the effectiveness of DAVE and PET. In fact, the overhead incurred when using these tools is not unreasonable

since each of these tools need only be used a few times. Some of the types of errors found by these tools are not readily detected by other means. Resulting improvements made on test programs at LaRC have shown that these tools are cost effective aids in improving software quality.

Langley Research Center
National Aeronautics and Space Administration
Hampton, VA 23665
June 26, 1980

APPENDIX

PROCEDURES FOR USE OF DAVE AND PET AT LANGLEY RESEARCH CENTER

DAVE

The DAVE system is simple to use. It requires only two inputs: (1) the source programs to be analyzed and (2) a file specifying the DAVE options desired. Available options include simulation of missing subprograms (SI) and suppression of specified diagnostics (SU). Through the SU parameter, the user can optionally suppress selected diagnostics or whole category of diagnostics. The maximum memory required by the DAVE system is 207 000_8 words. The DAVE system is accessed on the Network Operating System (NOS) at Langley Research Center (LaRC) by calling procedure file DAVE, using the following control cards:

```
GET,DAVE/UN=82487ON.
CALL(DAVE(INPUT=infl,OPTIONS=opfl)
```

where

infl = name of file containing the subject program

opfl = options file

The procedure file controls the execution of the four phases of DAVE and the transfer of files between these phases. A sample deck setup using DAVE would be:

```
GET,MYFILE.
GET,DAVE/UN=82487ON.
CALL(DAVE(OPTIONS=INPUT,INPUT=MYFILE)
END OF RECORD
SI=ON
END OF FILE
```

In the example, the user's FORTRAN program is on file MYFILE. The user has selected the option SI=ON, which means that missing subroutines will be simulated by DAVE. Also, no diagnostics will be suppressed. Additional information is available in The DAVE System User's Manual (ref. 2).

PET

Usage of the PET system is controlled by two procedure files, PREPET and PSTPET. PREPET activates the preprocessor and, based on the options selected, the preprocessor inserts code into the programs which is used to generate an intermediate file of run-time statistics. After instrumenting the code, PREPET also controls compilation of the FORTRAN source code and, if specified, handles

the modular features. Upon completion of PREPET, control is returned to the user for execution. After execution, the user calls PSTPET to activate the postprocessor. PSTPET automatically sorts the run-time statistics and generates the PET reports.

PET options are selected by inserting PET directives into the user's program. These directives look like FORTRAN comments with a C in column 1. There are three types of PET directives: (1) the COPT card which controls the type of run-time statistics, (2) the CONENDCASE card which controls when run-time statistics are to be collected, and (3) the CMOD card used with the modular mode.

The most frequently used directive is COPT which can be used to select or change the run-time statistics. The options which can be selected are:

Option	Action
1	min/max on assignment statements
2	min/max on DO-loop variable parameters
16	first/last on assignment statements
32	timing on modules

These options may be selected singly or in combination (by summing the options selected). For example, COPT=17 would give min/max and first/last values on assignment parameters. A complete description of the PET directives and the control cards needed to run PET is given in the PET user's manual (ref. 4).

An example of a nonmodular (sequential) case would be:

```
      .
      .
user control cards
      .
      .
GET,PREPET,PSTPET/UN=82487ON.
CALL(PREPET(COMPILE=INPUT)
      .
      .
user load and execute sequence
```

```
  .
  .
CALL(PSTPET)
EOR
COPT=51
  .
  .
user's source program
  .
  .
EOR
  .
program data
  .
EOF
```

This example represents a sequential run in which the entire program is to be instrumented for PET option COPT=51. This is the combination of options 1, 2, 16, and 32.

REFERENCES

1. Osterweil, Leon J.; and Fosdick, Lloyd D.: DAVE - A Validation Error Detection and Documentation System for Fortran Programs. Software - Pract. & Exper., vol. 6, no. 4, Oct. - Dec. 1976, pp. 473-486.

2. Fosdick, Lloyd D.; and Miesse, Carol: The DAVE System User's Manual. Cu-CS-106-77, Univ. of Colorado, Mar. 1977.

3. Stucki, Leon G.: A Prototype Automatic Program Testing Tool. AFIPS Conference Proceedings, Volume 41, Part II - 1972, Fall Joint Computer Conference, AFIPS Press, c.1972, pp. 829-836.

4. Churchwell, J. B.: Program Evaluator and Tester (PET) User's Manual. NASA CR-159295, 1977.

TABLE I.- DAVE TIMING STUDY

Computer	DAVE test case	No. of blocks[a]	Execution time, sec	Time/line of code, sec
CYBER 175	1	228	12.438	0.0545
CYBER 175	2	500	46.250	.0925
CYBER 175	3	715	61.021	.0853
CYBER 175	4	704	76.016	.107
CYBER 173	5	704	290.075	.412
6400[b]	6	228	57.946	.254
6400	7	965	592.301	.613
6400	8	1579	2068.779	1.31

[a]Note blocks are lines of executable code.
[b]Control Data series 6400 computer system at LaRC.

TABLE II.- PET RESOURCES STUDY

	With PET					Without PET		
PET test case	PREPET execution time, sec	FORTRAN compile time, sec	Test case execution time, sec	PSTPET execution time, sec	Memory, octal words	FORTRAN compile time, sec	Test case execution time, sec	Memory, octal words
1	25.273	23.83	4.568	3.347	66 562	12.352	3.544	52 370
2	15.639	17.366	8.335	1.94	40 455	12.675	5.170	27 753
3	38.942	29.94	1976.	4.703	114 313	8.688	899.1	105 000
4	5.553	11.13	.643	9.226	26 610	1.395	.01	17 027
5	35.772	23.613	256.611	4.829	121 131	21.095	230.551	112 454

Note: Times are in central processing unit (CPU) seconds on a CYBER 173.

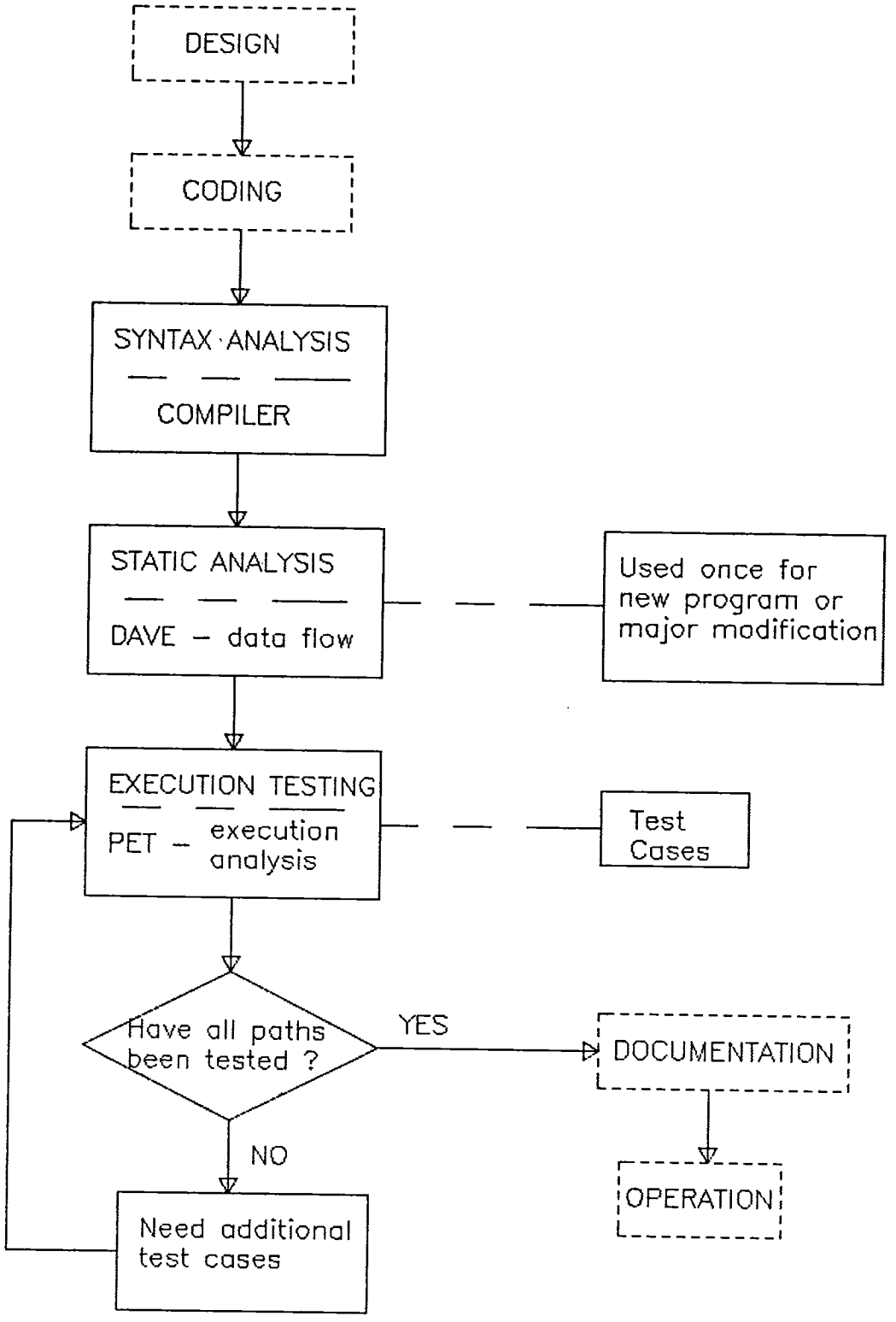

Figure 1.- Software tools used in verification and testing of programs.

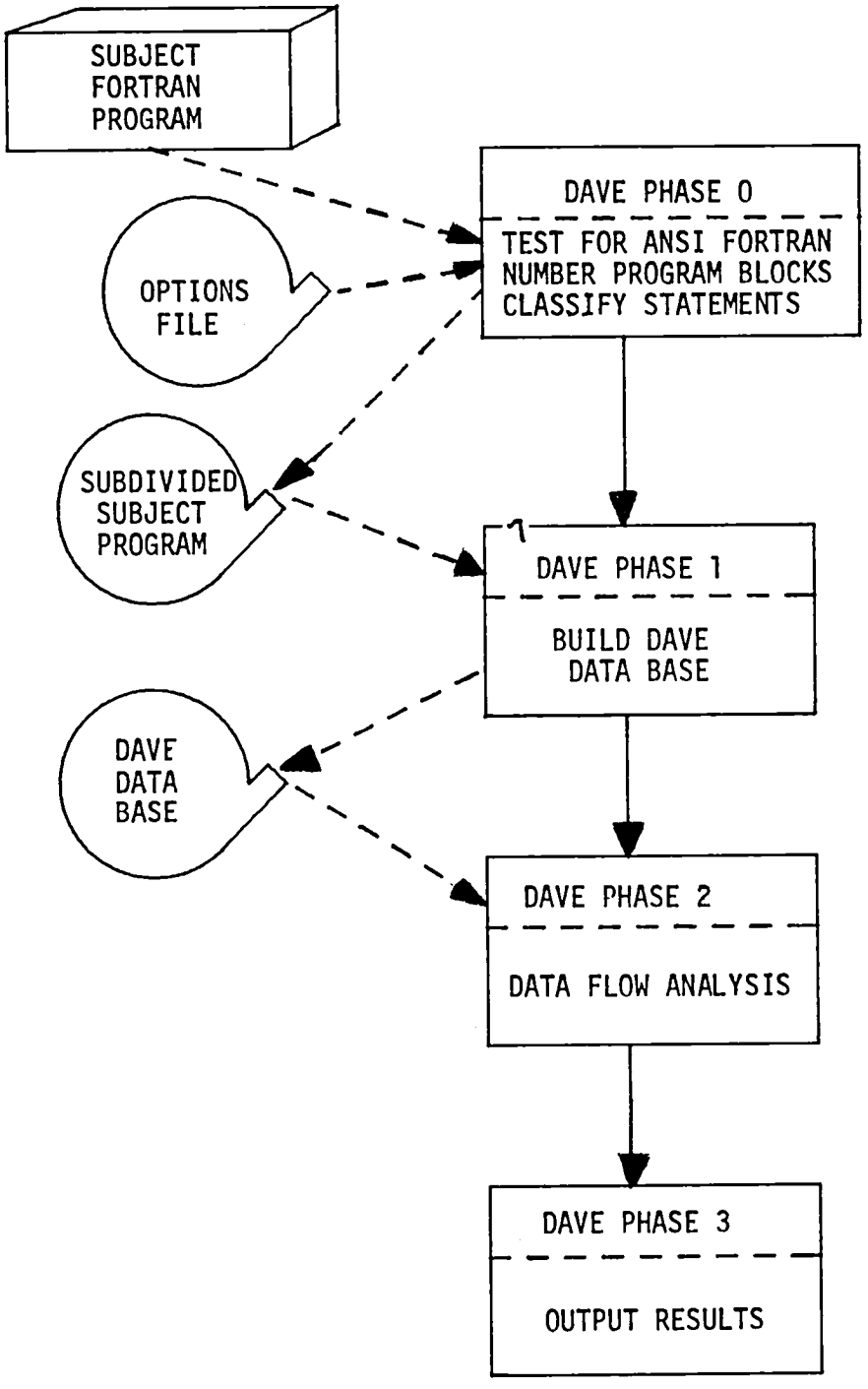

Figure 2.- Steps in DAVE execution.

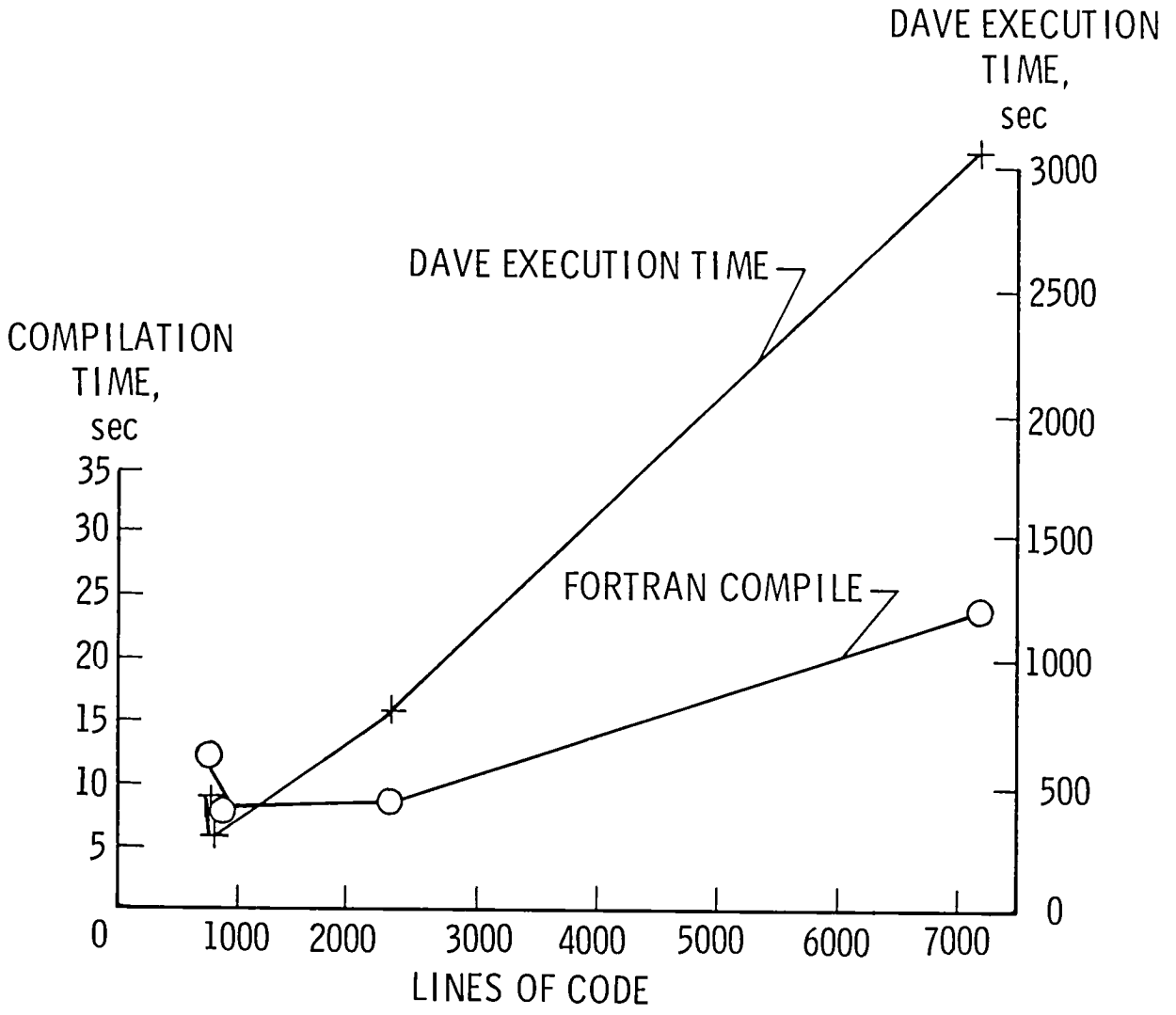

Figure 3.- Comparison of FORTRAN compilation time and DAVE execution time.

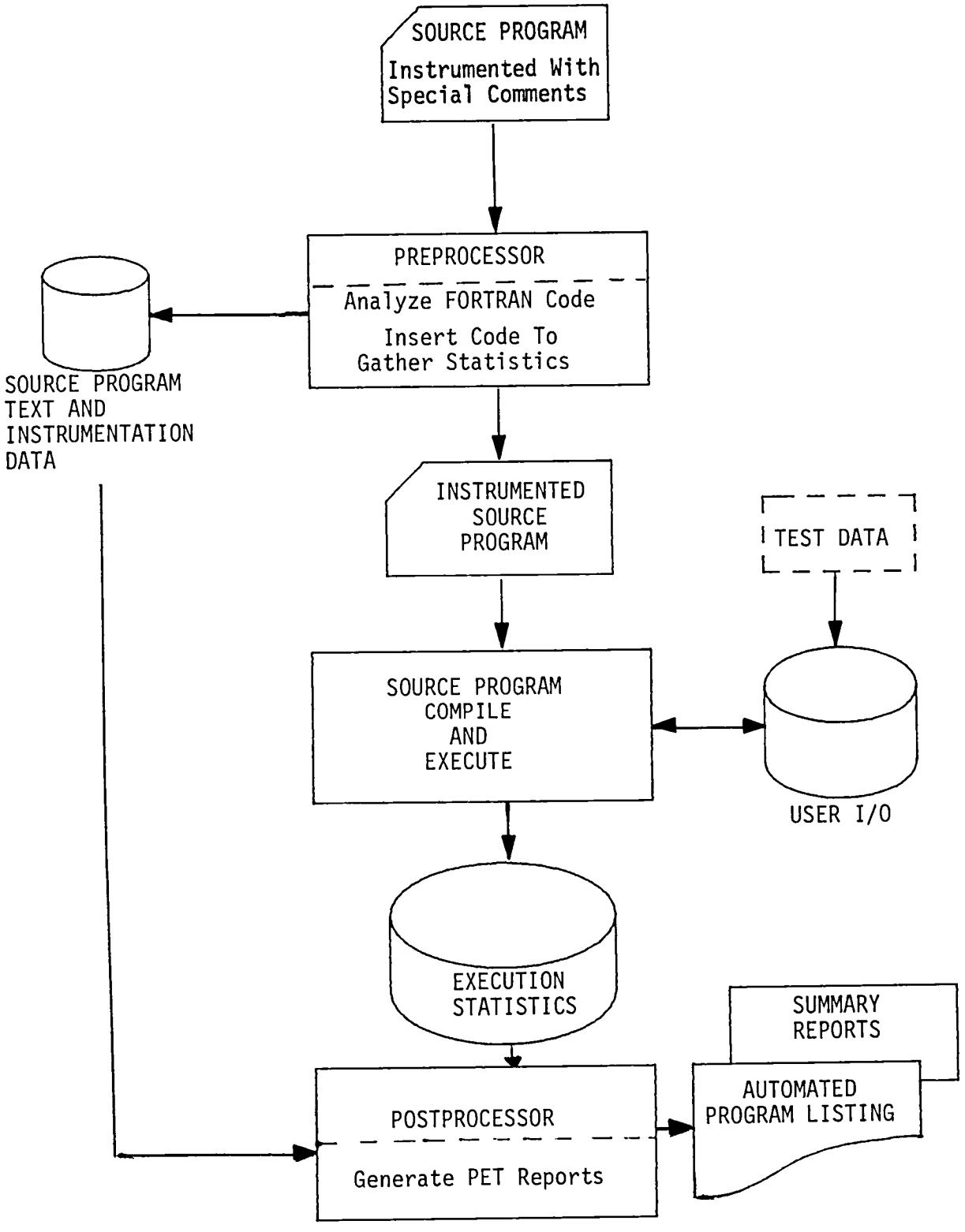

Figure 4.- PET interaction with user's program.

```
*****************************************
*                                       *
* PROGRAM EVALUATOR AND TESTER REPORT   *
*                                       *
*          FOR MODULE PCH               *
*                                       *
*****************************************
```

SYNTACTIC PROFILE

TOTAL NUMBER OF SOURCE STATEMENTS 91

TYPE OF STATEMENT	NUMBER	PERCENT OF TOTAL
EXECUTABLE SOURCE	52	57.1
NONEXECUTABLE SOURCE	31	34.1
COMMENT	8	8.8
ASSERTION DIRECTIVE	0	0.0
NONSTANDARD	15	16.5
BRANCH	5	N/A
CALL	2	N/A
UNFORMATTED I/O	2	N/A
FORMATTED I/O	15	N/A

OPERATIONAL PROFILE

TOTAL EXECUTION COUNT 13755

TYPE OF STATEMENT	NO EXECUTED	PERCENT EXECUTED
EXECUTABLE SOURCE	51	98.1
BRANCH	5	100.0
CALL	2	100.0
UNFORMATTED I/O	2	100.0
FORMATTED I/O	14	93.3

Figure 5.- PET syntactic and operational profiles.

```
STATEMENT
 NUMBER    LISTING FOR MODULE PCH          (LEADING N INDICATES CONVERSION WARNINGS)      COUNT       SPECIFIC EXECUTION DATA

   59        AMAT(2,1)=SST                                                              617 MIN  = .18016E-03 MAX  = .18040E-03
                                                                                           FIRST= .18040E-03 LAST = .18018E-03
   60        AMAT(1,2)=HSV                                                              617 MIN  = .55310E-04 MAX  = .42883E-03
                                                                                           FIRST= .42883E-03 LAST = .55310E-04
   61        AMAT(2,2)=SSV                                                              617 MIN  = .18261E-02 MAX  = .50846E-02
                                                                                           FIRST= .50846E-02 LAST = .18281E-02
   62        BMAT(1)=-HTV                                                               617 MIN  =-.18037E-03 MAX  = .61416E+00
                                                                                           FIRST= .61416E+00 LAST = .11576E-11
   63        BMAT(2)=-STV                                                               617 MIN  =-.16975E-02 MAX  = .36019E-02
                                                                                           FIRST= .36019E-02 LAST = .12022E-05
   64        IOP=1                                                                      617
   65        CALL MATINV(2,2,AMAT,1,BMAT,IOP,DETERM,ISCALE,IPIVOT,INDEX)                617
   66        DEL1=ABS(BMAT(1)/TC(I))                                                    617 MIN  = .11018E-09 MAX  = .18640E-02
                                                                                           FIRST= .18640E-02 LAST = .11018E-09
   67        DEL2=ABS(BMAT(2)/VC(I))                                                    617 MIN  = .81910E-05 MAX  = .25962E-01
                                                                                           FIRST= .25962E-01 LAST = .98571E-05
  68N        PRINT 9,DEL1,DEL2,BMAT                                                     617
   69        IF(DEL1.LT.E1.AND.DEL2.LT.E2) GO TO 33                                     617          18 TRUE          599 FALSE
   70        TC(I)=TC(I)+BMAT(1)                                                        599 MIN  = .12924E+04 MAX  = .12924E+04
                                                                                           FIRST= .12924E+04 LAST = .12924E+04
   71        VC(I)=VC(I)+BMAT(2)                                                        599 MIN  = .24623E+02 MAX  = .66718E+02
                                                                                           FIRST= .24623E+02 LAST = .66718E+02
   72     22 CONTINUE                                                                   599
  73N        PRINT 101                                                             *      0
   74    101 FORMAT(24HOMAX.ITERATIONS EXCEEDED)
         33 PRINT 100,NO(I),PRAT(I),VEI(I),TEI(I),XMACH(I),
  76N      1VC(I),TC(I)                                                                 18
   77    100 FORMAT(I5,6E16.8)
  78 C       P(V,T)
   79        P2T(I)=FONC(VC(I),TC(I))                                                   18 MIN  = .23621E+01 MAX  = .57077E+01
                                                                                           FIRST= .57077E+01 LAST = .23621E+01
   80        P2RT(I)=P2T(I)/PEE                                                         18 MIN  = .92199E-03 MAX  = .22278E-02
                                                                                           FIRST= .22278E-02 LAST = .92199E-03
```

Figure 6.- Sample PET output.

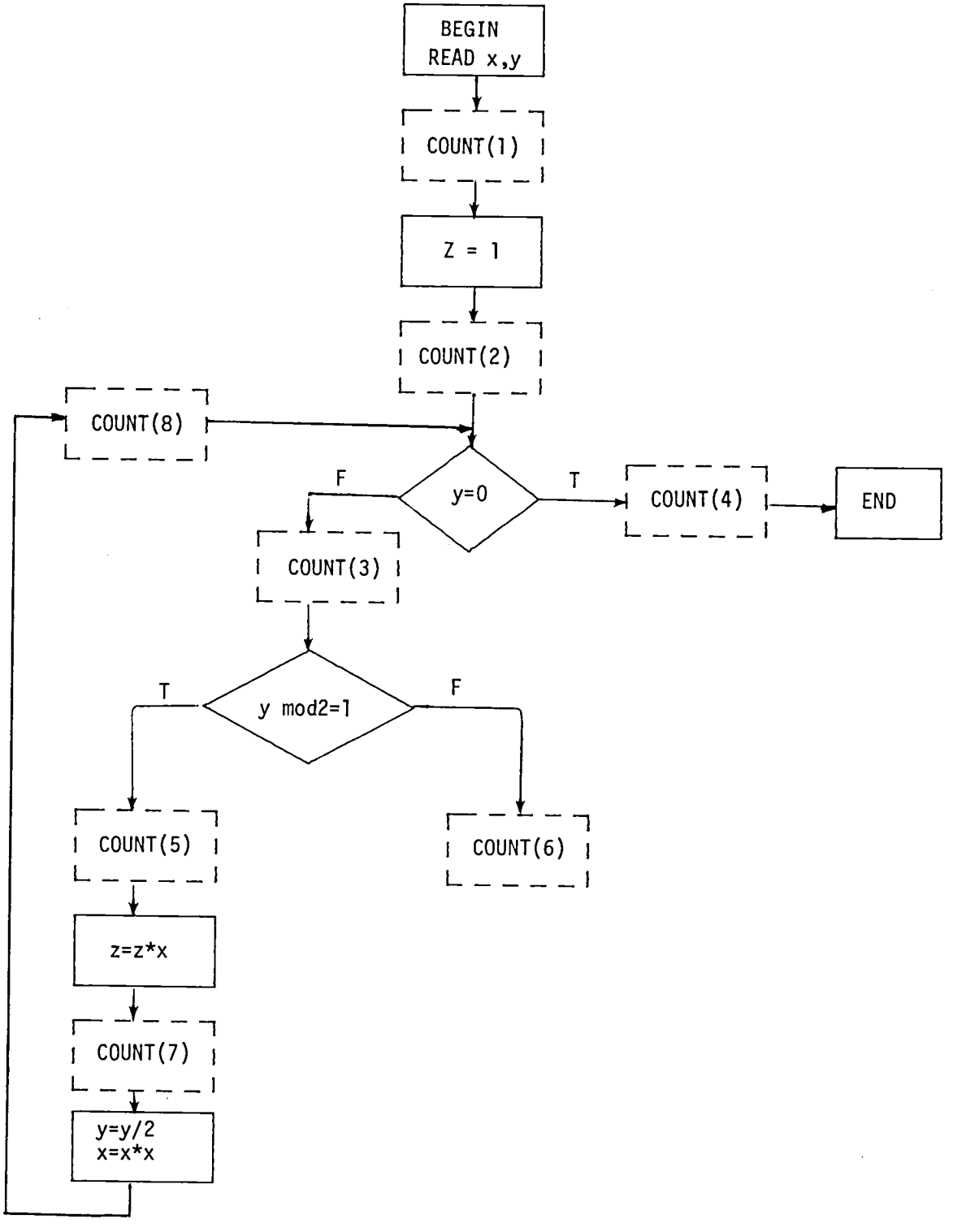

Figure 7.- Insertion of PET counters.

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM-80205		

4. Title and Subtitle	5. Report Date
EVALUATION OF VERIFICATION AND TESTING TOOLS FOR FORTRAN PROGRAMS	July 1980
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
Kathryn A. Smith	L-13665
	10. Work Unit No. 506-61-13-02
9. Performing Organization Name and Address NASA Langley Research Center Hampton, VA 23665	11. Contract or Grant No.
	13. Type of Report and Period Covered Technical Memorandum
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

In recent years, several automated software verification and testing systems have been developed for use in the analysis of computer programs. Such tools not only discover errors in computer programs, but they save computer and analyst time and help to improve the quality of programs. This paper discusses two such tools, the static analyzer DAVE and the dynamic analyzer PET, which are used in the analysis of FORTRAN programs on the Control Data (CDC) computers at NASA Langley Research Center. The report describes the evaluation of these systems based on a set of test cases. These tools were found to be effective and complementary, and they both are recommended for use in testing FORTRAN programs.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Software tools PET FORTRAN programs Verification Testing DAVE	Unclassified - Unlimited Subject Category 61

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	21	A02

For sale by the National Technical Information Service, Springfield, Virginia 22161

NASA-Langley, 1980

National Aeronautics and
Space Administration

Washington, D.C.
20546

Official Business
Penalty for Private Use, $300

SPECIAL FOURTH CLASS MAIL
BOOK

Postage and Fees Paid
National Aeronautics and
Space Administration
NASA-451

NASA

POSTMASTER: If Undeliverable (Section 158 Postal Manual) Do Not Return