NASA -CR - 166,346

NASA CONTRACTOR REPORT 166346

Automated Verification of Flight Software - User's Manual

S. H. Saib

CONTRACT NAS2- 10550
April 1982

NASA

```
ENTER:
      SELECT      RN/NASA-CR-166361          SE104: TERM NOT IN DICTIONARY
      SELECT      RN/NASA-CR-166353          SE104: TERM NOT IN DICTIONARY
   4       1      1 RN/NASA-CR-166346
            DISPLAY 04/6/1
        82N29958*#   ISSUE 20   PAGE 2882   CATEGORY 61   RPT#: NASA-CR-166346 NAS
        1.26:166346   CNT#: NAS2-10550   82/04/00  143 PAGES   UNCLASSIFIED
        DOCUMENT
UTTL: Automated verification of flight software.  User's manual
AUTH: A/SAIR, S. H.
CORP: General Research Corp., Santa Barbara, Calif.     AVAIL.NTIS   SAP: HC
        A07/MF A01
MAJR: /*AUTOMATIC FLIGHT CONTROL/*PROGRAM VERIFICATION (COMPUTERS)/*QUALITY
        CONTROL/*RELIABILITY ENGINEERING
MINR: / APPLICATIONS PROGRAMS (COMPUTERS)/ FORTRAN/ USER MANUALS (COMPUTER
        PROGRAMS)
ABA:   Author
```

NASA CONTRACTOR REPORT 166346

Automated Verification of Flight Software - User's Manual

S. H. Saib
General Research Corp.

Prepared for
Ames Research Center
under Contract NAS2-10550

National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, California 94035

ABSTRACT

AFVS (Automated Verifiction of Flight Software) is a collection of tools for analyzing source programs written in FORTRAN and AED. AVFS aids in improving the quality and the reliability of flight software by providing:

- Indented listings of source programs
- Static analysis to detect inconsistencies in the use of variables and parameters
- Automated documentation
- Instrumentation of source code
- Retesting guidance
- Analysis of assertions
- Symbolic execution
- Generation of verification conditions
- Simplification of verification conditions

This manual describes how to use AVFS in the verification of flight software.

AVFS is one important component of a Digital Flight Control System Verification Laboratory (DFCSVL) which has been established at NASA Ames Research Center, Moffett Field, California. The DFCSVL includes a PDP 11/60 processor and a palletized CAPS 6 based digital flight control system. Most of the AVFS files have been hosted in the Univac 1100 computer system in Santa Clara, California. Access to these files from the DFCSVL is provided via a direct link connecting the PDP 11/60 and the Univac 1100.

CONTENTS

FIGURES

FIGURES (Continued)

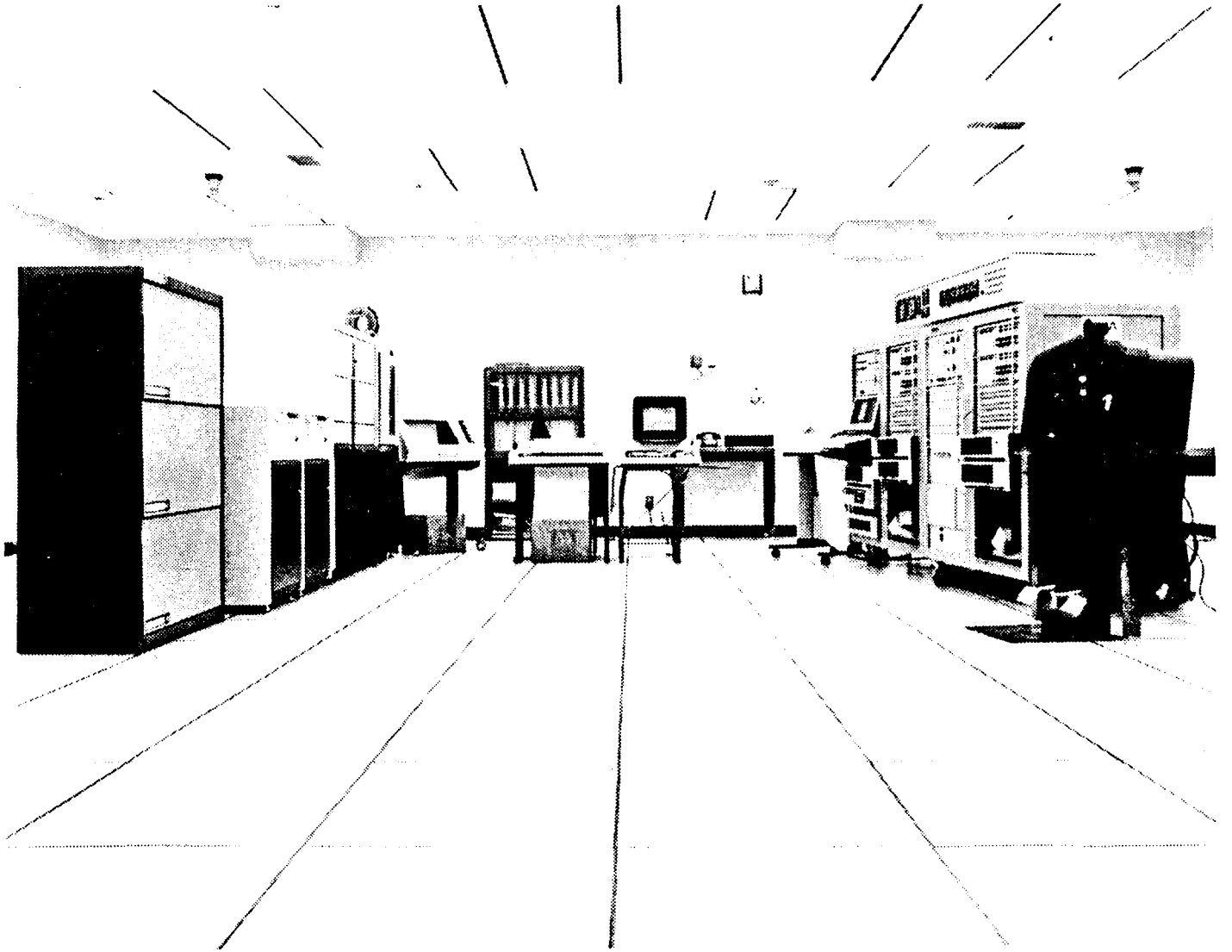


Figure 1. Digital Flight Control System Verification Laboratory

1 INTRODUCTION

AVFS (Automated Verification of Flight Software) is a collection of tools for analyzing flight software. AVFS is one important component of a Digital Flight Control System Verification Laboratory (DFCSVL), pictured in Figure 1 (DFCSVL block diagram shown in Figure 1.1), which has been established at NASA Ames Research Center, Moffett Field, California.

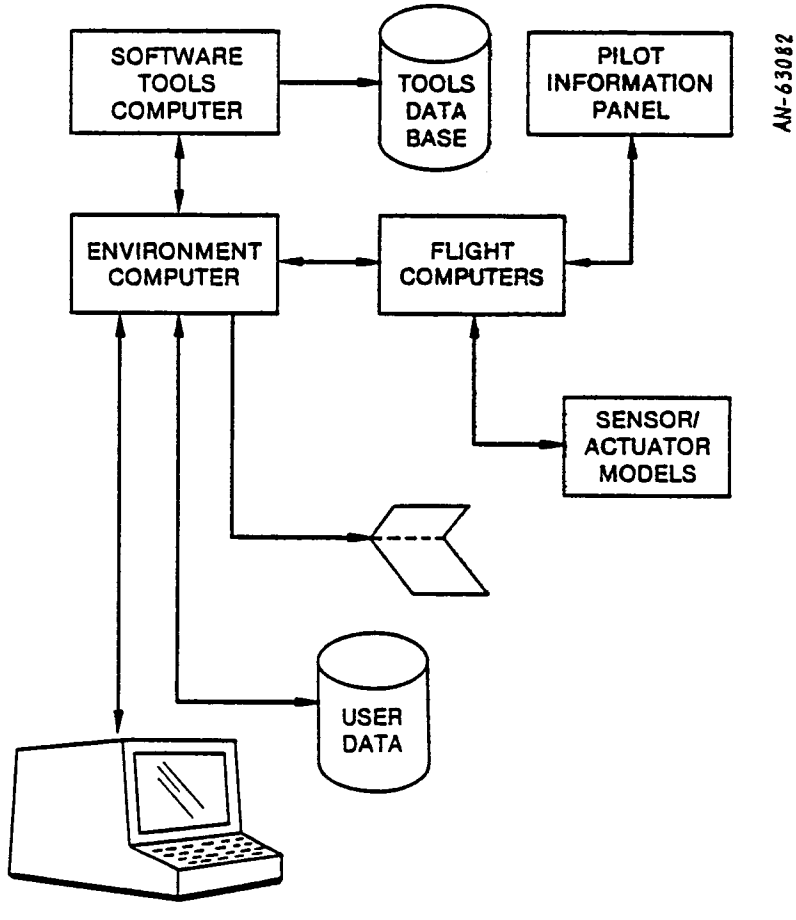


Figure 1.1 DFCS Verification Laboratory Block Diagram

The DFCSVL includes a PDP 11/60 processor (the environment computer) and a palletized CAPS 6 based digital flight control system (flight computers, pilot information panel, and sensor/acutator models). Most of the AVFS tools have been hosted in a Univac 1100 computer system (the software tools computer). Access to the AVFS tools from the environment computer is via a high speed data link over a telephone line.

AVFS analyzes source programs written in FORTRAN or AED (Automated Engineering Design). AED is an Algol-like programming language originally developed at MIT for the U.S. Air Force.

AVFS aids in improving the quality and reliability of flight software by providing:

- Indented listings of source programs
- Static analysis to detect inconsistencies in the use of variables and parameters
- Automated documentation
- Instrumentation of source code
- Retesting guidance
- Analysis of assertions
- Symbolic execution
- Generation of verification conditions
- Simplification of verification conditions

AVFS allows the addition of assertions to a program. These assertions make it possible for AVFS to include (1) a more comprehensive static analysis than is possible without the use of assertions, (2) more useful results from execution testing, and (3) formal verification by the generation and simplification of verification conditions. Figure 1.2 shows the various AVFS capabilities.

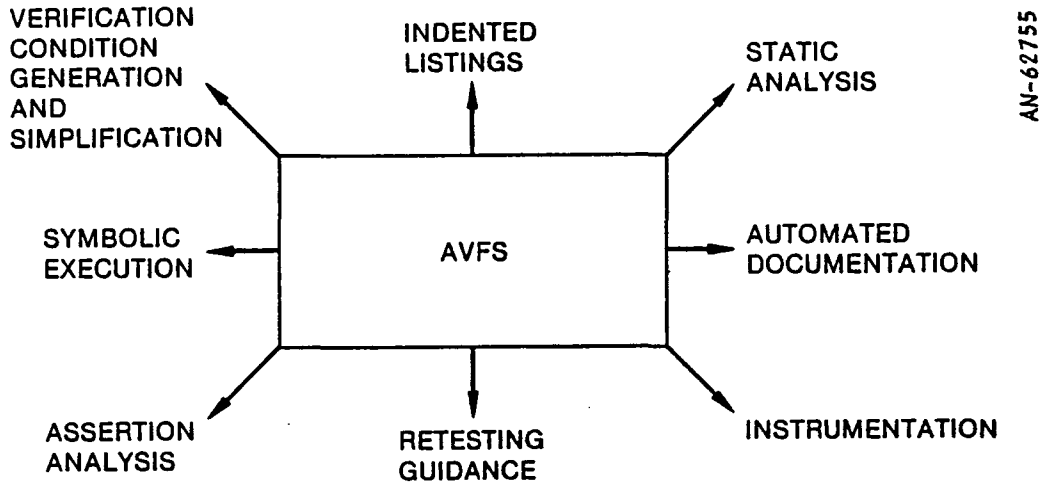


Figure 1.2. AVFS Capabilities

1.1 TESTING AND VERIFICATION SEQUENCE

The traditional method of detecting errors is limited to those errors which surface at compilation and during execution. AVFS, however, provides assistance to the user through system integration, testing, documentation, program verification, and maintenance.

As flight software is being written, the programmer can submit one or more modules for analysis by AVFS so that errors will be detected before they propagate through the entire system. Assertions should be included in the source text, as a valuable part of the analysis relies on information about the variables that are supplied in the assertions. Figure 1.3 illustrates the steps to be taken in validating a program with AVFS.

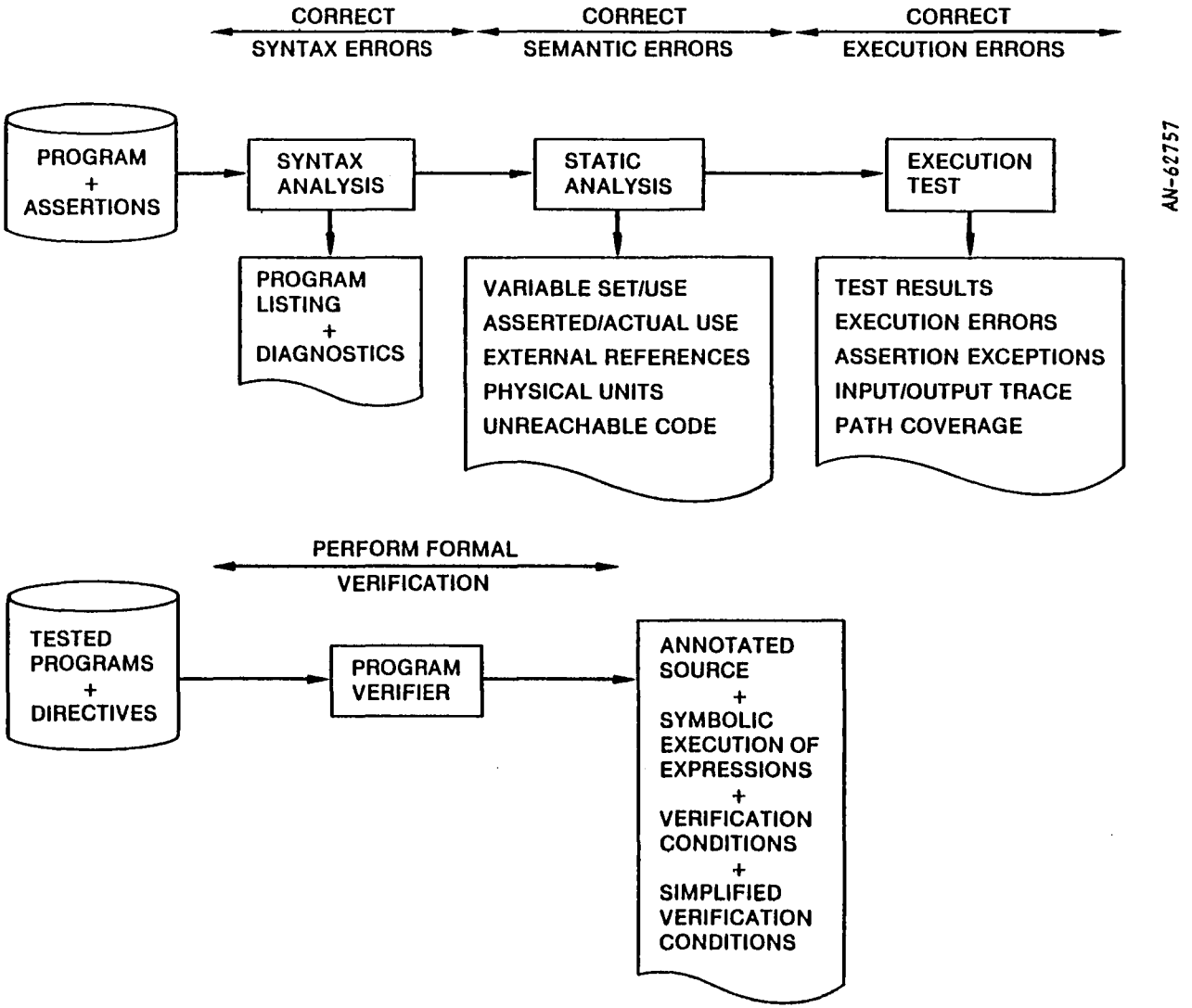


Figure 1.3. Steps in Validating a Program with AVFS

1.1.1 Syntax Errors

The first step in verifying a program is the elimination of syntax errors. If the source is written in FORTRAN, the FORTRAN compiler should be used to provide a listing and a diagnostic report. If the source is written in AED, the AED compiler should be used in a similar manner.

1.1.2 Static Analysis

Once syntax errors found by the compiler have been corrected, the source code is ready for the second step in verification (Fig. 1.2), which is static analysis to detect consistency errors. It is not necessary to wait for integration of the entire software system to begin using AVFS; one module at a time can be submitted for static analysis. By doing this errors will be detected as early as possible. The static analysis (which will be described in detail in Sec. 4) examines the source code for the following inconsistencies:

- Physical-units errors: operations on variables whose asserted physical units (feet, kilograms, etc.) do not correspond to actual use.

- Set/use errors: variables which are used before they are set to a value, or set to a value and then not used.

- Asserted/actual use errors: parameters which are used as output variables when they have been restricted, by assertions, to being input variables, or vice-versa.

- Graph errors: unreachable statements which cannot be executed because they are structurally disconnected from the main program.

- Loop errors: uninitialized loop variables.

- Poor practice errors: unused parameters, constant parameters, double parameters, expression parameters, function parameters.

1.1.3 Execution Testing

When a software system is fully integrated and the inconsistencies found by static analysis have been corrected, the next step is execution testing. When a program is ready for this step, AVFS offers assistance before, during, and after execution.

In preparation for testing, AVFS analyzes the program to determine the paths through the program. AVFS instruments the program by automatically inserting probes at appropriate points in the program to measure testing coverage, to check on assertion violations, or to trace variables in the code. During an execution test, these probes record information which can be used to report on execution coverage, assertion violations, execution time, and the value of important variables. When parts of a program have been found to be untested, another AVFS tool can provide help in selecting test cases to exercise these untested paths of the program by delineating the unreached portions of code. During the entire testing process, AVFS can be thought of as a partner, supplying a wide variety of automated aids to comprehensive testing.

1.1.4 Formal Verification

Following the correction of any errors found during static analysis and execution testing, the tested program is ready for formal verification, which is the final step in verifying a program (Fig. 1.2). When a program is executed, numerical data is supplied as input. The procedure for formal verification is to execute the program "symbolically," that is, using symbols as input data rather than specific numbers. The program is then verified or "proved" correct for a wider range of its variables then it is practical to assign to them during execution.

AVFS allows the symbolic execution of variables, expressions, or assertions. The symbolic execution of variables or expressions can be used to verify that a module's output is the same as the formula it has been specified to compute. The symbolic execution of assertion results in what has been termed verification conditions. By showing that the verification conditions for a module are true, it is said that the module has been formally verified with its assertions.

1.2 USER'S MANUAL ORGANIZATION

This manual describes how to use AVFS as an aid from the beginning to the end of the software development cycle. Information is presented in the order that the user is expected to need it. Section 2 is an overview of the type of aid AVFS provides. Section 3 explains how to use AVFS (commands, files, etc.). Section 4 describes each of the AVFS capabilities. Section 5 lists AVFS constraints. When the user's program has been instrumnted by AVFS and is ready for testing, special commands are needed to generate execution coverage analysis reports. These commands are described in Sec. 6.

Appendix A contains a summary of the AVFS commands. Tables listing the files used in AVFS processing are in Appendix B. Job streams for the Univac 1110 in Appendix C.

2 AVFS OVERVIEW

This section contains an ovreview of the way in which AVFS can aid the user not only in creating the code but during testing and documentation. The information presented here about what AVFS does is very general; the following sections contain more complete details of the full power of AVFS and how to use it.

Figure 2.1 shows how AVFS fits into the software development cycle to augment software analysis and testing. The additional features are indicated by diagonal lines. The user's source code can be analyzed by AVFS and the results will be presented in reports which help the user decide if acceptance criteria are met. AVFS can also instrument source code prior to execution to provide a measure of test coverage, to provide automatic checks on the behavior of a program and to trace module variables. AVFS can be used to perform formal verification of formulas and assertions via symbolic execution.

The usual program analysis and testing sequence is shown in Fig. 2.2. AVFS analyzes either FORTRAN or AED code and generates the following information:

- An enhanced listing of each module
- A static analysis of each module
- Interface data and module relationships
- Information about each module in brief form
- Structural information about each module
- Trace information about variables
- Assertion violations
- Assistance in retesting
- Symbolically executed formulas
- Verification conditions

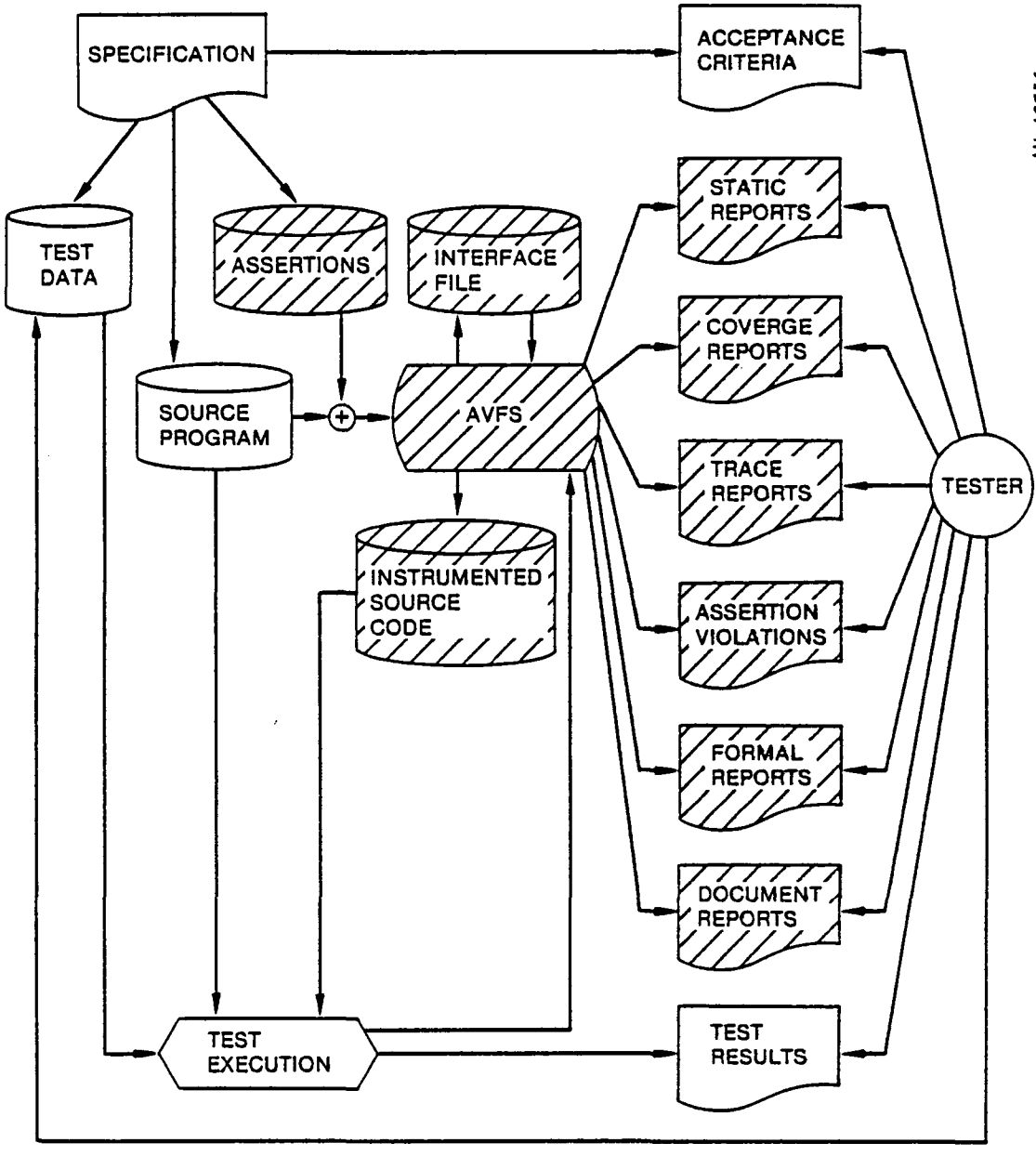


Figure 2.1. Software Verification Augmented by AVFS

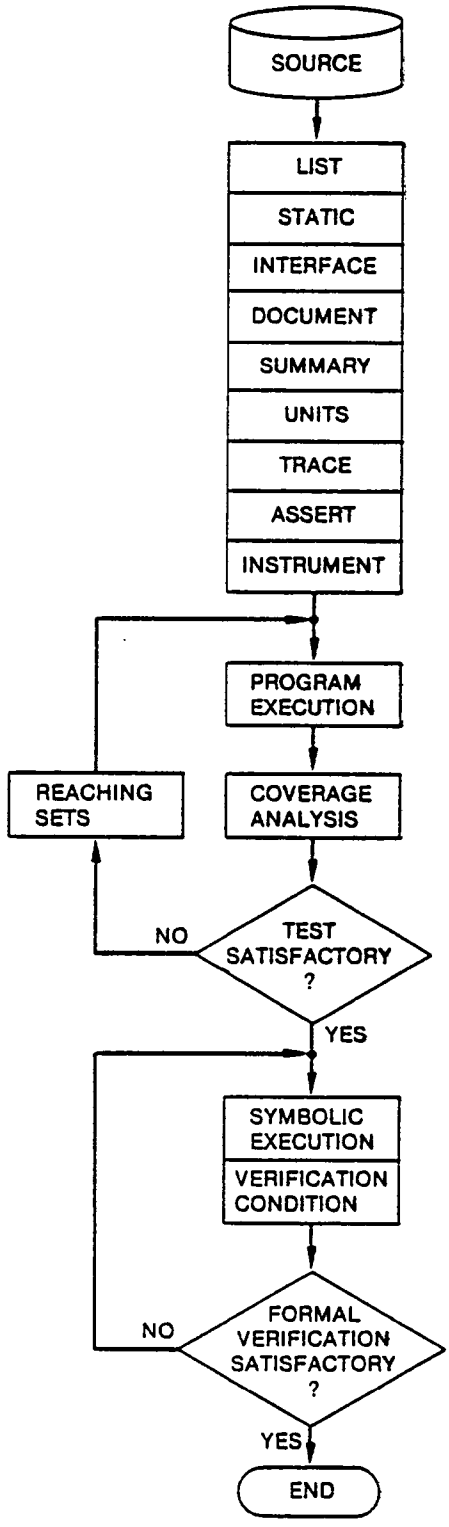


Figure 2.2. Sequence of Source Program Analysis, Test, and Formal Verification

The TRACE, ASSERT, and INSTRUMENT functions prepare the user's program for execution testing. Utilizing the knowledge it obtains about the structure of the program, AVFS can instrument the user's program by inserting software probes in each path. In addition, INPUT and OUTPUT assertion statements which list selected variables can be added to a module and AVFS will automatically generate code to output the values of these variables during execution of the modules. ASSERT statements which place conditions on selected variables will cause AVFS to automatically generate code to report on assertion violations during module execution.

When an instrumented module is executed, data is recorded each time a path is traversed. For FORTRAN programs, the path data is stored on a file for later analysis by a coverage analyzer on the Univac. For AED programs, the path data is stored in the CAPS memory for later analysis by a coverage analyzer on the PDP 11/60. In either case, these reports show the user where to focus retesting. AVFS can make further tests easier by furnishing a Reaching Set Report to list the source code on paths which were not executed. The user can then determine the values that must be assigned to the variables on these paths in order to reach the set of untested statements. The program is executed again and the procedure is repeated until the user is satisfied that testing is complete.

After execution testing, the user can formally verify the program by symbolically executing expressions or assertions. This process will result in formulas which can be checked against a specification or in verification conditions which can be proved to be valid. During the formal verification process, the user will often alter the program or assertions to validate the program until verification is complete.

During analysis of a program by AVFS, an interface file can be generated and stored. The interface file is the key to multiple module

interface checks. The interface report lists each module name which has been anlayzed and indicates changes in interface properties, such as parameters added or deleted, changes in type or use of parameters, changes to common, and changes to invocations.

FORTRAN and AED programs can be made up of single or multiple modules. In the analysis of a FORTRAN program, the modules are placed in a single data file and analyzed together with a single execution of the tool. In the analysis of an AED program, the modules are placed in elements of a program file and analyzed with separate executions of the tool for each module. Information for multiple module reports is gathered during the analysis of the separate modules and printed after all the modules have been analyzed. The analysis of a large FORTRAN program often results in a large listing for which a report index has been provided. An example of a report index is shown in Fig. 2.3.

```
REPORT INDEX...
                                                        PAGE        MODULE NAME

    MULTI-MODULE REPORTS

              INTERFACE CHANGES                            1
              INVOCATION SUMMARY                       16- 17
              COMMON MATRICES                          18- 26
               I/O STATEMENTS                             27
              CROSS REFERENCE                             28

          SUBROUTINE PTRSTR ( MODULE, ISTMT, IRETRN )                 PTRSTR

              SYMBOLS                                      2
              CROSS REFERENCE                              3
              INVOCATION SPACE                             4
              INVOCATION BANDS                             5

          SUBROUTINE SDBASA ( MODULE, ISTMT, IRETRN )                 SDBASA

              SYMBOLS                                  6-  8
              CROSS REFERENCE                          9- 12
              INVOCATION SPACE                        13- 14
              INVOCATION BANDS                            15
              CROSS REFERENCE                         29- 31

...WRITING INTERFACE LIBRARY

...  4075 WORDS WRITTEN
```

Figure 2.3. Report Index

3 USING AVFS

AVFS is a software system which reads as input a user's source program in order to perform a number of functions on the source program (Fig. 3.1). The source program may be written in FORTRAN or AED and may be read into AVFS directly from a standard input device (card reader, terminal, or tape) or from a previously prepared source text file.

The user tells AVFS which functions to perform through a series of commands. These command can be sent to AVFS via a standard input device or from a previously prepared command text file.

For FORTRAN, the analysis of several modules can be accomplished if they are part of a single data file. For AED, modules are stored as elements of a program file.

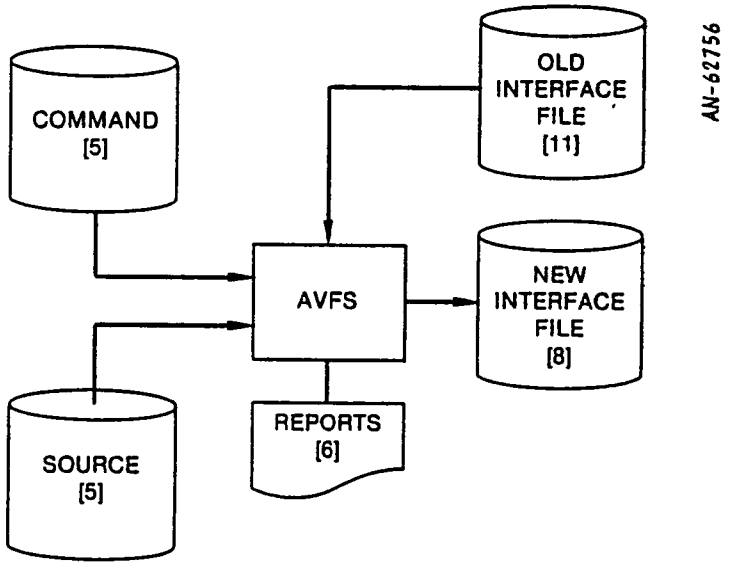


Figure 3.1. AVFS Processing

During an interface run, the source code is analyzed and stored in a data base known as the interface file. The interface file stores the module name and information on its interface to other modules such as information on invoked subprograms and commons.

During an instrumentation run, the structure of the source code is analyzed for the paths between the decision points. Each module's first path always starts at program entry and includes all the statements until a decision or the end of the routine is reached. These paths form the basis for instrumentation and subsequent execution coverage analysis. AVFS uses the path information to produce reports.

AVFS can operate on the source code which it reads during its run and produce reports. AVFS can also produce reports using information (from previously processed programs) that is in the data base called the interface file. The interface file contains the external characteristics of previously analyzed programs and allows reports to be generated for only the modules under anlaysis. This allows re-analysis of a changed system at a far lower cost than if the complete system were re-analyzed. It is a good practice to use separate interface files for separate software projects.

The logical unit numbers for the files are shown in brackets in Fig. 3.1. Appendix B has a list of all the file names and numbers used by AVFS.

Before submitting source text for analysis, the user should take certain preliminary steps:

1. The source text should be compiled by a FORTRAN or AED compiler to confirm it is free of syntax errors.

2. The program should be executed if it will be dynamically tested.

AVFS processing is specified by commands. For FORTRAN, either an OPTION or REPORT command is required for each run. For AED, a separate XQT command is required for each function. Other commands are used to assign appropriate files, to change default values, or to make certain specifications. The order of commands is important to AVFS. When multiple commands are given to the FORTRAN AVFS, the following order must be observed:

```
RESTART.
EXPAND.
FILE,PUNCH = <file number>.
INSTRUMENT,PUNCH,PROBE,(<file number>).
FIRSTLINE = (<run stream command>).
OPTION = <option list>.
REPORT = <report list>.
FOR MOUDLES = (<name 1>,<name 2>, ... ).
TESTBOUND,MODULE = (<name>),STATEMENT = <number>.
REACHING SET,MODULE = (<name>),TO = <DD-path number>,
                                FROM = <DD-path number>{,ITERATIVE}.
```

Each command consists of a sequence of terms separated by a comma or an equal sign. These commands--one to a card, or input record--are free form; blanks are ignored. The commands may be abbreviated by using the first four letters of the first word in the command. The names of the options also may be abbreviated in the same way.

Table 3.1 shows the conditions under which the RESTART or EXPAND commands should be used. The first line gives the case where it is desired to analyze new source without using an existing interface file. This case covers the situation of a first run of the tool and the situation when it is desired to analyze a module in isolation without using an interface file. In this case, neither command is used. Page C-1 contains a sample job control for this case.

TABLE 3.1

EXPAND - RESTART COMMAND CONDITIONS

New Source?	Existing Interface File?	Command to Use
Yes	No	---
Yes	Yes	EXPAND
No	No	Do Nothing
No	Yes	RESTART

The second line gives the case when there is an existing interface file and there is new source code to be analyzed by the tool. In this case, the EXPAND command is used. Page C-2 contains a sample job control for this case.

The third line gives a case that should not occur. If there is neither source nor an interface file to process, then no commands should be given.

The fourth lines gives the case where there is no new source, but it is desired to reanalyze an existing interface file which was created from a previous execution of the tool. Page C-3 contains a sample job control for this case.

The FILE, INSTRUMENT, and FIRSTLINE commands are used with the INSTRUMENT or INPUT/OUTPUT options. Table 3.2 shows the conditions under which these commands should be used. Normally, neither the FILE command nor the INSTRUMENT command is used. The FIRSTLINE command is usually used.

The FILE command is used to change the default file number for where the tool sends the instrumented code. If unit 9 is suitable for the instrumented code, the FILE command need not be specified. The job control sequence shown on page C-4 instruments a source program and places the instrumented source one unit 10.

The INSTRUMENT command is used to change the default file number for where the tool collects data during execution of the instrumented code. If unit 12 is suitable for the data collection file, the INSTRUMENT command need not be used. The job control control sequence shown on page C-4 instruments a source program to collect data on unit 20.

The FIRSTLINE command is used to insert the FORTRAN compilation Univac job control between modules. Each module on the file must have a job control card to cause compilation and storage of the relocatable element in a program file. In the example shown on page C-4, the ASCII compiler, FTN, is invoked to store the relocatable element in the temporary program file, TPF$.

The OPTION command is used to select from one of many possible processing options for the tool. Examples of the use of the STATIC option are shown in C-1, C-2, and C-3. Examples of the use of the INSTRUMENT and LIST options are shown in C-4. The various options are discussed in detail in Sec. 4.

The REPORT command is used to obtain the flowgraph or picture report under the DOCUMENT option. An example of the use of this command is given in C-5.

TABLE 3.2

FILE - INSTRUMENT - FIRSTLINE

COMMAND CONDITIONS

Is unit 9 suitable for the instrumented code output by the tool?	Yes	do not use FILE command
	No	use FILE command
Is unit 12 suitable for the data collected during an execution of instrumented code?	Yes	do not use INSTRUMENT command
	No	use INSTRUMENT command
Do Univac job control cards need to be inserted between modules?	Yes	use FIRSTLINE command
	No	do not use FIRSTLINE command

The FOR MODULES command is used to instrument or report on selected modules from among a group of modules that have been read into the tool. Page C-6 shows an example where three modules named MAIN SORT and CALC are statically analyzed from among the modules input to the tool.

The TESTBOUND command is used in the instrumentation of a program when it is desired to specify a statement at which a test case to be counted. When the TESTBOUND command is omitted, the test boundary is taken to be the first statement in the main program. Page C-4 shows an

example where the TESTBOUND command was used to specify that line 10 of the main program is defined as a test case boundary.

The REACHING SET command is used after an instrumentation run has found that some or several paths were not executed. The REACHING SET command is used together with the REACHING SET option to cause a set of paths to be printed between the two specified paths. A user can then concentrate on causing these paths to be executed. Page C-7 shows the use of the REACHING SET command to list the paths between path 3 and path 7 in Module SORT.

3.1 INTERFACE FILE

3.1.1 FORTRAN

The FORTRAN AVFS differentiates between when source code is being read into the tool and when the interface file is being used by itself to produce intermodule reports.

- On the first run of a FORTRAN AVFS, save the interface file that is created on unit 8. Do not use the RESTART or EXPAND command on the first run.

- On the second and subsequent runs of a FORTRAN AVFS, assign the saved file to unit 11. If there is just to be a re-analysis of the file, use the RESTART command. If there is new or changed source to be read in, use the EXPAND command instead. A new interface file will be created on unit 8. This file may be saved.

3.1.2 AED

The AED AVFS interface analysis tool will read an old interface file from unit 8 and new source from unit 5. If there is new source, the interface file will be generated on unit 11. The invocation of the AED interface analysis requires no additional commands.

3.2 INSTRUMENT FILE

The primary function of several AVFS tools is to produce source output on unit 9 (FORTRAN) or unit 30 (AED). This source normally goes to a temporary file (which may be saved after the AVFS run). If unit 9, the default assignment for the source output file is not appropriate for a FORTRAN program (usually because the program uses that unit for its own purposes); it may be reassigned with the command

FILE,PUNCH = <file-number>.

where <file-number> is the desired file number. (See Appendix B for assigned file numbers).

In AED, it is never necessary to change the file number, because the AED program will not be executing on a Univac.

3.3 FIRSTLINE

When a FORTRAN program is to be INSTRUMENTED, the user's instrumented source program will be written to unit 9. The user may use the command,

FIRSTLINE = (<run stream command>)

to specify a Univac run stream command that will be added as the first line of every element of the source program.

For example, when a FORTRAN source program will be instrumented, then compiled and executed, the user could use the command

FIRSTLINE = (@FOR<I TPF$.+).

AVFS will insert the Univac command,

@FOR,I TPF$.<element name>

as the first line of each element with the appropriate element name following TPF$. If the Univac FTN compiler is being used, the command could be

FIRSTLINE = (@FTN,I TPF$.+).

If the source code is written in AED, the FIRSTLINE command is not needed. Each AED element is in its own program element and is processed separately.

3.4 PROCESSING OPTIONS

The processing functions for FORTRAN are handled by the OPTION command. The processing functions for AED are handled by different execute commands. The possible options are as follows:

FORTRAN	AED
LIST	GRC*LIST.LIST
STATIC	GRC*STATIC.STATIC
DOCUMENT	GRC*CROSS.CROSS
	GRC*SYMBOL.SYMBOL
	GRC*INVOKE.INVOKE
	GRC*TREE.TREE
	GRC*DEPEND.DEPEND
	GRC*GLOBAL.GLOBAL
SUMMARY	GRC*PROFILE.PROFILE
UNITS	GRC*UNITS.UNITS
INPUT/OUTPUT	GRC*TRACE.TRACE
ASSERT	GRC*ASSERT.ASSERT
INSTRUMENT	GRC*INST.INST
REACHING SET	GRC*REACH.REACH
VCG	GRC*VCG.VCG

- LIST - Produces an enhanced listing of each module
- STATIC - Produces a static analysis of each module
- DOCUMENT - Produces six reports for each module; the cross reference, symbol tables, dependence matrix, calling-tree, invocations report, and global cross reference reports (for AED, the reports are requested separately)

- SUMMARY - Provides an analysis of statements
- UNITS - Checks for consistency of units that have been defined in a UNITS assertion
- INPUT/OUTPUT - Translates INPUT/OUTPUT assertions
- ASSERT - Translates logical assertions
- INSTRUMENT - Instruments the source code
- REACHING SET - Provides assistance in path identification
- VCG - Symbolic execution and verifictaion condition generation

In the FORTRAN tools, the command form is

OPTION{S} = <option list>

where more than one option may be specified. There must be at least one option. Multiple options may be separated by commas on a single 80-character line. More than one OPTION line may be submitted. Detailed descriptions of each option with examples of the reports that the option produces may be found in Sec. 4.

In the AED tools, the Univac command is

@XQT option

Separate commands may be given to obtain multiple reports in a single job. The examples of each AED option may be found in Sec. 4.

3.5 REPORT

Selection of individual reports to be produced can be accomplished by the REPORT command for FORTRAN programs. The separate reports for AED are handled by different execute commands. The possible reports are as follows:

REPORT NAME	FORTRAN COMMAND	AED COMMAND
COMMONS	CO	
PROFILE	PR	GRC*PROFILE.PROFILE
INVOCATIONS	L	GRC*INVOKE.INVOKE
COMMON MATRICES	CO/E	
CALLING TREE	B	GRC*TREE.TREE
SPACE	SP	GRC*DEPEND.DEPEND
SYMBOLS	SY	GRC*SYMBOL.SYMBOL
I/O STATEMENTS	R	
CROSS	CR	GRC*CROSS.CROSS
FLOWGRAPH	PI	GRC*FLOW.FLOW

In the FORTRAN tools, the command form is

REPORT = <report list>

where more than one report may be specified. Blanks within the list are ignored. This command may appear within the command stream in any location that is valid for the OPTION command. The REPORT command must fit in 80 characters or separate commands can be given.

AED does not have common reports or I/O statement reports. AED has no I/O statements and only blank common. The blank common is saved on the interface library.

Normally the OPTION command is used. The REPORT command is only used to restrict reports. If the same report is requested via an OPTION and a REPORT command, the report will not be duplicated.

3.6 FOR MODULES

AVFS will normally analyze all modules. In AED, each module is individually analyzed via a separate command. In FORTRAN, a data file

may contain several modules which are not all to be analyzed. The FOR MODULES command allows the selection of individual FORTRAN modules. This command has the form

FOR MODULES = (<name 1>,<name 2>, ...).

where <name 1> and <name 2> are FORTRAN names for modules which have been input to AVFS. Main programs which do not have a program card are given the name MAIN. Only one FOR MODULES command per AVFS run is allowed.

4 OPTION DESCRIPTIONS

This section contains descriptions of each option which may be selected by the user to instruct AVFS as to which type of processing to perform on the modules being input. An example of each type of report generated by an option follows each description. Table 4.1 shows the AVFS options and suggested uses for each.

4.1 LIST

The LIST option produces a source listing which shows the statement line number and the automatically indented source code. All references to statements in reports developed by AVFS are keyed to statement line numbers and module name.

The indented listing clearly indicates the control structures and makes the program much more readable, not only to the original programmer, but especially to someone unfamiliar with the code who is trying to understand it. The indented source listing on the output file is the sole report from the LIST option. Figure 4.1 illustrates a sample listing for a FORTRAN program and for an AED program.

Some of the other options also provide listings in a different format. Examples of those listings are presented in the section describing those options.

FORTRAN Command	Report
OPTION=LIST or	FORTRAN Listing (Fig. 4.1a)
@XQT,L GRC*AVFS.AVFS	
See Appendix C-8 for complete JCL example	

AED Command	Report
@XQT GRC*LIST.LIST	AED Listing (Fig. 4.1b)
See Appendix C-11 for complete JCL example	

TABLE 4.1

AVFS PROCESSING OPTIONS WITH SUGGESTED USES FOR EACH OPTION

	Options									
Suggested Uses	LIST	STATIC	DOCUMENT	SUMMARY	UNITS	INPUT/OUTPUT	ASSERT	INSTRUMENT	REACHING SET	VCG
Software Documentation	X		X	X	X	X	X			X
Maintenance	X	X	X	X	X	X	X	X		X
Implementation	X	X	X	X	X	X	X	X		X
Obtain Interface Data		X	X	X						
Trace Ranges of Variables						X				
Check Variables					X		X			X
Execution Test						X	X	X	X	
Incomplete Test Coverage						X		X	X	
System Test Information	X		X	X	X		X			
Single Module Information	X	X	X	X	X		X			X
Code Changes	X	X	X	X	X		X			X
Unknown Behavior		X	X		X	X	X	X		X
Integration		X	X		X	X		X		
Acceptance		X	X	X	X		X	X		X
Dimensional Analysis	X				X					
Symbolic Execution										X
Formal Verification							X			X

```
STATEMENT LISTING                    SUBROUTINE EXAMPL (INFO,LENGTH)

 STMT NEST  LINE  SOURCE...                                          ...SOURCE TAB
-------------------------------------------------------------------------------
    1          1       SUBROUTINE EXAMPL (INFO,LENGTH)
               2  C                                                  EXAMPL2
               3  C    ILLUSTRATION OF DMATRAN SYNTAX                EXAMPL3
               4  C                                                  EXAMPL4
    2          5       IF (INFO.LE.10 .AND. LENGTH.GT.0)THEN         EXAMPL5
    3    1     6       .   CALL CALLER ( INFO )                      EXAMPL6
    4          7       ELSE                                          EXAMPL7
    5    1     8       .   LENGTH=50                                 EXAMPL8
    6          9       END IF                                        EXAMPL9
    7         10       CASE OF (INFO+6)                              EXAMPL10
    8         11       CASE (14)                                     EXAMPL11
    9    1    12       .   LENGTH=LENGTH-INFO                        EXAMPL12
   10         13       CASE (17)                                     EXAMPL13
   11    1    14       .   DO WHILE (INFO.LT.20)                    EXAMPL14
   12    2    15       .   .   DO UNTIL (LENGTH.LE.INFO)            EXAMPL15
   13    3    16       .   .   .   INVOKE (COMPUTE LENGTH)          EXAMPL16
   14    3    17       .   .   .   IF (LENGTH.GE.30) THEN           EXAMPL17
   15    4    18       .   .   .   .   INVOKE (PRINT-RESULTS)       EXAMPL18
   16    3    19       .   .   .   END IF                           EXAMPL19
   17    2    20       .   .   END UNTIL                            EXAMPL20
   18    2    21       .   .   INFO=INFO+1                          EXAMPL21
   19    1    22       .   END WHILE                                EXAMPL22
   20         23       CASE ELSE                                     EXAMPL23
   21    1    24       .   DO WHILE (LENGTH.GT.0)                   EXAMPL24
   22    2    25       .   .   INVOKE (COMPUTE LENGTH)              EXAMPL25
   23    1    26       .   END WHILE                                EXAMPL26
   24         27       END CASE                                      EXAMPL27
   25         28       BLOCK (PRINT-RESULTS)                         EXAMPL28
   26    1    29       .   WRITE (6,1)INFO,LENGTH                   EXAMPL29
   27    1    30     1 .   FORMAT (10X,I5,20X,I5)                   EXAMPL30
   28         31       END BLOCK                                     EXAMPL31
   29         32       BLOCK (COMPUTE LENGTH)                        EXAMPL32
   30    1    33       .   LENGTH = LENGTH -10                      EXAMPL33
   31         34       END BLOCK                                     EXAMPL34
   32         35       RETURN                                        EXAMPL35
   33         36       END                                           EXAMPL36
```

This report contains the indented module listing with statement numbers, source line numbers, and nesting levels.

Figure 4.1a. FORTRAN Listing

```
14   DEFINE PROCEDURE B.LONG.COM TOBE
15
16   COMMENT ITERATION RATE = 10 / SEC ;
17        BEGIN
18        NORM.ACC = VOTER(NORM.ACC.PTR) ... VOTE NORMAL ACCELERATION ;
19        NORM.ACC.LP = DLIMIT(NORM.ACC.LP+(NORM.ACC-NORM.ACC.LP)*.5D-2,
20         .442550D-1)              ... NORM.ACC.LP IS SUBTRACTED FROM
21                                      NORM.ACC BY OTHER PROCEDURES FOR A 20
22                                      SEC WASHOUT ON NORM.ACC ;
23        TAS = TAS.MA              ... BUFFER TAS FOR GAIN PROGRAMERS ;
24        KVTAS =                   ... GAIN PROGRAMER //
25        IF TAS < .146484          ... 150 KTS //
26            THEN .5
27            ELSE IF TAS < .341797     ... 350 KTS //
28                     THEN .75-TAS/.585936
29                     ELSE .166667;
```

This report contains the indented module listing with statement line numbers.

Figure 4.1b. AED Listing

4.2 STATIC

The static analysis available in AVFS is designed to uncover inconsistencies in the use of variables and inconsistencies in the structure of a program. When an inconsistency is found, it indicates the existence of an error or the possibility of an error.

Static analysis is divided into single module analysis, units analysis, and interface analysis. Assertions are not required for static analysis, but a more comprehensive analysis is possible when assertions have been added to the user's source program. A complete static analysis checks for the following types of errors:

Set and Use Checking

Variables used before being set to a value or set and not used.

Loop Checking

Uninitialized loop variables.

Type Checking

Possible misuse of variables in assignments.

Graph Checking

Unreachable statements.

Interface Checking

Checking of actual invocations against formal declarations; checking for consistency in number of parameters and type.

Input/Output Checking

Check that asserted use of a variable is the same as its actual value

UNITS

Units assertions can be inserted into a program so that consistency checks can be made on the use of units. Each variable for which units are to be specified has its units declared in the form:

COMMENT UNITS <variable> = <units expression>;

For example, to state that the variable named SPEED has the units of FEET/SEC, write

COMMENT UNITS SPEED = FEET/SEC;

To state that the variable named DIST has the units of FEET, write

COMMENT DIST = FEET;

The units analyzer will check that operations on variables which have specified units is done in a consistent manner. That is, if an assignment was made such as

SPEED = DIST;

the units analyzer would report on a units error stating that

FEET = FEET/SEC;

was attempted.

Units are combined symbolically across multiplication and division to form new units. Checks are made across addition, subtraction and assignment operations to ensure units consistency.

After the analysis is complete, the units for each variable is listed in the units table.

The example in Fig. 4.4 shows units specified for speed, distance, time, work and force. The first assignment to speed specifies that it will have the units of SAMPLE. Since SAMPLE does not have its units specified, no checking is done. The second assignment statement shows what is printed when there is an error detected in the units. The units

of DIST*TIME are FEET*SEC which is printed on the right of the equals sign. The units of SPEED are FEET/SEC which is printed on the left hand side of the equals sign. Since the units on the left do not match the units on the right, an error is declared. The other two assignments have the correct units.

Input/Output

Input/Output assertions are used in a static analysis to check for consistency between the intended use of a variable and the actual use of a variable.

Variables which provide input data to a module should be asserted with an input assertion.

An input assertion has the form

COMMENT INPUT <type> <variable>;

For example, the assertion

COMMENT INPUT REAL HEIGHT;

states that the vraiable named HEIGHT is an input to the module.

Variables which provide output data from a module should be asserted with an output assertion. An output assertion has the form

COMMENT OUTPUT <type> <variable>.

Variables which are used both as input and output are asserted in both assertions.

FORTRAN Command	Report
OPTION=STATIC,LIST.	Static Analysis (Fig. 4.2)
or @XQT,LS GRC*AVFS.AVFS	
See Appendix C-6 for complete JCL example	

AED Command	Report
@XQT GRC*SYMBOL.SYMBOL	Static Analysis (Fig. 4.3)
See Appendix C-12 for complete JCL example	

AED Command	Report
@XQT GRC*UNITS.UNITS	Units Analysis (Fig. 4.4)
See Appendix C-13 for complete JCL example	

AED Command	Report
@XQT GRC*INTER.INTER	Interface Analysis (Fig. 4.5)
See Appendix C-14 for complete JCL example	

```
STATIC ANALYSIS                    SUBROUTINE CIRCLE ( AREA )

 STMT NEST  LINE  SOURCE...                                          ...SOURCE TAB

    1          1     SUBROUTINE CIRCLE ( AREA )
    2          2     INTEGER AREA
    3          3     DATA PI / 3.1416 /
    4          5     INPUT (/R/ RADIUS )
    5          6     RADIUS = DIAMTR / 2
                     -------------------------------------------------------------
                     -                      SET/USE ERROR                        -
                     - VARIABLE DIAMTR     USED BUT NEVER SET       REFER TO STATEMENT(S)-
                     -  5
                     -------------------------------------------------------------
    6          7     AREA = PI * RADIUS**2
                     -------------------------------------------------------------
                     -                      MODE WARNING                         -
                     - LEFT HAND SIDE HAS MODE INTEGER   RIGHT HAND SIDE HAS MODE REAL -
                     -------------------------------------------------------------
    7          8     IF ( AREA .GT. 50 ) THEN
    8    1     9     .   CALL PRINT ( AREA )
                     -------------------------------------------------------------
                     -                      MODE WARNING                         -
                     -PARAMETER  1 OF PRINT       ACTUAL PARAMETER HAS MODE INTEGER -
                     -                            FORMAL PARAMETER HAS MODE REAL -
                     -------------------------------------------------------------
                     -------------------------------------------------------------
                     -                      CALL ERROR                           -
                     -         PRINT      CALLED WITH  1 ACTUALLY HAS  2 ARGUMENTS -
                     -------------------------------------------------------------
    9         10     END IF
   10         11     OUTPUT (/R/ AREA )
   11         13     RETURN
   12         14     CALL STACK ( RADIUS, AREA )
                     -------------------------------------------------------------
                     -                      GRAPH WARNING                        -
                     -       STATEMENT  12  IS UNREACHABLE OR IS IN AN INFINITE LOOP -
                     -------------------------------------------------------------
   13         15     END
                     -------------------------------------------------------------

                      STATIC   ANALYSIS   SUMMARY      ERRORS  WARNINGS
                      ------------------------------   ------  --------
                      GRAPH CHECKING                        0         1
                      CALL CHECKING                         1         0
                      MODE CHECKING                         0         2
                      SET/USE CHECKING                      1         0
      CALL CHECKING WAS NOT PERFORMED FOR THE FOLLOWING UNKNOWN EXTERNALS ...
      STACK
```

The FORTRAN Static Analysis Summary contains the warning and error messages interspersed in the code. Unknown subprograms are listed at the bottom of the report. A tabulation of errors and warnings is listed at the bottom. The STMT column lists the line numbers for each input source statement. The NEST column lists the nesting level for each statement in a control structure. The LINE column lists the line numbers for each generated source statement. Since input and output assertions require two generated lines, the LINE column differs from the STMT column.

Figure 4.2 Static Analysis - FORTRAN

NAME	CLASS	MODE	FIRST STMT.	TOTAL USES	LAST STMT.	ASSERTED USE	ACTUAL USE
A	PARAMETER	REAL	1	3	6		INPUT
N	PARAMETER	INTEGER	1	4	9		BOTH
ANS	PARAMETER	REAL	1	3	9		OUTPUT
I	LOCAL	INTEGER	2	5	7		
		- VARIABLE I				SET/USE ERROR USED BEFORE BEING ASSIGNED A VALUE	
O	LOCAL	INTEGER	2	2	4		
SUM	LOCAL	REAL	3	4	9		
		- VARIABLE SUM				SET/USE ERROR USED BEFORE BEING ASSIGNED A VALUE	

Set use checking and input/output analysis is reported after the listing is presented for AED.

Figure 4.3. Static Analysis - AED

```
 59
 60 COMMENT UNITS SPEED = FEET/SEC;
 61 COMMENT UNITS DIST  = FEET;
 62 COMMENT UNITS TIME = SEC;
 63 COMMENT UNITS WORK = POUND*FEET;
 64 COMMENT UNITS FORCE = POUND;
 65
 66         SPEED = SAMPLE;
 67         SPEED = DIST * TIME;
*****units error*****
FEET/SEC=FEET*SEC
 68         DIST = SPEED * TIME;
 69         WORK = FORCE * DIST;
 70
 71


      UNITS TABLE
SPEED                          FEET/SEC

DIST                           FEET

TIME                           SEC

WORK                           POUND*FEET

FORCE                          POUND
```

Inconsistent units are reported during units analysis.

Figure 4.4. Units Analysis - AED

```
INTERFACE REPORT

MODULE                    TYPE OF CHANGE
-------------------------------------------------------------------

A.FORE.EXEC               NEW MODULE

A.BAK.EXEC                UPDATED MODULE
                          **** PARAMETER LENGTH CHANGE

A.YAW.END                 UPDATED MODULE
                          **** COMMON TYPE CHANGE


INTERFACE ANALYSIS

MODULE                    TYPE OF ERROR
-------------------------------------------------------------------

A.MAKE.IT                 PARAMETER LENGTHS INCONSISTENT
                          ACTUAL - 2 PARAMETERS
                             A.BAK.EXEC (C,D)
                          FORMAL - 3 PARAMETERS
                             A.BAK.EXEC (A,B,C)

   1 INTERFACE ERRORS
```

Checking between modules on a library is performed to produce an interface analysis along with changes to the interface.

Figure 4.5. Interface Analysis - AED

4.3 DOCUMENT

The DOCUMENT option generates a set of reports for individual modules and multiple modules. Figures 4.6 - 4.16 contain examples and a description of each report.

The set of reports are useful for maintenance and testing. Together with the execution coverage reports, they help to identify which modules require retesting when changes are made to the source code. The cross reference reports are particularly useful in finding where variables are set in order to alter test cases, and also where a variable is being used that is affected by a change in a module. The flowgraph or picture report is useful for breaking up large FORTRAN programs.

AED programs have no I/O statements and only one common block. There is no AED I/O Statements Report or commons matrices. The AED commons and externals cross references are combined in a single global cross reference.

Report	FORTRAN Command	AED Command
Symbols	OPTION=DOCUMENT (Fig. 4.6a)	GRC*SYMBOL.SYMBOL (Fig. 4.6b) Appendix C-12
Cross Reference	OPTION=DOCUMENT (Fig. 4.7a)	GRC*CROSS.CROSS (Fig. 4.7b) Appendix C-15
Invocation Space	OPTION=DOCUMENT (Fig. 4.8a)	GRC*INVOKE.INVOKE (Fig. 4.8b) Appendix C-16
Invocation Summary	OPTION=DOCUMENT (Fig. 4.9a)	GRC*DEPEND.DEPEND (Fig. 4.9b) Appendix C-17
Invocation Bands	OPTION=DOCUMENT (Fig. 4.10a)	GRC*TREE.TREE (Fig. 4.10b) Appendix C-18

Common Matrices	OPTION=DOCUMENT (Fig. 4.11)	
I/O Statements	OPTION=DOCUMENT (Fig. 4.12)	
Commons Cross Reference	OPTION=DOCUMENT (Fig. 4.13)	GRC*GLOBAL.GLOBAL (Fig. 4.14b) Appendix C-19
Externals Cross Reference	OPTION=DOCUMENT (Fig. 4.14a)	GRC*GLOBAL.GLOBAL (Fig. 4.14b) Appendix C-19
Flowgraph	OPTION=DOCUMENT REPORT=PICTURE (Fig. 4.15)	GRC.FLOW.FLOW (Fig. 4.16)

The FORTRAN command

OPTION=DOCUMENT

maybe replaced with the UNIVAC command

@XQT,D GRC*AVFS.AVFS.

The complete JCL for the document command is shown on page C-9. This command will produce the nine reports shown in Figs. 4.6-4.14. The flowgraph report (Fig. 4.15) will also be generated if the command

REPORT=PICTURE

is added after the OPTION command or the XQT command.

The corresponding AED commands are listed beside the FORTRAN commands. In most cases a single AED command produces a single report which corresponds to a FORTRAN report. The exceptions are: the command which causes AED invocation analysis to produce two reports (Figs. 4.8b, 4.9b) and the command which presents a multiple module cross reference to incorporate two reports (Fig. 4.14b)

```
SYMBOLS                                SUBROUTINE SDBASA ( MODULE, ISTMT, IRETRN )

   NAME       SCOPE          TYPE        MODE        USE        OTHER INFORMATION...
-------------------------------------------------------------------------------------
   KXPAR      KDELMS       VARIABLE    INTEGER     USED
   KXPARM     RPTCOM       VARIABLE    INTEGER     USED
   KXTPLS     RPTCOM       VARIABLE    INTEGER     USED
   LIST       MTHSTO       ARRAY       INTEGER     SET/USED
   MARGS      MDB          VARIABLE    INTEGER     SET
   MBLOKS     MDB          VARIABLE    INTEGER     EQUIV
   MBRCHN     MTHTYP       VARIABLE    INTEGER     USED
   MCALL      MTHTYP       VARIABLE    INTEGER     USED
   MCIO       MTHTYP       VARIABLE    INTEGER     USED
   MCMMNS     MDB          VARIABLE    INTEGER     SET/USED
   MDUM26     (LOCAL)      VARIABLE    INTEGER     EQUIV
   MENTR      MTHTYP       VARIABLE    INTEGER     USED
   MENTRS     MDB          VARIABLE    INTEGER     SET/USED
   MENTR2     MTHTYP       VARIABLE    INTEGER     USED
   MEQLS      (LOCAL)      VARIABLE    INTEGER     SET/USED
   MEQUVS     MDB          VARIABLE    INTEGER     SET/USED
   MEXEC      MTHTYP       VARIABLE    INTEGER     USED
   MEXIT      MTHTYP       VARIABLE    INTEGER     USED
   MGOTO      MTHTYP       VARIABLE    INTEGER     USED
   MIF        MTHTYP       VARIABLE    INTEGER     USED
   MJUNCT     MTHTYP       VARIABLE    INTEGER     USED
   MMODE      MDB          VARIABLE    INTEGER     SET/USED
   MNAME      MDB          VARIABLE    INTEGER     EQUIV
   MNONX      MTHTYP       VARIABLE    INTEGER     USED
   MODULE     PARAMETER    VARIABLE    INTEGER
   MPRSET     MTHTYP       VARIABLE    INTEGER     USED
   MREAD      MTHTYP       VARIABLE    INTEGER     USED
   MREADS     MDB          VARIABLE    INTEGER     SET/USED
   MTYPE      MDB          VARIABLE    INTEGER     SET
   MWRITS     MDB          VARIABLE    INTEGER     SET/USED
   NONEXS     (LOCAL)      ARRAY       INTEGER     SET/USED
   NUMEXS     (LOCAL)      VARIABLE    INTEGER     SET/USED
   NUMNON     (LOCAL)      VARIABLE    INTEGER     SET/USED

        THE FOLLOWING LOCAL VARIABLES WERE DEFINED BUT NOT USED...
        TOKADD

        THE FOLLOWING NONLOCAL VARIABLES ARE SET...
        IRETRN  MTYPE   MMODE   MCMMNS  MENTRS  MARGS   MEQUVS  MREADS  MWRITS  ISTYPE  ISCODE
        ISINFO  LIST
```

This report is generated for each module analyzed during an AVFS run. The symbols are ordered alphabetically, and symbols which are only defined and never referenced are not included. Symbols which have the scope (LOCAL) are known only within the module being reported on. Symbols with the scope parameter are formal parameters for the module. All other scope classifications indicate the name of the common block the common variables are defined in. Each symbol is either of type variable or array, and of mode integer, real, logical, character, complex, or double precision. The use column provides a summary of how the symbol is used in the module. Local symbols which were defined but not referenced and all non-local variables (parameters and common variables) which are set within the module are noted at the end of the report.

Figure 4.6a. FORTRAN Symbols Report

```
SET/USE ANALYSIS AND PARAMETER REPORT   MODULE  VOTER

                                                    1ST  LAST  TOTL     ASSERTED    ACTUAL
NAME                    SCOPE      CLASS          STMT STMT  USES       USE         USE
-------------------------------------------------------------------------------------------
K3                      LOCAL      VARIABLE         74   74     1
K4                      LOCAL      VARIABLE         74   74     1
K5                      LOCAL      VARIABLE         74   74     1
K6                      LOCAL      VARIABLE         74   74     1
K7                      LOCAL      VARIABLE         74   74     1
K8                      LOCAL      VARIABLE         74   74     1
K9                      LOCAL      VARIABLE         74   74     1
LIMIT                   EXTERNAL   PROCEDURE        20   23     4
LX1                     LOCAL      VARIABLE         17   18     2
LX2                     LOCAL      VARIABLE         17   18     2
LX3                     LOCAL      VARIABLE         17   17     1
LX4                     LOCAL      VARIABLE         17   17     1
MONDAY                  EXTERNAL   PROCEDURE       104  104     1
MPS1                    -          -                47   65     4
MPS2                    -          -                51   68     4
MPS3                    -          -                55   69     5
NEWPROC                 EXTERNAL   PROCEDURE        85  117     3
NEWPROC2                -          -               119  119     1
    -------------------------------------------------------------------------------
    -                                  SET/USE ERROR                              -
    -   VARIABLE NEWPROC2              USED BEFORE BEING ASSIGNED A VALUE         -
    -------------------------------------------------------------------------------
NOCALLFROM              EXTERNAL   PROCEDURE        88   94     2
OWN.PTR                 -          -                15   62    11
PAR1                    PARAMETER  VARIABLE          5   26     4    INPUT       INPUT
PAR2                    PARAMETER  VARIABLE          5   28     5    OUTPUT      OUTPUT
PAR3                    PARAMETER  VARIABLE          5   28     3    NONE        OUTPUT
PROC1                   EXTERNAL   PROCEDURE       10   46     2
PROC2                   EXTERNAL   PROCEDURE       13   46     2
PROCCALL0               -          -                37   37     1
PROCCALL1               EXTERNAL   PROCEDURE       39  115    10
    -------------------------------------------------------------------------------
    -                                  PARAMETER ERROR                            -
    - PROCEDURE PROCCALL1              MULTIPLE USE OF SAME ACTUAL                -
    -                                  PARAMETER IN LINE  39                      -
    -------------------------------------------------------------------------------
    -------------------------------------------------------------------------------
    -                                  PARAMETER ERROR                            -
    - PROCEDURE PROCCALL1              CONSTANT USED AS ACTUAL                    -
    -                                  PARAMETER IN LINE  40                      -
    -------------------------------------------------------------------------------
    -------------------------------------------------------------------------------
    -                                  PARAMETER ERROR                            -
    -PROCEDURE PROCCALL1               FUNCTION/PROCEDURE USED AS ACTUAL          -
    -                                  PARAMETER IN LINE  41                      -
    -------------------------------------------------------------------------------
```

Figure 4.6b. AED Symbols Report

Figure 4.6(b) continued

This report is generated for each module analyzed by AVFS. The symbols are ordered alphabetically. Symbols which have the scope LOCAL are known only within the module being reported on. Other symbols with the EXTERNAL classification or COMMON classification are known outside the module. Each symbol is classified as to its class: variable, array, procedure, and to its type. The use column provides a summary of how the symbol is used in the module.

CROSS REFERENCE SUBROUTINE SDBASA (MODULE, ISTMT, IRETRN)

NAME	SCOPE	MODULE	USED/SET/EQUIVALENCED (* INDICATES SET)
ADDEPT	EXTERNAL	SDBASA	65 84 250
CALLED	(LOCAL)	SDBASA	24* 274 275*
ERROR	EXTERNAL	SDBASA	151
I	(LOCAL)	SDBASA	48* 49 50 51* 51 75* 76 77 78* 78 120* 121 145 146 161* 162 171* 172 172 173* 173 175 177 191* 227 228 342* 343 343 344* 344 346 353* 354 354 355*
IAGT	ANSI	SDBASA	149
IBAPAR	EXTERNAL	SDBASA	115 170 210
ICGT	ANSI	SDBASA	147
ICOMAS	(LOCAL)	SDBASA	119* 123* 123 127
IDUM	(LOCAL)	SDBASA	37 54 139 169
IDX	(LOCAL)	SDBASA	32* 33 33 33 34* 34 36 41 41 43 49 56 76 80* 80 82* 82 84
IEND	ANSI	SDBASA	307
IENT	ANSI	SDBASA	246 316
IEXECS	(LOCAL)	SDBASA	307* 308* 309* 310* 311* 312* 313* 314* 315* 316* 317* 318* 327* 328* 329* 330* 331* 332* 333* 334* 335* 336* 337* 354
IFT	ANSI	SDBASA	113 308
IF2	ANSI	SDBASA	128
IF3	ANSI	SDBASA	131
IGT	ANSI	SDBASA	136 311
IGTTOK	EXTERNAL	SDBASA	137
IMDB	(LOCAL)	SDBASA	10E
IPAR	(LOCAL)	SDBASA	115* 116 117 120 170* 172 175 180* 182 182 184
IREADC	FTNEXT	SDBASA	235
IRET	ANSI	SDBASA	188 315
IRETRN	PARAMETE	SDBASA	25* 269*
ISCLAS	EXTERNAL	SDBASA	138
ISCODE	SDB	SDBASA	29* 67* 89* 97* 112* 114* 118* 140* 155* 159* 167* 176*
ISDB	(LOCAL)	SDBASA	16E
ISEXEC	(LOCAL)	SDBASA	111 358* 360*
ISINFO	SDB	SDBASA	31* 129* 132* 139* 169* 199* 204* 210* 212 215 228* 249*
ISLABL	SDB	SDBASA	16E
ISLONG	SDB	SDBASA	33 36 37 41 59 70 73 115 116 121 139 142 192 206* 206 210 224 227
ISNONX	(LOCAL)	SDBASA	88 347* 349*
ISPTR	SDB	SDBASA	117* 145* 177* 184* 185 216* 234 240
ISTMT	PARAMETE	SDBASA	65 84 103 250
ISTOP	ANSI	SDBASA	188 190 318
ISTYPE	SDB	SDBASA	27 27 27 42* 59 64 74* 90 93 96 99 100* 166 188 188 190 198 203 208 208 208 208 208 208 252 255 255 255 343 354
ITEM	(LOCAL)	SDBASA	137* 138
IVMODE	EXTERNAL	SDBASA	54
IWRTEC	FTNEXT	SDBASA	241
IXABNL	FTNEXT	SDBASA	302
IXASS	ANSI	SDBASA	252 317

This report provides a symbol cross reference for each module analyzed during an AVFS run. All local symbols, external symbols, and common symbols referenced in the module are included. Symbol names are ordered alphabetically in the first column. The scope column indicates symbols known only within this module (LOCAL), external symbols, and symbols which are defined in common blocks included in the module (all others). Statements (AVFS statement numbers) which use a symbol are followed by a blank, statements which set a symbol are followed by a '*', and equivalence statements containing the symbol are followed by an 'E'.

Figure 4.7a. FORTRAN Cross Reference Report

CROSS REFERENCE MODULE VOTER

NAME D /SET (S) /DEFINED (D)	SCOPE	CLASS	TYPE	USED OR REFERENCE
K6	LOCAL	VARIABLE	INTEGER	74D
K7	LOCAL	VARIABLE	INTEGER	74D
K8	LOCAL	VARIABLE	INTEGER	74D
K9	LOCAL	VARIABLE	INTEGER	74D
LIMIT	EXTERNAL	PROCEDURE	REAL	20 21 22 23
LX1	LOCAL	VARIABLE	LABEL	17D 18D
LX2	LOCAL	VARIABLE	LABEL	17D 18D
LX3	LOCAL	VARIABLE	LABEL	17D
LX4	LOCAL	VARIABLE	LABEL	17D
MONDAY	EXTERNAL	PROCEDURE	-	104D
MPS1	-	-	-	47S 63 64
MPS2	-	-	-	51S 63 67
MPS3	-	-	-	55S 64 66
NEWPROC	EXTERNAL	PROCEDURE	-	85 97 117D
NEWPROC2	-	-	-	119
NOCALLFROM	EXTERNAL	PROCEDURE	-	88D 94
OWN.PTR	-	-	-	15S 20 20 35 62
PAR1	PARAMETER	VARIABLE	INTEGER	5 6D 24 26
PAR2	PARAMETER	VARIABLE	INTEGER	5 6D 25 27S 28
PAR3	PARAMETER	VARIABLE	INTEGER	5 6D 28S
PROC1	EXTERNAL	PROCEDURE	-	10D 46
PROC2	EXTERNAL	PROCEDURE	INTEGER	13D 46
PROCCALL0	-	-	-	37
PROCCALL1	EXTERNAL	PROCEDURE	-	39 40 41 115
PROCCALL2	EXTERNAL	PROCEDURE	-	46 110 113D
PROCCALL3	-	-	-	45
PTR	PARAMETER	VARIABLE	INPUT.POINTE	5 6D 15
R1	EXTERNAL	VARIABLE	INTEGER	11D 12
ROBERT	LOCAL	SWITCH	LABEL	18D
ROBERT1	EXTERNAL	PROCEDURE	-	82D 98
ROBERT2	EXTERNAL	PROCEDURE	-	100D
ROBERTG	-	-	-	38
SIGNAL	-	-	-	20 21 22
SIGNAL.MA	-	-	-	20S 48 49
SIGNAL.MB	-	-	-	21S 48 50
SIGNAL.OA	-	-	-	22S 52 54
SIGNAL.OB	-	-	-	23S 56 58
SYN1	LOCAL	SYNONYM	-	16D
SYN1A	LOCAL	SYNONYM	-	16D 36
SYN1B	LOCAL	SYNONYM	-	16D

Figure 4.7b. AED Cross Reference

Figure 4.7(b) continued

This report provides a symbol cross reference for each module analyzed by AVFS. All local symbols, external symbols, common symbols, and parameters referenced in the module are included. Symbol names are ordered alphabetically in the first column. The scope column indicates symbols known only in this module (LOCAL), external symbols (EXTERNAL), and common symbols (COMMON), and parameters (PARAMETER). Statements are identified by line number as to whether the symbol was defined, set, or used in a particular line.

```
INVOCATION SPACE                          SUBROUTINE CRDOUT(CARD,KEY)

-----------------------------------------

INVOCATIONS FROM WITHIN THIS MODULE
-----------------------------------------

MODULE A1IO
STMT =  117          CALL A1IO ( 2 , IOUT , LINEX , LLINEX )
STMT =  173          CALL A1IO ( 2 , IOUT , HEADX , 120 )
STMT =  174          CALL A1IO ( 2 , IOUT , SKIP , 10 )

MODULE BUFOUT
STMT =  132          CALL BUFOUT
STMT =  148          CALL BUFOUT
STMT =  153          CALL BUFOUT

MODULE CVD
STMT =   50          CALL CVD ( NCARD , NCARD5 )
STMT =   67          CALL CVD ( NEST , NEST5 )
STMT =  172          CALL CVD ( NPAGE , HEADX ( 111 ) )

MODULE IABS
STMT =   48          IABS ( INDENT )

MODULE ICMTST
STMT =   57          ICMTST ( CARD ( 2 ) )
STMT =   88          ICMTST ( CARD ( I ) )
STMT =  121          ICMTST ( CARD ( 2 ) )

INVOCATIONS TO THIS MODULE FROM WITHIN LIBRARY
-----------------------------------------

MODULE CMTOPT
STMT =  112          CALL CRDOUT ( CARD , ICOM )

MODULE IFTRA
STMT =   69          CALL CRDOUT ( CARD , 4 )
STMT =   80          CALL CRDOUT ( CARD , 4 )
STMT =   88          CALL CRDOUT ( CARD , 4 )
STMT =   97          CALL CRDOUT ( CARD , 1 )
STMT =  108          CALL CRDOUT ( CARD , 1 )
STMT =  176          CALL CRDOUT ( CARD , 1 )
STMT =  181          CALL CRDOUT ( 0 , - 101 )
STMT =  188          CALL CRDOUT ( CARD , 1 )
STMT =  233          CALL CRDOUT ( CARD , 1 )
STMT =  238          CALL CRDOUT ( CARD , 0 )
STMT =  248          CALL CRDOUT ( CARD , 0 )
```

This module report shows all invocations, along with the AVFS statement numbers, to and from the specified module. It is useful in examining actual parameter usage. The text printed is reformatted and may contain more or fewer blanks than the original text line. A summary of this report is provided by the externals cross reference report. This report is produced for modules analyzed during an AVFS run.

Figure 4.8a. FORTRAN Invocation Space Report

```
INVOCATIONS TO THIS MODULE

PROCEDURE NEWPROC
  STMT   97          XPROC

  STMT   85          ROBERT1

PROCEDURE PROCCALL2
  STMT  110          PROCCALL1

  STMT   46          VOTER

PROCEDURE PROCCALL1
  STMT  115          PROCCALL2

  STMT  111          PROCCALL1

  STMT  106          MONDAY

  STMT   44          VOTER

  STMT   43          VOTER

INVOCATIONS REPORT                                  MODULE VOTER
                                              PAGE    1

INVOCATIONS FROM WITHIN THIS MODULE

PROCEDURE NEWPROC
  STMT  119          NEWPROC2;

PROCEDURE PROCCALL2
  STMT  115          PROCCALL1;

PROCEDURE PROCCALL1
  STMT  111          PROCCALL1;

  STMT  110          PROCCALL2;

PROCEDURE MONDAY
  STMT  106          PROCCALL1;

PROCEDURE ROBERT2
                     -CONTAINS NO PROCEDURE CALLS
```

This module report shows all invocations, along with the AVFS statement numbers, to and from the specified module. It is useful in examining actual parameter usage.

Figure 4.8b. AED Invocation Space Report

```
INVOCATION SUMMARY

ENTRY          LISTS OF CALLS
-------------------------------------------------------------------------------

PUTLST   WHICH IS DEFINED IN  GETBLK
         IS CALLED BY - -NONE-
         AND CALLS    - GETFRG  MAKFRG  XMIT

PUTWRD   WHICH IS DEFINED IN  GETBLK
         IS CALLED BY - PUTBEF  PUTBOT
         AND CALLS    - GETFRG  MAKFRG  XMIT

XMIT     WHICH IS UNDEFINED
         IS CALLED BY - GETBLK  NEXT    PREV    PUTAT   PUTBEF  PUTBOT

THE FOLLOWING ENTRIES ARE NOT CALLED
                        GETBLK  GETLST  GETWRD  ISRTAB  NEXT    PREV    PUTAT
```

This report shows the dependencies of the modules on the library by listing all modules which call an entry point and all calls from that entry point. If an entry is defined as an entry point within a module, the name of the module is indicated. This report includes all modules and entrys on the restart file. An updated version of the report may be obtained by reanalyzing all changed modules and using the EXPAND option. The actual statements where invocations to a given entry point occur can be found in the externals cross reference report.

Figure 4.9a. FORTRAN Invocation Summary Report

```
MODULE DEPENDENCE REPORT                    MODULE VOTER
                                      PAGE  1

PROCEDURE          DEPENDENCY
-------------------------------------------------------------------------

NEWPROC            IS INVOKED BY  ROBERT1              XPROC

                   AND INVOKES    NEWPROC2

PROCCALL2          IS INVOKED BY  PROCCALL1            VOTER

                   AND INVOKES    PROCCALL1

PROCCALL1          IS INVOKED BY  MONDAY               PROCCALL1
     PROCCALL2         VOTER
                   AND INVOKES    PROCCALL1            PROCCALL2

MONDAY             IS INVOKED BY  -NONE

                   AND INVOKES    PROCCALL1

ROBERT2            IS INVOKED BY  -NONE

                   AND INVOKES    -NONE

XPROC              IS INVOKED BY  ROBERT1

                   AND INVOKES    NEWPROC              NOCALLFROM
     ROBERT1           XPROC1
                                  XPROC2

NOCALLFROM         IS INVOKED BY  XPROC

                   AND INVOKES    -NONE

ROBERT1            IS INVOKED BY  XPROC

                   AND INVOKES    NEWPROC              XPROC

VOTER              IS INVOKED BY  -NONE

                   AND INVOKES    PROCCALL0            PROCCALL1
     PROCCALL2         PROCCALL3
                                  ROBERTG
```

This report shows the dependencies of the modules on the interface library. It lists all modules which invoke a module and all invocations in a module. The actual statements where invocations to a given module occur can be found in the invocation space report or the global cross reference report.

Figure 4.9b. AED Invocation Summary Report

```
INVOCATION BANDS n                SUBROUTINE BLDSDB ( MODULE, ISTMT, IACT, IASERT )
TO LEVEL 2

LEVEL      -5        -4        -3        -2        -1         0         1         2         3         4         5
                                                                BLDSDB
                                                    PASTWO              ISASGN
                                                                        SDBASA
                                                                                  ADDEPT
                                                                                  ERROR
                                                                                  IBAPAR
                                                                                  IGTTOK
                                                                                  ISCLAS
                                                                                  IVMODE
                                                                                  JGET
                                                                                  NMBARG
                                                                                  RSTMPL
                                                                                  XMIT
                                                                        SDBAST
                                                                        SDBIFT
                                                                        SDBRUN
                                                                        SDBSTR
                                                                        TOKADD
```

This report shows the selected module within the invocation hierarchy. At the center is the specified module. Each successive band of modules from the center to the left shows the calling modules; each successive band to the right shows the called entry points. The left (calling) modules reside on the library; the right (called) modules can include modules external to the AVFS library. A summary of this report is provided by the invocation summary report. More detailed information about modules and statement numbers containing invocations can be found in the externals cross reference. This report is produced for modules analyzed during an AVFS run. The default level of two may be changed by using the command: REPORT = B/n where n may be any integer from 1 to 5.

Figure 4.10a. FORTRAN Invocations Band

```
CALLING TREE                                          MODULE VOTER
                                                PAGE    1

LEVEL        -9   -8   -7   -6   -5   -4   -3   -2   -1        0         +1   +2
-------------------------------------------------------------------------------

                                                                NEWPROC

                                                        ROBERT1             NEWPRO
                                                                              (C2)
                                                XPROC

                                            ROBERT1

                                        XPROC

                                    ROBERT1

                                XPROC

                            ROBERT1

                        XPROC

                    ROBERT1

                XPROC

            ROBERT1

                                                                XPROC
```

This report shows the selected module in a calling tree. At the center is the specified module. The left hand modules are the calling modules. The right hand modules are the called modules. A summary of this report is found in the invocation summary report. The statement numbers containing invocations can be found in the global cross reference.

Figure 4.10b. AED Invocations Band

COMMON MATRICES

LEGEND (C=FIRST USED IN A CALL.E=EQUIVALENCED,S=SET,U=USED,X=SET AND USED)

```
--------------------------------.------
         **           *         .     *
         * * MODULE   * N P P P P.G I *
         *  *         * E R U U U.E S *
         *   *        * X E T T T.T R *
         *    *       * T V A B B.B T *
         *     *      *     T E O.L A *
         *      *     *       F T.K B *
         *       *    *         .     *
 COMMON  * SYMBOL *   *         .     *
         *           **         .     *
--------------------------------.------
AISTO    * FLCXXX     * U U U U U.U U *
         * FNUXXX     * U U U U U.U U *
         * FRGXXX     * U U U U U.U U *
         * FSZXXX     *          .U C *
         * ICHXXX     *     S S S.S   *
         * IXXXXX     * X X X X X.X X *
         * LNGXXX     * U U U U U.U U *
         * MAXXXX     *          .X   *
-------------------------------- ------
```

```
------------------------------------
         **           *         *
         * * MODULE   * B P P S *
         *  *         * L A T D *
         *   *        * D S R B *
         *    *       * S T S A *
         *     *      * D W T S *
         *      *     * B O R A *
         *       *    *         *
 COMMON  * SYMBOL *   *         *
         *           **         *
------------------------------------
SDB      * ISCODE     * X X U S *
         * ISCONT     * U X     *
         * ISINDT     *     S   *
         * ISINFO     * S S   X *
         * ISLABL     * E E E E *
         * ISLONG     * X X   X *
         * ISPTR      *   X S X *
         * ISTYPE     * S X U X *
------------------------------------
```

The common matrices report lists symbols which are used by at least one of the modules on the restart file. The symbol usage is explained in the legend at the top of the report; a blank space indicates that the symbol is not used in any way in that particular module. The symbols within each common are listed alphabetically in this report. Only modules which use at least one variable of a common block will occur in the matrix for that common. This report includes all commons and modules on the restart file. An updated version of this report may be produced by reanalyzing all changed modules and using the EXPAND option. When all modules in a software system have been entered onto a RESTART file, this report can be used to check for global set/use inconsistencies. A row of one or more U's indicates that a common variable is used but not set. A row of one or more S's indicates a common variable which is set but not used. A common variable which is not included in the matrix is never referenced in an executable statement. The statement numbers where common variables are referenced can be found in the common variable cross reference report.

Figure 4.11. FORTRAN Commons Matrices

```
THE FOLLOWING MODULES CONTAIN I/O STATEMENTS

            A1IO
            IFTRAX

-------------------------------------------------------------------------------

I/O STATEMENTS AND ASSOCIATED FORMATS

-- A1IO   --

 STMT NEST  LINE  SOURCE...                                          ...SOURCE TAB
-------------------------------------------------------------------------------
    5        16       READ(UNIT,901                                  A1IO  17
    5        17      1 ) (TEXT(I),I=1,LEN)                           A1IO  18
             18  C        END-FILE TEST                              A1IO  19
   12        29       WRITE(UNIT,901) (TEXT(I),I=1,LEN)              A1IO  30
             30  C                                                   A1IO  31
   23        48  901  FORMAT(132A1)                                  A1IO  49

-- IFTRAX  --

 STMT NEST  LINE  SOURCE...                                          ...SOURCE TAB
-------------------------------------------------------------------------------
   17        25       READ ( IN, 900 ) KNDENT, KCONV, KOCOM, KFORIN, KIN, KOUT, KOUTC  IFTRAX26
   17        26      *                 KENCRD, KNLINE, KSAVE         IFTRAX27
   18        27    900 FORMAT ( I2,3I1,6I3 )                         IFTRAX28
   44        51       WRITE(IOUT,901) NCARD, NERR                    IFTRAX52
   45        52    901 FORMAT ( / 25H STRUCTRAN-1 STATISTICS         IFTRAX53
   45        53       1 / 1X,I6,20H CARDS READ                       IFTRAX54
   45        54       2 / 1X,I6,20H ERROR(S) FOUND                   IFTRAX55
   45        55      * )                                             IFTRAX56

-------------------------------------------------------------------------------
```

This report provides a list of all the program modules in which any type of READ or WRITE statement appears. The source statements are reproduced along with the defining FORMAT. This report may be used to locate all the points where variables are being input or output from the system. This report is produced only for source text which is analyzed during an AVFS run.

Figure 4.12. FORTRAN I/O Statements

CROSS REFERENCE

NAME	SCOPE	MODULE	USED/SET/EQUIVALENCED (* INDICATES SET)
AIDBG	DBGCOM	ISRTAB	82
CORE		GETBLK	26E
		ISRTAB	25E
		NEXT	18E
		PREV	18E
		PUTAT	18E
		PUTBEF	18E
		PUTBOT	18E
FLCXXX	AISTO	GETBLK	146 190 231 266
	AISTO	ISRTAB	67
	AISTO	NEXT	39
	AISTO	PREV	39
	AISTO	PUTAT	33
	AISTO	PUTBEF	43
	AISTO	PUTBOT	43
FNUXXX	AISTO	GETBLK	142 187 227 261
	AISTO	ISRTAB	62
	AISTO	NEXT	36
	AISTO	PREV	36
	AISTO	PUTAT	30
	AISTO	PUTBEF	40
	AISTO	PUTBOT	40
FRGDIR	POOLCM	GETBLK	149* 193* 240* 268*
	POOLCM	ISRTAB	79*
	POOLCM	NEXT	43*
	POOLCM	PREV	43*
	POOLCM	PUTAT	38*
	POOLCM	PUTBEF	50*
	POOLCM	PUTBOT	50*
FRGXXX	AISTO	GETBLK	148 192 240 268
	AISTO	ISRTAB	79
	AISTO	NEXT	43
	AISTO	PREV	43
	AISTO	PUTAT	38
	AISTO	PUTBEF	50
	AISTO	PUTBOT	50
FSZXXX	AISTO	GETBLK	140 141 185 186 216 219 220 248 249
	AISTO	ISRTAB	66
ICHXXX	AISTO	GETBLK	56* 81* 118* 168* 235*
	AISTO	PUTAT	35*
	AISTO	PUTBEF	47*
	AISTO	PUTBOT	47*
IXXXXX	AISTO	GETBLK	147* 147 149 191* 191 193 239* 239 240 267* 267 268
	AISTO	ISRTAB	78* 78 79
	AISTO	NEXT	42* 42 43
	AISTO	PREV	42* 42 43
	AISTO	PUTAT	37* 37 38

This multi-module report shows where variables in common blocks are used, set or equivalenced. The report is alphabetically ordered by the name of the common variable. The common block which contains the variable is indicated in the scope column. Modules which reference the common variable are alphabetically ordered in the module column. Statements (AVFS statement numbers) within each module are shown next. A blank following the statement number indicates the variable is used there, a '*' indicates the variable is set, and an 'E' indicates the variable is equivalenced. This report is produced for all modules and all commons on the restart file. An updated version may be obtained by reanalyzing all changed modules and using the EXPAND option. A summary of the information in this report is provided in the common matrices report.

Figure 4.13. FORTRAN Commons Cross Reference

CROSS REFERENCE

NAME	SCOPE	MODULE	USED/SET/EQUIVALENCED	(* INDICATES SET)		
EROR	EXTERNAL	ISRTAB	87			
FRELNK	EXTERNAL	PUTBEF	31			
	EXTERNAL	PUTBOT	29	31		
GETFRG	EXTERNAL	GETBLK	262			
	EXTERNAL	ISRTAB	63			
	EXTERNAL	NEXT	37			
	EXTERNAL	PREV	37			
	EXTERNAL	PUTAT	31			
	EXTERNAL	PUTBEF	41			
	EXTERNAL	PUTBOT	41			
IGTWRD	EXTERNAL	NEXT	31	33		
	EXTERNAL	PREV	31	33		
	EXTERNAL	PUTBEF	32			
	EXTERNAL	PUTBOT	35			
ITSFRG	EXTERNAL	ISRTAB	60			
	EXTERNAL	NEXT	35			
	EXTERNAL	PREV	35			
	EXTERNAL	PUTAT	29			
	EXTERNAL	PUTBEF	39			
	EXTERNAL	PUTBOT	39			
LGTMLT	EXTERNAL	ISRTAB	52	53		
MAKFRG	EXTERNAL	GETBLK	224			
MINO	EXTERNAL	ISRTAB	66			
PUTWRD	EXTERNAL	PUTBEF	33	35	37	
	EXTERNAL	PUTBOT	32	36	38	
XMIT	EXTERNAL	GETBLK	45	55	234	237
	EXTERNAL	NEXT	40	46		
	EXTERNAL	PREV	40	46		
	EXTERNAL	PUTAT	34			
	EXTERNAL	PUTBEF	44			
	EXTERNAL	PUTBOT	44			

This multi-module report shows the AVFS statement number where each external is referenced. The report is alphabetically ordered by the name of the external. Modules which reference the external are alphabetically ordered in the module column. Statements (AVFS statement numbers) within each module are shown in the next column. This report is produced for all modules on the restart file. An updated version may be obtained be reanalyzing all changed modules and using the EXPAND option. A summary of information contained in this report is provided by the Invocation Summary Report. The text of each invocation can be found by referring the AVFS statement listing or Invocation Report for each module. Note that these reports are not generated from the restart file but rather from source analyzed during an AVFS run.

Figure 4.14a. FORTRAN Externals Cross Reference

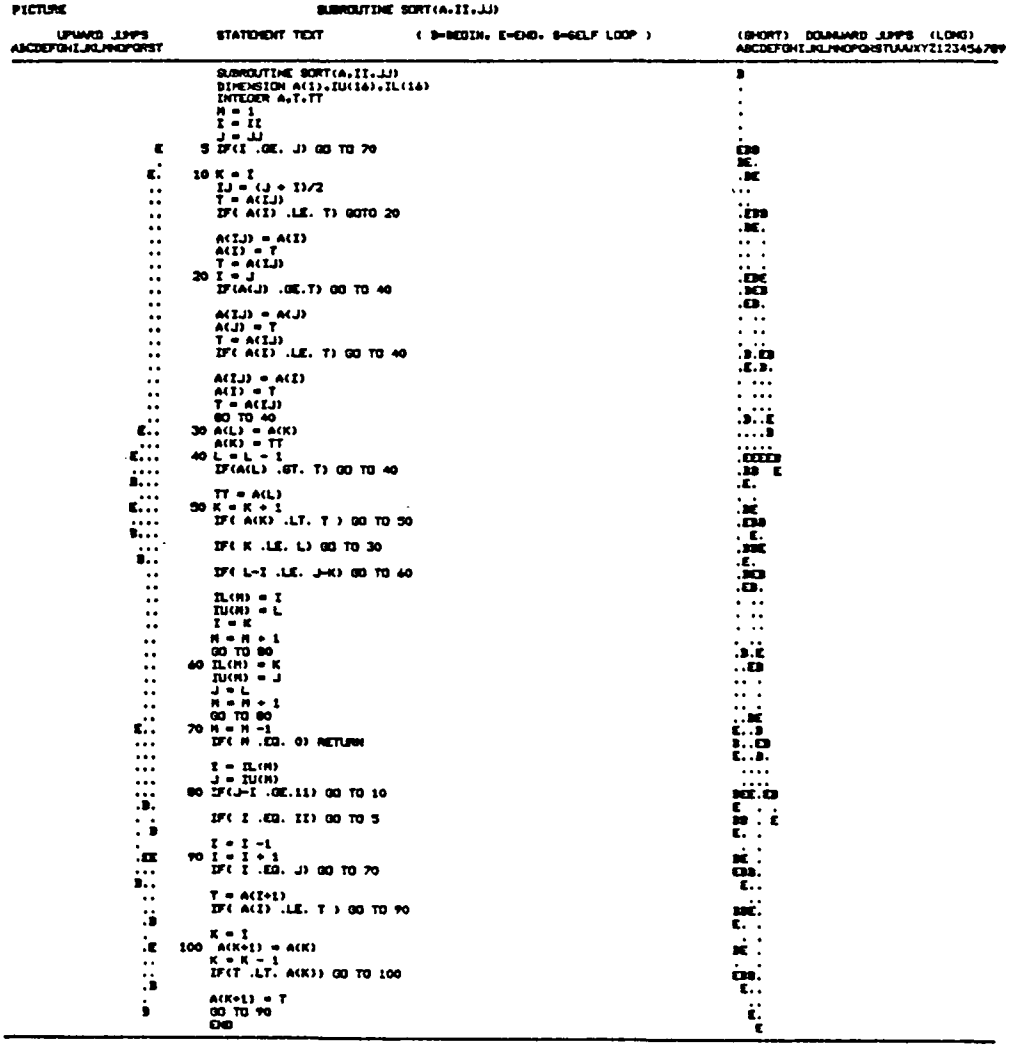
CROSS REFERENCE MULTI-MODULE REPORT PAGE 1

NAME	CLASS	MODULE	USED/SET (- INDICATES SET)
C1	COMMON	VOTER	71
C2	COMMON	VOTER	71
C3	COMMON	VOTER	71
C4	COMMON	VOTER	76
CC1	COMMON	VOTER	9
CC2	COMMON	VOTER	9
E1	EXTERNAL	VOTER	72
E2	EXTERNAL	VOTER	72
E3	EXTERNAL	VOTER	72
E4	EXTERNAL	VOTER	73
E5	COMMON	VOTER	78
E6	COMMON	VOTER	78
EE1	EXTERNAL	VOTER	14
EE2	EXTERNAL	VOTER	14
NEWPROC	EXTERNAL	ROBERT1	85 97 117
NOCALLFROM	EXTERNAL	NOCALLFROM	88 94
PROC1	EXTERNAL	VOTER	10 46
PROC2	EXTERNAL	VOTER	13 46
PROCCALL1	EXTERNAL	VOTER	39 40 41 42 43 44 106 108 111 115
PROCCALL2	EXTERNAL	VOTER	46 110 113
R1	EXTERNAL	VOTER	11 12
ROBERT1	EXTERNAL	ROBERT1	82 98
ROBERT2	EXTERNAL	ROBERT2	100
VOTER	EXTERNAL	VOTER	5 41 43 44 45 45 -61
XPROC	EXTERNAL	ROBERT1	86 92

Figure 4.14b. AED Global Cross Reference

Figure 4.14b continued

This multi module report shows the AVFS statement number where each global variable is referenced. The report is alphabetically ordered by the name of the global variable in the first column. The class column denotes whether the variable is EXTERNAL or in COMMON. The moudles which reference the global variable are alphabetically ordered in the module column. The remaining columns contains the line numbers within each module where the variable is referenced.



The PICTURE report can only be obtained by using the REPORT=PICTURE command; it is not included in any of the options because the PICTURE report has limited use for structured source programs. The primary function of this report is to delineate the control flow of FORTRAN programs. The downward flows are shown on the right of the report. The upward flows are shown on the left. The B stands for the start of a path and the E stands for the end of a path. This report is especially helpful in breaking down large FORTRAN programs into smaller parts that are more manageable to restructure. Since the PICTURE report shows the beginning and ending of paths, it helps the user determine which are logically cohesive sections of code.

Figure 4.15. FORTRAN Picture of Module Structure

```
FLOWGRAPH          PROCEDURE YAW.ENG

   UPWARD JUMPS          STATEMENT TEXT                  DOWNWARD JUMPS

ABCDEFGHIJK                                                ABCDEFGHIJK

                  DEFINE PROCEDURE YAW.ENG TO BE            B
                  BEGIN                                     .
                  Y.FAIL.MON=PGM.V AND NOT YAW.SAS.2FAIL;   .
                  IF Y.TEST. COMPL                          EB
                                                            B
                  THEN BEGIN                                ..
                        IF IN.CH.A                          .EB
                                                            B.
                        THEN BEGIN                          ...
                           ASNBIT(Y.FAIL.MON,8,DO.BUFF.3);  ...
                                END                         ..E
                        ELSE                                .E
                           ASNBIT(Y.FAIL.MON,5,DO.BUFF.3);  .
                        END;                                .
                  GSP.V = REFBIT(13,CD.15.MA);              E
```

Figure 4.16. AED Picture of Module Structure

4.4 SUMMARY

The SUMMARY option is intended to be used when a brief introduction to a module or a set of modules is desired. For FORTRAN it provides an analysis of source statements, common blocks, and entry points. For AED it provides an analysis of source statements on a module basis.

The statements of individual modules are classified according to categories which are appropriate to that language. Under each classification a tabulated account of the various statement types are listed. An individual Statement Profile report with this information is generated for each module. This report is shown in Fig. 4.17.

For FORTRAN, the COMMONS report lists the common blocks in each module and a report index for multiple module analysis as was shown in Fig. 2.4. For AED, there is a single COMMON which is analyzed during interface analysis and reported on in the interface report (see Fig. 4.5).

An Invocation Summary Report is also generated for this option and for the document option as shown in Fig. 4.9b.

PROFILE

The AED statement profile report (Fig. 4.17b) lists the name of the module and the number of lines at the top of the report. The module name is the name of the procedure contained in the module. The number of lines includes the lines from any inserts and includes blank lines.

AED statements are classified into declarations and statements. While there can be more than one declaration or statement per line. Typically there is less than one of each per line. Hence the number of declarations plus the number of statements will be less than the number of lines in most cases.

Under declarations, there are declarations for arrays, beads, commons, components, defines, externals, packs, presets, procedures, switches, synonyms, and variables. Each of these is counted separately. Inserts are listed under declarations but are not included in the count for declarations. In AED, the inserts normally contain declarations. The line percentage is computed on the basis of the total number of lines printed on the top of the profile.

Under statements, there are assignment, compond, if, for, goto, procedure and while categories. A line can contain several categories. For example, a compound statement is made up a BEGIN..END construct. Thus the module is counted as a compound statement. Then an IF statement often contains a BEGIN...END compound statement as well as a procedure invocation.

Comments and asserts are listed under statements but are counted separately.

Report	FORTRAN Command	AED Command
Statement Profile	OPTION=SUMMARY (Fig. 4.17a)	GRC*PROFILE.PROFILE (Fig. 4.17b) Appendix C-20
Invocation Summary	OPTION=SUMMARY (Fig. 4.9a)	GRC*DEPEND.DEPEND (Fig. 4.9b) Appendix C-17
Common Summary	OPTION=SUMMARY (Fig. 4.18)	GRC*INTER*INTER (Fig. 4.5) Appendix C-14

The FORTRAN command

```
OPTION=SUMMARY
```

may be replaced by the UNIVAC command

```
@XQT,B GRC*AVFS.AVFS
```

The complete UNIVAC JCL is shown on page C-10.

to produce the three reports at one time. The AED commands to produce the three corresponding reports are separate XQT commands.

STATEMENT PROFILE SUBROUTINE EXAMPL (INFO.LENGTH)

INTERFACE CHARACTERISTICS

ARGUMENTS	2
ENTRY	1
EXIT	1
WRITE	1

STATEMENT CLASSIFICATION	STATEMENT TYPE	NUMBER	PERCENT
DECLARATION...			
	FORMAT	1	2.8
	TOTAL	1	2.8
EXECUTABLE...			
	ASSIGNMENT	4	11.1
	BLOCK	2	5.6
	CALL	1	2.8
	CASEELSE	1	2.8
	DOUNTIL	1	2.8
	ELSE	1	2.8
	END	1	2.8
	ENDBLOCK	2	5.6
	ENDCASE	1	2.8
	ENDIF	2	5.6
	ENDWHILE	2	5.6
	INVOKE	3	8.3
	RETURN	1	2.8
	SUBROUTINE	1	2.8
	WRITE	1	2.8
	TOTAL	24	66.7
DECISION...			
	CASEOF	1	2.8
	CASE	2	5.6
	DOWHILE	2	5.6
	ENDUNTIL	1	2.8
	IF-THEN	2	5.6
	TOTAL	8	22.2
DOCUMENTATION...			
	COMMENT	3	8.3
	TOTAL	3	8.3

This report classifies each statement of a module as either a declaration, executable, decision, or documentation statement. Under these classifications, a tabulation of the statement types are listed.

Figure 4.17a. FORTRAN Statement Profile

STATEMENT PROFILE
module YAW.ENG
number of lines 156

	number	percentage of lines
declarations	83	53.2
array	0	0.0
bead	1	0.6
common	0	0.0
component	1	0.6
define	1	0.6
external	25	16.0
insert	3	1.9
pack	0	0.0
preset	0	0.0
procedure	1	0.6
switch	0	0.0
synonym	2	1.3
variable	52	33.3
statements	45	28.8
assignment	24	15.4
comment	19	12.2
compound	5	3.2
if	6	3.8
for	0	0.0
goto	0	0.0
procedure	11	7.1
while	1	0.6
assert	5	3.2

AED statements are classified into declarations and statements. Under each classification, the type of statement is listed along with its number and frequency in terms of liens. The number of lines (including blank ilnes) in the module is listed.

Figure 4.17b. AED Statement Profile

```
COMMON SUMMARY

COMMON      MODULES WHICH INCLUDE THE COMMON
--------------------------------------------------------------------------------

AISTO       MAKTAB

ALPHA       CMATRX

ANSI        CMATRX REFVAR

BLKSTO      DEPVOK

OBGCOM      MAKTAB

DEPCOM      DEPBND DEPVOK

EPT         DEPGRP DEPVOK REFVAR

FILES       CMATRX DEPBND DEPGRP DEPVOK XREFER

GLOBAL      DEPBND

HALPHA      XREFER

HCHARS      DEPGRP XREFER

HOIGIT      CMATRX

ICMMOS      STEP15

KDELMS      DEPVOK

MACHNE      DEPVOK

MOB         DEPVOK REFVAR

MMRY15      STEP15

MTHSTO      DEPGRP DEPVOK

MTHST1      DEPVOK XREFER
```

This report lists all modules and all common blocks encountered.

Figure 4.18. FORTRAN Common Summary

4.5 INSTRUMENT

Figure 4.19 illustrates AVFS instrumentation of a FORTRAN or AED program to prepare it for an execution coverage test. There are three forms of instrumentation:

- path instrumentation
- trace instrumentation
- assertion instrumentation

In each case the instrumented modules will be written to a sequential file which must be compiled, loaded, and executed in the normal test environment. During execution, data is collected in a file (FORTRAN) or in memory (AED) for later analysis. Section 4.5.1 describes FORTRAN instrumentation and Sec. 4.5.2 describes AED instrumentation.

4.5.1 FORTRAN Instrumentation

Path Instrumentation

During FORTRAN instrumentation, the instrumented modules will be written to UNIT 9 (LPUNCH). A DD path definitions report will be generated to aid in the interpretation of test results. The user should use the FIRSTLINE command (see Section 3.3) to specify the name of a FORTRAN compiler in a UNIVAC run stream command, and AVFS will automatically insert this specification as the first line of every subprogram. For example, if the FTN compiler will be used, the command should be:

```
FIRSTLINE = (@FTN,I TPF$.+).
```

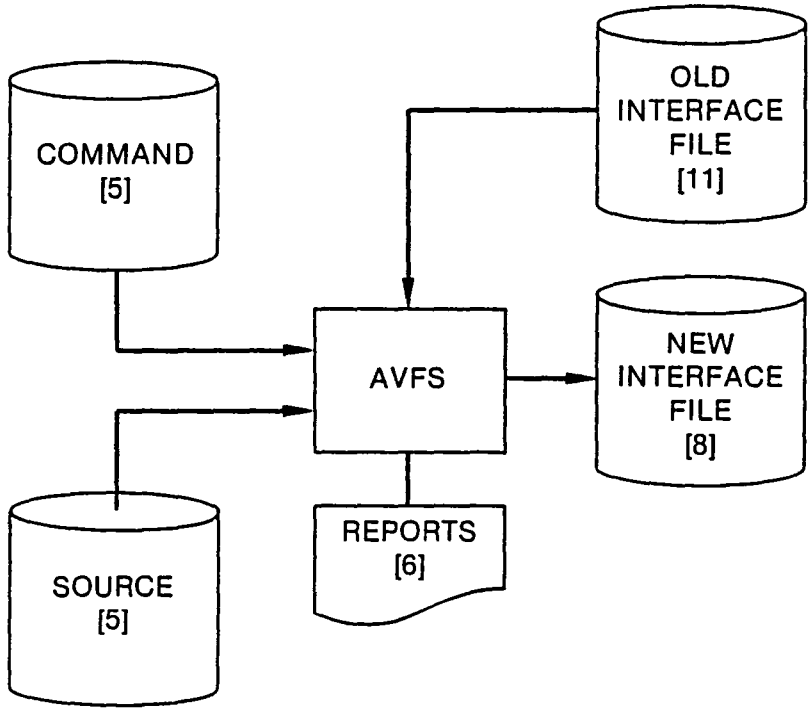


Figure 4.19. AVFS Instrumentation of Source Code

A DD-path is a sequence of executable statements emanating from a decision statement and continuing to the next decision statement. Since complete DD-path testing means exercising all possible outways of decision statements, this is a more rigorous testing measure than exercising all program statements. All AVFS execution coverage reports are presented in terms of DD-paths, not statements.

INSTRUMENT inserts a set of probe statements into each module. The probe statements are inserted into the source text at each entry and each exit of the modules and at each statement which begins a DD-path. Each probe includes a call to a data collection routine which records information concerning the flow of control in the executing module(s). A special probe is inserted at the end of the main program to signal the end of test execution. The user can also have this special probe inserted at other points in his code, which has the effect of breaking one test execution into multiple test cases.

The instrumented source text along with an automatically supplied data collection routine is written to UNIT 9 (LPUNCH) in FORTRAN. The file is then compiled by a FORTRAN compiler. The instrumented object code is then ready for loading and test execution (Fig. 4.20).

During execution of the instrumented program, the probes record execution data, which results from processing the set of test cases for this run, on UNIT 12 (LTEST). If UNIT 12 is already assigned to a user file, the command,

INSTRUMENT,PUNCH,PROBE,(<file number>).

will cause the data collected during execution to be written to the unit number specified.

There is a special instrumentation command which allows the user to insert special probes into his instrumented code which delineate test cases within the test execution. The user specifies a statement within

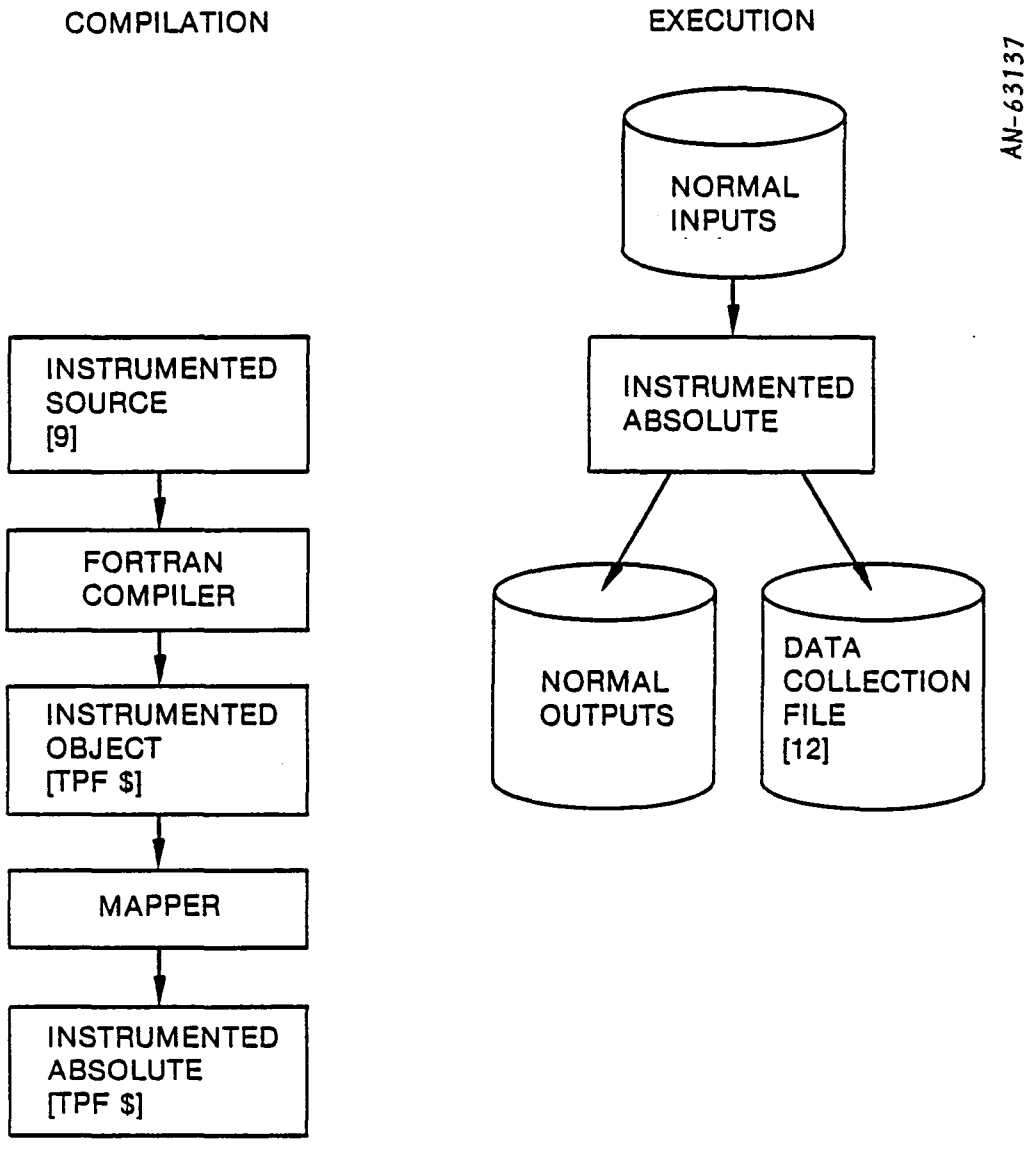


Figure 4.20. Loading and Test Execution

a given module. Before each execution of this statement, the last test case is terminated and a new test case is begun. The form of the command for identifying a test execution boundary is:

TESTBOUND,MODULE = (<name>),STATEMENT = <number>

where <number> is the AVFS statement number in module <name> where the test-case probe is to be put. The probe is inserted before the number specified; therefore, the number should be that of the first statement not to be included in the test case. Up to ten TESTBOUNDS may be specified during any one instrumented run. All must immediately follow the OPTIONS command (preceding all REACHING SET commands). The output of this option when the LIST option is also specified is a DD-path Definitions report, as shown in Fig. 4.21. It is an indented source listing of an individual module with additional DD-path information. At each decision point, the DD-path generated is described in terms of its decision outways. When measuring testing coverage, the user can refer to this report to associate the DD-path definitions with his original source text.

Commands

INSTRUMENT,PUNCH,PROBE,(<file number>).	(optional)
FIRSTLINE = (<run stream command>).	(optional)
OPTION = INSTRUMENT,LIST. @XQT,IL GRC.AVFS.AVFS	
TESTBOUND,MODULE=(<name>),STATEMENT=<number>.	(optional)

See Appendix C-4 for a complete JCL example.

Report

DD-path Definitions (Fig. 4.21)

```
DD-PATH DEFINITIONS                    SUBROUTINE EXAMPL (INFO,LENGTH)

STMT NEST  LINE  SOURCE...                                                    ...SOURCE TAB
  1          1     SUBROUTINE EXAMPL (INFO,LENGTH)
             2 C
             3 C   ILLUSTRATION OF DMATRAN SYNTAX                                  EXAMPL3
             4 C                                                                   EXAMPL4
                                                  ** DDPATH   1 IS PROCEDURE ENTRY
  2          5     IF (INFO.LE.10 .AND. LENGTH.GT.0)THEN                          EXAMPL5
                                                  ** DDPATH   2 IS TRUE BRANCH
                                                  ** DDPATH   3 IS FALSE BRANCH
  3    1     6     .   CALL CALLER ( INFO )                                       EXAMPL6
  4          7     ELSE                                                           EXAMPL7
  5    1     8     .   LENGTH=50                                                  EXAMPL8
  6          9     END IF                                                         EXAMPL9
  7         10     CASE OF (INFO+6)                                               EXAMPL10
                                                  ** DDPATH   4 IS BRANCH OUTWAY  1
                                                  ** DDPATH   5 IS BRANCH OUTWAY  2
                                                  ** DDPATH   6 IS BRANCH OUTWAY  3
  8         11     CASE (14)                                                      EXAMPL11
                                                  ** DDPATH   7 IS TRUE BRANCH
                                                  ** DDPATH   8 IS FALSE BRANCH
  9    1    12     .   LENGTH=LENGTH-INFO                                         EXAMPL12
 10         13     CASE (17)                                                      EXAMPL13
                                                  ** DDPATH   9 IS TRUE BRANCH
                                                  ** DDPATH  10 IS FALSE BRANCH
 11    1    14     .   DO WHILE (INFO.LT.20)                                     EXAMPL14
                                                  ** DDPATH  11 IS LOOP AGAIN
                                                  ** DDPATH  12 IS LOOP ESCAPE
 12    2    15     .   .   DO UNTIL (LENGTH.LE.INFO)                            EXAMPL15
 13    3    16     .   .   .   INVOKE (COMPUTE LENGTH)                          EXAMPL16
 14    3    17     .   .   .   IF (LENGTH.GE.30) THEN                           EXAMPL17
                                                  ** DDPATH  13 IS TRUE BRANCH
                                                  ** DDPATH  14 IS FALSE BRANCH
 15    4    18     .   .   .   .   INVOKE (PRINT-RESULTS)                       EXAMPL18
 16    3    19     .   .   .   END IF                                            EXAMPL19
 17    2    20     .   .   END UNTIL                                             EXAMPL20
                                                  ** DDPATH  15 IS LOOP ESCAPE
                                                  ** DDPATH  16 IS LOOP AGAIN
 18    2    21     .   .   INFO=INFO+1                                           EXAMPL21
 19    1    22     .   END WHILE                                                 EXAMPL22
 20         23     CASE ELSE                                                      EXAMPL23
 21    1    24     .   DO WHILE (LENGTH.GT.0)                                   EXAMPL24
                                                  ** DDPATH  17 IS LOOP AGAIN
                                                  ** DDPATH  18 IS LOOP ESCAPE
 22    2    25     .   .   INVOKE (COMPUTE LENGTH)                             EXAMPL25
 23    1    26     .   END WHILE                                                 EXAMPL26
 24         27     END CASE                                                       EXAMPL27
 25         28     BLOCK (PRINT-RESULTS)                                         EXAMPL28
                                                  ** DDPATH  19 IS PROCEDURE ENTRY
 26    1    29     .   WRITE (6,1)INFO,LENGTH                                   EXAMPL29
 27    1    30   1 .   FORMAT (10X,I5,20X,I5)                                   EXAMPL30
 28         31     END BLOCK                                                      EXAMPL31
 29         32     BLOCK (COMPUTE LENGTH)                                        EXAMPL32
                                                  ** DDPATH  20 IS PROCEDURE ENTRY
 30    1    33     .   LENGTH = LENGTH -10                                      EXAMPL33
 31         34     END BLOCK                                                      EXAMPL34
 32         35     RETURN                                                         EXAMPL35
 33         36     END                                                            EXAMPL36
```

This report is useful for testing purposes because it defines the decision paths.

Figure 4.21. DD-Path Definitions

TRACE Instrumentation

Additional information may be gathered during execution test by inserting INPUT and OUTPUT statements into each source module. The INPUT statements are used to list global variables (either parameters or in COMMON) that will have a value whenever the routine is entered; the OUTPUT statements are used to list variables that will be assigned a value in the routine. An INPUT variable may also be an OUTPUT variable. The INPUT/OUTPUT option provides a dynamic tracing of the values of the program variables.

A type specification must be provided for each variable so the value will be printed the correct format. Omitted types will result in variables being printed according to the most recent previous type, and if there wasn't a previous type the variable(s) will not be printed. The syntax to provide type information is:

INPUT (/<type>/<variable list>,/<type>/<variable list>,...)
OUTPUT (/<type>/<variable list>,/<type>/<variable list>,...)

<type> may be REAL, INTEGER, HOLLERITH, or LOGICAL or the respective abbreviations for each, R, I, H, or L. <variable list> may contain non-subscripted variable names, array names, individual elements of an array, or an array subrange, such as (LIMIT(I), I = M,N) where LIMIT is an array with a dimension of at least N. I is a variable whose value will be undefined after the INPUT or OUTPUT statement is executed.

Some examples are:

```
      INPUT (/I/NUMBER,(LIMIT(I),I=M,N),/R/AREA,RANGE,
     *       /L/DEBUG,/H/TEST)
      OUTPUT (/REAL/AREA,/LOGICAL/DEBUG)
```

The INPUT and OUTPUT statements are turned into comments by AVFS so they may be left in the code when the instrumented code is compiled.

The INPUT/OUTPUT option also performs the same functions as the INSTRUMENT option so the instrumented code on UNIT 9 may be used in the same way as described in Sec. 4.5.1.

The output of this option is the inclusion of a FORTRAN translation of the INPUT and OUTPUT statements in the code written on UNIT 9 (LPUNCH). When the program is executed, the names and values of the variables with type specifications listed in INPUT and OUTPUT statements, will be reported. In addition, a DD-path Definitions report identical to the one from the INSTRUMENT option will be generated, if the LIST option is also specified. Figure 4.22 shows this report for a subroutine with INPUT and OUTPUT statements.

Command

```
OPTION = INPUT/OUTPUT,LIST.   @XQT,TL GRC*AVFS.AVFS
```

Report

Input/Output Listing (Fig. 4.22)

```
DD-PATH DEFINITIONS                SUBROUTINE TIMES ( NUM, RESULT, SUM )

 STMT NEST  LINE  SOURCE...                                                ...SOURCE TAB
-------------------------------------------------------------------------------------
    1          1       SUBROUTINE TIMES ( NUM, RESULT, SUM )
               2
               3
    1          4
               5  C    USING ADDITION TO MULTIPLY
                                                    ** DDPATH   1 IS PROCEDURE ENTRY
    2          7       INTEGER RESULT, SUM, Y
    3          8       INPUT (/I/ NUM, RESULT )
    4          9
    5         10       SUM =0
    6         11       Y = NUM
    7         12       DO WHILE ( Y .GT. 0 )
                                                    ** DDPATH   2 IS LOOP AGAIN
                                                    ** DDPATH   3 IS LOOP ESCAPE
    8    1    13       .   SUM = SUM + RESULT
    9    1    14       .   Y = Y - 1
   10    1    15       .
   11         16       END WHILE
   12         17       OUTPUT (/I/ SUM )
   13         18
   14         19       RETURN
   15         20       END
```

This example shows INPUT and OUTPUT statements in the code.

Figure 4.22. Input/Output Listing

ASSERT Instrumentation

Checks on variables during execution test can be obtained by inserting ASSERT statements into each source module. The ASSERT statements present logical conditions which are assumed to be true. If an assertion goes false, a report giving the module and line number of the false assertion will be printed.

The syntax to provide assertions is:

```
ASSERT ( boolean expression )
```

Some examples are:

```
ASSERT ( HEIGHT .GT. MIN)
ASSERT ( DELAY .LT. MAX .AND. DELAY .GT. MIN)
```

The ASSERT statements are turned into comments by AVFS so they may be left in the code when the instrumented code is compiled.

The ASSERT option performs the same functions as the INSTRUMENT option so the instrumented code on Unit 9 may be used in the same way as described in Sec. 4.5.1.

The output of this option is the inclusion of a FORTRAN translation of the INPUT and OUTPUT statements in the code written on UNIT 9 (LPUNCH). When the program is executed, assertion violations will be recorded along with their location. Figure 4.23 shows a listing of a subroutine with ASSERT statements.

Command	Report
OPTION=ASSERT,LIST.	Assert Listing (Fig. 4.22)

```
SEQ NEST SOURCE                    PROGRAM XPONEN ( INPUT, OUTPUT, TAPE6 = OUTPUT )

  1           PROGRAM XPONEN ( INPUT, OUTPUT, TAPE6 = OUTPUT )
  2      CASON
  3      CUNIT 6
  4      CMODN XPONEN
  5      C
  6      C     EXPONENTIATION BY MULTIPLICATION USING
  7      C     SUBROUTINE TIMES
  8      C
  9           INTEGER ANSWER, RESULT, SUM
 10           INITIAL (.TRUE.)
 11           PRINT 1
 12         1 FORMAT (*1       'NUM  IEXPON  ANSWER    *)
 13           DO ( M = 1, 4 )
 14  1        .   READ 2, NUM, IEXPON
 15  1      2 .   FORMAT ( 2(I5) )
 16  1        .   ASSERT ( NUM .GE. 0 .AND. IEXPON .GE. 0 )
 17  1        .   RESULT = 1
 18  1        .   I = 1
 19  1        .   WHILE ( I .LT. IEXPON )
 20  2        .   .   ASSERT ( RESULT .EQ. NUM ** I .AND. NUM .GE. 0
             *.   .         .AND. IEXPON .GE. 0 .AND. I .LT. IEXPON )
 21  2        .   .   CALL TIMES ( NUM, RESULT, SUM )
 22  2        .   .   RESULT = SUM
 23  2        .   .   I = I + 1
 24  1        .   END WHILE
 25  1        .   ANSWER = RESULT
 26  1        .   PRINT 3, NUM, IEXPON, ANSWER
 27  1      3 .   FORMAT ( 3(I8) )
 28           END DO
 29           FINAL ( ANSWER .EQ. NUM ** IEXPON )
 30           STOP
 31           END
```

Figure 4.23. FORTRAN Assert Instrumentation

4.5.2 AED Instrumentation

Path Instrumentation

During AED instrumentation, the instrumented module is written to UNIT 30 (INSTFL). A DD path definitions report is generated to aid in interpreting test results. The instrumented code is ready for compiling by an AED compiler. The instrumented object code is then ready for loading and test execution.

Also during instrumentation, a special AED problem routine is generated on the file named PROBE. This routine must be compiled and loaded with the instrumented modules. During executiton of the instrumented program, the AED probe progam records the test history in the memory of the CAPS computer. After execution halts, this information is sent to the PDP 11/60 for analysis.

Command	Report
@XQT GRC*INST.INST	DD path definitions (Fig. 4.24)

See Appendix C-21 for a complete JCL example.

```
1     DEFINE PROCEDURE YAW.ENG TO BE
   PATH 1
2     BEGIN
3     Y.FAIL.MON = PGM.V AND NOT YAW.SAS.2FAIL;
4     IF Y.TEST.COMP
5     THEN BEGIN
   PATH 2
6                          IF IN.CH.A
7                          THEN BEGIN
   PATH 3
8                          ASNBIT(Y.FAIL.MON,8,DO.BUFF.3);
9                               END
10                         ELSE
   PATH 4
11                            ASNBIT(Y.FAIL.MON,5,DO.BUFF.3);
   PATH 5
   PATH 6
                               END;
   PATH 7
12                         GSP.V=REFBIT(13,CD.15.MA);
```

Figure 4.24. AED Path Definitions

Trace Instrumentation

Additional information may be gathered during execution test by inserting INPUT and OUTPUT assertions in each source module as was discussed for FORTRAN. In AED these assertions are written as COMMENTS in the following syntax:

COMMENT INPUT <type><variable>;

<type> may be one of the AED data types

<variable> is one of the AED data names.

During execution the value of the variable will be stored along with the execution time, the module name, and the statement number. All this information is sent to the PDP 11/60 for later analysis during execution test.

Figure 4.25 shows a listing of some source with AED INPUT/OUTPUT assertions.

Command	Report
@XQT GRC*TRACE.TRACE	Trace assertion report (Fig. 4.25)

See Appendix C-22 for a complete JCL example.

```
1     BEGIN
2       BOOLEAN ALIGN;
3       BOOLEAN ROLL.OUT;
4       COMMENT INPUT BOOLEAN ALIGN;
5       COMMENT OUTPUT BOOLEAN ROLL.OUT;
6
7       ROLL.OUT = ALIGN OR ROLL.OUT;
8     END FINI
```

Figure 4.25. Trace Assertion Report

Assert Instrumentation

Checks on AED variables during execution can be obtained by inserting ASSERT statements into each source module. In AED these assertions are written as COMMENTS in the following syntax:

```
COMMENT ASSERT boolean-expression;
```

Some examples of assertions are:

```
COMMENT ASSERT HEIGHT > MIN;
COMMENT ASSERT TIME < MAX;
```

The output of the assertion command is instrumented code on UNIT 30 which may be compiled, loaded, and executed.

During execution false assertions will be saved in the CAPS and sent to the PDP 11/60 when some maximum number (presently 30) of false assertions have been detected.

Command	Report
@XQT GRC*ASSERT.ASSERT	Assert Listing (Fig. 4.26)

See Appendix C-23 for a complete JCL example.

```
1     BEGIN
2       BOOLEAN ALIGN; EXTERNAL ALIGN;
3       BOOLEAN ROLL.OUT; EXTERNAL ROLL.OUT;
4       COMMENT ASSERT ROLL.OUT;
5       COMMENT ASSERT ALIGN AND ROLL.OUT;
6       ALIGN = FALSE;
7     END FINI
```

Figure 4.26. AED Assert Listing

4.6 REACHING SET

The analysis specified by the REACHING SET option executes the module retesting assistance capability of AVFS. Presuming that a set of untested DD-paths has been isolated, AVFS helps the user identify sections of code to exercise. The user specifies the desired DD-path number to be "reached," and AVFS generates the reaching set of paths from module entry or from a designated DD-path up to the second DD-path number which has been specified. The user may specify either iterative (explained below) or non-iterative reaching sets to be generated. AVFS prints a list of DD-paths on the reaching set. With this output, the user is able to identify which parts of the program need to be executed (and therefore which program values need to be modified) in order for the selected DD-path to be executed (or reached). Once this determination is made, new test cases can be constructed, and the program can be run again to execute the DD-paths which were not traversed in the previous tests.

The FORTRAN command

```
OPTION = REACHING SET
```

or the AED command

```
@XQT GRC*REACH.REACH
```

enables reaching set analysis to be performed. However, no analysis is performed unless one or more reaching sets are specified. The command for specifying a reaching set is:

```
REACHING SET,MODULE= (<name>),TO= <DD-path number>,
        FROM= <DD-path number> {,ITERATIVE}.
```

The reaching set which includes all possible iterative paths may be generated by appending ITERATIVE (preceded by a coma) to this command, otherwise the command generates a non-iterative reaching set.

A Reaching Set report is in Fig. 4.27; it lists the set of DD-paths within the reaching set, followed by the source statements which make up that set of paths.

FORTRAN Command	Report
OPTION = REACHING SET	Reaching Set (Fig. 4.27a)

See Appendix C-27 for a complete JCL example

AED Command	Report
@XQT GRC*REACH.REACH	Reaching Set (Fig. 4.27b)

See Appendix C-24 for a complete JCL example

```
REACHING SET ANALYSIS                    SUBROUTINE EXAMPL (INFO,LENGTH)

        NON-ITERATIVE REACHING SET FROM DD-PATH   3 TO DD-PATH  14

        DDPATHS IN REACHING SET
                    3    4    5    8    9   11   14

        SOURCE CODE IN REACHING SET
-------------------------------------------------------------------------------
                    . . .
   2         5      IF (INFO.LE.10 .AND. LENGTH.GT.0)THEN
                    . . .
   4         7      ELSE
   5    1    8      .   LENGTH=50
   6         9      END IF
   7        10      CASE OF (INFO+6)
   8        11      CASE (14)
                    . . .
  10        13      CASE (17)
  11    1   14      .   DO WHILE (INFO.LT.20)
  12    2   15      .   .   DO UNTIL (LENGTH.LE.INFO)
  13    3   16      .   .   .   INVOKE (COMPUTE LENGTH)
  14    3   17      .   .   .   IF (LENGTH.GE.30) THEN
                                                    **TARGET DD-PATH BEGINNING**
                    . . .
  16    3   19      .   .   .   END IF
  17    2   20      .   .   END UNTIL
                    . . .
-------------------------------------------------------------------------------
```

This report shows which DD-paths must be traversed, beginning with a specified DD-path to reach the target DD-path. Both the beginning and the ending DD-path numbers are designated by the user in the REACHING SET specification command. Coordination of this report with DD-Path Definitions report allows the user to determine what values must be supplied to the variables to affect the decision predicates so the appropriate path will be taken.

Figure 4.27a. FORTRAN Reaching Set

REACHING SETS FOR MODULE YAW.ENG

1. REACHING SET FROM STATEMENT 0 TO STATEMENT 0

********** REACHING SET ERROR **********

ILLEGAL INPUT STATEMENT NUMBERS... NO COMPUTATION PERFORMED.

****

HERE ARE THE STARTING AND ENDING STMT NUMBERS
FOR PROCEDURES IN THIS MODULE:

1. PROCEDURE YAW.ENG : FROM STMT 14 TO STMT 83

2. REACHING SET FROM STATEMENT 14 TO STATEMENT 50

STMTS IN REACHING SET NO. 1:	14	15	16	17	18	19	20
	33	34	35	36	37	38	39
	40	41	42	44	45	46	47
	48	49	50				
STMTS IN REACHING SET NO. 2:	14	15	16	17	18	19	20
	33	34	35	36	37	38	39
	40	41	43	44	45	46	47
	48	49	50				
STMTS IN REACHING SET NO. 3:	14	15	16	17	18	19	20
	21	22	23	24	25	26	27
	33	34	35	36	37	38	39
	40	41	42	44	45	46	47
	48	49	50				
STMTS IN REACHING SET NO. 4:	14	15	16	17	18	19	20
	21	22	23	24	25	26	27
	33	34	35	36	37	38	39
	40	41	43	44	45	46	47
	48	49	50				
STMTS IN REACHING SET NO. 5:	14	15	16	17	18	19	20
	21	22	28	29	30	31	32
	33	34	35	36	37	38	39
	40	41	42	44	45	46	47
	48	49	50				
STMTS IN REACHING SET NO. 6:	14	15	16	17	18	19	20
	21	22	28	29	30	31	32
	33	34	35	36	37	38	39
	40	41	43	44	45	46	47

Figure 4.27b. AED Reaching Set

4.7 FORMAL VERIFICATION

Two steps are required to perform symbolic execution or verification condition generation. The first is a preliminary step to obtain necessary data, the second is the actual generation.

The formal program verifier uses the program paths of a module to perform symbolic execution or verification condition generation of FORTRAN programs. In AED programs, line numbers are used instead.

4.7.1 FORTRAN Verification

For a preliminary analysis of a FORTRAN program, the command is

OPTION = VCG.

This option generates a DD path definitions report as was shown in Fig. 4.21.

The command to generate a verification condition is

VCG,PATH=<number of paths>,<path list>

where the path list consists of a set of DD path numbers. For example, the command to cover the path from program entry to the first decision statement would be

VCG,PATH=1,1

To cover DD paths 2,4 in a loop construct, the command would be

VCG,PATH=2,2,4

Usually several paths are processed at once.

The DD-Path Definitions Report for the subroutine CIRCLE, Fig. 4.28, shows three DD-path numbers. The IF statement provides two alternative paths, DD-path 2 when the IF expression is evaluated as "true" and DD-path 3 when it is "false".

```
DD-PATH DEFINITIONS              SUBROUTINE CIRCLE ( RADIUS, HEIGHT, AREA, VOLUME )
-------------------------------------------------------------------------------------------------
    1                SUBROUTINE CIRCLE ( RADIUS, HEIGHT, AREA, VOLUME )
                                                                    ** DDPATH   1 IS PROCEDURE ENTRY
    2                DATA PI / 3.1416 /
    3
    4                INITIAL ( RADIUS .GT. 0 )
    5                VOLUME = 0
    6                AREA = PI * RADIUS ** 2
    7                IF ( AREA .GT. RADIUS )
                                                                    ** DDPATH   2 IS TRUE BRANCH
                                                                    ** DDPATH   3 IS FALSE BRANCH
    8 ( 1)           .  VOLUME = HEIGHT * AREA
    9 ( 1)           .  PRINT 1, ( RADIUS, VOLUME )
   10 ( 1)     1     .  FORMAT( 1X, 2 ( F6.2 ) )
   11                ENDIF
   12                FINAL ( VOLUME .EQ. 0 .OR. VOLUME .EQ. HEIGHT * AREA )
   13                RETURN
   14                END
-------------------------------------------------------------------------------------------------
```

Figure 4.28. DD Path Definitions for Verification

For each of these two paths, it is necessary to provide a VCG,PATH command. The commands for generating the verification conditions for CIRCLE would be:

```
OPTION = VCG.
MODULE = (CIRCLE).
VCG,PATH = 2,1,2
VCG,PATH = 2,1,3
```

As a result of these commands, a verification path and a verification condition will be generated as shown in Fig. 4.29.

```
VCG.PATH=2.1.2.                    SUBROUTINE CIRCLE ( RADIUS, HEIGHT, AREA, VOLUME )

 LINE                PATH SOURCE TEXT

    1                SUBROUTINE CIRCLE ( RADIUS, HEIGHT, AREA, VOLUME )

    4                INITIAL ( RADIUS .GT. 0 )
    5                VOLUME = 0
    6                AREA = PI * RADIUS ** 2
    7                IF ( AREA .GT. RADIUS )
    8 ( 1)           .  VOLUME = HEIGHT * AREA
    9 ( 1)           .  PRINT 1, 6 RADIUS, VOLUME )
   11                ENDIF
   12                FINAL ( VOLUME .EQ. 0 .OR. VOLUME .EQ. HEIGHT * AREA )
```

Verification Path

```
SYMBOLICALLY EXECUTED VERIFICATION CONDITION
--------------------------------------------

  LINE     VERIFICATION CONDITION

     4     RADIUS .GT. 0

             AND

     7     PI * ( RADIUS ** 2 ) .GT. RADIUS

---------- IMPLIES ----------------------------------------------------

    12     HEIGHT * PI * ( RADIUS ** 2 ) .EQ. 0 .OR. HEIGHT * PI * ( RADIUS ** 2
           ) .EQ. HEIGHT * PI * ( RADIUS ** 2 )
```

Verification Condition

Figure 4.29. Verification Condition Generation

The VCG EXPRESSION command together with the previous path command allows the symbolic execution of a given expression over a specified path.

Command

```
VCG,EXPRESSION
<expression>   $
VCG,PATH <number of paths>,<path list>
```

Some samples of commands are:

```
OPTION=VCG
MODULE=(EXAMPL).
VCG,EXPRESSION
VOLUME
VCG,PATH=3,1,2,3
VCG,EXPRESSION
DIAMTR .GT. HEIGHT   $
VCG,PATH=3,2,5,3
```

4.7.2 AED Verification

AED Verification is similar to FORTRAN verification except that lines are specified instead of paths.

To symbolically execute an expression C.F.LCH over lines 10, 11, 12 one specifies during the execution of GRC*VCG.VCG:

```
FOR LINES = 10, 11, 12 DO
    C.FO.LCH
END FOR
```

This would result in the report shown in Fig. 4.30 which lists the original expression, the specified lines, and the executed expression. A range of lines is specified by

```
first line .. last line
```

so the previous command could be

```
FOR LINES = 10..12 DO
    C.FO.LCH
END FOR
```

SYMBOLIC EXECUTION REPORT

ORIGINAL EXPRESSION

C.FD.LCH

SOURCE CODE

```
10  ALIGN = TRUE;
11  RUN.WAY = FALSE;
12  C.FD.LCH = ALIGN OR C.FD.LCH;
```

FINAL EXPRESSION

TRUE OR C.FD.LCH

Figure 4.30. AED Symbolic Execution Report

To generate a verification condition, it is necessary to specify assertions at each end of the path to be verified. By specifying VCG as the expression to be symbolically executed, a verification condition will be generated using assertions previously placed in the text. Figure 4.31 illustrates an AED verification condition report listing the source and verification condition. The command to generate the figure would be

```
FOR LINES 9..12 DO
    VCG
END FOR.
```

SOURCE CODE

```
9   COMMENT ASSERT GLIDE SLOPE > 2 AND RANGE > 1000;
10
11  ALTITUDE = RANGE * GLIDE SLOPE * 3.14159/180.0;
12  COMMENT  ASSERT ALTITUDE > 30;
```

VERIFICATION CONDITION

```
GLIDE SLOPE > 2 AND RANGE > 1000
IMPLIES
RANGE * GLIDE SLOPE * 3.14159/180.0 > 30
```

See Appendix C-28 for a complete JCL example.

Figure 4.31. AED Verification Condition Report

5 AVFS CONSTRAINTS

AVFS imposes certain restrictions on the size of the interface file, the command language, and the source text to be analyzed. Most of the limitations based on size are generous (e.g., the maximum number of nested IF statements is one hundred). AVFS is capable of handling quite large source text files. Unusually large programs may have to be processed by several successive executions, each operating on a separate file of modules.

5.1 UNIVERSAL CONSTRAINTS

- At most 250 modules may use the same common block
- Maximum of one card for any given command
- Maximum of 24 commas in any given command
- Maximum of 80 characters per source card image read
- The maximum number of DD-paths which can begin at a statement is 50
- The maximum number of statements on a single DD-path is 100
- The sizes of the two random files UNIT 2 (LIBNEW) and UNIT 13 (LIBWSP) are established using a DEFINE FILE statement in FAVS. The current sizes are 500 records (of 300 words each)
- Maximum of 250 tokens per statement

5.2 SYNTAX CONSTRAINTS

The following implementation constraints are the current ones which must be observed:

- Each module placed on the same interface library must have a unique name
- If any errors are detected in the source, one or more statements may be flagged as not parsed

- Maximum nesting depth of 25 DOs in FORTRAN
- DELETE, START EDIT, STOP EDIT are not recognized
- Switch labels may appear only in assigned GO-TO statements
- ** is the only valid exponentiation symbol
- ASCII FORTRAN debug statements are not recognized
- An initial comment statement will be recognized as the start of a module

6 ANALYZER COMMANDS

A variety of coverage analysis reports can be generated from data collected during execution of a FORTRAN program that was instrumented by AVFS. (The INSTRUMENT option was discussed in Sec. 4.5.) Figure 6.1 shows the execution coverage sequence.

In order to proceed with instrumented software testing, the source text (which has been instrumented by AVFS) is compiled and executed. At program linkage time, any user externals necessary for execution of the instrumented code must be supplied. During test execution the program operates normally, reading its own data and writing its own outputs. The instrumented modules call the data collection routine as their test probes are encounted which records (on UNIT 12) the accumulated data on module DD-path traversals.

Each test execution may consist of a number of test cases. The program identifies the end of each test case by executing a special call to the data collection routine. The identification calls are automatically inserted at the end of main programs. Other are inserted by direction of the user, via the TESTBOUND command, at instrumentation time as discussed in Sec. 4.5.

The coverage reports are generated by a set of commands that differ slightly from the AVFS commands (Sec. 3, 4, 5); for this reason the ANALYZER commands are presented in this separate section.

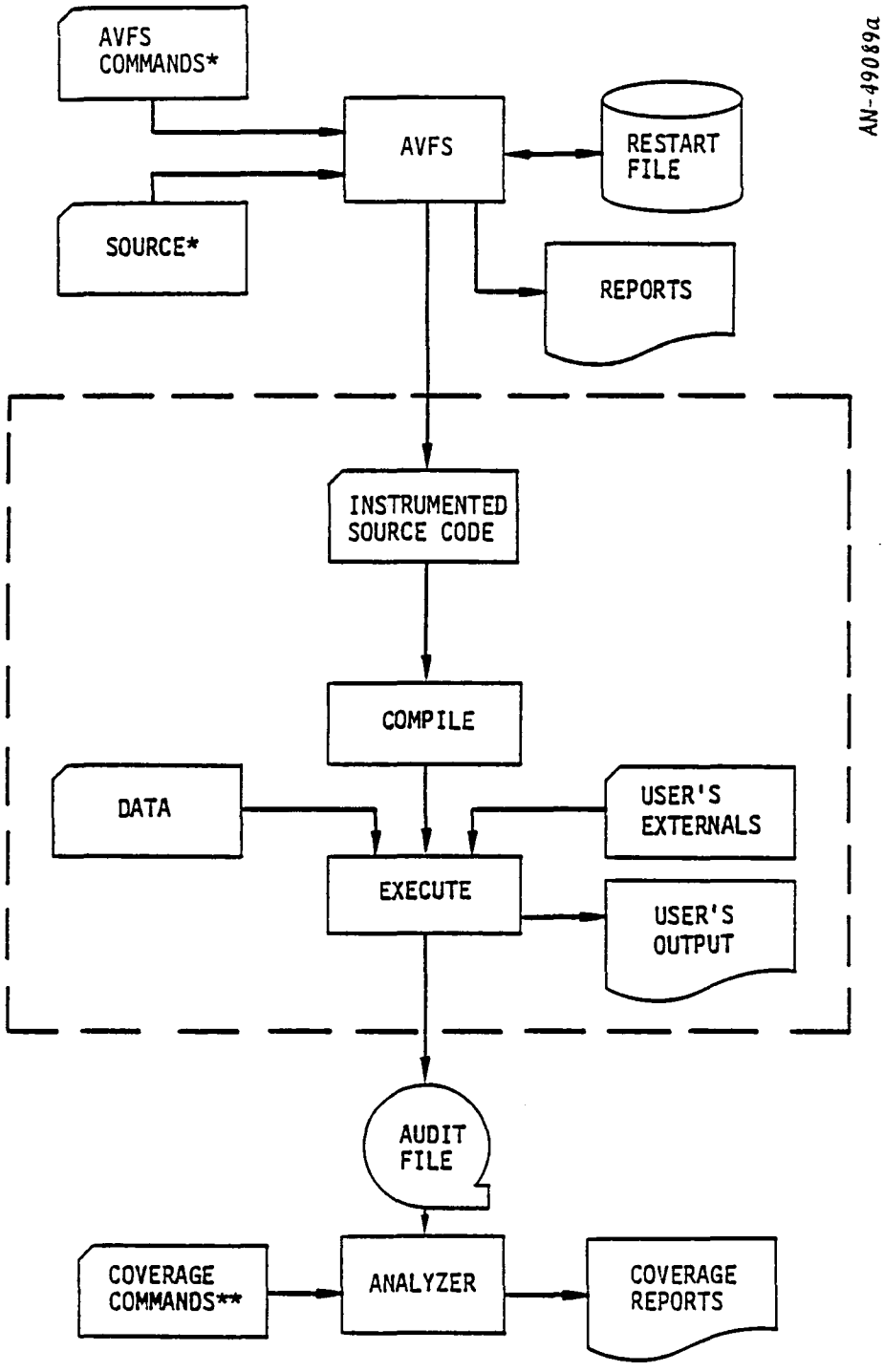


Figure 6.1. Execution Coverage Sequence

There are two ANALYZER commands, an option selection and a module selection command. The type of report is specified by the command:

OPTION{S} = <list>

<list> may be one or more of the three options: SUMMARY, NOTHIT, or DETAILED.

If the DETAILED option is specified, then the OPTION command must be preceded by one or more module selection commands:

FOR MODULE{S} = (<name-1>, <name-2>, ... <name-n>)

<name> is the name of the module (subroutine, function or program). A maximum of 100 modules may be specified at one time. More than one FOR MODULE{S} command may be used to accommodate all specified modules. The DETAILED reports will be generated only for the modules named in this command which have been both instrumented and invoked.

Since the Coverage Analysis program records execution trace data in internal tables, the amount of data recorded is limited by table size. The limitations are given below:

Maximum number of modules to analyze	100
Maximum number of test cases	10
Maximum number of DD-paths to analyze	2000
Maximum number of DD-paths not traversed in any test case	1000

6.1 SUMMARY

The SUMMARY option produces a report which summarizes testing coverage for all instrumented and invoked modules. Figure 6.2 shows a sample SUMMARY report, which lists the following information:

- Test case number
- Module names and numbers of DD-paths
- Number of module invocations, number of DD-paths traversed, and percent coverage for this test case
- Cumulative number of module invocations, number of DD-paths traversed, and percent coverage for all test cases

When multiple test cases are involved, the SUMMARY report shows data from the current test case and the immediately preceding test case. When the end of the trace data is encountered, a cumulative summary of all test cases is produced (Fig. 6.3).

Command

```
OPTION = SUMMARY
```

Reports

DD-path Summary	(Fig. 6.2)
Multiple Test Summary	(Fig. 6.3)

TEST CASE	MODULE NAME	NUMBER OF D-D PATHS	SUMMARY -- THIS TEST: NUMBER OF INVOCATIONS	D-D PATHS TRAVERSED	PER CENT COVERAGE	CUMULATIVE SUMMARY: NUMBER OF TESTS	INVOCATIONS	TRAVERSED	COVERAGE
2	MAIN	5	0	2	40.00	2	1	3	60.00
	CLASS	98	1	18	18.37	2	2	29	29.59
	EXAMPL	16	1	6	37.50	2	2	7	43.75
	CALLER	3	0	0	0.00	2	1	2	66.67
	SSALLSS	122		26	21.31	2		41	33.61
3	MAIN	5	0	2	40.00	3	1	4	80.00
	CLASS	98	1	34	34.69	3	3	52	42.86
	EXAMPL	16	1	11	68.75	3	3	15	93.75
	CALLER	3	0	0	0.00	3	1	2	66.67
	SSALLSS	122		47	38.52	3		63	51.64

Figure 6.2. DD-Path Summary (with the Immediately Preceding Test Case)

TEST CASE	MODULE NAME	NUMBER OF D-D PATHS	SUMMARY -- THIS TEST: NUMBER OF INVOCATIONS	D-D PATHS TRAVERSED	PER CENT COVERAGE	CUMULATIVE SUMMARY: NUMBER OF TESTS	NUMBER INVOCATIONS	D-D PATHS TRAVERSED	PER CENT COVERAGE
1	MAIN	5	1	3	60.00	1	1	3	60.00
	CLASS	98	1	26	26.53	1	1	26	26.53
	EXAMPL	16	1	6	37.50	1	1	6	37.50
	CALLER	3	1	2	66.67	1	1	2	66.67
	$$ALL$$	122		37	30.33	1		37	30.33
2	MAIN	5	0	2	40.00	2	1	3	60.00
	CLASS	98	1	18	18.37	2	2	29	29.59
	EXAMPL	16	1	6	37.50	2	2	7	43.75
	CALLER	3	0	0	0.00	2	1	2	66.67
	$$ALL$$	122		26	21.31	2		41	33.61
3	MAIN	5	0	2	40.00	3	1	4	80.00
	CLASS	98	1	34	34.69	3	3	42	42.86
	EXAMPL	16	1	11	68.75	3	3	15	93.75
	CALLER	3	0	0	0.00	3	1	2	66.67
	$$ALL$$	122		47	38.52	3		63	51.64
4	MAIN	5	0	2	40.00	4	1	4	80.00
	CLASS	98	1	34	34.69	4	4	42	42.86
	EXAMPL	16	1	6	37.50	4	4	15	93.75
	CALLER	3	0	0	0.00	4	1	2	66.67
	$$ALL$$	122		42	34.43	4		63	51.64

Figure 6.3. Multiple Test DD-Path Summary

6.2 NOTHIT

The NOTHIT option requests a report which lists DD-paths not executed for all instrumented and invoked modules. Figure 6.4 shows a sample NOTHIT report, which lists the following information:

- Module names
- Test case number
- Number of DD-paths not traversed, for this test case and for all test cases
- DD-path numbers not traversed for this test case and for all test cases.

Command

```
OPTION = NOTHIT
```

Report

DD-paths Not Executed (Fig. 6.4)

MODULE NAME	TEST NUMBER	PATHS NOT HIT	LIST OF DECISION TO DECISION PATHS NOT EXECUTED
<MAIN >	3	3	1 2 5
	CUMUL	1	2
<CLASS >	3	64	3 5 6 7 10 13 17 19 20 21 26 28 30 32 34 36 37 38 39 40 41 42 43 46 48 50 51 52 53 54 57 58 59 60 62 65 66 67 68 70 72 73 74 75 76 77 78 79 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98
	CUMUL	56	3 5 6 7 10 13 17 19 20 21 26 30 32 34 36 37 38 39 40 42 43 48 51 52 53 54 58 59 62 65 66 67 68 70 74 75 76 77 78 79 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98
<EXAMPL >	3	5	2 4 6 13 14
	CUMUL	1	4
<CALLER >	3	3	1 2 3
	CUMUL	1	2

Figure 6.4. DD-Paths Not Executed

6.3 DETAILED

The DETAILED option command selects a report which shows a breakdown of individual DD-path coverage. A single testcase report like the one in Fig. 6.5 is generated for each specified module which was instrumented and invoked. Figure 6.6 shows the cumulative report, which is generated after the individual testcase reports. Both provide the following information:

- Module name
- Test case number
- List of DD-path numbers, with an indication of those which were not executed, a graphical representation of the number of executions, and an itemized listing of the number of executions
- Overall module coverage data

Command

FOR MODULES = (<name-1>,<name-2>,...<name-n)

Reports

Single Test DD-path Execution	(Fig. 6.5)
Cumulative DD-path Executions	(Fig. 6.6)

Rule

1. Maximum of 100 modules names specified.
2. Repeat the module selection command as necessary; e.g.,
   FOR MODULES = (<name-1>,...,<name-i>)
   FOR MODULES = (<name-i+1>,...,<name-n>)
3. The module selection command must precede the DETAILED option.

RECORD OF DECISION TO DECISION (DD PATH) EXECUTION

MODULE SCLASS 8 TEST CASE NO. 3

DD PATH NUMBER	NO.	NOT EXECUTED	NUMBER OF EXECUTIONS -- NORMALIZED TO MAXIMUM .....20.....40.....60.....80.....100.		NUMBER OF EXECUTIONS
1			XXX	1	1
2			XXX	2	1
3	3	00000			
4			XXX	4	1
5	5	00000			
...	...	00000	... ...		
7	7	00000			
8			XXXXXXXXXX	8	3
9			XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	9	11
10	10	00000			
11			XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	11	10
12			XXX	12	1
13	13	00000			
14			XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	14	13
15			XXX	15	1
16			XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	16	9
17	17	00000			
18			XXX	18	1
19	19	00000			
...	...	00000	... ...		
21	21	00000			
22			XXX	22	1
23			XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	23	10
24			XXX	24	1
25			XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	25	9
26	26	00000			
27			XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	27	9
28	28	00000			
29			XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	29	9
30	30	00000			
31			XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	31	9
32	32	00000			
33			XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	33	9
34	34	00000			
35			XXX	35	1
36	36	00000			
...	...	00000	... ...		
43	43	00000			
44			XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	44	9
45			XXX	45	1
46	46	00000			
47			XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	47	13
48	48	00000			
49			XXX	49	1
50	50	00000			
...	...	00000	... ...		
54	54	00000			
55			XXXXXXX	55	2
56			XXXXXXX	56	2
57	57	00000			
...	...	00000	... ...		
60	60	00000			
61			XXX	61	1
62	62	00000			
63			XXX	63	1
64			XXX	64	1
65	65	00000			
...	...	00000	... ...		
68	68	00000			
69			XXX	69	1
70	70	00000			
71			XXX	71	1
72	72	00000			
...	...	00000	... ...		
79	79	00000			
80			XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	80	10
81			XXX	81	1
82			XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	82	9
83	83	00000			
...	...	00000	... ...		
98	98	00000			

TOTAL NUMBER OF DD PATH EXECUTIONS = 160

TOTAL OF 64 NOT EXECUTED EXECUTED 34/ 98 PERCENT EXECUTED = 34.69

Figure 6.5. Single Test DD-Path Execution

```
RECORD OF DECISION TO DECISION (DD PATH) EXECUTION

MODULE  $CLASS  $          CUMULATIVE RESULTS OF     6 TEST CASES

DD PATH  I  NO.  NOT EXECUTED  I   NUMBER OF EXECUTIONS -- NORMALIZED TO MAXIMUM          I      I  NUMBER OF
NUMBER   I                     I.-------20.-------40.-------60.-------80.------100.      I      I  EXECUTIONS
---------I-------------------------------------------------------------------------------I------I-----------
         I                     I                                                         I      I
       1 I                     I                                                         I    1 I           6
       2 I                     I                                                         I    2 I           6
       3 I   3         00000 I                                                         I      I           7
       4 I                     I                                                         I    4 I           6
       5 I   5         00000 I                                                         I      I
     ... I ...         00000 I ... ... ...                                             I      I
       7 I   7         00000 I                                                         I      I
       8 I                     I XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX    I    8 I         176
       9 I                     I XXXXXXXX                                              I    9 I          36
      10 I  10         00000 I                                                         I      I
      11 I                     I XXXXXXXX                                              I   11 I          34
      12 I                     I                                                       I   12 I           2
      13 I  13         00000 I                                                         I      I
      14 I                     I XXXX                                                  I   14 I          20
      15 I                     I                                                       I   15 I           2
      16 I                     I XXXX                                                  I   16 I          18
      17 I  17         00000 I                                                         I      I
      18 I                     I                                                       I   18 I           2
      19 I  19         00000 I                                                         I      I
     ... I ...         00000 I ... ... ...                                             I      I
      21 I  21         00000 I                                                         I      I
      22 I                     I                                                       I   22 I           2
      23 I                     I XXXXXXXX                                              I   23 I          34
      24 I                     I                                                       I   24 I           2
      25 I                     I XXXXXXX                                               I   25 I          32
      26 I  26         00000 I                                                         I      I
      27 I                     I XXXXXXX                                               I   27 I          32
      28 I                     I                                                       I   28 I           1
      29 I                     I XXXXXXX                                               I   29 I          31
      30 I  30         00000 I                                                         I      I
      31 I                     I XXXXXXX                                               I   31 I          31
      32 I  32         00000 I                                                         I      I
      33 I                     I XXXXXXX                                               I   33 I          31
      34 I  34         00000 I                                                         I      I
      35 I                     I                                                       I   35 I           2
      36 I  36         00000 I                                                         I      I
     ... I ...         00000 I ... ... ...                                             I      I
      40 I  40         00000 I                                                         I      I
      41 I                     I                                                       I   41 I           1
      42 I  42         00000 I                                                         I      I
      43 I  43         00000 I                                                         I      I
      44 I                     I XXXXXXX                                               I   44 I          33
      45 I                     I                                                       I   45 I           1
      46 I                     I                                                       I   46 I           3
      47 I                     I XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX    I   47 I         208
      48 I  48         00000 I                                                         I      I
      49 I                     I                                                       I   49 I           3
      50 I                     I                                                       I   50 I           1
      51 I  51         00000 I                                                         I      I
     ... I ...         00000 I ... ... ...                                             I      I
      54 I  54         00000 I                                                         I      I
      55 I                     I X                                                     I   55 I           6
      56 I                     I X                                                     I   56 I           6
      57 I                     I XX                                                    I   57 I           9
      58 I  58         00000 I                                                         I      I
      59 I  59         00000 I                                                         I      I
      60 I                     I XX                                                    I   60 I           9
      61 I                     I                                                       I   61 I           3
      62 I  62         00000 I                                                         I      I
      63 I                     I                                                       I   63 I           3
      64 I                     I                                                       I   64 I           3
      65 I  65         00000 I                                                         I      I
     ... I ...         00000 I ... ... ...                                             I      I
      68 I  68         00000 I                                                         I      I
      69 I                     I                                                       I   69 I           3
      70 I  70         00000 I                                                         I      I
      71 I                     I                                                       I   71 I           1
      72 I                     I                                                       I   72 I           2
      73 I                     I                                                       I   73 I           2
      74 I  74         00000 I                                                         I      I
     ... I ...         00000 I ... ... ...                                             I      I
      79 I  79         00000 I                                                         I      I
      80 I                     I XX                                                    I   80 I          10
      81 I                     I                                                       I   81 I           1
      82 I                     I XX                                                    I   82 I           9
      83 I  83         00000 I                                                         I      I
     ... I ...         00000 I ... ... ...                                             I      I
      98 I  98         00000 I                                                         I      I
---------------------------------------------------------------------------------------------------------

                                                          TOTAL NUMBER OF DD PATH EXECUTIONS =     817

  TOTAL OF  56  NOT EXECUTED     EXECUTED  42/ 98        PERCENT EXECUTED =    42.86
```

Figure 6.6. Cumulative DD-path Execution

The AVFS options are specified by a command file. For FORTRAN the command file contains the OPTIONS command as described in Section 3.3 or a UNIVAC executes statement. For AED the command file contains a UNIVAC execute statement. Sample command files are given in Appendix C.

APPENDIX A

AVFS COMMAND SUMMARY AND CHECKLIST

FORTRAN Commands

Either an OPTION or REPORT command is required; the other commands are used when it is appropriate. They must appear in the order shown. Where an abbreviation is allowed, it appears to the right of the command; the appropriate UNIVAC run stream command is at the far right.

RESTART REST @XQT,R

RESTART instructs AVFS to use a saved restart file from a previous run.

EXPAND EXPA @XQT,E

EXPAND allows additional source to be added to a restart file.

FILE,PUNCH = <file number> FILE,PUNC=<file number>

Instructs AVFS to reassign PUNCH; the default is UNIT 9.

INSTRUMENT, PUNCH, PROBE, (<file number>)

Instructs AVFS to reassign the data collection file. The default is UNIT 12.

FIRSTLINE = (<run stream command>)

Instructs AVFS to insert the given run stream command as the first line of each element.

OPTIONS=<list> OPTI = <list>

<list> may contain one or more of the following options, separated by commas:

LIST	LIST	@XQT,L
DOCUMENT	DOCU	@XQT,D
SUMMARY	SUMM	@XQT,B
STATIC	STAT	@XQT,S
INSTRUMENT	INST	@XQT,I
INPUT/OUTPUT	INPU	@XQT,T
REACHING SET	REAC	none

REPORT = <list> REPO = <list>

<list> may contain one or more of the following reports, separated by commas:

REPORT NAME	MINIMUM ABBREVIATION	REPORT GENERATED
COMMONS	CO	Commons Summary
PROFILE	PR	Statement Profile
INVOCATIONS	L	Entrys and invocation summary
COMMONS/ENHANCED	CO/E	Common Matrices
BANDS/n	B or B/n	Invocation Bands where n is the number of levels
SPACE	SP	Invocation Space
SYMBOLS	SY	Symbol Report
READS	R	I/O Statements
CROSS	CR	Symbol Cross Reference
PICTURE	PI	Picture of module structure

FOR MODULE = (<name1>,<name2>,....)

module selection command.

TESTBOUND,MODULE = (<name>),STATEMENT = <number>

Used with instrumentation command for setting test case boundaries.

REACHING SET,MODULE = (<name>),TO = <DD-path number>,
FROM = <DD-path number>{,ITERATIVE}

When the option, REACHING SET, is used, it is necessary to specify one or more reaching sets with the above command. The use of ITERATIVE is optional; if present, an iterative reaching set is generated.

ANALYZER COMMANDS

Selection of ANALYZER reports desired must be made by the user. The type or report is specified in the command,

OPTION(S) = <list>

<list> may contain one or more of the following options, separated by commas:

DETAILED	DETA
NOTHIT	NOTH
SUMMARY	SUMM

When the DETAILED option is listed, reports will be generated only for those modules that are listed in a command,

FOR MODULE(S) = (<name-1>,<name-2>, ...,<name-n>).

<name> is the name of the subroutine, function or program. This module selection command must precede the OPTION = DETAILED command.

```
@XQT GRC*LIST.LIST
      enhanced listing

@XQT GRC*STATIC.STATIC
      static analysis

@XQT GRC*CROSS.CROSS
      local cross reference

@XQT GRC*SYMBOL.SYMBOL
      symbols report

@XQT GRC*DEPEND.DEPEND
      module dependencies

@XQT GRC*GLOBAL.GLOBAL
      global cross reference

@XQT GRC*PROFILE.PROFILE
      statement profile

@XQT GRC*UNITS.UNITS
      dimensional analysis

@XQT GRC*TRACE.TRACE
      trace instrumentation

@XQT GRC*ASSERT.ASSERT
      assertion instrumentation
```

```
@XQT GRC*INST.INST
      path instrumentation

@XQT GRC*REACH.REACH
      reaching set generation

@XQT GRC*VCG.VCG
      symbolic execution and verification condition generation

@XQT GRC*TREE.TREE
      invocation hierarcy (calling tree)

@XQT GRC*FLOW.FLOW
      flowgraph

@XQT GRC*INTER.INTER
      interface report
```

APPENDIX B

FILE DESCRIPTIONS

FILE NUMBER	DATA STRUCTURE	MODE (1)	STORAGE (2)	RECORD FORMAT	RECOMMENDED ALLOCATION	USAGE (3)
2	library	B	R	system standard (4)	scratch file	R/W
13	workspace	B	R	system standard (4)	scratch file	R/W
4	user commands	H	S	card image	scratch file	R/W
5	commands input	H	S	card image	system card reader permanent file	R
6	reports	H	S	128 characters/ line maximum	system printer	W
9	instrumented/ restructured source	H	S	card image	scratch file	R/W
5	source	H	S	card image	system card reader permanent file	R
8	new restart file	B	S	system standard	permanent file	W
11	old restart file	B	S	system standard	permanent file	R
12	probe test data trace file	B	S	system standard	permanent file	R

Notes: (1) B = binary; H = character
(2) R = random; S = sequential
(3) R = read only; R/W = read and/or write; W = write only
(4) Installation dependent

AED FILE DESCRIPTIONS

FILE NUMBER	DATA STRUCTURE	MODE	STORAGE	FORMAT	RECOMMENDED ALLOCATION
5	SOURCE	H	S	CARD IMAGE	STANDARD INPUT
6	REPORTS	H	S	TEXT LINES	STANDARD OUTPUT
20	TOKEN FILE	H	S	TEXT LINES	TEMPORARY
25	MULTI-MODULE INFO.	H	S	TEXT LINES	TEMPORARY
30	INSTRUMENTED CODE	H	S	CARD IMAGE	TEMPORARY

APPENDIX C

JOB STREAMS FOR AVFS AT UNIVAC INSTALLATIONS

AVFS INITIAL RUN - CREATES AN INTERFACE FILE

```
@HDG                                **  AVFS INITIAL RUN **
@ASG,A  YOURSOURCE.                 .   YOUR FORTRAN OR SOURCE
@USE    Y.,YOURSOURCE.              .
@ASG,CP YOURFILE.,F///400           .   (OPTIONAL) CATALOG INTERFACE FILE
@USE 8.,YOURFILE.                   .
@ASG,A  GRC*AVFS.                   .   ASG AVFS TRAN, ANALYZER
@USE    R.,GRC*AVFS.                .
@ASG,T  2.,F///1500                 .
@ASG,T  13.,F///1500                .
@XQT    R.AVFS                      .   EXECUTE AVFS
FOR MODULES=(LIST OF MODULES).      .   (OPTIONAL) DEFAULT IS ALL MODULES
OPTION=STATIC.                      .   ANY LIST OF VALID OPTIONS (SEC. 4)

@EOF                                .   SEPARATES AVFS COMMANDS FROM YOUR
                                        SOURCE
@ADD,P  Y.PROCS                     .   (OPTIONAL) ADD PROCS HERE
@ADD,P  Y.ELEMENTS                  .   ADD SOURCE ELEMENTS HERE
@FIN
```

AVFS EXPAND RUN - EXPANDS AN INTERFACE FILE

```
@HDG                                **  AVFS EXPAND RUN **
@ASG,A   YOURSOURCE.                .  YOUR FORTRAN OR SOURCE
@USE     Y.,YOURSOURCE.             .
@ASG,A   YOURFILE.                  .  ASG AVFS INTERFACE FILE (FROM
                                       PREVIOUS RUN)
@USE     11.,YOURFILE.              .
@ASG,A   GRC*AVFS.                  .  ASG AVFS, TRAN, ANALYZER
@USE     R.,GRC*AVFS.               .
@ASG,CP  NEWFILE.                   .  NEW INTERFACE FILE
@USE     8.,NEWFILE.                .
@ASG,T   2.,F///1500                .
@ASG,T   13.,F///1500               .
@XQT     R.AVFS                     .  EXECUTE AVFS (EXPAND PLUS OPTIONS)
EXPAND.                             .  EXPAND INTERFACE FILE
FOR MODULES=(LIST OF MODULES).      .  (OPTIONAL) DEFAULT IS ALL MODULES
OPTION=STATIC.                      .  ANY LIST OF VALID OPTIONS (SEC.
                                       4.)
@EOF                                .  SEPARATES AVFS COMMANDS FROM YOUR
                                       SOURCE
@ADD,P   Y.PROCS                    .  (OPTIONAL) ADD PROCS HERE
@ADD,P   Y.ELEMENTS                 .  ADD SOURCE ELEMENTS HERE
@FIN
```

AVFS RESTART RUN - USES AN INTERFACE FILE

```
@HDG                                **  AVFS RESTART RUN **
@ASG,A  YOURFILE.                   .   AVFS INTERFACE FILE (FROM PREVIOUS
                                        RUN)
@USE 11.,YOURFILE.                  .
@ASG,A  GRC*AVFS.                   .   ASG AVFS, TRAN, ANALYZER
@USE    R.,GRC*AVFS.                .
@ASG,T  2.,F///1500                 .
@ASG,T  13.,F///1500                .
@XQT    R.AVFS                      .   EXECUTE AVFS USING A RESTART FILE
RESTART.
OPTION=STATIC.                      .   ANY LIST OF VALID OPTIONS (SEC 4.)
@FIN
```

AVFS INSTRUMENT, EXECUTE AND ANALYZE RUN

```
@HDG                                    **  AVFS INSTRUMENT, EXECUTE, AND
                                            ANALYZE RUN  **

@ASG,A  YOURSOURCE.                     .   YOUR FORTRAN SOURCE

@USE    Y.,YOURSOURCE.                  .

@ASG,A  GRC*AVFS.                       .   ASG AVFS, TRAN, ANALYZER

@USE    R.,GRC*AVFS.                    .

@ASG,T  2.,F///1500                     .

@ASG,T  13.,F///1500                    .

@XQT    R.AVFS                          .   EXECUTE AVFS (INSTRUMENT AND LIST)

FILE,PUNCH=10.                          .   PLACE INSTRUMENTED SOURCE ON UNIT
                                            10

INSTRUMENT,PUNCH,PROBE, (20).           .   COLLECT DATA ON UNIT 20

FIRSTLINE = (@FTN,S TPF$.+).            .   PREPARE SOURCE FOR ASCII COMPILER

OPTION = INSTRUMENT,LIST.               .   INSTRUMENT AND LIST

TESTBOUND,MODULE=(MAIN),                .   TEST CASE BOUNDARY
STATEMENT=10.

@ADD,P  Y.PROCS                         .   (OPTIONAL) ADD PROCS HERE

@ADD,P  Y.ELEMENTS                      .   ADD SOURCE ELEMENTS HERE

@EOF                                    .

@ADD,P 10.                              .   YOUR INSTRUMENTED SOURCE IS ON 10.

@MAP                                    .   MAP FOR YOUR PROGRAM

@XQT                                    .   EXECUTE YOUR INSTRUMENTED PROGRAM

        ( YOUR DATA )

@XQT    R.ANALYZER                      .   EXECUTE COVERAGE ANALYZER

FOR MODULES=(LIST OF INSTRUMENTED ELEMENTS).

OPTION=SUMMARY,NOTHIT,DETAILED.         .   ANY LIST OF VALID OPTIONS (SEC. 6)

@FIN
```

AVFS FLOWGRAPH ANALYSIS

```
@HDG                ** AVFS FLOWGRAPH ANALYSIS **
@ASG,A GRC*AVFS.
@ASG,A YOURSOURCE.
@ASG,T TEMP.
@ASG,T 2.,F40///1500
@ASG,T 13.,F40///1500
@USE 8.,TEMP.
@XQT GRC*AVFS.AVFS
OPTION = DOCUMENT
REPORT = PICTURE
@EOF
@ADD,P YOURSOURCE.
@EOF
@FIN
```

AVFS STATIC ANALYSIS

```
@HDG                ** AVFS STATIC ANALYSIS **
@ASG,A GRC*AVFS.
@ASG,A YOURSOURCE.
@ASG,T TEMP.
@ASG,T 2.,F40///1500
@ASG,T 13.,F40///1500
@USE 8.,TEMP.
@XQT GRC*AVFS.AVFS
OPTION = STATIC
FOR MODULES = (MAIN,SORT,CALC)
@EOF
@ADD,P YOURSOURCE.
@EOF
@FIN
```

AVFS REACHING SET ANALYSIS

```
@HDG                ** AVFS REACHING SET ANALYSIS **
@ASG,A GRC*AVFS.
@ASG,A YOURSOURCE.
@ASG,T TEMP.
@ASG,T 2.,F40///1500
@ASG,T 13.,F40///1500
@USE 8.,TEMP.
@XQT GRC*AVFS.AVFS
OPTION = REACH
REACHING SET, MODULE = (SORT), TO = 7, FROM =3.
@EOF
@ADD,P YOURSOURCE.
@EOF
@FIN
```

AVFS INDENTED LISTING

```
@HDG                ** AVFS INDENTED LISTING **
@ASG,A GRC*AVFS.
@ASG,A YOURSOURCE.
@ASG,T TEMP.
@ASG,T 2.,F40///1500
@ASG,T 13.,F40///1500
@USE 8.,TEMP.
@XQT GRC*AVFS.AVFS
OPTION = LIST
@EOF
@ADD,P YOURSOURCE.
@EOF
@FIN
```

AVFS DOCUMENTATION

```
@HDG                ** AVFS DOCUMENTATION **
@ASG,A GRC*AVFS.
@ASG,A YOURSOURCE.
@ASG,T TEMP.
@ASG,T 2.,F40///1500
@ASG,T 13.,F40///1500
@USE 8.,TEMP.
@XQT GRC*AVFS.AVFS
OPTION = DOCUMENT
@EOF
@ADD,P YOURSOURCE.
@EOF
@FIN
```

AVFS SUMMARY

```
@HDG                **   AVFS SUMMARY **
@ASG,A GRC*AVFS.
@ASG,A YOURSOURCE.
@ASG,T TEMP.
@ASG,T 2.,F40///1500
@ASG,T 13.,F40///1500
@USE 8.,TEMP.
@XQT GRC*AVFS.AVFS
OPTION = SUMMARY
@EOF
@ADD,P YOURSOURCE.
@EOF
@FIN
```

AVFS INDENTED LISTING

```
@HDG                **   AVFS INDENTED LISTING **
@ASG,A GRC*LIST.
@ASG,A YOURSOURCE.
@ASG,AX INSERTS.
@XQT GRC*LIST.LIST
@ADD,P YOURSOURCE.
@FIN
```

AVFS SYMBOLS

```
@HDG                    ** AVFS SYMBOLS **
@ASG,AX GRC*SYMBOL.SYMBOL
@ASG,A YOURSOURCE.
@ASG,T TKFIL.
@USE 21.,TKFIL.
@ASG,AX INSERTS.
@ASG,T LKFIL.
@USE 22.,LKFIL.
@XQT GRC*SYMBOL.SYMBOL
@ADD,P YOURSOURCE.
@FIN
```

AVFS UNITS ANALYSIS

```
@HDG                **   AVFS UNITS ANALYSIS **
@ASG,A GRC*UNITS.
@ASG,A YOURSOURCE.
@ASG,AX INSERTS.
@XQT GRC*UNITS.UNITS
@ADD,P YOURSOURCE.
@FIN
```

AVFS INTERFACE

```
@HDG                    ** AVFS INTERFACE **
@ASG,AX GRC*INTER.
@ASG,A YOURSOURCE.
@ASG,T TKFIL.
@USE 21.,TKFIL.
@ASG,AX INSERTS.
@ASG,T LKFIL.
@USE 22.,LKFIL.
@ASG,AX  LIBOLD.
@USE 11.,LIBOLD.
@CAT LIBNEW.
@ASG,AX LIBNEW.
@USE 8.,LIBNEW.
@XQT GRC*INTER.INTER
@ADD,P YOURSOURCE.
@FIN
```

AVFS CROSS REFERENCE

```
@HDG                    ** AVFS CROSS REFERENCE **
@ASG,AX GRC*CROSS.
@ASG,A YOURSOURCE.
@ASG,T TKFIL.
@USE 21.,TKFIL.
@ASG,AX INSERTS.
@ASG,T LKFIL.
@USE 22.,LKFIL.
@XQT GRC*CROSS.CROSS
@ADD,P YOURSOURCE.
@FIN
```

AVFS INVOCATIONS

```
@HDG                    ** AVFS INVOCATIONS **
@ASG,AX GRC*INVOKE.
@ASG,A YOURSOURCE.
@ASG,T TKFIL.
@USE 21.,TKFIL.
@ASG,AX INSERTS.
@ASG,T LKFIL.
@USE 22.,LKFIL.
@XQT GRC*INVOKE.INVOKE
@ADD,P YOURSOURCE.
@FIN
```

AVFS DEPEND

```
@HDG                    ** AVFS DEPEND **
@ASG,AX GRC*DEPEND.
@ASG,A YOURSOURCE.
@ASG,T TKFIL.
@USE 21.,TKFIL.
@ASG,AX INSERTS.
@ASG,T LKFIL.
@USE 22.,LKFIL.
@XQT GRC*DEPEND.DEPEND
@ADD,P YOURSOURCE.
@FIN
```

AVFS TREE

```
@HDG                    ** AVFS TREE **
@ASG,AX GRC*TREE.
@ASG,A YOURSOURCE.
@ASG,T TKFIL.
@USE 21.,TKFIL:
@ASG,AX INSERTS.
@ASG,T LKFIL.
@USE 22.,LKFIL.
@XQT GRC*TREE.TREE
@ADD,P YOURSOURCE.
@FIN
```

AVFS GLOBAL CROSS REFERENCE

```
@HDG                    ** AVFS CROSS REFERENCE **
@ASG,AX GRC*INTER.
@ASG,A YOURSOURCE.
@ASG,T TKFIL.
@USE 21.,TKFIL.
@ASG,AX INSERTS.
@ASG,T LKFIL.
@USE 22.,LKFIL.
@ASG,T FTN028.
@USE 28.,FTN028.
@XQT GRC*GLOBAL.GLOBAL
@ADD,P YOURSOURCE.FIRST
@XQT GRC*GLOBAL.GLOBAL
@ADD,P YOURSOURCE.SECOND
@XQT GRC*GLOBAL.PRINT
@FIN
```

AVFS STATEMENT PROFILE

```
@HDG                ** AVFS STATEMENT PROFILE **
@ASG,A GRC*PROFILE.
@ASG,A YOURSOURCE.
@ASG,AX INSERTS.
@XQT GRC*PROFILE.PROFILE
@ADD,P YOURSOURCE.
@FIN
```

AVFS INSTRUMENTATION

```
@HDG                ** AVFS INSTRUMENTATION  **
@ASG,A GRC*INST.
@ASG,A INSERTS.
@ASG,T INSTFL.
@USE 30.,INSTFL.
@ASG,T TABFIL.
@USE 25.,TABFIL.
@ASG,A YOURSOURCE.
@XQT GRC*INST.INST
@ADD,P YOURSOURCE.
@ASG,T PROBE.
@ASG,A AUTO.
@ASG,A ALTLIB*FTN.
@ALTLIB*FTN.FTN,F AUTO.MAIN
@PACK AUTO.
@TPF AUTO.
@PACK
@PREP
@USE 3.,PROBE.
@XQT AUTO.MAP
@COPY,I INSTFL.,YOURSOURCE.INSTFL
@COPY,I PROBE.,YOURSOURCE.PROBE
@CAPS*CROSS.AEDCAPS,SC YOURSOURCE.INSTFL,YOURSOURCE.INSTFL
@CAPS*CROSS.AEDCAPS,SC YOURSOURCE.PROBE,YOURSOURCE.PROBE
@CAPS*CROSS.CASM2,S YOURSOURCE.INSTFL,YOURSOURCE.OBJNAM
@CAPS*CROSS.CASM2,S YOURSOURCE.PROBE,YOURSOURCE.DDPATH
@CAPS*CROSS.LINK,LRSI ,YOURSOURCE.INSTFL
ORIGIN OF PROGRAM
INCLUDE FOR PROGRAM
END
@XQT CAPS*CROSS.HPLDTAPE
YOURSOURCE      .INSTFL
@FIN
```

AVFS TRACE INSTRUMENTATION

```
@HDG                ** AVFS TRACE **

@ASG,A GRC*TRACE.

@ASG,A INSERTS.

@ASG,T INSTFL.

@USE 30.,INSTFL.

@ASG,T TABFIL.

@USE 25.,TABFIL.

@ASG,A YOURSOURCE.

@XQT GRC*TRACE.TRACE

@ADD,P YOURSOURCE.

@COPY,I INSTFL.,YOURSOURCE.INSTFL

@CAPS*CROSS.AEDCAPS,SC YOURSOURCE.INSTFL,YOURSOURCE.INSTFL

@CAPS*CROSS.CASM2,S YOURSOURCE.INSTFL,YOURSOURCE.OBJNAM

@CAPS*CROSS.LINK,LRSI ,YOURSOURCE.INSTFL

ORIGIN OF PROGRAM
INCLUDE FOR PROGRAM

END

@XQT CAPS*CROSS.HPLDTAPE

YOURSOURCE      .INSTFL

@FIN
```

AVFS ASSERTION INSTRUMENTATION

```
@HDG                ** AVFS ASSERTION INSTRUMENTATION  **

@ASG,A GRC*ASSERT.

@ASG,A INSERTS.

@ASG,T INSTFL.

@USE 30.,INSTFL.

@ASG,T TABFIL.

@USE 25.,TABFIL.

@ASG,A YOURSOURCE.

@XQT GRC*ASSERT.ASSERT

@ADD,P YOURSOURCE.

@COPY,I INSTFL.,YOURSOURCE.INSTFL

@CAPS*CROSS.AEDCAPS,SC YOURSOURCE.INSTFL,YOURSOURCE.INSTFL

@CAPS*CROSS.CASM2,S YOURSOURCE.INSTFL,YOURSOURCE.OBJNAM

@CAPS*CROSS.LINK,LRSI ,YOURSOURCE.INSTFL

ORIGIN OF PROGRAM
INCLUDE FOR PROGRAM

END

@XQT CAPS*CROSS.HPLDTAPE

YOURSOURCE      .INSTFL

@FIN
```

AVFS REACHING SET LISTING

```
@HDG                ** AVFS REACHING SET LISTING **
@ASG,A GRC*REACH.
@ASG,A YOURSOURCE.
@ASG,AX INSERTS.
@XQT GRC*REACH.REACH
@ADD,P YOURSOURCE.
@EOF
    1   5
@FIN
```

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-166346		

4. Title and Subtitle	5. Report Date
Automated Verification of Flight Software - User's Manual	April 1982
	6. Performing Organization Code
7. Author(s)	8. Performing Organization Report No.
S. H. Saib	
9. Performing Organization Name and Address	10. Work Unit No.
General Research Corp. P.O. Box 6770 Santa Barbara, CA 93111	T3236Y
	11. Contract or Grant No.
	NAS 2-10550
12. Sponsoring Agency Name and Address	13. Type of Report and Period Covered
National Aeronautics and Space Administration Washington, D.C. 20546	Contractor Report
	14. Sponsoring Agency Code
	512-54-11

15. Supplementary Notes

Technical Monitor: Pio de Feo, Mail Stop 210-9, NASA-Ames Research Center, Moffett Field, CA 94035
FTS 448-5048 Com'l: 415-965-5048

16. Abstract

AVFS (Automated Verification of Flight Software) is a collection of tools for analyzing source programs written in FORTRAN and AED. AVFS aids in improving the quality and the reliability of flight software by providing:

- o Indented listings of source programs
- o Static analysis to detect inconsistencies in the use of variables and parameters
- o Automated documentation
- o Instrumentation of source code
- o Retesting guidance
- o Analysis of assertions
- o Symbolic execution
- o Generation of verification conditions
- o Simplification of verification conditions

This manual describes how to use AVFS in the verification of flight software.

AVFS has been installed at NASA-Ames Research Center, Moffett Field, California. The AVFS tools interface with a PDP 11/60 computer and a CAPS 6 based digital flight control systems to form a complete flight software V&V environment.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Flight Software Test Tools, Software Verification	Unlimited STAR Category 61

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price*
Unclassified	Unclassified	109	

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

End of Document