
S. H. Saib

CONTRACT NAS2– 10550
April 1982

S. H. Saib
General Research Corp.

Prepared for
Ames Research Center
under Contract NAS2-10550
AFVS (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:

- 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

<table>
<thead>
<tr>
<th>Section</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>1</td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Testing and Verification Sequence</td>
<td>3</td>
</tr>
<tr>
<td>1.2 User's Manual Organization</td>
<td>7</td>
</tr>
<tr>
<td>2 AVFS OVERVIEW</td>
<td>9</td>
</tr>
<tr>
<td>3 USING AVFS</td>
<td>15</td>
</tr>
<tr>
<td>3.1 Interface File</td>
<td>21</td>
</tr>
<tr>
<td>3.2 Instrument File</td>
<td>22</td>
</tr>
<tr>
<td>3.3 Firstline</td>
<td>22</td>
</tr>
<tr>
<td>3.4 Processing Options</td>
<td>23</td>
</tr>
<tr>
<td>3.5 Report</td>
<td>24</td>
</tr>
<tr>
<td>3.6 For Modules</td>
<td>25</td>
</tr>
<tr>
<td>4 OPTION DESCRIPTIONS</td>
<td>27</td>
</tr>
<tr>
<td>4.1 List</td>
<td>27</td>
</tr>
<tr>
<td>4.2 Static</td>
<td>31</td>
</tr>
<tr>
<td>4.3 Document</td>
<td>39</td>
</tr>
<tr>
<td>4.4 Summary</td>
<td>61</td>
</tr>
<tr>
<td>4.5 Instrument</td>
<td>67</td>
</tr>
<tr>
<td>4.6 Reaching Set</td>
<td>82</td>
</tr>
<tr>
<td>4.7 Formal Verification</td>
<td>86</td>
</tr>
<tr>
<td>5 AVFS CONSTRAINTS</td>
<td>93</td>
</tr>
<tr>
<td>5.1 Universal Constraints</td>
<td>93</td>
</tr>
<tr>
<td>5.2 Syntax Constraints</td>
<td>93</td>
</tr>
<tr>
<td>6 ANALYZER COMMANDS</td>
<td>95</td>
</tr>
<tr>
<td>6.1 Summary</td>
<td>98</td>
</tr>
<tr>
<td>6.2 Nothit</td>
<td>101</td>
</tr>
<tr>
<td>6.3 Detailed</td>
<td>103</td>
</tr>
</tbody>
</table>

## APPENDIX

- A AVFS COMMAND SUMMARY AND CHECKLIST
- B FILE DESCRIPTIONS
- C JOB STREAMS FOR AVFS AT UNIVAC INSTALLATIONS
### FIGURES

<table>
<thead>
<tr>
<th>NO.</th>
<th>Description</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Digital Flight Control System Verification Library</td>
<td>vi</td>
</tr>
<tr>
<td>1.1</td>
<td>DFCS Verification Laboratory Block Diagram</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td>AVFS Capabilities</td>
<td>3</td>
</tr>
<tr>
<td>1.3</td>
<td>Steps in Validating a Program with AVFS</td>
<td>4</td>
</tr>
<tr>
<td>2.1</td>
<td>Software Verification Augmented by AVFS</td>
<td>10</td>
</tr>
<tr>
<td>2.2</td>
<td>Sequence of Source Program Analysis, Test, and Formal Verification</td>
<td>11</td>
</tr>
<tr>
<td>2.3</td>
<td>Report Index</td>
<td>14</td>
</tr>
<tr>
<td>3.1</td>
<td>AVFS Processing</td>
<td>15</td>
</tr>
<tr>
<td>4.1a</td>
<td>FORTRAN Listing</td>
<td>29</td>
</tr>
<tr>
<td>4.1b</td>
<td>AED Listing</td>
<td>30</td>
</tr>
<tr>
<td>4.2</td>
<td>Static Analysis - FORTRAN</td>
<td>35</td>
</tr>
<tr>
<td>4.3</td>
<td>Static Analysis - AED</td>
<td>36</td>
</tr>
<tr>
<td>4.4</td>
<td>Units Analysis - AED</td>
<td>37</td>
</tr>
<tr>
<td>4.5</td>
<td>Interface Analysis - AED</td>
<td>38</td>
</tr>
<tr>
<td>4.6a</td>
<td>FORTRAN Symbols Report</td>
<td>41</td>
</tr>
<tr>
<td>4.6b</td>
<td>AED Symbols Report</td>
<td>42</td>
</tr>
<tr>
<td>4.7a</td>
<td>FORTRAN Cross Reference Report</td>
<td>44</td>
</tr>
<tr>
<td>4.7b</td>
<td>AED Cross Reference</td>
<td>45</td>
</tr>
<tr>
<td>4.8a</td>
<td>FORTRAN Invocation Space Report</td>
<td>47</td>
</tr>
<tr>
<td>4.8b</td>
<td>AED Invocation Space Report</td>
<td>48</td>
</tr>
<tr>
<td>4.9a</td>
<td>FORTRAN Invocation Summary Report</td>
<td>49</td>
</tr>
<tr>
<td>4.9b</td>
<td>AED Invocation Summary Report</td>
<td>50</td>
</tr>
<tr>
<td>4.10a</td>
<td>FORTRAN Invocations Band</td>
<td>51</td>
</tr>
<tr>
<td>4.10b</td>
<td>AED Invocations Band</td>
<td>52</td>
</tr>
<tr>
<td>4.11</td>
<td>FORTRAN Common Matrices</td>
<td>53</td>
</tr>
<tr>
<td>4.12</td>
<td>FORTRAN I/O Statements</td>
<td>54</td>
</tr>
<tr>
<td>4.13</td>
<td>FORTRAN Commons Cross Reference</td>
<td>55</td>
</tr>
<tr>
<td>4.14a</td>
<td>FORTRAN Externals Cross Reference</td>
<td>56</td>
</tr>
<tr>
<td>4.14b</td>
<td>AED Global Cross Reference</td>
<td>57</td>
</tr>
<tr>
<td>4.15</td>
<td>FORTRAN Picture of Module Structure</td>
<td>59</td>
</tr>
<tr>
<td>NO.</td>
<td>FIGURE</td>
<td>PAGE</td>
</tr>
<tr>
<td>-------</td>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>4.16</td>
<td>AED Picture of Module Structure</td>
<td>60</td>
</tr>
<tr>
<td>4.17a</td>
<td>FORTRAN Statement Profile</td>
<td>64</td>
</tr>
<tr>
<td>4.17b</td>
<td>AED Statement Profile</td>
<td>65</td>
</tr>
<tr>
<td>4.18</td>
<td>FORTRAN Common Summary</td>
<td>66</td>
</tr>
<tr>
<td>4.19</td>
<td>AVFS Instrumentation of Source Code</td>
<td>68</td>
</tr>
<tr>
<td>4.20</td>
<td>Loading and Test Execution</td>
<td>70</td>
</tr>
<tr>
<td>4.21</td>
<td>DD-Path Definition</td>
<td>72</td>
</tr>
<tr>
<td>4.22</td>
<td>Input/Output Listing</td>
<td>75</td>
</tr>
<tr>
<td>4.23</td>
<td>FORTRAN Assert Instrumentation</td>
<td>77</td>
</tr>
<tr>
<td>4.24</td>
<td>AED Path Definition</td>
<td>79</td>
</tr>
<tr>
<td>4.25</td>
<td>Trace Assertion Report</td>
<td>80</td>
</tr>
<tr>
<td>4.26</td>
<td>AED Assert Listing</td>
<td>81</td>
</tr>
<tr>
<td>4.27a</td>
<td>FORTRAN Reaching Set</td>
<td>84</td>
</tr>
<tr>
<td>4.27b</td>
<td>AED Reaching SET</td>
<td>85</td>
</tr>
<tr>
<td>4.28</td>
<td>DD-Path Definition for Verification</td>
<td>87</td>
</tr>
<tr>
<td>4.29</td>
<td>Verification Condition Generation</td>
<td>88</td>
</tr>
<tr>
<td>4.30</td>
<td>AED Symbolic Execution Report</td>
<td>90</td>
</tr>
<tr>
<td>4.31</td>
<td>AED Verification Condition Report</td>
<td>91</td>
</tr>
<tr>
<td>6.1</td>
<td>Execution Coverage Sequence</td>
<td>96</td>
</tr>
<tr>
<td>6.2</td>
<td>DD-Path Summary (with the Immediately Preceding Test Case)</td>
<td>99</td>
</tr>
<tr>
<td>6.3</td>
<td>Multiple Test DD-Path Summary</td>
<td>100</td>
</tr>
<tr>
<td>6.4</td>
<td>DD-Paths Not Executed</td>
<td>102</td>
</tr>
<tr>
<td>6.5</td>
<td>Single Test DD-Path Execution</td>
<td>104</td>
</tr>
<tr>
<td>6.6</td>
<td>Cumulative DD-Path Execution</td>
<td>105</td>
</tr>
</tbody>
</table>
Figure 1. Digital Flight Control System Verification Laboratory
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.
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 than it is practical to assign to them during execution.

6
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 instrumented 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.
AVFS OVERVIEW

This section contains an overview 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 analyzed 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

<table>
<thead>
<tr>
<th>MULTI-MODULE REPORTS</th>
<th>PAGE</th>
<th>MODULE NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERFACE CHANGES</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>INVOCATION SUMMARY</td>
<td>16-17</td>
<td></td>
</tr>
<tr>
<td>COMMON MATRICES</td>
<td>18-26</td>
<td></td>
</tr>
<tr>
<td>I/O STATEMENTS</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>CROSS REFERENCE</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>SUBROUTINE PTRSTR (MODULE, ISTM, IRETRN)</td>
<td>PTRSTR</td>
<td></td>
</tr>
<tr>
<td>SYMBOLOGY</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>CROSS REFERENCE</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>INVOCATION SPACE</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>INVOCATION BANDS</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>SUBROUTINE SDBASA (MODULE, ISTM, IRETRN)</td>
<td>SDBASA</td>
<td></td>
</tr>
<tr>
<td>SYMBOLOGY</td>
<td>4-9</td>
<td></td>
</tr>
<tr>
<td>CROSS REFERENCE</td>
<td>9-12</td>
<td></td>
</tr>
<tr>
<td>INVOCATION SPACE</td>
<td>13-14</td>
<td></td>
</tr>
<tr>
<td>INVOCATION BANDS</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>CROSS REFERENCE</td>
<td>29-31</td>
<td></td>
</tr>
</tbody>
</table>

...WRITING INTERFACE LIBRARY

... 4075 WORDS WRITTEN

Figure 2.3. Report Index
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 analysis. 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 MODULES = <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>
<thead>
<tr>
<th>New Source?</th>
<th>Existing Interface File?</th>
<th>Command to Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>No</td>
<td>---</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>EXPAND</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
<td>Do Nothing</td>
</tr>
<tr>
<td>No</td>
<td>Yes</td>
<td>RESTART</td>
</tr>
</tbody>
</table>

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 line 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**

<table>
<thead>
<tr>
<th></th>
<th>Is unit 9 suitable for the instrumented code output</th>
<th>Is unit 12 suitable for the data collected during an execution of instrumented code?</th>
<th>Do Univac job control cards need to be inserted between modules?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>do not use FILE command</td>
<td>do not use INSTRUMENT command</td>
<td>use FIRSTLINE command</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>do not use FIRSTLINE command</td>
</tr>
<tr>
<td></td>
<td>use FILE command</td>
<td>use INSTRUMENT command</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>FORTRAN</th>
<th>AED</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST</td>
<td>GRC*LIST.LIST</td>
</tr>
<tr>
<td>STATIC</td>
<td>GRC*STATIC.STATIC</td>
</tr>
<tr>
<td>DOCUMENT</td>
<td>GRC*CROSS.CROSS</td>
</tr>
<tr>
<td></td>
<td>GRC*SYMBOL.SYMBOL</td>
</tr>
<tr>
<td></td>
<td>GRC*INVOC.INVOKE</td>
</tr>
<tr>
<td></td>
<td>GRC*TREE.TREE</td>
</tr>
<tr>
<td></td>
<td>GRC*DEP.DEPEND</td>
</tr>
<tr>
<td></td>
<td>GRC*GLOBAL.GLOBAL</td>
</tr>
</tbody>
</table>

| 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 verification condition generation

In the FORTRAN tools, the command form is

\[ \text{OPTION(S)} = \text{<option list>} \]

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

\[ \text{@XQT} \text{ option} \]

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 | GRC*PROFILE.PROFILE
PROFILE | PR | GRC*INVOC.INVOKE
INVOCATIONS | L | GRC*INVOC.INVOKE
COMMON MATRICES | CO/E | GRC*DEPEND.DEPEND
CALLING TREE | B | GRC*TREE.TREE
SPACE | SP | GRC*DEPEND.DEPEND
SYMBOLS | SY | GRC*SYMBOL.SYMBOL
I/O STATEMENTS | R | GRC*CROSS.CROSS
CROSS | CR | GRC*CROSS.CROSS
FLOWGRAPH | PI | GRC*FLOW.FLOW

In the FORTRAN tools, the command form is

\[ \text{REPORT} = \langle \text{report list} \rangle \]

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

\[ \text{FOR MODULES} = (<\text{name 1}>, <\text{name 2}>, \ldots) \]

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

OPTION=LIST or
@XQT,L GRC*AVFS.AVFS

Report

FORTRAN Listing (Fig. 4.1a)

See Appendix C-8 for complete JCL example

AED Command

@XQT GRC*LIST.LIST

Report

AED Listing (Fig. 4.1b)

See Appendix C-11 for complete JCL example
<table>
<thead>
<tr>
<th>Options</th>
<th>LIST</th>
<th>STATIC</th>
<th>DOCUMENT</th>
<th>SUMMARY</th>
<th>UNITS</th>
<th>INPUT/OUTPUT</th>
<th>ASSERT</th>
<th>INSTRUMENT</th>
<th>REACHING SET</th>
<th>VCD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Documentation</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maintenance</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Implementation</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Obtain Interface Data</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trace Ranges of Variables</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Check Variables</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Execution Test</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Incomplete Test Coverage</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Test Information</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Single Module Information</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Code Changes</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Unknown Behavior</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Integration</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Acceptance</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dimensional Analysis</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Symbolic Execution</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Formal Verification</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

TABLE 4.1
AVFS PROCESSING OPTIONS WITH SUGGESTED USES FOR EACH OPTION

28
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 TO BE
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       .44255D-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 \text{DIST*TIME} are \text{FEET*SEC} which is printed on the right of the equals sign. The units of \text{SPEED} are \text{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.

\textbf{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

\begin{verbatim}
COMMENT INPUT <type> <variable>;
\end{verbatim}

For example, the assertion

\begin{verbatim}
COMMENT INPUT REAL HEIGHT;
\end{verbatim}

states that the variable named \text{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

\begin{verbatim}
COMMENT OUTPUT <type> <variable>.
\end{verbatim}

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
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
<table>
<thead>
<tr>
<th>NAME</th>
<th>CLASS</th>
<th>MODE</th>
<th>FIRST_STMT</th>
<th>TOTALUSES</th>
<th>LAST_STMT</th>
<th>ASSERTED_USE</th>
<th>ACTUAL_USE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>PARAMETER</td>
<td>REAL</td>
<td>1</td>
<td>3</td>
<td>6</td>
<td>INPUT</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>PARAMETER</td>
<td>INTEGER</td>
<td>1</td>
<td>4</td>
<td>9</td>
<td>BOTH</td>
<td></td>
</tr>
<tr>
<td>ANS</td>
<td>PARAMETER</td>
<td>REAL</td>
<td>1</td>
<td>3</td>
<td>9</td>
<td>OUTPUT</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>LOCAL</td>
<td>INTEGER</td>
<td>2</td>
<td>5</td>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- VARIABLE I

0 LOCAL INTEGER 2 2 4

SET/USE ERROR USED BEFORE BEING ASSIGNED A VALUE

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
Inconsistent units are reported during units analysis.

Figure 4.4. Units Analysis - AED
### INTERFACE REPORT

<table>
<thead>
<tr>
<th>MODULE</th>
<th>TYPE OF CHANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.FORE.EXEC</td>
<td>NEW MODULE</td>
</tr>
<tr>
<td>A.BAK.EXEC</td>
<td>UPDATED MODULE</td>
</tr>
<tr>
<td></td>
<td>**** PARAMETER LENGTH CHANGE</td>
</tr>
<tr>
<td>A.YAW.END</td>
<td>UPDATED MODULE</td>
</tr>
<tr>
<td></td>
<td>**** COMMON TYPE CHANGE</td>
</tr>
</tbody>
</table>

### INTERFACE ANALYSIS

<table>
<thead>
<tr>
<th>MODULE</th>
<th>TYPE OF ERROR</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.MAKE.IT</td>
<td>PARAMETER LENGTHS INCONSISTENT</td>
</tr>
<tr>
<td></td>
<td>ACTUAL - 2 PARAMETERS</td>
</tr>
<tr>
<td></td>
<td>A.BAK.EXEC (C,D)</td>
</tr>
<tr>
<td></td>
<td>FORMAL - 3 PARAMETERS</td>
</tr>
<tr>
<td></td>
<td>A.BAK.EXEC (A,B,C)</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Report</th>
<th>FORTRAN Command</th>
<th>AED Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbols</td>
<td>OPTION=DOCUMENT</td>
<td>GRC*SYMBOL.SYMBOL</td>
</tr>
<tr>
<td></td>
<td>(Fig. 4.6a)</td>
<td>(Fig. 4.6b)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Appendix C-12</td>
</tr>
<tr>
<td>Cross Reference</td>
<td>OPTION=DOCUMENT</td>
<td>GRC*CROSS.CROSS</td>
</tr>
<tr>
<td></td>
<td>(Fig. 4.7a)</td>
<td>(Fig. 4.7b)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Appendix C-15</td>
</tr>
<tr>
<td>Invocation Space</td>
<td>OPTION=DOCUMENT</td>
<td>GRC*INVOKE.INVOKE</td>
</tr>
<tr>
<td></td>
<td>(Fig. 4.8a)</td>
<td>(Fig. 4.8b)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Appendix C-16</td>
</tr>
<tr>
<td>Invocation Summary</td>
<td>OPTION=DOCUMENT</td>
<td>GRC*DEPEND.DEPEND</td>
</tr>
<tr>
<td></td>
<td>(Fig. 4.9a)</td>
<td>(Fig. 4.9b)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Appendix C-17</td>
</tr>
<tr>
<td>Invocation Bands</td>
<td>OPTION=DOCUMENT</td>
<td>GRC*TREE.TREE</td>
</tr>
<tr>
<td></td>
<td>(Fig. 4.10a)</td>
<td>(Fig. 4.10b)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Appendix C-18</td>
</tr>
</tbody>
</table>
Common Matrices \hspace{1cm} \text{OPTION=DOCUMENT} \\
\hspace{1cm} \hspace{1cm} \text{(Fig. 4.11)}

I/O Statements \hspace{1cm} \text{OPTION=DOCUMENT} \\
\hspace{1cm} \hspace{1cm} \text{(Fig. 4.12)}

Commons Cross Reference \hspace{1cm} \text{OPTION=DOCUMENT} \\
\hspace{1cm} \hspace{1cm} \text{GRC*GLOBAL.GLOBAL} \\
\hspace{1cm} \hspace{1cm} \text{(Fig. 4.13) (Fig. 4.14b)} \\
\hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \text{Appendix C-19}

Externals Cross Reference \hspace{1cm} \text{OPTION=DOCUMENT} \\
\hspace{1cm} \hspace{1cm} \text{GRC*GLOBAL.GLOBAL} \\
\hspace{1cm} \hspace{1cm} \text{(Fig. 4.14a) (Fig. 4.14b)} \\
\hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \text{Appendix C-19}

Flowgraph \hspace{1cm} \text{OPTION=DOCUMENT} \\
\hspace{1cm} \hspace{1cm} \text{REPORT=PICTURE} \\
\hspace{1cm} \hspace{1cm} \text{GRC.FLOW.FLOW} \\
\hspace{1cm} \hspace{1cm} \text{(Fig. 4.15) (Fig. 4.16)}

The FORTRAN command

\text{OPTION=DOCUMENT}

maybe replaced with the UNIVAC command

\text{@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

\text{REPORT=PICTURE}

is added after the \text{OPTION} command or the \text{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).
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
<table>
<thead>
<tr>
<th>NAME</th>
<th>SCOPE</th>
<th>CLASS</th>
<th>1ST TOTL</th>
<th>ASSERTED</th>
<th>ACTUAL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>STMT USES</td>
<td>USE</td>
<td>USE</td>
</tr>
<tr>
<td>K3</td>
<td>LOCAL</td>
<td>VARIABLE</td>
<td>74 74</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>K4</td>
<td>LOCAL</td>
<td>VARIABLE</td>
<td>74 74</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>K5</td>
<td>LOCAL</td>
<td>VARIABLE</td>
<td>74 74</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>K6</td>
<td>LOCAL</td>
<td>VARIABLE</td>
<td>74 74</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>K7</td>
<td>LOCAL</td>
<td>VARIABLE</td>
<td>74 74</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>K8</td>
<td>LOCAL</td>
<td>VARIABLE</td>
<td>74 74</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>K9</td>
<td>LOCAL</td>
<td>VARIABLE</td>
<td>74 74</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>LIMIT</td>
<td>EXTERNAL</td>
<td>PROCEDURE</td>
<td>20 23 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LX1</td>
<td>LOCAL</td>
<td>VARIABLE</td>
<td>17 18 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LX2</td>
<td>LOCAL</td>
<td>VARIABLE</td>
<td>17 18 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LX3</td>
<td>LOCAL</td>
<td>VARIABLE</td>
<td>17 17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LX4</td>
<td>LOCAL</td>
<td>VARIABLE</td>
<td>17 17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MONDAY</td>
<td>EXTERNAL</td>
<td>PROCEDURE</td>
<td>104 104</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>MPS1</td>
<td>-</td>
<td>-</td>
<td>47 65 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPS2</td>
<td>-</td>
<td>-</td>
<td>51 68 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPS3</td>
<td>-</td>
<td>-</td>
<td>55 69 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEWPROC</td>
<td>EXTERNAL</td>
<td>PROCEDURE</td>
<td>85 117 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEWPROC2</td>
<td>-</td>
<td>-</td>
<td>119 119 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.6b. AED Symbols Report
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.
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'.
<table>
<thead>
<tr>
<th>NAME</th>
<th>SCOPE</th>
<th>CLASS</th>
<th>TYPE</th>
<th>USED OR REFERENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>K6</td>
<td>LOCAL</td>
<td>VARIABLE</td>
<td>INTEGER</td>
<td>74D</td>
</tr>
<tr>
<td>K7</td>
<td>LOCAL</td>
<td>VARIABLE</td>
<td>INTEGER</td>
<td>74D</td>
</tr>
<tr>
<td>K8</td>
<td>LOCAL</td>
<td>VARIABLE</td>
<td>INTEGER</td>
<td>74D</td>
</tr>
<tr>
<td>K9</td>
<td>LOCAL</td>
<td>VARIABLE</td>
<td>INTEGER</td>
<td>74D</td>
</tr>
<tr>
<td>LIMIT</td>
<td>LOCAL</td>
<td>VARIABLE</td>
<td>INTEGER</td>
<td>74D</td>
</tr>
<tr>
<td>K7</td>
<td>LOCAL</td>
<td>VARIABLE</td>
<td>INTEGER</td>
<td>74D</td>
</tr>
<tr>
<td>K8</td>
<td>LOCAL</td>
<td>VARIABLE</td>
<td>INTEGER</td>
<td>74D</td>
</tr>
<tr>
<td>MONDAY</td>
<td>EXTERNAL</td>
<td>PROCEDURE</td>
<td>REAL</td>
<td>20 21 22</td>
</tr>
<tr>
<td>MPS1</td>
<td>LOCAL</td>
<td>VARIABLE</td>
<td>LABEL</td>
<td>47D 63 64</td>
</tr>
<tr>
<td>MPS2</td>
<td>LOCAL</td>
<td>VARIABLE</td>
<td>LABEL</td>
<td>51S 63 67</td>
</tr>
<tr>
<td>MPS3</td>
<td>LOCAL</td>
<td>VARIABLE</td>
<td>LABEL</td>
<td>55S 64 66</td>
</tr>
<tr>
<td>NEWPROC</td>
<td>EXTERNAL</td>
<td>PROCEDURE</td>
<td>-</td>
<td>85 97 117D</td>
</tr>
<tr>
<td>NEWPROC2</td>
<td>LOCAL</td>
<td>VARIABLE</td>
<td>LABEL</td>
<td>119</td>
</tr>
<tr>
<td>NOCALLFROM</td>
<td>EXTERNAL</td>
<td>PROCEDURE</td>
<td>-</td>
<td>88D 94</td>
</tr>
<tr>
<td>OWN.PTR</td>
<td>LOCAL</td>
<td>VARIABLE</td>
<td>LABEL</td>
<td>158 20 20</td>
</tr>
<tr>
<td>PAR1</td>
<td>LOCAL</td>
<td>VARIABLE</td>
<td>INTEGER</td>
<td>35 62</td>
</tr>
<tr>
<td>PAR2</td>
<td>LOCAL</td>
<td>VARIABLE</td>
<td>INTEGER</td>
<td>5 6D 24</td>
</tr>
<tr>
<td>PAR3</td>
<td>LOCAL</td>
<td>VARIABLE</td>
<td>INTEGER</td>
<td>5 6D 28S</td>
</tr>
<tr>
<td>PROC1</td>
<td>EXTERNAL</td>
<td>PROCEDURE</td>
<td>-</td>
<td>10D 46</td>
</tr>
<tr>
<td>PROC2</td>
<td>EXTERNAL</td>
<td>PROCEDURE</td>
<td>-</td>
<td>13D 46</td>
</tr>
<tr>
<td>PROCCALLO</td>
<td>LOCAL</td>
<td>VARIABLE</td>
<td>INTEGER</td>
<td>37</td>
</tr>
<tr>
<td>PROCCALL1</td>
<td>LOCAL</td>
<td>VARIABLE</td>
<td>INTEGER</td>
<td>39 40 41</td>
</tr>
<tr>
<td>PROCALL2</td>
<td>EXTERNAL</td>
<td>PROCEDURE</td>
<td>-</td>
<td>115</td>
</tr>
<tr>
<td>PROCALL3</td>
<td>LOCAL</td>
<td>VARIABLE</td>
<td>INPUT.POINTE</td>
<td>46 110 113D</td>
</tr>
<tr>
<td>PTR</td>
<td>LOCAL</td>
<td>VARIABLE</td>
<td>LABEL</td>
<td>65 6D 15</td>
</tr>
<tr>
<td>R1</td>
<td>EXTERNAL</td>
<td>SWITCH</td>
<td>LABEL</td>
<td>11D 12</td>
</tr>
<tr>
<td>ROBERT</td>
<td>LOCAL</td>
<td>SWITCH</td>
<td>LABEL</td>
<td>18D</td>
</tr>
<tr>
<td>ROBERT1</td>
<td>EXTERNAL</td>
<td>PROCEDURE</td>
<td>-</td>
<td>82D 98</td>
</tr>
<tr>
<td>ROBERT2</td>
<td>EXTERNAL</td>
<td>PROCEDURE</td>
<td>-</td>
<td>100D</td>
</tr>
<tr>
<td>ROBERTG</td>
<td>LOCAL</td>
<td>VARIABLE</td>
<td>INTEGER</td>
<td>38</td>
</tr>
<tr>
<td>SIGNAL</td>
<td>LOCAL</td>
<td>VARIABLE</td>
<td>LABEL</td>
<td>20 21 22</td>
</tr>
<tr>
<td>SIGNAL.MA</td>
<td>LOCAL</td>
<td>VARIABLE</td>
<td>LABEL</td>
<td>20S 48 49</td>
</tr>
<tr>
<td>SIGNAL.MB</td>
<td>LOCAL</td>
<td>VARIABLE</td>
<td>LABEL</td>
<td>21S 48 50</td>
</tr>
<tr>
<td>SIGNAL.OA</td>
<td>LOCAL</td>
<td>VARIABLE</td>
<td>LABEL</td>
<td>22S 52 54</td>
</tr>
<tr>
<td>SIGNAL.OB</td>
<td>LOCAL</td>
<td>VARIABLE</td>
<td>LABEL</td>
<td>23S 56 58</td>
</tr>
<tr>
<td>SYN1</td>
<td>LOCAL</td>
<td>SYNONYM</td>
<td>-</td>
<td>16D</td>
</tr>
<tr>
<td>SYN1A</td>
<td>LOCAL</td>
<td>SYNONYM</td>
<td>-</td>
<td>16D 36</td>
</tr>
<tr>
<td>SYN1B</td>
<td>LOCAL</td>
<td>SYNONYM</td>
<td>-</td>
<td>16D</td>
</tr>
</tbody>
</table>

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 AII0
STMT = 117  CALL AII0 ( 2 , IOU , LINEX , LLINEX )
STMT = 173  CALL AII0 ( 2 , IOU , HEADX , 120 )
STMT = 174  CALL AII0 ( 2 , IOU , 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 ( 1 ) )
STMT = 121  ICMTST ( CARD ( 2 ) )

INVOCATIONS TO THIS MODULE FROM WITHIN LIBRARY

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

MODULE IFTRA
STMT = 69   CALL CRDOUT ( CARD , 4 )
STMT = 80   CALL CRDOUT ( CARD , 4 )
STMT = 99   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
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

<table>
<thead>
<tr>
<th>ENTRY</th>
<th>LISTS OF CALLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUTLST</td>
<td>WHICH IS DEFINED IN GETBLK</td>
</tr>
<tr>
<td></td>
<td>IS CALLED BY - -NONE- -</td>
</tr>
<tr>
<td></td>
<td>AND CALLS - GETFRG MAKFRG XMIT</td>
</tr>
<tr>
<td>PUTWRD</td>
<td>WHICH IS DEFINED IN GETBLK</td>
</tr>
<tr>
<td></td>
<td>IS CALLED BY - PUTBEF PUTBOT</td>
</tr>
<tr>
<td></td>
<td>AND CALLS - GETFRG MAKFRG XMIT</td>
</tr>
<tr>
<td>XMIT</td>
<td>WHICH IS UNDEFINED</td>
</tr>
<tr>
<td></td>
<td>IS CALLED BY - GETBLK NEXT PREV PUTAT PUTBEF PUTBOT</td>
</tr>
<tr>
<td></td>
<td>THE FOLLOWING ENTRIES ARE NOT CALLED</td>
</tr>
<tr>
<td></td>
<td>GETBLK GETLST GETWRD ISRTAB NEXT PREV PUTAT</td>
</tr>
</tbody>
</table>

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 entries 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
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
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
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
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.

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
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
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 'X' 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

<table>
<thead>
<tr>
<th>NAME</th>
<th>SCOPE</th>
<th>MODULE</th>
<th>USED/SET/EQUIVALENCED ( * INDICATES SET )</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIDBG</td>
<td>DBGCDI1</td>
<td>ISRTAB 82</td>
<td>GETDLK 26E</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ISRTAB 25E</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>NEXT 18E</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PREV 18E</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PUTAT 18E</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PUTDEF 18E</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PUTROT 18E</td>
</tr>
<tr>
<td>FLGXO</td>
<td>AISTO</td>
<td>GETDLK 146</td>
<td>190 231 266</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ISRTAB 67</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>NEXT 39</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PREV 39</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PUTAT 33</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PUTDEF 43</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PUTROT 43</td>
<td></td>
</tr>
<tr>
<td>FMXXXX</td>
<td>AISTO</td>
<td>GETDLK 142</td>
<td>187 227 261</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ISRTAB 62</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>NEXT 36</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PREV 36</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PUTAT 30</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PUTDEF 40</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PUTROT 40</td>
<td></td>
</tr>
<tr>
<td>FRGDIR</td>
<td>POOLCH</td>
<td>GETDLK 149</td>
<td>193# 240# 268#</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ISRTAB 77#</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>NEXT 43#</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PREV 43#</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PUTAT 40#</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PUTDEF 50#</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PUTROT 50#</td>
<td></td>
</tr>
<tr>
<td>FRGXXX</td>
<td>AISTO</td>
<td>GETDLK 148</td>
<td>192 240 268</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ISRTAB 79</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>NEXT 43</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PREV 43</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PUTAT 38</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PUTDEF 50</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PUTROT 50</td>
<td></td>
</tr>
<tr>
<td>FEZZX</td>
<td>AISTO</td>
<td>GETDLK 140</td>
<td>141 125 186 216 219 220 248 249</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ISRTAB 64</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>NEXT 43</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PREV 43</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PUTAT 47#</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PUTDEF 47#</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PUTROT 47#</td>
<td></td>
</tr>
<tr>
<td>IKXXX</td>
<td>AISTO</td>
<td>GETDLK 147# 147 149 191# 191 193 239# 239 240 267# 267 268</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ISRTAB 78# 78 79</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>NEXT 42# 42 43</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PREV 42# 42 43</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PUTAT 37# 37 38</td>
<td></td>
</tr>
</tbody>
</table>

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 'X' 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
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 by 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
<table>
<thead>
<tr>
<th>NAME</th>
<th>CLASS</th>
<th>MODULE</th>
<th>USED/SET ( - INDICATES SET )</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>COMMON</td>
<td>VOTER</td>
<td>71</td>
</tr>
<tr>
<td>C2</td>
<td>COMMON</td>
<td>VOTER</td>
<td>71</td>
</tr>
<tr>
<td>C3</td>
<td>COMMON</td>
<td>VOTER</td>
<td>71</td>
</tr>
<tr>
<td>C4</td>
<td>COMMON</td>
<td>VOTER</td>
<td>76</td>
</tr>
<tr>
<td>CC1</td>
<td>COMMON</td>
<td>VOTER</td>
<td>9</td>
</tr>
<tr>
<td>CC2</td>
<td>COMMON</td>
<td>VOTER</td>
<td>9</td>
</tr>
<tr>
<td>E1</td>
<td>EXTERNAL</td>
<td>VOTER</td>
<td>72</td>
</tr>
<tr>
<td>E2</td>
<td>EXTERNAL</td>
<td>VOTER</td>
<td>72</td>
</tr>
<tr>
<td>E3</td>
<td>EXTERNAL</td>
<td>VOTER</td>
<td>72</td>
</tr>
<tr>
<td>E4</td>
<td>EXTERNAL</td>
<td>VOTER</td>
<td>73</td>
</tr>
<tr>
<td>E5</td>
<td>COMMON</td>
<td>VOTER</td>
<td>78</td>
</tr>
<tr>
<td>E6</td>
<td>COMMON</td>
<td>VOTER</td>
<td>78</td>
</tr>
<tr>
<td>EE1</td>
<td>EXTERNAL</td>
<td>VOTER</td>
<td>14</td>
</tr>
<tr>
<td>EE2</td>
<td>EXTERNAL</td>
<td>VOTER</td>
<td>14</td>
</tr>
<tr>
<td>NEWPROC</td>
<td>EXTERNAL</td>
<td>ROBERT1</td>
<td>85 97 117</td>
</tr>
<tr>
<td>NOCALLFROM</td>
<td>EXTERNAL NOCALLFROM</td>
<td>88 94</td>
<td></td>
</tr>
<tr>
<td>PROC1</td>
<td>EXTERNAL</td>
<td>VOTER</td>
<td>10 46</td>
</tr>
<tr>
<td>PROC2</td>
<td>EXTERNAL</td>
<td>VOTER</td>
<td>13 46</td>
</tr>
<tr>
<td>PROCALL1</td>
<td>EXTERNAL PROCALL1</td>
<td>39 40 41 42 43 44 106 108 111 115</td>
<td></td>
</tr>
<tr>
<td>PROCALL2</td>
<td>EXTERNAL PROCALL2</td>
<td>46 110 113</td>
<td></td>
</tr>
<tr>
<td>R1</td>
<td>EXTERNAL</td>
<td>VOTER</td>
<td>11 12</td>
</tr>
<tr>
<td>ROBERT1</td>
<td>EXTERNAL</td>
<td>ROBERT1</td>
<td>82 98</td>
</tr>
<tr>
<td>ROBERT2</td>
<td>EXTERNAL</td>
<td>ROBERT2</td>
<td>100</td>
</tr>
<tr>
<td>VOTER</td>
<td>EXTERNAL</td>
<td>VOTER</td>
<td>5 41 43 44 45 45 -61</td>
</tr>
<tr>
<td>XPROC</td>
<td>EXTERNAL</td>
<td>ROBERT1</td>
<td>86 92</td>
</tr>
</tbody>
</table>

Figure 4.14b. AED Global Cross Reference
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 modules which reference the global variable are alphabetically ordered in the module column. The remaining columns contain 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
<table>
<thead>
<tr>
<th>UPWARD JUMPS</th>
<th>STATEMENT TEXT</th>
<th>DOWNWARD JUMPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABCDEFGHIJK</td>
<td>DEFINE PROCEDURE YAW.ENG TO BE</td>
<td>ABCDEFGHIJK</td>
</tr>
<tr>
<td></td>
<td>BEGIN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Y.FAIL.MON = PGM.V AND NOT YAW.SAS.2FAIL;</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>IF Y.TEST. COMPL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>THEN BEGIN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IF IN.CH.A</td>
<td>EB</td>
</tr>
<tr>
<td></td>
<td>THEN BEGIN</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>ASNBIT(Y.FAIL.MON, 8, DO.BUFF.3);</td>
<td>EB</td>
</tr>
<tr>
<td></td>
<td>END</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>ELSE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ASNBIT(Y.FAIL.MON, 5, DO.BUFF.3);</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>END;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GSP.V = REFBIT(13, CD.15.MA);</td>
<td>E</td>
</tr>
</tbody>
</table>

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, compound, 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.

<table>
<thead>
<tr>
<th>Report</th>
<th>FORTRAN Command</th>
<th>AED Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statement Profile</td>
<td>OPTION=SUMMARY</td>
<td>GRC*PROFILE.PROFILE</td>
</tr>
<tr>
<td></td>
<td>(Fig. 4.17a)</td>
<td>(Fig. 4.17b)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Appendix C-20</td>
</tr>
<tr>
<td>Invocation Summary</td>
<td>OPTION=SUMMARY</td>
<td>GRC*DEPEND.DEPEND</td>
</tr>
<tr>
<td></td>
<td>(Fig. 4.9a)</td>
<td>(Fig. 4.9b)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Appendix C-17</td>
</tr>
<tr>
<td>Common Summary</td>
<td>OPTION=SUMMARY</td>
<td>GRC<em>INTER</em>INTER</td>
</tr>
<tr>
<td></td>
<td>(Fig. 4.18)</td>
<td>(Fig. 4.5)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Appendix C-14</td>
</tr>
</tbody>
</table>
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.
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
declarations 83 53.2
array 0 0.0
bead 1 0.0
common 0 0.0
component 1 0.0
define 1 0.0
external 25 16.0
insert 3 1.9
pack 0 0.0
preset 0 0.0
procedure 1 0.0
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 lines. The number of lines (including blank lines) 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 REFWAR
BLKSTO    DEPVOK
DBGCOM    MAKTAB
DEPGCM    DEPBNO DEPVOK
EPT       DEPGRP DEPVOK REFWAR
FILES     CMATRX DEPBNO DEPGRP DEPVOK XREFER
GLOBAL    DEPBNO
HALPHA    XREFER
HCHARS    DEPGRP XREFER
MOGIT     CMATRX
ICMOS     STEP15
KDELMS    DEPVOK
MACHINE   DEPVOK
MOD       DEPVOK REFWAR
MRY15     STEP15
MTHST0    DEPGRP DEPVOK
MTHST1    DEPVOK XREFER

This report lists all modules and all common blocks encountered.

Figure 4.18. FORTRAN Common Summary

66
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,

\texttt{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)
```
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)
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.

<table>
<thead>
<tr>
<th>Command</th>
<th>Report</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPTION=ASSERT,LIST.</td>
<td>Assert Listing (Fig. 4.22)</td>
</tr>
</tbody>
</table>

76
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 execution of the instrumented program, the AED probe program 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.
DEFINE PROCEDURE YAW.ENG TO BE
PATH 1
BEGIN
Y.FAIL.MON = PGM.V AND NOT YAW.SAS.2FAIL;
IF Y.TEST.COMP
THEN BEGIN
PATH 2
IF IN.CH.A
THEN BEGIN
PATH 3
ASNBIT(Y.FAIL.MON,8,DO.BUFF.3);
END
ELSE
ASNBIT(Y.FAIL.MON,5,DO.BUFF.3);
END;
PATH 5
PATH 6
END;
PATH 7
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*ASSERTASSERT 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

81
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 comma) 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

OPTION = REACHING SET

Report

Reaching Set (Fig. 4.27a)

See Appendix C-27 for a complete JCL example

AED Command

@XQT GRC*REACH.REACH

Report

Reaching Set (Fig. 4.27b)

See Appendix C-24 for a complete JCL example
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

<table>
<thead>
<tr>
<th>STMTS IN REACHING SET NO. 1:</th>
<th>14 15 16 17 18 19 20</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>33 34 35 36 37 38 39</td>
</tr>
<tr>
<td></td>
<td>40 41 42 44 45 46 47</td>
</tr>
<tr>
<td></td>
<td>48 49 50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STMTS IN REACHING SET NO. 2:</th>
<th>14 15 16 17 18 19 20</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>33 34 35 36 37 38 39</td>
</tr>
<tr>
<td></td>
<td>40 41 43 44 45 46 47</td>
</tr>
<tr>
<td></td>
<td>48 49 50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STMTS IN REACHING SET NO. 3:</th>
<th>14 15 16 17 18 19 20</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>21 22 23 24 25 26 27</td>
</tr>
<tr>
<td></td>
<td>33 34 35 36 37 38 39</td>
</tr>
<tr>
<td></td>
<td>40 41 42 44 45 46 47</td>
</tr>
<tr>
<td></td>
<td>48 49 50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STMTS IN REACHING SET NO. 4:</th>
<th>14 15 16 17 18 19 20</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>21 22 23 24 25 26 27</td>
</tr>
<tr>
<td></td>
<td>33 34 35 36 37 38 39</td>
</tr>
<tr>
<td></td>
<td>40 41 43 44 45 46 47</td>
</tr>
<tr>
<td></td>
<td>48 49 50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STMTS IN REACHING SET NO. 5:</th>
<th>14 15 16 17 18 19 20</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>21 22 23 24 25 26 27</td>
</tr>
<tr>
<td></td>
<td>33 34 35 36 37 38 39</td>
</tr>
<tr>
<td></td>
<td>40 41 42 44 45 46 47</td>
</tr>
<tr>
<td></td>
<td>48 49 50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STMTS IN REACHING SET NO. 6:</th>
<th>14 15 16 17 18 19 20</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>21 22 23 24 25 26 27</td>
</tr>
<tr>
<td></td>
<td>33 34 35 36 37 38 39</td>
</tr>
<tr>
<td></td>
<td>40 41 43 44 45 46 47</td>
</tr>
</tbody>
</table>

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

```plaintext
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

```plaintext
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

```plaintext
VCG,PATH=1,1
```

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

```plaintext
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".
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.
SUBROUTINE CIRCLE ( RADIUS, HEIGHT, AREA, VOLUME )

PATH SOURCE TEXT

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

5 INITIAL ( RADIUS .GT. 0 )

6 VOLUME = 0

7 AREA = PI * RADIUS ** 2

8 ( 1 ) IF ( AREA .GT. RADIALS )

9 ( 1 ) VOLUME = HEIGHT * AREA

11 ENDIF

12 FINAL ( VOLUME .EQ. 0 .OR. VOLUME .EQ. HEIGHT * AREA )

Verification Path

SYMBOLICALLY EXECUTEC VERIFICATION CONDITION

-----------------------------------------------
LINE VERIFICATION CONDITION

* 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

\[ \text{VCG,EXPRESSION} \]
\[ <\text{expression}> \] $
\text{VCG,PATH} <\text{number of paths}>,<\text{path list}>$

Some samples of commands are:

\[
\begin{align*}
\text{OPTION}=\text{VCG} \\
\text{MODULE}=(\text{EXAMPL}) \\
\text{VCG,EXPRESSION} \\
\text{VOLUME} \\
\text{VCG,PATH}=3,1,2,3 \\
\text{VCG,EXPRESSION} \\
\text{DIAMTR} \cdot \text{GT. HEIGHT} \quad $ \\
\text{VCG,PATH}=3,2,5,3 \\
\end{align*}
\]

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:

\[
\begin{align*}
\text{FOR LINES} = 10, 11, 12 \text{ DO} \\
\text{C.FO.LCH} \\
\text{END FOR}
\end{align*}
\]

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.FD.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
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 encountered 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:

\[ \text{OPTION\{S\} = <list>} \]

<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:

\[ \text{FOR Module\{S\} = (<name>-1), (<name>-2), \ldots (<name>-n>)} \]

{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)
Figure 6.2. DD-Path Summary (with the Immediately Preceding Test Case)
<table>
<thead>
<tr>
<th>Case</th>
<th>Module</th>
<th>Number of Q-D Paths</th>
<th>Number of Q-D Pairs</th>
<th>coverage %</th>
<th>Number of Invocations Traversed</th>
<th>Coverage of Tests</th>
<th>Invocations Traversed</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Main</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>60.00</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Class</td>
<td>1</td>
<td>1</td>
<td>26</td>
<td>24.37</td>
<td>1</td>
<td>1</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>Example</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>37.00</td>
<td>2</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Caller</td>
<td>3</td>
<td>2</td>
<td>66.67</td>
<td></td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>ALLLES</td>
<td>122</td>
<td>37</td>
<td>30.33</td>
<td></td>
<td>1</td>
<td>37</td>
<td>30.33</td>
</tr>
<tr>
<td>2</td>
<td>Main</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>60.00</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Class</td>
<td>98</td>
<td>1</td>
<td>18</td>
<td>14.37</td>
<td>3</td>
<td>2</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>Example</td>
<td>14</td>
<td>1</td>
<td>6</td>
<td>37.00</td>
<td>2</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>Caller</td>
<td>3</td>
<td>0</td>
<td>66.67</td>
<td></td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>ALLLES</td>
<td>122</td>
<td>26</td>
<td>21.21</td>
<td></td>
<td>2</td>
<td>25</td>
<td>21.21</td>
</tr>
<tr>
<td>3</td>
<td>Main</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>40.00</td>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Class</td>
<td>98</td>
<td>1</td>
<td>34</td>
<td>34.09</td>
<td>3</td>
<td>3</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>Example</td>
<td>14</td>
<td>1</td>
<td>11</td>
<td>68.75</td>
<td>3</td>
<td>3</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>Caller</td>
<td>3</td>
<td>0</td>
<td>66.67</td>
<td></td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>ALLLES</td>
<td>122</td>
<td>47</td>
<td>38.52</td>
<td></td>
<td>3</td>
<td>63</td>
<td>31.64</td>
</tr>
<tr>
<td>4</td>
<td>Main</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>40.00</td>
<td>4</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>Class</td>
<td>98</td>
<td>1</td>
<td>34</td>
<td>34.09</td>
<td>4</td>
<td>4</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>Example</td>
<td>14</td>
<td>1</td>
<td>6</td>
<td>37.00</td>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Caller</td>
<td>3</td>
<td>0</td>
<td>66.67</td>
<td></td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>ALLLES</td>
<td>122</td>
<td>49</td>
<td>39.43</td>
<td></td>
<td>4</td>
<td>63</td>
<td>51.44</td>
</tr>
</tbody>
</table>

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)
### Figure 6.4. DD-Paths Not Executed

<table>
<thead>
<tr>
<th>MODULE</th>
<th>TEST</th>
<th>PATHS</th>
<th>LIST OF DECISION TO DECISION PATHS NOT EXECUTED</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAIN</td>
<td>3</td>
<td>1</td>
<td>3 1 1 1 2 5</td>
</tr>
<tr>
<td>CLASS</td>
<td>1</td>
<td>1</td>
<td>6 1 3 5 4 7 10 13 17 19 20 21 26 28 30 32 34 36 37 38 39 40 41 43 45 46 48 50 51 52 53 54 55 56 57 59 60 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 90 91 92 93 94 95 96 97 98</td>
</tr>
<tr>
<td>CRANKE</td>
<td>1</td>
<td>1</td>
<td>2 4 13 14</td>
</tr>
<tr>
<td>CALLER</td>
<td>1</td>
<td>1</td>
<td>1 1 1 1</td>
</tr>
</tbody>
</table>
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.
### Figure 6.5. Single Test DD-Path Execution
### Record of Decision to Decision (DD) Path Execution

**Module SC Class 3**

**Cumulative Results of 6 Test Cases**

<table>
<thead>
<tr>
<th>DD Path</th>
<th>No. Not Executed</th>
<th>Number of Executions</th>
<th>Number of Failures - Normalized to Max Run</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>17</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>18</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>21</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>22</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>23</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>26</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>27</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>28</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>29</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>30</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>31</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>32</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>33</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>34</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>35</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>40</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>41</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>42</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>43</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>44</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>45</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>50</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>51</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>52</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>53</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>54</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>55</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>56</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>57</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>58</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>59</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>60</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>61</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>62</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>63</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>64</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>70</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>71</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>72</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>73</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>79</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>80</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>81</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>82</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>98</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>99</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>100</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Total of 50 Not Executed Executed 78/78 Percent Execution 92.86</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.
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

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

EXPAND

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>

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

- LIST
- DOCUMENT
- SUMMARY
- STATIC
- INSTRUMENT
- INPUT/OUTPUT
- REACHING SET

OPTI = <list>

REPORT = <list>

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

<table>
<thead>
<tr>
<th>REPORT NAME</th>
<th>MINIMUM NAME</th>
<th>REPORT GENERATED</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMONS</td>
<td>CO</td>
<td>Commons Summary</td>
</tr>
<tr>
<td>PROFILE</td>
<td>PR</td>
<td>Statement Profile</td>
</tr>
<tr>
<td>INVOCATIONS</td>
<td>L</td>
<td>Entrys and invocation summary</td>
</tr>
<tr>
<td>COMMONS/ENHANCED</td>
<td>CO/E</td>
<td>Common Matrices</td>
</tr>
<tr>
<td>BANDS/n</td>
<td>B or B/n</td>
<td>Invocation Bands where n is the number of levels</td>
</tr>
<tr>
<td>SPACE</td>
<td>SP</td>
<td>Invocation Space</td>
</tr>
<tr>
<td>SYMBOLS</td>
<td>SY</td>
<td>Symbol Report</td>
</tr>
<tr>
<td>READS</td>
<td>R</td>
<td>I/O Statements</td>
</tr>
<tr>
<td>CROSS</td>
<td>CR</td>
<td>Symbol Cross Reference</td>
</tr>
<tr>
<td>PICTURE</td>
<td>PI</td>
<td>Picture of module structure</td>
</tr>
</tbody>
</table>
FOR MODULE = (<namel>,<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,

\[ \text{OPTION(S)} = \langle\text{list}\rangle \]

\(\langle\text{list}\rangle\) may contain one or more of the following options, separated by commas:

- DETAILED
- NOTHIT
- SUMMARY
- DETA
- NOTH
- SUMM

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

\[ \text{FOR MODULE(S)} = (\langle\text{name-1}\rangle, \langle\text{name-2}\rangle, \ldots, \langle\text{name-n}\rangle) \]

\(\langle\text{name}\rangle\) is the name of the subroutine, function or program. This module selection command must precede the \text{OPTION = DETAILED command}.
AED COMMANDS

@XQT GRC*LIST.LIST
   enhanced listing

@XQT GRC*STATIC.STATIC
   static analysis

@XQT GRC*CROSS.CROSS
   local cross reference

@XQT GRC*Ssymbol.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*ASSERTASSERT
   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 hierarchy (calling tree)

@XQT GRC*FLOW.FLOW
    flowgraph

@XQT GRC*INTER.INTER
    interface report
APPENDIX B

FILE DESCRIPTIONS

<table>
<thead>
<tr>
<th>FILE NUMBER</th>
<th>DATA STRUCTURE</th>
<th>MODE</th>
<th>STORAGE</th>
<th>RECORD FORMAT</th>
<th>RECOMMENDED ALLOCATION</th>
<th>USAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>library</td>
<td>B</td>
<td>R</td>
<td>system standard (1)</td>
<td>scratch file</td>
<td>R/W</td>
</tr>
<tr>
<td>13</td>
<td>workspace</td>
<td>B</td>
<td>R</td>
<td>system standard (1)</td>
<td>scratch file</td>
<td>R/W</td>
</tr>
<tr>
<td>4</td>
<td>user commands</td>
<td>H</td>
<td>S</td>
<td>card image</td>
<td>scratch file</td>
<td>R/W</td>
</tr>
<tr>
<td>5</td>
<td>commands input</td>
<td>H</td>
<td>S</td>
<td>card image</td>
<td>system card reader</td>
<td>R</td>
</tr>
<tr>
<td>6</td>
<td>reports</td>
<td>H</td>
<td>S</td>
<td>120 characters/line maximum</td>
<td>system printer</td>
<td>W</td>
</tr>
<tr>
<td>9</td>
<td>instrumented/restructured source</td>
<td>H</td>
<td>S</td>
<td>card image</td>
<td>scratch file</td>
<td>R/W</td>
</tr>
<tr>
<td>5</td>
<td>source</td>
<td>H</td>
<td>S</td>
<td>card image</td>
<td>system card reader</td>
<td>R</td>
</tr>
<tr>
<td>8</td>
<td>new restart file</td>
<td>B</td>
<td>S</td>
<td>system standard</td>
<td>permanent file</td>
<td>W</td>
</tr>
<tr>
<td>11</td>
<td>old restart file</td>
<td>B</td>
<td>S</td>
<td>system standard</td>
<td>permanent file</td>
<td>R</td>
</tr>
<tr>
<td>12</td>
<td>probe test data trace file</td>
<td>B</td>
<td>S</td>
<td>system standard</td>
<td>permanent file</td>
<td>R</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>File Number</th>
<th>Data Structure</th>
<th>Mode</th>
<th>Storage</th>
<th>Format</th>
<th>Recommended Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Source</td>
<td>H</td>
<td>S</td>
<td>Card Image</td>
<td>Standard Input</td>
</tr>
<tr>
<td>6</td>
<td>Reports</td>
<td>H</td>
<td>S</td>
<td>Text Lines</td>
<td>Standard Output</td>
</tr>
<tr>
<td>20</td>
<td>Token File</td>
<td>H</td>
<td>S</td>
<td>Text Lines</td>
<td>Temporary</td>
</tr>
<tr>
<td>25</td>
<td>Multi-Module Info.</td>
<td>H</td>
<td>S</td>
<td>Text Lines</td>
<td>Temporary</td>
</tr>
<tr>
<td>30</td>
<td>Instrumented Code</td>
<td>H</td>
<td>S</td>
<td>Card Image</td>
<td>Temporary</td>
</tr>
</tbody>
</table>

B-2
APPENDIX C

JOB STREAMS FOR AVFS AT UNIVAC INSTALLATIONS

AVFS INITIAL RUN - Creates an interface file

** AVFS INITIAL RUN **

@HDG
@ASG,A YOURSOURCE.
@USE Y.,YOURSOURCE.
@ASG,CP YOURFILE.,F///400
@USE 8.,YOURFILE.
@ASG,A GRC*AVFS.
@USE R.,GRC*AVFS.
@ASG,T 2.,F///1500
@ASG,T 13.,F///1500
@XQT R.AVFS
FOR MODULES=(LIST OF MODULES).
OPTION=STATIC.
@EOF
@ADD,P Y.PROCS
@ADD,P Y.ELEMENTS
@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,C,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 **

- AVFS INTERFACE FILE (FROM PREVIOUS RUN)
- ASG AVFS, TRAN, ANALYZER
- EXECUTE AVFS USING A RESTART FILE
- ANY LIST OF VALID OPTIONS (SEC 4.)
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 TPFS.+). . PREPARE SOURCE FOR ASCII COMPILER
OPTION = INSTRUMENT,LIST. . INSTRUMENT AND LIST
TESTBOUND,MODULE=(MAIN), STATEMENT=10. . TEST CASE BOUNDARY

@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

C-4
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

C-5
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

@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

@DG ** AVFS SUMMARY **
@SG A GRC*AVFS.
@SG A YOURSOURCE.
@SG T TEMP.
@SG T 2.,F40///1500
@SG 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

C-11
AVFS SYMBOLS

@HDG
** AVFS SYMBOLS **

@ASG,AX GRC*SYMBOL-SYMBOL
@ASG,A YOURSOURCE.
@ASG,T TCFIL.
@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

C-13
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 **

@ASC,AX GRC*CROSS.
@ASC,A YOURSOURCE.
@ASC,T TKFIL.
@USE 21.,TKFIL.
@ASC,AX INSERTS.
@ASC,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,A YOURSOURCE.
@ASG,T TKFIL.
@USE 21.,TKFIL.
@ASG,A INSERTS.
@ASG,T LKFIL.
@USE 22.,LKFILE.
@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 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 FTPN028.
@USE 28.,FTP028.
@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 AUTOMAIN
@PACK AUTO.
@TPF AUTO.
@PACK AUTO.
@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
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:

- 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 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.
End of Document