NASA Technical Memorandum 88313

Digital-Flight-Control-System Software Written in Automated-Engineering-Design Language: A User's Guide of Verification and Validation Tools

Jim Saito, Ames Research Center, Moffett Field, California

January 1987

National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, California 94035

CONTENTS

LIST OF SYMBOLS

AED	Automated Engineering Design. A high level programming language for flight software. Similar to Algol, and developed at MIT under USAF sponsorship.
ASCII	American Standard Code for Information Interchange.
assertion	a statement which will appear as a COMMENT statement to the AED language compiler, but to a tool, a logical expression or values to variables.
AVFS	automated verification of flight software. An integrated system for the verification of digital flight control software.
baud	bits per second
bpi	bits per inch
bps	bits per second
byte	eight bits
C	a general purpose programming language originally designed for and implemented on the UNIX™ operating system on a DEC PDP-11.
CAPS	Collins Adaptive Processing System
CAPS-6	Collins Adaptive Processing System model 6
CRT	cathode ray tube
CTA	CAPS Test Adapter
da11-b	an interprocessor link for half-duplex, parallel, direct memory access data transfer between two PDP-11 computers.
DD-path	decision to decision path
DEC	Digital Equipment Corporation
decimal	decimal, base 10 numbering sysytem
DFCS	digital-flight-control system
DFCSVL	Digital Flight Control System Verification Laboratory

download	the action of transferring a computer program or routine from a storage device to FCC memory through a communications link
EXEC 8	a Univac 1100 operating system
extr	extract program
FCC	flight-control computer
FORTRAN	Formula Translator, a scientific programming language
HASP	Houston Automatic Spooling Program. A collection of computer programs that provide two-way communications between a front end computer (PDP-11/60) and a main frame computer (Univac 1100) which serves as the host
hex	hexadecimal, base 16 numbering system
IFTRAN	a FORTRAN preprocessor developed by General Research Corp.
I/O	Input/Output
JCL	Job Control Language, sometimes called ECL, executive control language. An assembly like language that identifies the input stream to a host system
K	1024 decimal (from "kilo")
MDICU	Modular Digital Interface Control Unit
octal	base 8 numbering system
pif	pallet interface program (an abbreviation developed at NASA Ames Research Center)
RJE	remote job entry
UNIX	a trademark of Bell Laboratories. A licensed, general-purpose, interactive operating system capable of time sharing and of handling multiple users
upload	the action of transferring information or blocks of information stored in FCC memory to a storage device through a communications link
VCG	Verification Condition Generation, see Symbolic Execution.
V&V	verification and validation

SUMMARY

The user's guide of verification and validation (V&V) tools for the Automated-Engineering-Design (AED) language is specifically written to update the information found in several documents pertaining to the automated verification of flight software tools. The intent of this document is to provide, in one document, all the information necessary to adequately prepare a run to use the AED V&V tools. No attempt is made to discuss the FORTRAN V&V tools since they were not updated and are not currently active. Additionally, this document contains the current descriptions of the AED V&V tools and provides information to augment the NASA TM 84276 entitled "An Integrated User-Oriented Laboratory for Verification of Digital-Flight-Control Systems--Features and Capabilities."

The AED V&V tools are accessed from the digital-flight-control-systems verification laboratory (DFCSVL) via a PDP-11/60 digital computer. The AED V&V tool-interface handlers on the PDP-11/60 generate a Univac run stream which is transmitted to the Univac via a Remote Job Entry (RJE) link. Job execution takes place on the Univac 1100 and the job output is transmitted back to the DFCSVL and stored as a PDP-11/60 printfile.

INTRODUCTION

The increased use of digital processors in recent civil and military aircraft's flight control and management systems is forcing a reappraisal of the tools and techniques used for avionic-system design, development and test. A joint NASA/Federal Aviation Administration (FAA) program on the verification and validation (V&V) of digital-flight-control systems (DFCS) led to the establishment of a verification laboratory at NASA Ames Research Center. The laboratory includes an integrated, user oriented environment for tool development and analysis, an initial set of static and dynamic verification tools and a redundant DFCS for use as a test bed. The verification tools are designed to aid the control engineer and avionic system designer in the development and checkout of the flight-control system.

An in-house research project to analyze the effectiveness of the static tools was started and deficiencies in the tools were found. The tools were upgraded to correct these deficiencies before an independent analysis by industry was to be started under NASA contract.

This user's guide revises and upgrades part of a previous document, "Automated Verification of Flight Software--User's Manual" (ref. 1). It will, however, pertain only to the V&V tools specifically designed for the Automated-Engineering-Design (AED) Language, which is the language the digital-flight-control-system software uses in the DFCS verification laboratory (DFCSVL). Information and descriptions of the FORTRAN V&V tools were excluded from this document because they were not updated after delivery of the tools. Throughout this document the term "user" and "control engineer" will refer to the same person.

This guide augments the software description of the DFCSVL environment in the NASA TM-84276 entitled "An Integrated User-Oriented Laboratory for Verification of Digital Flight Control Systems--Features and Capabilities" (ref. 2), provides a description of the tool environment, consolidates the description of each tool, describes the PDP-11/60 commands necessary to get the tools into execution, and lists the constraints associated with the tools. Flow diagrams are provided to help clarify the tool's processing paths. Appendices A and B describe the tools' developmental environment and the rehosting requirements necessary for effective and successful hosting onto the target computer. Appendix C provides a description of the tool's interface handlers on the PDP-11/60.

DFCSVL OVERVIEW

The DFCSVL, as seen in figure 1, was established in 1981 at Ames Research Center to perform research experiments related to the DFCS. The Ames Research Center's studies in fault-tolerant and V&V software tools used a near-term DFCS system in the DFCSVL.

DFCSVL Environment

Figure 2 is a block diagram of the functions of the DFCSVL. The DFCSVL includes a PDP-11/60 digital computer (Environment Computer), a palletized DFCS, based on the Collins Adaptive Processing (CAPS) model 6)), a CRT terminal, and a remote computer (Software Tools Computer). The V&V tools are hosted on the Univac computer which is located at the Pacific Missile Test Center facility in Point Mugu, California (referred to as the Point Mugu facility) and accessed via a Remote Job Entry (RJE) communications link between the DFCSVL and Point Mugu.

Environment Computer- The "environment computer" is a PDP-11/60 digital computer with 256K words of memory, two disk drives with a storage capacity of 52 Mbytes, one 1600 bpi density tape drive, one 600-line/min line printer, and three CRT terminals connected to the system. For further details refer to section 5 of reference 2.

Remote Link- The Univac 1100 is connected to the PDP-11/60 through a RJE communications (dataphone) link or a standard dataphone link as seen in figure 3.

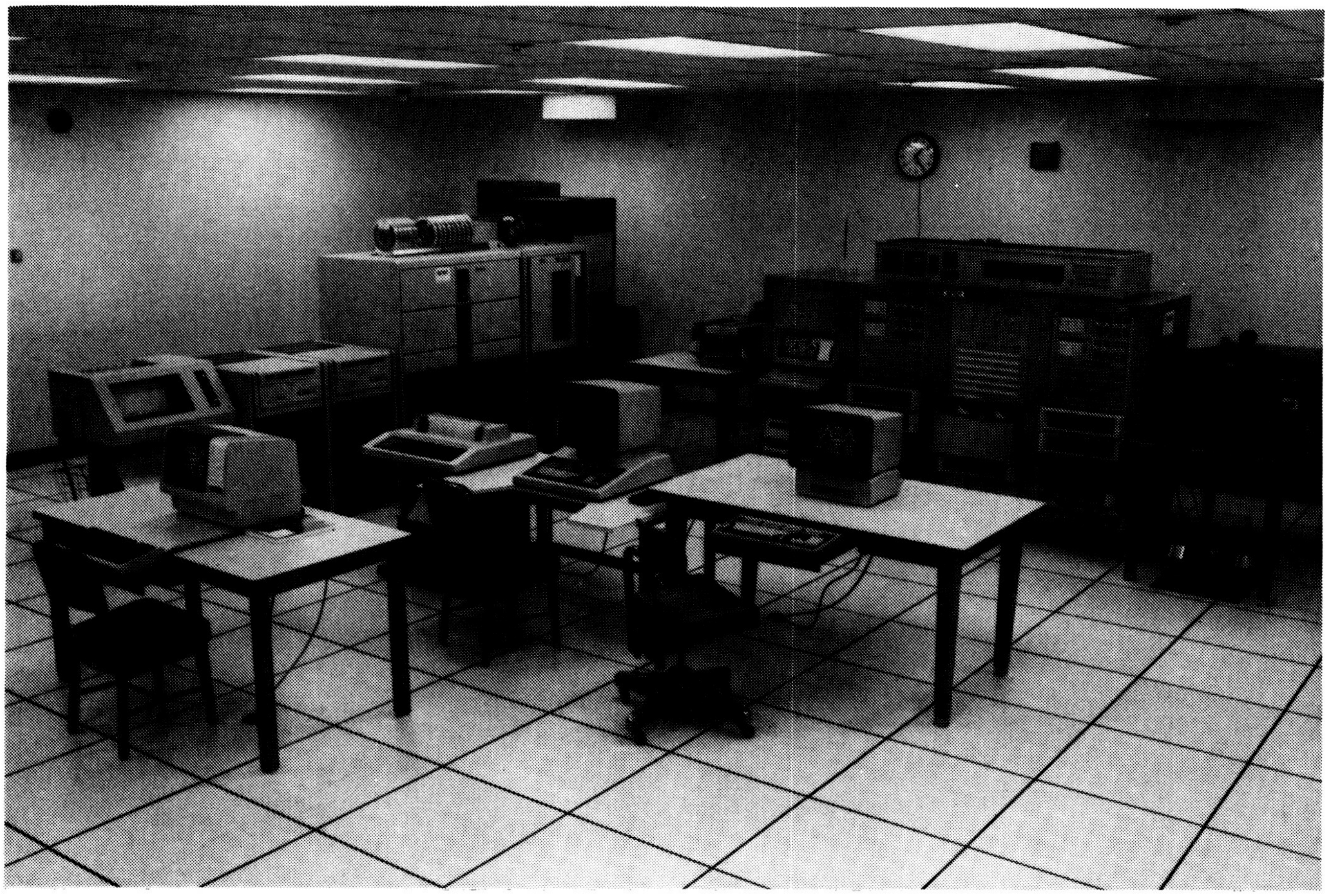

Figure 1.- The DFCSVL.

The RJE communications link uses a Bell model 208B-L1B data set operating at a transmission and receive rate of 4800 bits/sec (bps) in a "dial-up" mode and permits the PDP-11/60 to function as a RJE station. The RJE link currently uses the Houston-Automatic-Spooling program's (HASP) multileaving, nontransparent transmission mode at 400 bytes/block and all transmission over the link are ASCII code print files. Consequently, load-module files received from the Univac must be reconverted to a load-module executable code prior to any further use.

For communications as a demand terminal, the DFCSVL has a Racal-Vadic acoustic-coupler modem (VA/VC3412/13) capable of transmitting or receiving at either 300 or 1200 bps.

DFCSVL Software- The PDP-11/60 uses the UNIX™ operating system with special interface handlers written in C programming language to aid the control engineer in setting up the job stream for the Univac 1100. With simple UNIX commands, the control engineer is capable of selecting and exercising any V&V tool.

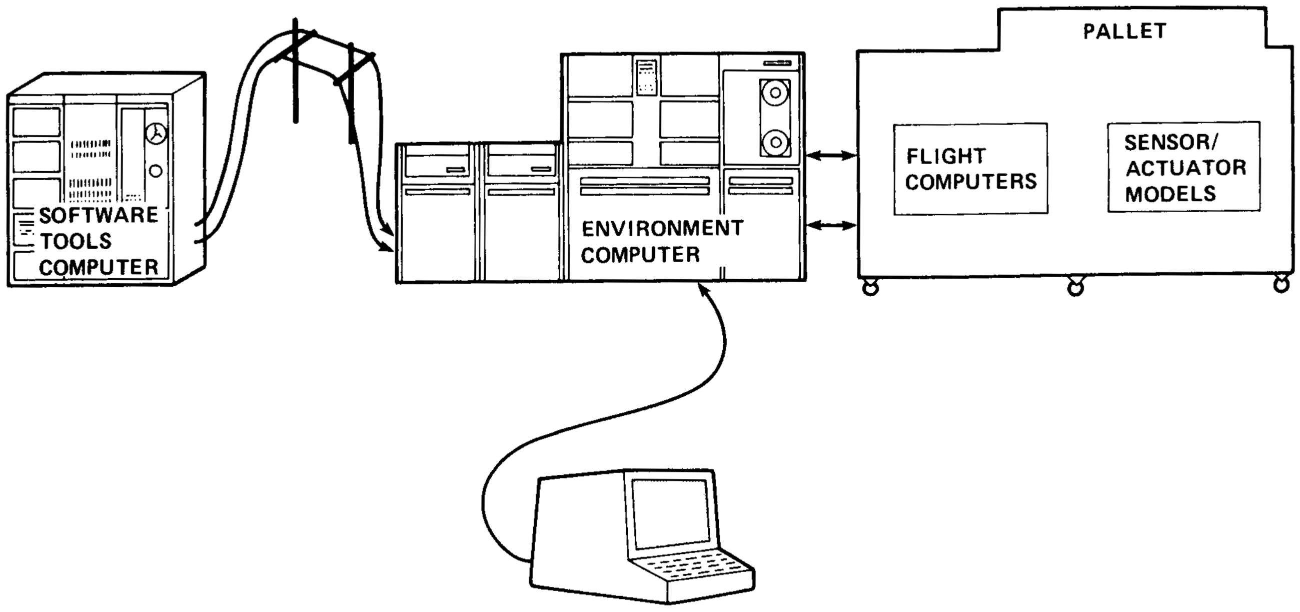

Figure 2.- Block diagram of the DFCSVL.

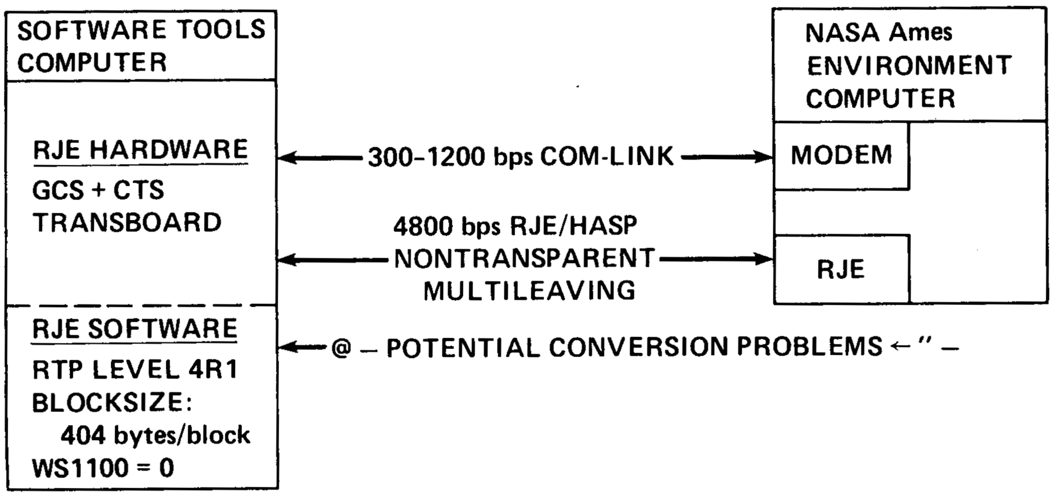

Figure 3.- RJE link.

Univac Environment

The Univac environment is off site and consists of a Univac 1100 operating under the EXEC 8 operating system with an RJE communications link running under HASP. The Univac hosts the AED processors as well as the V&V tools.

Requirements for the off-site Univac are:

1. Hardware
   a. Univac 1100/63 computer
   b. Tape units; 1600-bpi and 6250-bpi capabilities
   c. Disk units; mass-storage medium

2. Software
   a. EXEC 8 level 38R5A operating system
   b. AED processors; CAPS cross compiler
      CAPS cross assembler
      AED link editor
      tape transmission program

   (These processors execute under the EXEC 8 system and were originally developed and executed under a Univac 1100 using the EXEC 8 operating system.)
   c. Software V&V tools reside and execute on the Univac 1100

The system will receive and return batch jobs across the RJE link which originates at Ames Research Center. The system will also provide the hardware and computer time to execute AED and FORTRAN IV programs.

TESTING

The verification of the software for DFCS is usually the responsibility of control engineers and is based on the analysis of data obtained from testing the flight software in closed-loop, real-time simulations (fig. 4). Real-time testing is a well-established method which enables all type of errors to be detected, from specification to coding. However, real-time simulation testing is costly and cannot provide consistent and quantifiable test coverage.

AED V&V TOOL DESCRIPTIONS

An alternative to the real-time conventional testing method is for the user to check the software he has written by submitting one or more modules for analysis by the AED V&V tools so that errors are detected before they propagate through the entire flight software. The V&V tools also aid the user while creating the DFCS software code and documentating the flight software. The source code of the program is assumed to have been compiled and has no compiler detected errors when using these V&V tools. The V&V tools can only detect possible error conditions, but cannot correct them.

Figure 5 shows how the V&V tools augment the developmental cycle of software analysis and testing. The shaded blocks indicate the V&V tool features. The user's

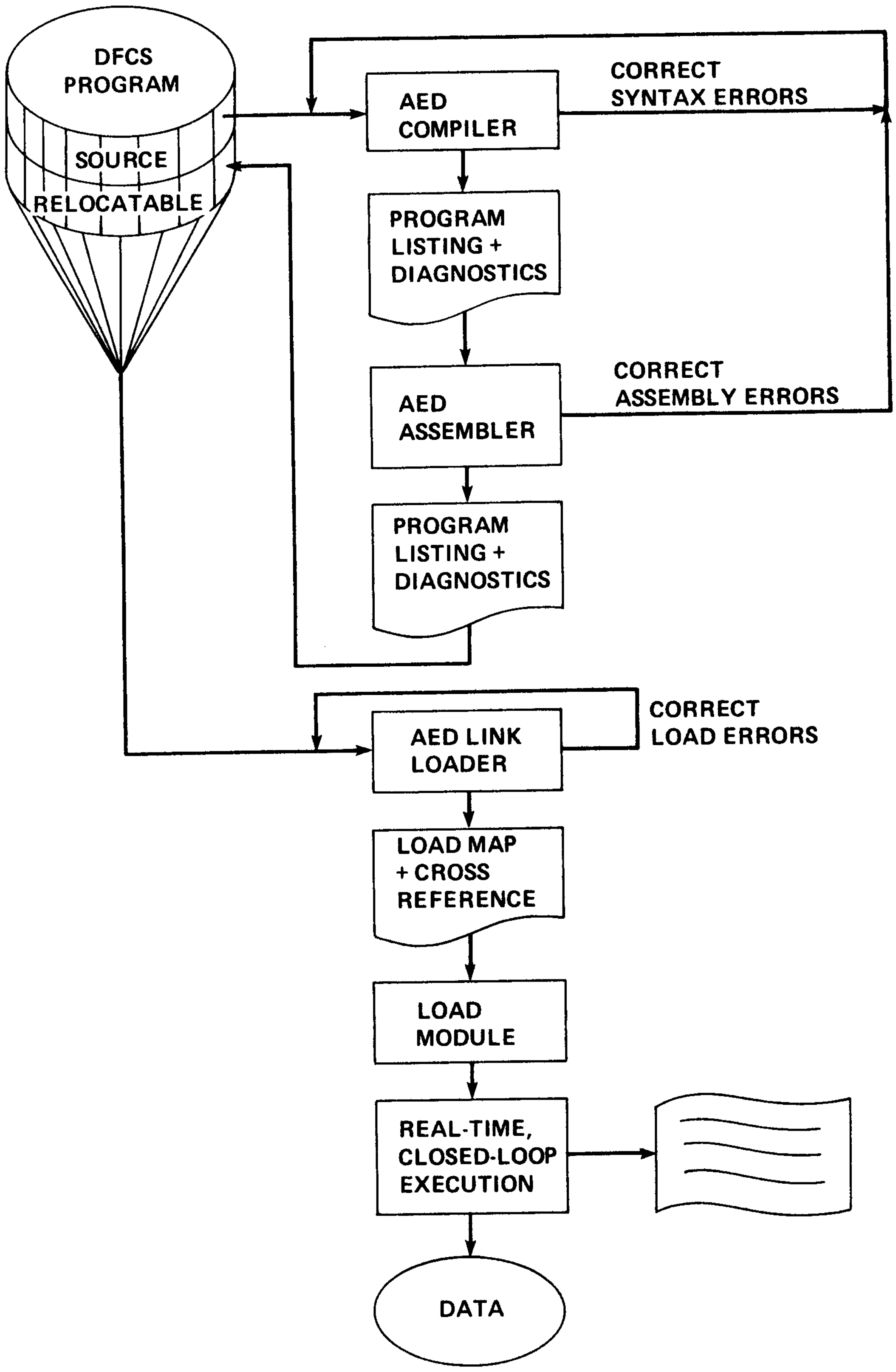

Figure 4.- Testing by real-time simulation.

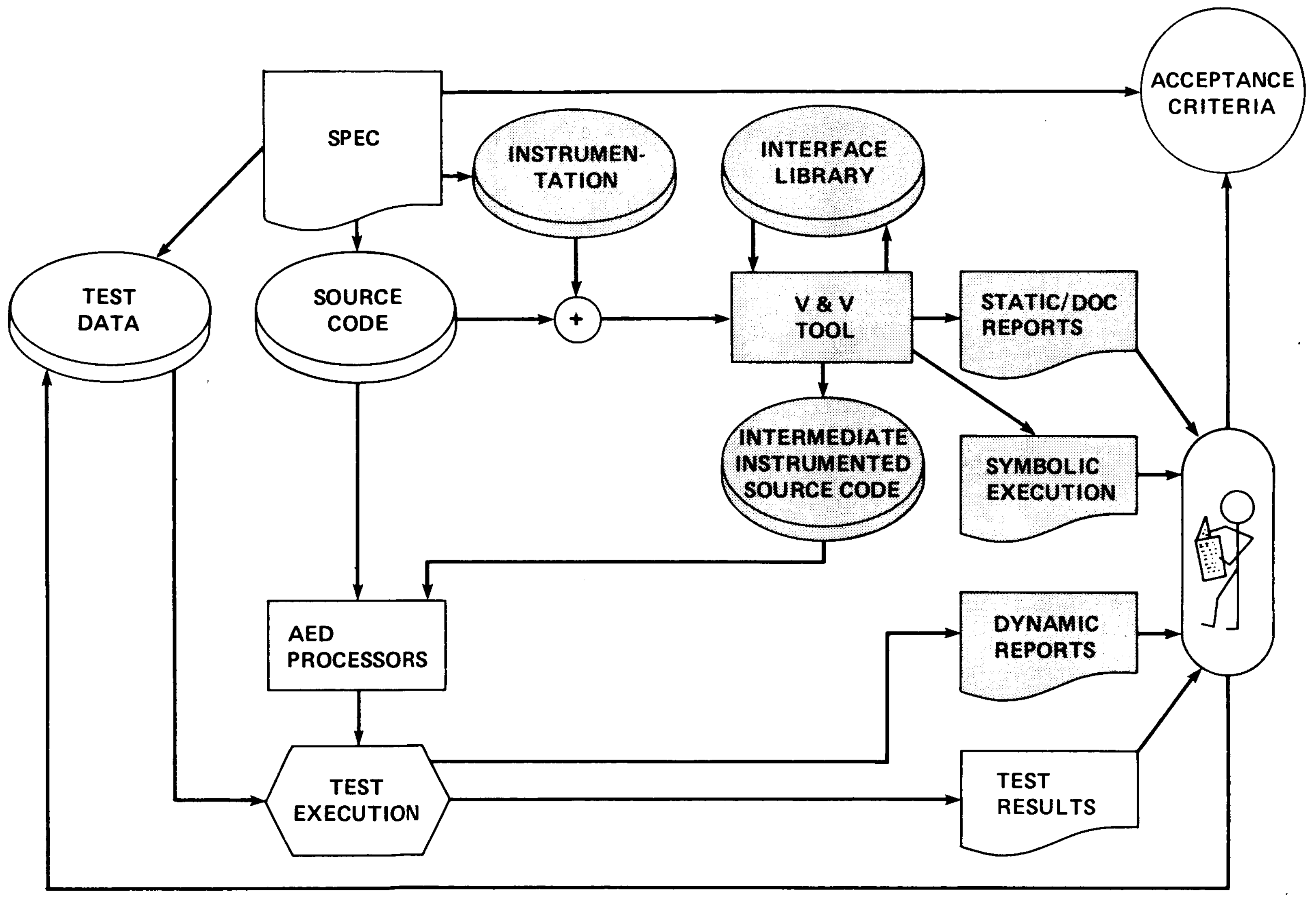

Figure 5.- Software verification augmented by V&V tools.

source code is analyzed by the static V&V tools to assist in finding inconsistencies in the use of the variables and in the structure of a program, and produces a static analysis or documentation report. Assertions, which are logical statements yielding either a true or false condition, can be added to the source text to further detect static or dynamic errors. An assertion statement is interpreted by the AED compiler as just another comment statement.

The AED dynamic tools automatically insert probes at appropriate points in a module to determine testing coverage, to insert assertions into a module to check on assertion violations, or to trace variables in the code. During dynamic test, these probes record data which are used to report execution coverage, assertion violations, execution time, and the values of important variables. The data recorded are stored in CAPS memory for later analysis by the report generator on the PDP-11/60. This information is used with other tests to indicate the focus of retesting which is discussed in more detail under "Symbolic Execution."

The tools are used to perform formal verification of formulas and assertions via symbolic execution. When a program is executed, numerical data is supplied as input. The procedure for formal verification is to execute the program "symbolically"; that is to use symbols as input data rather than specific numbers. The program is verified or "proven" correct for a wider range of its variables than is practical to assign during the execution test.

During the analysis by the static INTERFACE tool, an interface library is created and maintained as shown in figure 5. This interface-library file is the key to multiple module interface checks. The INTERFACE tool analyzes each module for changes in interface properties; e.g., addition or deletion of parameters, changes in the type or use of parameters, changes to COMMON, and changes to invocations.

The following sections contain more complete details of the full power of the AED V&V tools and how to use them.

Static Tools

Static Tools encompass the static analysis tools and the tools designed specifically for documentation purposes. The reason for this grouping is consistency with the UNIX command. When the tool is documentation type, the tool is identified as such.

Static analysis tools are designed to uncover inconsistencies in the use of variables and in the structure of the module. An inconsistency indicates the existence of an error or the possibility of an error.

Static analysis tools perform: set and use checking, loop checking, type checking, path checking, interface checking, and input/output (I/O) checking. Detailed descriptions and sample outputs are shown as each tool is discussed in this section. The general format presented here is of the UNIX command to exercise the static tools.

NAME

static - Static consistency checking/documentation

SYNOPSIS

static [-c class] [-mp#] [-mt#] [-dgilstuvxz] [-r filename.dat] AED_ file...(files must end in [.aed], [.fof] or [.isd])

DESCRIPTION

The static command invokes the V&V tool static analyzer on the Univac 1100. It accepts publicly readable files whose file-names end with .aed for AED source modules; .fof for file-of-files or .isd for files residing on the Univac computer site. Files ending in .aed or .fof may be sent at the same time. Files ending in .isd must all be sent together and must reside on the Univac 1100 computer. The module is analyzed and outputs are routed as specified by the following flag arguments.

Output listings from the Univac are routed to the user's directory on the PDP-11/60 with a file name consisting of the first four characters from the first file name with a ".rpt" extension attached. Each tool presents different types of output listings.

The following arguments are interpreted by static:

-c	Univac 1100 job priority, which has priority ranges from A, the highest priority, through Z, the lowest priority. The default is class A (standard).
-mp#	This option allows the user to specify the maximum number of pages for the Univac 1100 printout. The default number of pages used by this command is computed based on the number of static-tool options selected and the number of files (AED modules), approximately 16 pages for one option and one file; e.g., -mp50.
-mt#	This option allows the user to specify the maximum time for execution on the Univac 1100. A number preceded by an 's' (e.g., -mts40) is assumed to be in seconds. The default time limit used by this command is computed based on the number of static tool options and the number of files (AED modules), approximately 60 sec for one option and one file.
-dgilstuvxr	Each option selects an associated tool. The description of each tool for the type of report it generates is presented in detail in the respective sections on options.

-z DISPLAY/DEBUG option, the -z option, allows for displaying or debugging the run stream created by the option selected or defaulted. In this example, the default option -l is invoked as well as the display/debug option. The run stream is printed or typed to the terminal in which the UNIX command was given.

NOTE

Source files manipulated by this command must be publicly readable in order for them to be copied to the RJE queue or accessible on the Univac.

DIAGNOSTICS

The job may successfully be sent to the Univac 1100, but fail to run for many reasons. The user's output listing will contain one or more cryptic messages why the run failed. If it is not obvious, seek a user consultant for diagnostic help.

DISPLAY/DEBUG EXAMPLE:

```
static -z file.aed
```

This command displays the Univac run stream for file "file.aed" on the input terminal where the UNIX command was given as seen below.

```
@RUN Univac run card
@ASG,AX INSERTS.
@FREE AVFS$$.
@ASG,T AVFS$$.
@ELT,I AVFS$$.FILE

    THIS IS THE BEGINNING OF A DUMMY AED SOURCE FILE
                            :
            AED STATEMENTS USUALLY FOLLOW HERE
                            :
                            :
    THIS IS THE END OF THE DUMMY AED SOURCE FILE
@HDG,X *** AED ENHANCED LISTING, MODULE HDGSEL ***
@XQT GRC*LIST.LIST
@ADD,E AVFS$$.FILE
@HDG,X *** UPDATED AED STATEMENT PROFILE ***
@XQT AMES*PROFILE.PROFILE
@ADD,E AVFS$$.FILE
@FIN
```

-d option: MODULE DEPENDENCIES.-

UNIX COMMAND EXAMPLE:

```
static -d alatcom.aed
```

This command generates a module-dependency report for the AED module alatcom.

UNIVAC COMMAND EXAMPLE:

```
@XQT AMES*DEPEND.DEPEND
@ADD,E AVFS$$.filename
```

The -d option generates the module-dependencies documentation report. This option is sometimes called "invocation summary" report in the documents listed in reference 1. The dependency report shows the dependencies of the modules on the interface library. All modules are listed which invoke a module and all invocations in a module. Figure 6 shows a dependency report for the module ALATCOM. The statement line number of where invocations to a given module occur is found in the invocation report or the global cross-reference report.

```
*** UPDATED AED DEPENDENCE MATRIX ***

MODULE DEPENDENCE REPORT                MODULE  A.LAT.COM

PROCEDURE             DEPENDENCY                        PAGE 1
-------------------------------------------------------------------

A.LAT.COM             IS INVOKE BY    -NONE
                      AND INVOKES     -NONE
-------------------------------------------------------------------
```

Figure 6. Dependency report.

-g option: GLOBAL CROSS REFERENCE-

UNIX COMMAND EXAMPLE:

```
static -g alatcom.aed
```

This command generates a global cross-reference report for the AED module alatcom.

UNIVAC COMMAND EXAMPLE:

```
@XQT AMES*GBLXREF.INITIAL
@XQT AMES*GBLXREF.GLOBAL
@ADD,E AVFS$$.filename
@XQT AMES*GBLXREF.FINAL
```

The -g option generates a global cross-reference documentation report. The global cross-reference report is a multimodule report showing the 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 next column denotes the scope of the variable, EXTERNAL or COMMON. The modules which reference the global variable are alphabetically ordered in the module column. The last column contains the statement line numbers within each module where the variable is referenced. The example shown in figure 7 is a global cross-reference of a single module called ALATCOM.

```
*** UPDATED AED GLOBAL CROSS REFERENCE ***

GLOBAL CROSS REFERENCE         MULTIMODULE REPORT                PAGE   1

NAME             SCOPE      MODULE      USED/SET ( - INDICATES SET )
-------------------------------------------------------------------------

A.LAT.COM        EXTERNAL   A.LAT.COM    16
AL.TRK.M         EXTERNAL   A.LAT.COM    27  43  43
ARCTAN           EXTERNAL   A.LAT.COM    22  42
DISAGREE.O       EXTERNAL   A.LAT.COM    42
DLIMIT           EXTERNAL   A.LAT.COM    30  42
FLP.GT.30        EXTERNAL   A.LAT.COM   -20  43  43
FLP.GT.4         EXTERNAL   A.LAT.COM   -21  43  43
HEADING          EXTERNAL   A.LAT.COM   -22  23  43  43
IGNORE.OVRF      EXTERNAL   A.LAT.COM    42
KTAS             EXTERNAL   A.LAT.COM   -35  43  43
LAT.ACC.LP       EXTERNAL   A.LAT.COM    28 -30  30  33  43  43
LAT.ACC.PTR      EXTERNAL   A.LAT.COM    24  43  43
LAT.ACC.VLP      EXTERNAL   A.LAT.COM   -33  43  43
MAJORITY.O       EXTERNAL   A.LAT.COM    20  21  42
PROTECT.OVRF     EXTERNAL   A.LAT.COM    42
ROLL.PTR         EXTERNAL   A.LAT.COM    25  43  43
VOTER            EXTERNAL                14  24  25
XOR              EXTERNAL   A.LAT.COM    42
-------------------------------------------------------------------------
```

Figure 7.- Global cross-reference report.

-i option: INTERFACE-

UNIX COMMAND EXAMPLE:

```
static -i aforexec.aed
```

This command generates an interface analysis report for the AED module aforexec.

UNIVAC COMMAND EXAMPLE:

```
@XQT AMES*INTER.INTER
@ADD,E AVFS$$.filename
```

The -i option generates an interface-library report. The AED interface analysis tool uses a library file on the Univac to check consistency in number of parameters and type of the module. It also checks for changes to the interface via the library. Figure 8 shows the interface processing flow on the Univac.

Figures 9 and 10 depict the interface report outputs. In figure 9, the interface report gives the module names and procedures in the column identifed as "SYMBOL NAME," then the type of change in the next column, and additional information in the last column. In figure 10, the second part of the Interface report presents the interface analysis entitled "PROCEDURE CALL CHECKING REPORT." The first column identifies the name of the procedure called. The second column identifies the line number where the procedure call was made. The last column identifies the error description from the analysis of the module.

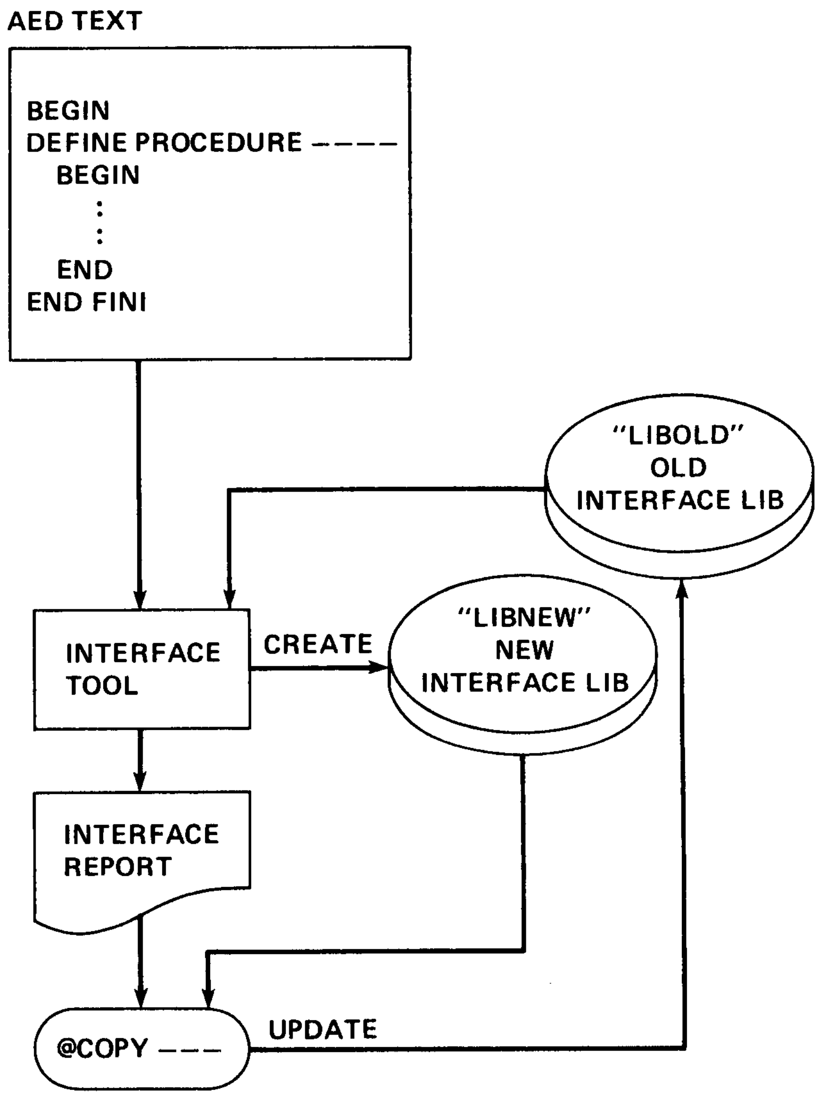

Figure 8.- Interface-processing flow.

*** UPDATED AED INTERFACE REPORT ***

I N T E R F A C E   R E P O R T

SYMBOL NAME	TYPE OF CHANGE	OTHER
A.FORE.EXEC	NEW MODULE	1 ENTRY POINT(S) NO BLANK COMMON
A.FORE.EXEC	NEW PROCEDURE	0 FORMAL PARAM(S)

Figure 9.- Interface report.

```
               P R O C E D U R E   C A L L   C H E C K I N G   R E P O R T
____________________________________________________________________________

NAME OF CALLED PROCEDURE      LINE   MESSAGE
____________________________________________________________________________

MAJORITY.O                      33   NOT FOUND ON INTERFACE LIBRARY
   :                             :                :
   :                             :                :
SPEED.COM                       47   NOT FOUND ON INTERFACE LIBRARY

ASNBIT                          50   UNABLE TO VERIFY CORRECT DATA TYPE
                                     OF THE CONSTANT IN POSITION  1

ASNBIT                          50   UNABLE TO VERIFY CORRECT DATA TYPE
                                     OF THE CONSTANT IN POSITION  2

SENS.MON                        54   NOT FOUND ON INTERFACE LIBRARY
   :                             :                :
   :                             :                :
B.ALT.XMIT                      99   NOT FOUND ON INTERFACE LIBRARY

A.TIME.SYNC                    100   NOT FOUND ON INTERFACE LIBRARY

                            SUMMARY OF CALL CHECKING
              ______________________________________________

              TOTAL PROCEDURE CALLS PROCESSED            =  41
              PARAMETER LIST LENGTH ERRORS               =   0
              PARAMETER TYPE CLASH  ERRORS               =   0
              CALLS IN WHICH NOT ALL PARAMETERS VERIFIED =  14
              CALLS FOR WHICH NO CHECKING WAS PERFORMED  =   0
              CALLS TO PROCEDURES NOT ON INTERFACE LIB   =  31
              ______________________________________________
____________________________________________________________________________
```

Figure 10.- Procedure call checking report.

-1 option: ENHANCED LISTING AND STATEMENT PROFILE-

UNIX COMMAND EXAMPLE:

```
static -l alatcom.aed
```

This command generates an enhanced listing of the AED source code and an AED statement profile for module alatcom.aed.

UNIVAC COMMAND EXAMPLE:

```
@XQT GRC*LIST.LIST
@ADD,E AVFS$$.filename
@XQT AMES*PROFILE.PROFILE
@ADD,E AVFS$$.filename
```

The -l option generates an enhanced listing, sometimes called indented listing or pretty print, and statement profile, or profile, report. This is the DEFAULT option for the static command.

No changes were made to the enhanced listing tool. The enhanced listing is a source listing which shows the statement line number and the automatically indented source code. All references to statements in other reports by the V&V tools are keyed to statement line numbers and module name. The indented listing clearly indicates the control structures and improves readability, not only to the original programmer, but especially to someone unfamiliar with the code. Figure 11 illustrates a sample listing for an AED module.

```
*** AED ENHANCED LISTING, MODULE [name] ***

16 DEFINE PROCEDURE A.LAT.COM TOBE
17
18 COMMENT ITERATION RATE = 5 / SEC ;
19      BEGIN
20      FLP.GT.30 = MAJORITY.O(10) ... MAJORITY OF 30 DEG FLAP SWITCHES ;
21      FLP.GT.4 = MAJORITY.O(11) ... MAJORITY OF 4 DEG FLAP SWITCHES ;
22      HEADING = ARCTAN(HDG.SIN,HDG.COS) ... GET HEADING ;
23      CD.13 = HEADING           ... TRANSMIT TO OTHER CHANNELS ;
24      LAT.ACC = VOTER(LAT.ACC.PTR) ... VOTE LATERAL ACCELERATION ;
25      ROLL = VOTER(ROLL.PTR)    ... VOTE BANK ANGLE ;
26      X1 =
27      IF AL.TRK.M OR (ABS(ROLL) < .026178) ... ROLL < 3 DEG //
28      THEN LAT.ACC*.858667+ROLL*.429368-LAT.ACC.LP
29      ELSE .0                   ... HOLD BY GROUNDING SWITCH ;
30      LAT.ACC.LP = DLIMIT(LAT.ACC.LP+X1*.166667D-2,.133333D-1) ...
31                                    WASHOUT INTEGRATOR LIMITED AT 2 DEG /
32                                    SEC ;
33      LAT.ACC.VLP = LAT.ACC.LP/.858667;
34      TAS = TAS.MS              ... BUFFER TAS ;
35      KTAS =                    ... GAIN PROGRAMER //
36      IF TAS > .439453          ... TAS>450 KTS //
37      THEN .500000
38      ELSE IF TAS > .146484     ... TAS>150 KTS //
39           THEN TAS/.878906
40           ELSE .166667;
41      CD.19 = SEL.HDG;
```

Figure 11. Enhanced listing.

The AED statement-profile report lists the name of the module and the number of lines at the top of the report, as shown in figure 12. The module name is the name of the procedure contained in the module. The number of lines includes the lines from any inserts and blank lines.

```
*** UPDATED AED STATEMENT PROFILE ***

        STATEMENT PROFILE
MODULE A.LAT.COM
NUMBER OF LINES   103

                    NUMBER    PERCENTAGE OF LINES
__________________________________________________

DECLARATIONS          46          44.7
    ARRAY              1           1.0
    BEAD               0            .0
    COMMON             0            .0
    COMPONENT          0            .0
    DEFINE             1           1.0
    EXTERNAL          10           9.7
    INSERT             3           2.9
    PACK               0            .0
    PRESET             0            .0
    PROCEDURE          8           7.8
    SWITCH             0            .0
    SYNONYM            1           1.0
    VARIABLE          25          24.3
STATEMENTS            32          31.1
    ASSIGNMENT        12          11.7
    COMMENT           18          17.5
    COMPOUND           1           1.0
    IF                 0            .0
    FOR                0            .0
    GOTO               0            .0
    PROCEDURE          0            .0
    WHILE              0            .0
    ASSERT             0            .0
__________________________________________________
```

Figure 12.- Statement profile report.

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, common, 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 top of the profile report.

Under statements, there are statements for assignment, compound, if, for, goto, procedure, and while categories. A line can contain several categories. For example, a statement consisting of a BEGIN...END construct is counted as a compound statement. Another example of a statement containing several categories is the IF statement. It usually contains a BEGIN...END compound statement as well as a procedure invocation. Comments and assertions categories are listed under statements and are counted separately.

-s option: SYMBOLS (SET/USE).-

UNIX COMMAND EXAMPLE:

```
static -s alatcom.aed
```

This command generates a SET/USE listing for module alatcom.aed.

UNIVAC COMMAND EXAMPLE:

```
@XQT AMES*SETUSE.SETUSE
@ADD,E AVFS$$.filename
```

The -s option generates a symbols report, also called SET/USE. The static analysis of an AED source module for SET/USE checking analyzes the variables used before they are set to a value, or set and not used. When I/O assertions are added to the source module, a complete static analysis check is made with the SET/USE tool.

Input/Output assertions are used in a static analysis to check for consistency between the intended use of a variable and the actual use of a variable. Variables which provide input data to a module should be asserted with an input assertion.

An input assertion has the form:

```
COMMENT INPUT* <type> <variable>;
```

An example of an input assertion is

```
COMMENT INPUT* REAL HEIGHT;
```

where the variable named HEIGHT is an input to the module.

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

```
COMMENT OUTPUT* <type> <variable>;
```

Variables which are used both as input and as output should have both assertions.

The SET/USE report is generated for each module analyzed. The symbols are ordered alphabetically, scoped, classed, and their use analyzed. Symbols which have the scope LOCAL are known only within the module reported on. Other symbols with EXTERNAL or COMMON classification are known outside the module. Each symbol is organized into its class: variable, array, procedure, and type. The use column provides a summary of how the symbol is used in the module. Figure 13 shows an AED SET/USE Report.

-t option: CALLING TREE-

UNIX COMMAND EXAMPLE:

```
static -t pitchdis.aed
```

This command generates the calling-tree report for module pitchdis.aed.

UNIVAC COMMAND EXAMPLE:

```
@XQT AMES*CTREE.CTREE
@ADD,E AVFS$$.filename
```

The -t option generates the calling-tree documentation report, also called invocation bands. This report shows the selected module in a calling tree, as shown in figure 14. At the center is the specified module. The left-side modules are the calling modules. The right-side modules are the called modules. A summary of this report is found in the invocation-summary report. The statement numbers containing invocations are found in the global cross-reference report.

A word of caution for those who will not be using the DFCSVL-type environment. To process multimodules by the calling-tree tool, the modules must be concatenated. To accomplish this task, successive Univac commands "@ADD filename" must be used after the @XQT AMES*CTREE.CTREE.

```
*** UPDATED AED SETUSE REPORT ***

SET/USE ANALYSIS AND PARAMETER REPORT   MODULE  A.LAT.COM         PAGE    1

                                  1ST LAST TOTL   ASSERTED    ACTUAL
NAME            SCOPE     CLASS        STMT STMT USES      USE         USE
-------------------------------------------------------------------------

A.LAT.COM       EXTERNAL  PROCEDURE    16   16    1
AL.TRK.M        EXTERNAL  VARIABLE     27   43    3      NONE        INPUT
ARCTAN          EXTERNAL  PROCEDURE    22   42    2
CD.13           LOCAL     VARIABLE     23   43    3
DISAGREE.O      EXTERNAL  PROCEDURE    42   42    1
DISCRETE.IN     LOCAL     SYNONYM      42   42    1
DISCRETE.OUT    LOCAL     SYNONYM      42   42    1
DLIMIT          EXTERNAL  PROCEDURE    30   42    2
FLP.GT.30       EXTERNAL  VARIABLE     20   43    3      NONE        OUTPUT
FLP.GT.4        EXTERNAL  VARIABLE     21   43    3      NONE        OUTPUT
HDG.COS         LOCAL     VARIABLE     22   43    3
-------------------------------------------------------------------------

-                                 SET/USE ERROR                          -
-   VARIABLE HDG.COS              USED BEFORE BEING ASSIGNED A VALUE     -
-------------------------------------------------------------------------

HDG.SIN         LOCAL     VARIABLE     22   43    3
-------------------------------------------------------------------------

-                                 SET/USE ERROR                          -
-   VARIABLE HDG.SIN              USED BEFORE BEING ASSIGNED A VALUE     -
-------------------------------------------------------------------------

HEADING         EXTERNAL  VARIABLE     22   43    4      NONE        OUTPUT
INST.ARRAY      EXTERNAL  ARRAY        42   42    2      NONE        NONE
KTAS            EXTERNAL  VARIABLE     35   43    3      NONE        OUTPUT
LAT.ACC         LOCAL     VARIABLE     24   43    3
LAT.ACC.LP      EXTERNAL  VARIABLE     28   43    6      NONE        BOTH
LAT.ACC.PTR     EXTERNAL  VARIABLE     24   43    3      NONE        INPUT
LAT.ACC.VLP     EXTERNAL  VARIABLE     33   43    3      NONE        OUTPUT
ROLL            LOCAL     VARIABLE     25   43    4
ROLL.PTR        EXTERNAL  VARIABLE     25   43    3      NONE        INPUT
SEL.HDG         LOCAL     VARIABLE     41   43    3
-------------------------------------------------------------------------

-                                 SET/USE ERROR                          -
-   VARIABLE SEL.HDG              USED BEFORE BEING ASSIGNED A VALUE     -
-------------------------------------------------------------------------
```

Figure 13.- SET/USE report.

```
                                1ST LAST TOTL    ASSERTED     ACTUAL
NAME          SCOPE     CLASS         STMT STMT USES          USE         USE
_____________________________________________________________________________

TAS.MS        LOCAL     VARIABLE    34   43    3
_____________________________________________________________________________

-                                 SET/USE ERROR                            -
-   VARIABLE TAS.MS               USED BEFORE BEING ASSIGNED A VALUE -
_____________________________________________________________________________

VOTER         EXTERNAL  PROCEDURE   14   25    3
X1            LOCAL     VARIABLE    26   43    3
_____________________________________________________________________________
```

Figure 13.- Concluded.

```
*** UPDATED AED CALLING TREE ***

CALLING TREE                                     MODULE PITCH.DIS

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

                                                               PITCH.DIS
                                                    A.FORE.EXEC         ASNBIT
                                                                     ARINC.C.
                                                                     NAV.DIS.VAL
                                                                     PITCH.NUMS
________________________________________________________________________________
```

Figure 14.- Calling-tree report.

-u option: UNITS-

UNIX COMMAND EXAMPLE:

```
static -u voter.aed
```

This command generates the units analysis report for module voter.aed.

UNIVAC COMMAND EXAMPLE:

```
@XQT AMES*UNITS.UNITS
@ADD,E AVFS$$.filename
```

The -u option generates the units analysis report. Units assertions are inserted into a program so consistency checks are made on the use of units. Each variable for which units are specified has units declared in the form

```
COMMENT UNITS* <variable> = <units expression>;
```

For example, to state that the variable named SPEED has units of FEET/SEC, and TIME has units of SEC, type in

```
COMMENT UNITS* SPEED = FEET/SEC;
COMMENT UNITS* TIME = SEC;
```

To state that the variable named DIST has units of METER, type in

```
COMMENT UNITS* DIST = METER;
```

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

```
SPEED = DIST * TIME;
```

the units analyzer would report a units error of the form:

```
*****UNITS ERROR*****
FEET/SEC=METER*SEC
```

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. Figure 15 shows the flow diagram for processing the units tool. Currently the tool is incapable of handling embedded operations such as an argument in the calling list of a procedure, or embedded by parentheses.

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

Figure 16 shows units specified for speed, distance, time, acceleration, force, and work. The error messages are displayed in the source code listing when inconsistent units occur during analysis. The units table summarizing the units assertions, follows the source-code listing.

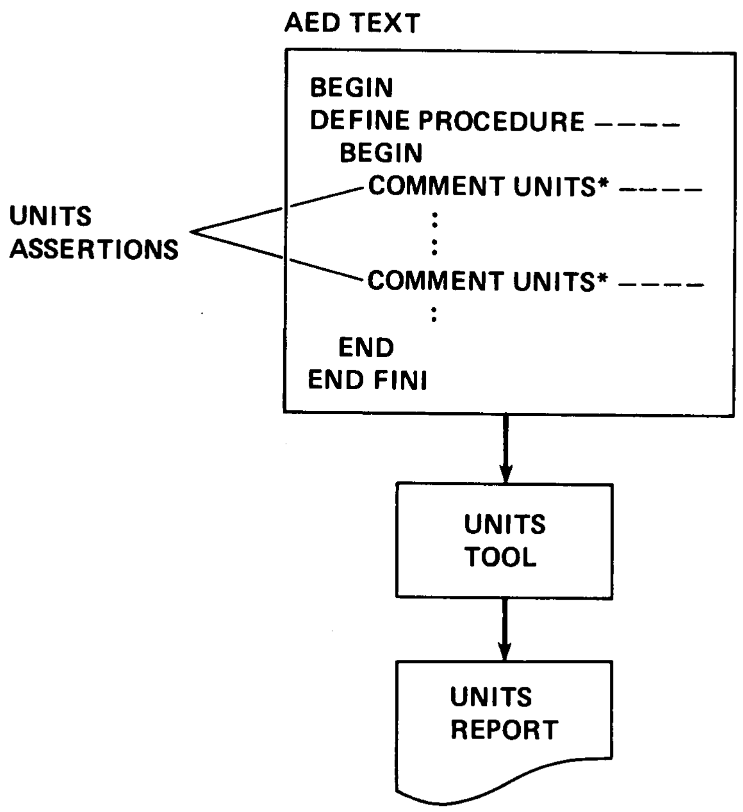

Figure 15.- Units-processing flow.

```
*** UPDATED AED UNITS ANALYSIS ***

:            :
:            :
:            :
39      DEFINE PROCEDURE LOAD.ARRAYS(L,M,A) WHERE INPUT.INTEGER L,M;
40      INTEGER ARRAY A TOBE
41           BEGIN
42 COMMENT ***** TOOL 12 TEST ************************ ;
43 COMMENT UNITS* SPEED = FEET/SEC ;
44 COMMENT UNITS* DIST = METER ;
45 COMMENT UNITS* TIME = SEC ;
46 COMMENT UNITS* ACCEL = FEET/(SEC*SEC) ;
47 COMMENT UNITS* FORCE = POUND ;
48 COMMENT UNITS* WORK = FEET*POUND ;
49
50         SPEED = DIST * TIME ;
*****UNITS ERROR*****
FEET/SEC=METER*SEC
51         WORK = FORCE * DIST ;
*****UNITS ERROR*****
FEET*POUND=METER*POUND
52         FORCE = WORK * DIST ;
*****UNITS ERROR*****
POUND=FEET*METER*POUND
53         ACCEL = SPEED / TIME ;
*****UNITS ERROR*****
FEET/SEC*SEC=FEET/SEC/SEC
54         DIST = SPEED * TIME ;
*****UNITS ERROR*****
METER=FEET
55
56 COMMENT ***** END UNITS TOOL TEST ************************ ;
57
:            :
:            :
:            :

*** UPDATED AED UNITS ANALYSIS ***

          UNITS TABLE
SPEED                         FEET/SEC
DIST                          METER
TIME                          SEC
ACCEL                         FEET/SEC*SEC
FORCE                         POUND
WORK                          FEET*POUND
```

Figure 16.- Units report.

-v option: INVOCATIONS-

UNIX COMMAND EXAMPLE:

```
static -v aforeinit.aed
```

This command generates the INVOCATION-documentation report for module aforeinit.aed

UNIVAC COMMAND EXAMPLE:

```
@XQT AMES*INVOKE.INVOKE
@ADD,E AVFS$$.filename
```

The -v option generates the INVOCATION-documentation report. The report shows all invocations, along with the statement-line numbers, to and from the specified module. This report is useful for examining actual parameter usage. Figure 17 is an example of the INVOCATIONS Report.

```
*** UPDATED AED INVOCATIONS REPORT ***

 INVOCATIONS REPORT                          MODULE A.FORE.INIT     PAGE    1
____________________________________________________________________________

INVOCATIONS FROM WITHIN THIS MODULE
____________________________________________________________________________

PROCEDURE A.FORE.INIT
STMT   20         CLEAR.FAIL();
____________________________________________________________________________

INVOCATIONS TO THIS MODULE
____________________________________________________________________________

PROCEDURE A.FORE.INIT
                -IS NOT CALLED
____________________________________________________________________________
```

Figure 17.- Invocations report.

-x option: CROSS REFERENCE.-

UNIX COMMAND EXAMPLE:

```
static -x altitude.aed
```

This command generates the cross-reference documentation report for module altitude.aed

UNIVAC COMMAND EXAMPLE:

```
@XQT AMES*XREF.XREF
@ADD,E AVFS$$.filename
```

The -x option generates the cross-reference documentation report (fig. 18). This report provides a "symbol" to cross reference each module analyzed (fig. 18). For this guide the definition of symbol means an AED variable. 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). The CLASS column identifies the symbol as a variable, a procedure, or a synonym. The TYPE column identifies the symbol type as a long, real, boolean, integer, or pointer. The last four columns identify every occurrence of a symbol by line number and if the symbol was used or referenced, set, or defined in a particular line.

-r option: REACHING SET-

UNIX COMMAND EXAMPLE:

```
static -r ctrl.dat aforexec.aed
```

This command generates the REACHING SET analysis report for module aforexec.aed within the range specified in "ctrl.dat".

UNIVAC COMMAND EXAMPLE:

```
@XQT AMES*REACH.REACH
@ADD,E AVFS$$.filename
```

The -r option generates the REACHING-SET analysis report. The analysis specified by the REACHING-SET option, executes the module retesting capability of the V&V tools. When a set of untested decision-to-decision paths (DD-paths) has been isolated, the V&V tools help the user identify the sections of code for further testing. To reach the desired DD-path number, the user specifies the beginning statement-line number and the ending-statement line number bounding the DD-path

```
*** UPDATED AED SINGLE MODULE CROSS REFERENCE ***

CROSS REFERENCE                 MODULE ALTITUDE                  PAGE    1

NAME          SCOPE    CLASS      TYPE     USED OR REFERENCED /SET(S)/DEFINED(D)
-------------------------------------------------------------------------------

ALT.BIAS      LOCAL    VARIABLE   LONG       82S   83    85    135D
ALT.CAP.COND  LOCAL    VARIABLE   BOOLEAN    57S   61    135D
ALT.CAP.M     EXTERNAL VARIABLE   BOOLEAN    45    65    79    132   135   135D
ALT.CAP.M.S   LOCAL    VARIABLE   BOOLEAN   114   121    132S  135D
ALT.ERR       LOCAL    VARIABLE   LONG       73S  110S   110   111   135D
ALT.HLD.COND  LOCAL    VARIABLE   BOOLEAN    55S   59    71    135D
ALT.HLD.M     EXTERNAL VARIABLE   BOOLEAN    30    70    107   135   135D
ALT.NORM.SEL  LOCAL    VARIABLE   BOOLEAN    40S   42    135D
ALT.RATE      LOCAL    VARIABLE   LONG       54S   57    83    135D
ALT.REF       LOCAL    VARIABLE   LONG       66S   72S   73    135D
ALT.SEL.M     EXTERNAL VARIABLE   BOOLEAN   135   135D
ALTITUDE      EXTERNAL PROCEDURE  -          21D
ANALOG.IN     LOCAL    SYNONYM    -         134D
ANALOG.OUT    LOCAL    SYNONYM    -         134D
    :           :         :       :          :
    :           :         :       :          :
    :           :         :       :          :
SEL.ALT       LOCAL    VARIABLE   REAL       50   135D   135D
SEL.ALT.ERR   LOCAL    VARIABLE   LONG       52S   55    57    66    82    83
TM.M          EXTERNAL VARIABLE   BOOLEAN   109   135    135D
VERSINE       EXTERNAL VARIABLE   REAL      135   135D
VOTER         EXTERNAL PROCEDURE  REAL       19    27    62
X1.D          LOCAL    VARIABLE   REAL       29S   33    34    135D
X1.D.S        LOCAL    VARIABLE   REAL       33    34S   135D
X2.D          LOCAL    VARIABLE   LONG       33S   33    54    111   135D
X3.D          LOCAL    VARIABLE   LONG       87S   91    95    97    113S  117
X3.D.S        LOCAL    VARIABLE   LONG       91S   95    97S   117S  123   125S
X4.D          LOCAL    VARIABLE   LONG       90S   95S   95    98    116S  122S
                                            135D
X5.D          LOCAL    VARIABLE   LONG       92S   99S   99    102   118S  127S
X6            LOCAL    VARIABLE   REAL       73S   76    135D
X7            LOCAL    VARIABLE   REAL       28S   29    74    75    135D
X7.S          LOCAL    VARIABLE   REAL       74    75S   135D
X8.D          LOCAL    VARIABLE   LONG       74S   74    85    135D
X9.D          LOCAL    VARIABLE   LONG       85S   87    135D
XOR           EXTERNAL PROCEDURE  BOOLEAN   134D
-------------------------------------------------------------------------------
```

Figure 18.- Cross-reference report.

number in a file with a unique file-name terminator "----.dat", such as "ctrl.dat" as just shown. Once specified, the REACHING-SET tool generates the reaching-set report of paths from the specified beginning statement line number to the ending-statement line numbers. The flow diagram for processing the reaching set tool is seen in figure 19. A reaching-set listing (fig. 20) shows the DD-paths within the reaching set in the AED source code. In figure 21, the reaching-set report summarizes the statements line numbers within the reaching set.

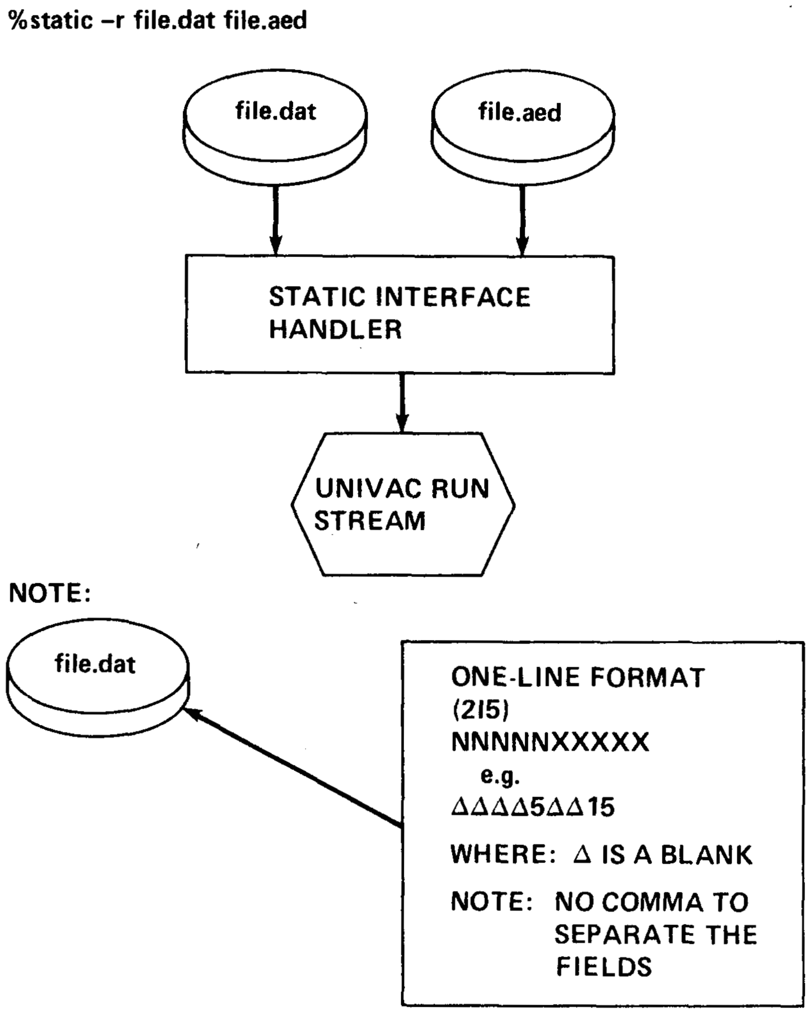

Figure 19.- Reaching-set processing flow.

```
    *** UPDATED AED REACHING SET ***

    :               :
    :               :
    :               :
    20 DEFINE PROCEDURE A.FORE.EXEC TOBE
    21      BEGIN
 PATH NUMBER =     1
    22      A.FORE.INIT();
    23      WHILE TRUE
    24      DO   BEGIN                  ... INFINITE FOREGROUND LOOP //
 PATH NUMBER =    2
    25                                  ... PATH1 , PATH2 , PATH3 , PATH4 ITERATE

    26                                      ONCE EVERY 200 MSEC //
    27                                  ... PATH13 AND PATH24 ITERATE ONCE EVERY

    28                                      100 MSEC //
    29                                  ... PATH 1234 ITERATES EVERY 50 MSEC //

    30           TOTAL.TIME = TOTAL.TIME+^00000001^;
    31           ON.GND = MAJORITY.0(5);
    32           IF PATH.NUM == 1 OR PATH.NUM == 3
    33           THEN BEGIN
 PATH NUMBER =    3
    34                CMD.SEL = REFBIT(12,DI.WO) AND REFBIT(12,DI.WO.MB);
    35                CWS.SEL = REFBIT(12,DI.W1) AND REFBIT(12,DI.W1.MB);
    36                A.LAT.COM();
    37                END
    38           ELSE BEGIN
 PATH NUMBER =    4
    39                AL.TRK.M = (AL.TRK.M OR AL.TRK.RDY OR REFBIT(1,
    40                 MDE.WRD.1.OA) AND O.BOX.V) AND NOT GA.M;
    41                A.LONG.COM();
    42                SPEED.COM();
    43                END;
 PATH NUMBER =    5
    44           GOTO PATH(PATH.NUM);


 ***** WARNING: AT STATEMENT  44: GOTO SWITCH "PATH.NUM"
 *****          CANNOT BE PROCESSED IN REACHING SETS

    45 PATH1 :   ASNBIT(FALSE,15,DO.BUFF.3);
 PATH NUMBER =    6
    46           DO.W3 = DO.BUFF.3;
    47           SENS.MON();
    48           GOTO PATH13;
    :             :
    :             :
    :             :
```

Figure 20.- Reaching-set listing.

```
*** UPDATED AED REACHING SET ***

                         REACHING SETS FOR MODULE A.FORE.EXEC
--------------------------------------------------------------------------------
           1.  REACHING SET FROM STATEMENT    22 TO STATEMENT    47
--------------------------------------------------------------------------------
STMTS IN REACHING SET NO.  1:    22     23     24     25     26     27     28
                                 29     30     31     32     33     34     35
                                 36     37     43     44     45     46     47

STMTS IN REACHING SET NO.  2:    22     23     24     25     26     27     28
                                 29     30     31     32     38     39     40
                                 41     42     43     44     45     46     47

END OF REACHING SET INPUT...
--------------------------------------------------------------------------------
```

Figure 21.- Reaching-set report.

Dynamic Tools

The dynamic tools for the AED V&V tools are intended for use after the source code has been successfully compiled. Dynamic tools are used during incremental development and debugging during execution of actual code. Dynamic tools aid in distinguishing errors caused by data or logic.

The dynamic tools are the TRACE, ASSERT, and PROBE tools and have two phases in its execution. The first phase is when the AED source file is instrumented and upon execution of the tool generates intermediate source code to link to dynamic routines. However, if the instrumented source code were encountered by the static tools or the AED compiler, the instrumented source code would be interpreted as ordinary COMMENT statements. This "intermediate instrumented" source code is passed to the AED compiler, assembler, and link loader to generate a complete load module.

After a successful load the load module with the asserted modules appears as a file on the PDP-11/60 in the user's directory as "____.obj". The first four characters from the source file name appear as the name in the "____.obj" file. At this point it is advisable to move the "____.obj" to another directory and rename it, to keep it distinctly unique; e.g., link.obj ---> Alink.obj. Once the load module is sent back to the PDP-11/60, the "____.obj" file is processed by the EXTRACT (extr) program and an executable load module is created with the same name as the "____ .obj" file, except it has the file extension ".exe". The "extr" program is described in the section "EXTRACT Program (extr)".

The second phase begins when the "____.exe" file is downloaded to the flight computers. When execution takes place in the flight computers, data is recorded and stored in the spare memory areas of the flight computer each time a path is traversed. Later, either by exhaustion of the table space or upon command, the data is collected and uploaded to a data file on the PDP-11/60. When the data is on the PDP-11/60 it is massaged and dynamic reports are generated which help to show where retesting should be addressed in more detail. See the section "Symbolic Execution."

The DFCSVL and Dynamic Execution.- As just mentioned, the dynamic tool links to some dynamic routines. These routines are unique and must be included in the "link-edit deck" (usually referred by the appendage "____.map"). Specific routines are covered in each dynamic tool description in the subsequent sections. The DFCSVL environment uses the interactive pallet interface program (pif). The pif program allows the user to perform a number of tasks necessary to control the execution of the software in the pallet. A friendly, easy to use collection of commands enables the user (1) to display or modify a memory cell through any of the four CAPS Test Adapters (CTA), (2) to set breakpoints, download or upload data or programs, (3) to search core to match a data pattern, (4) to halt the processors on the pallet, or (5) to run the software in one of three different modes. The pif program also synchronizes the generation of model data with the flight control software. A pseudo airplane model residing on the PDP-11/60 is automatically spawned by pif. The routine "TOP.OF.LOOP" reads the data created by the psuedo airplane model and uses it in the flight software. Therefore, certain routines must be present in the link-edit deck at load time. Figure 22 and 23 shows "TOP.OF.LOOP" and a section of a

```
 :         :                :
 :         :                :
 :         :                :
COMMENT ************************************************************** ;
.INSERT DFCSYSTEM;

COMMENT ***REMOVE COMMENT FOR DYNAMIC TESTING ** PROCEDURE TOP.OF.LOOP;

DEFINE PROCEDURE A.FORE.EXEC TOBE
 :         :                :
 :         :                :
 :         :                :
 :         :                :
PATH1 :   ASNBIT(FALSE,15,DO.BUFF.3);
COMMENT ** REMOVE COMMENT AND COMMENT OF NEXT LINE FOR DYNAMIC TESTING;
COMMENT **TOP.OF.LOOP();          ... REQUEST NEW MODEL DATA @ 200 MS //
          DO.W3 = DO.BUFF.3;
          SENS.MON();
          GOTO PATH13;
PATH2 :   ASNBIT(TRUE,15,DO.BUFF.3);
 :         :                :
 :         :                :
 :         :                :
```

Figure 22.- Executive-routine changes.

```
          :               :      :         :
          :               :      :         :
          :               :      :         :
>        ORIGIN @3600@,0;
|        INCLUDE DFCS(AEXTERNAL);
|       END.EXT.ARRA EQU @3BFF@; INCLUDING ALL OF THE RAM
|       ;
|       ; DEFINE EXTERNALS FOR INSTRUMENTATION ARRAY
|       ;
|        ORIGIN @3700@,0;
|        INCLUDE DFCS(INSTARRAY);
|       ;
|       ;
|       ;  F / T ONLY
|       ;
|       ;
|        ORIGIN @3B00@,0;
|        INCLUDE DFCS(FTCASE);
|        INCLUDE DFCS(FTINSERTE);
|       ;
|       ;
|       ; INCLUDE ALL OF THE ROUTINES REQUIRED FOR PROBE, TRACE
|       ; AND ASSERTION PROCESSING
|       ;
|       ;
|        INCLUDE DFCS(ALOOPTOP);  WAITS FOR MODEL DATA TO REFRESH
|        INCLUDE DFCS(GOUT);      STANDARD OUTPUT
|        INCLUDE DFCS(ASSERT);    LOGICAL ASSERTION VIOLATION ROUTINE
|        INCLUDE DFCS(OUTPUT);    TRACE ASSERTION ROUTINE
|        INCLUDE DFCS(INPUT);     TRACE ASSERTION ROUTINE
|        ORIGIN @79D9@,2;        OFFSET FOR DECISION PATH...TABLE AT 7A00
*       ;INCLUDE DFCS(DDPATH);    DECISION PATH ROUTINE
|       WAIT          EQU @7000@;  WORD TO WAIT FOR DATA
|       ASSERT.CAT    EQU @7001@;  INTERRUPT CATEGORY
|       ASSERT.SEL    EQU @7002@;  SELECTED NUMBER OF ASSERTIONS
|       ASSERT.PTR    EQU @7003@;  POINTER TO THE LOGICAL ASSERTION TABLE
|       ASSERT.BUFFER EQU @7004@;  LOGICAL ASSERTION VIOLATION TABLE
|       INPUT.CAT     EQU @7001@;  INTERRPUT CATEGORY
|       INPUT.PTR     EQU @7500@;  POINTER TO THE TRACE TABLE
|       INPUT.BUFFER  EQU @7501@;  TRACE TABLE
|       DDPATH.BUFFER EQU @7A00@;  DDPATH TABLE
>       ;
        :             :      :        :
        :             :      :        :
        :             :      :        :
```

Note: * In column 1, there is a semicolon (;) making this line a comment.
DDPATH is not required for Assertion testing.

Figure 23.- Link-deck changes showing where special routines are included.

typical link deck with the special routines, respectively. In figure 23, the INCLUDE statement for ALOOPTOP causes the object module ALOOPTOP to be loaded. The command ALOOPTOP contains the entry point for "TOP.OF.LOOP."

The executive routine must have a PROCEDURE declaration and procedure call for "TOP.OF.LOOP". A section of code from an executive routine is presented below showing where the code should be changed. After the changes have been made, the source code must be compiled and linked.

Each of the dynamic tools are discussed separately, but they do have a similar UNIX command format as seen below.

NAME

```
assert - process AED source for assertions.
probe  - process AED source for path instrumentation.
trace  - process AED source for I/O assertions.
```

SYNOPSIS

```
assert
probe  [-c class] [-mp#] [-mt#] [-z] -l mapfile.map AED_file...
trace
     ..(files must end in [.aed], [.fof] or [.isd])
```

DESCRIPTION

The dynamic command (assert, probe or trace) invokes the V&V dynamic tool on the Univac 1100. The command accepts publicly readable files whose file names end with .aed for AED source modules; .fof for file-of-files or .isd for files residing on the Univac computer site. Files ending in .aed or .fof may be sent at the same time. Files ending in .isd must all be sent together and must reside on the Univac 1100 computer. The module is instrumented, compiled, assembled, and link edited, and the output is routed as specified by the following flag arguments.

Output listings from the Univac are routed to the user's directory on the PDP-11/60 with a file name consisting of the first four characters from the first file name with a ".obj" extension attached.

The following arguments are interpreted by the dynamic tool:

-c Univac 1100 job priority. Priorities range from A through Z, the highest to lowest priority, respectively. The default is class A (standard).

-mp# This option allows the user to specify the maximum number of pages for the Univac 1100 printout. The default number of pages used by this command is computed based on the number of files (AED modules) which is approximately 16 pages per file; e.g., -mp50

-mt# This option allows the user to specify the maximum time for execution on the Univac 1100. A number preceded by an "s" (e.g., -mts40) is assumed to be in seconds. The default time limit used by this command is computed based on the number of files (AED modules), approximately 60 sec/file.

-z The z option allows for debugging or displaying the run stream to determine if the run was generated correctly. See static tool format for detailed description of the -z option.

-l This flag is required for dynamic processing. It is to link the program from a file, "____.map", which specifies modules, addresses, and so on, for the link editor. The "____.map" MUST BE PRESENT with the -l option.

Each tool will be discussed in detail.

NOTE

Source files manipulated by this command must be publicly readable in order for them to be copied to the RJE queue or accessable on the Univac.

DIAGNOSTICS

The job may successfully be sent to the Univac 1100 but fail to run for many reasons. The user's output listing will contain one or more cryptic messages why the run failed. If it is not obvious, seek a user consultant for diagnostic help.

Instrumenting the Source Code.- A module is instrumented by placing assertion statements in areas of the source code where variables or logic paths are to be tested. The ASSERT and TRACE dynamic tools interpret the instrumented source, generate dynamic tool numbers, and insert linkage to dynamic routines in the source code. Because these dynamic tool numbers exist and the source has been instrumented, the modules must be compiled, assembled, and linked to keep the dynamic tool numbers consistent. The dynamic tool numbers are referenced by the report generator to produce dynamic tool reports after execution.

An exception to the instrumentation process is the PROBE dynamic tool. The PROBE dynamic tool performs a path analysis on the AED source code first, then generates dynamic tool numbers and inserts the linkage dynamic routines in the source code. The subsequent procedures for the PROBE tool after the source code has been instrumented is the same as for the ASSERT and TRACE tool.

Assertion Tool.-

UNIX COMMAND EXAMPLE:

```
assert -l Alink.map alatcom.aed
```

This command calls on the ASSERTION tool to generate and insert linkage to dynamic routines for the AED module alatcom. Then compile, assemble and link the module based on the link-edit deck, "Alink.map".

UNIVAC COMMAND EXAMPLE:

```
@XQT AMES*ASSERT.ASSERT
@ADD,E AVFS$$.filename
```

The user instruments the AED source code by inserting assertion statements in areas of the source code to track a variable or variables. A maximum of 30 assertion statements is allowed in the source code. Assertions can be placed:

After subprogram entry
Before subprogram exit
After subprogram invocations
At decision points
Within long computations
Where data enters
Where boundary checks should be made

There are several formats available for assertion statements in the source code. They are listed (not ranked) as follows:

```
COMMENT ASSERT* boolean expression;

COMMENT ASSERT* ALL control IN (initial value,
     final value) boolean expression;

COMMENT ASSERT* SOME control IN (initial value,
     final value) boolean expression;

COMMENT INITIAL* -- synonym of ASSERT.

COMMENT FINAL* -- synonym of ASSERT.
```

Examples

```
COMMENT ASSERT* HEIGHT >500;

COMMENT ASSERT* ALL I IN(1,N) X(I) > Y(I);

COMMENT ASSERT* SOME I IN(1,M) X(I) == 3000.0;
```

After the AED source module has been asserted, the source module is analyzed by the ASSERT tool and an intermediate file containing dynamic tool numbers and links to dynamic assertion routines is generated. This intermediate file is passed to the AED processors for further processing. The dynamic tool numbers are retained in a file named "TABFIL" on the Univac. The TABFIL file is listed and returned to the PDP-11/60 along with the results from the AED processors. This print output is subsequently used by the report generator.

A special note for the program using assertions. The link-edit deck MUST OMIT the "INCLUDE program-file(DDPATH);" for the loading process. Figure 24 depicts the flow of the AED source from input to the load module's completion.

Figure 25 shows the results of the ASSERTION report. In figure 25, Module 1 is a dynamic tool number assigned by the ASSERT tool; the line number references the statement number within the module where the assertion violation has occurred and at the nth tick time. Each tick is 50 msec.

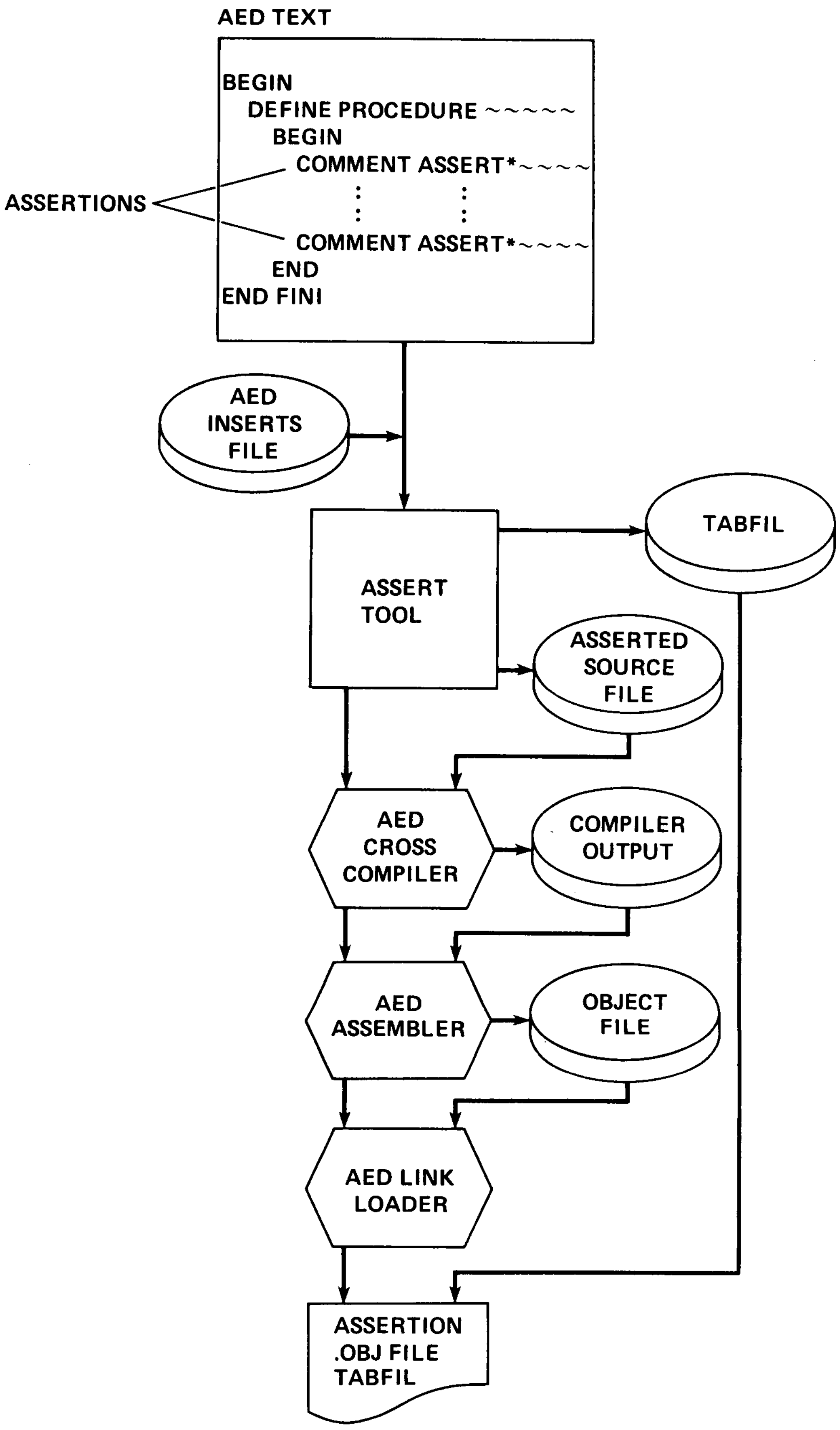

Figure 24.- Assertion-tool flow.

```
CTA 1: Recording assertion violations

CTA 1: 30 assertion violations

Module #1, line 40, time 2
Module #1, line 40, time 5
Module #1, line 40, time 8
Module #1, line 39, time 10
Module #1, line 39, time 11
Module #1, line 40, time 11
    :    :    :    :    :    :
    :    :    :    :    :    :
    :    :    :    :    :    :
Module #1, line 40, time 26
Module #1, line 39, time 27
Module #1, line 39, time 28
Module #1, line 39, time 29
Module #1, line 39, time 30
Module #1, line 40, time 30
```

Figure 25.- Assertion report.

Trace Tool-

UNIX COMMAND EXAMPLE:

```
trace -l Alink.map alatcom.aed
```

This command calls on the TRACE tool to generate and insert linkage to dynamic routines for the AED module alatcom. Then compile, assemble, and link the module based on the link edit deck, "Alink.map."

UNIVAC COMMAND EXAMPLE:

```
@XQT AMES*TRACE.TRACE
@ADD,E AVFS$$.filename
```

The user instruments the AED source code by inserting assertion statements in areas of the source code where it is desirable to track the variable or variables in question. Although there is no limit to the number of assertions placed in the source code there is a dynamic table-size limit of 40 entries. The table is circular in performance; that is, after the fortieth entry, the forty-first entry occupies position number one of the table, the forty-second entry occupies number two, the eighty-first entry occupies position number one of the table, so when execution is halted, the last 40 entries are in the table.

The TRACE tool automatically processes the TRACE assertion statements "COMMENT INPUT* ----" and "COMMENT OUTPUT* ----", but the "COMMENT ASSERT* ----" statements cause the TRACE tool problems. Do not include the "COMMENT ASSERT* ----" statements in the source code.

The user has two formats available to put a TRACE assertion statement in the source code. They are listed (not ranked) as follows:

```
COMMENT INPUT* expression;

COMMENT OUTPUT* expression;
```

Examples

```
COMMENT INPUT* BOOLEAN ALIGN;

COMMENT OUTPUT* LONG TOTAL.TIME;
```

After the AED source module has been asserted, the source module is analyzed by the TRACE tool, and an intermediate file containing links to dynamic trace routines is generated. This intermediate file is passed to the AED processors for further processing. The link edit deck for the module having assertions MUST OMIT the "INCLUDE program-file(DDPATH);" in the link-edit deck. Figure 26 depicts the flow of the AED source from input to load-module completion. Figure 27 shows the results of the TRACE report in ascending time.

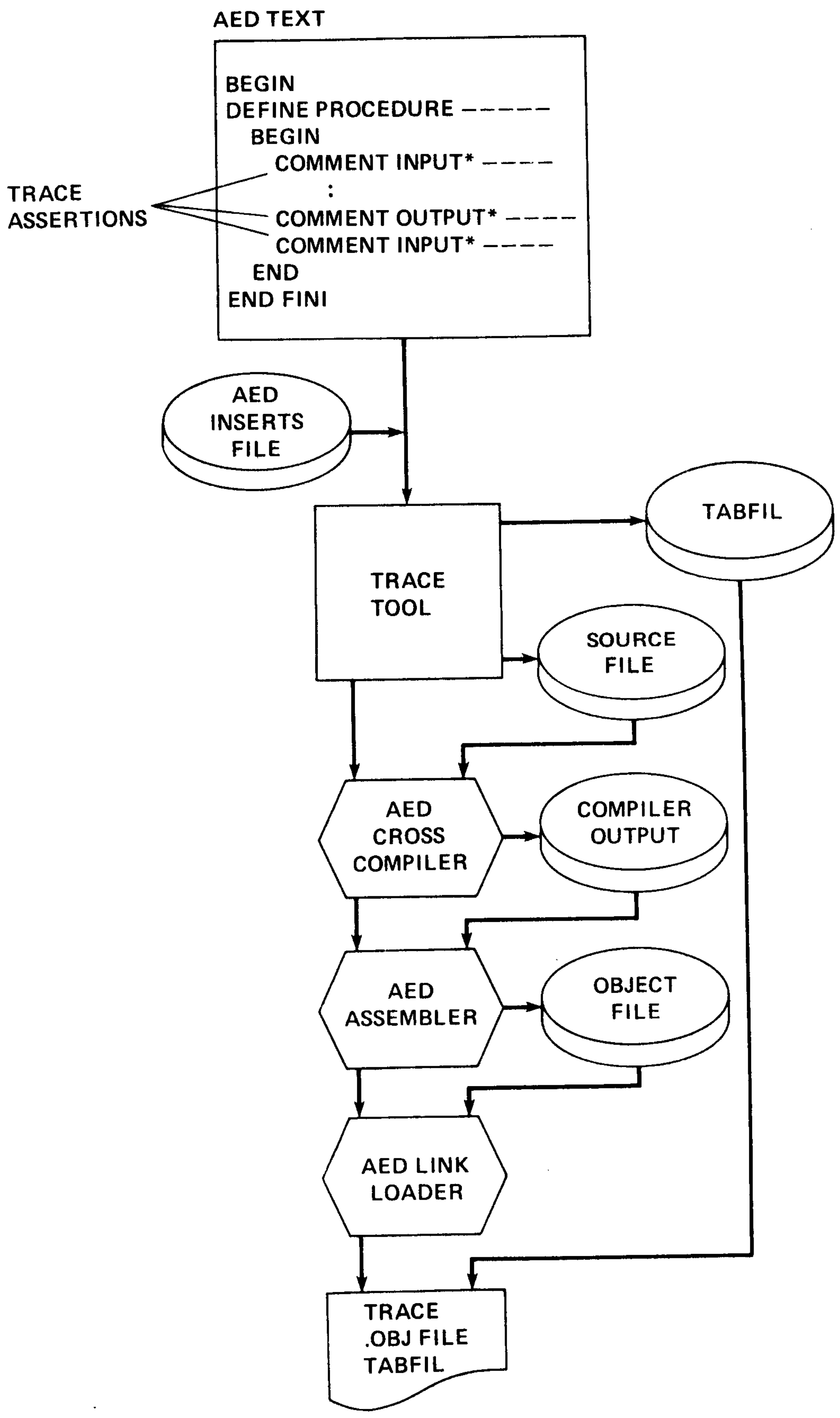

Figure 26.- Trace-tool flow.

```
                         TRACE REPORT
                         ENTRIES 40

 ------------------------------------------------------------------------
    Procedure Name   Variable Name      Type     Usage  Time   Value (hex)
 ------------------------------------------------------------------------

 1. LIMIT                        Y      REAL     INPUT   89     00002000
 2. LIMIT                        Y      REAL     INPUT   89     00000555
 3. LIMIT                        Y      REAL     INPUT   89     000003BC
 4. MPYE                         X      REAL     INPUT   89     0000FFF3
 5. MPYE                         X      REAL     INPUT   89     000014D3
 6. ROLL.FLTDIR                DPC      REAL     INPUT   89     0000999A
 7. ROLL.FLTDIR         TOTAL.TIME      LONG     INPUT   89     00000059
 8. ROLL.FLTDIR             FDSELL   BOOLEAN    OUTPUT   90     00000001
 9. ROLL.FLTDIR               DIFF      REAL     INPUT   90     00000AB5
10. ROLL.FLTDIR             FDSELL   BOOLEAN    OUTPUT   90     00000001
 :    :                        :         :         :      :         :
 :    :                        :         :         :      :         :
 :    :                        :         :         :      :         :
30. ROLL.FLTDIR             FDSELL   BOOLEAN    OUTPUT   92     00000001
31. ROLL.FLTDIR               DIFF      REAL     INPUT   92     00000A48
32. ROLL.FLTDIR             FDSELL   BOOLEAN    OUTPUT   92     00000001
33. ROLL.FLTDIR               DIFF      REAL    OUTPUT   92     00000A48
34. LIMIT                        Y      REAL     INPUT   92     00002000
35. LIMIT                        Y      REAL     INPUT   92     00000555
36. LIMIT                        Y      REAL     INPUT   92     000003BC
37. MPYE                         X      REAL     INPUT   92     0000FFF6
38. MPYE                         X      REAL     INPUT   92     00001485
39. ROLL.FLTDIR                DPC      REAL     INPUT   92     0000999A
40. ROLL.FLTDIR         TOTAL.TIME      LONG     INPUT   92     0000005C

End of data.

                       Report Generated: Mon Jul 22 10:10:54 1985
                                 By User: jim
 ------------------------------------------------------------------------
```

Figure 27.- Trace report.

Probe Tool-

UNIX COMMAND EXAMPLE:

```
probe -l Alink.map rollfltdir.aed
```

This command calls on the PROBE tool to generate and insert linkage to dynamic routines for the AED module rollfltdir. Then compile, assemble, and link the module based on the link-edit deck, "Alink.map."

UNIVAC COMMAND EXAMPLE:

```
@XQT AMES*PROBE.PROBE
@ADD,E AVFS$$.filename
```

No instrumentation is necessary of the AED source module prior to the execution of the PROBE tool. Once the PROBE tool is executed, a data-path analysis is performed on the AED source code and for each data path a link is generated to the dynamic PROBE routine. This intermediate source code is passed to the AED processors for compiling, assembling, and linking. The link-edit deck for the dynamic PROBE execution MUST include the "INCLUDE program-file(DDPATH);". Figure 28 depicts the flow of the AED source from input to load-module completion. Figure 29 shows the PROBE report after dynamic-flight software execution with PROBE routines collecting data during execution.

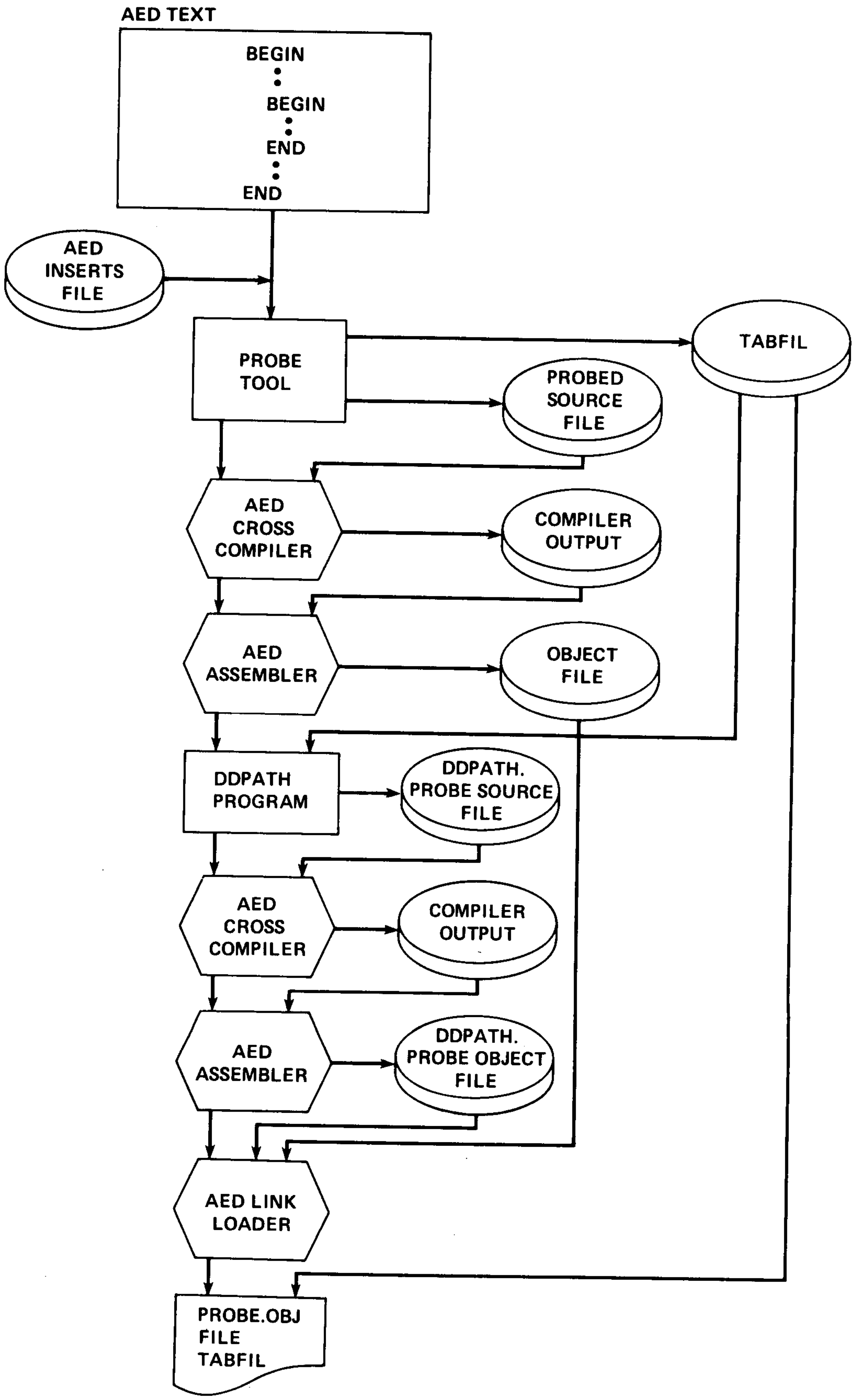

Figure 28.- Probe-tool flow.

```
                         PROBE REPORT

  Procedure name:          MPYE
  Number of paths:         1

              1     2     3     4     5     6     7     8     9    10
            ----------------------------------------------------------
          |
 1 - 10   |  363
          |

  Procedure name:          LIMIT
  Number of paths:         1

              1     2     3     4     5     6     7     8     9    10
            ----------------------------------------------------------
          |
 1 - 10   |  242
          |

  Procedure name:          ROLL.FLTDIR
  Number of paths:         19

              1     2     3     4     5     6     7     8     9    10
            ----------------------------------------------------------
          |
 1 - 10   |  120   121   121     0   121     0   121     0   121   121
          |
11 - 20   |    0   121   121   121   121     0   121     0   121
          |

     End of data.

                         Report Generated: Mon Jul 22 10:05:04 1985
                                By User: jim
```

Figure 29.- Probe report.

Symbolic Execution

UNIX COMMAND EXAMPLE:

```
symbolic ctrl.dat hdgsel.aed
```

This command symbolically executes the module hdgsel.aed within the range specified in the control file "ctrl.dat," and generate a symbolic execution or verification condition generation report.

UNIVAC COMMAND EXAMPLE:

```
@XQT AMES*SYMEXEC.SYMEXEC
@ADD,E AVFS$$.filename
```

Symbolic execution, verification condition generation (VCG), or formal verification (as it is often called) of variables or expressions can be used to verify a module's output is the same as the formula it has been specified to compute. The symbolic execution of assertions is termed "verification conditions." By showing that the verification conditions for a module are true, it is said the module has been formally verified with its assertions.

Two steps are required to perform symbolic-execution or verification-condition generation. The first step is a preliminary step to obtain the necessary data; the second step is the actual generation. The program verifier uses line numbers of AED source modules to perform symbolic execution.

The UNIX command format for symbolic execution is as shown in the following list:

NAME

symbolic - AED Symbolic Execution

SYNOPSIS

```
symbolic [-c class] [-mp#] [-mt#] [-z] file.dat AED_file...
           ..(files must end in [.aed], [.fof] or [.isd])
```

DESCRIPTION

The symbolic command invokes the AED Symbolic Execution tool on the UNIVAC 1100. Variables are initialized to their default values in the module specified for the range specified in the control file ending with the file extension .dat. It accepts publicly readable files whose file names end with .aed for AED source modules, .fof for file of files, or .isd for files residing on the Univac computer site. Files ending in .aed or .fof may be sent at the same time. Files ending in .isd must all be sent together and must reside on the Univac 1100 computer. The module is analyzed and outputs are routed as specified by the following flag arguments.

Output listings from the Univac are routed to the user's directory on the PDP-11/60 with a file name consisting of the first four characters from the first file name with a ".rpt" extension attached.

The following arguments are interpreted by Symbolic:

-c	Univac 1100 job priority. Priorities range from A through Z, the highest to lowest priority, respectively. The default is class A (standard).
-mp#	This option allows the user to specify the maximum number of pages for the Univac 1100 printout. The default number of pages used by this command is computed based on the number of files (AED modules) which is approximately 16 pages per file; e.g., -mp50.
-mt#	This option allows the user to specify the maximum time for execution on the Univac 1100. A number preceded by an "s" (e.g., -mts40) is assumed to be in seconds. The default time limit used by this command is computed based on the number of files (AED modules) approximately 60 sec/file.
-z	The z option allows for debugging or displaying the run stream to determine if the run was generated correctly. See static tool format for detail description of the -z option.

NOTE

Source files manipulated by this command must be publicly readable in order for them to be copied to the RJE queue or accessable on the Univac.

DIAGNOSTICS

The job may successfully be sent to the Univac 1100, but may fail to run for many reasons. The user's output listing will contain one or more cryptic messages why the run failed. If the problem is not obvious, seek a user consultant for diagnostic help.

The symbolic execution control file, "filename.dat," is basically a FORTRAN "FOR" statement loop. The first statement is the "FOR" statement in the form

"FOR LINES = n .. m DO", where "n" is the beginning line number and "m" is the ending line number. The periods, ".." MUST be present in the statement. The second statement in the control file is the "variable" to be evaluated. The last statement of the "FOR" loop is the "END FOR" statement. To symbolically execute an expression, such as LAT.LIM.CMD between the lines 10 through 20, the following format must be specified in the symbolic execution control file:

```
FOR LINES = 10 .. 20 DO
LAT.LIM.CMD
END FOR
```

The symbolic execution tool reads the control file, lists the original expression, then it generates the executed expression for the variable specified. Figure 30 shows the results of the module "hdgsel" symbolically executed.

```
ORIGINAL EXPRESSION
________________
   LAT.LIM.CMD
   1 BEGIN
   2
   :       :
   :       :
   :       :
   6 DEFINE PROCEDURE HDG.SEL TOBE
   7
   8 COMMENT ITERATION RATE = 5 / SEC ;
   9      BEGIN
   10      IGNORE.OVRF()            ... IGNORE ARITHMATIC OVERFLOW ;
   11      HDG.ERROR = SEL.HDG-.736667*VOTER(YAW.RATE.PTR)-HEADING ...
   12                                   DETERMINE HEADING ERROR SIGNAL ;
   13      PROTECT.OVRF()           ... REINSTATE OVERFLOW PROTECTION ;
   14      LAT.LIM.CMD = LIMIT(KTAS*HDG.ERROR,.027778)/.055556 ... DETERMIME
   15                                   LAT.LIM.CMD AS A FUNCTION OF TAS AND
   16                                   HEADING ERROR, LIMIT FOR RESCALING,
   17                                   CHANGE OUTPUT SCALE TO 60 DEG / FS ;
   18      .INSERT COMMONINSERT;
   19      .INSERT HDGSELINSERT;
   20      END;
   :       :
   :       :
   :       :

FINAL EXPRESSION
________________
( LIMIT ( KTAS * ( SEL.HDG - .736667 * VOTER ( YAW.RATE.PTR
) - HEADING ) , .027778 ) / .055556 )
```

Figure 30.- Symbolic-execution report.

Figure 31 shows the control path used in executing the Symbolic Execution tool.

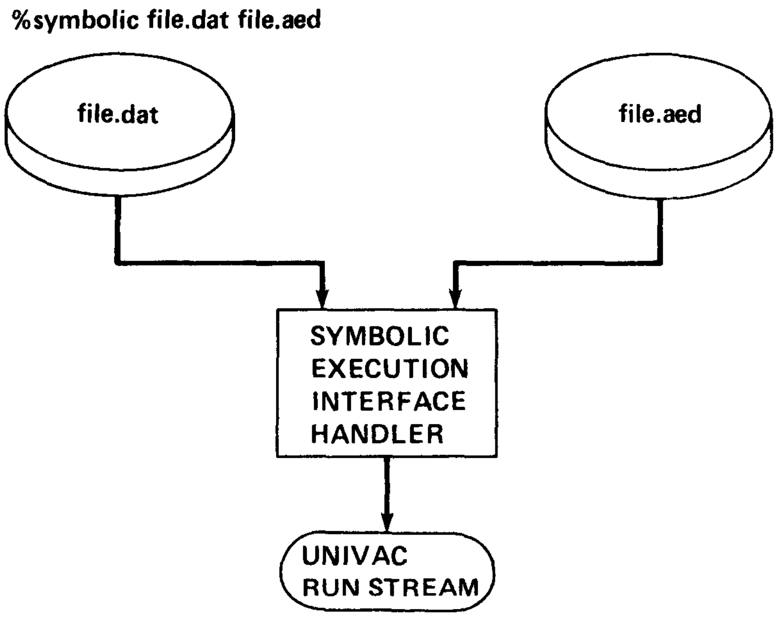
%symbolic file.dat file.aed
file.dat
file.aed
SYMBOLIC EXECUTION INTERFACE HANDLER
UNIVAC RUN STREAM

Figure 31.- Symbolic-execution flow.

EXTRACT PROGRAM (extr)

UNIX COMMAND EXAMPLE:

```
extr file.obj
```

Execute the extract program to extract CAPS 6 executable code from "file.obj".

The extr program extracts CAPS executable code from listings that have been returned from the Univac 1100. The nontransparent transmission of the RJE link requires all Univac object code sent over the RJE link be converted to ASCII code such that the file is suitable only for printing. The "extr" command creates a new file that contains the actual executable object code. The program flow is shown in Figure 32.

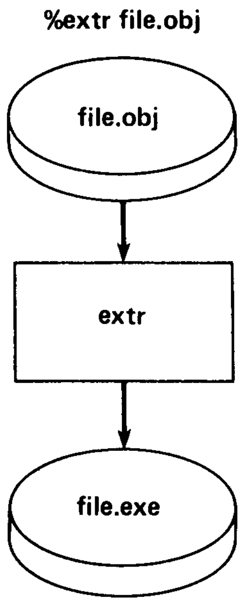

Figure 32.- Extr-program flow.

The UNIX command format for "extr" is as shown below.

NAME

extr - extract executable program from linkedit listing

SYNOPSIS

```
extr file.obj
```

DESCRIPTION

The "extr" command extracts CAPS 6 executable object code from listings that have been returned (via RJE) from the UNIVAC 1100. Because of the nontransparent RJE transmission, all object code returned over RJE are returned to a file suitable only for printing. This file contains the ASCII representation of the executable object code. The "extr" command creates a new file that contains the actual executable object code. The "extr" accepts as input an 11/60 file with a filename that ends with a ".obj" extension (this is the listing returned from the UNIVAC 1100), and outputs a filename but with the extension ".exe". It is strongly suggested that all files be inspected with the editor before proceeding with the "extr" command. An object listing is often created by the UNIVAC even if the latest compile aborted.

DIAGNOSTICS

"Filename not created with CARTRIDGE TAPE ..."
The "____.obj" file provided was not created using the "-l" option to "axc [avfs]" or the file was somehow truncated (possibly because of a bad RJE transmission). Examine the file in the editor or print the contents of the file for examination.

"Truncated object file"
This message occurs when the object file that has been input is somehow incomplete. The file could have been interrputed while being transmitted, the UNIVAC run may have terminated abnormally (aborted because of maximum pages, etc.). Examine the file in the editor or print the contents of the file for examination.

EXAMPLES

extr Alink.obj
The 11/60 file "Alink.obj" is read and the extr program determines if it was created with the cartridge tape transmission program. If so, a file named "Alink.exe" is created which contains the executable program to be downloaded to the pallet.

PALLET INTERFACE PROGRAM (pif)

UNIX COMMAND EXAMPLE:

pif
Execute the pallet interface program to perform a variety of tasks on the pallet.

The pif is an interactive UNIX program which accepts commands to perform a variety of tasks necessary to control the execution of the software in the pallet. A friendly, easy to use collection of commands enables the user to display or modify a memory cell in the flight computers through any of the four CTAs, set breakpoints, download or upload data or programs, search core to match a data pattern, halt the processors on the pallet, or run the software in one of three different modes. The pif also synchronizes the generation of model data with the flight control software. The airplane model resides in a file named /avfs/bin/model and is automatically spawned by pif.

The UNIX command format for "pif" is as shown below.

NAME

pif - pallet interface program

SYNOPSIS

```
pif [-a assert_file] [-t trace_file]
```

DESCRIPTION

The pif command invokes the interactive pallet interface program. This program allows a user to perform a number of tasks necessary to control the execution of the software in the pallet. A friendly, easy to use collection of commands enables the user to display or modify a memory cell through any of the four CTAs, set breakpoints, download or upload data or programs, search core to match a data pattern, halt the processors on the pallet, or run the software in one of three different modes. The pif synchronizes the generation of model data with the flight-control software. The airplane model resides in a file named /avfs/bin/model and is automatically spawned by pif.

The following arguments are interpreted by pif:

-a assert_file

Logical assertion violations accumulated during dynamic testing is accumulated in the file "assert_file". This file is an ASCII file containing the dynamic data produced by the software tools. The default is to print the data on the DEC-writer. See the description of the AED V&V tool assert.

-t trace_file

Dynamic data generated by the AED V&V trace tool is accumulated in file "trace_file". This data may later be formatted into a report using the Report Generator, "rpt". Omission of this option causes the trace data to remain on the pallet where it was accumulated. See trace and Report Generator descriptions.

Once pif is called, it prompts the user for the input of a command. The following commands are recognized by pif:

"help [command]"

The help command provides the user with the proper syntax and usage of the legal pif commands. The "help" command without any arguments provides a list of the legal commands and their associated syntax. The "command" argument to help is assumed to be the name of a command that the user requires additional information. The command "help help" provides information similar to that in this paragraph.

"ctas [number_of_ctas]"

The ctas command allows the user to change the number of CTAs in use by pif. The command "ctas" with no arguments echos the number of CTAs currently in use. This may be changed by using the optional argument "number_of_ctas". The command "ctas 4" enables the user to access all four CTAs, "ctas 3" allows CTAs 1, 2, and 3 to be accessed, etc. The default allows the user to access CTAs 1 and 2.

"base [octal | decimal | hex]"

The base command is used to change or display the working number system used by pif. The command "base" with no arguments echos the current number system (octal, decimal, or hexadecimal). The optional argument to base is used if the user wishes to change the number system. The command "base decimal" changes the base to decimal (unsigned decimal), etc. The default base is hexadecimal.

"quit"

This command simply exits pif and returns the user to UNIX. The model is terminated and intermediate files created by the model are removed. The use of this command causes a normal exit.

"halt"

The halt command is used to halt the execution of the flight-control software in the CAPS. It only halts those processors currently in use by pif.

"search [caps_address value]"

Search is a routine that sequentially reads through through CAPS core and compares each memory cell to the bit pattern specified. As with all of the other pif commands the two arguments (caps_address and value) are taken to be values specified in the current working number system (see the "base" command). Hence, if the command input is "search 1.100 0667", and the working number system is hexadecimal, pif begins searching at address hexadecimal 100 on CTA #1 and stops when it finds a cell that contains hexadecimal 0667. It is not really necessary to type leading zeros, all bit patterns are right justified in the word. If no match is found, the user is notified. The command "search" with no arguments begins searching at "the last address that contained a match" + 1 for the same data pattern. This command enables a user to find all occurrences of the data pattern without having to retype the entire command several times. If the working number system is octal and the command "search 1.100 888" is given, unusual results will occur.

"break [caps_address] [data]"

The break command is used to set the breakpoint bit in the control register of the specified CTA. Only one breakpoint may be set for each CTA. There are two types of breakpoints that may be set. The first is an "address breakpoint." This type causes a bus halt when the specified address occurs of the transfer bus. When a breakpoint is encountered on any CTA, all CTAs are halted by pif and the user is notified. This type of breakpoint may be set by specifying the "caps_address" argument and not the "data" argument. For example, "break 2.3F66" sets CTA #2 to bus halt when the address 3F66 occurs on its transfer bus. The second type of breakpoint that may be set is an "address and data breakpoint." It is set by specifying both arguments to the break command. The command "break 3.548A 1010" causes a bus halt on CTA #3 when the address 548A is on its transfer bus and that address contains the value 1010. The "break" command with no arguments cancels all breakpoints on all CTAs.

"monitor"

The monitor command allows the user to display and/or modify the contents of any positive address in the CAPS memory. Note that a positive address is one which is not on the I/O page. Monitor can access addresses up to and including "7FFF" and display them one at a time, or dump 50 values with a single key-stroke. The operation of this routine includes a subset of the directives allowed by the Monitor program that runs on the PDP 11/04 on the pallet. Once the monitor command is issued, the following subcommands are legal:

"/" Display current address, see notes regarding buffering below.

"\\" Display current address - 1, and back up pointer.

"^" Display current address + 1, and move pointer forward.

" " (space) Dump the buffer and increment pointer (contains 50 memory locations).

"q" Quit (exit) the monitor processor and return to pif.

A memory cell is modified by typing the address (CTA.address... "1.1000" for example), followed by a "/" directive, then typing a new value. It is important to note that this monitor routine differs from the 11/04 routine in many ways. Most important is the fact that all I/O in the 11/60 version is buffered. Repeatedly typing "/" will not necessarily display an updated memory location. The buffer may be written to CAPS memory by typing a carriage return. The same applies to modification of a particular location. Again, the location is updated when a new value is entered followed by a carriage return.

"upld [-v] cta.address word_count 11/60_file"

The "upld" command uses the da11b link to request data from the target computer to be made available for test reporting in file "11/60_file." The "upld" retrieves the "word_count" words from the target computer starting from "address." The range of address that may be accessed by "upld" begins at 0 and runs up to 7FFF inclusive.

The following arguments are interpreted by upld:

-v Verbose mode, notify the user when upload is complete. The "upld" without this option works silently.

"dnld [-v] cta[.address] 11_60/file"

The "dnld" command loads an executable program or data to the CAPS through any of the four CTAs. It accepts files whose names end with .exe (CAPS executable programs). The program/data is passed to the target computer via the da11b link. Two types of links may be downloaded. If "address" is supplied in the command line, it is assumed that the file is data only containing no address or loading information. If "address" is not supplied, "dnld" assumes that the file contains object code that has been received from the AED cross compiler and processed with the extr command. This object code contains all of the address information needed to download the program. The range of address that may be accessed by "dnld" is from 0 to 7FFF inclusive.

The following arguments are interpreted by dnld:

-v Verbose mode, notify the user when download is complete. The "dnld" without this option works silently.

"sh [UNIX_command] [args]]"

This command allows the user to execute UNIX commands without leaving the pif processor. The argument(s) is taken to be the UNIX command that the user wishes to execute. The "sh" with no arguments spawns a new shell. Be careful...this is not the same as the "quit" command! Because of the method of airplane model data synchronization, "cd" and "chdir" are disabled.

"assert [number_of_asserts]"

The "assert" command allows the user to change the maximum number of logical assertion violations that are accumulated in the CAPS before the user is notified. The maximum number of asserts that may be set is 30 (this is the default). "Assert" sets the same number of maximum assertions for both the A and B channel of FCC #1. The "assert" with no arguments performs a completely different task. It is used in this way to check to see if there are any assertions already detected, but waiting to be sent. For example, if the maximum number of assertions is set at 20, and only 10 are collected during a given test session, the user would have to issue "assert" to collect the assertion data.

"mode [continuous | cycle | step]"

The mode command is used to alter the way in which pif interprets the "go" command. Three different modes are possible. Continuous mode implies that when the "go" command is issued, the 11/60 airplane model continuously furnishes model data to the MDICU and lets the software in the pallet run indefinitely. Cycle mode is used when the user wishes to run only one iteration through the flight software. Only one block of data from the airplane model is transmitted to the MDICU. Use of cycle mode assumes that the user has previously set a breakpoint at the bottom (or top) of the flight-software loop. The third mode is step mode. With pif set in step mode, each go command that is issued causes a bus step (the same as pushing the bus step key on the CTA). In step mode, pif echoes the contents of the data, and the data and address registers after each bus step. Breakpoints are disabled in step mode. The "mode" command without arguments echoes the current mode.

"go [reset]"

This command places the CTAs in the run mode as selected by the "mode" command. The optional argument "reset" causes a system reset (the same as pushing the reset key on the CTA). Without the argument, execution begins at the address that is currently on the address bus.

DIAGNOSTICS

All diagnostics are self-explanatory.

REPORT GENERATOR

UNIX COMMAND EXAMPLE:

```
rpt [-t] file.dyn file.obj
```

Produce a dynamic report from the data in file.dyn in conjunction with the dynamic file pointers stored in the listing file, file.obj.

The report generator produces a formatted dynamic report based on the inputs it receives from the UNIX command line. Probe or Trace data collected during execution is stored in the flight computer then uploaded to the PDP-11/60 and stored in a file ending with a name ".dyn", e.g. "____.dyn". An associated file, "____.obj", for the probe- or trace-load module is used as input to the report generator to produce the formatted dynamic report. In the section Dynamic V&V Tools, dynamic tool numbers were generated and used in the linkages to dynamic routines. They are also used by the report generator. The program flow is shown in figure 33.

The UNIX command format for the report generator is as shown below.

NAME

rpt - Produce formatted report from dynamic data

SYNOPSIS

```
rpt [-t] file.dyn file.obj
```

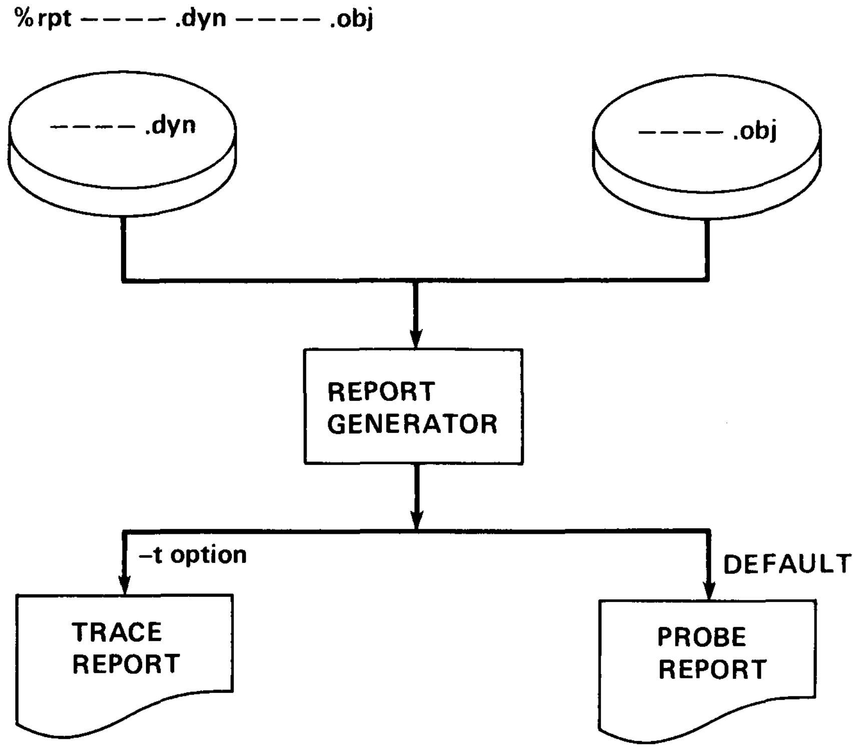

Figure 33.- Report-generator program flow.

DESCRIPTION

The "rpt" command formats a report from data collected by a dynamic test of flight software and the listing created by the V&V tool that prepared that flight software for testing. Two types of reports may be created.

The first is a "path" or "PROBE" report. For this report, the data in file "file.dyn" is assumed to have been collected during dynamic execution of the software prepared by the probe tool. The data in the file "file.dyn" contains path numbers and an associate time for each path number. The time recorded represents the number of times the corresponding path had been executed as a result of the dynamic test.

The second type of report is a TRACE report. The data in file "file.dyn" is assumed to have been collected during dynamic execution of the flight software variable prepared by the trace tool. The output of "rpt" has in its heading the number entries in the table and the CTA number on which the variables were found. The report then lists the entry number, the module/procedure name containing the variable, the variable name, the variable type, the usage, the time (from the flight software) the variable was traced, and the value of the variable when the trace was encountered, see figure 27. The data file returned by dynamic execution must be input to rpt in file ending "____.dyn"; e.g., "file.dyn".

Both reports require a file, "file.obj" in addition to the file "file.dyn". "File.obj" is a listing file returned by the Univac and contains the load module listing for either the TRACE or PROBE flight-software load. Within "file.obj" is a table containing dynamic tool numbers which are associated with the numbers in the data file "file.dyn."

Note that "file.dyn" and "file.obj" are unique files, are generally paired together and usually have unique names.

The following arguments are interpreted by rpt:

-t Format a TRACE report. Use of this option causes rpt to format a TRACE report. The default is to format a path or PROBE report. Be sure to input the correct data file!

EXAMPLES

```
rpt -t test.dyn afor.obj
```

Format a TRACE report from dynamic data test.dyn and listing afor.obj.

```
rpt paths.dyn alat.obj
```

Format a path (PROBE) report from dynamic data paths.dyn and listing alat.obj.

CONSTRAINTS

The AED V&V tools impose 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). The tools are capable of handling quite large source text files. However, an unusually large program may have to be processed by several successive executions, each operating on a separate file of modules.

Universal Constraints

The universal constraints are as follows:

At most 250 modules may use the same COMMON block

Maximum of 80 characters per source-card image read

The maximum number of DD-paths is 50.

The maximum number of statements on a single DD-path is 100.

The sizes of the two random files on logical unit 8 (LIBNEW) and logical unit 11 (LIBOLD) are established using an OPEN statement in tool subroutine INITIN. The current sizes are 500 records (of 300 words each).

Subroutine NITTOK of the tools uses an OPEN statement to establish two direct access scratch files, TOKFIL and LNKFIL, on logical units 21 and 22, respectively. Improper arguments can lead to excessive execution time for the tools.

Maximum of 250 tokens per statement.

Syntax Constraints

The following implementation constraints 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.

Tool Deficiencies

The AED V&V tools were evaluated and tested under contract by Boeing Commercial Airplane Company. The final report is to be published and was not available for this User's Guide. The following list (not ranked) of tool deficiencies is not complete for the current version of the AED V&V tools:

If a tool encounters "SYNTAX ERROR DUMP" during its lexical analysis of a module, a tool report is still generated but not accurate.

The PROBE tool does not check the path for branches if the branches occur to the right of an equal sign.

The PROBE tool does not generate the proper calling sequence for the dynamic routine DDPATH when processing multi-procedure --- multi-modules.

The COMMENT ASSERT* statement CANNOT be present in the source code when executing the TRACE tool.

COMMENT INPUT* and COMMENT OUTPUT* statements are permitted in the source code when executing the ASSERT tool.

UNITS tool cannot properly process units in expressions within an argument list.

UNITS tool has difficulty processing units such as acceleration (feet/sec**2 or feet/sec/sec) in expressions.

A maximum of 30 logical assertions violations can be accumulated during dynamic execution.

REACHING SET control file requires the input to be in a specific type format. It must be of the FORTRAN format 2I5. The REACHING SET tool reads the starting statement line number and the ending statement line number before the tool starts reaching analysis.

Multimodules are handled differently by the CALLING TREE tool. The modules MUST be CONCATENATED sequentially then given to the CALLING TREE for processing. The Calling Tree interface handler takes care of concatenation.

Dynamic tools ASSERT,TRACE, and PROBE require special loading procedures (refer to "Dynamic Tools" section for details).

SUMMARY OF AED V&V TOOL COMMANDS

A summary of commands is presented in the next two sections. In the first section, a summary of all the PDP-11/60 UNIX commands for all the tools is presented. An example is given for a typical situation of one AED module being processed.

In the second section, a summary of all Univac commands necessary to execute each tool is presented. Again, the typical situation is of one AED module being processed.

PDP-11/60 UNIX* Tool Interface Commands

The off-site Univac (table 1) shows the commands that would be used to process a typical AED static module called "sample_stat.aed", a typical AED Asserted module called "samp_assert.aed", a typical AED Trace module called "samp_trace.aed", and a typical AED Probe module called "samp_probe.aed." Also sample control files for reaching set and symbolic execution called "reach.dat" and "symbolic.dat," respectively, were used. Sample link edit decks appear for the Assert, Trace, and Probe tools whose names are "trass.map" for assertion and trace link edit decks and "probe.map" for the probe link-edit deck.

TABLE 1.- PDP-11/60 UNIX TOOL INTERFACE COMMANDS

AED V & V Tool	PDP-11/60 UNIX* Command
Module Dependencies	%static -d sample_stat.aed
Global Cross Reference	%static -g sample_stat.aed
Interface	%static -i sample_stat.aed
Enhanced Listing & Profile	%static -l sample_stat.aed or %static sample_stat.aed
SET/USE	%static -s sample_stat.aed
Calling Tree	%static -t sample_stat.aed
Units	%static -u sample_stat.aed
Invocations	%static -v sample_stat.aed
Cross Reference	%static -x sample_stat.aed
Reaching Set	%static -r reach.dat sample_stat.aed
Assertion	%assert -l trass.map samp_assert.aed
Trace	%trace -l trass.map samp_trace.aed
Probe	%probe -l probe.map samp_probe.aed
Symbolic Execution	%symbolic symbolic.dat samp_symbol.aed
Extract Program	%extr link.obj
Report Generator	%rpt data.dyn list.obj
Pallet Interface Program	%pif

Univac 1100 Commands

The following pages prefaced by "UNIVAC 1100 V & V Tool name" represent the equivalent Univac runstreams necessary to execute the pertinent AED V&V tool.

UNIVAC 1100 AED Dependencies Matrix

```
@RUN Univac run card
@ASG,AX INSERTS.
@FREE AVFS$$.
@ASG,T AVFS$$.
@ELT,I AVFS$$.SAMPLE_STAT
   THIS IS THE BEGINNING OF A DUMMY AED STATIC SOURCE FILE
                  :
          AED STATEMENTS WOULD USUALLY BE HERE
                  :
   THIS IS THE END OF THE DUMMY AED STATIC SOURCE FILE
@HDG,X *** UPDATED AED DEPENDENCE MATRIX ***
@FREE TKFIL.
@ASG,T TKFIL.
@USE 21.,TKFIL.
@FREE LKFIL.
@ASG,T LKFIL.
@USE 22.,LKFIL.
@XQT AMES*DEP.DEPEND
@ADD AVFS$$.SAMPLE_STAT
@FIN
```

UNIVAC 1100 AED Global Cross Reference

```
@RUN Univac run card
@ASG,AX INSERTS.
@FREE AVFS$$.
@ASG,T AVFS$$.
@ELT,I AVFS$$.SAMPLE_STAT
    THIS IS THE BEGINNING OF A DUMMY AED STATIC SOURCE FILE
                    :
           AED STATEMENTS WOULD USUALLY BE HERE
                    :
    THIS IS THE END OF THE DUMMY AED STATIC SOURCE FILE
@HDG,X *** UPDATED AED GLOBAL CROSS REFERENCE ***
@FREE ELSE$$.
@ASG,T ELSE$$.
@USE 24.,ELSE$$.
@FREE TKFIL.
@ASG,T TKFIL.
@USE 22.,TKFIL.
@FREE LKFIL.
@ASG,T LKFIL.
@USE 21.,LKFIL.
@FREE FTN028.
@ASG,T FTN028.
@USE 28.,FTN028.
@ASG,T TKTAB.
@XQT AMES*GBLXREF.INITIAL
@XQT AMES*GBLXREF.GLOBAL
@ADD,E AVFS$$.SAMPLE_STAT
@XQT AMES*GBLXREF.FINAL
@FIN
```

UNIVAC 1100 AED Interface

```
@RUN Univac run card
@ASG,AX INSERTS.
@FREE AVFS$$.
@ASG,T AVFS$$.
@ELT,I AVFS$$.SAMPLE_STAT
    THIS IS THE BEGINNING OF A DUMMY AED STATIC SOURCE FILE
                      :
           AED STATEMENTS WOULD USUALLY BE HERE
                      :
    THIS IS THE END OF THE DUMMY AED STATIC SOURCE FILE
@HDG,X *** UPDATED AED INTERFACE REPORT ***
@FREE TKFIL.
@ASG,T TKFIL.
@USE 21.,TKFIL.
@FREE LKFIL.
@ASG,T LKFIL.
@USE 22.,LKFIL.
@ASG,AX user_login_id*LIBOLD.
@USE 11.,user_login_id*LIBOLD.
@DELETE,C LIBNEW.
@CAT,P LIBNEW.
@ASG,AX LIBNEW.
@USE 8.,LIBNEW.
@XQT AMES*INTER.INTER
@ADD,E AVFS$$.SAMPLE_STAT
@COPY LIBNEW.,user_login_id*LIBOLD.
@FIN
```

UNIVAC 1100 AED List and Profile

```
@RUN Univac run card
@ASG,AX INSERTS.
@FREE AVFS$$.
@ASG,T AVFS$$.
@ELT,I AVFS$$.SAMPLE_STAT
    THIS IS THE BEGINNING OF A DUMMY AED STATIC SOURCE FILE
                    :
           AED STATEMENTS WOULD USUALLY BE HERE
                    :
    THIS IS THE END OF THE DUMMY AED STATIC SOURCE FILE
@HDG,X *** AED ENHANCED LISTING, MODULE HDGSEL ***
@XQT GRC*LIST.LIST
@ADD,E AVFS$$.SAMPLE_STAT
@HDG,X *** UPDATED AED STATEMENT PROFILE ***
@XQT AMES*PROFILE.PROFILE
@ADD,E AVFS$$.SAMPLE_STAT
@FIN
```

UNIVAC 1100 AED Set/Use

```
@RUN Univac run card
@ASG,AX INSERTS.
@FREE AVFS$$.
@ASG,T AVFS$$.
@ELT,I AVFS$$.SAMPLE_STAT
    THIS IS THE BEGINNING OF A DUMMY AED STATIC SOURCE FILE
                   :
           AED STATEMENTS WOULD USUALLY BE HERE
                   :
    THIS IS THE END OF THE DUMMY AED STATIC SOURCE FILE
@HDG,X *** UPDATED AED SETUSE REPORT ***
@FREE TKFIL.
@ASG,T TKFIL.
@USE 21.,TKFIL.
@FREE LKFIL.
@ASG,T LKFIL.
@USE 22.,LKFIL.
@ASG,T TKTAB.
@XQT AMES*SETUSE.SETUSE
@ADD,E AVFS$$.SAMPLE_STAT
@FIN
```

UNIVAC 1100 AED Calling Tree

```
@RUN Univac run card
@ASG,AX INSERTS.
@FREE AVFS$$.
@ASG,T AVFS$$.
@ELT,I AVFS$$.SAMPLE_STAT
    THIS IS THE BEGINNING OF A DUMMY AED STATIC SOURCE FILE
                     :
           AED STATEMENTS WOULD USUALLY BE HERE
                     :
    THIS IS THE END OF THE DUMMY AED STATIC SOURCE FILE
@HDG,X *** UPDATED AED CALLING TREE ***
@FREE TKFIL.
@ASG,T TKFIL.
@USE 21.,TKFIL.
@FREE LKFIL.
@ASG,T LKFIL.
@USE 22.,LKFIL.
@XQT AMES*CTREE.CTREE
@ADD AVFS$$.SAMPLE_STAT
@FIN
```

UNIVAC 1100 AED Units

```
@RUN Univac run card
@ASG,AX INSERTS.
@FREE AVFS$$.
@ASG,T AVFS$$.
@ELT,I AVFS$$.SAMPLE_STAT
    THIS IS THE BEGINNING OF A DUMMY AED STATIC SOURCE FILE
                    :
           AED STATEMENTS WOULD USUALLY BE HERE
                    :
    THIS IS THE END OF THE DUMMY AED STATIC SOURCE FILE
@HDG,X *** UPDATED AED UNITS ANALYSIS ***
@XQT AMES*UNITS.UNITS
@ADD,E AVFS$$.SAMPLE_STAT
@FIN
```

UNIVAC 1100 AED Invocation

```
@RUN Univac run card
@ASG,AX INSERTS.
@FREE AVFS$$.
@ASG,T AVFS$$.
@ELT,I AVFS$$.SAMPLE_STAT
    THIS IS THE BEGINNING OF A DUMMY AED STATIC SOURCE FILE
                    :
           AED STATEMENTS WOULD USUALLY BE HERE
                    :
    THIS IS THE END OF THE DUMMY AED STATIC SOURCE FILE
@HDG,X *** UPDATED AED INVOCATIONS REPORT ***
@FREE TKFIL.
@ASG,T TKFIL.
@USE 21.,TKFIL.
@FREE LKFIL.
@ASG,T LKFIL.
@USE 22.,LKFIL.
@XQT AMES*INVOKE.INVOKE
@ADD AVFS$$.SAMPLE_STAT
@FIN
```

UNIVAC 1100 AED Cross Reference

```
@RUN Univac run card
@ASG,AX INSERTS.
@FREE AVFS$$.
@ASG,T AVFS$$.
@ELT,I AVFS$$.SAMPLE_STAT
    THIS IS THE BEGINNING OF A DUMMY AED STATIC SOURCE FILE
                      :
           AED STATEMENTS WOULD USUALLY BE HERE
                      :
    THIS IS THE END OF THE DUMMY AED STATIC SOURCE FILE
@FREE ELSE$$.
@ASG,T ELSE$$.
@USE 24.,ELSE$$.
@FREE TKFIL.
@ASG,T TKFIL.
@USE 22.,TKFIL.
@FREE LKFIL.
@ASG,T LKFIL.
@USE 21.,LKFIL.
@ASG,T TKTAB.
@HDG,X *** UPDATED AED SINGLE MODULE CROSS REFERENCE ***
@XQT AMES*XREF.XREF
@ADD,E AVFS$$.SAMPLE_STAT
@FIN
```

UNIVAC 1100 AED Reaching Set

```
@RUN Univac run card
@ASG,AX INSERTS.
@FREE AVFS$$.
@ASG,T AVFS$$.
@CAT,P DDPATH.
@ASG,AX DDPATH.
@USE 25.,DDPATH.
@ELT,I AVFS$$.SAMPLE_STAT
    THIS IS THE BEGINNING OF A DUMMY AED STATIC SOURCE FILE
                   :
           AED STATEMENTS WOULD USUALLY BE HERE
                   :
    THIS IS THE END OF THE DUMMY AED STATIC SOURCE FILE
@HDG,X *** UPDATED AED REACHING SET ***
@XQT AMES*REACH.REACH
@ADD,E AVFS$$.SAMPLE_STAT
    5   10
@FLIST DDPATH.
@DELETE,C DDPATH.
@FIN
```

UNIVAC 1100 AED Assertion

```
@RUN Univac run card
@ASG,AX user_login_id*INSERTS.
@ASG,AX user_login_id.
@ASG,T AVFS$$.
@DELETE,C TABFIL.
@CAT,P TABFIL.
@DELETE,C TOKFIL.
@CAT,P TOKFIL.
@DELETE,C INST.
@CAT,P INST.
@ASG,AX TOKFIL.
@USE 20.,TOKFIL.
@ASG,AX INST.
@USE 30.,INST.
@ASG,AX TABFIL.
@USE 25.,TABFIL.
@ELT,I AVFS$$.SAMP_ASSERT
    THIS IS THE BEGINNING OF A DUMMY AED ASSERTED SOURCE FILE
                    :
           AED STATEMENTS WOULD USUALLY BE HERE
                    :
    COMMENT ASSERT* HEIGHT>500;
    COMMENT ASSERT* ALL I ALPHA(1,N) X(I) > Y(I);
                    :
    THIS IS THE END OF THE DUMMY AED ASSERTED SOURCE FILE
@XQT AMES*ASSERT.ASSERT
@ADD,E AVFS$$.SAMP_ASSERT
@COPY,I INST.,user_login_id.INST
@CAPS*CROSS.AEDCAPS,C user_login_id.INST,user_login_id.INST
@CAPS*CROSS.CASM2 user_login_id.INST,user_login_id.SAMPLE_DYN
@DELETE,SC user_login_id.INST
@DATA,L TABFIL.
@END
@CAPS*CROSS.LINK,LTSI ,user_login_id.LINK
         :              :     :        :
         :              :     :        :
         :              :     :        :
          ORIGIN @3B00@,0;
          INCLUDE prog_file(FTCASE);
          INCLUDE prog_file(FTINSERTE);
         ;
         ;
         ; INCLUDE ALL OF THE ROUTINES REQUIRED FOR PROBE, TRACE
         ; AND ASSERTION PROCESSING
         ;
         ;
```

```
          INCLUDE prog_file(ALOOPTOP);  WAITS FOR MODEL DATA TO REFRESH
          INCLUDE prog_file(GOUT);      STANDARD OUTPUT
          INCLUDE prog_file(ASSERT);    LOGICAL ASSERTION VIOLATION ROUTINE
          INCLUDE prog_file(OUTPUT);    TRACE ASSERTION ROUTINE
          INCLUDE prog_file(INPUT);     TRACE ASSERTION ROUTINE
          ORIGIN @79D9@,2;        OFFSET FOR DECISION PATH...TABLE AT 7A00
         ;INCLUDE prog_file(DDPATH);    DECISION PATH ROUTINE
         WAIT          EQU @7000@;  WORD TO WAIT FOR DATA
         ASSERT.CAT    EQU @7001@;  INTERRUPT CATEGORY
         ASSERT.SEL    EQU @7002@;  SELECTED NUMBER OF ASSERTIONS
         ASSERT.PTR    EQU @7003@;  POINTER TO THE LOGICAL ASSERTION TABLE
         ASSERT.BUFFER EQU @7004@;  LOGICAL ASSERTION VIOLATION TABLE
         INPUT.CAT     EQU @7001@;  INTERRPUT CATEGORY
         INPUT.PTR     EQU @7500@;  POINTER TO THE TRACE TABLE
         INPUT.BUFFER  EQU @7501@;  TRACE TABLE
           DDPATH.BUFFER EQU @7A00@;  DDPATH TABLE
         ;
         :             :     :       :
         :             :     :       :
         :             :     :       :
@XQT CAPS*CROSS.HPLDTAPE
user_login_id
LINK
@EOF
@DELETE,SC user_login_id.LINK
@PACK user_login_id.
@FIN
```

UNIVAC 1100 AED Trace

```
@RUN Univac run card
@ASG,AX INSERTS.
@ASG,AX user_login_id.
@ASG,T AVFS$$.
@DELETE,C TABFIL.
@CAT,P TABFIL.
@DELETE,C TOKFIL.
@CAT,P TOKFIL.
@DELETE,C TRACE.
@CAT,P TRACE.
@ASG,AX TOKFIL.
@USE 20.,TOKFIL.
@ASG,AX TRACE.
@USE 30.,TRACE.
@ASG,AX TABFIL.
@USE 25.,TABFIL.
@ELT,I AVFS$$.SAMP_TRACE
    THIS IS THE BEGINNING OF A DUMMY AED TRACED SOURCE FILE
                    :
           AED STATEMENTS WOULD USUALLY BE HERE
                    :
    COMMENT INPUT*BOOLEAN ALIGN;
    COMMENT OUTPUT* LONG TOTAL.TIME;
                    :
    THIS IS THE END OF THE DUMMY AED TRACED SOURCE FILE
@XQT AMES*TRACE.TRACE
@ADD,E AVFS$$.SAMP_TRACE
@COPY,I TRACE.,user_login_id.TRACE
@CAPS*CROSS.AEDCAPS,C user_login_id.TRACE,user_login_id.TRACE
@CAPS*CROSS.CASM2 user_login_id.TRACE,user_login_id.SAMP_TRACE
@DELETE,SC user_login_id.TRACE
@DATA,L TABFIL.
@END
@CAPS*CROSS.LINK,LTSI ,user_login_id.LINK
        :              :      :        :
        :              :      :        :
        :              :      :        :
         ORIGIN @3B00@,0;
         INCLUDE user_login_id(FTCASE);
         INCLUDE user_login_id(FTINSERTE);
        ;
        ;
        ; INCLUDE ALL OF THE ROUTINES REQUIRED FOR PROBE, TRACE
        ; AND ASSERTION PROCESSING
        ;
        ;
```

```
           INCLUDE user_login_id(ALOOPTOP);  WAITS FOR MODEL DATA TO REFRESH
           INCLUDE user_login_id(GOUT);      STANDARD OUTPUT
           INCLUDE user_login_id(ASSERT);    LOGICAL ASSERTION VIOLATION ROUTINE
           INCLUDE user_login_id(OUTPUT);    TRACE ASSERTION ROUTINE
           INCLUDE user_login_id(INPUT);     TRACE ASSERTION ROUTINE
           ORIGIN @79D9@,2;        OFFSET FOR DECISION PATH...TABLE AT 7A00
          ;INCLUDE user_login_id(DDPATH);    DECISION PATH ROUTINE
          WAIT          EQU @7000@;  WORD TO WAIT FOR DATA
          ASSERT.CAT    EQU @7001@;  INTERRUPT CATEGORY
          ASSERT.SEL    EQU @7002@;  SELECTED NUMBER OF ASSERTIONS
          ASSERT.PTR    EQU @7003@;  POINTER TO THE LOGICAL ASSERTION TABLE
          ASSERT.BUFFER EQU @7004@;  LOGICAL ASSERTION VIOLATION TABLE
          INPUT.CAT     EQU @7001@;  INTERRPUT CATEGORY
          INPUT.PTR     EQU @7500@;  POINTER TO THE TRACE TABLE
          INPUT.BUFFER  EQU @7501@;  TRACE TABLE
            DDPATH.BUFFER EQU @7A00@;  DDPATH TABLE
          ;
          :             :     :        :
          :             :     :        :
          :             :     :        :
@XQT CAPS*CROSS.HPLDTAPE
user_login_id
LINK
@EOF
@DELETE,SC user_login_id.LINK
@PACK user_login_id.
@FIN
```

UNIVAC 1100 AED Probe

```
@RUN Univac run card
@ASG,AX INSERTS.
@ASG,T AVFS$$.
@DELETE,C TABFIL.
@CAT,P TABFIL.
@ASG,AX TABFIL.
@USE 25.,TABFIL.
@ELT,I AVFS$$.SAMP_PROBE
    THIS IS THE BEGINNING OF A DUMMY AED PROBED SOURCE FILE
                    :
           AED STATEMENTS WOULD USUALLY BE HERE
                    :
              BEGIN -------|
                    :     |
                    :     |-one loop or path among other paths
                    :     |
              END   -------|
                    :
    THIS IS THE END OF THE DUMMY AED PROBED SOURCE FILE
@ASG,AX user_login_id.
@DELETE,C TOKFIL.
@CAT,P TOKFIL.
@DELETE,C INSTFL.
@CAT,P INSTFL.
@ASG,AX TOKFIL.
@USE 20.,TOKFIL.
@ASG,AX INSTFL.
@USE 30.,INSTFL.
@XQT AMES*PROBE.PROBE
@ADD,E AVFS$$.SAMP_PROBE
@COPY,I INSTFL.,user_login_id.INSTFL
@HDG,X *** UPDATED INSTRUMENTED AED MODULE ***
@CAPS*CROSS.AEDCAPS,C user_login_id.INSTFL,user_login_id.INSTFL
@CAPS*CROSS.CASM2 user_login_id.INSTFL,user_login_id.SAMP_PROBE
@DELETE,SC user_login_id.INSTFL
@DATA,L TABFIL.
@END
@DELETE,C PROBE.
@CAT,P PROBE.
@ASG,AX PROBE.
@USE 3.,PROBE.
@ASG,AX TABFIL.
@USE 2.,TABFIL.
@XQT AMES*DDPATH.DDPATH
@COPY,I PROBE.,user_login_id.PROBE
@DELETE,C PROBE.
```

```
@CAPS*CROSS.AEDCAPS,C user_login_id.PROBE,user_login_id.PROBE
@CAPS*CROSS.CASM2 user_login_id.PROBE,user_login_id.DDPATH
@DELETE,SC user_login_id.PROBE
@CAPS*CROSS.LINK,LTSI ,user_login_id.LINK
          :              :     :        :
          :              :     :        :
          :              :     :        :
           ORIGIN @3B00@,0;
           INCLUDE user_login_id(FTCASE);
           INCLUDE user_login_id(FTINSERTE);
          ;
          ;
          ; INCLUDE ALL OF THE ROUTINES REQUIRED FOR PROBE, TRACE
          ; AND ASSERTION PROCESSING
          ;
          ;
           INCLUDE user_login_id(ALOOPTOP);  WAITS FOR MODEL DATA TO REFRESH
           INCLUDE user_login_id(GOUT);      STANDARD OUTPUT
           INCLUDE user_login_id(ASSERT);    LOGICAL ASSERTION VIOLATION ROUTINE
           INCLUDE user_login_id(OUTPUT);    TRACE ASSERTION ROUTINE
           INCLUDE user_login_id(INPUT);     TRACE ASSERTION ROUTINE
           ORIGIN @79D9@,2;         OFFSET FOR DECISION PATH...TABLE AT 7A00
           INCLUDE user_login_id(DDPATH);    DECISION PATH ROUTINE
          WAIT          EQU @7000@;  WORD TO WAIT FOR DATA
          ASSERT.CAT    EQU @7001@;  INTERRUPT CATEGORY
          ASSERT.SEL    EQU @7002@;  SELECTED NUMBER OF ASSERTIONS
          ASSERT.PTR    EQU @7003@;  POINTER TO THE LOGICAL ASSERTION TABLE
          ASSERT.BUFFER EQU @7004@;  LOGICAL ASSERTION VIOLATION TABLE
          INPUT.CAT     EQU @7001@;  INTERRPUT CATEGORY
          INPUT.PTR     EQU @7500@;  POINTER TO THE TRACE TABLE
          INPUT.BUFFER  EQU @7501@;  TRACE TABLE
            DDPATH.BUFFER EQU @7A00@;  DDPATH TABLE
          ;
          :              :     :        :
          :              :     :        :
          :              :     :        :
@XQT CAPS*CROSS.HPLDTAPE
user_login_id
LINK
@EOF
@DELETE,SC user_login_id.INSTFL,user_login_id.DDPATH,user_login_id.LINK
@PACK user_login_id.
@FIN
```

UNIVAC 1100 AED Symbolic Execution

```
@RUN Univac run card
@ASG,AX INSERTS.
@ASG,T AVFS$$.
@ELT,I AVFS$$.SAMP_SYMBOL
    THIS IS THE BEGINNING OF A DUMMY AED SYMBOLIC SOURCE FILE
                    :
           AED STATEMENTS WOULD USUALLY BE HERE
                    :
    THIS IS THE END OF THE DUMMY AED SYMBOLIC SOURCE FILE
@XQT AMES*SYMEXEC.SYMEXEC
        FOR LINES = 10 .. 20 DO
        LAT.LIM.CMD
        END FOR

@ADD,E AVFS$$.SAMP_SYMBOL
@FIN
```

APPENDIX A

V&V TOOL DEVELOPMENT COMPUTER

The software verification and validation tools for the DFCSVL were developed on a VAX 11/780. The software tools were coded in IFTRAN,* a preprocessor for FORTRAN, featuring structured language capability. Once the IFTRAN source code is processed and the FORTRAN source code created, the FORTRAN source code is passed to the FORTRAN compiler for compiling. Unfortunately, comments that occurred in the structured code are not included in the FORTRAN source code so very little in-code documentation is available at the FORTRAN source level. The FORTRAN source code of the V&V tools was provided under contract, therefore, any modification or maintenance of the V&V tool software must be performed by the developer of the tools (ref. 1).

Preliminary testing of the tools occurred on the VAX. However, since there are differences between the VAX and the Univac 1100, especially in memory allocation and disk usage, care must be used in rehosting the tools onto the Univac 1100.

*IFTRAN is a product of General Research Corporation.

APPENDIX B

REHOSTING THE TOOLS

The comments that occur in this appendix assumes the source code the user is rehosting on the Univac originated on the VAX. A list is presented below of special areas to be aware of and areas where experts may be needed for consultation.

Area	Comments
Source code transfer	Create a VAX source tape compatible with the Univac 1100 computer. Recommended tape characteristics: tape density (1600 bpi or higher) 9 track tape unlabelled tape source code ASCII format one module per file or one tool per file. Modules should be separated from each other if there is one tool per file. This is to ease in editing the file and separating modules.
Logical unit assignments	Make sure unit assignments are compatible with the Univac environment. Logical units for reports may need changing, such as PRTSYM routine.
Disk usage	Several routines in the tools require careful usage. OPNINS enable FORTRAN function call FAC2SF. If no OPNINS routine, then the call will reside in the LRLEX routine. NITTOK OPEN statement must be properly set, else execution of the tool may be EXCESSIVE.
Memory Allocation	Some tools may require special handling of memory to successfully execute. The tools may be very large when instruction and data areas are combined resulting in aborting the load of the tool. Solution: Optimization of code. If optimization does not solve load aborts;then overlays may be required.

Dynamic AED V&V tool routines

ASSERT, PROBE and TRACE tools require and build dynamic AED V & V tool routines. These routines need to be compiled and assembled by the AED processors so that the object codes will be available for the link loader. The list below provides information regarding these routines.

Subroutine	Comments
DDPATH	Generated by the PROBE tool.
INPUT.TRACE	AED V&V TRACE tool routine. See figure B1.
OUTPUT.TRACE	AED V&V TRACE tool routine. See figure B2.
ASSERT.FALSE	AED V&V ASSERTION tool routine. See figure B3.
GOUT	AED Standard output routine. DDPATH references GOUT. See figure B4.

```
BEGIN
DEFINE PROCEDURE INPUT.TRACE(MODULE.INDEX, VAR.INDEX, VAR.TYPE,
                             SHORT.VALUE, LONG.VALUE)
WHERE INTEGER MODULE.INDEX, VAR.INDEX, VAR.TYPE, SHORT.VALUE;
      LONG LONG.VALUE TOBE
     BEGIN
        LONG      TOTAL.TIME ;
        LONG      ARRAY INPUT.BUFFER(400) ;
        INTEGER   INPUT.PTR ;
        INTEGER   PTR.PRIME ;
        EXTERNAL  TOTAL.TIME ;
        EXTERNAL  INPUT.PTR ;             ... LOAD AT @7500@  //
        EXTERNAL  INPUT.BUFFER;           ... LOAD AT @7501@  //
        EXTERNAL  PTR.PRIME  ;

        LONG      DUMMY ;
        INTEGER   ARRAY SHORT(1) ;
        SHORT $=$ DUMMY ;

COMMENT **** IF MORE THAN 40 ENTRIES, THEN BIAS COUNTER VALUE BY 100 **;
          PTR.PRIME=IF INPUT.PTR GRT 40 THEN INPUT.PTR-100 ELSE INPUT.PTR;

        INPUT.BUFFER(PTR.PRIME * 5 + 0) = MODULE.INDEX  ;
        INPUT.BUFFER(PTR.PRIME * 5 + 1) = VAR.INDEX     ;
        INPUT.BUFFER(PTR.PRIME * 5 + 2) = 0             ;
        INPUT.BUFFER(PTR.PRIME * 5 + 3) = TOTAL.TIME    ;

        IF VAR.TYPE == 1 THEN
           BEGIN
           SHORT(0) = SHORT.VALUE;
           SHORT(1) = 0
           END
           ELSE
           DUMMY = LONG.VALUE;

        INPUT.BUFFER(PTR.PRIME * 5 + 4) = DUMMY ;

        INPUT.PTR = IF PTR.PRIME  EQL 39 THEN  100
        ELSE INPUT.PTR + 1;
     END
END FINI
```

Figure B1.- Subroutine INPUT.TRACE.

```
BEGIN
DEFINE PROCEDURE OUTPUT.TRACE(MODULE.INDEX, VAR.INDEX, VAR.TYPE,
                              SHORT.VALUE, LONG.VALUE)
WHERE INTEGER MODULE.INDEX, VAR.INDEX, VAR.TYPE, SHORT.VALUE;
     LONG LONG.VALUE TOBE
     BEGIN
        LONG      TOTAL.TIME ;
        LONG      ARRAY INPUT.BUFFER(400) ;
        INTEGER   INPUT.PTR ;
        INTEGER   PTR.PRIME ;
        EXTERNAL  TOTAL.TIME ;
        EXTERNAL  INPUT.PTR ;              ... LOAD AT @7500@  //
        EXTERNAL  INPUT.BUFFER;            ... LOAD AT @7501@  //
        EXTERNAL  PTR.PRIME  ;

        LONG      DUMMY ;
        INTEGER   ARRAY SHORT(1) ;
        SHORT $=$ DUMMY ;

COMMENT **** IF MORE THAN 40 ENTRIES, THEN BIAS COUNTER VALUE BY 100 **;
        PTR.PRIME=IF INPUT.PTR GRT 40 THEN INPUT.PTR-100 ELSE INPUT.PTR;

        INPUT.BUFFER(PTR.PRIME * 5 + 0) = MODULE.INDEX   ;
        INPUT.BUFFER(PTR.PRIME * 5 + 1) = VAR.INDEX      ;
        INPUT.BUFFER(PTR.PRIME * 5 + 2) = 1              ;
        INPUT.BUFFER(PTR.PRIME * 5 + 3) = TOTAL.TIME     ;

        IF VAR.TYPE == 1 THEN
           BEGIN
           SHORT(0) = SHORT.VALUE;
           SHORT(1) = 0
           END
           ELSE
           DUMMY = LONG.VALUE;

        INPUT.BUFFER(PTR.PRIME * 5 + 4) = DUMMY ;

        INPUT.PTR = IF PTR.PRIME  EQL 39 THEN  100
        ELSE INPUT.PTR + 1;
     END
END FINI
```

Figure B2.- Subroutine OUTPUT.TRACE.

```
BEGIN
DEFINE PROCEDURE ASSERT.FALSE (MODULE.INDEX, LINE.INDEX)
WHERE INTEGER MODULE.INDEX, LINE.INDEX TOBE
     BEGIN
     BOOLEAN  ASSERT.FLAG ;
     LONG     TOTAL.TIME ;
     INTEGER  DO.WO ;
     INTEGER  DO.BUFF.0 ;
     INTEGER  ASSERT.SEL ;
     INTEGER  ASSERT.PTR ;
     INTEGER  ASSERT.CAT ;
     INTEGER  ARRAY ASSERT.BUFFER(160) ;
     EXTERNAL TOTAL.TIME ;
     EXTERNAL DO.BUFF.0 ;                         ... LOAD AT @370B@  //
     EXTERNAL DO.WO ;                             ... LOAD AT @F2E1@  //
     EXTERNAL ASSERT.CAT ;                        ... LOAD AT @7001@  //
     EXTERNAL ASSERT.SEL ;                        ... LOAD AT @7002@  //
     EXTERNAL ASSERT.PTR ;                        ... LOAD AT @7003@  //
     EXTERNAL ASSERT.BUFFER;                      ... LOAD AT @7004@  //
     ASSERT.CAT  =  1   ;                 ... IMPLIES ASSERT VIOLATED //
     ASSERT.BUFFER(ASSERT.PTR * 3 + 0) = MODULE.INDEX  ;
     ASSERT.BUFFER(ASSERT.PTR * 3 + 1) = LINE.INDEX    ;
     ASSERT.BUFFER(ASSERT.PTR * 3 + 2) = TOTAL.TIME    ;
     ASSERT.PTR = ASSERT.PTR + 1;
     IF ASSERT.SEL <= 0 THEN ASSERT.SEL = 30 ;
     IF ASSERT.PTR >= ASSERT.SEL THEN
        BEGIN
             ASSERT.FLAG = TRUE ;
             ASNBIT (ASSERT.FLAG,11,DO.BUFF.0);   ... FOR A CHANNEL //
             DO.WO = DO.BUFF.0 ;
             WHILE ( ASSERT.CAT  NEQ -1 ) DO ;  ... BUSY LOOP  //
             ASSERT.CAT = 0 ;
             ASSERT.FLAG = FALSE ;
             ASNBIT (ASSERT.FLAG,11,DO.BUFF.0);   ... FOR A CHANNEL //
             DO.WO = DO.BUFF.0 ;
             ASSERT.PTR = 0;
        END
     END
END FINI
```

Figure B3.- Subroutine ASSERT.FALSE.

```
BEGIN
DEFINE PROCEDURE GOUT(STRING,VALUE)
     WHERE INTEGER STRING, VALUE TOBE
     BEGIN
INTEGER   ERRORS  ;EXTERNAL     ERRORS;
PRESET     ERRORS = 0;
ERRORS = ERRORS + 1
     END;
END FINI
```

Figure B4.- Subroutine GOUT.

APPENDIX C

PDP-11/60 INTERFACE HANDLERS

The PDP-11/60 Interface Handlers are a series of UNIX shell programs which make the utilization of the V&V tools "user friendly". These Interface Handlers help the user to prepare the Univac run-stream for a particular tool in the proper sequence, contents, and options of the Univac control commands. Consequently, the user needs to become familiar with the UNIX commands and the V&V tool capabilities.

The following information in this appendix is intended for the person who is establishing the V&V tool capability on their environment computer. The source for the Interface Handlers are written in C programming language and under the source code control system as a means of maintaining source code configuration control. Since most of the source code for these handlers is relatively long, source listings are not included in this document. Source code listings can be requested from the author. A flow diagram of a handler is presented in figure C1.

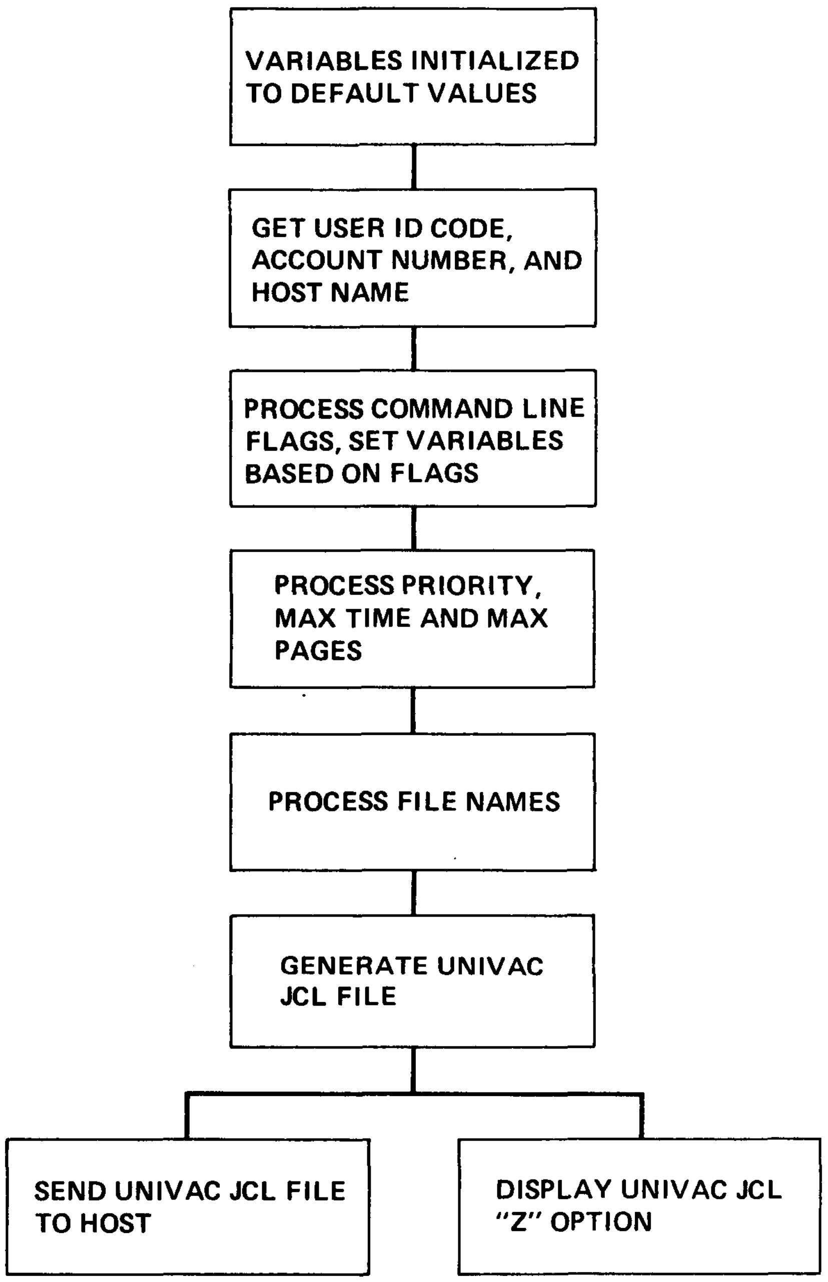
VARIABLES INITIALIZED TO DEFAULT VALUES
GET USER ID CODE, ACCOUNT NUMBER, AND HOST NAME
PROCESS COMMAND LINE FLAGS, SET VARIABLES BASED ON FLAGS
PROCESS PRIORITY, MAX TIME AND MAX PAGES
PROCESS FILE NAMES
GENERATE UNIVAC JCL FILE
SEND UNIVAC JCL FILE TO HOST
DISPLAY UNIVAC JCL "Z" OPTION

Figure C1.- Flow diagram for interface handlers.

REFERENCES

1. Saib, S. H.: Automated Verification of Flight Software - User's Manual. NASA CR-166346, 1982

2. de Feo, P.; Doane, D.; and Saito,J.: An Integrated User-Oriented Laboratory for Verification of Digital Flight Control Systems--Features and Capabilities. NASA TM 84276, 1982.

3. McLees, R. E.: Analysis of Verification Tools for Digital Flight Control Systems (DFCS) Software. NASA CR-177391, Feb. 1986.

1. Report No. NASA TM 88313	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle DIGITAL-FLIGHT-CONTROL-SYSTEM SOFTWARE WRITTEN IN AUTOMATED-ENGINEERING-DESIGN LANGUAGE: A USER'S GUIDE OF VERIFICATION AND VALIDATION TOOLS		5. Report Date January 1987 6. Performing Organization Code
7. Author(s) Jim Saito		8. Performing Organization Report No. A-86282
9. Performing Organization Name and Address Ames Research Center Moffett Field, CA 94035		10. Work Unit No. 11. Contract or Grant No.
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546		13. Type of Report and Period Covered Technical Memorandum 14. Sponsoring Agency Code 505-66-21

15. Supplementary Notes

Point of Contact: Jim Saito, M/S 210-9, Ames Research Center, Moffett Field, CA 94035, (415)694-5048 or FTS 464-5048

16. Abstract

The user's guide of verification and validation (V&V) tools for the Automated-Engineering-Design (AED) language is specifically written to update the information found in several documents pertaining to the automated verification of flight software tools. The intent of this document is to provide, in one document, all the information necessary to adequately prepare a run to use the AED V&V tools. No attempt is made to discuss the FORTRAN V&V tools since they were not updated and are not currently active. Additionally, this document contains the current descriptions of the AED V&V tools and provides information to augment the NASA TM 84276 entitled "An Integrated User-Oriented Laboratory for Verification of Digital-Flight-Control Systems--Features and Capabilities."

The AED V&V tools are accessed from the digital-flight-control-systems verification laboratory (DFCSVL) via a PDP-11/60 digital computer. The AED V&V tool-interface handlers on the PDP-11/60 generate a Univac run stream which is transmitted to the Univac via a Remote Job Entry (RJE) link. Job execution takes place on the Univac 1100 and the job output is transmitted back to the DFCSVL and stored as a PDP-11/60 printfile.

17. Key Words (Suggested by Author(s)) Verification of digital flight control system Flight computer AED computer language	18. Distribution Statement Unclassified - Unlimited Star Category: 08

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 93	22. Price* A05

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