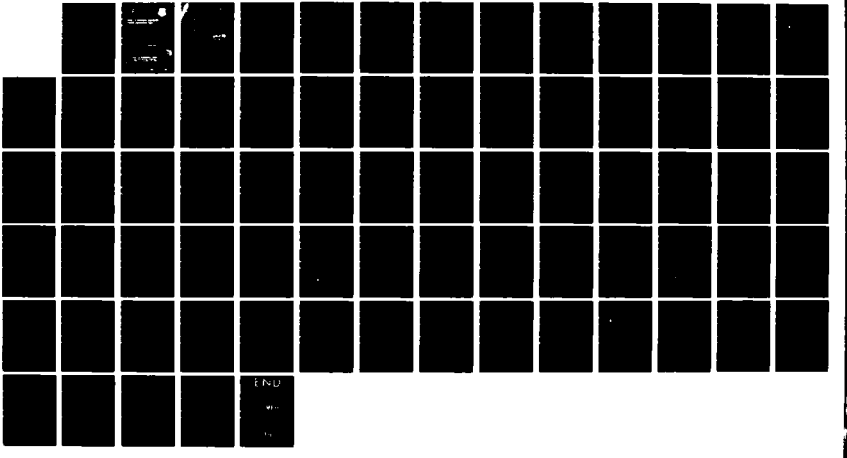
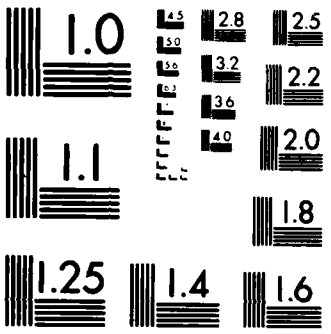
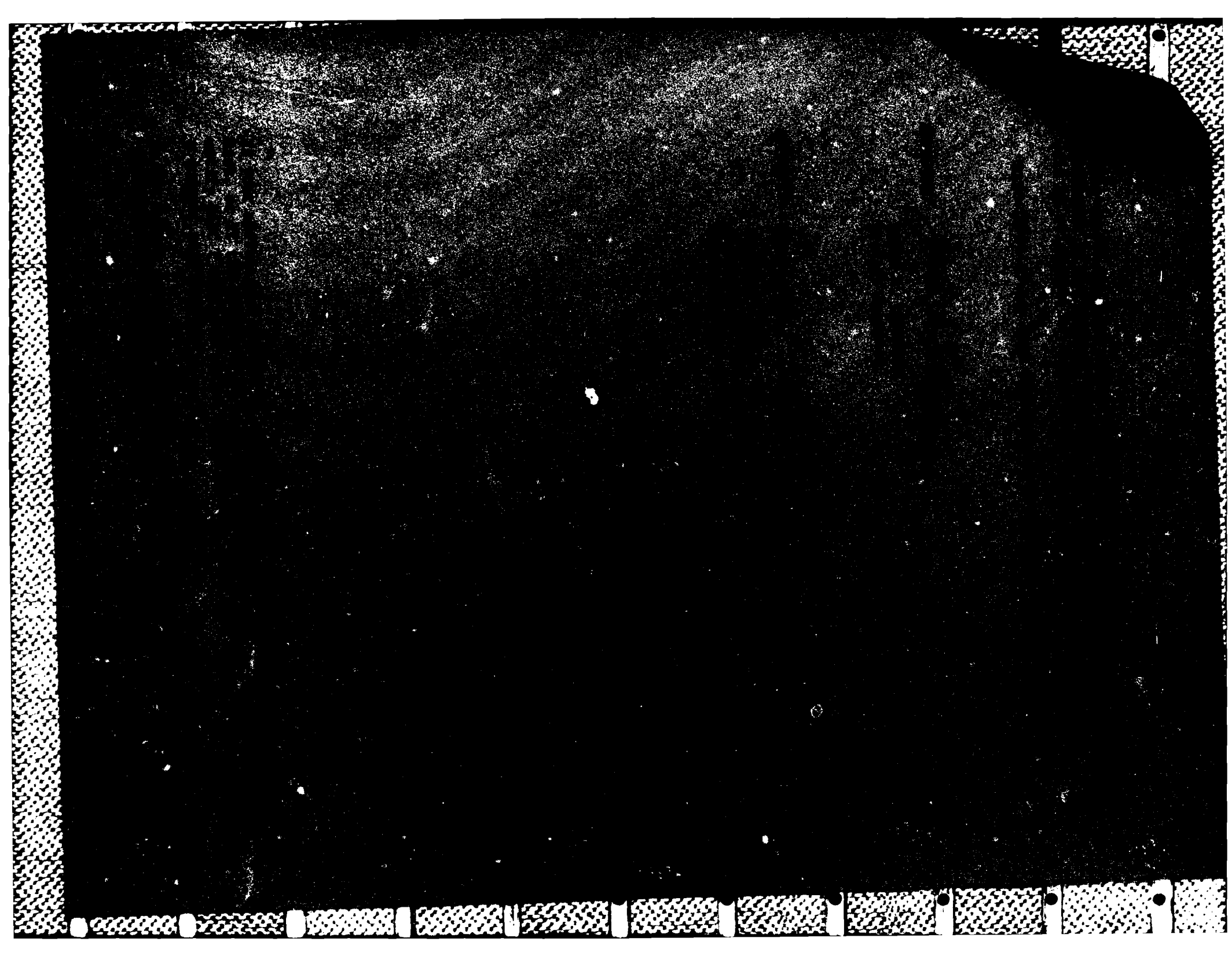
AD-A148 732 JOVIAL J73 AUTOMATED VERIFICATION SYSTEM: IMPLEMENTATION PHASE(U) GENERAL RESEARCH CORP SANTA BARBARA CA C GANNON MAR 84 GRC-CR-9-947 RADC-TR-84-33 1/1

UNCLASSIFIED F30602-79-C-0265 F/G 9/2 NL







UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

AD-A148732

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) CR-9-947	5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-84-33
6a. NAME OF PERFORMING ORGANIZATION General Research Corporation 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center
6c. ADDRESS (City, State and ZIP Code) P O Box 6770 Santa Barbara CA 93111	7b. ADDRESS (City, State and ZIP Code) Griffiss AFB NY 13441
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Rome Air Development Center 8b. OFFICE SYMBOL (If applicable) COEE	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F30602-79-C-0265
8c. ADDRESS (City, State and ZIP Code) Griffiss AFB NY 13441	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO. 63278F; PROJECT NO. 2532; TASK NO. 02; WORK UNIT NO. 06

11. TITLE (Include Security Classification)
JOVIAL J73 AUTOMATED VERIFICATION SYSTEM - IMPLEMENTATION PHASE

12. PERSONAL AUTHOR(S)
Carolyn Gannon

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
Final	FROM May80 TO Sep83	March 1984	86

16. SUPPLEMENTARY NOTATION
N/A

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB. GR.	Verification; Static testing; Software testing; Dynamic testing; JOVIAL J73
09	02		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The JOVIAL J73 Automated Verification System was developed under sponsorship by the Rome Air Development Center (RADC) at Griffiss AFB NY, under Contract F30602-79-C-0265.

The resulting tool, (J73AVS) is an interactive computer program that analyzes any source code written in JOVIAL J73. The primary objective of J73AVS is to provide static and dynamic testing assistance. To do this, J73AVS detects a variety of semantic and data flow errors, reports execution coverage (i.e., frequency of statements, control branches, and procedures executed by each test case), reports execution timing (in terms of CPU milliseconds) for procedures or certain user-designated portions of the JOVIAL J73 program, reports execution tracing (ordering) of control branches or procedures, and keeps track of the test coverage history from run to run. To assist with function testing of programs, J73AVS provides a local assertion construct which triggers an output message when it is violated.

(See reverse)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code)	22c. OFFICE SYMBOL
Frank S. LaMonica	(315) 330-3977	RADC (COEE)

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

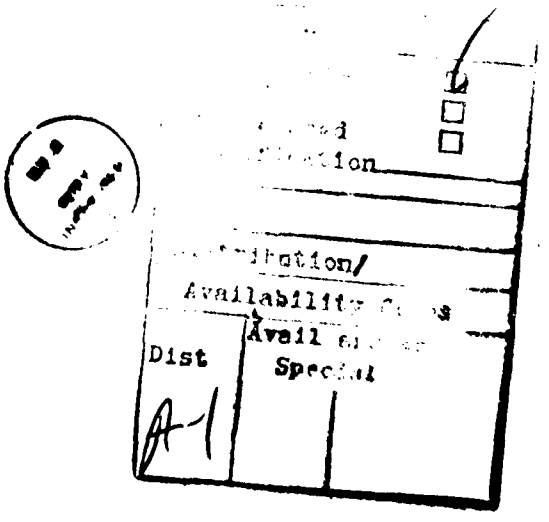
UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

J73AVS also provides a large number of source analysis reports, ranging from symbol cross references and descriptions of symbol usages to procedure calling trees and descriptions of all procedures being analyzed. These reports are very useful for original code development, rehosting "foreign" code, debugging and testing, code enhancement, and code maintenance.

This report describes the features of J73AVS in a brief manner, the history of the project, documents and technical papers resulting from the project, and specific details on how to obtain the J73AVS software and selected project reports. Also included are implications on further research in software engineering as it pertains to J73AVS.

CONTENTS



FIGURES

TABLES

1 INTRODUCTION

The JOVIAL J73 Automated Verification System was developed under sponsorship by the Rome Air Development Center (RADC) at Griffiss Air Force Base, NY, under Contract F30602-79-C-0265.

The resulting tool, J73AVS, is an interactive computer program that analyzes any source code written in JOVIAL J73. The primary objective of J73AVS is to provide static and dynamic testing assistance. To do this, J73AVS detects a variety of semantic and data flow errors, reports execution coverage (i.e., frequency of statements, control branches, and procedures executed by each test case), reports execution timing (in terms of CPU milliseconds) for procedures or certain user-designated portions of the JOVIAL J73 program, reports execution tracing (ordering) of control branches or procedures, and keeps track of the test coverage history from run to run. To assist with functional testing of programs, J73AVS provides a local assertion construct which triggers an output message when it is violated.

J73AVS also provides a large number of source analysis reports, ranging from symbol cross references and descriptions of symbol usages to procedure calling trees and descriptions of all procedures being analyzed. These reports are very useful for original code development, rehosting "foreign" code, debugging and testing, code enhancement, and code maintenance.

J73AVS features and its ease of operation are described more fully in Sec. 2. Since the contract has had two major phases (test tool study and tool implementation), the project history is covered in Sec. 3. J73AVS availability and installation configuration are discussed in Sec. 4. Each deliverable document is briefly described in Sec. 5 along with information concerning how to obtain each report.

Following the overview of J73AVS capabilities in Sec. 2 is a discussion in Sec. 6 of possible future research regarding J73AVS. Three conference papers resulted from the development of J73AVS. These are reprinted in

Appendix A. Terms and abbreviations common to J73AVS are provided in Appendix B. J73AVS operates interactively, driven by simple user commands. The commands that direct J73AVS to either perform some activity or to report specific results are given in Appendix C.

2 OVERVIEW OF CAPABILITIES

J73AVS can operate in either batch or interactive mode. The user directs J73AVS through a command language; each major processing activity has a command keyword associated with it and command parameters are used to specify processing or reporting options. The type of processing and its general sequence are shown in Fig. 2.1. While the flowchart in Fig. 2.1 shows a number of processing activities, J73AVS performs a number of them collectively when a user gives a J73AVS command. Prompts are output by J73AVS if commands are incorrect, out of order, or prerequisite information is needed. J73AVS process and reporting commands are given in App. C.

2.1 J73AVS OPERATION

One or more JOVIAL J73 source modules (compilation units) or J73 COPY texts are supplied to J73AVS as input. There are no restrictions on the length of modules. Up to 250 modules can be stored on the J73AVS database. The source text must be in the same form as if it were being input to the JOVIAL J73 compiler. Job control setups are provided in the User's Manual (Appendix B for the IBM and in Appendix C for the VAX).

The source text for an entire program need not be supplied to J73AVS for static and data flow processing and for procuring automated reports. Some modules may be supplied as stubs (e.g., module heading, possibly some logical assertions describing the function, and a return) or not at all. J73AVS builds a database of module interface information, which can be saved and re-used from run-to-run during program development or testing. Therefore, J73AVS can be used to incrementally build and test software. Updated modules automatically replace old ones on the J73AVS database. This database is highly compressed, even though it contains a wealth of information about the user's source code. The saved database is approximately three times the size of the user's source code file.

Execution analysis by J73AVS requires instrumenting the source text, which J73AVS performs automatically when the user requests it. Anywhere from one J73 procedure to all the source modules available on the database can be

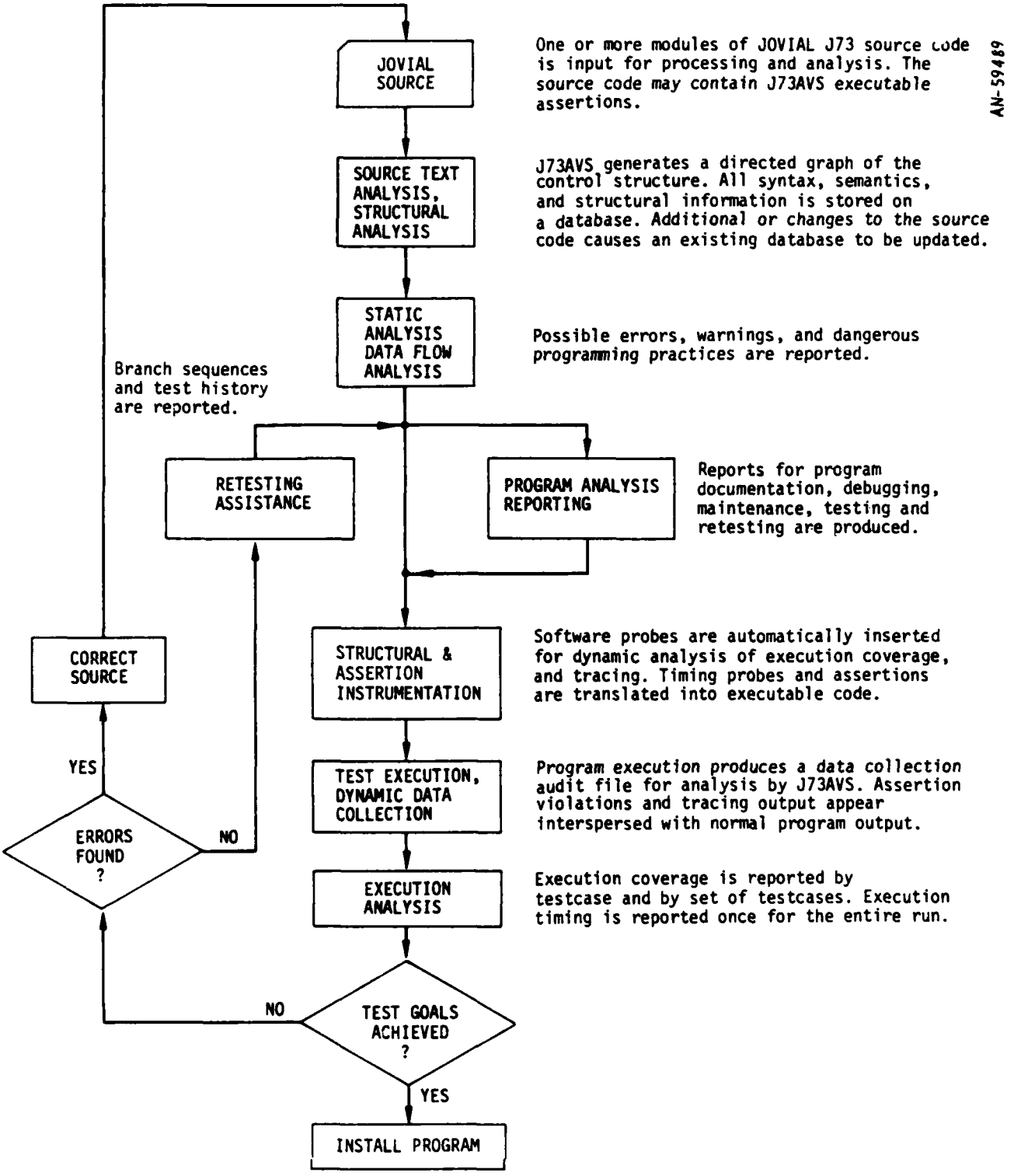


Figure 2.1. Overview of J73AVS

instrumented. J73AVS outputs the instrumented source as a complete module, ready for input to a JOVIAL J73 compiler. To obtain J73AVS output for assertion violations, timing, coverage, or tracing, the program (with some or all modules instrumented) is executed with test data supplied by the user. Programs instrumented for tracing or assertions will produce short messages from J73AVS, interspersed with the program's own output, during execution. For coverage and timing analysis, program execution in the J73AVS environment differs from normal program execution in the following ways: (1) one or more modules have been instrumented before compilation, (2) a small J73AVS data collection routine is loaded before execution, and (3) a J73AVS audit file is written during execution. This file is used by J73AVS to produce reports for coverage and timing analysis. The J73AVS database is not used during the program's execution. These major functions and data files are illustrated in Fig. 2.2.

Although J73AVS exists as a single program, it is best considered as a collection of tools that interact with the user. Some of the facilities, such as automated documentation, static error reporting, and instrumentation are completely automated and require only that the user initiate the tasks. Other processes, such as execution-time data collection or retesting assistance, require more information. The user must provide test data and select the test targets.

J73AVS provides detailed information about the static and execution-time behavior of the program. The user directs the processing performed by J73AVS, analyzes the output produced by J73AVS, and determines subsequent action.

2.2 ROLE IN THE SOFTWARE DEVELOPMENT CYCLE

The role of J73AVS in the software development cycle is to provide automated assistance wherever possible during program development: coding, debugging, testing, maintenance, and retesting. J73AVS users play an active part in the cycle, as shown in Fig. 2.3. This figure breaks up the development cycle and shows the flow of control and information between J73AVS and the user.

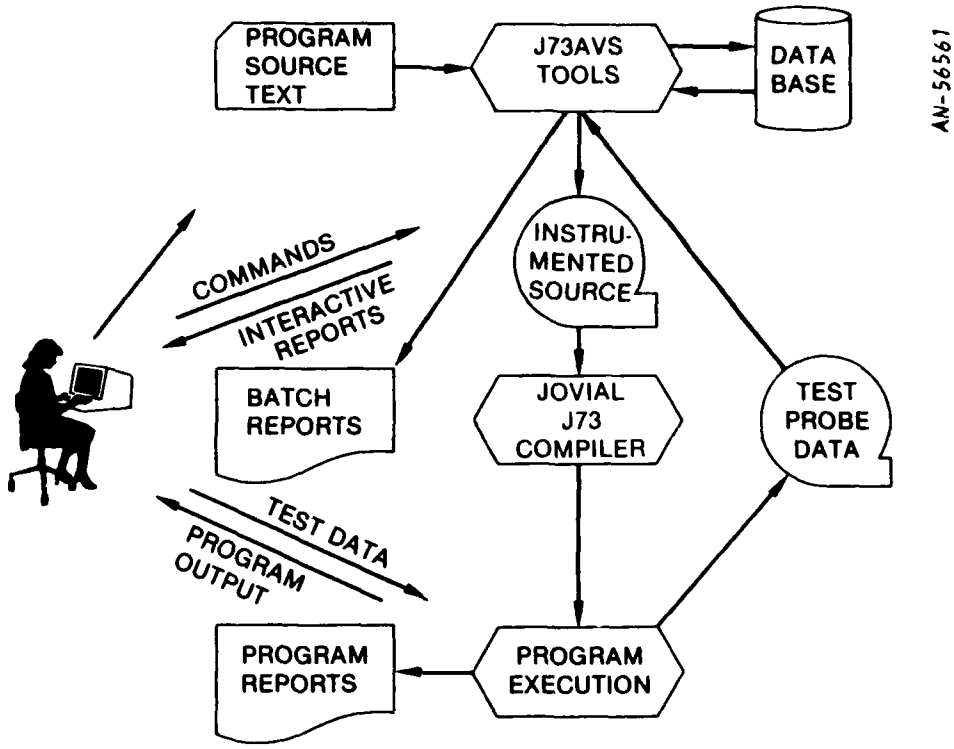


Figure 2.2. J73AVS Interaction with User

Using Fig. 2.3 as a basis, a typical sequence of J73AVS-supported operations is:

1. JOVIAL J73 source text, perhaps with assertions, is read by J73AVS as one or more compilable modules.

2. J73AVS produces program analysis reports showing control structure, symbol usage, calling hierarchy, etc., as well as a static analysis report showing errors and dangerous programming practices.

3. Using the reports as a guide, the source modules are changed or new modules are added to the program.

4. J73AVS reports the interaction of the new or changed modules with the rest of the program. This information, in turn, may show the need to modify other modules.

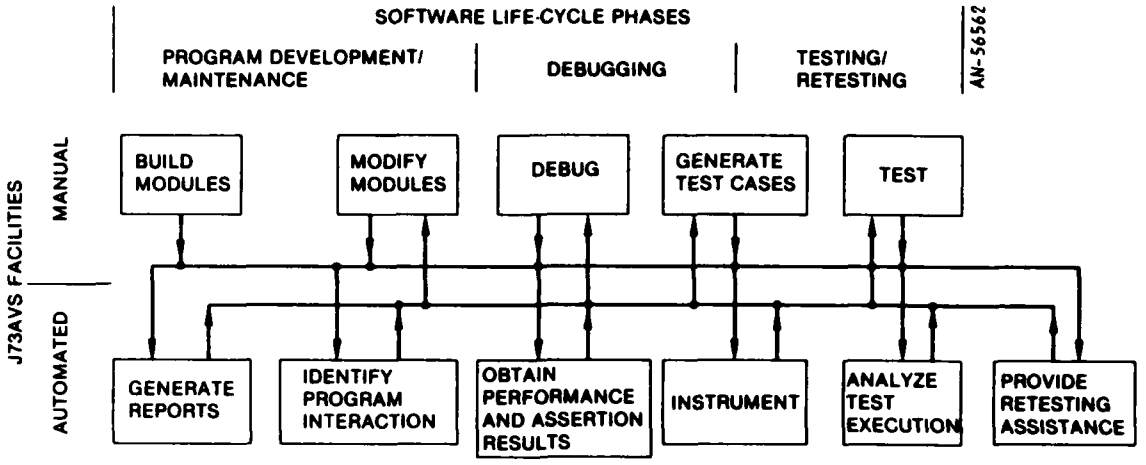


Figure 2.3. Role of J73AVS in the Software Development Cycle

5. For debugging, the program is instrumented by J73AVS and executed with an initial test case supplied by the user.

6. Assertion messages, variable and module tracing, and execution timing reports can be used for debugging.

7. Using the J73AVS reports, the user chooses to create more test data or instrument other modules.

8. For testing, the same cycle of instrumentation and execution is repeated, but for a different goal: rather than detecting and locating errors, testing aims to demonstrate that the entire program has been exercised to some degree. The J73AVS execution analysis reports show the thoroughness of execution coverage.

9. The user evaluates execution coverage reports, the program's own execution results, and the program specification to determine if testing is complete.

10. J73AVS provides branch sequence information to retest targets chosen by the user. A test history of execution coverage assists the user in choosing targets for retesting.

Program Development and Maintenance

Executable assertions provide a means for a programmer to specify expected behavior. Logical condition assertions can be used for reporting execution-time exceptions, stress testing, and manual or automated test data generation. When assertions are left as comments in the source code they can be used as inline documentation of the program's specifications.

To assist with reliable system development and maintenance, J73AVS provides substantial program analysis reporting on structural hierarchy, symbol usage, invocations, certain J73 constructs, and system characteristics. The user has control over obtaining high- or low-level information through the tool's command language.

Debugging

Normal compilation using JOVIAL J73 compilers can detect many syntax and semantic errors. Other errors, such as uninitialized variables, possible infinite loops, unreachable code, certain improper constructs, and dangerous coding practices (like transferring into CASE or IF statements) will be reported by J73AVS. The user can specify the degree of analysis to the error, warning, or message level.

Debugging is also supported by assertion exceptions, variable and module execution tracing, and execution timing reports. When the program's execution behavior deviates from the acceptable logical behavior as specified by the assertions, it is immediately reported in the program's output. The user-embedded assertions have no effect on program control flow until they are violated; at that time the violation is reported with the source statement number of the assertion.

Testing

The primary purpose of program coverage analysis is to provide a measure of the level of testing. This measuring technique uses the program's control structure as a guide. Structure-based testing means that the program's control structures are analyzed for execution behavior; that is,

whether the structures are exercised and in what order. Structure-based testing can uncover errors due to untested branches or improper sequences of branches. J73AVS provides program unit or branch tracing and analyzes execution coverage of program units, branches, and statements. Further, J73AVS assembles the timing information from program unit tracing and user-directed timing probes into an execution timing report.

Retesting Assistance

Retesting software is performed when analysis shows that prior testing is inadequate (insufficient branch coverage, not all functions demonstrated, etc.) or when program changes have taken place. The proper approach to take in retesting is highly dependent upon the characteristics of the program being tested as well as the measures being used to evaluate testing completeness. Section 2 of the User's Manual provides a methodology for testing and re-testing software for the purpose of improving structural-testing completeness.

To determine the sequences of branches which lead to an untested branch or statement, the user can request that the "reaching set" be computed between two specified statements (or from the program unit's entry). After the flow of control is identified by J73AVS, the user can backtrack through the program to the actual test data. New test data can be created by using J73AVS module interaction, invocation, and execution coverage reports. Unfortunately, automatic test data generators which use symbolic execution are not yet general enough, easy to use, or reliable. Therefore, J73AVS has no test data generation capability at this time.

The testing history maintained by J73AVS is useful in attaining coverage testing goals and for determining targets for retesting. Program unit and coverage information is saved in a concise way for each test case. The results of subsequent execution runs can be added, providing a cumulative report of all tests.

3 PROJECT HISTORY

3.1 PROJECT GOALS

The J73AVS contract was awarded in September 1979 during the period of final JOVIAL J73 language definition.[1,2] The primary goals of the project were to identify useful automated testing techniques for the new dialect of JOVIAL and to develop an automated tool that incorporated the best and most feasible of the identified techniques. The intent was to make such a tool available to Air Force contractors shortly after the JOVIAL J73 compilers became available. Therefore, the project was in two parts: (1) a six-month study phase and (2) a seventeen-month implementation phase (separated by a short project review period).

3.2 PROJECT DELIVERABLES

The basic contract called for delivery of J73AVS, written in JOVIAL J73, on two computers: the ITEL AS/5-3 (OS/MVS) at the Aeronautical Systems Division Computer Center (ASD/AD) at Wright-Patterson Air Base and the DEC20 (TOPS20) at RADC. Modifications to the contract resulted in one delivery of J73AVS written in structured FORTRAN first, then in JOVIAL J73, on the Amdahl 470 (TSO, OS/MVS) at ASD/AD, a partial delivery on the DEC20 at RADC, and a delivery in structured FORTRAN on the VAX 11/780 (VMS) at ASD/AD.

Two sets of J73AVS training classes were given at ASD/AD: one for the Amdahl[3] (the IBM 370-equivalent version of J73AVS) and one for the VAX (the VAX version of J73AVS). Both fully delivered versions of J73AVS underwent formal acceptance testing. The VAX version included a six-month maintenance period. Both versions of J73AVS are functionally equivalent.

---

[1] MIL-STD-1589A, Military Standard JOVIAL J73, March 15, 1979.

[2] MIL-STD-1589B, Military Standard JOVIAL J73, June 6, 1980.

[3] The Amdahl computer at ASD was later replaced by an NAS 7000 (another IBM 370-equivalent computer).

3.3 PROJECT ACTIVITIES

The primary project activities can be highlighted as follows:

- Sept. 1979 - Study phase began.
- Jan. 1980 - Preliminary J73AVS design briefing given at the JOVIAL User's Group. Solicited input on desirable J73AVS capabilities.
- March 1980 - Study phase reports[1] delivered:
  1. Functional Description
  2. System/Subsystem Specification
  3. Final Report
- May 1980 - Implementation phase began.
- July 1980 - Gave status briefing at the JOVIAL User's Group. Solicited input on what J73AVS analysis is desired for the J73 DEFINE construct and how J73 programmers plan to handle input and output.
- August 1980 - Contract modified to add J73AVS analysis of MIL-STD-1589B.
- October 1981 - Test plan delivered for the Amdahl host.
- Nov. 1981 - Formal acceptance testing and training course given on the Amdahl at ASD/ADOL. User's Manual delivered.
- June 1982 - Contract was modified to translate J73AVS from structured FORTRAN to JOVIAL J73 on the Amdahl and complete a data flow analysis capability.
- Sept. 1982 - Contract modified to add structured FORTRAN version of J73AVS on the VAX 11/780, along with six months of maintenance.

---

[1] All reports are listed in Sec. 5.

- Nov. 1982 - Formal acceptance testing on the J73 Amdahl version of J73AVS. DEC20 version partially installed.
- Dec. 1982 - Program Specification and Program Maintenance manuals delivered. Final Amdahl J73AVS User's Manual delivered.
- March 1983 - VAX J73AVS installation completed. Acceptance testing and training performed at ASD/AD. New User's Manuals issued containing both IBM-equivalent and VAX operating instructions for J73AVS.
- May 1983 - VAX J73AVS Program Maintenance Manual delivered.
- Sept. 1983 - End of contract. Final VAX and Amdahl versions delivered to ASD/ADOL. Final Report delivered. Final Program Specification delivered.

3.4 PROJECT ADMINISTRATION

The contract has been sponsored and administered by RADC, with project monitoring performed by Frank S. LaMonica. VAX rehosting and maintenance were funded by the Embedded Computer Standardization Program Office (ASD/AXS).

4 J73AVS AVAILABILITY

J73AVS is available in source or binary object form from the Language Control Facility (LCF) at Wright-Patterson Air Force Base. The J73AVS User's Manual, Program Maintenance Manual (both described in Section 5), and a version description document are also available through the LCF. J73AVS software is distributed on 9-track, 1600 bits-per-inch magnetic tape. The LCF point of contact is:

Georgeanne Chitwood
ASD/ADOL
WPAFB, Ohio 45433
513-255-4472
AV785-4472

The LCF can provide distribution tapes for either the VAX or IBM 370-equivalent versions of J73AVS. A summary of each installation configuration is provided in Table 4.1 for the VAX and Table 4.2 for the IBM.

TABLE 4.1

J73AVS VAX INSTALLATION AT WRIGHT-PATTERSON

Date	September 27, 1983
Version	VF092783
Computer	VAX 11/780
Operating System	VMS 3.2
Source Language	JTRAN (Structured FORTRAN)
Preprocessor	JTRAN (generates ANSI FORTRAN-66)
Compiler	VAX-11 FORTRAN V3.3-45
Non-Standard Features	None
Configuration	ANSI FORTRAN 66 Internal J73AVS assertions disabled
Memory Requirements	Depends on host VAX system parameters. A host with a minimum of 1MB of core is recommended for reasonable performance.
Executable Task Image Size	J73AVS.EXE needs approximately 800 disk blocks

TABLE 4.2

J73AVS IBM-EQUIVALENT INSTALLATION AT WRIGHT-PATTERSON

Date	September 27, 1983
Version	IBMJ092783
Computer	NAS 7000 (IBM 370 equivalent)
Operating System	Multiple Virtual System (MVS) 3.8
Source Language	MIL-STD 1589B JOVIAL/J73 (utility routines in FORTRAN and assembler)
Preprocessor	None
Compilers	JOVIAL/J73 Version R0201-03.000 (040183) FORTRAN H Extended IBM Assembler
Non-Standard Features	FORTRAN I/O BAL PDS-member access system routines (provided by SofTech, Inc.) JOVIAL J73 PDS-member access utilities BAL CPU-clock access routines
Configuration	Batch, Interactive Non-overlayed JOVIAL/J73 code is not optimized
Core Memory Requirements:	954K Bytes (non-overlayed, non-optimized)
Executable Load Module:	A790684.J73AVS.LOADMOD(J73AVS)

5 RELATED DOCUMENTS

This section contains descriptive information on each documentation item that was delivered under the contract. The format for this information is the report title, abstract or purpose of the document, followed by the report's table of contents. For the Training Course workbook, the course agenda is provided as a description.

1. Title: JOVIAL J73 Automated Verification System Functional Description

Report No.: CR-1-947

Authors: C. Gannon, N.B. Brooks

Date: March 1980

Abstract:

This report describes the functions of an Automated Verification System (AVS) for the JOVIAL J73 computer language, from the point of view of the user. The purpose of the AVS is to improve the reliability and maintainability of JOVIAL J73 software. The internal operations of the AVS are described in another report, the System/Subsystem Specification.

Table of Contents:

CONTENTS

2. Title: JOVIAL J73 Automated Verification System System/Subsystem Specification

Report No: CR-2-947

Authors: C. Gannon, N.B. Brooks

Date: March 1980

Abstract:

This report defines the software functions, database, and system interface for the J73 Automated Verification System. A summary of the system's requirements and capabilities is provided as background. The software functions are described according to each independent processor segment. The database is described in terms of data structures, management of the database, and usage by each processor segment.

Table of Contents:

CONTENTS

3. Title: JOVIAL J73 Automated Verification System Final Report: Study Phase

Report No: CR-3-947

Author: C. Gannon

Date: March 1980

Abstract:

This report is primarily a review of the state-of-the-art of software testing and verification with emphasis on techniques applicable to JOVIAL J73 programs. Since the project concerns the development of computer-based tool, the need for such a tool, the capabilities for the tool, and the high-level design of the tool are also described. Future capabilities for the tool are identified.

This report is also available from the National Technical Information Service (NTIS), 5285 Port Royal Road, Springfield, Virginia, 22151. Reference RADC-TR-80-261, accession number A091-190.

Table of Contents:

CONTENTS

4. Title: JOVIAL J73 Automated Verification System User's Manual

Report No.: CR-4-947/2

Authors: C. Gannon, R.F. Else

Date: March 1983

Abstract:

This report describes the capabilities and user operation of an Automated Verification System for the JOVIAL J73 computer programming language. This tool, known as J73AVS, provides interactive and batch users with static and dynamic program evaluation, test coverage measurement, retesting assistance, and automated reports that are useful for debugging, documentation, and maintenance of the software.

J73AVS is a command-driven tool that analyzes one or more JOVIAL J73 modules during a session. J73AVS maintains a database that describes the J73 source being analyzed. The saved database allows incremental software analysis and testing. This report describes tool operation during code development, debugging, testing, and documentation. Sample output is provided, as well as job control setups, resource estimations, and error messages.

J73AVS operates on the Amdahl 470 (an IBM 370 equivalent) at the ASD Computer Center and the VAX 11/780 at ASD/AD, both at Wright Patterson AFB. J73AVS recognizes both MIL-STD-1589A and -B versions of JOVIAL J73. It produces instrumented code that can be compiled with any validated 1589A or -B compiler.

Table of Contents:

CONTENTS

5. Title: JOVIAL J73 Automated Verification System Test Plan

Report No.: CR-5-947

Authors: C. Gannon, R.F. Else, R.K. Boxer

Date: October 1981

Abstract:

This report provides a set of test specifications, procedures for testing, and evaluation criteria for the formal testing of the JOVIAL J73 Automated Verification System (J73AVS). J73AVS will be installed and formally tested at ASD/AD, Wright-Patterson AFB, on the ITEL AS/5 and at Rome Air Development Center (RADC), Griffiss AFB, on the DEC20. Formal testing of J73AVS requires a validated JOVIAL J73 (MIL-STD-1589B) compiler and a FORTRAN compiler (for input/output processing only).

Table of Contents:

CONTENTS

6. Title: JOVIAL J73 Automated Verification System Program Maintenance Manual (VAX and IBM-Equivalent Versions)

Report No.: CR-6-947/1

Authors: R.F. Else, C. Gannon, R.K. Boxer

Date: April 1983

Purpose of the Program Maintenance Manual:

The Program Maintenance Manual for an Automated Verification System for JOVIAL J73, called J73AVS, (PR No. B-9-3278) is written to provide the maintenance programmer with the information necessary to effectively maintain the system.

The installations of J73AVS are on the Amdahl 470 (IBM 370 equivalent, OS/MVS) at ASD, Wright-Patterson AFB and VAX 11/780 computers at ASD/ENAF, Wright-Patterson AFB and RADC/CO, Griffiss AFB. It is intended that J73AVS be compatible with all JOVIAL J73 and FORTRAN compilers; therefore, the system is designed to be highly machine-transferrable and not dependent upon particular operating systems or other support software.

The primary language in which J73AVS is written for the IBM 370 equivalent is JOVIAL J73 (MIL-Std-1589B). The IBM version contains a few modules written in BAL and a few written in structured FORTRAN. The structured FORTRAN processor source code is included on the IBM J73AVS magnetic tape (described in Sec. 4.4.1). For the VAX version of J73AVS, the language is structured FORTRAN (a subset of IFTRAN). The structured FORTRAN preprocessor (written in FORTRAN) will be included on that delivery tape.

Table of Contents:

CONTENTS

7. Title: JOVIAL J73 Automated Verification System Program Specification

Report No.: CR-7-947/1

Authors: R.F. Else, C. Gannon

Date: September 1983

Purpose of the Program Specification:

The objective of the Program Specification for an Automated Verification System for JOVIAL J73, called J73AVS, (PR No. B-9-3278) is to describe the program design in sufficient detail to permit program production by the programmer/coder.

Table of Contents:

CONTENTS

8. Title: J73AVS Training Course

Report No.: CR-8-947

Author: C. Gannon

Date: March 1983

Agenda:

First Day		
	9:00 - 10:30am	Overview of J73AVS Capabilities
	break	
	10:45 - 12:00 noon	Selected Output Capabilities
	lunch	
	2:00 - 3:00pm	Overview of J73AVS Operation
	break	
	3:15 - 4:30pm	J73AVS as a Test Tool
Second Day		
	9:00 - 10:30am	Discuss Class Exercises
	break	
	10:45 - 12:00 noon	Review J73AVS Analysis Reports
	lunch	
	2:00 - 3:00pm	Work on Sample Exercises
	break	
	3:15 - 4:30pm	Sample Exercises
Third Day		
	9:00 - noon	Continued Workshop

6 IMPLICATIONS FOR FURTHER RESEARCH

There are several fruitful research areas in which J73AVS can play an important role. Some of these areas call for modifying and extending J73AVS current capabilities and other areas make use of the tool as part of an overall system. The major areas are:

- Enhancements for testing embedded software.
- Enhancements for improving software retesting.
- Aids for project management.
- Aids for software measurement.

Each of these areas is discussed in more detail in the following subsections.

Even though JOVIAL J73 is a new dialect, its "phasing out" by the advent of Ada was known at the beginning of J73AVS design activities. Therefore, further research should support the continued JOVIAL J73 software developers as well as be applicable to Ada usage.

6.1 TESTING EMBEDDED SOFTWARE

Most of the J73AVS training course participants from industry expressed one major difficulty in using J73AVS to test embedded software: the use of a mainframe for program execution.

J73AVS is not intended to replace Performance Monitoring Units (PMUs) or Program Control and Monitoring Systems (PCMSs). These test stations provide powerful, but low-level support to embedded computers, such as Mil-Std-1750A. As currently configured, typical PCMS functions are debugging activities like stepping through instructions, observing all input/output transfers, maintaining "hindsight" of the last 500 instructions, and recording timing performance without the overhead of added instructions to log timing.

The type of PMU or PCMS developed by 1750A computer manufacturers might consist of:

- A microprocessor-based system with a CRT terminal for user interface.
- A Mil-Std-1553B interface to pass information to and from the 1750A machine.
- A RS-232C interface for loading the test program into the 1750A.
- Peripherals such as magnetic tape, floppy disk, hard disk, and a printer for recording execution behavior.
- Sometimes connection to a mainframe computer, like a VAX 11/780, for direct downlinking from the JOVIAL J73 compiler and VAX JOVIAL linker.

These types of development monitors can be more than adequate for extensive software testing, as well as the basic computer hardware and bus checkout, for which they are intended. The addition of J73AVS in the PMU/PCMS environment would provide considerable code development, testing, and documentation power for embedded 1750A software developers too.

In such an environment, J73AVS would reside on the VAX. Static code analysis and automated code documentation would take place on the VAX, just as in non-embedded software systems. Dynamic testing (execution coverage, tracing, timing, and assertion violation checking) would require these modifications to J73AVS:

1. Rewrite the J73AVS execution-time data collection routine in 1750A assembly code.
2. Replace the !TRACE directive usage in J73AVS with calls to appropriate assembly output routines (the !TRACE directive is used by J73AVS to report assertion violations and tracing results).
3. Replace the calls to the mainframe's system clock routine with appropriate 1750A clock calls, if available, for the timing analysis.

J73AVS provides a great deal of high-level capabilities under simple operator control. It is recommended that embedded software developers take advantage of these code checkout facilities before submitting 1750A software to final PMU/PCMS or other low-level analysis.

6.2 ENHANCEMENTS FOR IMPROVING SOFTWARE RETESTING

J73AVS currently offers these capabilities for retesting due to (1) modifying the originally tested code and (2) wanting to improve the thoroughness of test data sets:

- Checking all module interfaces (i.e., COMPOOL variables and module-level parameters) for changes and inconsistencies. Ramifications of interface changes are checked throughout the entire program's database (which is built by J73AVS and saved on a compressed, sequential file).
- Data flow analysis (set-use checking of variables) is performed on a designated (changed) module, looking at all interface variables in that module's calling tree.
- In order to improve thoroughness of test data sets, J73AVS provides a list of control branches not yet executed.
- Execution branch and statement coverage history is saved on the J73AVS-built database.
- The set of branches leading up to a designated (unexercised) branch is specified by J73AVS.

J73AVS provides a good foundation for additional retesting analysis. The information preserved in the database, the current assertion facility, interface checking, and data flow analysis can all be advanced to provide greatly needed retesting assistance.

Computer Sciences Corporation[1,2] (CSC) recently reported some techniques for improving software retesting in a very narrow, structural sense. The basic approach that CSC took during this study contract is a good one: identify a particular test case with changed code. If the changes do not involve the addition of branches or modules or the implementation of new functions, the test cases used during the development (called the testbed by CSC) may be used to re-test the system. Most of the time, however, new test cases will need to be constructed.

The reasons for making software changes are generally:

- To correct development errors
- To correct specification errors
- To satisfy changing requirements
- To improve performance

Therefore, changes to programs usually entail adding or deleting branches in the program, and they often entail slightly or radically modifying the function of the program.

To identify what has changed in a module, the re-testing tool must have an extensive database containing sequencing information, location information, and attribute information describing each module in the software system.

Interface descriptions of the original and modified modules must also be maintained in the database. These descriptions would include the entry point names of the module, a description of the parameters of the module, a description of any global variables set or used by the module, the input/output interfaces to the module, the other modules called by the module, and the sequence of these calls. The parameters to the module need to be described

---

[1] Software Retest Techniques Functional Description, prepared for RADC, Computer Sciences Corporation, Feb. 1982.

[2] Software Retest Techniques Final Technical Report, prepared for RADC, Computer Sciences Corporation, Feb. 1982.

by their attributes: number, type, dimensions, set/use, ranges and enumerated values. Global variables should be described by their type, dimension, use in the module, range, and initial value. Input and output operations (implemented as calls to non-JOVIAL routines in J73 programs) need to be described by their sequence, number of records read or written, file type, and record format.

Several things must be done in order to re-test software effectively:

- The changes to the software must be identified
- The parts of the software which need to be re-tested must be identified
- Test cases must be chosen to test these parts

The major problem with the CSC approach to software re-testing is that it focuses almost entirely on a very small class of re-testing needs; that is, the class of small changes to a program in which the actual structure is not modified. In the cases of adding or deleting branches to a program, the methodology offers very little assistance. In fact, the retesting tool specified in the Functional Description provides no additional help over what J73AVS currently offers in deriving new testcases to exercise modified code. The second major problem is that the methodology does not address re-testing new functions or performance changes. To support these testing activities, more extensive interface checking, a formal use of J73AVS assertions, and implementing the execution performance instrumentation component described in the J73AVS Functional Description.[1] Execution performance value ranges can be used to support unit-level testing of modified modules.

6.3 AIDS FOR SOFTWARE MEASUREMENT AND PROJECT MANAGEMENT

Quantifying certain aspects of software development can illuminate trends and problem areas. In particular, obtaining software measurements can:

---

[1] JOVIAL J73 Automated Verification System Functional Description, C. Gannon and N.B. Brooks, CR-1-947, General Research Corporation, March 1980.

- Guide the testing process
- Record project status
- Provide parameters for cost estimation
- Quantify software quality
- Help formulate system testing requirements

J73AVS already automatically collects a variety of measures through its execution coverage analyzer, static analyzer, and test history reporting. Other current features such as assertion violation reporting, system overview description (number and types of modules, dates analyzed, lines of code, etc.) and invocation structure reporting can be modified to provide project status monitoring.

We recommend augmenting the current J73AVS test history recording with accumulated statistics on the number and severity of static errors for each modified (presumably "corrected") version of a module. As part of measuring static errors, we also suggest adding more extensive data flow analysis capabilities to J73AVS. With JOVIAL J73's rich data types and scoping rules, many programming errors could be uncovered by more exhaustive analysis of referenced COMPOOL variables, table or block subscripted values, etc. A description of the current data flow capabilities of J73AVS is provided in the User's Manual.[1]

While the programmer may be interested in the types of errors uncovered in each module, the project manager may be more interested in how many modules have undergone static analysis, have been executed with assertions turned on, or have been analyzed for timing performance and branch coverage. Statistics of this nature would be easy to include in J73AVS.

Software metrics engineers usually lament about the quality of software error data collected during (or, usually, after) a project. Automatic

---

[1] JOVIAL J73 Automated Verification System User's Manual, CR-4-947/2, March 1983.

collection and output onto a sequential file by J73AVS would also be an easy and useful addition to J73AVS. The user could specify certain data collection formats (such as "by module," "by date of analysis," "by error type," etc.) for the output file. Such a data file could be valuable for software problem analysis and feedback programs, error data archiving, partial input to cost models, and project management programs.

APPENDIX A

This appendix consists of three technical papers that describe J73AVS and how it was implemented. Two of the three conference papers were presented to audiences who have a keen interest in high-level language support for avionics applications.

The technical papers included in this appendix are:

1. A Debugging, Testing, and Documentation Tool for JOVIAL J73
2. An Automated Verification System for JOVIAL J73
3. J73AVS: A JOVIAL J73 Automated Verification System

AT-22
RM-2236

A DEBUGGING, TESTING, AND DOCUMENTATION TOOL FOR JOVIAL J73

C. Gannon

COMPSAC

October 27-31, 1980
Chicago, Illinois

GENERAL RESEARCH CORPORATION
P.O. BOX 6770, SANTA BARBARA, CALIFORNIA 93111

A Debugging, Testing, and Documentation Tool for JOVIAL J73

C. Gannon

General Research Corporation

Introduction

This paper describes an automated tool which is being developed to assist in debugging, testing, and documenting software written in JOVIAL J73. Many of the techniques incorporated in this tool have already been found successful in testing and maintaining FORTRAN and JOVIAL J3 programs and may be considered viable techniques for future Ada tools. Features of this tool, called J73AVS (for JOVIAL J73 Automated Verification System), that distinguish it from other AVS's are primarily due to the specific characteristics of the JOVIAL J73 language and the applications programs written in J73. Additional distinguishing features are to provide a realistic approach to testing program paths, maintain multi-run testing history information, and operate in both batch and interactive modes.

As background material, characteristics of the J73 language and applications programs are described briefly. The paper then provides a functional description of the tool.

The Need For J73AVS

The need for this automated verification system is based upon (1) the emergence of a new JOVIAL language dialect which will supersede the previously approved JOVIAL dialects, (2) the characteristics of the language that make it complex and error-prone, (3) the type of applications expected to be written in the language, (4) and the standardization of certain testing measures.

In an effort to prescribe a standard policy for using computer programming languages and for testing computer programming language compilers, the Air Force issued AF Regulation 300-10 in 1976. Two JOVIAL languages, J3 and J73/I, were specified as Air Force standard high-order programming languages. Both JOVIAL languages are primarily designed for command and control system programming. They are especially well suited to large systems requiring efficient processing of a large volume of data with complex structure.

Another JOVIAL language, J3B, evolved from J3 for the purpose of developing computer programs for the Boeing B-1. Derivatives of J3B have been widely used for avionics computer programming. However, JOVIAL J3B is not a language approved by AF Regulation 300-10. Therefore, a blend of J73/I and J3B, plus additional features not in either language, has been created to satisfy the programming needs of both the avionics and systems communities. This language, JOVIAL J73, was specified in March 1979 as MIL-STD-1589A and was refined in June 1980 to become MIL-STD-1589B. In the spring of 1980, AF Regulation 300-10 was revised to cancel both J3 and J73/I languages, leaving J73 as the only approved JOVIAL language.

It was the desire to improve software reliability that prompted the Air Force's request for an Automated Verification System (AVS) to be developed and made available as soon as possible following release of validated JOVIAL J73 compilers. Encouragement for an AVS and other support tools also came from the JOVIAL Users Group, a body of interested management and technical people from industry, Government, and the Air Force.

Characteristics of J73 Programs

As defined in MIL-STD-1589B, JOVIAL J73 permits the independent processing of functional modules which communicate through compools and argument transmission. J73 permits both recursive and reentrant procedures for effective multi-processing. The language provides a rich variety of data types and supporting data manipulation functions, making assembly code programming unnecessary for most applications. However, except for a trace directive which supplies limited output facility, there is no input/output capability in the language. Linkages to assembly or alternate-language routines are required for input and output.

Storage allocation for data objects can be both automatic (in which storage is released when control exits from the program unit) or static (in which storage space is saved throughout the entire execution of the program). Automatic allocation uses storage efficiently but makes certain data-usage errors possible.

The DEFINE construct associates a name with a text string such that whenever that name is referenced, the text string replaces it. DEFINE statements can be nested and can be redefined based upon scope. Thus, while the ca-

pability is extremely useful, it adds another dimension of complexity to JOVIAL programs.

Unfortunately for advocates of structured programming, the control statements in JOVIAL J73 are not confined to the "structured programming" constructs of sequential flow, IF-THEN-ELSE, and WHILE-loops. The language does at least have these constructs, so that programmers can write structured code if they desire. However, unstructured statements as GOTO, FALLTHRU, EXIT, and ABORT are also permitted. The GOTO statement allows transfer from the outside of an IF or CASE construct into the body of the IF or CASE. GOTO statements can also be directed to labels that are external to a program unit, if the label is passed as a parameter. The FALLTHRU statement allows control to pass from one CASE alternative to another without making the test normally required at each CASE option. The EXIT statement allows escape out of an immediately enclosing loop. The ABORT statement provides transfer of control to the label specified in the most recently executed, currently active procedure having an ABORT phrase. Thus, control transfer is not defined until execution time.

The unstructured control statements provide flexibility and execution-time efficiency; but at the same time they increase the chance of committing errors and make the program more difficult to understand. Since 60% of the total cost of software is generally attributed to maintenance, source code scrutability is important.

J73AVS will provide extensive static and data-flow analysis to detect and report possible errors regarding control transfers, data contention due to static allocation, uninitialized variables, structurally unreachable code, potential infinite loops, etc. Program analysis reports can be generated on command by the user to describe such detailed information as DEFINE usage, label references, symbol properties, and global data. Static and data-flow analysis are testing and program analysis techniques in which the program is not executed with real data. Rather, the code is analyzed for inconsistencies (static analysis) or for variable set-use behavior (data flow analysis) using symbolic execution through a directed-graph of the program.

Characteristics of Application Programs

The programs that will be implemented in JOVIAL J73 will be of similar nature to those written in the separate JOVIAL dialects: J3, J3B, and J73. Applications will be for navigation, information management, flight controls, communications, etc. The software characteristics of the applications are varied. For example, flight control software has the following characteristics:

- synchronization
- distributed processing
- structurally simple control statements
- simple data types
- real-time processing

On the other hand, applications such as command and control systems have very different characteristics such as:

- batch and interactive modes
- complex data structures
- complex control structures
- large, monolithic modules
- non-real-time processing

Avionics applications are often destined for small on-board computers. For those computers not having a JOVIAL J73 or J73-subset compiler, the programs are developed on a host machine and cross-compiled to the target machine. There are no software checkout tools available on these small computers, so an AVS operating on the host computer must supply as much assistance as possible to detect errors in program performance and assure some level of testing thoroughness before the program is cross-compiled.

Command and control systems, on the other hand, tend to be very large (several hundred thousand lines of code). They also tend to evolve as needs change. Therefore, not only is testing a major problem — code modification and retesting only what is necessary are also difficult tasks. In the face of these problems, one of the most valuable assets of any software support tool is the ability to automatically produce concise but helpful program documentation.

Functional Description of J73AVS

This section presents a brief description of the capabilities of J73AVS and describes in what phases of the software life cycle the capabilities should be used. A thorough description is provided in the Functional Description.[1]

Our approach to the design of an AVS for JOVIAL J73 is to provide automated assistance for

- program development
- debugging
- testing
- retesting

The approach excludes

- verification of requirements
- verification of specifications
- automated design aids
- formal program verification (proof of correctness)

The techniques for automating these processes are not developed well enough to be reliable for general-purpose, large software systems.

The specifications for the J73 dialect and compilers include rigorous data-type checking and scope rules. The language allows, however, constructs and control structures which demand caution in their usage (such as recursive and reentrant procedures, jumps into certain control structures, abnormal exits, etc.). Further, the language does not contain a mechanism for specifying expected behavior or reporting user-specified abnormalities (since there is no input/output facility).

J73AVS will not duplicate the static consistency checking of the compiler, but, rather, provide the following set of facilities to support program development, debugging, testing, maintenance, and documentation of JOVIAL J73 programs:

1. Logical assertions and timing probes
2. Static and data flow analysis
3. Program structure and characteristic reporting
4. Statement performance dynamic analysis
5. Branch, path, and program unit execution coverage analysis
6. Branch and program unit execution trace analysis
7. Execution timing analysis
8. Structural retesting assistance
9. Test history reporting

J73AVS will support interactive and batch facilities since the various stages of program development through testing and maintenance lend themselves to both modes of operation. The command language will be similar for interactive and batch usage, except that the interactive user will be prompted for information where necessary.

Summary of Capabilities

A summary of capabilities is provided as a flow diagram in Fig. 1. This diagram describes the primary functions supported by J73AVS as well as the sequence in which they are performed. Figure 2 shows the interaction between J73AVS and the user. The user can direct the sequence of analysis activities, using information provided at each stage of processing.

Although J73AVS will exist as a single program (with overlays to keep it compact), it is best considered as a collection of tools or facilities with which the user interacts. Some of the facilities, such as automated documentation, static error reporting, and instrumentation, are completely automated and require only that the user initiate the tasks by command. Other processes, such as execution-time data collection or retesting assistance, require more information from the user like test data input and test target selection.

J73AVS provides detailed information both statically and dynamically about the program being analyzed. It is the role of the user to direct the processing performed by J73AVS, to analyze the output produced by J73AVS, and to determine subsequent action.

The role of J73AVS in the software development cycle is to provide automated assistance wherever possible during the program development and maintenance, debugging, testing, and retesting phases of the cycle. Because J73AVS systematically provides execution coverage, timing analysis, execution performance reporting, and test history, it is expected to be valuable to independent verification and validation contractors as well as software developers of J73 programs.

The user of J73AVS plays an active part in the cycle as shown in Fig. 3. This figure partitions the phases of the development cycle and shows the flow between the automated processing of J73AVS and user-supplied input or direction.

Using Fig. 3 as a basis, a typical sequence of J73AVS-supported processing can be described as follows:

1. JOVIAL J73 source text is generated and provided to J73AVS as one or more compilable modules.
2. J73AVS produces program analysis reports showing control structure, symbol usage, calling hierarchy, etc., as well as a static analysis report showing errors and dangerous programming practices.
3. Using the reports as a guide, the source modules can be modified or new modules added to the program.
4. J73AVS identifies the interaction of the new or modified modules with the rest of the program; this information, in turn, is used as the basis for modifying other modules.
5. For dynamic debugging, the program is instrumented (i.e., calls to data collection routines are automatically inserted) by J73AVS and executed with an initial test case supplied by the user.
6. J73AVS reports assertion violations, if any, and generates an evaluation of statement and variable performance.

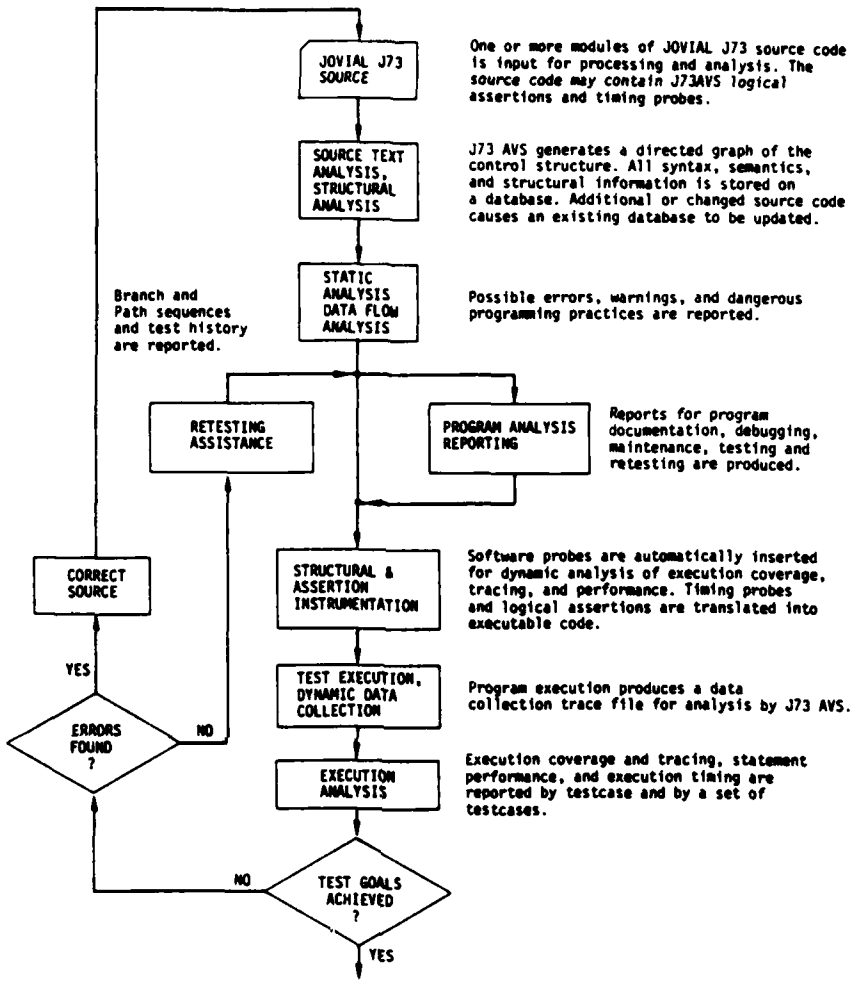


Figure 1. Overview of J73AVS

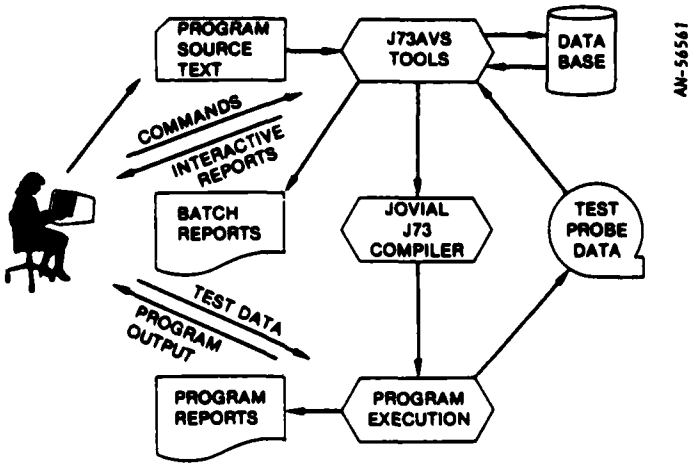


Figure 2. J73AVS Interaction with User

7. Using this evaluation, the user may choose to generate additional test data to pinpoint errors or instrument other modules for additional dynamic debugging.

8. Like debugging, the same procedures of test data generation, instrumentation, and execution are performed for testing but for a different goal: rather than detecting and locating errors, testing aims to demonstrate the absence of errors and that the software conforms to its specification. Therefore, J73AVS produces execution analysis reports in terms of the thoroughness of execution coverage and violations of asserted behavior.

9. The user evaluates execution coverage and other program performance output, along with the program's own execution results and the program specification, to determine if testing is complete.

10. J73AVS provides branch sequence information to retest targets chosen by the user. A test history of execution coverage and assertion violations assists the user in choosing targets for retesting.

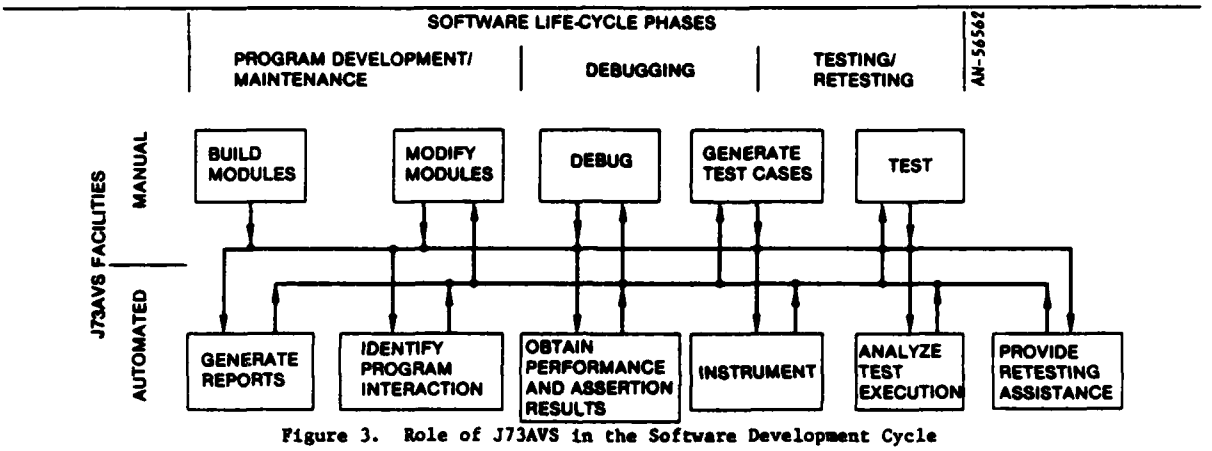


Figure 3. Role of J73AVS in the Software Development Cycle

Program Development and Maintenance

Executable assertions permit a programmer to specify expected behavior. J73AVS supports the technique of embedding programmer-specified assertions into the code through the use of the ASSERT keyword followed by any legal logical (Boolean) expression. Logical assertions can be used for execution-time exception reporting, stress testing, test data generation filtering, and (left as comments in the source code) stating in-line specifications.

To assist with reliable system development, maintenance, and documentation, J73AVS will provide substantial program analysis reporting on structural hierarchy, symbol usage, invocations, certain J73 constructs, and system characteristics. The user has control over obtaining high- or low-level information through the command language. The types of program analysis reporting include the following:

- indented source listing with control structure identification
- symbol cross reference with set-use information
- compool symbol description
- properties of all or specified symbols
- declaration and reference of labels (statement names)
- declaration and reference of user-defined data types
- declaration and reference of constants
- usage of external reference (REF) and definition (DEF)
- declaration and reference of DEFINE text strings
- description of program units on the database

Debugging

Normal compilation using JOVIAL J73 compilers will detect many syntax and semantic errors. Additional errors such as uninitialized variables, possible infinite loops, unreachable code, certain improper constructs, and dangerous coding practices (like transferring into CASE or IF statements) will be reported by J73AVS. The user can command different levels of static reporting.

Dynamic debugging will be supported by statement execution performance and assertion exception reporting. Statement execution performance provides execution counts of statements, values and ranges of variables in assignments and loops, and the execution behavior of IF statements. This debugging information appears adjacent to the source statements themselves, which assists the task of code correction. The execution of timing probes (inserted by command) can be reported in the debugging performance report at the user's request.

When the program's execution behavior deviates from the acceptable logical behavior specified by the embedded assertions, it will be reported during execution. This is called an assertion violation. The user-supplied assertions remain relatively transparent to the program until they are violated; at that time the violation is reported along with the module name and source statement number where the violation occurred.

Testing

When used in conjunction with static checking and statement-level performance analysis, structure-based testing can uncover errors due to untested branches (where a branch is a control flow outcome due to a decision statement) or improper sequences of branches. J73AVS will provide execution tracing of program units and branches and execution coverage analysis of program units, branches, and sequences of branches (paths). Further, J73AVS will assemble the timing information from program unit tracing and user-supplied timing probes into an execution timing report.

Although an AVS can provide an objective measure of testing thoroughness in terms of statement or branch execution coverage, frequently errors in software are overlooked during testing because only certain sequences of branches are ever executed. Obviously, it is generally impossible to define all paths in programs because of loops. Furthermore, the most likely subset of paths to test can best be identified by a person familiar with the function of the program. The most efficient role of an AVS in this regard is to identify the set of control paths between two statements in a program unit (an invokable unit of code) to which the human tester attaches importance. Of the set of paths identified by the tool, the user can choose those that are to be analyzed for coverage during execution. If the set of paths is too large to enumerate, a descriptive message will be issued and the user allowed to choose another pair of statements for path identification.

Retesting Assistance

Retesting software is performed when analysis shows that prior testing is inadequate (insufficient branch coverage, not all functions demonstrated, etc.) or when program changes have taken place. The proper approach to take in retesting is highly dependent upon the characteristics of the program being tested as well as the measures being used to evaluate testing completeness.

In order to determine the sequences of branches which must be executed to reach an untested branch or statement, the user can request that the "reaching set" be computed between two specified statements (or from the program unit's entry). The user can also request a list, in terms of branches, of all control paths between two specified statements. If certain loop structures make this list impossible, subsets of the paths will be identified.

With the control flows identified, the user can develop new test data by backtracking through the program to the input space, using statement execution performance reports, module interaction and invocation reports, and execution coverage information for each testcase. Unfortunately, automatic test data generators which use symbolic execution are not yet developed to the point of being general-purpose, easy to use, or reliable.

The cumulative test coverage history maintained by J73AVS will be useful in attaining testing goals and determining targets for retesting. Typically, the results of executing test cases are difficult to manage. Program unit and branch coverage information will be saved in a concise way on the database for each test case. The results of subsequent execution runs can be added, providing a cumulative report of all tests. Also saved in the history database table will be any assertion violations that occur. This will provide a mechanism for identifying which input test case caused a violation.

Acknowledgement

The design and development of J73AVS is being sponsored by the Rome Air Development Center, Griffiss Air Force Base, New York.

Reference

[1] C. Gannon and N. B. Brooks, JOVIAL J73 Automated Verification System Functional Description, General Research Corporation, CR-1-947, March 1980.

IM-2420
AT-31

AN AUTOMATED VERIFICATION SYSTEM FOR JOVIAL J73

Robert F. Else

NAECON
May 18-20, 1982
Dayton, Ohio

GENERAL RESEARCH CORPORATION
P.O. BOX 6770, SANTA BARBARA, CALIFORNIA 93111-0770

AN AUTOMATED VERIFICATION SYSTEM FOR JOVIAL J73

ROBERT F. ELSE

GENERAL RESEARCH CORPORATION
P.O. BOX 6770
SANTA BARBARA, CALIFORNIA 93111

ABSTRACT

The introduction of the J73 dialect of the JOVIAL programming language created a need for new software tools to help develop, test, and maintain J73 software systems. This need is especially great in software for aviation electronics (avionics), where rigid functional and reliability requirements make the automation of software analysis a necessity.

To fill this need, General Research Corporation (GRC) has developed the JOVIAL/J73 Automated Verification System (J73AVS).[1,2] It is a tool that processes J73 source code like a compiler, but maintains a permanent database of information making repeated processing of source code unnecessary. It operates in either batch or interactive mode. This paper describes the features and use of J73AVS.

BACKGROUND

In 1976, in an effort to standardize computer programming languages and compilers, the Air Force issued AF Regulation 300-10 which specified two JOVIAL languages, J3 and J73/I, as standards for avionics and communication systems. Another JOVIAL dialect, J3B, had evolved from J3 primarily for use in the Boeing B-1 computer system, although J3B was not approved by the AF regulation. A new dialect, J73, specified in 1979 as MIL-STD-1589A was a blend of J73/I and J3B, plus some new features; this specification was refined in mid-1980 to become MIL-STD-1589B. In the spring of 1980, AF Regulation 300-10 was revised to eliminate J3 and J73/I, leaving J73 as the only approved JOVIAL dialect.

The request for a J73 Automated Verification System (AVS) was part of the Air Force's desire to improve software reliability, and it was planned for release as soon as possible after the release of new validated J73 compilers. Encouragement for an AVS and other support tools came also from the JOVIAL User's Group, a body of interested management and technical people from industry, Government, and the Air Force.

JOVIAL J73 is an extremely rich and complex language, and will be used for a fairly wide variety of applications, including navigation, information management, flight controls, communications, and command and control systems. The language's complexity and its varying applications reinforce the need for an AVS to help analysts and programmers handle the difficult task of developing and maintaining J73 software systems.

J73AVS FUNCTIONS

J73AVS was specifically designed to be used in the post-design phases of the software life cycle: program development, debugging, testing (and retesting), and maintenance. Since automated tools already exist for the specification and verification of requirements and design, we felt that concentrating on the actual code production and maintenance phases would result in a more powerful, concentrated set of capabilities. J73AVS can also produce a wide variety of program documentation; although this activity is not generally included in the software life cycle, it is often a weak link in the chain holding together an otherwise strong system.

The tool's capabilities were designed to complement those of the J73 compiler, so that the two together would cover as many trouble spots as possible. The compiler provides rigorous data-type checking and scoping rules, but there are cases where the programmer can circumvent these checks, whether intentionally or not. In addition, the language itself allows potentially disastrous control constructs such as jumps into CASE and IF statements, abnormal exits from procedures, etc., and it is beyond the scope of the compiler to issue warning messages for programs which use them. We also wanted to avoid duplicating capabilities of other tools which have a bearing on J73; SDDL (System Design and Documentation Language), for example, works at the pseudo-code level, and the Jovial Interactive Debugger[3] will operate at execution time.

J73AVS provides the following set of functions:

1. Static and data-flow analysis
2. Program structure and interface reporting
3. Execution coverage analysis (at statement, branch, and procedure levels)
4. Execution tracing analysis (at branch, procedure, and variable levels)
5. Execution timing analysis
6. Test history reporting

7. Structural retesting assistance
8. Logical assertions for reporting deviation from expected program behavior
9. A wide range of documentation reports

A summary of all the J73AVS capabilities is provided in Figure 1. This diagram describes the primary functions supported, as well as the sequence in which they are performed. Figure 2 shows the interaction between J73AVS and the user; a command language is used to direct the sequence of activities. The command language is identical in either batch or interactive mode. It is keyword-oriented, has parameters that specify options, allows abbreviations, and is flexible in the order of parameters. In interactive mode, prompts and a help function assist users.

Although J73AVS exists as a single program, it is really a collection of tools or facilities from which the user can choose. Some, such as automated documentation, static error reporting, and instrumentation, are completely automated and require only a single command from the user. Others, such as execution-time data collection and retesting assistance, require more input and interaction with the user.

SAMPLE OUTPUT

Some samples of actual output give a quick picture of how J73AVS can be used. Figure 3(a) is a J73AVS report summarizing the contents of the user's database, itemized by module name. It shows each module's name, type (compool, program, or procedure), number of statements, program units (individual invokable procedures), compool references, DEFINE name declarations, and creation and update dates. J73AVS databases are retained from run to run, allowing them to grow incrementally as a system is being developed. One database may be used to retain data about an entire system; alternatively, different programmers may maintain different databases which can be combined at a convenient point in development. Figure 3(b) is another form of J73AVS database summary, itemized by program unit name. Each procedure's parent (enclosing procedure, if any) is given, along with nesting information.

The next report, Figure 4, shows execution timing data produced by J73AVS. This report is obtained by instrumenting one or more procedures with timing probes (inserted automatically on command by J73AVS), executing the instrumented code, then using J73AVS to analyze the "audit" file produced by the execution. The bottom half of this example shows that two segments of code were chosen by the J73AVS user for monitoring execution time. These "clock intervals" need not be wholly contained in the same procedure or module. The execution times shown are in milliseconds and are as accurate as the system-level CPU clock.

The user may also instrument source code to record which control branches are covered (executed) during a test execution of the code. J73AVS reports exactly which branches have or have not been hit, and maintains a history of the coverage results in its database. Statement and branch execution coverage can be reported by J73AVS as part of a source listing, or more abbreviated reports can be selected at user option. For example, Figure 5 shows the "NOT HIT" report for two testcases (sections of source code chosen as test boundaries by the user) and a cumulative branch coverage history. The branch numbers in this report can be keyed back to the source code through several other J73AVS reports.

A system-level branch coverage report is available and is shown in Figure 6. This report shows the results of all testing so far and includes the number or procedure invocations, branches, and statements hit, and percentage of total coverage.

THE HOST/TARGET DICHOTOMY

A major concern in avionics software is the difficulty of performing thorough dynamic testing on embedded software. Target computers are usually much smaller than the original host, with limited memory and processing resources and little or no I/O capability. Dynamic testing, and measuring its thoroughness, are difficult problems in these small real-time environments.

The instrumentation capabilities of J73AVS offer a solution to this problem. The software to be tested can be instrumented automatically under the user's control, and the resulting code compiled, linked, and loaded (but not executed) on the host computer, then transferred to the target on magnetic tape, punched cards, or whatever I/O medium is available. Executing the instrumented code on the target produces an audit file for subsequent analysis by J73AVS on the host. Alternately, if the target has enough main memory, or lacks an output device for the audit file, the J73AVS data collection and analysis routines can easily be modified to use a "HITS" array (one cell per branch, each holding a "hits" counter); post-execution analysis of this data would then take place on the target computer.

Since the user has complete control over which segments of code to instrument, overhead and code expansion can be minimized; testing can be done in segments, or the testers can "zero in" on a particularly critical procedure or module. This is a significant gain over previous approaches which involve either the tedium of inserting the software probes by hand, or a lack of a fine-grained automatic control over the segments of code to be instrumented, thus causing unnecessary overhead.

This method of testing on a small target computer has already been proven with a tool named EAVS (Extensible AVS), also developed at GRC.[4] EAVS was an early precursor of J73AVS, and the software technology advances since that time make J73AVS a very important testing tool for embedded systems.

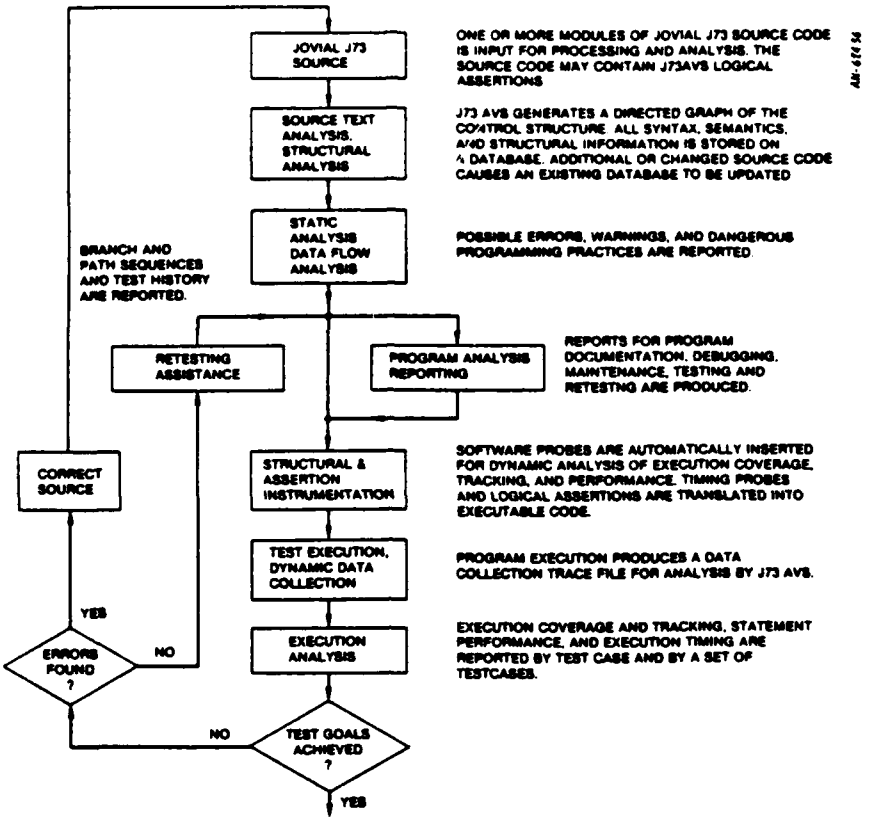


Figure 1. Overview of J73AVS

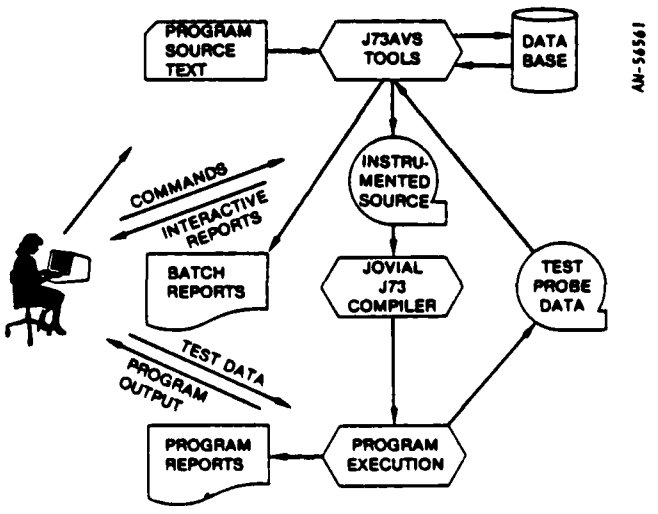


Figure 2. J73AVS Interaction With User

J73AVS DATABASE 11/06/81 PAGE 2

ITEMIZED BY MODULE NAME

NO.	MODULE NAME	TYPE	STMTS	UNITS	CMPLS	DEFNS	DATE OF CREATION	LAST ALTERED
1.	TYPES	CMPL	7	1	0	1	11/06/81	11/06/81
2.	EXTERNS	CMPL	23	1	1	1	11/06/81	11/06/81
3.	LOOKUP	PROG	68	2	1	2	11/06/81	11/06/81
4.	FINDP	PROC	27	1	1	1	11/06/81	11/06/81
5.	ERROR	PROC	25	1	0	0	11/06/81	11/06/81

Figure 3(a). J73AVS Database Summary

J73AVS DATABASE 11/06/81 PAGE 3

ITEMIZED BY PROGRAM UNIT NAME

NO.	UNIT NAME	UNIT TYPE	PARENT NAME	MODULE NAME	NEST	FIRST STMT	LAST STMT
1.	TYPES	CMPL	(COMPOOL)	TYPES	0	1	7
2.	EXTERNS	CMPL	(COMPOOL)	EXTERNS	0	1	23
3.	LOOKUP	PROG	(MAIN PRGRM)	LOOKUP	0	3	67
4.	CONVERT	PROC	LOOKUP	LOOKUP	1	50	66
5.	FINDP	PROC	(NON-NESTED)	FINDP	0	4	26
6.	ERROR	PROC	(NON-NESTED)	ERROR	0	2	24

Figure 3(b). J73AVS Database Summary

EXECUTION TIMING REPORT 07/06/81 PAGE 2

PROGRAM UNIT	MODULE	TIMES CALLED	MIN UNIT	MAX UNIT	AVG UNIT	TOTAL UNIT	TOTAL LEVEL
ERROR	ERROR	9	2	4	3	24	24
TSTINST	TSTINST	1	5	5	5	5	93
TSTABRT	TSTABRT	1	4	4	4	4	4
TESTLOOP	TSTINST	1	78	78	78	78	78
TESTIF	TSTINST	1	6	6	6	6	6

CLOCK INTERVAL TIMES

START MODULE	START STMT	END MODULE	END STMT	COUNT	TOTAL TIME	AVG TIME
ERROR	23	ERROR	30	1	17	17
ERROR	27	TSTABRT	11	1	20	20

Figure 4. J73AVS Timing Report

BRANCHES NOT HIT REPORT PAGE 1

TEST CASE 1 07/06/81 TERMINATING AT STMT 27 OF MODULE ERROR

MODULE NAME	I	TOTAL BRCHS	BRCHS HIT	PER CENT	I	BRANCHES NOT HIT
ERROR	I	7	3	43	I	3 4 6 7

TEST CASE 2 07/06/81 TERMINATING AT STMT 7 OF MODULE TSTINST

MODULE NAME	I	TOTAL BRCHS	BRCHS HIT	PER CENT	I	BRANCHES NOT HIT
ERROR	I	7	6	86	I	4
TSTINST	I	29	1	3	I	2 3 4 5 6 7 8 9 10 11
	I				I	12 13 14 15 16 17 18 19 20 21
	I				I	22 23 24 25 26 27 28 29
TSTABRT	I	13	3	23	I	3 4 6 7 8 9 10 11 12 13
	I				I	

CUMULATIVE

MODULE NAME	I	TOTAL BRCHS	BRCHS HIT	PER CENT	I	BRANCHES NOT HIT
ERROR	I	7	7	100	I	*NONE*
TSTINST	I	29	17	59	I	8 9 13 16 19 20 22 24 25 26
	I				I	28 29
TSTABRT	I	13	3	23	I	3 4 6 7 8 9 10 11 12 13

Figure 5. Branch Execution Coverage Report

TEST COVERAGE SUMMARY

TEST CASE 1 07/06/81 TERMINATING AT STMT 27 OF MODULE ERROR

				TEST CASE						CUMULATIVE				
MODULE NAME	TOTAL BRCHS	TOTAL STMTS	I	NO. OF INVKS	BRCHS HIT	PER CENT	STMTS HIT	PER CENT	I	NO. OF INVKS	BRCHS HIT	PER CENT	STMTS HIT	PER CENT
ERROR	7	23	I	4	3	43%	13	57%	I	4	3	43%	13	57%

.
.
.
.

TEST CASE 10 07/08/81 TERMINATING AT END OF RUN EXECUTION

				TEST CASE						CUMULATIVE				
MODULE NAME	TOTAL BRCHS	TOTAL STMTS	I	NO. OF INVKS	BRCHS HIT	PER CENT	STMTS HIT	PER CENT	I	NO. OF INVKS	BRCHS HIT	PER CENT	STMTS HIT	PER CENT
ERROR	7	23	I	0	0	0%	0	0%	I	9	7	100%	23	100%
TSTINST	29	78	I	0	16	55%	54	69%	I	1	17	59%	61	78%
TSTABRT	13	29	I	0	0	0%	0	0%	I	1	3	23%	12	41%
TOTAL	49	130	I		16	33%	54	42%	I		27	55%	96	74%

Figure 6. Testing History Summary

LOGICAL ASSERTIONS

This is a simple yet powerful technique in which the programmer inserts special statements called "assertions" into the source code that specify the expected behavior of the program at a given point. This facility provides the most payoff when assertions are programmed during design or early coding stages. For example, the assertion

```
". ASSERT ( STACK'POINTER >= 0 ) "
```

states that the value of the data item STACK' POINTER is at least zero.

Assertions provide a very convenient way of reporting execution-time deviations from expected or required behavior and can be used for stress testing, test-data generation, etc. After code is thoroughly checked out, they can simply be left as comments in the code (as shown above). One of the instrumentation options provided by J73AVS expands these statements into executable code; when the expanded code is executed, violations of the asserted conditions are reported as part of the program output. For example, a negative value of STACK'POINTER (above) would produce a message like

```
IN MODULE POP'STACK: ASSERT FALSE AT STMT 149
```

A special FAIL statement is also provided for the programmer to use in specifying "contingency code" for deviations from the asserted conditions. In developing J73AVS, we made liberal use of assertions to help us debug and test the code, and the assertions still exist in the source code of the tool.

OTHER FUNCTIONS OF J73AVS

Besides those discussed above, J73AVS provides the following:

- indented source listings
- single- or multi-module symbol cross references with set-use information. The J73AVS user can select symbol names and/or data types.
- static analysis to pinpoint error-prone coding practices, mismatched formal and actual parameters (when the compiler is unable to do so), unused labels, data contention in recursive or reentrant procedures, possible infinite loops, and unreachable code.
- instrumentation to provide statement or branch execution counts and procedure, branch, or user-specified variable tracing.
- DEFINE name declarations and usage cross-reference.
- module and system interface descriptions, including calling trees, etc.

SUMMARY

J73AVS is a tool that can lower software production costs and improve software reliability. It provides a framework for the use of standard-

ized, organized testing measures, and is useful as a software development and maintenance tool.

It can operate in either batch or interactive mode and uses a flexible command language complete with a "HELP" feature. Keeping J73AVS as machine-independent as possible has been a goal throughout its development. Dynamic testing on target computers can be performed by analyzing the execution-data file, produced by any target that can output sequential files to off-line storage, on the host computer. Current and planned host computers for J73AVS are the IBM 370, DEC 20, and VAX 11/780.

ACKNOWLEDGEMENT

The design and development of J73AVS was sponsored by the Rome Air Development Center, Griffiss Air Force Base, New York, under contract F30602-79-C-0265. The contract monitor was Mr. Frank LaMonica.

REFERENCES

1. C. Gannon and N.B. Brooks, JOVIAL J73 Automated Verification System Functional Description, General Research Corporation CR-1-947, March 1980.

2. C. Gannon and R.F. Else, JOVIAL J73 Automated Verification System User's Manual, General Research Corporation CR-4-947, November 1982.

3. Maj. D. Burton, S. May, and T. Fujawa, "A JOVIAL Interactive Debugger," NAECON 81, pp. 1121-1129.

4. S.O. Campbell and S.H. Saib, "Embedded Software Verification Through Instrumentation," NAECON 81, pp. 395-401.

AT-38

J73AVS: A JOVIAL J73 Automated Verification System

Carolyn Gannon

AFSC Standardization Conference
Dayton, Ohio November 29 -
December 2, 1982

GENERAL RESEARCH CORPORATION
P.O. BOX 6770, SANTA BARBARA, CALIFORNIA 93111-0770

J73AVS: A JOVIAL J73 Automated Verification System

Carolyn Gannon

General Research Corporation
P.O. Box 6770
Santa Barbara, CA 93111
805-964-7724

ABSTRACT

The development of J73AVS reflects the commitment of the Air Force to facilitate JOVIAL J73 standardization. This paper describes software verification as it is automated by the J73AVS tool. The concept of software verification is discussed, as well as the capabilities and operation of J73AVS. J73AVS provides much of its payoff by detecting certain software errors and measuring the thoroughness of testing far more accurately and efficiently than could be achieved manually. While J73AVS operates as a standalone program on several host computers, it augments the JOVIAL J73 support environment when used with other Air Force-sponsored tools such as the code auditor [1], debugger [2], and Program Support Library [3].

INTRODUCTION

What is "software verification"? According to the IEEE committee on standardizing software terminology, it is

> "the iterative evaluation of evolving software to ensure compliance with requirements"

Thus, verification differs from validation or certification in that it is an activity that is performed continuously throughout a software development cycle. It incorporates a variety of automated and manual techniques to determine consistency between the requirements, design, coding, testing, and documenting stages of software projects.

Our focus in designing and building Automated Verification Systems (AVS) for JOVIAL [4], FORTRAN [5], and COBOL [6] has been on static and dynamic code analysis. That is, each AVS reads source code as input for static analysis and uses the program's regular input data during dynamic analysis. Therefore, requirements and design are not verified directly by the AVS. However, because the AVS analyzes the actual code, the tool reports true program characteristics. This approach interferes very little with normal program development, since the AVS does not require additional information other than the source code and initial set of test data.

One very important, but often neglected, area of software verification is that of ensuring that the program documentation reflects the real code. The most time-consuming aspect of conforming to Mil-Std-483 or other documentation standards is generating program symbol, structure, and interface information. Because it maintains a database of intra- and inter-module characteristics, the AVS can generate some of this information automatically. As code is modified due to error correction or enhancements, the reports can be easily regenerated to reflect the changed code. In contrast, manually generated documentation is rarely up to date.

J73AVS CAPABILITIES

J73AVS should play a role in JOVIAL J73 software development as soon as some of the modules are compilable. The source code is generated based upon a design, which in turn is based on a set of requirements. It is recommended that the expected program output (acceptance criteria) and at least an initial set of input test data be generated concurrently with the program's design. The requirements, design, and acceptance criteria play an indirect role in J73AVS's analysis of the software.

The types of J73AVS analysis capabilities are:

- Static and data-flow analysis (symbol usage anomalies and dangerous coding)
- Reporting of program structure and characteristics
- Measuring execution coverage of statements, branches, and program units
- Execution tracing of variables, branches, and program units
- Execution timing
- Structural (branch) retesting assistance
- Test history reporting

Figure 1 shows how the requirements, design specification, acceptance criteria, and test data interact with J73AVS-supported testing. The acceptance criteria are used to judge the proper performance of the program. J73AVS provides detailed source analysis reporting which aids the analyst in determining that certain acceptance criteria are being met. The bold path marked number one in Figure 1 indicates the cycle of static code checking.

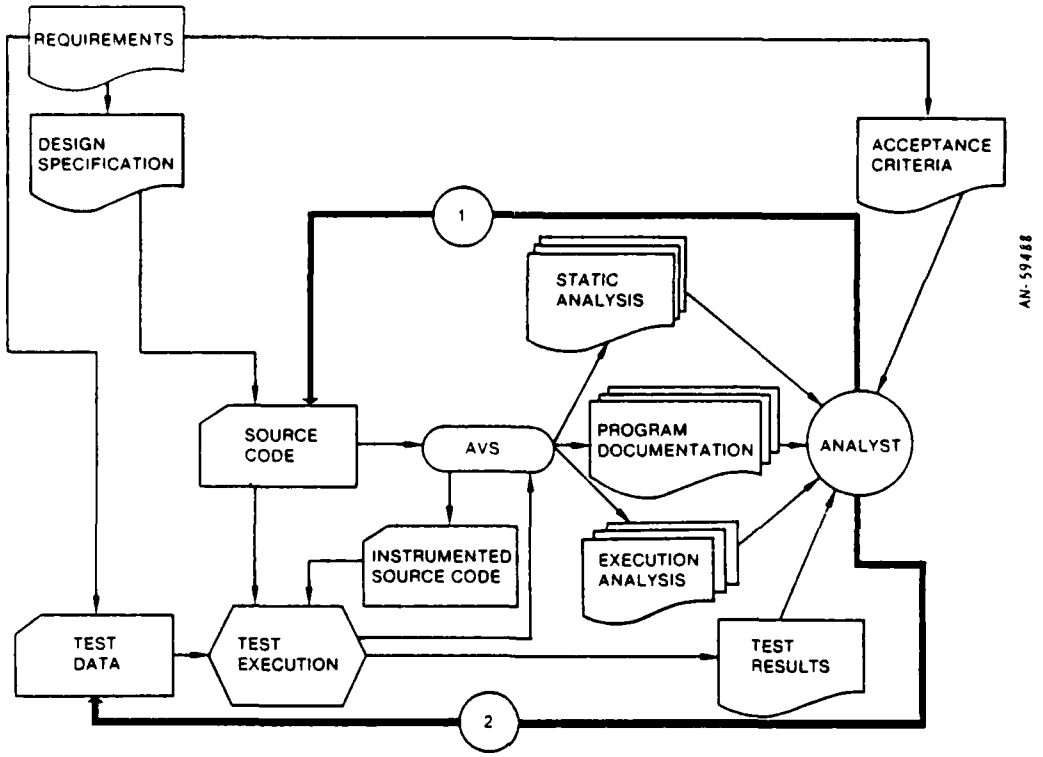


Figure 1. Using an AVS

Once the static errors are removed, the program can be analyzed dynamically by driving it with the initial set of test data. Dynamic analysis is indicated by bold path number two in Figure 1. J73AVS outputs execution coverage, timing, and tracing information, along with the normal program output, which aids the analyst in determining acceptable performance. Unexercised statements and branches are indicated by J73AVS so that additional test data can be generated to ensure that all parts of the code are tested. Dynamic analysis, therefore, is usually an iterative activity that continues until the desired level of exercise is achieved. J73AVS maintains the coverage levels for each test in its database.

As shown in Figure 1, compilable source code generally is first analyzed by J73AVS to detect semantic errors that are outside the scope of the compiler's static analysis capabilities. As each module is analyzed by J73AVS, a database is built that contains single and multi-module detailed characteristics. This database is used and augmented each time additional analysis (static or dynamic) is requested by the user. Thus, J73AVS is a partner in the development, testing, and documentation phases.

It should be pointed out that, because of the database feature, J73AVS supports top-down code development in the following way. High-level modules can be coded early with stubs (module skeletons) for lower-level modules. Both fully coded modules and stubs can be input to J73AVS for analysis and documentation. J73AVS includes the stubs in its database. Module interaction and interfaces (COMPOOL usage and parameter passing) will be analyzed and reported to the extent that they occur in the code. Then, as lower-level stubs are replaced with full source code, J73AVS replaces the modules on the database.

A typical sequence of J73AVS-supported verification of fully coded source modules is:

1. JOVIAL J73 source text, perhaps with assertions (Boolean expressions, recognized by J73AVS, that specify expected behavior), is read by J73AVS as one or more compilable modules.

2. J73AVS produces program analysis reports showing control structure, symbol usage, calling hierarchy, etc., as well as a static analysis report showing errors and dangerous programming practices.

3. Using the reports as a guide, the source modules are changed or new modules are added to the program.

4. J73AVS reports the interaction of the new or changed modules with the rest of the program. This information, in turn, may show the need to modify other modules.

5. For debugging, the program is instrumented by J73AVS and executed with an initial test case supplied by the user.

6. Assertion messages, variable, branch, and module tracing, and execution timing reports can be used for debugging.

7. Using the J73AVS reports, the user chooses to create more test data or instrument other modules.

8. For testing, the same cycle of instrumentation and execution is repeated, but for a different goal: rather than detecting and locating errors, testing aims to demonstrate that the entire program has been exercised to some degree. The J73AVS execution analysis reports show the thoroughness of execution coverage.

9. The user evaluates execution coverage reports, the program's own execution results, and the program specification to determine if testing is complete.

10. J73AVS provides branch sequence information to retest targets chosen by the user. A test history of execution coverage assists the user in choosing targets for retesting.

As just noted, J73AVS can be used to assist with several phases of software development. These phases can be grouped as:

- Program development and maintenance
- Debugging
- Testing
- Retesting assistance

Program Development and Maintenance

Executable assertions provide a means for a programmer to specify expected behavior. Assertions can be used for reporting execution-time exceptions, stress testing, and manual or automated test data generation. When assertions are left as comments in the source code they can be used as inline documentation of the program's specifications. An example of an executable assertions is:

```
". ASSERT (STACK'POINTER >= 0)"
```

To assist with reliable system development and maintenance, J73AVS provides substantial program analysis reporting on structural hierarchy, symbol usage, invocations, certain J73 constructs, and system characteristics. The user has control over obtaining high- or low-level information through the tool's command language.

Debugging

Normal compilation using JOVIAL J73 compilers can detect many syntax and semantic errors. Other errors, such as uninitialized variables, possible infinite loops, unreachable code, certain improper constructs, and dangerous coding practices (like transferring into CASE or IF statements) will be reported by J73AVS. The user can specify the degree of analysis to the error, warning, or message level.

Debugging is supported by assertion exceptions, variable and module execution tracing, and execution timing reports. When the program's execution behavior deviates from the acceptable logical behavior as specified by the assertions, it is immediately reported in the program's output. The user-embedded assertions have no effect on program control flow until they are violated; at that time the violation is reported with the source statement number of the assertions.

Testing

The primary purpose of program coverage analysis is to provide a measure of the level of testing. One measuring technique uses the program's control structure as a guide. Structure-based testing means that the program's control structures are analyzed for execution behavior; that is, whether the structures are exercised and in what order. Structure-based testing can uncover errors due to untested branches or improper sequences of branches. J73AVS provides program unit or branch tracing and analyzes execution coverage of program units, branches, and statements. Further, J73AVS assembles the timing information from program unit tracing and user-directed timing probes into an execution timing report.

Retesting Assistance

Software is retested when analysis shows that prior testing is inadequate (insufficient branch coverage, not all functions demonstrated, etc.) or when program changes have taken place. The proper approach to take in retesting is highly dependent upon the characteristics of the program being tested as well as the measures being used to evaluate testing completeness.

To determine the sequence of branches which lead to an untested branch or statement, the user can request that the "reaching set" be computed between two specified statements (or from a procedure's entry). After the flow of control is identified by J73AVS, the user can backtrack through the program to the actual test data. New test data can be created by using J73AVS module interaction, invocation, and execution coverage reports. Unfortunately, automatic test data generators which use symbolic execution are not yet general enough, easy to use, or reliable. Therefore, J73AVS has no test data generation capability at this time.

The testing history maintained by J73AVS is useful in attaining testing coverage goals and for determining targets for retesting. Procedure invocation and coverage information is saved in a concise way for each test case. The results of subsequent execution runs can be added, providing a cumulative report of all tests.

J73AVS OPERATION

J73AVS operates in either batch or interactive mode on a host computer. If the JOVIAL J73 code being analyzed is destined for execution on a target computer, the J73AVS dynamic analysis operation is modified slightly, as shown in Figure 2.

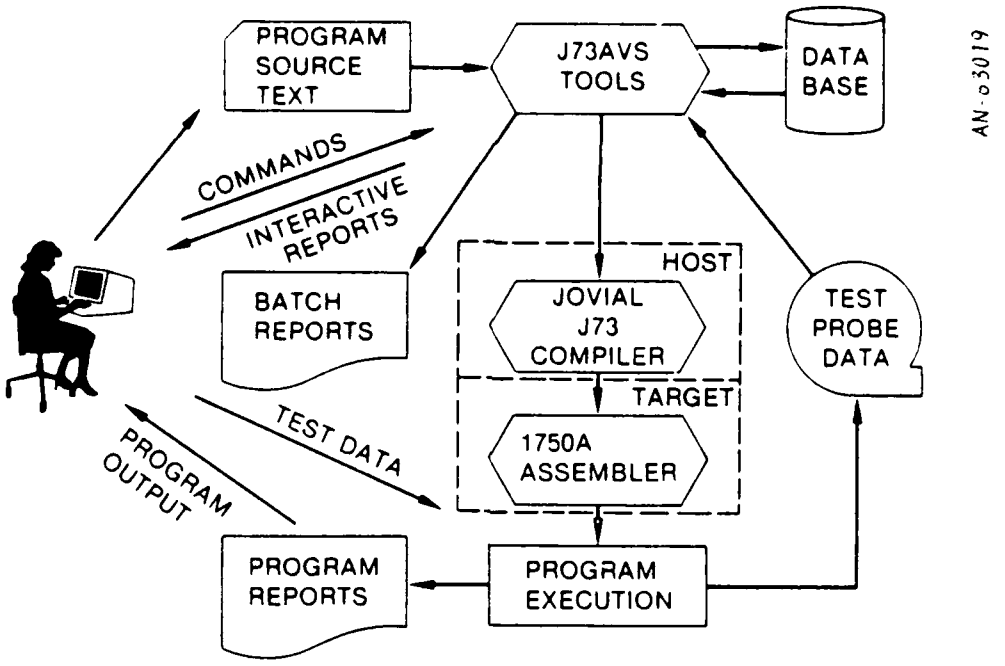


Figure 2. Using J73AVS in a Host-Target Environment

The user directs J73AVS analysis through a simple command language. The basic commands are verbs that select the type of analysis, followed by command parameters that specify the scope of the analysis or level of error reporting. As much as a whole program or as little as a single symbol can be analyzed.

J73AVS displays or prints reports during static analysis. For dynamic analysis, the instrumented source (augmented by expanded assertions and by probes for execution coverage, tracing, or timing) is passed to the JOVIAL J73 compiler. In a host execution environment, input data is read by the program and normal program output is accompanied by an execution data collection file required for the J73AVS post-execution analysis reports. J73AVS uses that file, along with its database, to provide readable, user-selected reports that describe execution behavior.

In a host-target environment, the target computer must have a sequential output device such as a disk or tape to transfer the data collected during execution back to the host. In the absence of any sequential output device, the J73AVS data collection routines can be modified to output test coverage information on the target in an abbreviated manner.

J73AVS was developed for operation on IBM 370 and DEC 20 computers. It is currently being rehosted to the VAX 11/780. J73AVS is written in JOVIAL J73, except for a few small input/output routines written in FORTRAN. The VAX version of J73AVS is written in a structured dialect of FORTRAN.

ACKNOWLEDGEMENT

The design, development, and current rehosting of J73AVS is being sponsored by Rome Air Development Center, Griffiss Air Force Base, New York, under Contract F30602-79-C-0265. The project officer is Frank LaMonica, RADC/COEE. The VAX rehosting effort is being funded by the Embedded Computer Standardization Program Office (ECSPO), ASD-AFALD/AXS, Air Force Wright Aeronautical Laboratory.

REFERENCES

1. L. Brownell and R. J. Gilinsky, JOVIAL (J73) Code Auditor User's Manual, Proprietary Software Systems, Inc., 16 March 1982.

2. Maj. D. Burton, S. May, and T. Fujawa, "A JOVIAL Interactive Debugger," NAECON 82, pp. 1121-1129.

3. JOVIAL J73 Programming Support Library User's Manual, SofTech, Inc., Jan. 1982.

4. C. Gannon and R. F. Else, JOVIAL J73 Automated Verification System User's Manual, General Research Corporation CR-4-947, November 1981.

5. RXVP80™ User's Manual, (Preliminary Draft) General Research Corporation RM-2419.

6. P. Roberson, R. Melton, and C. Andrews, COBOL Automated Verification System User's Manual, General Research Corporation CR-4-970, May 1982.

APPENDIX B
TERMS AND ABBREVIATIONS

The following terms pertain to software verification and to the characteristics of JOVIAL software.

Assertion -

Statement of a condition that must be true whenever control reaches that point in the program.

AVS -

Automated Verification System. A computer program or collection of programs which assists in verifying the correspondence between software and the set of functional specifications defining the software.

Branch -

A continuously executable sequence of statements between two decision statements. It may include unconditional transfers.

Branch testing -

A testing technique that measures the number of executions of each branch in a module and computes a measure for testing thoroughness in terms of the fraction of all branches executed at least once.

Data Flow Analysis -

A program analysis technique which tracks the usage of symbols through a program. The technique is used to detect uninitialized variables and other program anomalies based on symbol usage and control flow. The program is not actually executed.

Dynamic Analysis -

A debugging or testing technique in which the program is executed with data and the program output, together with any additional execution-time reports, is analyzed for conformity to functional or structural performance specification.

Instrumentation -

The technique of automatically inserting software probes (subroutine calls or counters) into code or of translating special statements (like an assertion) into executable code for the purpose of collecting information during program execution.

Interface Description -

Information in the AVS describing global data passed between modules.

J73AVS -

An Automated Verification System for JOVIAL J73 Software.

JOVIAL J73 -

A JOVIAL programming language as defined in MIL-STD-1589B for command and control, avionics, and defense systems applications.

Module -

The smallest entity in JOVIAL J73 that can be separately compiled. (Note the difference between module and program unit.)

Path -

A continuous sequence of control flow (branches) between two points in a program (usually between a program unit's entry and exit).

Program Unit -

The smallest entity in JOVIAL J73 that can be invoked, or, in the case of compools, referenced by name. In JOVIAL J73 program units are compool-modules, main-programs, procedures, and functions.

Reaching Set -

A set of statements that incorporates all of the branches leading to a specified statement.

Static Analysis -

A technique which analyzes program source but does not actually execute the program. Program statements and symbols are analyzed to detect inconsistencies in semantics or in asserted versus actual conditions.

Test Target Selection -

The process of selecting a module, or a set of statements within a module, to be executed with data for the purpose of improving the quality of testing. Goals for improving the quality of testing may be exercising unexecuted branches, paths, or statements, executing all boundary conditions, etc.

APPENDIX C

J73AVS Processing and Reporting Commands

Command (Defaults Underlined)

```
READ {,ECHO}

STATIC {,DATA} {,SUMMARY/FULL}
       {,LIST=ERRORS,WARNINGS,MESSAGES}

LIST
LIST,DATABASE {,UNITS}

INSTRUMENT {,ASSERTIONS} {,COVERAGE} {,TRACE=BRANCH/ENTRY}
INSTRUMENT,NEWTEST,<m-name>,<stmt>
INSTRUMENT,CLOCK,ON=<m-name>,<stmt>,OFF=<m-name>,<stmt>
INSTRUMENT,VARIABLE,<v-name> {,<start-stmt>},{<stop-stmt>}
INSTRUMENT,GO.

DOCUMENT,INVOCATIONS {,SOURCE} {,BANDS/BANDS=<n>}
       {,TREE} {,GLOBAL}
DOCUMENT,SYMBOLS {,SOURCE/XREF} {,NAMES=<s-name>,...}
DOCUMENT,LABELS/TYPES/CNSTANTS/COMPOOLS/REFDEF {,SOURCE/XREF}
       {,NAMES=<name1>,...}
DOCUMENT,DEFINES {,XREF/EXPANSIONS/FULL} {,NAMES=<name1>,...}

EXECUTION {,SUMMARY} {SOURCE} {,NOTHIT} {,TIMING}
EXECUTION,GO

ASSIST,BRANCHES{{,<first stmt>},<last stmt>{,ITERATIVE}}
ASSIST,HISTORY{,RESET}
```

note: {} indicates optional parameter
<> indicates user-supplied name
/ indicates selection of one parameter

All command keywords and parameters can be abbreviated to two or more letters.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.

END

FILMED

1-85

DTIC