DTIC FILE COPY

(2)

NAVAL POSTGRADUATE SCHOOL
Monterey, California

AD-A231 955

DTIC
ELECTE
FEB 14 1991
S B D

THESIS

VIEWER
A USER INTERFACE FOR
FAILURE REGION ANALYSIS

by

Vicki Sue Abel
and
Medio Monti

December, 1990

Thesis Advisor: Timothy Shimeall

Approved for public release; distribution is unlimited.

91 2 12 060

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Computer Science Dept. Naval Postgraduate School; 6b. OFFICE SYMBOL (if applicable): 37	7a. NAME OF MONITORING ORGANIZATION: Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
VIEWER A USER INTERFACE FOR FAILURE REGION ANALYSIS(U)

12. PERSONAL AUTHOR(S)
Vicki Sue Abel Medio Monti

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Master's Thesis	FROM 03/90 TO 12/90	December 1990	271

16. SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Failure Regions User Interfaces
Testing Tools

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

Two issues gaining interest in the computer community are the development of software testing tools and the increase of graphical user interfaces in all types of software. VIEWER is a program that provides support to a set of tools that work in an integrated fashion to analyze Pascal programs to determine the failure regions associated with identified faults in the program. It is a graphical user interface that facilitates the process of analyzing the program. It provides automated coordination between the tools and as such maintains a certain level of abstraction for the analyst. It allows for rapid and customized improvement in the automation of the analysis process.

The thesis discusses the background involved in testing tools, user interfaces, and the combination of the two into a useful tool. An implemented prototype is discussed and an example of failure region analysis performed with the graphical user interface is included.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: [X] UNCLASSIFIED/UNLIMITED [ ] SAME AS RPT. [ ] DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL: Timothy J. Shimeall	22b. TELEPHONE (Include Area Code): (408) 646-2509; 22c. OFFICE SYMBOL: CSSm

Approved for public release; distribution is unlimited.

VIEWER

A User Interface for

Failure Region Analysis

by

Vicki Sue Abel

Lieutenant Commander, United States Navy

B.A., University of Dallas, 1979

and

Medio Monti

Captain, United States Marine Corps

B.S., Allegheny College, 1979

Submitted in partial fulfillment

of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

December 1990

Authors: Vicki Sue Abel

Vicki Sue Abel

Medio Monti

Medio Monti

Approved by: Timothy J. Shimeall

Timothy Shimeall, Thesis Advisor

Rachel Griffin

LCDR Rachel Griffin, Second Reader

Robert B. McGhee

Robert B. McGhee, Chairman

Department of Computer Science

ABSTRACT

Two issues gaining interest in the computer community are the development of software testing tools and the increase of graphical user interfaces in all types of software. VIEWER is a program that provides support to a set of tools that work in an integrated fashion to analyze Pascal programs to determine the failure regions associated with identified faults in the program. It is a graphical user interface that facilitates the process of analyzing the program. It provides automated coordination between the tools and as such maintains a certain level of abstraction for the analyst. It allows for rapid and customized improvement in the automation of the analysis process.

The thesis discusses the background involved in testing tools, user interfaces, and the combination of the two into a useful tool. An implemented prototype is discussed and an example of failure region analysis performed with the graphical user interface is included.

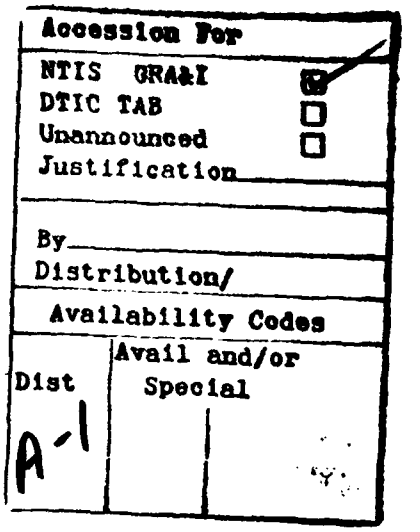

TABLE OF CONTENTS

I. INTRODUCTION 1

A. TOOL TAXONOMIES 2

B. TYPES OF TESTING 4

C. ENVIRONMENT OF TESTING TOOLS 9

D. INTERFACE DESIGN 10

1. Strive for consistency 13

2. Enable frequent users to use shortcuts 13

3. Offer informative feedback 14

4. Design dialogues to yield closure 15

5. Offer simple error handling 15

6. Permit easy reversal of actions 16

7. Support the internal locus of control 16

8. Reduce short-term memory load 17

E. TESTING TOOLS 17

1. EFFIGY 19

2. ASSET 21

3. Mothra 21

F. PROBLEM STATEMENT 23

G. OVERVIEW OF REMAINING CHAPTERS 24

II. TOOL DESCRIPTION 25

A. TOOL OVERVIEW 25

B. VIEWER OVERVIEW 27

C. SUNVIEW OVERVIEW 28

1. VIEWER Internal Structure 32

D. WALKTHROUGH OF FUNCTIONS OF MAIN PARTS 36

1. VIEWWIN 36

a. Declarations 36

b. Frame and Subwindows 37

c. Procedures 40

2. WALKWIN algorithms 44

a. Declarations 44

b. Frame and Subwindows 44

c. Algorithms 46

(1) Circles and lines 46

(2) Graphical Representation of ACFG 47

(3) Buttons and Menu 49

E. CONCLUSION 50

III. EXAMPLE VIEWER SESSION 54

A. ASSUMPTIONS 54

B. EXAMPLE 54

1. Initiate SunView 54

2. VIEWER 55

a. Initiating VIEWWIN 55

b. Selecting Analysis Tools 56

3. WALKWIN 56

a. Initiating WALKWIN 56

b. Starting WALKER 60

c. Traversing the graph 62

d. Saving work 78

e. Running LISTER 80

4. FALTWIN 80

a. Initiating FALTWIN 80

b. Traversing the graph and setting the fault 82

c. Saving work 88

5. SPACEWIN 90

IV. CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE RESEARCH 91

A. CONCLUSIONS 91

B. LESSONS LEARNED 92

C. RECOMMENDATIONS FOR FUTURE RESEARCH 93

APPENDIX A 96

APPENDIX B 105

APPENDIX C 113

APPENDIX D 131

APPENDIX E 206

APPENDIX F 208

APPENDIX G 226

APPENDIX H 239

APPENDIX I 242

LIST OF REFERENCES 256

BIBLIOGRAPHY 258

INITIAL DISTRIBUTION LIST 260

ACKNOWLEDGEMENTS

We begin our acknowledgements with heartfelt thanks to our thesis advisor, Timothy Shimeall. His patience is beyond measure, and he was able to keep us motivated during the long process. His guidance and wisdom were our mainstay for the past months. We thank our second reader, LCDR Rachel Griffin, for her wonderful suggestions along the way.

We appreciate the support from those near and dear to us. Jim Merry provided insight and assistance in the use of WordPerfect, and the creation of several key figures in the document. Kathy Monti bore two daughters (Natalie and Lia) days before completion of this thesis, and in the months leading up to delivery (of babies and thesis) was a source of inspiration to us by her cheer and patience.

We appreciate also the support of our classmates through the past months, as well as from the beginning of our studies here at the Naval Postgraduate School.

I. INTRODUCTION

Research in software system development continues to search for methods to improve the quality and reliability of systems at reduced cost. Software continues to be the major cost aspect in total system development. Large systems consist of many components with complicated interactions. The focus of testing is to prevent bugs or, if it cannot prevent bugs, to detect them in software. (Beizer, 1983, pp. 1, 3) The size and complexity of large software systems make thorough testing difficult. Software system failures continue to surface in some of the most thoroughly tested systems in the world like those at AT&T and NASA.

Software testing tools in themselves are a recent development in the life time of computers and programs. The future of software engineering is an integrated environment containing all the development and testing tools necessary for software engineers to produce reliable, efficient, and cost-effective software. These types of environments, on the development side of software engineering, have become synonymous with the term CASE. Most CASE texts don't cover the testing side of the development project. Fisher talks about "stress testing" on a system, using a testing harness to test individual modules. Separate quality assurance groups (the testers) usually build these test harnesses. (Fisher, 1988, p. 48) Few authors of CASE tools address the complete software testing package needed to validate thoroughly future large scale

projects. Even code generated via future specification languages will still need thorough testing, to ensure that the specified system meets operational needs.

Unfortunately software testers have a long way to go before they can realize this dream. Adrion, Branstad, and Cherniavsky reported in June 1982 that of the methods used to test software systems, "the most successful have been the disciplined manual techniques, such as walk-throughs, reviews, and inspections, applied to all stages in the life cycle." (Adrion, Branstad and Cherniavsky, 1982) Their observations are still true today. The human mind is flexible, adaptable, portable and robust when analyzing code. It works well for small systems. For large systems, with their attendant large amount of detail affecting reliability, only automation will make thorough testing feasible, practical, efficient and cost-effective.

Automation requires formalization of development and design of software. But in testing there has been little progress toward developing a theoretical basis from which to relate software behavior to validation and verification. (Adrion, Branstad and Cherniavsky, 1982) This lack of a sound theoretical foundation is a problem with which researchers continue to struggle.

A. TOOL TAXONOMIES

Before examining any specific tool or environment now available, a brief look at a taxonomy of automated tools is in order. Ramamoorthy and Ho broadly classify automated tools under three groups: operational mode, application, or function. The operational mode grouping partitions tools into static or dynamic analysis methods.

Static analysis analyzes the logic and structure of a program without running the program, and looks for properties determinable from the source code alone. Dynamic analysis requires the tester to execute the program, observe its behavior, and look for runtime properties.

The application grouping partitions tools by their use in a specific phase of development. These tools should span the entire spectrum of the software development cycle when used correctly. A software development system might include tools for design analysis, debugging, testing and partial validation.

The functional classification partitions tools by their specific functions. For example, static analysis tools would include: code analysis, program structure checks and module interface checks. Code analysis tools perform syntax analysis on the source code, extracting information used to find error prone constructions, and module relationships. For instance, a compiler is a complex example of a code analysis tool. (Ramamoorthy and Ho, 1975) Program structure analysis tools are essential for software validation. This analysis technique often includes program graph generation, modelling a program as a directed graph with nodes representing the flow of control.

Code analysis and structure analysis are module tests that analyze the active module. Module interface tools look for various anomalies across module boundaries and are global checks. (Ramamoorthy and Ho, 1975)

Dynamic analysis tools might include: runtime behavior monitors and test case generation. Testing tools that monitor a program's runtime behavior can trace a variable

throughout the execution of the program, or record paths executed by particular test cases.

B. TYPES OF TESTING

Functional testing views the program or system as a black box. Functional testing provides inputs to the program or system, and verifies the outputs for conformance to preset objectives. Functional testing normally takes the user's point of view. Structural testing, on the other hand, deals with the implementation details of the program or system. Included in the considerations are the style of programming, the source language, and the coding details. The boundary between the two is fuzzy, and the user will see only the outermost layer. (Beizer, 1983, pp. 4 - 5)

Since exhaustive testing is impractical, Ramamoorthy and Ho define "complete" testing in more relaxed terms. For example, one might consider a program "tested" if all executable statements execute at least once. This is just one possible definition of complete testing. Different testers may find their own definitions of necessary and sufficient test cases. Automation suits complete testing.

Formal proofs of correctness rely on both functional and structural concepts. A formal language describes the requirements, and an inductive proof uses each statement to show that all possible input sequences will produce correct output. These proofs are very expensive, and normally limited to programs like a security kernel for a system or other crucial software. (Beizer, 1983, p. 14) At present, a formal proof of program

correctness is infeasible for large systems. A program in Algol with 433 statements required 46 pages of proofs. (Ramamoorthy and Ho, 1975)

Structural and functional testing are not contradictory ideas. There is a time and place for use of each. Both methods are effective, though each has certain limitations. In general, system testing is functional, while unit testing is structural in nature. In principle, a functional test could detect all bugs in a program, but it would require an impractical amount of time. On the other hand, while structural tests are finite they cannot detect all bugs even if executed to completion. (Beizer, 1984, p. 8)

To have a complete functional test, input must include all possible input streams. In each case, the program would either accept or reject the input stream. If the program accepts the stream, it either produce a correct output or an incorrect output. If the program rejects the stream, it should produce an output showing the rejection, so in every possible case there would be some sort of output. (Beizer, 1983, p. 13) A command line interface would suffice in this particular set of testing, as the user could enter data from the keyboard or from an input file. A way to capture the output would be necessary in order to provide the capability of running several input streams at once, and then evaluating the results later.

Debugging tracers are dynamic software tools that tell the user what happened during a particular run of the program. When debugging tracers use a certain set of input data, they can show what parts of the program execute in what order. There has been much development in this area, and most mainframe compilers are capable of

supplying runtime traces at varying degrees of sophistication. At times the process of debugging is so difficult, the programmer welcomes any assistance. (Green, 1981)

The use of path testing is the cornerstone of all testing. It focuses on the control structures of a program. For our purposes, a path is any executable sequence of instructions or statements that go through a routine. To be more specific, the path will start at a junction (the program entrance or designated statement) and proceed until it reaches another junction (the program exit or designated statement). The goal is to show that the actual behavior of the routine matches its intended behavior. Testers show this by exercising multiple paths, but in practice the number of paths is often infinite. Each decision doubles the numbers of paths, each case statement multiplies the number of paths by the number of cases, and each loop has a potentially infinite number of cases. So achieving absolute assurance is not only unrealistic but unachievable as well. Therefore, path testing is accomplished over a smaller set of paths, carefully selected so they meet the above stated goal. If unable to accomplish that, they should at least catch most bugs a unit-level test would catch. (Beizer, 1984, pp. 37 - 38)

Selective testing exercises a program with data prepared by the programmers. Because the programmers prepare the data to test their own code, they unavoidably bias the set of test inputs.

Exhaustive testing requires the tester to execute every executable path at least once. Exhaustive testing is impractical and unrealizable usually because of the enormous number of paths requiring execution. The number of executable paths grows

exponentially and testing them even for small programs would exceed the lifetime of most testers.

Each path will have associated with it the inputs that force the execution of that particular path and the results (outputs, state changes, database changes) for that path. Tactics should include sufficient paths for coverage of the routine/program and a selection of paths that are short and functionally sensible. The number of changes from path to path should be kept to a minimum. It is better to use a greater number of simple paths than a small number of complex paths. (Beizer, 1984, p. 41)

One way to obtain path confidence is with instrumentation, the simplest of which is running the routine under a trace. We encounter several problems with this scenario:

- A typical trace produces voluminous output. The tester gets more data than he wanted. (Beizer, 1984, p. 47) Studying the source listing simplifies the process of analyzing this data and determining an answer to a given question. (Green, 1981)
- Traces involve a very high overhead, which may destroy timing relations in the program. Interrupt handlers, device handlers, communication interfaces, and other such routines may not work in the trace mode.
- Because a trace may change location dependencies, bugs may exist when the program runs without the trace and disappear with a trace. (Beizer, 1984, p. 47)
- These programs are usually not interactive. (Green, 1981)

A recent development in software testing is the study of failure region analysis. This is the study of errors occurring in proximity to other errors, which Myers referred to as error clustering. Ammann and Knight described this error clustering as "failure regions." (Bolchoz, 1990, p. 3)

A failure region is a region of a program's input space, mapped by faults in the program's source code, to failures in the output space. The purpose of failure region analysis is to improve testing efficiency by reducing the number of redundant initial tests, preventing redundant multiple error detection. Failure region testing may also act as a guide for later regression testing.

Upon isolating a failure region, the tester can omit any test cases that fall within that region from the testing process until the programmers correct the fault. Programmers correct faults by changing the source code. The tester can then use the failure region as a guide for testing the correctness of changes. By incorporating failure region testing into the complete testing package, testers can avoid duplicating test cases and proceed on to as yet unexplored test cases. In this way testers can uncover multiple errors before sending a program back to the programming team for corrections.

Inequalities derived from the combination of three sources bound the failure region. Those sources are:

- The reachability condition for the code with the defect.
- The conditions under which the calculations in that defective code produce an erroneous result.
- The conditions in which later process does not mask the erroneous value

Research efforts are underway to automate the failure region analysis process. The motivation for this thesis is the research effort to integrate these automated failure region

tools into a graphical user interface. Later chapters discuss the tools and environment in more detail.

C. ENVIRONMENT OF TESTING TOOLS

Future program development environments will provide the tester with many tools designed to ease the program testing process. These include lexical analyzers, dataflow analyzers, automated instrumentation, smart compilers, automated test data selection, and knowledge-based program testing aides. This is a relatively new area of research and new software testing technology will lead to new testing tools.

During the early 70's reliability and quality-assurance concerns in software engineering gave rise to systematic testing procedures, notations of formal program correctness, and models of fault tolerance and total system reliability. Analysis of actual allocation of development effort and expense appeared. The 80's saw the rise of CASE, with the development of automated tools corresponding to each phase of the software life cycle, appearing on stand alone workstations. In the area of software testing tools, there are few articles addressing how to set the foundation for these types of environments to enable testers to add/delete tools to/from the environment at arbitrary points in time. (Lewis and Oman, 1990)

This critical part of the software life cycle must continue to evolve. Part of this evolution is the recent emphasis on human-computer interaction. A survey of 15 representatives of industry and academia revealed that the pervasiveness of graphical user interfaces, including speech and other ways to interact with a computer is a clear sign

that all software of the future must address the users' needs for ease of use. The designers of a software testing tool support environment must consider the needs, desires and frustration of the individual tester. As simple as it sounds, the environment should provide the tester with information that helps analyze the program. Many tools now available output large amounts of confusing data with little real analytical value. The environment designer should try to help the tester focus on what is important, and remove as much ambiguity from the analysis as possible. (Lewis and Oman, 1990)

D. INTERFACE DESIGN

Most testing tools are research efforts, and give little consideration to the interface design. This makes them difficult to use. Tool environments should use current interface technology so users won't have to rely on short term memory to use the system. The system should supply all the support and documentation users need.

The interest in designing "good" user interfaces has grown in recent years, as computers must serve a more diverse group of people with varying technical expertise. Before the proliferation of personal computers, there were custom-built programs and personnel specially trained to use them. It was easier to train the few users than to build software that worked naturally with the user. Now user interface quality has become a primary issue, and phrases like "user-friendly" are everyday language. (Brown and Cunningham, 1989, p. 1)

Human-computer interaction software is a challenging endeavor. There are three developments that have altered this field. First, there now exist powerful workstations

with bitmapped screens and pointing devices. These provide the technological base for the design of the interface. The computer's interactive capabilities have drawn attention as applications have become increasingly innovative. Last, because today's software is becoming more complex, the ability to communicate better with the computer is no longer just a luxury; it is a necessity. (Fischer, 1989)

In writing human-computer interface software, we must begin with the human as the fixed point. We must distinguish between what the computer will do and what the human will and can do. During this process the designer will make assumptions about what the user will want to do. While making these assumptions, the designer must remember that each user is an individual, with their own talents, goals, knowledge and preferences. (Fischer, 1989)

How do people process various sorts of information displays? How do they use this information to solve problems? The study of human factors has concerned the physical aspect of the interface. For hardware, this has produced rules saying to space and align the knobs properly and design the displays and workstation layouts based upon the idiosyncrasies of human anatomy. Researchers have labeled this area of research "cognitive ergonomics". Ergonomics is an accepted term meaning "the study of the problems of people in adjusting to their environment; especially, the science that seeks to adapt work or working conditions to suit the worker." It now must include problem solving and the analysis of information displays. (Curtis, 1981, p. 1)

Can one describe what makes a good interface? Every user of a computer has a different idea of what makes a "good" user interface. Begin with the definition of user

interface - it is not simply a collection of code objects. A user interface is a way of ensuring that the user can work effectively with the software. (Brown and Cunningham, 1989, p. 1)

A user interface should handle all input and output. It should convey all the instructions about the program's use, and allow the user to control the program as naturally as possible. The interface also should provide the program's results to the user. (Brown and Cunningham, 1989, p. 5)

While there are no complete human-computer interface theories, researchers have proposed methods and principles. (Fischer, 1989) Shneiderman postulates eight rules of design for user interfaces. They are:

- Strive for consistency.
- Enable frequent users to use shortcuts.
- Offer informative feedback.
- Design dialogues to yield closure.
- Offer simple error handling.
- Permit easy reversal of actions.
- Support the internal locus of control.
- Reduce short-term memory load. (Shneiderman, 1987, pp. 61 - 62)

Now an examination of each of these principles.

1. Strive for consistency

This applies to single actions and sequences of action, and may involve prompts, menus and help screens. Interfaces should keep exceptions to a minimum and be comprehensible to the user. (Shneiderman, 1987, p. 61) Consistency is the mental model the user has of a particular application. It is the way in which the user controls the application. Besides consistency, the interface should provide some flexibility to the user, though the ideas of flexibility and consistency are not without some contradiction. (Brown and Cunningham, 1989, p. 9) The user should feel comfortable when entering the information, for an awkward input design is very noticeable. (Brown and Cunningham, 1989, p. 15) If there are several modes available within the single application, consistency becomes more difficult. The best form of consistency is in a modeless application. If the modeless aspect is too hard, make the primary application the standard mode, so the exceptions will be minimal. (Brown and Cunningham, 1989, p. 23)

2. Enable frequent users to use shortcuts

As the user gains knowledge of the program or system, they can reduce the interaction with the interface, and increase the speed of the interactions that remain. A knowledgeable user will want special keys, abbreviations, hidden commands, shorter response times and faster display times. (Shneiderman, 1987, p. 61) Humans are not static beings. Though all users begin as novices, and some may remain in that classification, they also may progress into the casual or expert user class. (Fischer, 1989) The options for input are recognition memory or recall memory. If both options

are available to the user, it will allow for greater flexibility. The recognition memory will associate the command choice with a given action, as in a menu. For the more experienced user, the command line options rely on the user's recall memory. (Brown and Cunningham, 1989, p. 22) Command systems are more complex than menus, require more training to use them, and require the user to possess some typing skills. The interface requires more error handling capability, since input is unlimited. Command line interfaces are usually fast and flexible, and preferred by an expert user when they require speed or special functions. A menu system limits the user to the choices displayed on the screen. While the error handling need not be as intensive, they are usually slower and less flexible than the command line mode. (Brown and Cunningham, 1989, pp. 25 - 26)

3. Offer informative feedback

Each action by the user should have feedback from the system. The level of response should equate to the action performed. A visual presentation can provide an environment for showing changes. (Shneiderman, 1987, p. 62) Users judge new computer systems more on the quality of their communication capabilities than on their processing speed and their memory size. The human-computer interfaces of the future should take advantage of the technology now available. The modern workstations provide many possibilities, yet many interfaces are still command line oriented. It will take an effort to take advantage of these possibilities in the design of future software. (Fischer, 1989)

The human-computer interface should help the user make intelligent choices. In every communication there is a speaker and a listener role. The speaker presents information and the listener tries to understand the information. The listener must be the more intelligent of the two, because he must not only understand the situation, but how the speaker presents it. In a human-computer interface, the human is the most intelligent agent. The computer should provide the appropriate cues to allow the user to choose the next step. (Fischer, 1989)

4. Design dialogues to yield closure

A sequence of actions should consist of a beginning, a middle and an end. The system should provide informative feedback at the end of a group (closure). The feedback provided can give the user a feeling of accomplishment or relief, and suggesting that they may prepare for the next sequence of actions. (Shneiderman, 1987, p. 62)

5. Offer simple error handling

The system should, as much as possible, prevent the user from committing an error. When an error occurs, the system should detect the error and offer a simple explanation of the remedy, or use simple mechanisms and handle the error. The user should only have to correct the command, not reenter it completely. Also, the program state should not change because of erroneous commands. (Shneiderman, 1987, p. 62) Any error messages should have enough information contained in them that the user feels comfortable with the message, and understand what they need to correct the condition. The programmer of the software may be content with cryptic messages, but the normal

user will not tolerate wording that makes them feel inadequate, or does not provide the information they need to correct the error.

6. Permit easy reversal of actions

While it is sometimes not possible to reverse every action, most actions should include an "undo" option. This can relieve user anxiety, and give them the freedom and flexibility to explore new portions of the software without fear of committing an irreversible action. (Shneiderman, 1987, p. 62) This will create an exploratory environment for the human user. A user may not know what they want to do, while the designer of the human-computer interface may not understand what the user needs or will accept. (Fischer, 1989)

7. Support the internal locus of control

Both novice and experienced users prefer to feel they are in control of the system and it is responsive to their actions. The user should be the initiator of the action, and the system the responder. (Shneiderman, 1987, p. 62) For a computer to be a truly useful tool, it should be invisible to the user to allow them to concentrate on their problems and tasks. (Fischer, 1989) To provide comfortable program control, the interface needs to understand the audience for the program. The designer must know how the program's intended users think about the program's problem and how they would go about solving it. The interface should provide a command system that allows the user to solve the problem with as little reference to the computer or the program as possible. (Brown and Cunningham, 1989, p. 23)

8. Reduce short-term memory load

The human information processing limit of "seven plus or minus two" means displays should be simple. Designers should reduce window motion, and there should be adequate training time for codes and mnemonics. They should provide help in the form of on-line access to syntax forms, abbreviations and code. (Shneiderman, 1987, pp. 61 - 62) Humans have some limitations the human-computer interface design should accommodate. Humans have only two hands, so it will not suffice to have an interface that requires two hands on the keyboard while concurrently using a pointing device such as a mouse. Humans have weaknesses (limited short term memory and execution errors) but they have strengths (powerful information processing and visual systems). Memory is a psychological factor of the human mind. There is a difference between recall and recognition memory. Command line interfaces use recall memory, while a menu driven interface uses recognition memory. (Fischer, 1989) Even an expert user will have times when they are away from a particular application. The interface can assist them while they are returning to their former level of expertise.

E. TESTING TOOLS

The basic tool for the programmer performing the tests is a description of the program. This may be a listing or a diagrammatic representation. (Brooke and Duncan, 1981) Is it always true that a diagrammatic notation will be easier to read than a conventional one? NO! Some diagrams are good, while others are not. One advantage is that it is easy to learn the conventions used in the design of a diagram, and a diagram

makes an excellent communication device for the less experienced user. The designer must have the right image of the user before using the diagrammatic approach to construct the user interface. (Fitter and Green, 1981)

Computing concerns processes, so the diagrams would need to be able to explain processes. Fitter and Green propose five principles for the development of diagrammatic notation. The information represented should

- be relevant
- be restricted to forms that are comprehensible
- redundantly recode important parts of the program
- reveal underlying processes they represent, in a responsive interactive system allowing manipulation of the diagrams
- have a readily revisable notation. (Fitter and Green, 1981)

Designing a human-computer interface through a standard programming language forces one to use a few primitives (read, write, format) based on a linear stream, vice a two dimensional screen. It is a massive undertaking for the designer to build the interface from low-level components. However, there are environments that are functionally rich in abstractions. There are different classes of windows, screen layout tools, and the designer may reduce the size of the application. What is the cost involved? The designer must learn the abstractions and understand their use, but this is a one-time cost for each designer. (Fischer, 1989)

There are construction/tool kits available for the design of human-computer interfaces. The interface is separate from the application, but usually this is an acceptable approach. Advantages of using a "toolkit" include the availability of an environment for rapid prototyping of a class of interfaces, a high quality interface achieved at a low cost, and architectures that provide uniformity and extensibility. (Fischer, 1989)

In the design of the interface, the designer should consider the use of prototypes. Since a detailed specification probably does not exist, and the interaction between the user and the system is a dynamic relationship, the prototype will allow the user a chance to "play" with the prototype interface. The user then provides feedback to the designer before extensive coding of the actual interface. Concern for the human should be the driving force in the design of the human-computer interface. The interface should be comprehensible to the user and use a natural communication process. (Fischer, 1989)

1. EFFIGY

EFFIGY is an interactive testing and debugging system designed by IBM. EFFIGY is a symbolic executer. Instead of supplying specific constants as input values, the tester supplies symbols and runs the program on the classes of input. When the control flow of a program is input dependent, EFFIGY does a case analysis, producing output formulae for each class of inputs determined by control flow dependencies. Appendix A is an EFFIGY session script with comments in italics. (Ramamoorthy and Ho, 1975)

EFFIGY is a good example of what many command-line interfaces looked like 15 years ago and still look like today. Limitations of the system hardware and operating system had some effect on the look of EFFIGY. The concerns of the designers at IBM were probably more on what EFFIGY did than how it presented its findings. The system prompts the user with an "▸" symbol. All the lines that begin with this symbol in the script are user input. Very little else went into the interface design. The system has no menus or help screens, which makes for a very steep learning curve for novice users. Experienced users should find the interface adequate. EFFIGY allows the user to go through the program and choose the path for symbolic testing. It provides informative feedback to the user provided the system receives correct input. There is no reversal of actions in EFFIGY. The user, once he masters the system, could find the output valuable. EFFIGY runs on CMS under VM/30 on an IBM 370 model 168. The CMS filing system and context editor are an integral part of EFFIGY for creating, changing and storing procedures and command files. Considering when IBM developed the system, it produced interesting output through a fair interface.

In recent years the UNIX operating system has been the development environment of most software testing tools, probably because of its widespread use in software development in colleges and universities. It is flexible and potent to work in, but demanding upon the novice user. Its proliferation made it necessary for inexperienced users to access UNIX's rich functionality. The predilection toward better UNIX interfaces is catching on with the introduction of new, more advanced hardware.

2. ASSET

ASSET (A System to Select and Evaluate Tests) is an interactive software testing tool based on dataflow testing. (Frankl, 1987, p.1) ASSET is still a "work in progress" designed to work in the UNIX environment. ASSET accepts syntactically correct programs written in a subset of Pascal. Appendix B is a script from an ASSET session. ASSET uses the command-line format, incorporating help messages, menus and consistent prompts. ASSET offers informative feedback to users when they make mistakes. It allows users to correct and continue processing, providing contained choices where necessary. The ASSET "command level prompt" is distinct from sub-prompts to help the user identify where he is at any time. The type of data that ASSET provides the tester is useful and interesting. It automatically instruments the program before compilation, and helps the tester by tracking which paths in a procedure or function the tester has yet to exercise. It also produces a graphical representation of the paths available in a procedure or function. This graph is useful when analyzed with the copy.p file. ASSET and tools like it have potential as part of the integrated environment discussed earlier. Some researchers are beginning to design their interfaces so other testing tools can eventually integrate into their environments. The Mothra testing tool is an available tool with this vision.

3. Mothra

Purdue University's Mothra is an integrated set of tools and interfaces that support the planning, definition, preparation, execution, analysis and evaluation of tests of large systems. (Lutz, 1990) Mothra lets you:

- selectively and systematically create mutant programs
- execute the mutants on test data
- compare the resulting mutant output with that of the original program
- measure test data adequacy

The basis for Mothra is mutation analysis, which evaluates the adequacy of test data used while testing a program. It does this by introducing syntatically correct changes into a program, where each change represents a likely error that the programmer might make. (Lutz, 1990) Appendix C is a script of a session with Mothra run on the VAX 711/85. A windowed interface exists for Mothra with additional graphics enhancements still under development. The designers of Mothra developed a much more sophisticated interface than the previous tools mentioned. Mothra makes extensive use of menus and sub-menus throughout, that jog the user's memory. It is a sophisticated tool that requires time to learn. Like ASSET, there are consistent sequences of actions and similar terminology for prompts, menus and help screen. Appendix C shows feedback from Mothra. The system is constantly reporting back to the user what is happening during each phase of mutant generation and testing, providing informative feedback to the user. Mothra allows the user to return to previous menus so actions are reversible. It also has on-line error checking and help. The developers organized Mothra's output in a logical, useful manner for the user. They used tables and histograms whenever possible to organize and display data. The user can look at whatever data he likes through a system of menus. The Mothra project is an ongoing

effort that will eventually incorporate other testing techniques into the interface to create the type of testing environment described earlier.

The environments examined here need to mature even more before they can resemble the integrated testing environment needed to test and verify the large software systems of the future. The literature suggests that Mothra is on the leading edge of automated testing technology coupled with advanced interface design. It deserves further study. Researchers should develop interface designs concurrently with the testing tools. They must consider human factors in their designs. Good interface designs will improve performance, speed, error rates and user satisfaction. The future of automated testing will have to make use of new technology to test the increasingly complex software systems of the future.

F. *PROBLEM STATEMENT*

Graphical user interfaces have become an integral aspect of software design. Though software testing tools tend to be used by a select group of skilled software testers, ease of use is a factor, and a good interface will help to move a testing tool from simply research to a practical level. Our goal is to use the eight principles proposed by Shneiderman, combined with testing tool output representation to design an interface to assist in the use of failure region analysis tools. In doing this, we will identify general principles for the design of testing tool interfaces and indicate issues that remain to be explored.

The work involved in the completion of this thesis was split between the authors. LCDR Abel concentrated on the user interface aspects of the research and coding of VIEWER, while CAPT Monti concentrated on the tool interface aspects of the research and coding of VIEWER.

G. OVERVIEW OF REMAINING CHAPTERS

Chapter II examines the way the tool interface has been developed and how it reflects the principles of "good" interfaces. A brief description of each tool is provided, and its relation to the interface developed as well as the representation of each tools information.

Chapter III provides an extended walkthrough and information necessary for the user to use the interface.

Chapter IV provides a conclusion and a summary of what has been accomplished thus far. It also indicates those areas that require more research. We explore the advantages of using the tools via the VIEWER graphic interface vice command line or other type of interface.

II. TOOL DESCRIPTION

A. TOOL OVERVIEW

The process of analyzing failure regions begins with a compilable Pascal program with one or more known faults. A set of five tools analyzes the failure regions associated with the identified faults in the program. This set consists of REACHER, WALKER, LISTER, FALTER and SPACER. (See Figure 2.1.) REACHER and WALKER derive the conditions under which each program block in the Pascal program, procedure or function may execute. The REACHER program is non-interactive and is the first step in the analysis of failure regions. It accepts compilable Pascal source code as input. REACHER creates an "acfg", an annotated control flow graph, for each module of the Pascal source code. REACHER annotates the graph with the conditions needed to reach each node from the preceding node. The output from REACHER is in a format usable by WALKER, LISTER and FALTER. WALKER interactively leads the user through a traversal of the control flow graph(s) associated with the program. The result of WALKER is an acfg annotated with the conditions under which the tester reaches each node from the start of execution. (Shimeall, REACHER, p. 6)

LISTER outputs an annotated source listing that provides a correspondence between statements in the Pascal source code and the control flow graph nodes. This listing assists the analyst in the use of WALKER and FALTER.

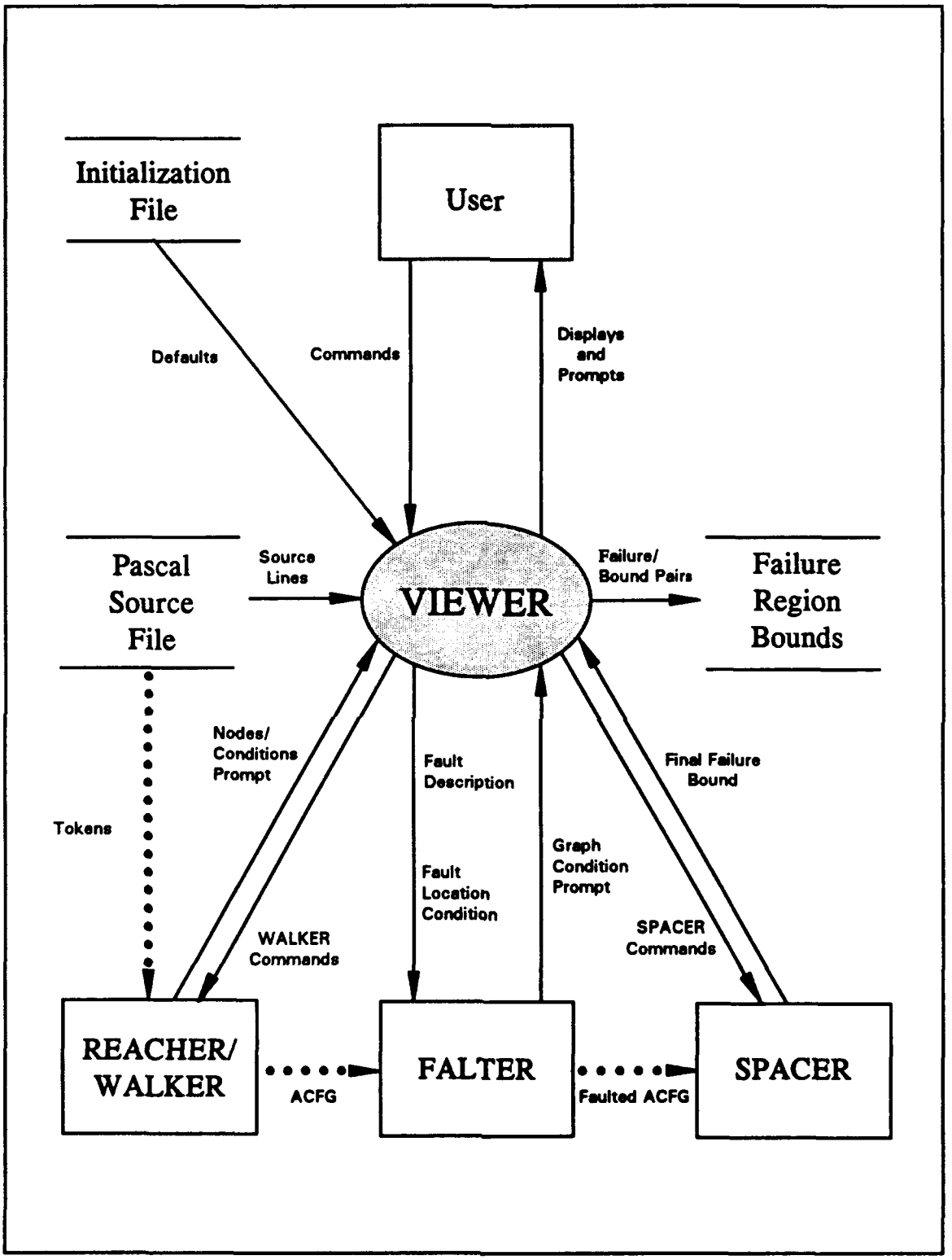

Figure 2.1 VIEWER Context Diagram

FALTER records the conditions under which calculations in the defective code produce an erroneous result. It accepts as input the annotated control flow graph (acfg) from REACHER or WALKER. FALTER also records the effect of a program defect on the local program state. The result is a faulted control flow graph that serves as input to SPACER. (Shimeall, FALTER, p. 2)

SPACER interactively analyzes the faulted control flow graph, calculating the final failure bounds. It analyzes those conditions where later processing does not mask the erroneous value.

B. VIEWER OVERVIEW

Testing tools are a recent development in the area of software engineering. Their design placed little or no emphasis on the interface provided to the tester. A command line interface is most common. A screen-oriented interface can free the tester from certain responsibilities and allow them to focus on the issue at hand, i.e., the testing process.

VIEWER is an interactive, graphical user interface. It facilitates the process of analyzing a program to determine the region of the input space that a known fault maps onto a failure. VIEWER provides a screen-oriented interface to the analysis tools. A screen-oriented interface provides many capabilities. It can reduce the amount of information shown to the user, so it allows the user to concentrate on the important facets of the information. Multiple views of the same information may be provided during program analysis. A screen-oriented interface can better isolate the properties of the

information. It allows the carrying across of information between tools to allow better tracking of the testing process. VIEWER provides automated coordination between the tools, freeing the analyst from dealing with automatable detail. This coordination is important in the maintenance of a certain level of abstraction for the user. VIEWER provides shorthand options for common commands and command sequences. This allows for rapid and customized improvement in the automation of the analysis process as more use of the tools suggests new automation strategies. (Shimeall, VIEWER, p. 2) VIEWER uses several windows to interface to the analysis tools.

C. SUNVIEW OVERVIEW

SunView is the **Sun** Visual/Integrated Environment for Workstations (SunView 1 Programmer's Guide, p. 3). It is a user interface toolkit that provides support for the creation of interactive, graphics based applications run within windows. The features of SunView aiding the design of user interfaces made it an obvious choice for the implementation of VIEWER.

SunView has building blocks for display, and a run-time system for input management. The building blocks are visual representations of objects used to assemble the user interface. (See Figure 2.2, from Beer, p. 49.)

Included in the building blocks are two basic classes of windows: frames and subwindows. The purpose of a frame is to tie subwindows of different types together so they can serve as a unit. There are four types of subwindows provided by SunView: canvases, text subwindows, panels and tty subwindows. A canvas is a subwindow into

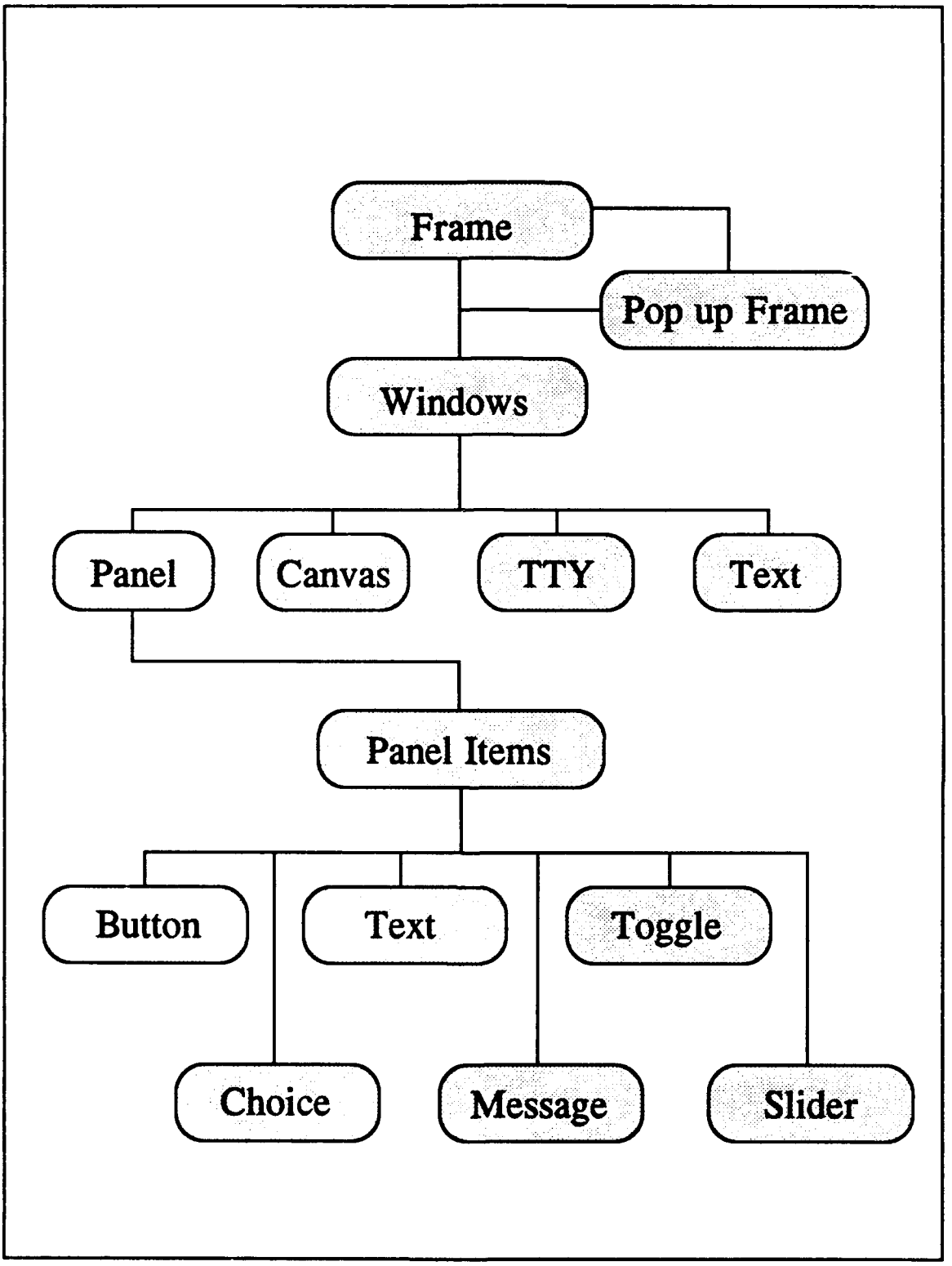

Figure 2.2 SunView Building Blocks

which a program can draw. A text subwindow displays text and has built-in editing capabilities. A panel subwindow contains items such as buttons, text items, menus, choice items, and sliders. A tty subwindow is a terminal emulator in which the user enters commands and executes programs. (SunView 1 Programmer's Guide, p. 3)

SunView applications draw graphics through Pixwin constructs. The most basic use is in a canvas subwindow. (SunView 1 Programmer's Guide, p. 101) SunView provides some built-in pixwin operations (vector, text, polygon, polypoint, line, etc.). Drawing operations should not spill over into other windows and they should not be visible in portions of the window covered by other windows. The pixwin construct and operations provide an interface that meets the two above conditions. (SunView Programmer's Manual, p. 103) Low-level operations on SunView images support and supplement these Pixwin operations.

A SunView image on the screen and in memory is composed of dots called pixels. SunView represents the image internally as a rectangle of pixels. The pixrect structure is the construct used at a low level to access an image and operate on the image. (SunView 1 Programmer's Guide, p. 103) This access and operation are not window-based, so the user must avoid overwriting areas of the screen.

SunView is a notification-based system. A central notifier in each application distributes input to the appropriate window. The notifier is a mechanism that distributes events within a process. (SunView 1 Programmer's Guide, p. 20) SunView provides predefined notify procedures or the programmer may define new ones.

While windows are the most important class of SunView objects, they are not the only objects available. An icon is a small image that represents the application. Command-related features include scrollbars, menus, and panel items.

SunView allows the programmer to attach scrollbars to canvases, panels and text subwindows. The scrollbar allows the user to determine the visible area when the object is larger than the window. A text subwindow has a vertical scrollbar by default, and cannot contain a horizontal scrollbar. Panels and canvases may have both horizontal and vertical scrollbars. It is the programmer's responsibility to create the scrollbars for these subwindows. (SunView 1 Programmer's Guide, p. 267)

A menu allows the user to make choices and issue commands. They differ from windows in that menus are visible while pressing the RIGHT mouse button. Also, menus are less flexible, for they allow the user to choose only from a list of options.

A panel item is a panel component that simplifies a particular user-application interaction. They include: message items, buttons, and text items. A message item is only visible as its label. They display text to the user. A button item allows the user to initiate commands. A default image provided by SunView represents the button, or the programmer may define an image for the button.

A text item is a type-in field. The notify procedure indicated may accept input by character, on specified characters, or by the entire field. (SunView 1 Programmer's Guide, p. 159) The value of the text item is the string entered by the user. This string appears on the screen after the label. (SunView 1 Programmer's Guide, p. 178) Panel items all have a label component. The label is a string or graphical image. The button,

choice, toggle, and text items have a menu component that allows the user to select the item directly or select from the item's menu.

Complex and flexible objects are the basis of the SunView model. Since SunView can only handle very simple topologies by default, there are attributes to allow the user to specify more complex layouts. The basic idea is to use a small set of functions with a large set of arguments (attributes) to manipulate the objects. SunView functions use variable length attribute lists, so a given call to create or modify an object mentions only the relevant attributes. Most functions in SunView take pairs of attributes and values. The number of pairs varies depending on how many attributes the programmer wishes to set. (Beer, p. 50)

SunView provides a high level of abstraction in the design of interfaces. It also provides the complexity needed to design an interface of the type desired. The subwindows available in SunView provide precisely the functions needed to represent/display the results and output from the analysis tools.

1. VIEWER Internal Structure

VIEWER includes interfaces for each of the interaction tools. The interfaces interact with one another and with the testing tools. There also is interaction between the subwindows of each interface. (See Figure 2.3.) The VIEWWIN interface contains a tty subwindow and a panel subwindow. The tty subwindow receives commands from the panel subwindow, or accepts command line input. The panel subwindow contains a text input area and buttons for initiating the interfaces for WALKER (WALKWIN), FALTER (FALTWIN) and SPACER (SPACEWIN).

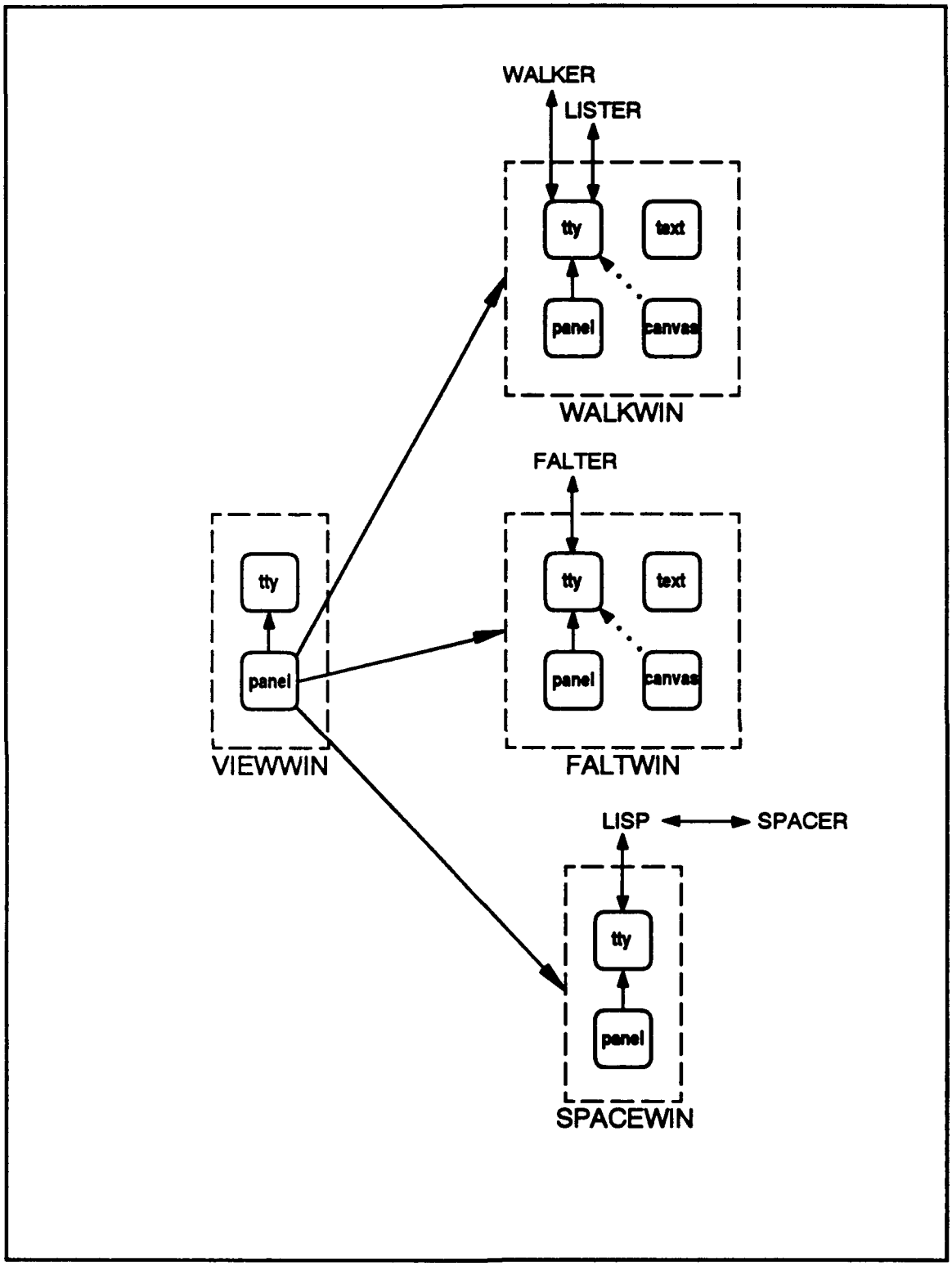

Figure 2.3 VIEWER Subblocks

The WALKER interface is WALKWIN. It allows for interactive use of WALKER and the ability to call the non-interactive REACHER and LISTER. WALKWIN is a frame with four subwindows in it. The text subwindow allows the user to view Pascal source code or other text files. The tty subwindow is the link to the operating system, and displays the output of WALKER. The panel subwindow contains buttons, text input area and a menu, and all interact with WALKER.

The canvas subwindow displays the control flow graph. The control flow graph is an integral part of the analysis of the failure region(s). A visual representation of this graph aids the user in the test process. It presents information so the user can track the conditions and location through the traversal of the graph. Though not present in the prototype interface, a facility for interaction directly with WALKER via the control flow graph display is under construction.

The text subwindow displays text files. While the user is testing a particular Pascal program, he displays the program in the text window. He also can load other files into the text subwindow. For example, during the running of WALKER, the user may want to display a textual representation of the control flow graph. He may want to review the output from LISTER or other information about the faults in the code. The scrollbar allows the user to move quickly to the top or bottom of the file, move line by line, or move to an area of the file.

The tty subwindow provides an area to record WALKER commands and displays. It allows the expert user to dispense with buttons and use keyboard entry. The tty subwindow also accepts UNIX commands, i.e., listing the files in the current directory.

The panel subwindow contains the panel items used for interaction between the user and WALKER. The buttons are single action items, some of which use the string input provided by the message item. The menu contains other functions, in the order in which the user probably will need them - run, help, save and quit. Buttons do not function unless the user is running WALKER. The user may find that additional common command sequences should be represented as buttons. The programmer can create new buttons and add them to the panel subwindow.

The FALTER interface is FALTWIN. It is similar in function to WALKWIN, and therefore similar in representation and composition. It also has a base frame with four subwindows - canvas, text, tty and panel. The menu selections are the same as for WALKWIN. But the use of the other component parts differs from WALKWIN.

During the interaction with FALTER, the user may need to view different text files than they used with WALKER. They may be interested in displaying fault information or results of debug sessions to characterize the effect of the fault. FALTWIN also provides different button choices from WALKWIN. These buttons relate to the interaction necessary to use FALTER. See Chapter III for an example of this use.

In the prototype of VIEWER, the interface to SPACER is minimal. A tty subwindow provides an area for the user to invoke LISP, and then SPACER. A panel subwindow contains buttons for use with SPACER.

The interface to SPACER is limited by the current level of knowledge concerning the use of SPACER. This portion of failure region analysis is the newest area of research. It is not yet clear what is the best way to represent the output. A strict

geometrical approach may not be suitable or effective. Two dimensional representations of a multi-dimensional region are inadequate or incomplete. Currently, textual output is a form of output that can be understood and evaluated by the tester. As more usage of the tool occurs, and testers develop a better idea of what the key properties are, a way to represent these properties also will develop.

The other tools (WALKER and FALTER) are similar to other testing tools (particularly static analysis tools that look for unreachable code). The understanding of the key properties in their output/results are clear, and therefore can be represented. While SPACER is still in its early stages, a less rigid prototype will allow the interface to develop with the tool.

D. WALKTHROUGH OF FUNCTIONS OF MAIN PARTS

This section contains specific explanations for the algorithms used in the development of the interfaces. A copy of the code is in Appendix E.

1. VIEWWIN

a. Declarations

The VIEWWIN code begins by including the SunView header files necessary for using the desired SunView functions and procedures. Following these header files are macros, variable declarations, and procedure declarations. To use an icon when closing a frame, the programmer must load the icon image into an array.

Using the *mpr_static()*[1] macro, the programmer puts this array into a pixrect structure for use with the *icon_create()* function within the main block of the program.

b. Frame and Subwindows

VIEWWIN contains a tty subwindow and a panel subwindow. (SunView 1 Programmer's Guide, p. 213) Attributes set the heights and widths. Each subwindow has attributes identifying that they belong to the base frame, and what type of subwindow they are. The WIN_BELOW attribute places the panel subwindow below the tty subwindow.

The function *window_fit(base_frame)* adjusts the VIEWER base frames to contain the various subwindows. This is useful when the user want to scale the window size in SunView. The function *window_create()* does not display the windows on the screen. The function *window_main_loop(base_frame)* displays all created (and visible) objects on the screen and puts the tool in a loop awaiting user input. The VIEWER interface objects must appear before the call to *window_main_loop()*. (Beer, p. 51)

All the VIEWER interfaces use the *window_create()* function for all types of windows. This function returns a handle to the object created, and the handle declaration appears at the top of the program. Changing the attributes passed to the *window_create()* function controls the appearance of the VIEWER interface windows. These attributes include: the parent item, a label, an icon and other information.

---

[1] In the text of this thesis, italicized text refers to SunView functions and procedures.

The base_frame does not have a parent item, so this attribute is NULL. For each base frame, a defined position ensures the presentation of the frames is in a useful and consistent manner. Defined positions means the starting location of the (0,0) coordinate for the base frame. The user could change these starting coordinates within the program, or move the frames via SunView mouse/menu commands. The label identifies the displayed interface for the user. The icon attribute provides the user the ability to close the frame and have a relevant picture appear. All the VIEWER interface base frames have icons for closing the frame. (See Figure 2.4.) Icon designs jog the memory of the user when they suspend an operation.

By default, a tty subwindow forks a shell. It is necessary to send button, menu, and text input to the tty subwindow. The function *ttysw_input()* sends input to the tty subwindow. It appends the character sequence in a buffer onto the tty's input queue. It returns the number of characters accepted. The program treats the characters as if they were a keyboard entry. This function provides a simple way for the user to send input to the program running in the tty subwindow. (SunView Programmer's Manual, p. 213)

The panel subwindow contains a filename input area (a text item) and a button area. The text item provides a way for us to input text to the panel subwindow that the tty subwindow uses for processing. VIEWWIN uses the *panel_create_item()* function with attributes for the subwindow in which to place the item, the prompt to display, and the length of input displayed. The subwindow displays 50 characters due to window size constraints. The length of the input field defaults to 80 characters.

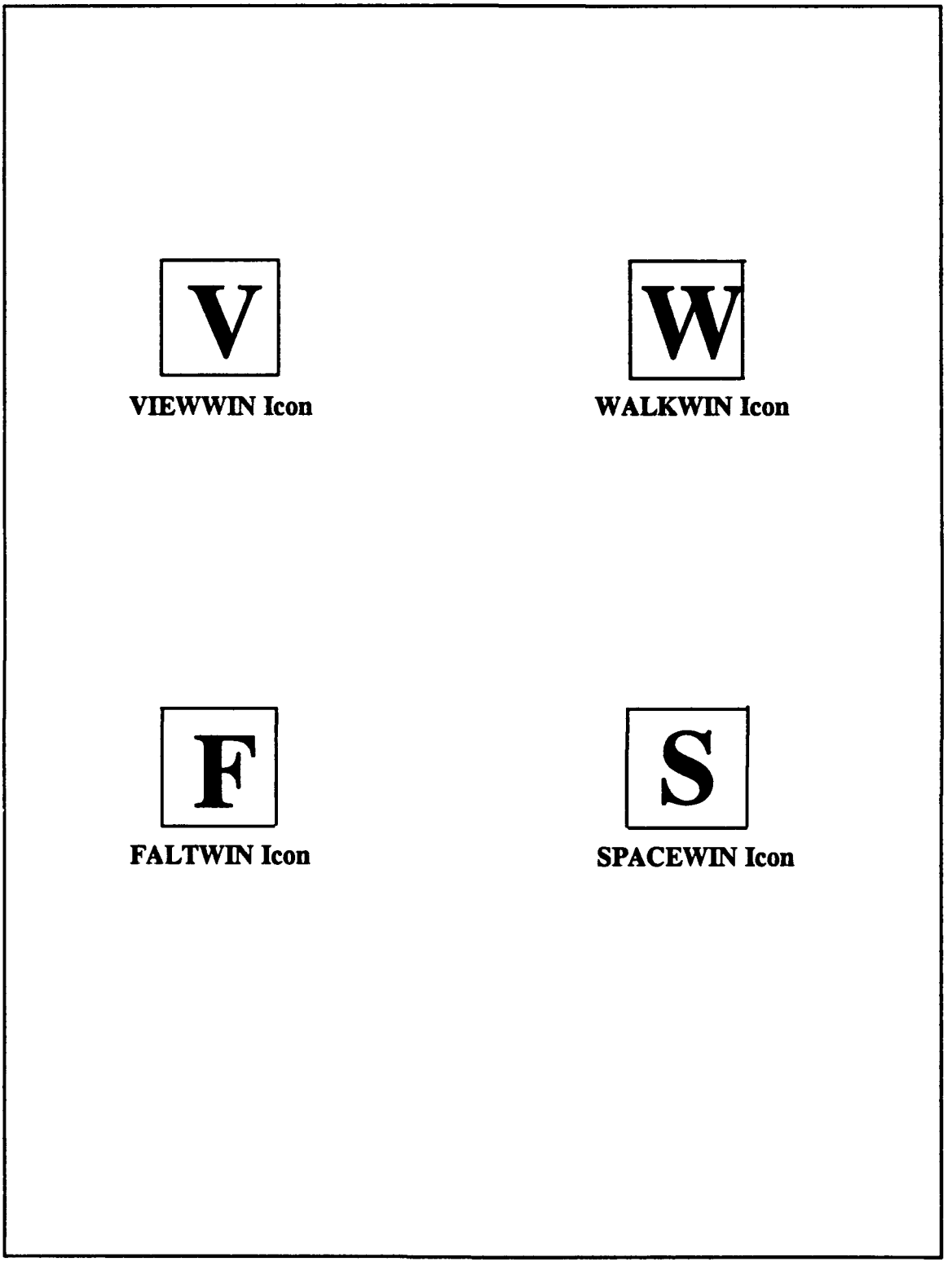

Figure 2.4 Icons

The buttons in VIEWWIN have similar definitions. Each button definition uses the SunView function *panel_create_item()*. Attributes for the buttons include information for which subwindow is the parent, that it is a button, the process to notify when the user selects the item, and the image to present within the panel. The SunView function *panel_button_image()* defines how the button looks in the panel. The image has attributes for the parent window, the name to put in the button, the width of the button, and the font style to use. The *panel_button_image()* function allowed us to keep the look and feel of the VIEWER interface buttons consistent across the interfaces. Instead of a name consisting of a string, the image may be an icon. (Our buttons that activate menus in WALKWIN and FALTWIN use this feature.)

c. Procedures

Notify procedures that serve as values for PANEL_NOTIFY_PROC attributes are included in the code. The procedure "call_reacher" uses the filename entered in the panel subwindow and sends a command to run REACHER on that file. The procedure stores the filename and the command in a character buffer, and sends it to the tty subwindow of VIEWWIN via the *ttysw_input()* function.

The "call_reacher" procedure displays WALKWIN with the text of the file shown in the text subwindow. To accomplish this, declarations include a character buffer of length 81, an event of type Event (SunView), and a result of type integer. In order to provide some error handling, the procedure looks for a "p" as the last character of the filename as an attempt to ensure proper filename format, i.e., a Pascal file. The procedure loads the filename into a buffer and the checks the last character. If the last

character is a "p", the procedure loads a call to REACHER concatenated with the filename into the buffer and sends the result to the tty subwindow. The algorithm sends a command to the tty subwindow to open WALKWIN with the filename as a parameter.

If the last character is not a "p", the "msg" procedure accomplishes the error handling. This procedure displays a pop-up frame to alert the user of an improper filename format. (See Figure 2.5.) It has two parameters: a string (the message to display) and an integer (indicating whether to output beeps).

Within the "msg" procedure is used an integer "result", and event, and a character string composed of the message "Press/"Continue/" to proceed." Result equals the return integer from the *alert_prompt()* procedure. A SunView alert is a frame that contains one subwindow, a panel. The *alert_prompt()* function may set attributes for the display of the text message(s), the position of the frame, beeping characteristics, and buttons. (SunView 1 Programmer's Guide, p. 202) It takes as parameters the frame under consideration, the address of the event (user action) and the strings used (the message sent in and the one hard coded within the function). It pops up on the screen to notify the user of a problem or other things requiring attention. The alert has full screen access - the screen freezes until the user gives a response. (SunView 1 Programmer's Guide, p. 199)

Alerts have better user interface facilities than *menu_prompt()*, which offers a simple box with no more than two choices. Alerts have an arrow to get the user's attention, buttons, fonts, beeps and a 3-D shadow, and there can be more than two choices. (SunView 1 Programmer's Guide, p. 199)

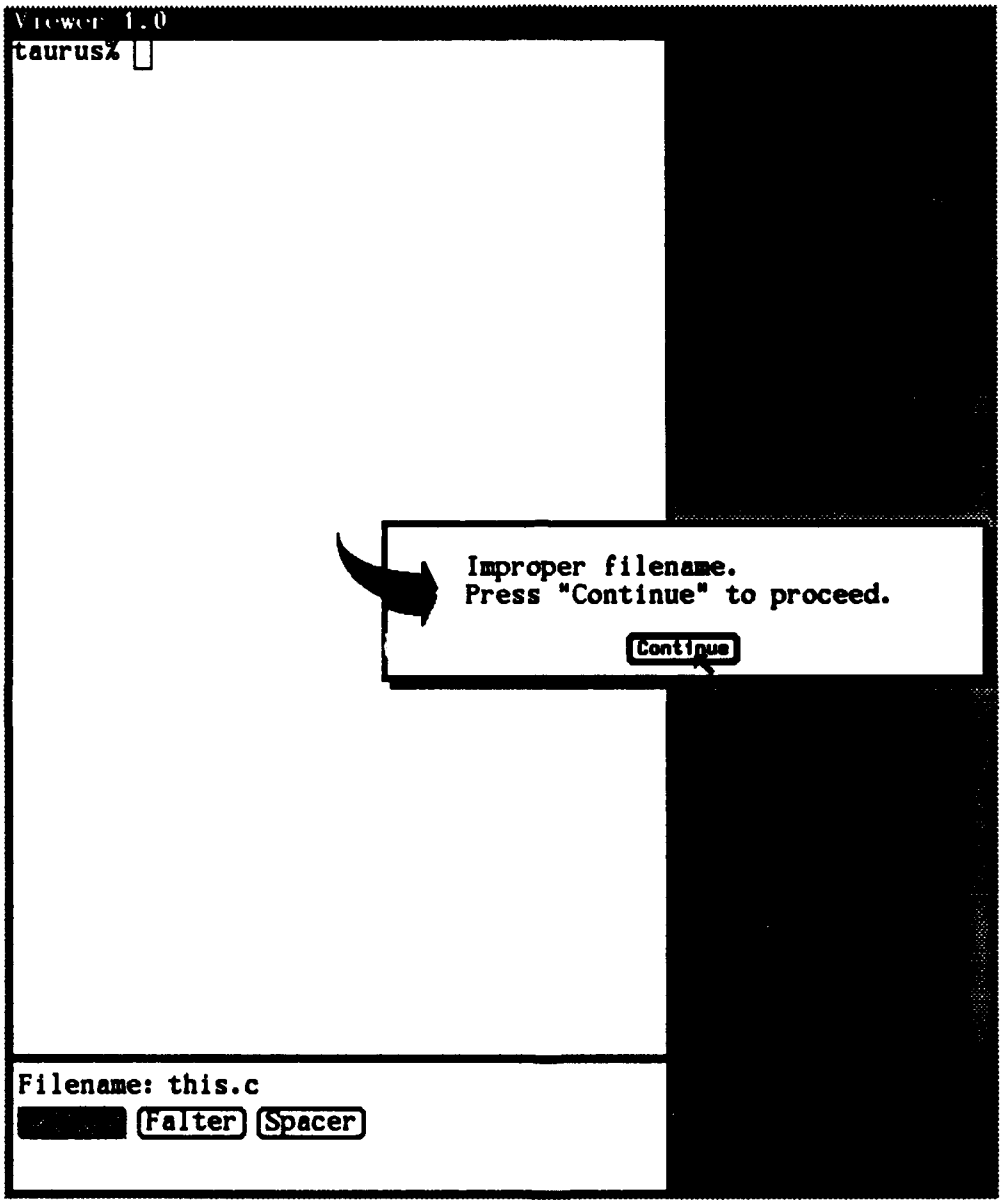

Figure 2.5 Alert Frame

The *alert_prompt()* function creates the alert, pops it on the screen, handles the user interaction, removes the alert, and returns a value. The alert used by VIEWWIN has only one button - "continue" - since it only serves to notify the user of an error in the format of the filename. (SunView 1 Programmer's Guide, p. 202) This is the only choice needed by the user, for this is a simple alert that the filename is in an improper format. Pressing the continue button clears the pop up frame from the screen. It also clears the filename item by a call to *panel_set_value()* with parameters for the panel item to set and what to set it to (an empty string).

The number of beeps sounded when the alert appears is the number set by the user in their defaultsedit file. The programmer can set the alert to no beeps despite the default. Programmers usually reserve beeping for unexpected events. A response to a request would normally not beep when displayed. (SunView 1 Programmer's Guide, p. 208)

The "call_falter" procedure uses the same algorithm, except that VIEWWIN does not call FALTER directly from the button invocation. Since FALTER is interactive, the procedure displays the interface, passing the filename as a parameter, and the user runs FALTER from FALTWIN. In a progression through the tools, the filename would be the same one as used in WALKWIN, so the procedure performs no error handling here.

2. WALKWIN algorithms

a. Declarations

WALKWIN uses several header files not found in the VIEWWIN source code. This interface has more subwindows, and involves graphics. WALKWIN includes two icon declarations. One represents the frame and the other is an image for a button with an associated menu. This allowed us to differentiate between a command button, which uses the default SunView image, and a menu button, which uses the imported image. The user will find this less confusing.

b. Frame and Subwindows

The VIEWWIN frame passes the filename to WALKWIN, which assigns it to a variable. WALKWIN uses that variable to load the file into the text subwindow using the SunView *window_set()* function. The WIN_X and WIN_Y attributes align the WALKWIN base frame with the VIEWWIN frame.

The layout of WALKWIN's base frame and subwindows considers several ergonomic factors. The size of each subwindow is large enough to be useful, yet small enough to create an organized interface. The text subwindow displays the text of the Pascal code being tested next to the representation of the acfg in the canvas subwindow. WALKWIN sets its subwindows' height and width attributes in pixels. This allows very precise arrangement and alignment within the frame. These dimensions are adjusted to allow for varying resolution in display screens.

WALKWIN's canvas subwindow has attributes for height and width of the window and height and width of the canvas. The canvas is larger than the actual subwindow, to accommodate large, complex graphical structures. Since the canvas is larger than the window, WALKWIN includes scrollbars for moving around the canvas. For consistency between machines, WALKWIN defines the location of each scrollbar explicitly, since different users may have different values in defaultsedit. WALKWIN uses the attributes CANVAS_AUTO_EXPAND and CANVAS_AUTO_SHRINK to adjust automatically the canvas size when scaling the window. (SunView 1 Programmer's Guide, p. 69)

The attributes of the text subwindow and the tty subwindow include placement of them in relation to the canvas subwindow, using WIN_RIGHT_OF and WIN_BELOW. WALKWIN's text subwindow has an attribute to set the word wrap at the word vice character level. This makes text displayed in the text subwindow much easier to read.

Each command recognized by WALKER has a corresponding button within the panel subwindow. The panel also has a menu for interaction with WALKER. The menu becomes visible when the user presses the right mouse button with the pointer on the menu button. We used the PANEL_CHOICE attribute to attach the menu to this button. The image is the address of the pixrect of the icon image, as discussed earlier.

The panel subwindow has three text items for entry of strings to use with WALKER commands and with the call to LISTER. The WALKER input area provides string input for all the WALKER commands. LISTER format requires an input filename

and a name for the output file it creates. We explicitly positioned the text items within the panel subwindow. The WALKER input displays up to 50 of the 80 allowable input characters, to allow for the input of the conditions or annotations. Since LISTER inputs consist of only filenames, these text items display 15 of the allowable 80 character input.

c. Algorithms

(1) Circles and lines

We conceived a graphical representation of the acfg as circles to represent the nodes and lines to represent the connections between the nodes. The SunView *pw_vector()* provides a simple mechanism for drawing lines in the canvas subwindow. Drawing a circle in SunView is not as simple. There is no predefined pixwin or pixrect function for a circle.

SunView allows the programmer to import an icon image into a pixwin object and display it within the canvas subwindow. The problem was that the shape of the image is a square. There was some difficulty involved in removing the corners so the circle remained.

A polygon, with the number of sides set high enough, approximates a circle. To draw polygons, WALKWIN uses the *pw_polygon_2()* function. Initially, we used the operation PIX_SRC to fill the polygon reverse of the background. Briefly, PIX_SRC sets all pixels to true, so the polygon appeared in reverse video on the white canvas background.

Aesthetically, we felt the circles representing nodes should be unfilled - that is the center should be the same as the background. We tried the operation PIX_CLR, which displays the polygon in the same color as the background. This made our polygons invisible, since the polygons lacked visible outlines. The solution was to display a polygon with the operation of PIX_CLR, then place vectors around the edge to provide the outline of the circle. The result was exactly as desired. By choosing a very large number for the sides, we achieved a smooth "circle". The large number of sides does not impact adversely on the speed of the display in this implementation.

(2) Graphical Representation of ACFG

After parsing the target Pascal source file, REACHER produces a control flow graph of the source code in textual form and saves this graph in a file called "reacher_out". WALKER reads the control flow graph in to memory so the user can step through the graph. VIEWER takes the control flow graph provided by REACHER and produces a graphical representation of reacher_out in the canvas subwindow of WALKWIN. To accomplish this we used the same macros, definitions and structures used by WALKER to dynamically allocate memory and load "reacher_out" into memory. With the help of our previous experience in constructing SunView lines and polygons, we designed an algorithm to read the control flow graph and produce the graphical representation of it. The next section briefly describes this algorithm.

After WALKWIN reads the control flow graph into memory, it sets a pointer called curnode to the root node in the control flow graph. We use a do-while loop to read the binary tree and draw the graph. The do-while loop processes the graph

as long as the following conditions exist: the current node's left child is not equal to null, or the current node's right child is not equal to null, or the top of the stack is greater than zero. When these conditions exist then processing inside the loop uses three conditionals to draw the graph in the walkwin canvas. The algorithm first checks to see if the current node has a right child. If it does, walkwin pushes the node onto an acfg stack, and increments the stack pointer. We assume that since the node has a right child it also must have a left child. The algorithm computes the (x,y) coordinates for the next left node and draws a line from the current (x,y) location to the left node (x,y) location using the SunView *pw_vector()* function. We then draw a circle at the current (x,y) location using techniques discussed earlier. We save the current (x,y) coordinates in a position array and move our current (x,y) position to the location of the left child. The program marks the current node and the curnode pointer moves to the left child.

If the right node is null the algorithm checks to see if the left node is null. If it is, we decrement the acfg stack pointer and the node on the top of the stack becomes the current node. The algorithm computes the (x,y) coordinates of the right node and stores the position of the current node in the position array. We call *pw_vector()* to draw a line from the current (x,y) location to the right node (x,y) location. We draw a circle at the current (x,y) location, then move the current (x,y) location to the position of the right node. The current node becomes the right child.

If the current node has no right child and the left node is not null then the current node's left child becomes the current node. The algorithm computes the (x,y) coordinates for the next left node. We store the previous curnode (x,y) location

in the position array and use *pw_vector()* to draw a line from the current (x,y) location to the new curnode (x,y) location. We draw a circle at the current (x,y) coordinates and move the current (x,y) coordinates to the position of the curnode.

During this looping process if the algorithm moves to a marked curnode, then we call a while loop to check the stack. If the top of the stack is greater that zero then this is the signal to pop the stack. We retrieve the position of the marked curnode from the position array and this becomes our current (x,y) coordinate. The algorithm computes the next right node location and uses *pw_vector()* to draw a line from the curnode (x,y) coordinates to the coordinates of the right child. We move the current (x,y) location to the position of the right child and the curnode becomes the right child. Then we pop the stack. This process continues in a while loop until we reach an unmarked node or TOS is equal to zero.

(3) Buttons and Menu

Each button has a corresponding notify procedure. They use a buffer to load an input string. The string includes the cryptic command for WALKER and the additional string input if needed by that particular command. The annotate, change condition, join and goto commands use the additional string input from the WALKER text item.

The "Walker Menu" button has its own procedure to handle input from the menu. It uses the event received from the mouse action. SunView assigns a value to each choice in the menu, beginning with zero for the first item. A switch statement delineates the action to take based on the value detected. The variable

walker_run indicates whether the user is running WALKER. It is initialized to zero in the declarations section of the program. When the user selects "Run" from the menu, the program loads the command to run WALKER on reacher_out in the string buffer. If walker_run equals zero, the program sends the string to the tty subwindow and sets walker_run to one. The *panel_set_value()* function resets the walker input area to an empty string.

The other menu selections check for walker_run equal to one before sending the string to the tty subwindow. Each contains the *panel_set_value()* function to reset the input area. In addition, the "Quit" selection clears the tty subwindow.

The "call_lister" procedure is the notifier for the LISTER button initiation. It composes a string to send to the tty subwindow including the strings for input and output filenames and stores it in a buffer. The *tty_input()* function sends the buffer contents to the tty subwindow, and *panel_set_value()* resets both the input string input areas.

E. CONCLUSION

This chapter provides a detailed look at the development and design of the interface. Every decision in the design considered the general principles outlined in Chapter I. To review those principles, they are:

- Consistency
- Frequent Users have shortcuts
- Informative Feedback

- Dialogue yield closure
- Simple error handling
- Easy reversal of actions
- Internal Locus of control
- Reduce short term memory load

Consistency is perhaps the most crucial of these principles. The size and appearance of WALKWIN's buttons reflect this consistency. The user will find that all functionally similar buttons look the same. WALKWIN displays buttons that represent a menu by an icon so the difference is readily apparent to the user. Though each tool has a different way, or ways, to exit the program, the interfaces to the tools only use "Quit" in the menu.

The tty subwindow in each interface provides the mechanism for expert users to use shortcuts. They have command line access in any level.

User feedback is present in many forms. The buttons reverse video when pressed, and are grey until action completes. This also provides closure dialogue. Of course, this usually occurs so quickly the user sees a flash of reverse color. Titles on each frame are a form of feedback. It is a way to verify that the proper interface has displayed. The bubbles in the scrollbars move at the command of the user. They show position of the user in the canvas or in the text subwindow.

The principle of user feedback also appears in the representation of the information. The user can see analysis results in several forms within the same interface. The text

subwindow presents the textual form of the Pascal source code, while the canvas subwindow presents the graphical representation of the acfg corresponding to the Pascal code. They can see the information change as the analysis proceeds.

The interface provides closure when the user selects quit from the menu of WALKER or FALTER. The program not only ends the execution of the testing tool, but also sends a command to the tty subwindow to clear, signally a completion of the action.

There is little error handling in VIEWER. One example is the alert pop up frame when entering the filename to use with REACHER and FALTER. When the filename does not end in "p", the alert appears. This is also a form of user feedback.

This prototype provides reversal of action in the use of icons. Clicking the left mouse button on *an icon* restores the operation.

The interface provides an internal locus of control. The user is in *control* of the interaction with WALKER and FALTER.

To ease the load on short term memory, the buttons' labels use a descriptive name vice the single letter command recognized by the program to which it interfaces. Since all buttons are available for the user to see, they do not have to remember the cryptic commands.

What does a testing tool interface need to represent? The analysis of failure regions has a control flow graph, so an ability to present the graph is important. These tools process and analyze Pascal source code. The user would want a way to display and review the text and examine the acfg graph while they are running the tool. The tty

subwindow supports the need to interact with a running program. The buttons and input areas in the panel are for easy access to commands and other reasons delineated above.

VIEWER allows the user to display multiple views of the information involved in the program analysis. They can isolate the pertinent properties and concentrate on the process of testing. The interfaces carry across information between the tools. The tester tracks the processes better since the interface reduces the amount of information they need to assimilate.

III. EXAMPLE VIEWER SESSION

A. ASSUMPTIONS

The discussion in this chapter uses an example Pascal program, called "getangle.p", which has a known fault. In this program, there is an observer, located at (XO, YO), and an observed object, located at (XT, YT), with width W and length L. The purpose is to compute the angle occupied by the observed object when seen from the observer's location. Appendix E contains a copy of the code, with comments showing the acfg node numbers.

We assume the user has some knowledge of Sun Workstations and SunView. Also, the user should understand failure region analysis and the purpose of each tool involved in the process.

B. EXAMPLE

1. Initiate SunView

The user begins by initiating SunView. He must be in the directory containing the Pascal code to process. The user's command path must contain directories for the tools.

2. VIEWER

a. Initiating VIEWWIN

To start VIEWER, the user enters "VIEWWIN" at the prompt within a shell window of SunView. VIEWWIN appears on the screen. The upper subwindow contains a UNIX command shell prompt. The lower subwindow contains a text entry area with the label "Filename:" and three buttons, one each for WALKER, FALTER, and SPACER. The maximum number of characters the user can enter in the text item is 80. The control panel only displays 50 characters, but the string remains valid up to 80 characters. The user will lose any character entered beyond the limit of 80. (SunView 1 Programmer's Guide, p. 179)

The input file should be compilable Pascal source code. To enter the filename for processing, the user moves the cursor so it is within the lower subwindow. The caret at the text entry item will be a dark triangle, indicating the location for text input. The interface will not allow WALKER to run, nor display WALKWIN, unless the filename provided ends in "p". If the filename does not have the proper format, VIEWER displays a pop-up frame indicating that the filename is incorrect, with a continue button. Depending on the default setting, a beep, or a series of beeps, alerts the user of an error. The alert automatically places the pointer on the "continue" button. When the user clicks on "continue" using the left mouse button, the erroneous filename disappears, and the text item is ready for a new entry. The pop-up frame is a blocking frame and recognizes no entries until the user presses the "continue" button.

b. Selecting Analysis Tools

The user enters "getangle.p" at the input text item, moves the pointer to the WALKER button, and depresses the left mouse button. (See Figure 3.1.) When pressing a button item, the rectangle inverts. Upon releasing the left mouse button, SunView paints the rectangle with a grey background, which provides feedback that the user selected the item and executed the command. The grey background clears when the program returns from the notify procedure. (SunView 1 Programmer's Guide, p. 167)

REACHER processes the Pascal file, and output appears in the VIEWWIN tty subwindow. If REACHER is successful, the message "Successful parse!" appears in the tty subwindow.

3. WALKWIN

a. Initiating WALKWIN

WALKWIN positions its base frame on the screen, relative to VIEWWIN. (See Figure 3.2.) The label at the top of the window identifies the WALKWIN interface. WALKWIN displays the code for "getangle.p" in the text subwindow. The canvas subwindow displays the graphical representation of the annotated control flow graph for the main module of "getangle.p". WALKWIN provides scrollbars for the text and canvas subwindows. The canvas subwindow has two scrollbars - horizontal and vertical - while the text subwindow has only a vertical scrollbar. Note that the scrollbars for the canvas do not have buttons on either end, while the text subwindow scrollbar has buttons. To scroll, the user moves the cursor

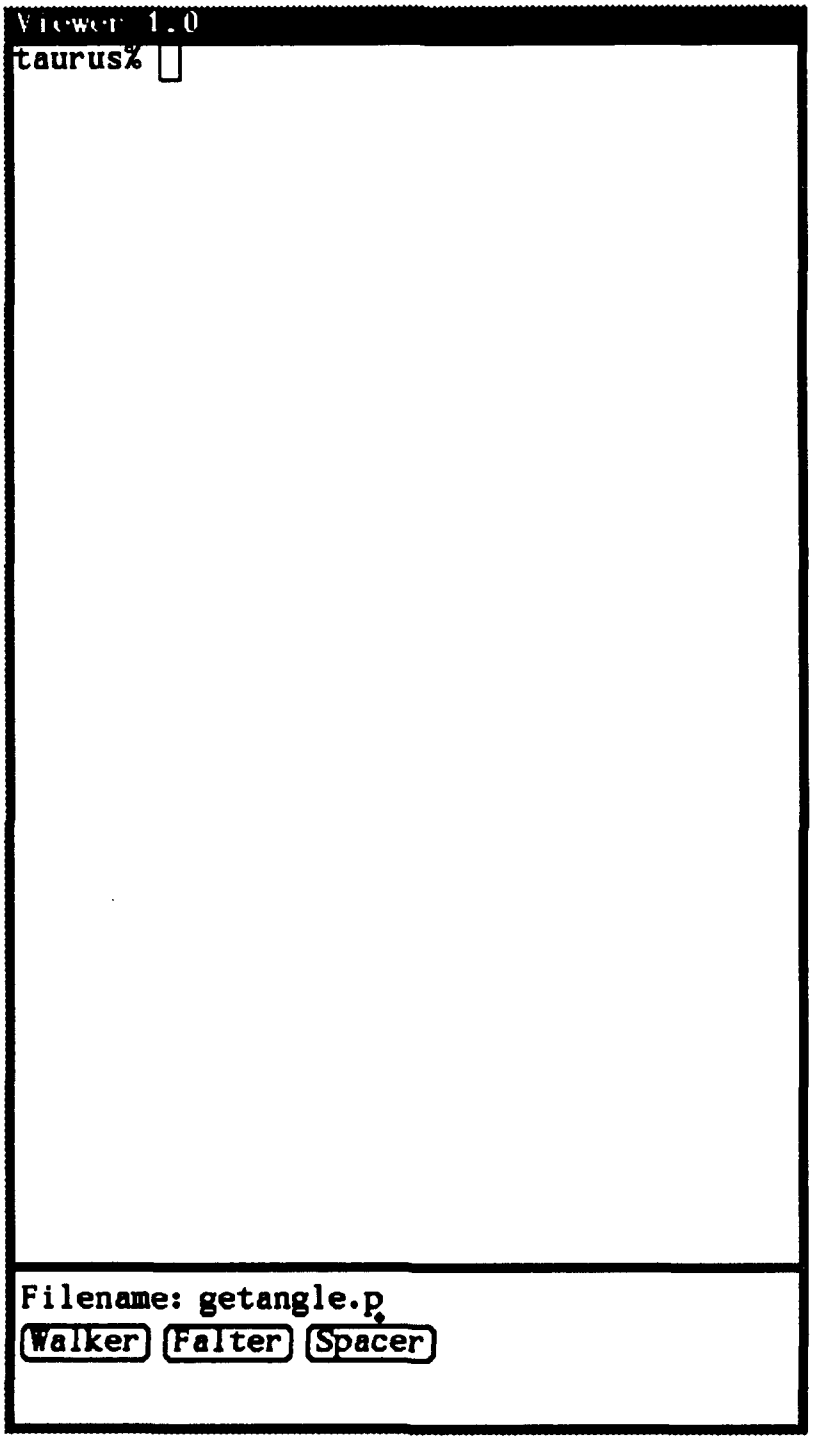

Figure 3.1 VIEWWIN Screen

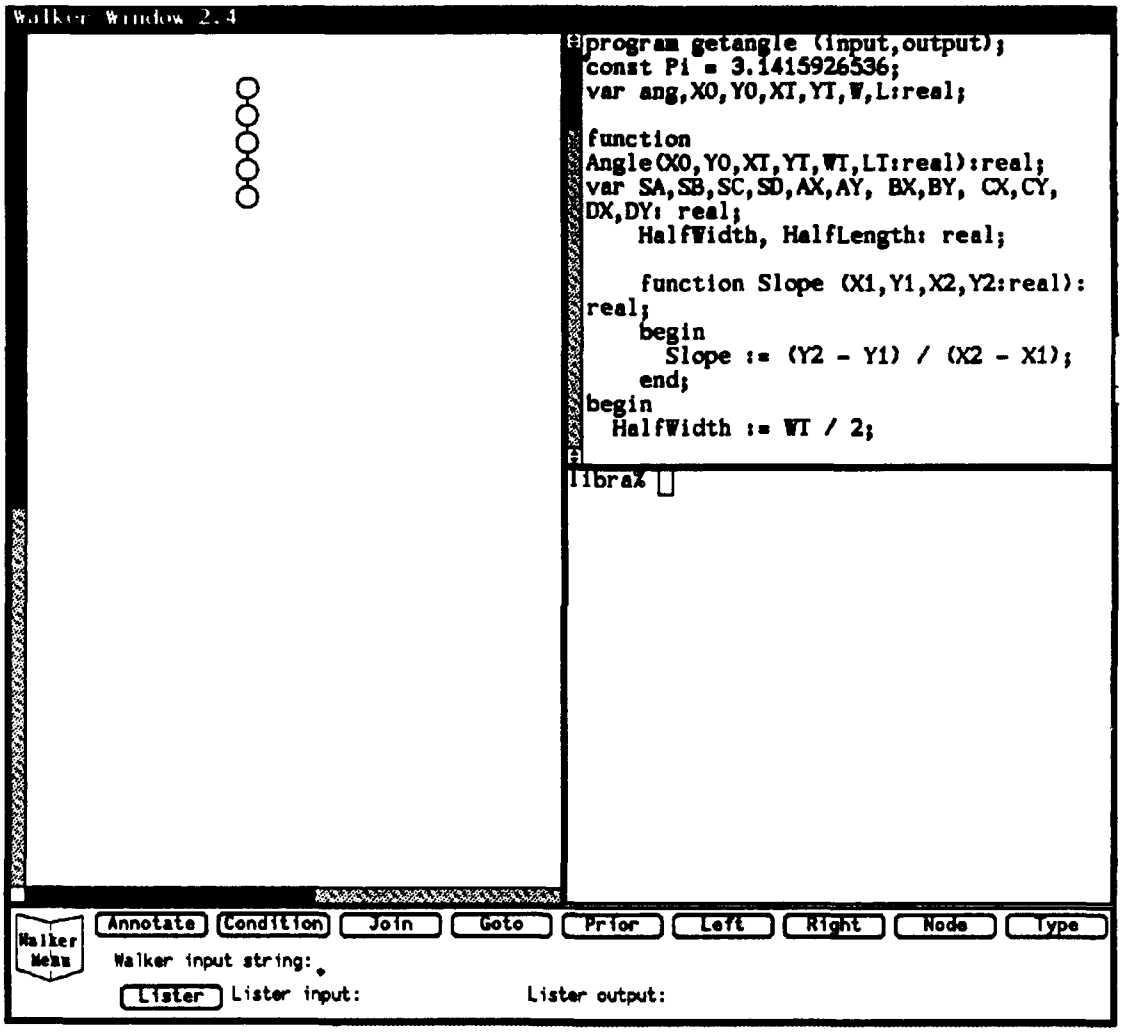

Figure 3.2 WALKWIN Frame

into the scrollbar (the bar itself or the buttons) and clicks a mouse button. (See Figure 3.3 for the possible combinations.) (SunView 1 Programmer's Guide, p.271)

Mouse	Pointer Location	Action
Left	Button	Line forward
Right	Button	Line backward
Middle	Button	Page forward
Middle (shifted)	Button	Page backward
Left	Bar	Line opposite cursor to top
Right	Bar	Top line comes to cursor
Left (shifted)	Bar	Bottom line comes to cursor
Right (shifted)	Bar	Line opposite cursor to bottom
Middle	Bar	Line whose offset into the scrolling object approximates that of the cursor into the scrollbar is positioned at top (thumbing)

Figure 3.3 Text Scrollbar Mouse Commands

When the user presses a mouse button inside the scrollbar, the cursor changes to preview the action for that button. Releasing the button causes the action to

execute. Holding the mouse button down repeats the action. (SunView 1 Programmer's Guide, p. 271) The scrollbars allow the user to see parts of the window not displayed.

SunView provides standard text menus for the code displayed in the text subwindow. To initiate these menus, the user moves the cursor to the text subwindow, and depresses the right mouse button. SunView provides these menus for the convenience of the user. They are not a necessary part of the failure region analysis tools. The user can find a detailed explanation of these menus in the SunView manual. The tty subwindow also has a built-in menu. The canvas and panel subwindows do not have these standard menus.

b. Starting WALKER

The panel subwindow runs across the bottom of the WALKWIN frame. It contains all those items necessary for the interaction with WALKER, and the running of LISTER. The first step in using WALKER is to run the program from the Walker menu. The user moves the pointer to the WALKER menu icon, and presses and holds the right mouse button. A menu appears with four choices - "Run", "Help", "Save" and "Quit". (See Figure 3.4.) Only the "Run" option will function at this point. All other menu choices and the buttons on the top row do not function unless the user is in the interactive mode with WALKER. To initiate WALKER, the user selects "Run" by moving the mouse until it highlights the "Run" command and releasing the mouse button. This action calls WALKER on the default file name produced by REACHER ("reacher_out"). A copy of "reacher_out" appears in Appendix F. The user is now in the interactive mode with WALKER and output appears in the tty subwindow. For a list

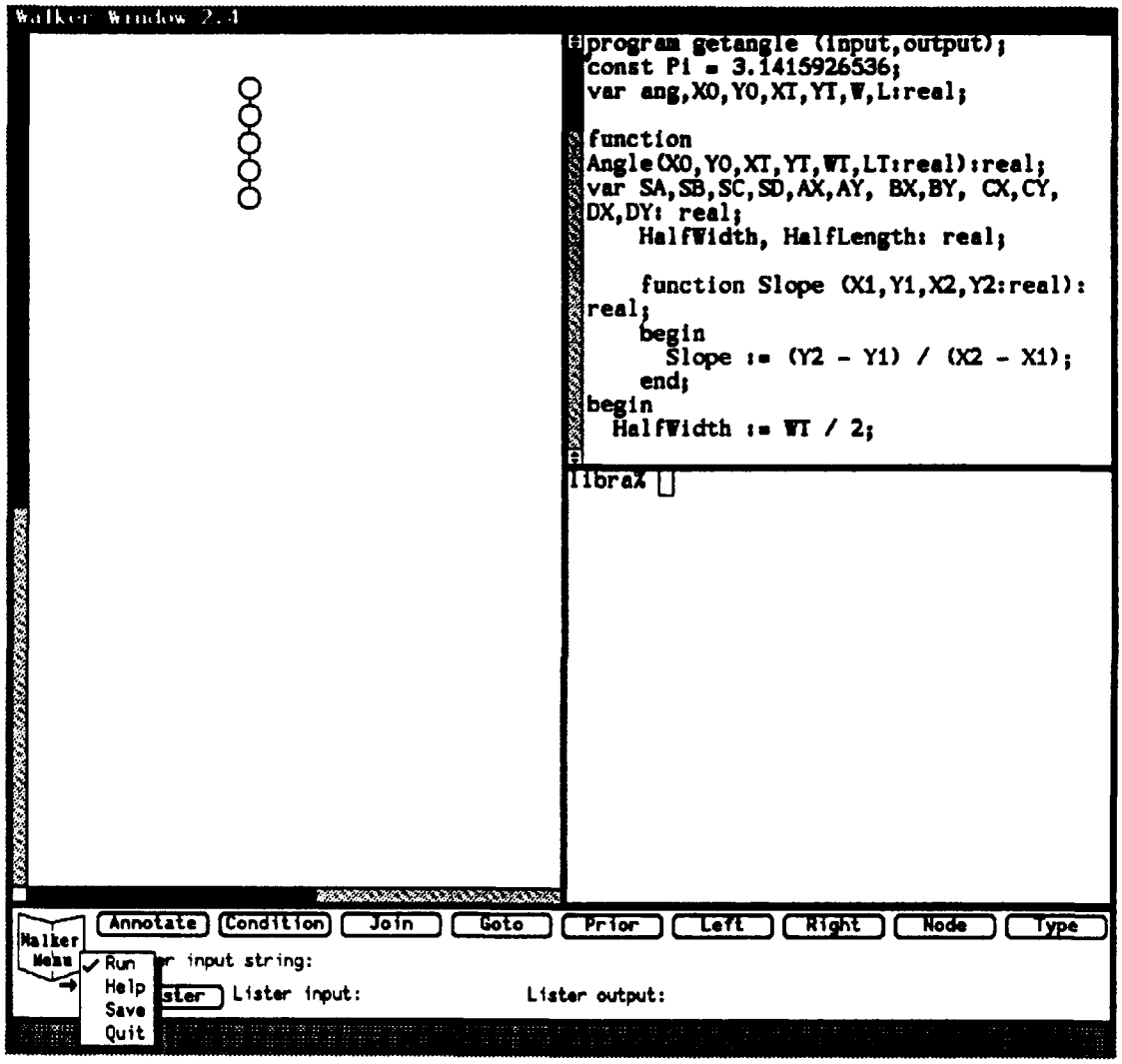

Figure 3.4 WALKWIN Menu

of possible commands, the user selects the "Help" choice from the WALKER menu. WALKER displays help information in the tty subwindow.

c. Traversing the graph

The user runs WALKER to achieve a couple of different functions. After REACHER sets up each link of the graph with the immediate reachability conditions, WALKER goes through the control flow graph and adds up those conditions. On the other hand, the user wants to get rid of pseudonodes. These are nodes that REACHER generates but the rest of the analysis doesn't require. The user should collapse these nodes, e.g., begin-end nodes. The user checks the code displayed in the text subwindow to be sure he is in the right place. The canvas subwindow gives the user an overview of the control structures; the text subwindow gives the user program details. The user selects buttons in the panel subwindow by positioning the pointer on the button and depressing the left mouse button.

The user starts at the begin node of the main program. This is node 71, located on line 70. The "@" prompt shows the line number in parentheses and the text of the line. A begin-end statement is indicated by a blank line. The "c" prompt shows the current reachability condition. It is always "true" for the first node. The program gives the condition it is about to set, not the condition used to reach this node. The program sets the conditions on the exit of the node. The left arc gives the condition and the right arc gives the not of the condition. The "a" prompt shows commentary made by the analyst. This field allows the analyst to make comments that will help him later

in the analysis process, but the comments are not processed by any tool. The user displays line 71 in the text subwindow by scrolling down the text.

The user knows he wants to traverse the nodes of all modules in the acfg. Looking at the acfg representation in the canvas subwindow, the user sees the traversal in the main module of getangle.p consists of a series of left branches. The user goes left from node 71 by clicking on the "Left" button in the panel subwindow. The program sends the proper command to the tty subwindow telling WALKER to go left. The user is now at the node 72, which is the "readln" statement of the main module. It is a good idea to join these two nodes, since node 71 is a pseudonode for the begin-end statement and unnecessary in later processing. The user enters the string "71 72" in the "Walker input" area. He then moves the pointer to the "Join" button, and clicks on this button to join nodes 71 and 72. The result is node 72, with a different line number and an updated text string. (See Figure 3.5.)

The condition is still true. The user goes left to node 73, which is a call to the function Angle. Node 74 is the assignment of the result of that call to the variable "ang". This is another pseudonode generated by REACHER. The other tools do not use pseudonodes so the user can join nodes 73 and 74. WALKER knows the text of one is a substring of the text if the other so it doesn't duplicate the text string. The line number is still 72. (See Figure 3.6.) The user goes left to node 75, which is the "writeln" statement of the main module. This is the last node in the main module. The user simplified the main module of getangle.p by joining pseudonodes to other nodes of the acfg.

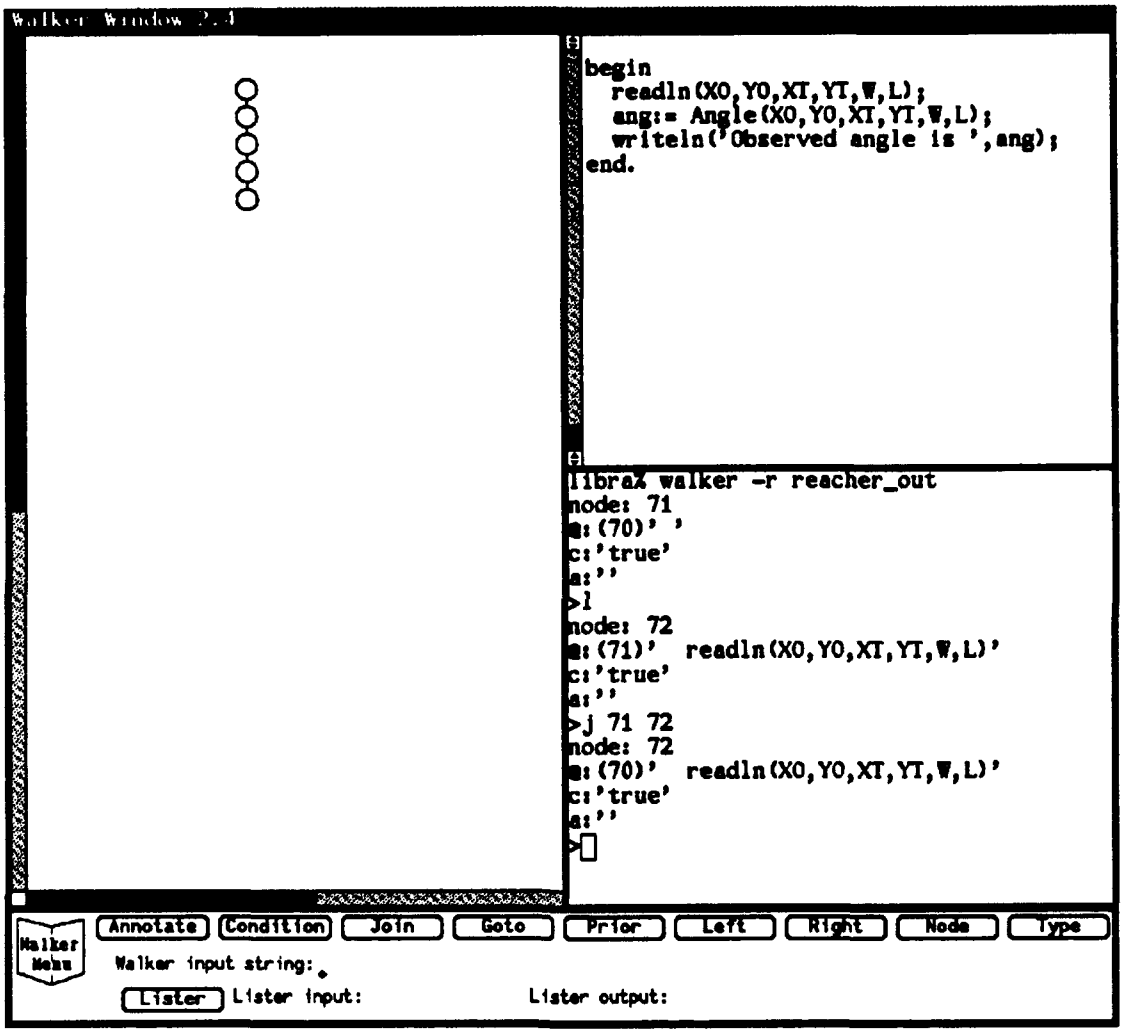

Figure 3.5 WALKWIN Screen With Initial Commands

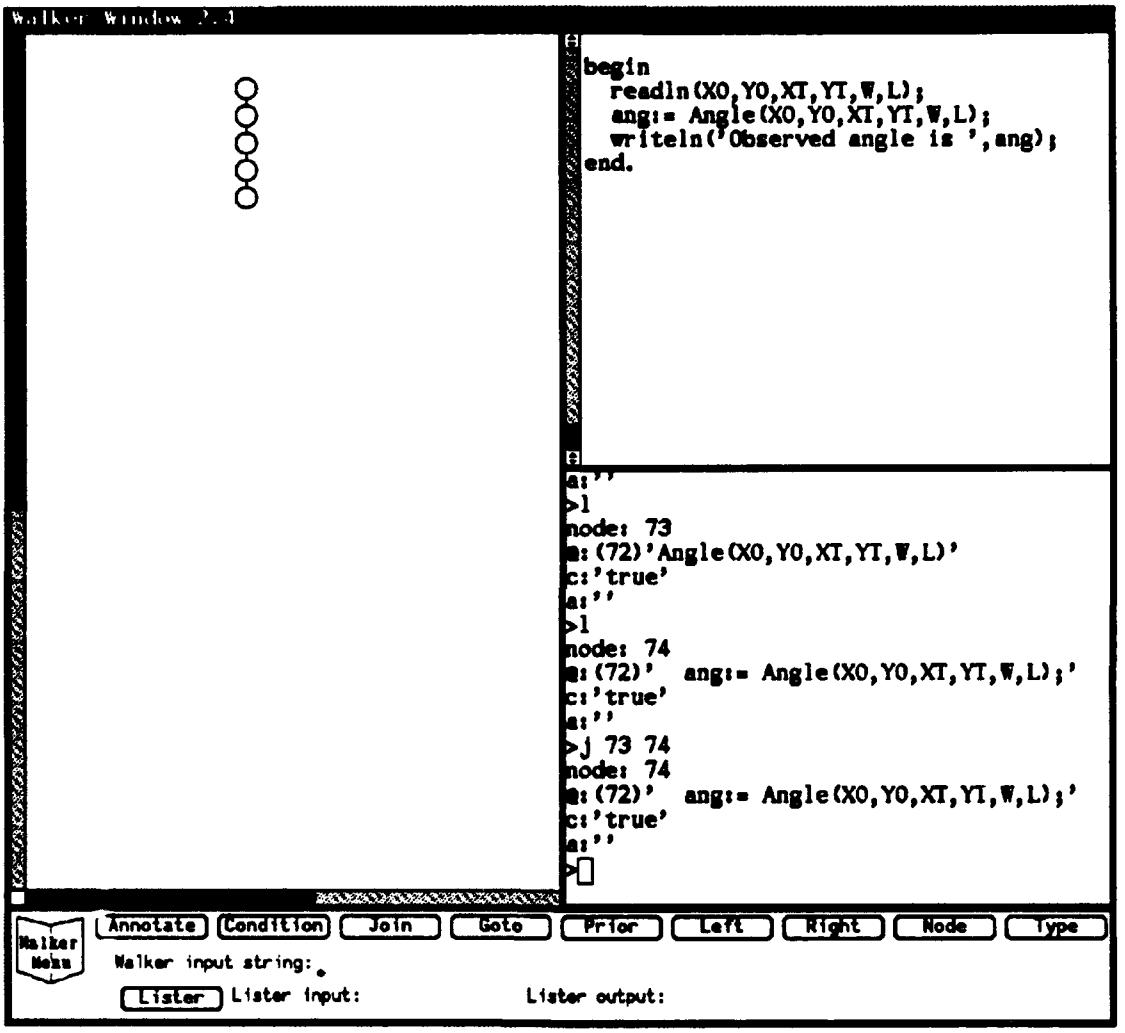

Figure 3.6 WALKWIN Screen While Processing Function Call Nodes

Now the user wants to go to the module Angle, the only program-defined module called from the main module. He enters the string "Angle" in the WALKER input area. The user then clicks on the "Goto" button. He is in the Angle module at node 4, line 13. He scrolls the text window to display the Angle function. This is a begin-end statement so the user goes left and joins nodes 4 and 5. The user traverses left several times until he reaches node 15. The statements he traverses are assignment statements. If the analyst desires, the effect of these assignments may be indicated by modifying the reachability conditions or SPACER may evaluate the effects by symbolic execution. In this case, the latter choice is made. (See Figure 3.7.)

From an examination of the code in the text subwindow, the user finds he is at the beginning of a series of assignment statements. Each assignment statement consists of a function call and the assignment of the result, so a pair of nodes represents each line of code. The user joins these node together as pairs, consisting of the function call and the assignment of the result. He joins nodes 15 and 16, nodes 17 and 18, nodes 19 and 20, nodes 21 and 22, nodes 23 and 24, nodes 25 and 26, nodes 27 and 28, and nodes 29 and 30, traversing left between the join operations. The analyst could proceed to further collapse these nodes further, reducing a linear code sequence to a single node, but this is not necessary to this analysis.

The difficulty in processing "if" structures lies in retaining the location in the code and the context of that location. In the Angle function, there are several "if" cases, which create a comblike "if" structure, starting with node 31. Only one of these cases will happen on each execution of Angle. All the cases gather together at the end

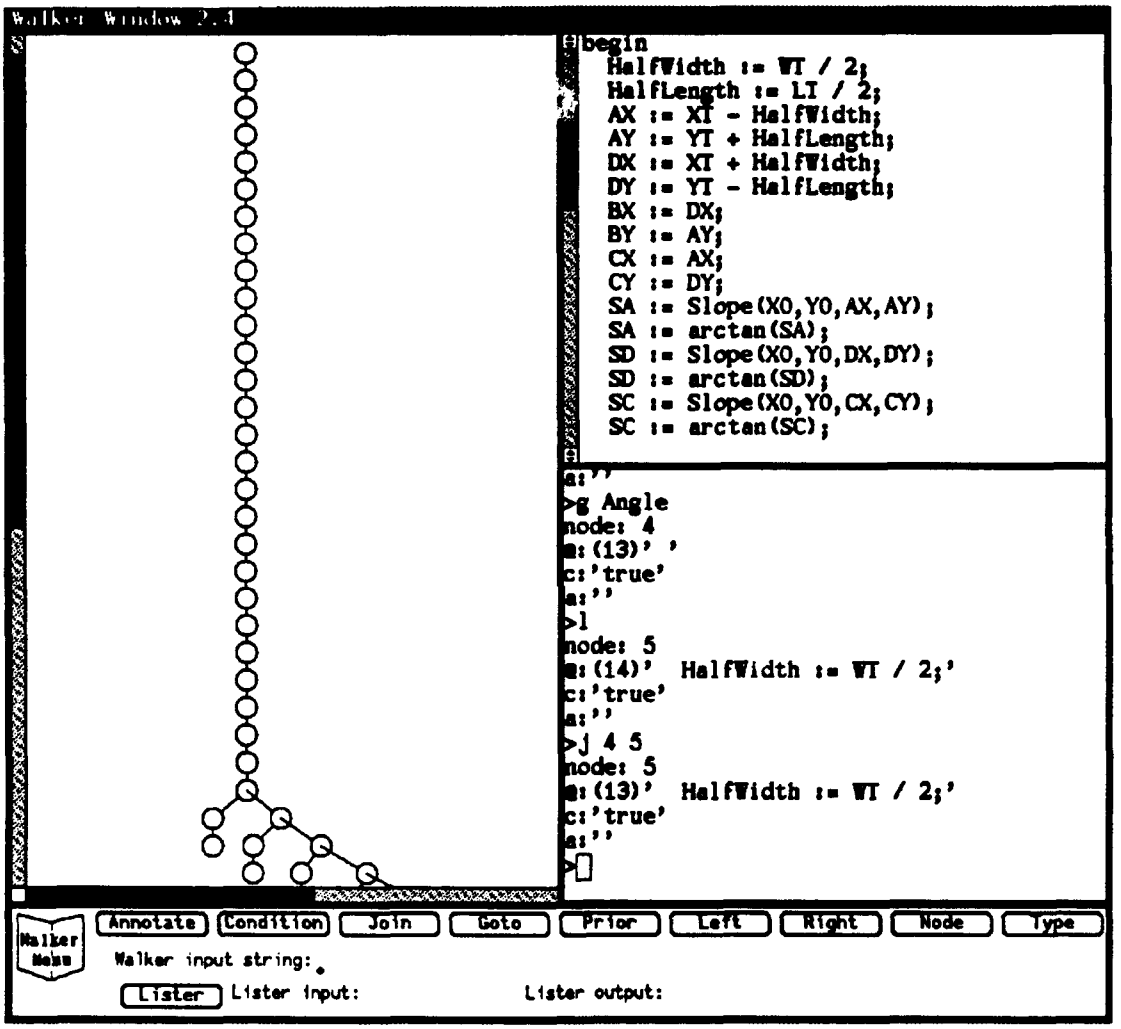

Figure 3.7 WALKWIN Screen While Processing Function Angle

into an exit node. The traversal process now involves forming up the condition that reaches that end. The user needs to keep track of each of these "if" cases. Each case has an immediate condition in the "if" statement and implied conditions it inherits by being the "else" of the previous "if" statements. WALKER does some automatic tracking of the conditions, but the user may need to intervene to simplify conditions. If the conditions are too complex, it can cause problems in later processing. SPACER has a limited amount of memory available, and the text has a limited allowable length.

To guide this simplification process, a sketch (such as Figure 3.8) may be used. Later extensions of WALKWIN may include some sketch capabilities, but in the prototype, external sketch facilities (e.g., a piece of paper) must be used. The sketch includes the location of the parameters (XT, YT) and reference points calculated by Angle from the parameters. These reference points (A, B, C, D on Figure 3.8) will be referred to in terms of their X and Y coordinates (e.g., A appears as AX and AY). The observer is in some area of the picture. There are more than nine cases because some are exactly on the lines, some are between lines only, and some include both lines and the area in between. The user keeps track of which cases he covers.

The user wants to systematically traverse the "if" structure. He will traverse the left branch first, and then return to cover the right branch of each "if" statement. The first "if" statement (node 31) contains the condition "(YO > BY) and (XO > BX)", so this area is in the upper right hand area, labeled Case 1. This case does not include the lines, because it is strictly greater than. The user goes left from node 31 and finds two nodes representing a function call and an assignment of the result.

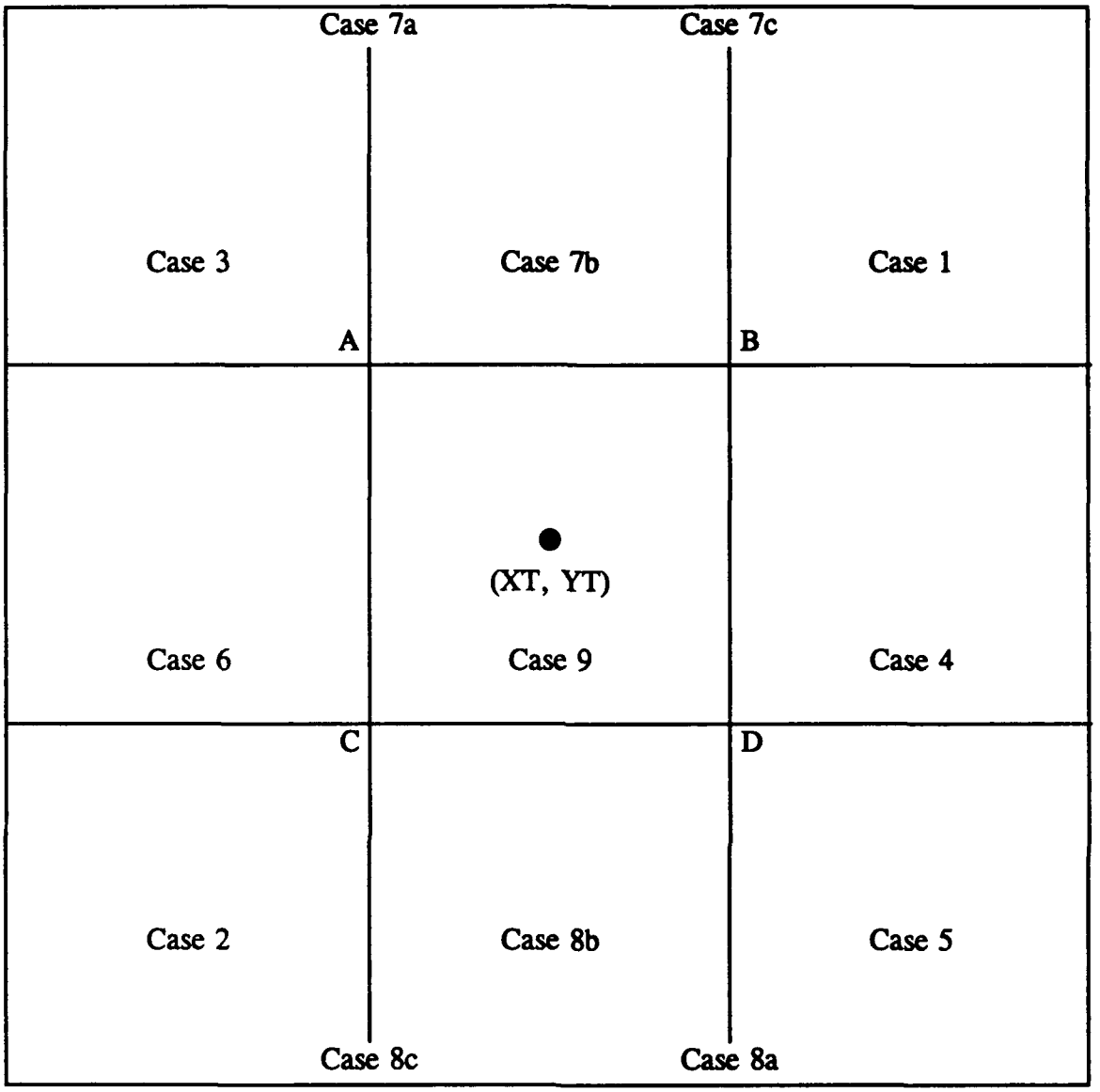

Figure 3.8 Sketch for getangle.p

These are node 32, which is the call to the "abs" function, and node 33, which is the assignment to the variable Angle. The user notices the condition is no longer true. It is now the inequality found in the "if" statement. The user joins nodes 32 and 33 to drop out the pseudonode call to the abs function. The user goes left from here and is at the

exit node. (See Figure 3.9.) This is a pseudonode created when REACHER generated the acfg. Each assignment statement could be the last statement in the procedure. FALTER and SPACER expect one exit node. So if there isn't just one exit node, WALKER adds an exit node and has everything that looks like an exit node goes there. The program automatically has a single entry node, since there is only one pointer in each header. The tools do not handle "goto" statements, so there is only one way into each module. Future versions of the tools may be able to handle "goto" statements.

The last node visited prior to the exit node was node 33. To continue the traversal, the user must jump back to the node corresponding to the "if" statement where the left branch was taken. That was node 31. The user goes to node 31 using the "Node" command. The user enters the number of the node he desires into the "WALKER input" area, and clicks on the "Node" button. Here the user enters "31" and returns to the "if" statement. The user goes right to node 34. The user goes left to the body of the "if" statement at node 35. The condition is "anded" onto itself, so the user simplifies the arc condition to the condition of the "if" statement only. He enters "((XO < CX) and (YO < CY))" in the "Walker input" area, and clicks on the "Condition" button.

This duplication is the result of coming out of a node and jumping to a node rather than using an arc. The conditions are on the arcs rather than on the nodes. When the user jumps out and returns to a node, there is no condition set. This condition is a copy of the condition in the "if" statement there. The user notes that this covers Case 2, the lower left hand area. The condition of Case 2 implies the negation of the

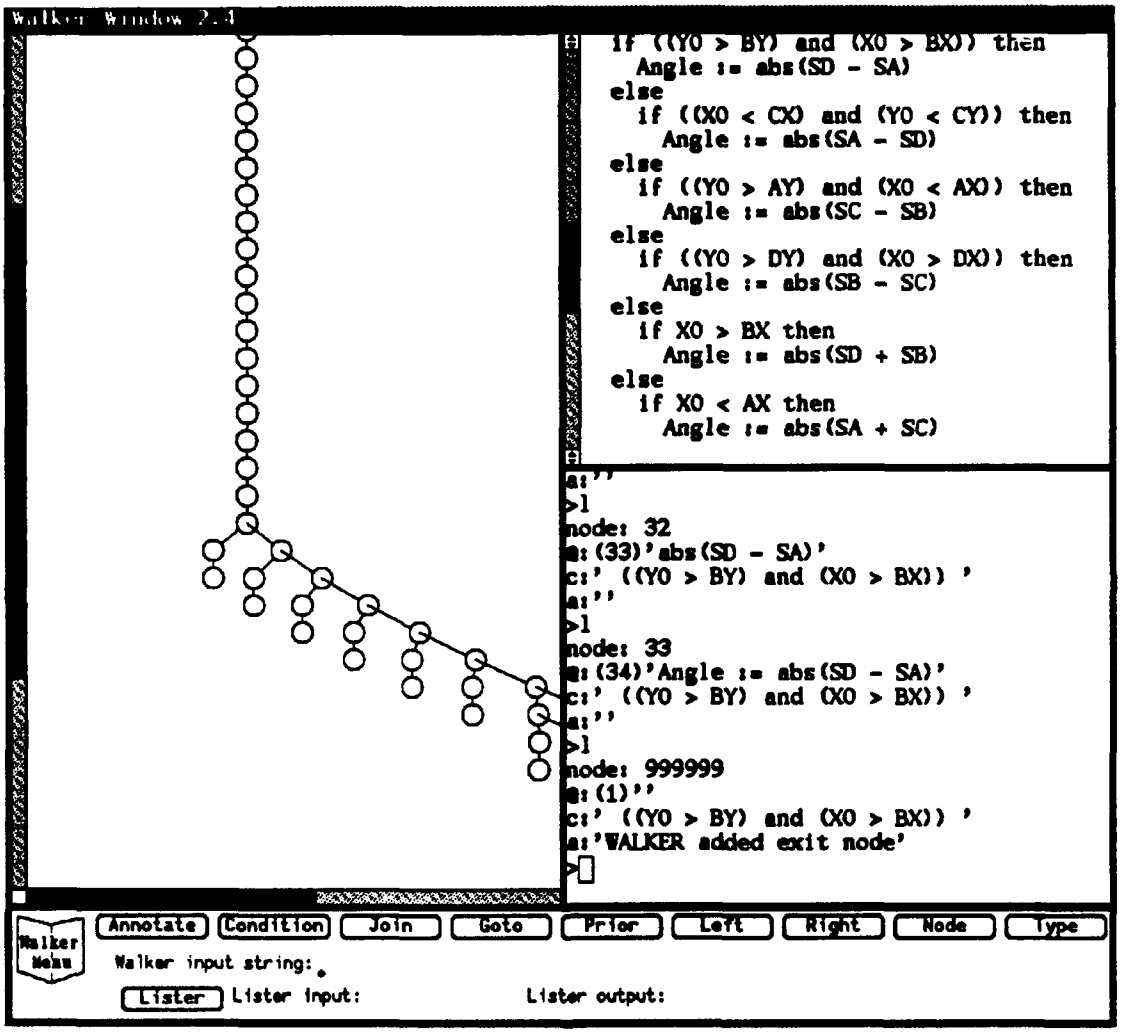

Figure 3.9 WALKWIN Screen While Processing If

conditions of Case 1 so the user doesn't need to add in the not of those conditions. The user goes left to node 35. The user joins nodes 35 and 36, getting rid of the pseudo call to the "abs" function. The user goes left and arrives at the exit node.

The user now must return to node 31 in order to traverse to the false branch of node 34. He returns to node 31 using the "Node" button. He goes right from node 31 to node 34 by clicking on the "Right" button. (See Figure 3.10.) The condition set is the "not" of the condition in the "if" statement and itself. The user simplifies the condition by eliminating the duplication. The user goes right to node 37. The user sees that this is Case 3, where "YO > AY and XO < AX" and it doesn't include the line. He enters this condition for node 37. By keeping track of the cases already covered, the user can later express what the condition is in a certain area rather than what it is not.

The user wants to cover both branches from node 37. He goes left to node 38. Again he simplifies the condition to "(YO > AY) and (XO < AX)". The user joins nodes 38 and 39, which is the call to the abs function and the assignment of the result. The user now goes left and is at the exit node again. He returns to node 37 to cover the right branch. The user simplifies the condition by setting it to "((YO > AY) and (XO < AX))". The user goes right to node 40 and simplifies the condition. Here the "if" condition is "YO > DY and XO > DX", so the user is in Case 4 of Figure 3.8. The condition on node 40 contains duplication, so the user simplifies the condition. The user adds in the two conditions to make one simpler condition.

Node 40 has both a left and right branch. The user goes left to node 41 and simplifies that condition. The user joins nodes 41 and 42, and goes left to the exit

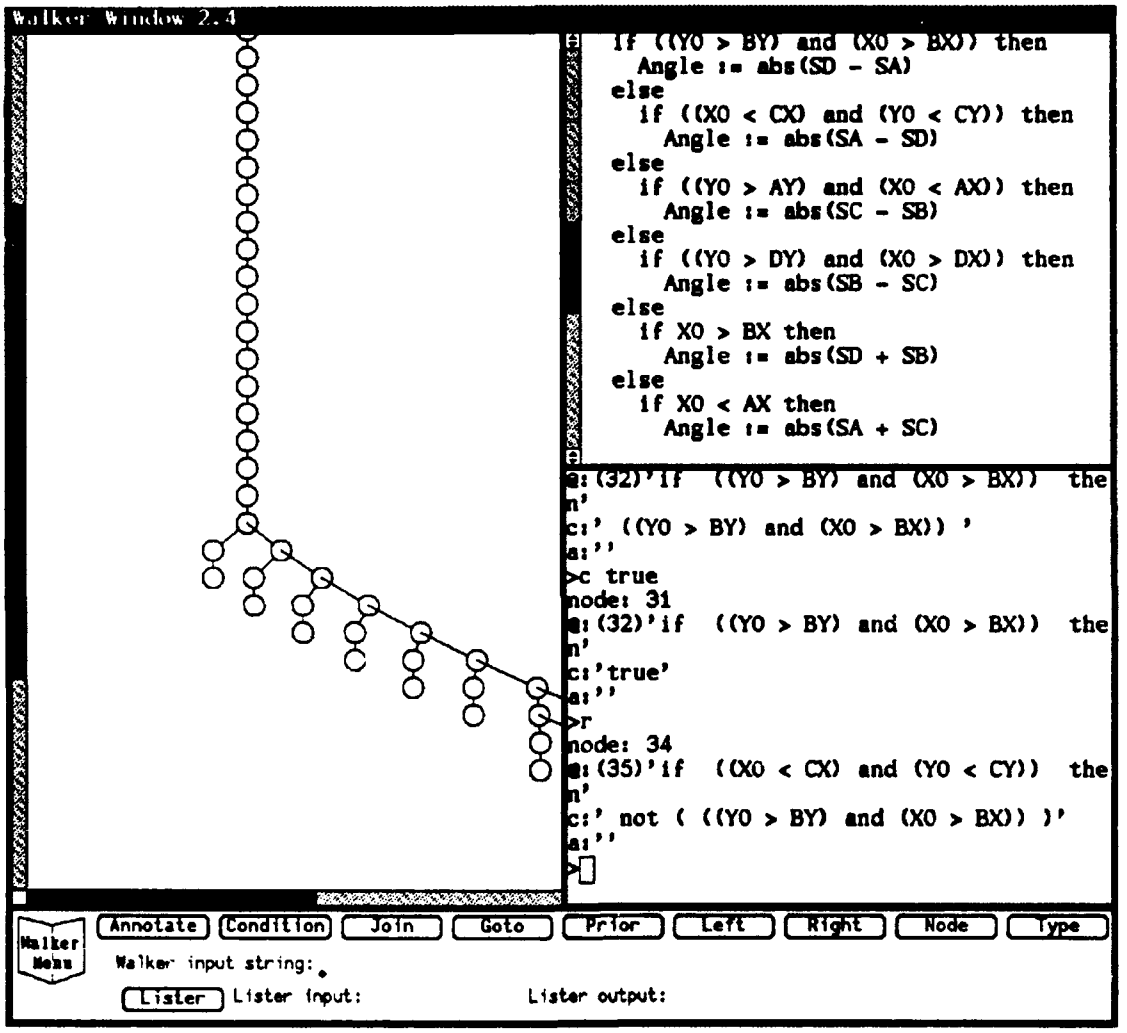

Figure 3.10 WALKWIN Screen With Further Node Commands

node. The user returns to node 40 and simplifies the condition by setting it to true. When the user goes right to node 43 WALKER automatically adds in the not of the condition "XO > BX" from line 44. The user already eliminated out Cases 1 and 4 so the condition includes "YO < = DY"; he adds that to the "if" condition. This adds in the conditions that have accumulated so far, so "XO > BX and YO < = DY". This is Case 5. The user goes left and joins nodes 44 and 45. He simplifies the condition and goes left again to reach the exit node.

The user returns to node 43, the "if" statement for "XO > BX", to cover the right branch off node 45. Node 43's condition is too complex, so the user simplifies it to "YO > = DY and YO < = BY". He goes left to node 45 and then right to node 46, putting himself in Case 6. This is the condition where "XO < AX" and Cases 2 and 3 have been covered. So he also has "YO < = AY and YO > = CY" as part of the condition and he changes the condition to reflect the more restrictive of the two.

From node 46 there is a left branch and a right branch. The user goes left and joins nodes 47 and 48. The user goes left and is at the exit node. The user returns to node 46 and this time goes right to node 49. He has the condition where "YO > = AY" so it is the case of the line AB in figure 3.4 and everything above it. Since the user already traversed Cases 1 and 3, he is in Case 7 of the diagram. However, the program divides Case 7 into several subcases. So the user can set some limits on XO by setting the context for YO here. It is not pertinent to worry about excluding the parts previously done. Another statement on AX will exclude more areas later. The user sets the condition to true. All he worries about is setting the YO condition on the next node.

The user goes left to node 50. Now he has the XO set up so if "XO = AX" and the "if" statement is true he is on the line AC above line AB, which is Case 7a in the diagram. The user adds the comment "XO is set up". The user goes left and then joins nodes 51 and 52. The user goes left and is at the exit node.

The user returns to node 50 to cover the right branch. He handled the body of the first double "if" so he goes back to the "XO = AX" condition. He changes the condition to "YO >= AY". The user goes right to node 53. He knows he's not on the line and "XO < BX" is the condition of the "if" statement. He wants to simplify this down to "XO > AX", because he handled the parts where XO < AX and where XO = AX. He knows XO < DX so he is not on the line BD, but is between the lines in Case 7b. He changes the condition to "(XO > AX) and (YO >= AY)". From node 53, the user goes left to node 54 where there is a call to the "abs" function. Node 55 is also a call to the "abs" function" and node 56 assigns the result of "Pi - abs(SA) - abs(SB)" to the variable. There are two pseudonodes, so he does a three way join. The user joins nodes 55 and 56, and then joins nodes 54 and 56. This order of joining keeps the links connected and avoids the loss of nodes. The user goes left to the exit node. (See Figure 3.11.)

The user returns to node 53 to traverse the right branch from the condition "XO < BX" in the "if" statement. He simplifies the condition of node 53 again. The user now goes right to node 57, and is under the else condition to reach Case 7c. The user has dealt with the line AC in Figure 3.8 and the area between the lines AC and BD and has to deal with the portion of line BD above line AB. He does some

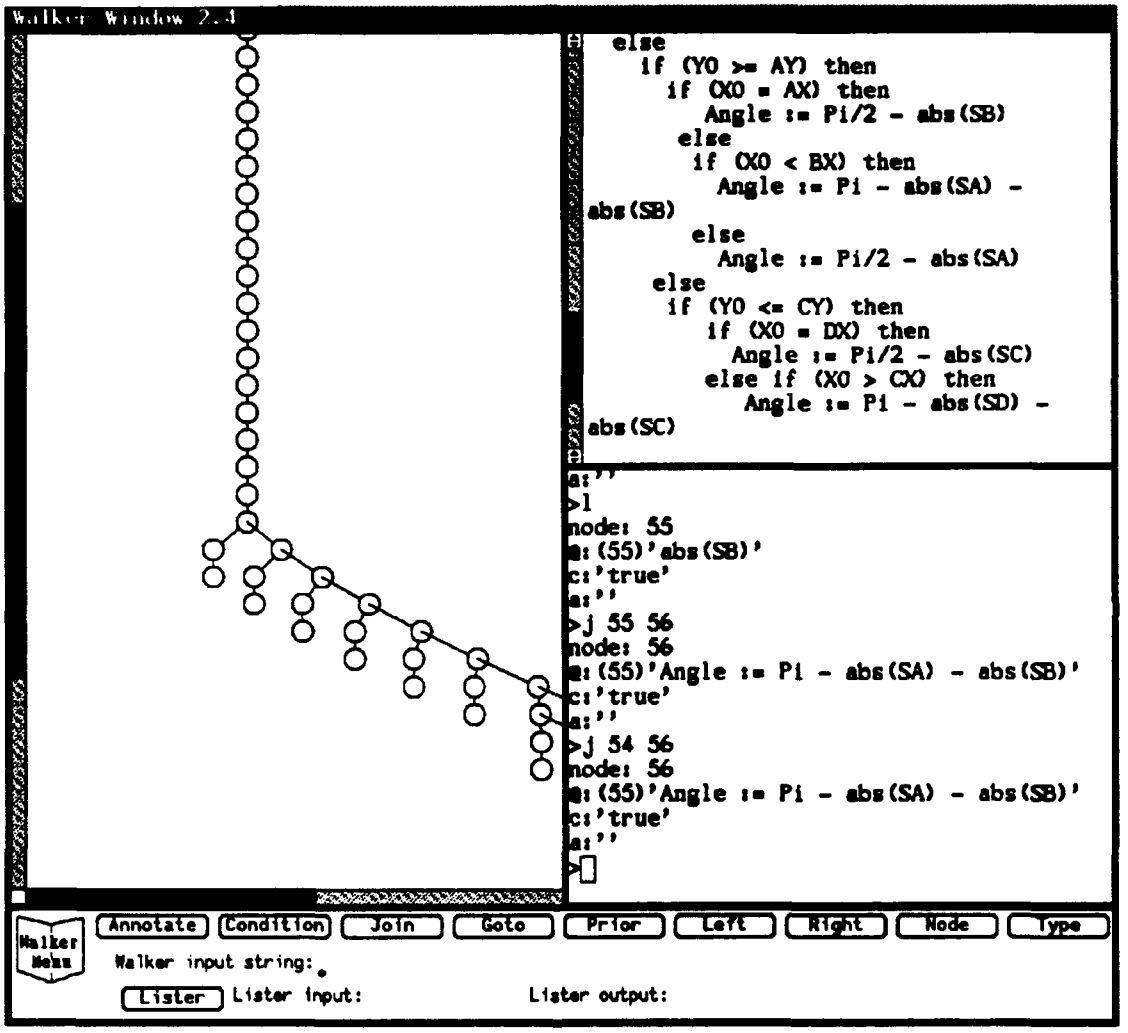

Figure 3.11 WALKWIN Screen Showing Multiple Join

simplification and sets the condition to "XO = BX and YO >= AY". The user goes left and joins nodes 57 and 58. The user goes left to the exit node.

He returns to node 49 to traverse the right branch from the "if" statement with "YO >= AY" as the condition. The user sets the condition to true and goes right to node 59, so "YO <= CY". The user is now in Case 8. Case 8 also has several subcases. If "YO <= CY" then it is not >= AY so the user sets the condition to true. The user then goes left to the condition "XO = DX" at node 60. This is the case where the observer is right on the line BD in Figure 3.8 and is in Case 8a. There is a left and right branch from node 60. The user goes left from there. Then he goes left again and joins nodes 61 and 62. He goes left again and reaches the exit node. The user goes back to the "if" statement at line 60 and sets the condition to "YO <= CY". The user goes right from node 60 to node 63 where "XO > CX". He knows here that in addition to "XO > CX", that "YO <= CY" and "XO < DX". He has handled the line BD in figure 3.8 and is in the middle at Case 8b. The user simplifies the condition to "YO <= CY and XO < DX".

Node 63 has a left and right branch. The user goes left to node 64 and finds another triple node assignment. He joins nodes 65 and 66, and then joins nodes 64 and 66. The user goes left to the exit node. The user returns to the XO > CX "if" statement at node 63 and simplifies the condition so "YO <= CY and XO < DX". The user goes right to node 67. He is down in the assignment statement, so he joins nodes 67 and 68. The user goes left and is at the exit node.

The user has to cover the else part of the "YO <= CY" condition, which is node 59. He goes to node 59, where the condition is "YO <= CY". The user has done everything except the internal square, Case 9. So he knows he is going to get the not of the condition and considers the previously covered cases. He changes the condition to "YO < AY and XO >= AX and XO <= BX". The user sets the upper limit on YO and the two limits on X. This case includes the side lines but not the top and bottom lines. The user goes right to node 69 and is in the angle assignment. He goes left to the exit node and completes the traversal of the Angle module.

There is one module left to traverse, the Slope function. The user goes to the Slope function using the "Goto" button. He is at node 1, and joins nodes 1 and 2. This completes the traversal of "getangle.p".

d. Saving work

Once the user finishes traversing the graph, he needs to save his work. He enters the filename "walker_out" in the WALKER input area and selects the "Walker Menu" with the right mouse button. He highlights the "Save" selection, and releases the mouse button. (See Figure 3.12.) WALKER writes the file to the disk. Appendix G contains a copy of "walker_out". This does not exit the user from WALKER. The user now selects the "Quit" choice from the "Walker Menu". This ends the interaction with WALKER and renders the buttons and menu options other than "Run" inoperative. WALKWIN sends a clear command to the tty subwindow to clear the screen.

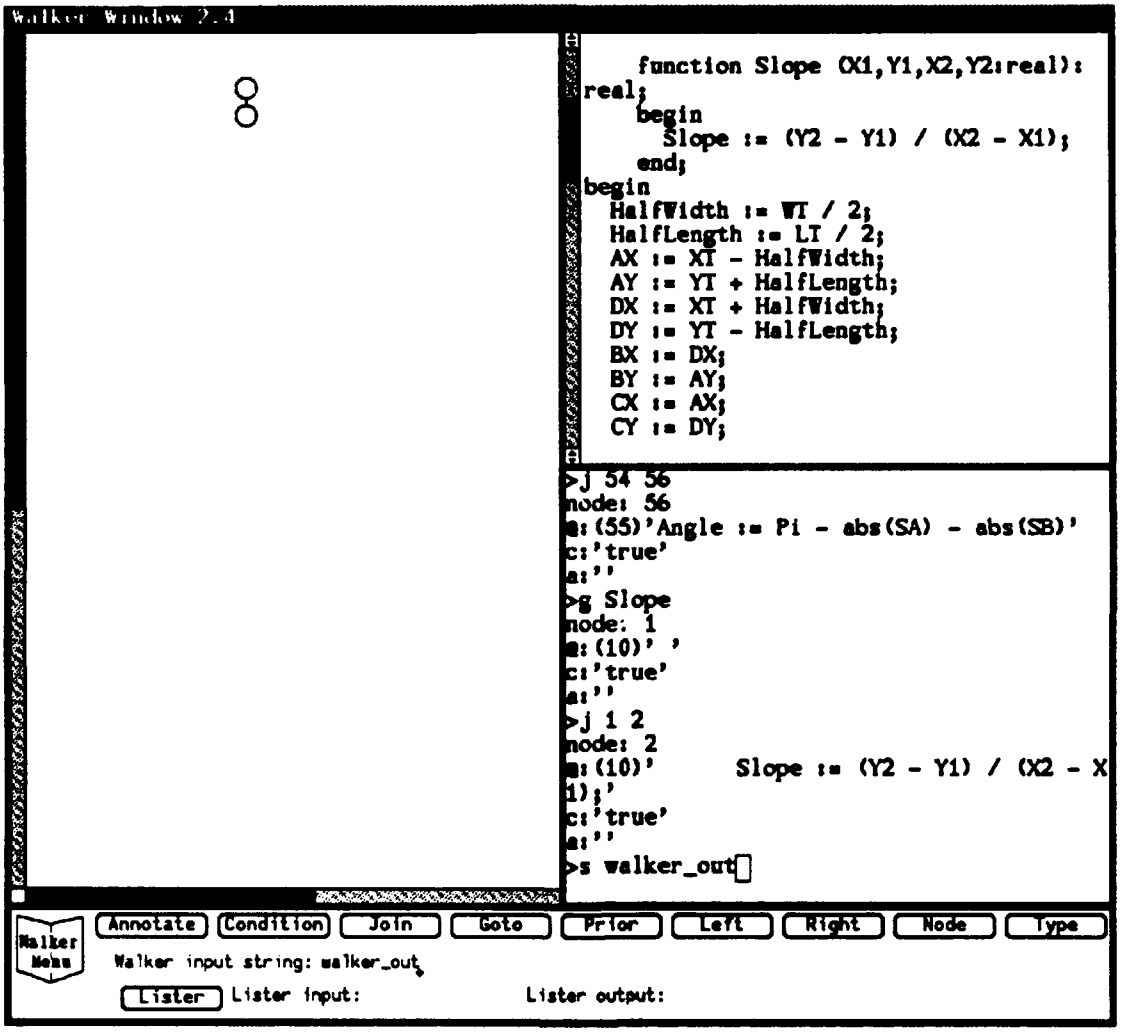

Figure 3.12 WALKWIN Screen Showing Save

e. Running LISTER

WALKWIN uses the button labeled LISTER in conjunction with the other two text items in the panel to get the annotated source listing. The user enters the input filename in the "Lister input" text item, in this case the filename is "reacher_out". To activate this text item, he moves the pointer to the rectangle containing the text item, and clicks the left mouse button. The movement of the caret to the end of the label signals that it is active. It also blinks. The user activates the "Lister output" text item and enters the filename desired. He chooses "getangle.list" as the name for the output. He moves the pointer to the "Lister" button and clicks the left mouse button. (See Figure 3.13.) The tty subwindow displays the output and provides feedback that LISTER saved the file. Appendix H contains a copy of "getangle.list".

This concludes the portion of failure region analysis done via WALKWIN. The user closes the frame to an icon or exits from the frame using standard SunView menu options.

4. FALTWIN

a. Initiating FALTWIN

The user displays the FALTER interface by clicking on the "Falter" button in the VIEWWIN panel subwindow. FALTWIN can use the same filename used by REACHER, or another text file, or no file at all. Normally FALTWIN uses the same Pascal file used by WALKWIN, so the filename remains in the text entry area. FALTWIN displays the Pascal file in its text subwindow (just like WALKWIN).

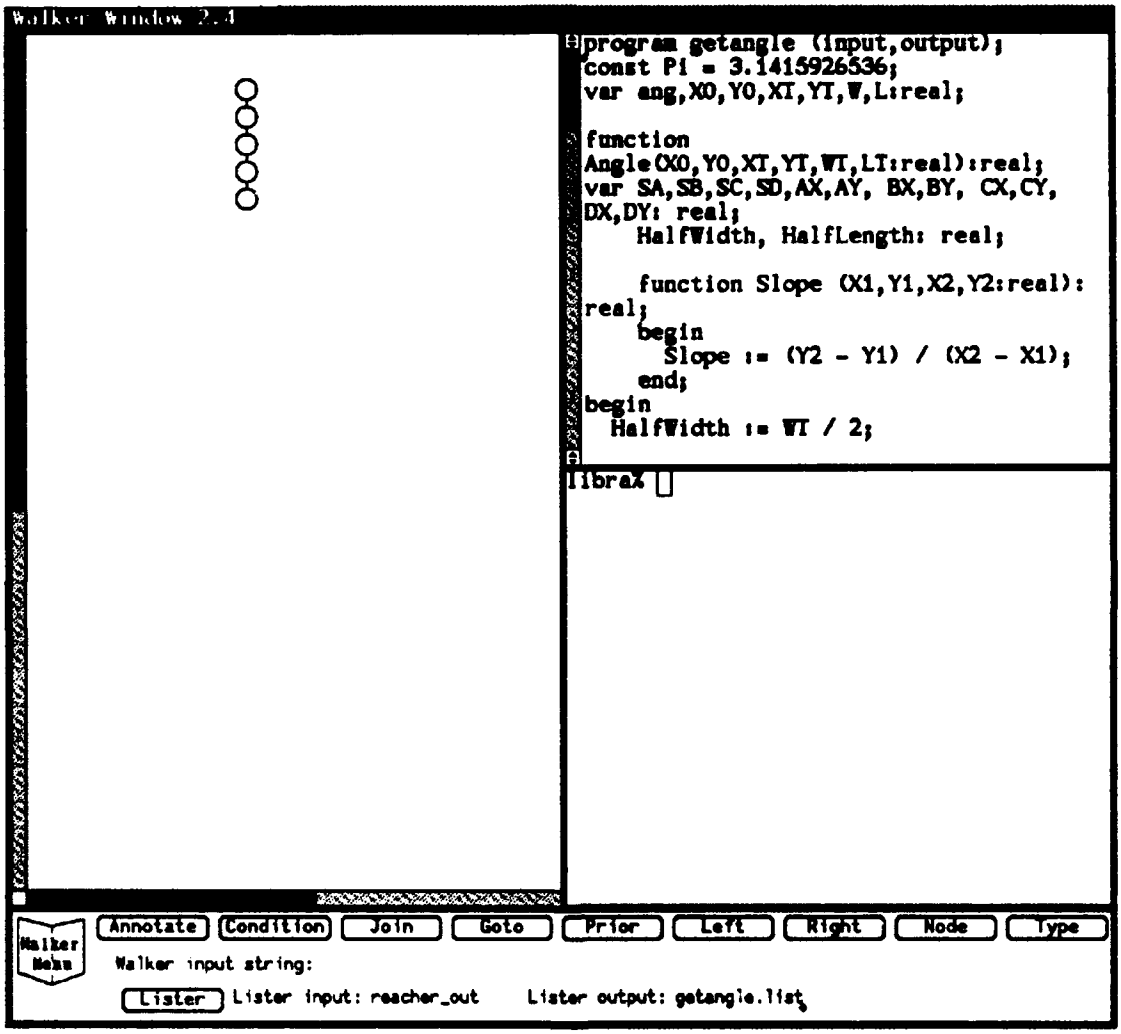

Figure 3.13 WALKWIN Screen Showing Call To LISTER

FALTWIN has more panel items than WALKWIN. They are for the additional commands used with FALTER. (See Figure 3.14.)

b. Traversing the graph and setting the fault

The directory contains the files reacher_out and walker_out. The user must enter a filename in the "Falter input" text item. He indicates the output saved from WALKER named "walker_out" in the "Falter input" area, and invokes FALTER by selecting the "Run" choice from the "Falter Menu" via the right mouse button. FALTER loads the file and places the user at the "readln" statement in the main module of the "getangle.p" program. The graph representation of the main module is in the canvas subwindow.

The prompt for FALTER is different from WALKER. It has some additional fields. The "@" prompt is the text of the line under consideration. The "l" is the local condition. The "e" prompt shows the error condition (initially 'false') and the "c" prompt shows any comments entered by the user.

Since WALKER has set up all the reachability conditions, the user uses FALTER to record what the fault effect will be and the conditions under which that fault will occur. The user will normally record a comment about the fault. FALTER doesn't really do any analysis. It is a recording mechanism, allowing the user to traverse the acfg and be sure he is where he wants to be. The process requires the user to debug the program and find the problem prior to using the tools. Now he defines the immediate effects of the bug, i.e., the effects on the next state. FALTER represents this effect by which variable values are contaminated. SPACER doesn't need to know about the bad

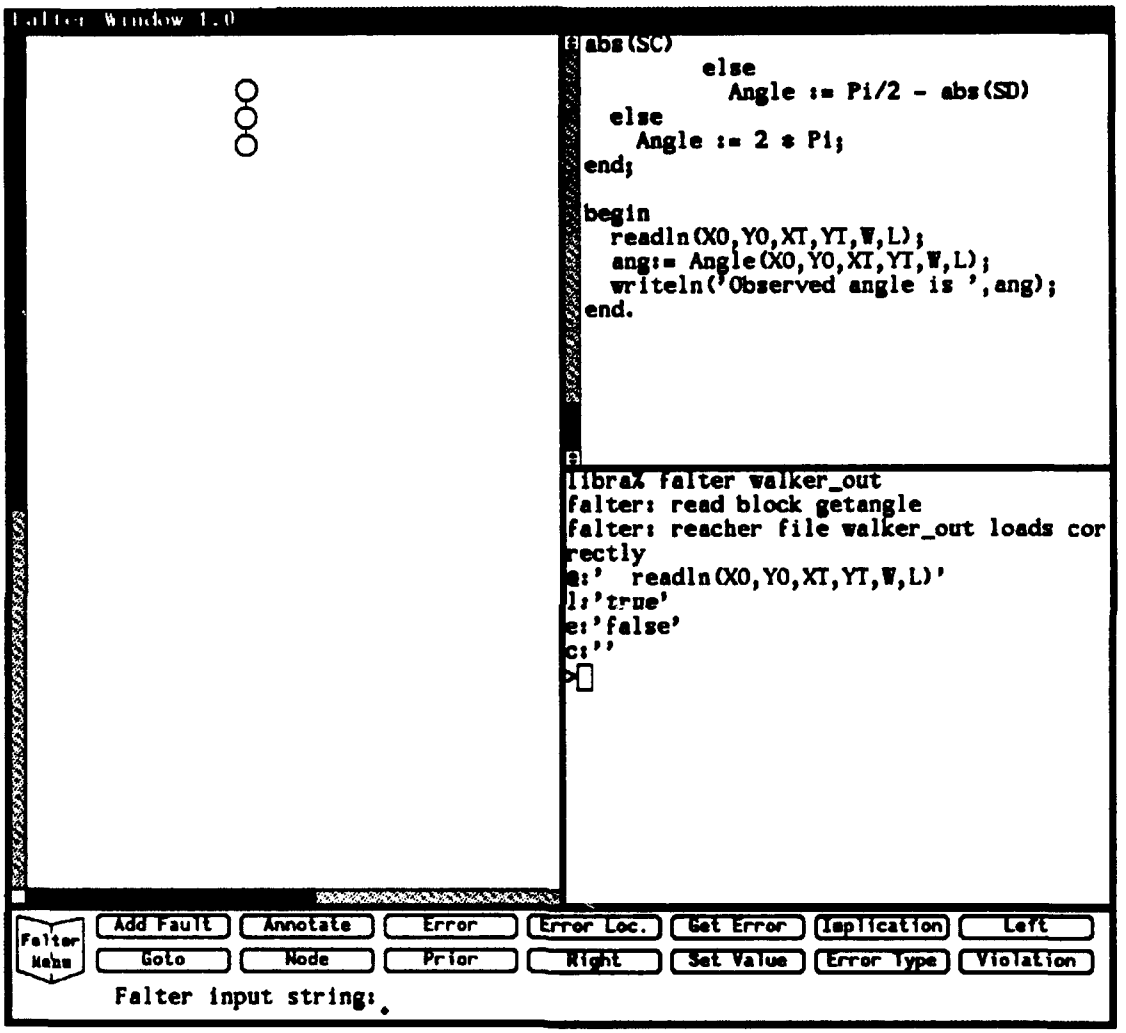

Figure 3.14 FALTWIN Screen Showing FALTER Initialization

value, only that it is contaminated. SPACER assumes that if the value is contaminated and the failure generation condition occurs that the value is always wrong. The user represents this in the error generation condition and identifies which variables have bad values.

The user goes to the Angle module using the "Falter input area" to enter the text "Angle" and clicking on the "Goto" button. The fault in "getangle.p" is in case 4. The SC in the computation "abs(SB - SC)" under the condition "YO > DY and XO > DX" is wrong. It should calculate the angle between points B and D, but is using points B and C instead. In certain cases the angle will be less than what it should be. This is a naturally occurring fault.

The user wants to go to the node in Case 4 where the fault occurs. He traverses left through the graph. If the user knew the node number automatically, he wouldn't have to go through the traversal. Note that FALTER doesn't make any changes until the user tells it to add a fault. The user can traverse at will with no changes made, unlike WALKER which modifies the structure. WALKER can go left and right without changing, but the default is to annotate the control flow graph as it sets up reachability conditions.

The user goes left from the nodes until he reaches the first "if" statement where "YO > BY and XO > BX". He goes right from the first "if" statement and right from the second "if" statement where "XO < CX and YO < CY". He goes right from the third "if" statement where "YO > CY and XO < AX" and goes left from the fourth "if" statement where "YO > DY and XO > DX". This is Case 4 of Figure 3.8. He

has found the bug and can add a fault. To add a fault the user clicks on the "Add Fault" button. FALTER identifies this fault as "Angle 1 type 13" with a "#" prompt. (See Figure 3.15.) The type is a user defined field to allow several different classes of faults. It can be used to draw statistics about the failure region analysis.

The "Viol" is where in the specification the violation occurs. This is why the code is wrong. This bug violated "observation condition 1" of the specification of the program. The user sets the violation by entering the text into the "Falter input" area and clicking on the "Violation" button. The "i" prompt is for the implication of the fault. The implication of this mis-assignment is the variable given a bad value, in this case "Angle." The user sets the implication by entering "Angle" in the text item area and clicking on the "Implication" button.

The error generation condition is currently false. The user sets the error condition based on the fact that the programmer used SC when he meant to use SD. When will this error matter? When SC and SD have different values. When will that not happen? When SC and SC have the same value, which would be when observing something with a zero width. It is possible that the item under observation may have a zero width. For all the user knows that is legal. Maybe there is other code before this that would prevent it but the user doesn't know that. The variable "W" is the value of width so when "W < > 0" the program will generate an error. The user enters the text in the "Walker input string" area and clicks on the "Error" button to set the error. (See Figure 3.16.)

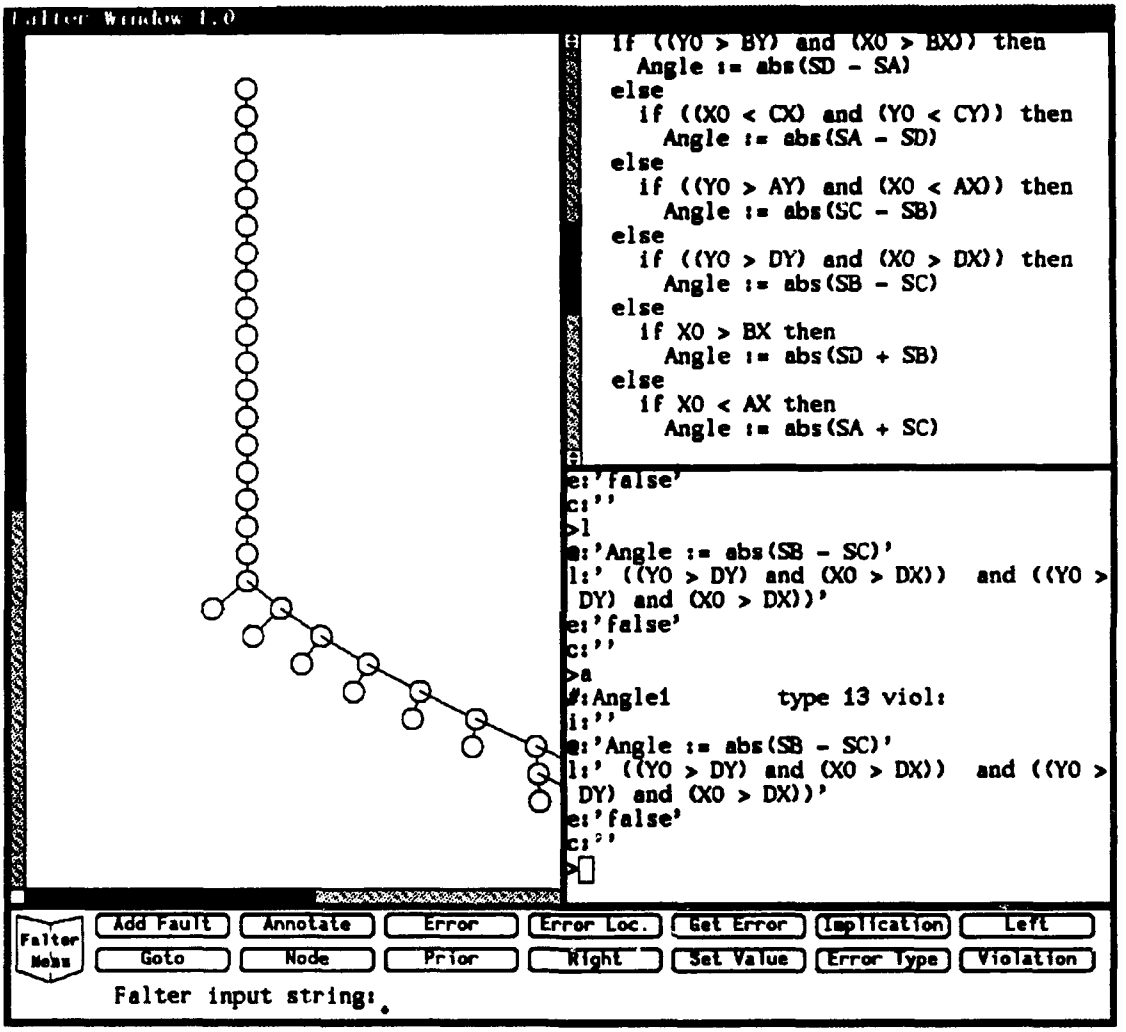

Figure 3.15 FALTWIN Screen Showing Preparation of Fault Annotation

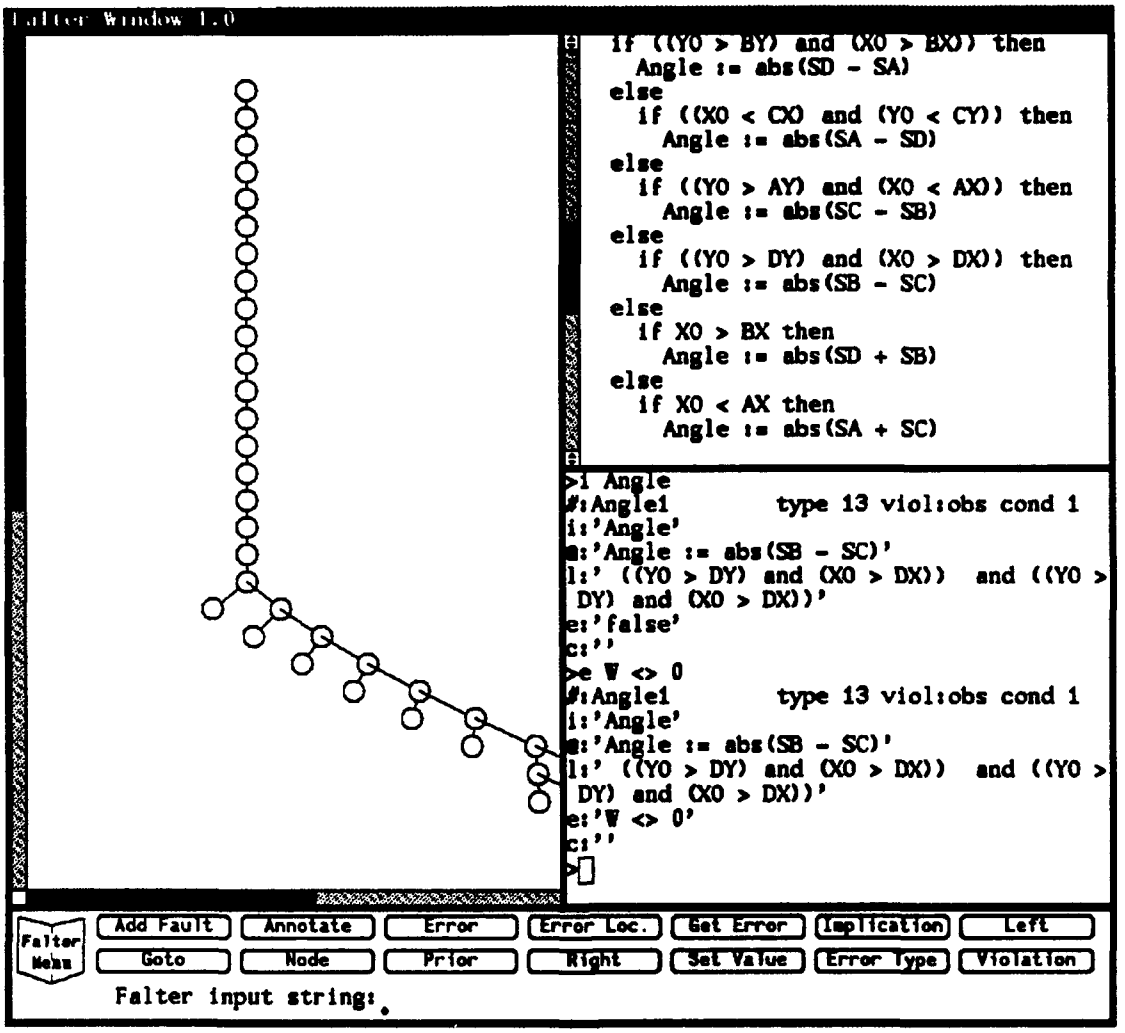

Figure 3.16 FALTWIN Screen Showing Fault Annotation

The user can annotate the acfg and record a comment about the bug. He enters "mis-substitution of SC for SD in this assignment" in the text input area, and clicks on the "Annotation" button.

c. Saving work

In setting up these fields, FALTER sets some of them directly and some of them in temporary variables. The user wants to have a consistent set saved to internal memory to have a complete record of the fault. The user can save the fault description in two ways. (See Figure 3.17.) It can be saved in Lisp format for use by SPACER. The user enters the filename in the "Falter input" text item, and selects "Save" from the "Falter Menu" button via the right mouse button. This sets the current error condition to match the temporary values displayed.

The file also can be saved in acfg format for use with FALTER. To save a file in acfg format, the user must include a "-" in front of the filename he enters in the "Falter input" area. This method saves the acfg so the user can read it back into FALTER and work with it. It allows the user to have several fault descriptions for the same acfg. The user could compare statistics for the faults in the file if there are other analysis tools to read acfg format statistics on failures in control flow graphs. Currently these types of tools don't exist.

The user saves the results as a SPACER-usable file with the name "spacer_in.l". The user selects "Quit" from the Falter menu to end the interactive session with FALTER. The "spacer_in.l" file is much larger than the "getangle.p" program. Appendix I contains a copy of "spacer_in.l" code. The "getangle.p" code is

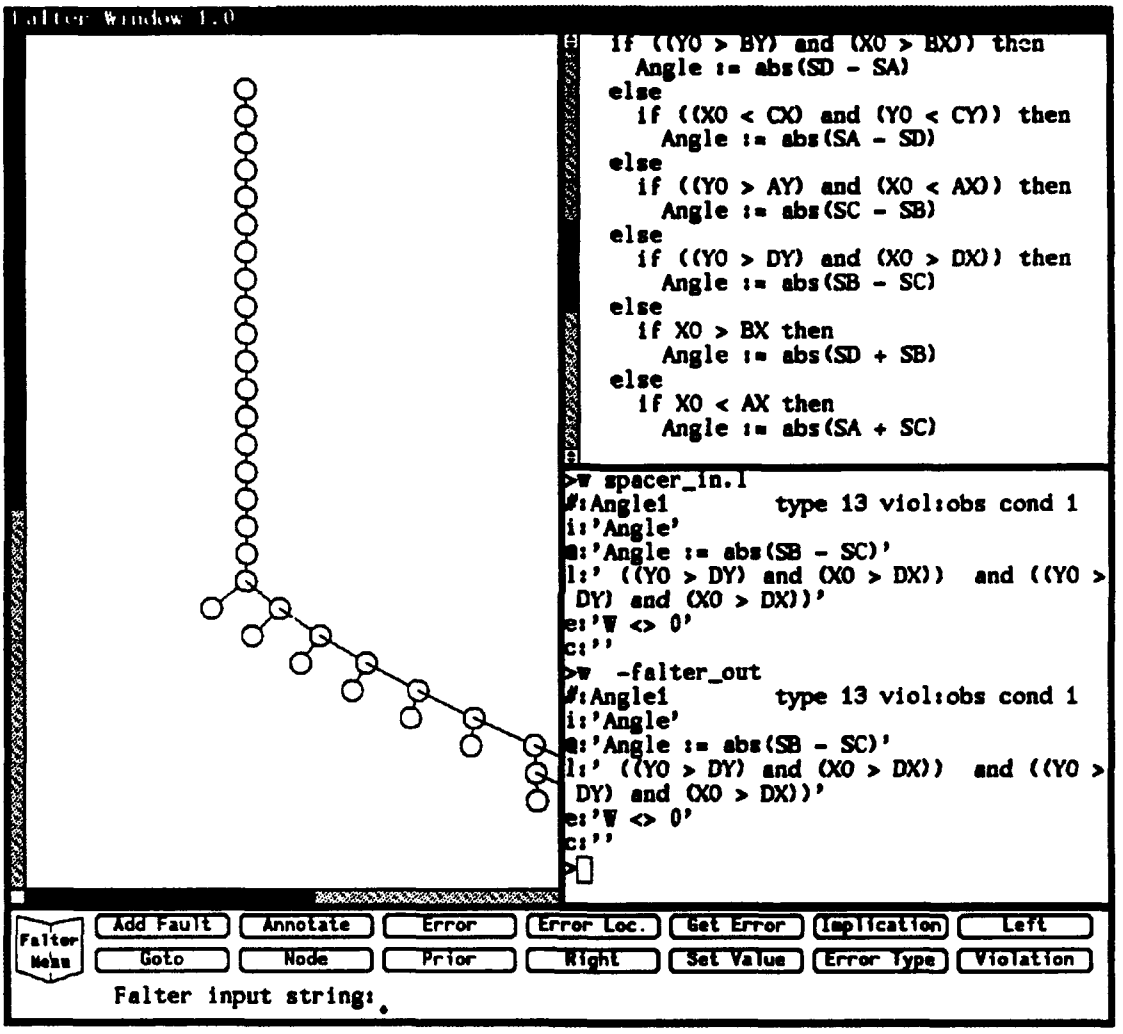

Figure 3.17 FALTWIN Screen Showing Save

74 lines, and "spacer_in.l" is 541 lines. (Reminder: Appendix E contains a copy of the source code for "getangle.p".)

5. SPACEWIN

The last step in failure region analysis is SPACER. SPACEWIN allows the user to invoke LISP, and then run SPACER using the output from FALTER. More sophistication in SPACEWIN is an area for future development.

The user copies the file "spacer_in.l" to "temp.l" since SPACER automatically loads "temp.l" during processing. A copy of "getangle.p" is available to SPACER as "temp.p" in the directory. SPACER loads the files then prompts the user to select the fault to be generated into a failure region. The user then is given the option of setting the level of information display and other SPACER options. When finished, the user types "go" and SPACER will emulate program execution to generate the failure region.

IV. CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE RESEARCH

A well-organized user interface contributes to the efficiency of the software tester. Alleviating the requirements to recall commands and providing feedback is essential. The use of graphics and a windowing environment is becoming more commonplace in software.

The interface can provide a logical organization of the information needed by and provided to the tester. The ability to move from code display to a graphical representation of control flow is useful.

A. CONCLUSIONS

This thesis has been an in-depth examination of interface design, testing tools, and the interaction between the two. It also is the first development of a graphical user interface for the failure region analysis tools. VIEWER satisfies many of the design principles for interfaces. It presents the results and outputs of the testing tools with consistency and coordination. It is similar to other tools in SunView, so it retains a feel that is natural to SunView users. There is a structure to the interface, while maintaining a degree of flexibility for the tester.

The complexity and richness of the SunView language allowed rapid construction of a prototype. However, further refinements and enhancements are necessary.

B. LESSONS LEARNED

Tools can represent information in a screen-oriented way. The interface can key the views around features of information known to be significant in the testing process. The use of multiple views enhances the scope of the tester's view of the system and allows the tester to detect problems in the analysis. The use of screen orientation reduces the load on the tester's memory.

User interface design should be a part of the tool development process. If the process is well understood, the user interface design may be highly integrated into the development. If the process is poorly understood, a series of prototypes may be appropriate.

Testing is a relatively new area of research. The tools used in testing are changing rapidly. They provide information to the user. How is that information processed? By the user. How would they like to see it represented? The interface may show graphs or the text of files. A simple prototype that has flexibility allows further development and improvement as the tool develops. Useful information can be obscured by forcing it into an interface that doesn't consider fully the ramifications of the information on the process as a whole, the type of information being presented, the target audience, etc. The interface designer shouldn't hesitate to use a textual representation of data until a better understanding of the properties of the data and desired qualities are known. Views must be focused around the analysis-related features, or the interface will be more distracting than helpful.

The designer should consider the use of a toolkit such as SunView when designing a graphical user interface. The toolkit provides a high level language to manipulate objects used in the creation of the interface. This enables the designer to maintain a level of abstraction during the design process. The designer has to be aware of the user's needs and desires. They should work closely with the user to incorporate these needs and desires into the interface. The interface will then increase productivity instead of increasing frustration in the user.

C. RECOMMENDATIONS FOR FUTURE RESEARCH

Each user has their own idea of a "good" interface. To allow for variations in taste, the user could customize the interface with regard to size and placement. Some may feel the text requires a larger display area, while others may not want to use scrollbars, but see the entire acfg representation at once. An initialization file (.viewer) could provide unique default settings for the VIEWER interface to accommodate a variety of users. It would allow tiling of the windows and save the setup to a file so each user could adjust the subwindows to suit their individual preferences.

We designed this interface on a Sun Workstation with a monochrome display, so color was not a consideration. If color is available, the use of color in the design of user interfaces is a direction of further research. Which colors work best together as background and foreground, text representation, and window colors are all possibilities for future consideration. This use of color could include the interface outline,

backgrounds and the representation of the acfg. Nodes could be color coded based on their type (begin-end, assignment, etc.).

Further refinement of the algorithms that construct the graphical representation of the control flow graph are necessary. Representing complex graphs in the acfg window is a complex programming problem worthy of further research and development.

The acfg provides another area of research. In the prototype interface, the user can display the graph, but it is non-interactive. An enhancement allowing traversal of the graph through use of the mouse would be useful vice the button facility provided in the prototype. While the buttons are useful, and in some ways preferable to the command line, a user could move through an acfg by pointing to a location and clicking on it.

Error handling is an important principle in the design of user interfaces. The prototype has some simple error handling procedures. There is much room for improvement here. The program could, for example, check for the existence of the filename entered before running any program.

The interface does not provide on-line help. Help only exists within the tools. Further versions of the interface could provide more detailed help facilities, an on-line manual, etc.

Levels of interface support are another possible area of continued research. The program could detect whether the user is a beginner, intermediate, or expert and provide different interfaces for each level. The beginner level would provide detailed help functions, and prompt the user with detailed explanations of what the program expects at various junctures. Artificial intelligence techniques have applications here.

Other languages also could be a source for further research. SunView is a powerful toolkit, but is not the only one available for Sun Workstations. An option provided by Sun is SunNeWS (Sun Network extensible Window Systems). It uses a version of PostScript as the programming language. SunNeWS can support graphical interfaces to remote programs. There is also the X Window System that originated at MIT. There are similarities between SunView and X Windows, but X Windows also can provide an interface to programs on remote machines like SunNeWS. (Beer, p. 64)

Follow-on tools to use the results from VIEWER are an area of further research possibilities. These tools could analyze the relationships between failure regions within a single program or in several programs. This analysis may provide information on the clustering of failure regions within software. Bolchoz, 1990 contains several additional ideas for further research in the area of failure regions analysis.

APPENDIX A

This appendix contains the text from an interactive session with the EFFIGY testing tool. Italicized text indicates comments.

```
▸ effigy                                  Invoke the EFFIGY system.
 EFFIGY  READY
▸ edit absolute effigy;
                                          Invoke the CMS file editor and
                                            type-n a new file called
                                            "absolute effigy".
 NEW FILE:
▸ input
▸ absolute: proc (x,y);
▸   dcl (x,y) integer;
▸   if x<O then y = -x
▸                 else y =  x;
▸   end;
▸                                         (end of input signified by null line.)
▸ file                                    Save file permanently &
                                            go back to EFFIGY.
▸ input absolute effigy;
                                          Have EFFIGY read input from
                                            that file.
          1: ABSOLUTE: PROC(X,Y);
          2: DCL (X,Y) INTEGER;
                                          Statements are numbered by EFFIGY.
          3: IF X<O THEN Y = -X
          4:        ELSE Y =  X;
          5: END;
                                          Last line of file - back to
                                            terminal input.
▸ dcl z integer;
                                          Declare a variable in MAIN.
▸ call absolute (55,z); display z;
                                          Try a numeric execution.
 55                                       Result of display statement.
▸ call absolute (-66,z); display z;
 66
▸ in absolute; turn all on all;
```

```
                                        All tracing on in proc. "absolute".
▸ in main;                              Set back to MAIN.
▸ call absolute ("a",z);
                                        Try a symbolic input "a".
          1: ABSOLUTE: PROC (X,Y);
                                        Each statement execution is traced
                                          by printing it.
          2: DCL (X,Y) INTEGER;
          3: IF X < 0 THEN Y = -X
 ((A¬>-1))                              Evaluated result of X<0--v(B).
                                          (A ≤ -1 or A < 0)
 TYPE GO TRUE OR GO FALSE
                                        Unresolved (forking) IF--User option.
▸ before 5;                             Stop before executing statement 5.
▸ save;                                 Save the current execution state.
 STATE  1 SAVED
                                        EFFIGY calls this state 1.
▸ go true;                              Follow case where A<0.
 ((A¬>-1))                              Current pc.
 TRUE BRANCH
 Y=-A                                   Result of assignment to Y.
 STOPPED BETWEEN  3 AND  5
                                        Stopped "before 5".
▸ display variables, pc;
                                        All local values and the pc.
 IN ABSOLUTE
 X=A
 Y=-A
 ((A¬>-1))                              Current pc.
▸ restore 1;                            Return to execution state 1,
                                          and try else path.
 STATE  1 RESTORED.  IN ABSOLUTE
▸ go false;
 ((A¬>-1))                              v(X<0).
 FALSE BRANCH
    4: ELSE Y = X;
 Y=A                                    New value of Y.
 STOPPED BETWEEN  4 AND  5
                                        Before 5.
▸ display variables, pc;
 IN ABSOLUTE
 X=A
 Y=A
 ((Z¬<0))
▸ xgo;                                  Resume execution and delete breakpoint.
 BACK FROM ABSOLUTE TO MAIN
```

```
▸ display z;
 A
▸ in main;
▸ erase pc;                                  Reset pc to true.
▸ call absolute("a" - "b", z); go true;
 TYPE GO TRUE OR GO FALSE
                                             Go true above anticipates question.
▸ display z;
 -A+B
▸ edit absolute effigy;
                                             Invoke editor to change absolute.
▸ next                                       Edit command to look at line 1 of file.
▸ change /absolute/newabs/
                                             Change proc name.
▸ bottom                                     Go to end of file.
▸ up 1                                       Well not quite.
▸ input   assert(y eq abs(x));
                                             Insert a correctness specification.
▸ file newabs                                File away as newabs effigy.
                                             Go back to EFFIGY.
▸ input newabs effigy;
                                             Enter into EFFIGY.
          1: NEWABS: PROC(X,Y);
          2:    DCL (X,Y) INTEGER;
          3:    IF  X<0  THEN Y = -X
          4:            ELSE Y =  X;
          5:  ASSERT(Y EQ ABS (X));
                                             New statement.
          6:  END;
▸ erase pc;
▸ call newabs("a",z); go true;
 TYPE GO TRUE OR GO FALSE
                                             Response was anticipated on
                                               previous line.
 ((abs(A)+A =0)) :: TRUE
                                             Result of executing assert (statement 5)
                                               Of form l :: r where...
                                               l is evaluated assertion and
                                               r is result of pc ⊃ l.
▸ display z, pc;
 -A                                          Value of z
 ((A¬>-1))                                   and pc.
▸ erase pc;
▸ call newabs("a",z); go false;
 TYPE GO TRUE OR GO FALSE
                                             Try only other case.
```

```
((abs(A)-A =0)) :: TRUE
```

That also gets proved.
Have correctness proof–both paths correct.

```
A
((A¬<0))
▸erase pc;
▸input times effigy;
```

Now read in procedure times.

```
        1: TIMES:PROC(X,Y,Z);
        2:   DCL (X,Y,Z) INTEGER;
        3:   Z=0;
        4:   IF X<0 THEN
        4:   DO;
        5:     CALL ABSOLUTE(X,X);
```

Times calls absolute.

```
        6:     Y=-Y;
        7:   END;
        8: L:
        8:   IF X>0 THEN
```

It multiplies by looping add.

```
        8:   DO;
        9:     X=X-1;
        10:    Z=Z+Y;
        11:    GO TO L;
        12:  END;
        13: END;
▸ call times (3,5,z); display z;
```

Try some numbers.

```
15
call times(-3,5,z); display z;
-15
call times(-34,"b",z);  display z;
```

A mixed case–determinate control flow.

```
-34*B
▸ in times; turn all on 4 5 6 8 9 10;
▸before 13; in main;
▸ call times("a", "b", z);
```

The completely symbolic case.

```
        4: IF X < 0 THEN DO;
((A¬>-1))
TYPE GO TRUE OR GO FALSE
▸ save;
STATE   2 SAVED
```

```
▸ go true;
((A¬>-1))
TRUE BRANCH
        5: CALL ABSOLUTE(X,X);
                                        Executed a resolved IF in absolute.
                                        Knows A≤-1.
        6: Y = -Y;
Y=-B
        8: L: IF X > 0 THEN DO;
((A¬>-1))
TRUE BRANCH
                                        Another resolved IF.
                                        A≤-1 so -A>0.
        9: X = X - 1;
X=A-1
    10: Z = Z + Y;
Z=-B
        8: L: IF X > 0 THEN DO;
((A¬>-2))
TYPE GO TRUE OR GO FALSE
▸ go true;
                                        Loop around.
((A¬>-2))
TRUE BRANCH
        9: X = X - 1;
X=-A-2
        10: Z = Z + Y;
Z=-2*B
        8: L: IF X > 0 THEN DO;
((Z¬>-3))
TYPE GO TRUE OR GO FALSE
▸go false;                              Now go out to end of proc.
((A¬>-3))
FALSE BRANCH
STOPPED BETWEEN   8 AND   13
                                        Breakpoint at end of proc.
▸ display variables, pc;
IN TIMES
X=-A-2
Y=-B
Z=-2*B
((A =-2))                               Path choices determine A=-2.
▸ restore 2;
STATE   2 RESTORED.  IN TIMES
```

```
▸ go false;                          Try another case.
 ((A¬>-1))
 FALSE BRANCH
         8: L: IF X > 0 THEN DO;
 ((A¬<1))
 TYPE GO TRUE OR GO FALSE
▸ assume("a">4);
                                     Add this assumption to the pc.
▸ go;                                Now retry the IF with new pc.
 ((A¬<1))
 TRUE BRANCH                         New pc resolves it.
         9: X = X - 1;
 X=A-2
         10: Z = Z + Y;
 Z=2*B
         8: L: IF X > 0 THEN DO;
 ((A¬<3))
 TRUE BRANCH
         9: X = X - 1;
 X=A-3
    10: Z = Z + Y;
 Z=3*B
         8: L: IF X > 0 THEN DO;
 ((A¬<4))
 TRUE BRANCH
         9: X = X - 1;
 X=A-4
    10: Z = Z + Y;
 Z=4*B
         8: L: IF X > 0 THEN DO;
 ((A¬<5))
 TRUE BRANCH
         9: X = X - 1;
 X=A-5
    10: Z = Z + Y;
 Z=5*B
         8: L: IF X > 0 THEN DO;
 ((A¬<6))
 TYPE GO TRUE OR GO FALSE
                                     Unresolved when X gets to A-5.
▸ go false;                          Leave loop.
 ((A¬<10))
 FALSE BRANCH
 STOPPED BETWEEN   8 AND   13
```

```
▸ display variables, pc;
 IN TIMES
 X=A-5
 Y=B
 Z=5*B
 ((A=5))
▸ restore 2;                          Go back and try another case.
 STATE   2 RESTORED.  IN TIMES
▸assume ( "a" eq "b" & "b" eq 2);
                                      Indirectly assume A is 2.
▸ go;                                 Does that assume resolve the if?
 ((A¬>-1))
 FALSE BRANCH                         Yes it does.
         8: L: IF X > 0 THEN DO;
 ((A¬<1))
 TRUE BRANCH                          This one resolved too.
         9: X = X - 1;
 X=A-1
    10: Z = Z + Y;
 Z=B
         8: L: IF X > 0 THEN DO;
 ((A¬<2))
 TRUE BRANCH                          This one resolved too.
         9: X = X - 1;
 X=A-2
    10: Z = Z + Y;
 Z=2*B
         8: L: IF X > 0 THEN DO;
 ((A¬<3))
 FALSE BRANCH
 STOPPED BETWEEN   8 AND   13
▸display variables, pc;
 IN TIMES
 X=A-2
 Y=B
 Z=2*B                                Result still in symbolic terms.
 ((Z-B =0&B =2))
▸assert(a eq 4);
                                      Does it know Z is really 4.
 ((B =2)) :: TRUE
                                      Yes.
▸go;                                 Go on out of times.
▸
 IN MAIN                              Response to previous null line.
▸next                                 In editor.
```

```
▸input    assume(x eq "x0" & y eq "y0");
                                        Insert correctness specifications.
▸bottom
▸up 1
▸input assert (z eq "x0" * "y0");
▸file                                   Replace original procedure
                                          and go back to EFFIGY.
▸erase times;
                                        Can't have two times routines.
▸erase pc;
▸input times effigy;
                                        Input from times file.
          1: TIMES:PROC(X,Y,Z);
          2:    ASSUME(X EQ "X0" & Y EQ "Y0");
                                        Used to name input values.
          3:    DCL (X,Y,Z) INTEGER;
          4:    Z=0;
          5:    IF X<0 THEN
          5:    DO;
          6:       CALL ABSOLUTE(X,X);
          7:       Y=-Y;
          8:    END;
          9: L:
          9:    IF X>0 THEN
          9:    DO;
          10:      X=X-1;
          11:      Z=Z+Y;
          12:      GO TO L;
          13:   END;
          14: ASSERT(Z EQ "X0" * "Y0");
                                        Relate input values to output.
          15: END;
▸in times; turn all on 5 9 14;
                                        Selectively trace.
▸ in main;
▸assume("a">4 & "a"<5);
                                        No integer between 4 and 5.
 CONTRADICTING ASSUMPTION.  IGNORED.
▸assume("a">4 & "a"<7);
                                        How about A is 5 or 6.
▸call times("a", "b",z);
          5:  IF X < 0 THEN DO;
 ((A¬>-1))
FALSE BRANCH
                                        For 5 and 6 X>0.
```

```
 9: L: IF X > O THEN DO;
((A¬<1))
TRUE BRANCH
```

For 5 and 6 loop some too.

```
 9: L: IF X > O THEN DO;
((A¬<2))
TRUE BRANCH
 9: L: IF X > O THEN DO;
((A¬<3))
TRUE BRANCH
 9: L: IF X > O THEN DO;
((A¬<4))
TRUE BRANCH
 9: L: IF X > O THEN DO;
((A¬<5))
TRUE BRANCH
 9: L: IF X > O THEN DO;
((A¬<6))
TYPE GO TRUE OR GO FALSE
```

Now must decide 5 or 6.
Pick 6.

```
GO TRUE
((A¬<6))
TRUE BRANCH
 9: L: IF X > O THEN DO;
((A¬<7))
FALSE BRANCH
```

Known not > 6.

```
 14: ASSERT( Z EQ XO * YO);
((6*B-O*YO=O)) :: TRUE
```

Results check by assert-O.K.

```
▸display pc;
```

What is the pc?

```
((A=6 & A-O=O & B-O=O))
```

Relates the symbolic inputs to the names given to inputs by assume in the procedure.

```
▸display variables;
```

MAIN has variables and values too.

```
IN MAIN
ABSOLUTE=PROC
```

Value "PROC" means it is a procedure.

```
Z=6*B
NEWABS=PROC
TIMES=PROC
▸quit
```

Leave EFFIGY system.

APPENDIX B

This appendix contai. ; the script from an interactive session with the ASSET testing tool.

```
Script started on Wed Jun 13 12:08:55 1990
% asset
==========================================
| ASSET was developed with the support of the National Science Foundation |
| under grant CCR-8501614 and the Office of Naval Research under contract |
| N00014-85-K-414.                                                        |
==========================================

Welcome to ASSET. For help type "help."


Enter relative pathname of initial default directory.
>>: ..

>>>: begin
Enter name of subject procedure file.
>>: abel.p
Separate Compilation? (Y/N) [N]
>>: n
Enter the name of the procedure to be instrumented.
If you would like to be prompted with the names of
the procedures in the subject program, just hit carriage return.
>>:

==> Should driver
be instrumented for testing? (Y/N)
>>: n
==> Should PositionUnit
be instrumented for testing? (Y/N)
>>: y

>>>: sel

No criterion has been selected yet.
```

SELECT A CRITERION

A. All-defs
B. All-c-uses
C. All-p-uses
D. All-c-uses/some-p-uses
E. All-p-uses/some-c-uses
F. All-uses
G. All-du-paths
H. All-edges

Enter letter representing the selected criterion
>>: c
Criterion is All-p-uses.

>>>: find

>>>: compile
Compilation begins ...
Done, and successful.

>>>: run
Command line arguments? (Y/N) [Y]
>>: n

Executing modified subject program ...

How many units?1
How many elements in unit 1?2
What x y position for unit 1? (no commas!)0.0 0.0
How many elements/row in unit 1?1
What row column separation for unit 1? (no commas!)1.0 1.0
What Endurance for unit 1 element 1?1
What Endurance for unit 1 element 2?1
Position[1, 1]=(0.0000, 0.5000)
Position[1, 2]=(0.0000, -0.5000)
Enter another data set?y
How many units?1
How many elements in unit 1?4
What x y position for unit 1? (no commas!)0.0 0.0
How many elements/row in unit 1?2
What row column separation for unit 1? (no commas!)1.0 1.0

```
What Endurance for unit  1 element  1?-1
What Endurance for unit  1 element  2?1
What Endurance for unit  1 element  3?-1
What Endurance for unit  1 element  4?1
Position[ 1, 1]=(  0.0000,  0.5000)
Position[ 1, 2]=( -0.5000,  0.5000)
Position[ 1, 3]=(  0.0000,  0.0000)
Position[ 1, 4]=(  0.5000,  0.5000)
Enter another data set?y
How many units?1
How many elements in unit  1?4
What x y position for unit  1? (no commas!)0.0 0.0
How many elements/row in unit  1?2
What row column separation for unit  1? (no commas!)1.0 1.0
What Endurance for unit  1 element  1?1
What Endurance for unit  1 element  2?1
What Endurance for unit  1 element  3?0
What Endurance for unit  1 element  4?1
Position[ 1, 1]=( -0.5000,  0.5000)
Position[ 1, 2]=(  0.5000,  0.5000)
Position[ 1, 3]=(  0.0000,  0.0000)
Position[ 1, 4]=( -0.5000, -0.5000)
Enter another data set?n
Do you want to run the subject program
on some additional test data? (Y/N) [N]
>>: n

>>>: d^Hc\^Hhc^Heh^Hck
ALL-P-USES
Still need to exercise all of the following of def-clear paths:

with respect to          from      to
Elements                   1     ( 2,  3)
Elements                   1     ( 14, 15)
Elements                   1     ( 15, 29)
Elements                   1     ( 15, 16)
Elements                   1     ( 17, 22)
Elements                   1     ( 17, 18)
Elements                   1     ( 23, 28)
Elements                   1     ( 23, 24)
Elements                   1     ( 30, 32)
Elements                   1     ( 30, 31)
Elements                   1     ( 32, 34)
```

Elements	1	(32, 33)
Elements	1	(36, 41)
Elements	1	(36, 37)
Elements	1	(42, 47)
Elements	1	(42, 43)
GRow	1	(2, 3)
Endurance	1	(18, 20)
Endurance	1	(18, 19)
Endurance	1	(24, 26)
Endurance	1	(24, 25)
Endurance	1	(30, 32)
Endurance	1	(30, 31)
Endurance	1	(37, 39)
Endurance	1	(37, 38)
Endurance	1	(43, 45)
Endurance	1	(43, 44)
Unit	1	(2, 3)
Unit	1	(6, 14)
Unit	1	(8, 13)
Unit	1	(14, 49)
Unit	1	(14, 15)
Unit	1	(15, 29)
Unit	1	(15, 16)
Unit	1	(17, 22)
Unit	1	(17, 18)
Unit	1	(18, 20)
Unit	1	(18, 19)
Unit	1	(23, 28)
Unit	1	(23, 24)
Unit	1	(24, 26)
Unit	1	(24, 25)
Unit	1	(30, 32)
Unit	1	(30, 31)
Unit	1	(32, 34)
Unit	1	(32, 33)
Unit	1	(36, 41)
Unit	1	(36, 37)
Unit	1	(37, 39)
Unit	1	(37, 38)
Unit	1	(42, 47)
Unit	1	(42, 43)
Unit	1	(43, 45)
Unit	1	(43, 44)

Squad	5	(6, 14)
Squad	5	(8, 13)
Squad	5	(14, 49)
Squad	5	(14, 15)
Squad	5	(17, 22)
Squad	5	(17, 18)
Squad	5	(23, 28)
Squad	5	(23, 24)
num	5	(15, 29)
num	5	(15, 16)
TotalSq	5	(6, 14)
TotalSq	5	(14, 49)
TotalSq	5	(14, 15)
TotalSq	5	(17, 22)
TotalSq	5	(17, 18)
TotalSq	5	(18, 20)
TotalSq	5	(18, 19)
TotalSq	5	(23, 28)
TotalSq	5	(23, 24)
TotalSq	5	(24, 26)
TotalSq	5	(24, 25)
TotalSq	5	(30, 32)
TotalSq	5	(30, 31)
TotalSq	5	(32, 34)
TotalSq	5	(32, 33)
row	7	(8, 13)
Unit	10	(14, 15)
Unit	10	(15, 29)
Unit	10	(15, 16)
Unit	10	(17, 22)
Unit	10	(17, 18)
Unit	10	(18, 20)
Unit	10	(18, 19)
Unit	10	(23, 28)
Unit	10	(23, 24)
Unit	10	(24, 26)
Unit	10	(24, 25)
Unit	10	(30, 32)
Unit	10	(30, 31)
Unit	10	(32, 34)
Unit	10	(32, 33)
Unit	10	(36, 41)
Unit	10	(36, 37)

Unit	10	(37, 39)
Unit	10	(37, 38)
Unit	10	(42, 47)
Unit	10	(42, 43)
Unit	10	(43, 45)
Unit	10	(43, 44)
Squad	10	(14, 15)
Squad	10	(17, 22)
Squad	10	(17, 18)
Squad	10	(23, 28)
Squad	10	(23, 24)
TotalSq	12	(14, 15)
TotalSq	12	(17, 22)
TotalSq	12	(17, 18)
TotalSq	12	(18, 20)
TotalSq	12	(18, 19)
TotalSq	12	(23, 28)
TotalSq	12	(23, 24)
TotalSq	12	(24, 26)
TotalSq	12	(24, 25)
TotalSq	12	(30, 32)
TotalSq	12	(30, 31)
TotalSq	12	(32, 34)
TotalSq	12	(32, 33)
tempcount	16	(17, 22)
tempcount	16	(17, 18)
Unit	19	(17, 22)
Unit	19	(17, 18)
Unit	19	(18, 20)
Unit	19	(18, 19)
Unit	19	(23, 28)
Unit	19	(23, 24)
Unit	19	(24, 26)
Unit	19	(24, 25)
Squad	19	(17, 22)
Squad	19	(17, 18)
Squad	19	(23, 28)
Squad	19	(23, 24)
TotalSq	21	(17, 22)
TotalSq	21	(17, 18)
TotalSq	21	(18, 20)
TotalSq	21	(18, 19)
TotalSq	21	(23, 28)

TotalSq	21	(23, 24)
TotalSq	21	(24, 26)
TotalSq	21	(24, 25)
Unit	25	(23, 28)
Unit	25	(23, 24)
Unit	25	(24, 26)
Unit	25	(24, 25)
Squad	25	(23, 28)
Squad	25	(23, 24)
TotalSq	27	(23, 28)
TotalSq	27	(23, 24)
TotalSq	27	(24, 26)
TotalSq	27	(24, 25)
TotalSq	31	(30, 32)
TotalSq	31	(30, 31)
TotalSq	31	(32, 34)
TotalSq	31	(32, 33)
Unit	33	(36, 41)
Unit	33	(36, 37)
Unit	33	(37, 39)
Unit	33	(37, 38)
Unit	33	(42, 47)
Unit	33	(42, 43)
Unit	33	(43, 45)
Unit	33	(43, 44)
Squad	35	(36, 41)
Squad	35	(36, 37)
Squad	35	(42, 47)
Squad	35	(42, 43)
tempcount	35	(36, 41)
tempcount	35	(36, 37)
TotalSq	35	(36, 41)
TotalSq	35	(36, 37)
TotalSq	35	(37, 39)
TotalSq	35	(37, 38)
TotalSq	35	(42, 47)
TotalSq	35	(42, 43)
TotalSq	35	(43, 45)
TotalSq	35	(43, 44)
Unit	38	(36, 41)
Unit	38	(36, 37)
Unit	38	(37, 39)
Unit	38	(37, 38)

```
Unit          38    ( 42, 47)
Unit          38    ( 42, 43)
Unit          38    ( 43, 45)
Unit          38    ( 43, 44)
Squad         38    ( 36, 41)
Squad         38    ( 36, 37)
Squad         38    ( 42, 47)
Squad         38    ( 42, 43)
TotalSq       40    ( 36, 41)
TotalSq       40    ( 36, 37)
TotalSq       40    ( 37, 39)
TotalSq       40    ( 37, 38)
TotalSq       40    ( 42, 47)
TotalSq       40    ( 42, 43)
TotalSq       40    ( 43, 45)
TotalSq       40    ( 43, 44)
Unit          44    ( 42, 47)
Unit          44    ( 42, 43)
Unit          44    ( 43, 45)
Unit          44    ( 43, 44)
Squad         44    ( 42, 47)
Squad         44    ( 42, 43)
TotalSq       46    ( 42, 47)
TotalSq       46    ( 42, 43)
TotalSq       46    ( 43, 45)
TotalSq       46    ( 43, 44)

To look at these again use the command 'view results'.

>>>: exit
%
script done on Wed Jun 13 12:32:51 1990
```

APPENDIX C

This appendix contains the script from an interactive session with the Mothra testing tool.

```
Script started on Sun Jun 17 18:51:09 1990
% mothra

+----------------------------------------+
|                                        |
|                MOTHRA                  |
|                                        |
|        Fortran-77 Mutation System      |
|                                        |
|             Standard Interface         |
|                Version 1.5             |
|                                        |
+----------------------------------------+

Please enter the experiment name: [?](?) jms2
Ready to continue a mutation experiment...

    ******************
     Mothra Main Menu
    ******************
(1) Test Case Management              ==>
(2) Execution Management              ==>
(3) GO!!!
(4) Status Review                     ==>
(5) View Source Code
(6) Exit Mothra
? 4
```

```
    ====================
     Status Review Menu
    ====================
(1) View Test Status
(2) View Status Summary
(3) View Mutant Information          ==>
? 1


Working...
24;1H?1h"/tmp/mut026991" [Read only] 34 lines, 2079 characters ;H2J
PERCENTAGE OF VERIFIED TEST CASES  0.0%H

Type    Genr'd Enabled  Live   %Live   Equiv   Dead   %Enable  %EScore %GScore
----    ------ -------  ----   -----   -----   ----   -------  ------- -------
abs        12      12     5    41.7       0      7     100.0     58.3     58.3
aor         6       6     0     0.0       0      6     100.0    100.0    100.0
crp         6       6     1    16.7       0      5     100.0     83.3     83.3
csr         4       4     0     0.0       0      4     100.0    100.0    100.0
der         1       1     0     0.0       0      1     100.0    100.0    100.0
san         3       3     0     0.0       0      3     100.0    100.0    100.0
scr         8       8     3    37.5       0      5     100.0     62.5     62.5
sdl         3       3     1    33.3       0      2     100.0     66.7     66.7
svr        12      12     1     8.3       0     11     100.0     91.7     91.7
uoi         9       9     0     0.0       0      9     100.0    100.0    100.0
----    ------ -------  ----   -----   -----   ----   -------  ------- -------
Totals     64      64    11    17.2       0     53     100.0     82.8     82.8

Class   Genr'd Enabled  Live   %Live   Equiv   Dead   %Enable  %EScore %GScore
-----   ------ -------  ----   -----   -----   ----   -------  ------- -------
ary         0       0     0     0.0       0      0       0.0      0.0      0.0
con         8       8     3    37.5       0      5     100.0     62.5     62.5
ctl         1       1     0     0.0       0      1     100.0    100.0    100.0
dmn        27      27     6    22.2       0     21     100.0     77.8     77.8H
 24;1H"/tmp/mut026991" [Read only] 34 lines, 2079 charactersH          H
```

```
KAopm      6     6    0    0.0     0     6   100.0  100.0   100.0
Aprd       0     0    0    0.0     0     0    0.0    0.0     0.0
Ascl      16    16    1    6.3     0    15   100.0   93.8    93.8
Astm       6     6    1   16.7     0     5   100.0   83.3    83.3
A
ASuperCl      Genr'd Enabled  Live   %Live   Equiv   Dead   %Enable  %EScore
%GScore
A
A-------      ------ -------  ----   -----   -----   ----   -------  -------  -------
A
Aall      64    64   11   17.2     0    53   100.0   82.8    82.8
Acca      30    30    4   13.3     0    26   100.0   86.7    86.7
Apda      27    27    6   22.2     0    21   100.0   77.8    77.8
Asal       7     7    1   14.3     0     6   100.0   85.7    85.7
?^HA^HK7mNo previous regular expressionm
K?1l

   ====================
    Status Review Menu
   ====================
(1) View Test Status
(2) View Status Summary
(3) View Mutant Information          ==>
? 2


Working...
24;1H?1h"/tmp/mut026991" [Read only] 21 lines, 632 characters ;H2J
PERCENTAGE OF VERIFIED TEST CASES   0.0%4;21HSummary for Experiment
jms2
                 ---------------------------


  Number of Test Cases            =   30
  Number of Test Cases Verified   =    0

  Number of Mutants Enabled       =   64
  Number of Mutants Generated     =   64
  Number of Live Enabled Mutants  =   11
  Number of Dead Enabled Mutants  =   53
  Number of Equiv Enabled Mutants =    0
```

```
 % Non-equivalent Enabled Mutants Killed      =    82.81
 % Non-equivalent Generated Mutants killed    =    82.81
 # of Mutants Types Remaining to be Generated = 0 out of 22

~
~H24;1H"/tmp/mut026991" [Read only] 21 lines, 632 charactersH          24;1HK?11

   ========================
    Status Review Menu
   ========================
(1) View Test Status
(2) View Status Summary
(3) View Mutant Information          ==>
? 3


   ============================
    Mutant Information Menu
   ============================
(1) View Inline Mutant Information
(2) Browse/Equivalence Mutants
(3) Histograms
? 3


Mutant Types: [[+,-]type,?](all) all
Histogram Parameters: [[+,-]ides?](live,equiv)


Working...
24;1H?1h"/tmp/mut026991" [Read only] 17 lines, 440 characters ;H2J
```

```
                PERCENTAGE OF VERIFIED TEST CASES  0.0%

                  Histogram of Live/Equivalent Mutants
------------------------------------------------------------------

Frequency    3   3   5

   5                 *
   4                 *
   3         *   *   *
   2         *   *   *
   1         *   *   *
------------------------------------------------------------------
Statement    1   2   3
~
~
~
~
~
~H24;1H"/tmp/mut026991" [Read only] 17 lines, 440 charactersH24;1HK?1l

    ==========================
     Mutant Information Menu
    ==========================
(1) View Inline Mutant Information
(2) Browse/Equivalence Mutants
(3) Histograms
? 1


Mutant Types: [[+,-]type,#,?](all)
Inline Parameters: [[+,-]lde?](live,equiv)


Working...
24;1H?1h"/tmp/mut026991" 34 lines, 971 characters ;H2J
------------------------------------------------------------------------------
SUM
------------------------------------------------------------------------------
# Type Mutant  #
```

```
      C     Sum -
      C     Computes the sum of N integers


            PROGRAM SUM

            INTEGER N, RESULT

      C     Loop through values and compute sum
1                   RESULT = 0
# sdl    41         #     CONTINUE
# svr    45         #     I = 0
# scr    34         #     RESULT = I
2                   DO 10 I = 1, N, 1
# crp    21         #     DO 10 I = 0, N, 1
H24;1H"/tmp/mut026991" 34 lines, 971 charactersH

KA# scr          36       #     DO 10 I = RESULT, N, 1
A# scr    37      #     DO 10 I = I, N, 1
A3                  RESULT = RESULT + I
A# abs     4      #     RESULT = ABS(RESULT) + I
A# abs     7      #     RESULT = RESULT + ABS(I)
A# abs     9      #     RESULT = RESULT + ZPUSH(I)
A# abs    10      #     RESULT = ABS(RESULT + I)
A# abs    12      #     RESULT = ZPUSH(RESULT + I)
A
A4             10   CONTINUE
A5                  END
?11

   ============================
    Mutant Information Menu
   ============================
(1) View Inline Mutant Information
(2) Browse/Equivalence Mutants
(3) Histograms
? 2


Mutant Types: [[+,-]type,#,?](all)
Browse Parameters: [[+,-]lder?](live,equiv,sorted)
```

```
Working...
Note that no equivalent mutants currently exist.
Sorting...
Browsing 11 mutants...
Filling the browse buffer...

Mutant number 45 is live:

1                RESULT = 0
# svr    45      #     I = 0

Command: [ln#euq?](next)


Mutant number 41 is live:

1                RESULT = 0
# sdl    41      #     CONTINUE

Command: [ln#euq?](next)


Mutant number 34 is live:

1                RESULT = 0
# scr    34      #     RESULT = I

Command: [ln#euq?](next)


Mutant number 21 is live:

2                DO 10 I = 1, N, 1
# crp    21      #     DO 10 I = 0, N, 1

Command: [ln#euq?](next)

Mutant number 36 is live:

2                DO 10 I = 1, N, 1
# scr    36      #     DO 10 I = RESULT, N, 1

Command: [ln#euq?](next)
```

Mutant number 37 is live:

```
2                DO 10 I = 1, N, 1
# scr    37      #     DO 10 I = I, N, 1
```

Command: [ln#euq?](next)

Mutant number 4 is live:

```
3                  RESULT = RESULT + I
# abs     4        #     RESULT = ABS(RESULT) + I
```

Command: [ln#euq?](next)

Mutant number 9 is live:

```
3                  RESULT = RESULT + I
# abs     9        #     RESULT = RESULT + ZPUSH(I)
```

Command: [ln#euq?](next)

Mutant number 7 is live:

```
3                  RESULT = RESULT + I
# abs     7        #     RESULT = RESULT + ABS(I)
```

Command: [ln#euq?](next)

Mutant number 10 is live:

```
3                  RESULT = RESULT + I
# abs     10       #     RESULT = ABS(RESULT + I)
```

Command: [ln#euq?](next)
Refilling the browse buffer...

```
Mutant number 12 is live:

3                    RESULT = RESULT + I
# abs    12      #     RESULT = ZPUSH(RESULT + I)

Command: [ln#euq?](next)


Mutant number 45 is live:

1                    RESULT = 0
# svr    45      #     I = 0

Command: [ln#euq?](next)


Mutant number 41 is live:

1                    RESULT = 0
# sdl    41      #     CONTINUE

Command: [ln#euq?](next)


Mutant number 34 is live:

1                    RESULT = 0
# scr    34      #     RESULT = I

Command: [ln#euq?](next)


Mutant number 21 is live:

2                    DO 10 I = 1, N, 1
# crp    21      #     DO 10 I = 0, N, 1

Command: [ln#euq?](next)

Mutant number 36 is live:

2                    DO 10 I = 1, N, 1
# scr    36      #     DO 10 I = RESULT, N, 1
```

```
Command: [ln#euq?](next)


Mutant number 37 is live:

2                DO 10 I = 1, N, 1
# scr    37      #     DO 10 I = I, N, 1

Command: [ln#euq?](next)


Mutant number 4 is live:

3                  RESULT = RESULT + I
# abs     4       #     RESULT = ABS(RESULT) + I

Command: [ln#euq?](next)


Mutant number 9 is live:

3                  RESULT = RESULT + I
# abs     9       #     RESULT = RESULT + ZPUSH(I)

Command: [ln#euq?](next)


Mutant number 7 is live:

3                  RESULT = RESULT + I
# abs     7       #     RESULT = RESULT + ABS(I)

Command: [ln#euq?](next)


Mutant number 10 is live:

3                  RESULT = RESULT + I
# abs     10      #     RESULT = ABS(RESULT + I)

Command: [ln#euq?](next)
```

Mutant number 12 is live:

```
3                    RESULT = RESULT + I
# abs    12       #      RESULT = ZPUSH(RESULT + I)
```

Command: [ln#euq?](next)

Mutant number 45 is live:

```
1                    RESULT = C
# svr    45       #      I = 0
```

Command: [ln#euq?](next)

Mutant number 41 is live:

```
1                    RESULT = 0
# sdl    41       #      CONTINUE
```

Command: [ln#euq?](next)

Mutant number 34 is live:

```
1                    RESULT = 0
# scr    34       #      RESULT = I
```

Command: [ln#euq?](next)

Mutant number 21 is live:

```
2                    DO 10 I = 1, N, 1
# crp    21       #      DO 10 I = 0, N, 1
```

Command: [ln#euq?](next)

Mutant number 36 is live:

```
2                    DO 10 I = 1, N, 1
```

```
# scr    36       #      DO 10 I = RESULT, N, 1
```

Command: [ln#euq?](next)

Mutant number 37 is live:

```
2                 DO 10 I = 1, N, 1
# scr    37       #      DO 10 I = I, N, 1
```

Command: [ln#euq?](next)

Mutant number 4 is live:

```
3                   RESULT = RESULT + I
# abs    4        #      RESULT = ABS(RESULT) + I
```

Command: [ln#euq?](next)

Mutant number 9 is live:

```
3                   RESULT = RESULT + I
# abs    9        #      RESULT = RESULT + ZPUSH(I)
```

Command: [ln#euq?](next)

Mutant number 7 is live:

```
3                   RESULT = RESULT + I
# abs    7        #      RESULT = RESULT + ABS(I)
```

Command: [ln#euq?](next)

Mutant number 10 is live:

```
3                   RESULT = RESULT + I
# abs    10       #      RESULT = ABS(RESULT + I)
```

Command: [ln#euq?](next)

Mutant number 12 is live:

```
3                    RESULT = RESULT + I
# abs    12      #     RESULT = ZPUSH(RESULT + I)
```

Command: [ln#euq?](next)

Mutant number 45 is live:

```
1                    RESULT = 0
# svr    45      #     I = 0
```

Command: [ln#euq?](next)

Mutant number 41 is live:

```
1                    RESULT = 0
# sdl    41      #     CONTINUE
```

Command: [ln#euq?](next)

Mutant number 34 is live:

```
1                    RESULT = 0
# scr    34      #     RESULT = I
```

Command: [ln#euq?](next)

Mutant number 21 is live:

```
2                    DO 10 I = 1, N, 1
# crp    21      #     DO 10 I = 0, N, 1
```

Command: [ln#euq?](next)

Mutant number 36 is live:

```
2                    DO 10 I = 1, N, 1
```

```
# scr    36       #      DO 10 I = RESULT, N, 1
```

Command: [ln#euq?](next)

Mutant number 37 is live:

```
2                 DO 10 I = 1, N, 1
# scr    37       #      DO 10 I = I, N, 1
```

Command: [ln#euq?](next)

Mutant number 4 is live:

```
3                   RESULT = RESULT + I
# abs    4        #      RESULT = ABS(RESULT) + I
```

Command: [ln#euq?](next)

Mutant number 9 is live:

```
3                   RESULT = RESULT + I
# abs    9        #      RESULT = RESULT + ZPUSH(I)
```

Command: [ln#euq?](next)

Mutant number 7 is live:

```
3                   RESULT = RESULT + I
# abs    7        #      RESULT = RESULT + ABS(I)
```

Command: [ln#euq?](next)

Mutant number 10 is live:

```
3                   RESULT = RESULT + I
# abs    10       #      RESULT = ABS(RESULT + I)
```

Command: [ln#euq?](next)

Mutant number 12 is live:

```
3                    RESULT = RESULT + I
# abs    12      #     RESULT = ZPUSH(RESULT + I)
```

Command: [ln#euq?](next)

Mutant number 45 is live:

```
1                    RESULT = 0
# svr    45      #     I = 0
```

Command: [ln#euq?](next)

Mutant number 41 is live:

```
1                    RESULT = 0
# sdl    41      #     CONTINUE
```

Command: [ln#euq?](next)

Mutant number 34 is live:

```
1                    RESULT = 0
# scr    34      #     RESULT = I
```

Command: [ln#euq?](next)

Mutant number 21 is live:

```
2                    DO 10 I = 1, N, 1
# crp    21      #     DO 10 I = 0, N, 1
```

Command: [ln#euq?](next)

Mutant number 36 is live:

```
2                    DO 10 I = 1, N, 1
# scr    36      #     DO 10 I = RESULT, N, 1
```

Command: [ln#euq?](next)

Mutant number 37 is live:

```
2                DO 10 I = 1, N, 1
# scr    37      #     DO 10 I = I, N, 1
```

Command: [ln#euq?](next)

Mutant number 4 is live:

```
3                  RESULT = RESULT + I
# abs    4        #     RESULT = ABS(RESULT) + I
```

Command: [ln#euq?](next)

Mutant number 9 is live:

```
3                  RESULT = RESULT + I
# abs    9        #     RESULT = RESULT + ZPUSH(I)
```

Command: [ln#euq?](next)

Mutant number 7 is live:

```
3                  RESULT = RESULT + I
# abs    7        #     RESULT = RESULT + ABS(I)
```

Command: [ln#euq?](next)

Mutant number 10 is live:

```
3                  RESULT = RESULT + I
# abs    10       #     RESULT = ABS(RESULT + I)
```

Command: [ln#euq?](next)

Mutant number 12 is live:

```
3                    RESULT = RESULT + I
# abs    12      #     RESULT = ZPUSH(RESULT + I)

Command: [ln#euq?](next)


Mutant number 45 is live:

1                    RESULT = 0
# svr    45      #     I = 0

Command: [ln#euq?](next)

Mutant number 41 is live:

1                    RESULT = 0
# sdl    41      #     CONTINUE

Command: [ln#euq?](next) q


   ============================
    Mutant Information Menu
   ============================
(1) View Inline Mutant Information
(2) Browse/Equivalence Mutants
(3) Histograms
? .


   ******************
    Mothra Main Menu
   ******************
(1) Test Case Management          ==>
(2) Execution Management          ==>
(3) GO!!!
(4) Status Review                 ==>
(5) View Source Code
(6) Exit Mothra
? 5


Working...
```

```
24;1H?1h"/tmp/mut026991" [Read only] 21 lines, 350 characters ;H2J
--------------------------------------------------------------------------------
SUM


      C     Sum -
      C     Computes the sum of N integers


            PROGRAM SUM

            INTEGER N, RESULT

      C     Loop through values and compute sum
1                 RESULT = 0
2                 DO 10 I = 1, N, 1
3                   RESULT = RESULT + I

4              10   CONTINUE
5                 END
~
~H24;1H"/tmp/mut026991" [Read only] 21 lines, 350 charactersH24;1HK?11

   ********************
    Mothra Main Menu
   ********************
(1) Test Case Management            ==>
(2) Execution Management            ==>
(3) GO!!!
(4) Status Review                ==>
(5) View Source Code
(6) Exit Mothra
? 6


No test cases have been deleted.
Saving experiment jms2...
Exiting mothra... bye.
% exit
%
script done on Sun Jun 17 18:54:46 1990
```

APPENDIX D

This appendix contains source code for the programs used in VIEWER. Each portion of code contains a short header with the title and a description of the code.

```
/* Makefile */

CC = cc

all: walkwin viewwin faltwin

walkwin: walkwin.c ablock.o acfg.o walker.h acfg.h ablock.h
    $(CC) -DHIGHRES -DDEBUG -DREADACFG -DACFGGRAPH -g -o walkwin
walkwin.c ablock.o acfg.o -lsuntool -lpixrect -lsunwindow -lm

viewwin: viewwin.c
    $(CC) -g -o viewwin viewwin.c -lsuntool -lpixrect -lsunwindow

faltwin: faltwin.c
    $(CC) -DHIGHRES -DDEBUG -DREADACFG -DACFGGRAPH -g -o faltwin
faltwin.c ablock.o acfg.o -lsuntool -lpixrect -lsunwindow -lm

spacewin: spacewin.c
    $(CC) -g -o spacewin spacewin.c -lsuntool -lpixrect -lsunwindow

ablock.o: ablock.c walker.h acfg.h
    $(CC) -c -g ablock.c

acfg.o: acfg.c walker.h
    $(CC) -c -g acfg.c
```

```
/* TITLE                : viewwin.c
 * AUTHORS              : Vicki Abel and Medio Monti
 * DATE                 : 15 September 90
 * REVISED              : 29 October 90
 * SYSTEM               : NPS SUN TAURUS
 * LANGUAGE             : SunView and C
 * COMPILER             : Unix cc
 * DESCRIPTION          : This program creates the viewer frame. This frame
 * consists of a Tty subwindow and a control panel. The user enters a filename to be
 * processed. There are buttons to call REACHER on the file, as well as the
 * interfaces REACHWIN, FALTWIN, and SPACEWIN.
 */

/* SunView header files needed for the program */
#include <suntool/sunview.h>
#include <suntool/tty.h>
#include <suntool/icon.h>
#include <suntool/scrollbar.h>
#include <suntool/panel.h>
#include <suntool/alert.h>
#include <stdio.h>

/* type declarations */
Frame           base_frame;
Tty             tty_sw;
Panel           panel;
Panel_item      filename_item;      /* This is for the filename input */
Icon            viewer_icon;


/* Default window sizes for HIGHRES SCREEN */
#define TTYWINHEIGHT        794
#define TTYWINWIDTH         511
#define PANELWINHEIGHT      100
#define PANELWINWIDTH       511
#define PANELWINX           0

/* These are notify procedures for the buttons */
static void call_walker();
static void call_faltwin();
static void call_spacewin();
static void file_proc();
```

```
/* loading icon image into array */
static short icon_image[] = {
#include "viewer.icon"
};

/* SunView macro for importing an icon into the program */
mpr_static(icon_pixrect, 64, 64, 1, icon_image);

/* Main */

main(argc, argv)
int       argc;
char      *argv[];
{
    /* file_name = argv[1];  not used */
    viewer_icon = icon_create(ICON_IMAGE, &icon_pixrect, 0);


    /* this is the base frame for the application */
    base_frame = window_create(
        NULL, FRAME,
        WIN_X, 0,  /* sets x position relative to owner */
        WIN_Y, 205,    /* sets y position relative to owner */
        FRAME_LABEL, "Viewer 1.0",  /* frame label */
        FRAME_ICON, viewer_icon,      /* icon used */
        FRAME_ARGS, argc, argv,  /* main args */
    0);

    /* this is a tty subwindow */
    tty_sw = window_create(base_frame, TTY,
        WIN_HEIGHT, TTYWINHEIGHT,
        WIN_WIDTH, TTYWINWIDTH
    0);

    /* this is a button subwindow */
    panel = window_create(
        base_frame, PANEL,
        WIN_BELOW, tty_sw,
        WIN_HEIGHT, PANELWINHEIGHT,
        WIN_WIDTH, PANELWINWIDTH,
        WIN_X, PANELWINX,
    0);
```

```
    /* This is the Filename input area */
    /* Filename is limited to 50 characters */

    filename_item = panel_create_item(
        panel, PANEL_TEXT,
        PANEL_LABEL_STRING, "Filename:",
        PANEL_VALUE_DISPLAY_LENGTH, 50,
    0);

    /* this is a button for calling WALKER */
    /* and then calls REACHWIN */
    panel_create_item(
        panel, PANEL_BUTTON,
        PANEL_NOTIFY_PROC, call_walker,
        PANEL_LABEL_IMAGE, panel_button_image (
            panel, "Walker", 0, 0),
    0);

    /* this is a button for calling FALTWIN */
    panel_create_item(
        panel, PANEL_BUTTON,
        PANEL_NOTIFY_PROC, call_faltwin,
        PANEL_LABEL_IMAGE, panel_button_image (
            panel, "Falter", 0, 0),
    0);

    /* this is a button for calling SPACEWIN */
    panel_create_item(
        panel, PANEL_BUTTON,
        PANEL_NOTIFY_PROC, call_spacewin,
        PANEL_LABEL_IMAGE, panel_button_image (
            panel, "Spacer", 0, 0),
    0);

    window_fit(base_frame);
    window_main_loop(base_frame);
    exit(0);
}
```

```
/* Process to call REACHER and WALKWIN */
static void
call_walker()
{
    char tmp_buf[81];
    Event event;
    int result;

    sprintf(tmp_buf, "%s", panel_get_value(filename_item));

    /* Check for "p" as last character. */
    /* Simple test for Pascal file. */

    if(tmp_buf[strlen(tmp_buf)-1] == 'p') {

        /* call REACHER on the filename entered by user */
        sprintf(tmp_buf, "reacher %s &\n", panel_get_value(filename_item));
        ttysw_input(tty_sw, tmp_buf, strlen(tmp_buf));

        /* call WALKWIN with filename as argument */
        sprintf(tmp_buf, "walkwin %s &\n", panel_get_value(filename_item));
        ttysw_input(tty_sw, tmp_buf, strlen(tmp_buf));
    }

    /* file name does not end in "p" */
    else {
        msg("Improper filename.", 1);
        panel_set_value(filename_item, "");
        return;
    }
}
```

```
/* Procedure to pop up alert frame with message about improper */
/* filename.  Called by call_walker. */
msg(msg, beep)
char *msg;
int  beep;
{
    int result;
    Event event;
    char *continue_msg = "Press \"Continue\" to proceed.";
    result = alert_prompt(base_frame, &event,ALERT_MESSAGE_STRINGS, msg,
        continue_msg, 0, ALERT_NO_BEEPING, (beep) ? 0:1,
        ALERT_BUTTON_YES, "Continue",  0);
    /* only one option for the user */
    switch (result) {
        case ALERT_YES:
        break;
    }
}

/* Procedure to call FALTWIN.  Uses filename in
 * filename_item.
 */

static void
call_faltwin()

{
    char tmp_buf[81];
    sprintf(tmp_buf, "faltwin %s &\n", panel_get_value(filename_item));
    ttysw_input(tty_sw, tmp_buf, strlen(tmp_buf));
}

/* Procedure to call SPACEWIN. */
static void
call_spacewin()
{
    char tmp_buf[81];
    sprintf(tmp_buf, "spacewin &\n", panel_get_value(filename_item));
    ttysw_input(tty_sw, tmp_buf, strlen(tmp_buf));
    panel_set_value(filename_item, "");
}
```

```
/* TITLE          : walkwin.c
 * AUTHORS        : Vicki Abel and Medio Monti
 * DATE           : 5 October 1990
 * REVISED        : 12 December 1990
 * SYSTEM         : NPS SUN TAURUS
 * LANGUAGE       : SunView and C
 * COMPILER       : Unix cc
 * DESCRIPTION    : This program is the interface for the failure region
 * testing tool WALKER.  Chapter II contains all the pertinent details relating
 * to this code.
 */

/* SunView header files needed in this program. */
#define MAIN   /* Necessary for the acfg graph */
#include <suntool/sunview.h>
#include <suntool/panel.h>
#include <suntool/textsw.h>
#include <suntool/tty.h>
#include <suntool/canvas.h>
#include <suntool/icon.h>
#include <suntool/scrollbar.h>
#include <pixrect/pr_line.h>
#include <stdio.h>
#include <math.h>

/* These macros contain declarations necessary for drawing the
 * acfg graph.  These header files are from Dr. Shimeall's testing
 * tools.
 */
#ifdef ACFGGRAPH
#include "walker.h"
#include "ablock.h"
#include "acfg.h"
#include XDISP      50
#endif ACFGGRAPH

#define NUM_POINTS  1000 /* Results in a smooth circle */
#define XDONE       200
#define YDONE       200
#define XSTART      200  /* Graph staring point */
#define YSTART      200  /* Graph starting point */
#define POSMAX      200
```

```
/* Presently necessary for defining a high resolution screen */
#ifdef HIGHRES
#define STARTX              200
#define STARTY              50
#define RADIUS              10
#define BASEFRAMEX          525
#define BASEFRAMEY          205
#define ACFGHEIGHT          794
#define ACFGWIDTH           500
#define CANVASHEIGHT        1400
#define CANVASWIDTH         900
#define TEXTWINHEIGHT       394
#define TEXTWINWIDTH        495
#define TTYWINHEIGHT        394
#define TTYWINWIDTH         495
#define PANELWINHEIGHT      100
#define PANELWINWIDTH       1000
#define PANELXGAP           10
#define FONT                1
#define XDISP               25
#define YDISP               25
#endif HIGHRES

/* Presently necessary for defining low resolution screens */
#ifdef LOWRES
#define STARTX              150
#define STARTY              25
#define RADIUS              5
#define BASEFRAMEX          365
#define BASEFRAMEY          163
#define ACFGHEIGHT          476
#define ACFGWIDTH           350
#define CANVASHEIGHT        540
#define CANVASWIDTH         890
#define TEXTWINHEIGHT       236
#define TEXTWINWIDTH        347
#define TTYWINHEIGHT        236
#define TTYWINWIDTH         347
#define PANELWINHEIGHT      100
#define PANELWINWIDTH       702
#define PANELXGAP           5
#define FONT                0
#define XDISP               15
```

```
#define YDISP                12
#endif LOWRES

/* these define pixrect ptrs */
#define NULLPR  ((Pixrect *) 0)

/* vlist is an array of structures pr_pos for the circle */
static struct pr_pos vlist0[NUM_POINTS];

/* SunView type declarations */
Frame           base_frame;
Panel           control_panel;
Canvas          acfg_canvas;
Textsw          text_sw;
Panel_item      walker_menu;
Panel_item      walker_input_string;
Panel_item      lister_input_string;
Panel_item      lister_output_string;
Tty             tty_sw;
Pixfont         *bold;
Pixfont         *controlfont;


Pixwin    *pw;      /*pixwin object */
int       i, j;     /* Loop variables */


/* Initializing graph variables */
int     startxval = XSTART;
int     startyval = YSTART;
int     finishxval = XDONE;
int     finishyval = YDONE;
float   angle;              /* for circle computations */
float   increment;          /* for circle computations */
int     counter = 0;        /* for circle computations */
int     walker_run = 0;
char    *file_name;    /* Name of file to load into text subwindow */


#ifdef ACFGGRAPH
FILE    *infile;       /* Logical file name for acfg graph */
#endif ACFGGRAPH
```

```
#ifdef ACFGGRAPH
/* Graph object declarations */
int       curx;
int       cury;
int       finalx;
int       finaly;
int       rootid;
/* structure for an (x,y) coordinate */
typedef struct
        {
              int x;
              int y;
        }POINT;
/* Array of the above structures */
POINT     position[POSMAX];
int       TOS = 0;        /* Top of Stack */
acfg      *curnode;       /*ptr to acfg */
ahbkref   curblock;       /*ptr to hdr structure */
acfg      *stack[POSMAX]; /*stack of acfg structures */
#endif ACFGGRAPH

/* SunView declarations for Icons */
Icon      walker_icon;
Icon      walker_menu_icon;

/* Procedures used by buttons and menus */
static void call_annotate();
static void call_change();
static void call_join();
static void call_goto();
static void call_prior();
static void call_left();
static void call_right();
static void call_node();
static void call_type();
static void walker_proc();
static void lister_proc();

#ifdef ACFGGRAPH
static void drawline();       /* For drawing arcs between nodes */
static void drawcircle();     /* For drawing the nodes */
#endif ACFGGRAPH
```

```
/* This is the icon for the frame */
static short icon_image[] = {
#include "walker.icon"
};
/* mpr_static is a SunView macro that imports an icon into the program*/
mpr_static(icon_pixrect, 64, 64, 1, icon_image);

/* This icon was created to allow differentiation between a button
 * with a related menu and a simple button.  It is used for the
 * menu for interaction with WALKER.
 */
static short walker_array[] = {
#include "walker_menu.icon"
};
mpr_static(walker_pixrect, 64, 64, 1, walker_array);

#define ADDED_EXIT_NODE 999999
int shifts[POSMAX];

void
subtreeshift(root,disp)
    acfg *root;
    int disp;
{
    if (root == NULL) return;
    if (root->acfgnum == ADDED_EXIT_NODE) return;
    shifts[root->acfgnum - rootid] += disp;
    subtreeshift(root->acfglft, disp);
    subtreeshift(root->acfgrt,disp);
}

int calcoffsets(root)
acfg *root;

{
    int leftval, rightval;
    if (root==NULL) return 0;
    if (root->marked == 1) return 0; /* crossovers possible */
    root->marked = 1;
    if (root->acfgnum == ADDED_EXIT_NODE) {
        shifts[POSMAX-1]=0;
   }
    else {
```

```
        shifts[root->acfgnum - rootid] = 0;
    }
    if (root->acfglft == NULL) return 0;
    if (root->acfgrt == NULL) return calcoffsets(root->acfglft);
    leftval = calcoffsets(root->acfglft);
    rightval = calcoffsets(root->acfgrt);
    if (leftval>0)
        if (rightval>0) {
            subtreeshift(root->acfglft, - (leftval+rightval+1)/2);
            subtreeshift(root->acfgrt, (leftval+rightval+1)/2);
        }
        else {
            subtreeshift(root->acfgrt, 1 + leftval/2);
            subtreeshift(root->acfglft, -(1+leftval/2));
        }
    else {
        subtreeshift(root->acfglft, - (rightval + 2)/2);
        subtreeshift(root->acfgrt, (rightval+2)/2);
    }
    return (int) 1.5*(leftval+rightval)+1;
}


void
drawgraph(root)
acfg *root;
{
    int done_draw = 0;
    curnode = root;
    rootid = root->acfgnum;
    curx = STARTX;
    cury = STARTY;


    calcoffsets(root);
    clearnode(root);


    pw_writebackground(pw,0,0,
        (pw->pw_pixrect)->pr_size.x,(pw->pw_pixrect)->pr_size.y,PIX_CLR);
```

```
    do {

#ifdef DEBUG
printf(" Drawing node %d (%d) into pos %d\n",curnode->acfgnum,
      curnode,curnode->acfgnum-rootid);
#endif DEBUG

        if (curnode == NULL) {
            printf(" ran off edge of graph\n");
            break;
        }

        if(curnode->acfgrt != NULL) {

#ifdef DEBUG
printf(" Pushing node %d (%d) into stack %d\n",curnode->acfgnum,curnode,TOS);
#endif DEBUG

            stack[TOS] = curnode;
            TOS++;
            finalx = curx - XDISP +
                ((curnode->acfglft->marked == 1) ? 0 :
                ((curnode->acfglft->acfgnum == ADDED_EXIT_NODE) ? 0 :
                shifts[curnode->acfglft->acfgnum-rootid]));
            finaly = cury + YDISP;
            drawline(curx, cury, finalx, finaly);
            drawcircle(curx, cury);
            if (curnode->acfgnum == ADDED_EXIT_NODE) {
                position[POSMAX-1].x = curx;
                position[POSMAX-1].y = cury;
            }
            else {
                position[curnode->acfgnum-rootid].x = curx;
                position[curnode->acfgnum-rootid].y = cury;
                curx = finalx;
                cury = finaly;
                curnode->marked = 1;

#ifdef DEBUG
printf(" Advancing left to node %d\n",curnode->acfglft);
#endif DEBUG

                curnode = curnode->acfglft;
```

```
            }
        }
        else if(curnode->acfglft == NULL && TOS>0) {

#ifdef DEBUG
printf(" Popping node %d (%d) from stack %d\n",stack[TOS-1]->acfgnum,
        stack[TOS-1],TOS-1);
#endif DEBUG

            TOS--;
            curnode = stack[TOS];
            finalx = curx + XDISP +
                ((curnode->acfgrt->marked == 1) ? 0 :
                ((curnode->acfgrt->acfgnum == ADDED_EXIT_NODE) ? 0 :
                shifts[curnode->acfgrt->acfgnum-rootid]));
            finaly = cury + YDISP;
            if (curnode->acfgnum == ADDED_EXIT_NODE) {
                position[POSMAX-1].x = curx;
                position[POSMAX-1].y = cury;
                drawcircle(curx, cury);
            }
            else {
                position[curnode->acfgnum-rootid].x = curx;
                position[curnode->acfgnum-rootid].y = cury;
                drawline(curx, cury, finalx, finaly);
                drawcircle(curx, cury);
                curx = finalx;
                cury = finaly;

#ifdef DEBUG
printf(" Advancing right to node %d\n",curnode->acfgrt);
#endif DEBUG

                curnode = curnode->acfgrt;
            }
        }

        else  if (curnode->acfglft != NULL) {
            if (curnode->acfgnum != ADDED_EXIT_NODE) {

#ifdef DEBUG
printf(" Advancing down to node %d\n",curnode->acfglft);
#endif DEBUG
```

```
                curnode = curnode->acfglft;
                finalx = curx;
                finaly = cury + YDISP;
                if (curnode->acfgnum == ADDED_EXIT_NODE) {
                    position[POSMAX-1].x = curx;
                    position[POSMAX-1].y = cury;
                    drawcircle(curx, cury);
                }
                else {
                    position[curnode->acfgnum-rootid].x = curx;
                    position[curnode->acfgnum-rootid].y = cury;
                    drawline(curx, cury, finalx, finaly);
                    drawcircle(curx, cury);
                    cury = finaly;
                }
            }
            else {done_draw = 1;}
        }

        if (curnode!=NULL)
            while((!done_draw) &&
                ((curnode->acfgnum == ADDED_EXIT_NODE)||(curnode->marked
== 1))
                && (TOS > 0)) {

#ifdef DEBUG
printf(" Popping node %d (%d) f  .n stack %d\n",stack[TOS-1]->acfgnum,
        stack[TOS-1],TOS-1);
#endif DEBUG

                curnode = stack[TOS-1];
                if (curnode == NULL) {done_draw=1; curnode=root; break;}
                    if (curnode->acfgnum == ADDED_EXIT_NODE) {
                        curx = position[POSMAX].x;
                        cury = position[POSMAX].y;
                    }
                    else    {
                        curx = position[curnode->acfgnum-rootid].x;
                        cury = position[curnode->acfgnum-rootid].y;
                    }
                if (curnode->acfgrt == NULL) {done_draw=1; break;}
                finalx = curx + XDISP +
                    ((curnode->acfgrt->marked == 1) ? 0 :
```

```
                ((curnode->acfgrt->acfgnum == ADDED_EXIT_NODE) ? 0 :
                shifts[curnode->acfgrt->acfgnum-rootid]));
            finaly = cury + YDISP;
            drawline(curx, cury, finalx, finaly);
            curx = finalx;
            cury = finaly;

#ifdef DEBUG
printf(" Advancing remaining link to node %d\n",curnode->acfgrt);
#endif DEBUG

            curnode = curnode->acfgrt;
            TOS--;
        }
        else {done_draw = 1;
            curnode = root;
        }

    } while (!done_draw && ((curnode->acfglft != NULL) ||
        (curnode->acfgrt != NULL) ||
        (TOS > 0)));

        if (!done_draw) drawcircle(curx, cury);
}

/* Main */

main(argc, argv)
int argc;
char *argv[];
{

/* This is the acfg window test procedure */
/* It can be deleted in future versions */
#ifdef MYGRAPH
static int npts[1] = {NUM_POINTS};
#endif MYGRAPH

#ifdef DEBUG
printf("started main\n");
#endif DEBUG
    /* assign second item of input line to file_name to display it */
    /* in the text subwindow */
```

```
    file_name = argv[1];

#ifdef DEBUG
printf("starting to initialize windows\n");
#endif DEBUG

    walker_icon = icon_create(ICON_IMAGE, &icon_pixrect, 0);

    /* this is the base frame for the application */
    base_frame = window_create(NULL, FRAME,
        WIN_X, BASEFRAMEX, /* sets x position relative to owner */
        WIN_Y, BASEFRAMEY, /* sets y position relative to owner */
        FRAME_LABEL, "Walker Window 2.4",  /* frame label */
        FRAME_ICON, walker_icon, /* icon used */
        FRAME_ARGS, argc, argv,       /* main args */
        0);

    /* this is a canvas subwindow */
    acfg_canvas = window_create(base_frame, CANVAS,
        WIN_HEIGHT, ACFGHEIGHT,
        WIN_WIDTH, ACFGWIDTH,
        CANVAS_AUTO_EXPAND, FALSE,
        CANVAS_AUTO_SHRINK, FALSE,
        CANVAS_WIDTH, CANVASWIDTH,
        CANVAS_HEIGHT, CANVASHEIGHT,
        WIN_VERTICAL_SCROLLBAR, scrollbar_create(
            SCROLL_PLACEMENT, SCROLL_WEST,
            SCROLL_PAGE_BUTTONS, FALSE,
            0),
        WIN_HORIZONTAL_SCROLLBAR, scrollbar_create(
            SCROLL_PLACEMENT, SCROLL_SOUTH,
            SCROLL_PAGE_BUTTONS, FALSE,
            0),
    0);

#ifdef ACFGGRAPH
#ifdef DEBUG
printf("Ready to start graph.c\n");
#endif DEBUG
    /*----------------graph.c main code-----------------------*/
    /* reacher_out is where we find the textual representation of the
     * control flow graph.  We must put it into the logical file to use
     * use it.
```

```
     */
    infile = fopen("reacher_out", "r");
    if (infile == NULL) {
        fprintf(stderr,"No reacher_out file\n");
        exit(1);
    }
    ahlen = 0;
    /* Dynamically allocate memory for the logical file and put it into
     * memory. The function readreacher is part of the failure region
     * testing tools. We used it with our program with Dr. Shimeall's
     * permission.
     */
    while(!feof(infile)) {
        int i;
        ahlen++;
        i = sizeof(ahbkref);

        if(ahlen > 1) {
            ahprocs = (ahbkhdr **)MYREALLOC(ahprocs, ahlen*i);
        }

        else {
            ahprocs = (ahbkhdr **) MYALLOC(i);
        }

        *(ahprocs + ahlen-1) = allocahbk();

        readreacher(infile, *(ahprocs + ahlen-1));
    }

#ifdef DEBUG
printf("Ready to draw graph\n");
#endif DEBUG
    /* initialize */

    curblock = *(ahprocs + ahlen-1);
    curnode = (curblock->abkgrph);
    if (curnode == NULL) {
        fprintf(stderr,"No nodes in reacher_out main block\n");
        exit(1);
    }

    /* Define pw and set up circle coordinates */
```

```
    pw = canvas_pixwin(acfg_canvas);
    increment = 2 * M_PI / NUM_POINTS;
    for(angle = 0; angle < (2 * M_PI); angle += increment) {
        vlist0[counter].x = RADIUS * cos(angle);
        vlist0[counter].y = RADIUS * sin(angle);
        counter++;
    }
   drawgraph(curblock->abkgrph);
#ifdef DEBUG
printf("Ready to initialize graphics panel\n");
#endif DEBUG

#endif ACFGGRAPH

#ifdef MYACFG
        pw = canvas_pixwin(acfg_canvas);

    /* Draw the circle */
    increment = 2 * M_PI / NUM_POINTS;
    for(angle = 0; angle < (2 * M_PI); angle += increment) {
        vlist0[counter].x = RADIUS * cos(angle);
        vlist0[counter].y = RADIUS * sin(angle);
        counter++;
    }

    /* This construct is designed to draw a vector, then to load an
     * array with the points to use to draw the circle, and the same
     * points are used to draw the lines around the polygon.
     */

    for(j = 1; j < 6; j++) {
        pw_vector(pw, startxval, startyval, finishxval, finishyval,
        PIX_SRC, 1);

        pw_polygon_2(pw, startxval, startyval, 1, npts, vlist0,
        PIX_CLR,  NULLPR, 0, 0);

        for(i = 0; i < (NUM_POINTS - 1); i++) {
            pw_vector(pw, vlist0[i].x + startxval, vlist0[i].y +
            startyval, vlist0[i+1].x + startxval, vlist0[i+1].y +
            startyval, PIX_SRC, 1);
        }
        pw_vector(pw, vlist0[NUM_POINTS -1].x + startxval,
```

```
            vlist0[NUM_POINTS -1].y + startyval,
            vlist0[0].x + startxval, vlist0[0].y + startyval,
            PIX_SRC, 1);

        startyval = finishyval;
        finishyval += YSTART;
    }
#endif MYACFG

#ifdef DEBUG
printf("Ready to initialize text panel\n");
#endif DEBUG

    /* Define a font for the text subwindow and control panel */

    if (FONT == 1){
        controlfont =
        pf_open("/usr/lib/fonts/fixedwidthfonts/screen.r.16");
    }
    else {
        controlfont =
        pf_open("/usr/lib/fonts/fixedwidthfonts/screen.r.11");
    }

    /* this is a text subwindow */
    text_sw = window_create(base_frame, TEXTSW,
        WIN_RIGHT_OF, acfg_canvas,
        WIN_HEIGHT, TEXTWINHEIGHT,
        WIN_WIDTH, TEXTWINWIDTH,
        /* word wrap vice default */
        TEXTSW_LINE_BREAK_ACTION, TEXTSW_WRAP_AT_WORD,
    0);

#ifdef DEBUG
printf("Ready to initialize comm panel\n");
#endif DEBUG
    /* this is a tty subwindow */
    tty_sw = window_create(base_frame, TTY,
        WIN_BELOW, text_sw,
        WIN_RIGHT_OF, acfg_canvas,
        WIN_HEIGHT, TTYWINHEIGHT,
        WIN_WIDTH, TTYWINWIDTH,
    0);
```

```
#ifdef DEBUG
printf("Ready to initialize control panel\n");
#endif DEBUG
    /* this is a panel subwindow */
    control_panel = window_create(base_frame, PANEL,
        WIN_BELOW, acfg_canvas,
        WIN_HEIGHT, PANELWINHEIGHT,
        WIN_WIDTH, PANELWINWIDTH,
        WIN_X, 0,
        PANEL_ITEM_X_GAP, PANELXGAP,
        WIN_FONT, controlfont,
    0);

#ifdef DEBUG
printf("Ready to initialize control panel\n");
#endif DEBUG
    /*  puts file in the text_window */
    window_set(text_sw, TEXTSW_FILE, file_name, 0);

#ifdef DEBUG
printf("Ready to initialize button menu and input\n");
#endif DEBUG
    /* Walker buttons and input */
    walker_menu = panel_create_item(control_panel,
        PANEL_CHOICE, PANEL_CHOICE_STRINGS,
            "Run",
            "Help",
            "Save",
            "Quit",
             0,
        PANEL_DISPLAY_LEVEL, PANEL_NONE,
        PANEL_LABEL_IMAGE, &walker_pixrect,
        PANEL_NOTIFY_PROC, walker_proc,
    0);

    /* Annotate Button */
    if(FONT == 1){
        bold = pf_open("/usr/lib/fonts/fixedwidthfonts/screen.b.16");
    }
    else {
        bold = pf_open("/usr/lib/fonts/fixedwidthfonts/screen.r.11");
    }
```

```
panel_create_item(control_panel, PANEL_BUTTON,
    PANEL_NOTIFY_PROC,  call_annotate,
    PANEL_LABEL_IMAGE,  panel_button_image(
        control_panel, "Annotate", 9, bold),
0);

/* Change Condition Button */
panel_create_item(control_panel, PANEL_BUTTON,
    PANEL_NOTIFY_PROC, call_change,
    PANEL_LABEL_IMAGE, panel_button_image(
        control_panel, "Condition", 9, bold),
0);

/* Join Button */
panel_create_item(control_panel, PANEL_BUTTON,
    PANEL_NOTIFY_PROC, call_join,
    PANEL_LABEL_IMAGE, panel_button_image(
        control_panel, "Join", 8, bold),
0);

/* Goto Button */
panel_create_item(control_panel, PANEL_BUTTON,
    PANEL_NOTIFY_PROC, call_goto,
    PANEL_LABEL_IMAGE, panel_button_image(
        control_panel, "Goto", 8, bold),
0);

/* Prior Button */
panel_create_item(control_panel, PANEL_BUTTON,
    PANEL_NOTIFY_PROC, call_prior,
    PANEL_LABEL_IMAGE, panel_button_image(
        control_panel, "Prior", 8, bold),
0);

/* Left Button */
panel_create_item(control_panel, PANEL_BUTTON,
    PANEL_NOTIFY_PROC, call_left,
    PANEL_LABEL_IMAGE, panel_button_image(
        control_panel, "Left", 8, bold),
0);

/* Right Button */
panel_create_item(control_panel, PANEL_BUTTON,
```

```
    PANEL_NOTIFY_PROC, call_right,
    PANEL_LABEL_IMAGE, panel_button_image(
        control_panel, "Right", 8, bold),
0);

/* Node Button */
panel_create_item(control_panel, PANEL_BUTTON,
    PANEL_NOTIFY_PROC, call_node,
    PANEL_LABEL_IMAGE, panel_button_image(
        control_panel, "Node", 8, bold),
0);

/* Type Button */
panel_create_item(control_panel, PANEL_BUTTON,
    PANEL_NOTIFY_PROC, call_type,
    PANEL_LABEL_IMAGE, panel_button_image(
        control_panel, "Type", 8, bold),
0);

/* Lister Button */
panel_create_item(control_panel, PANEL_BUTTON,
    PANEL_NOTIFY_PROC, lister_proc,
    PANEL_LABEL_IMAGE, panel_button_image(
        control_panel, "Lister", 8, bold),
    PANEL_ITEM_X, 100,
    PANEL_ITEM_Y, 70,
0);

/* This is the input string code for walker */
walker_input_string = panel_create_item(
    control_panel, PANEL_TEXT,
    PANEL_LABEL_STRING, "Walker input string:",
    PANEL_VALUE_DISPLAY_LENGTH, 50,
    PANEL_ITEM_X, 94,
    PANEL_ITEM_Y, 40,
0);

/* This is the input string code for lister */
lister_input_string = panel_create_item(
    control_panel, PANEL_TEXT,
    PANEL_LABEL_STRING, "Lister input:",
    PANEL_VALUE_DISPLAY_LENGTH, 15,
0);
```

```
    /* This is the output filename string code for lister */
    lister_output_string = panel_create_item(
        control_panel, PANEL_TEXT,
        PANEL_LABEL_STRING, "Lister output:",
        PANEL_VALUE_DISPLAY_LENGTH, 15,
    0);


#ifdef DEBUG
printf("All buttons initialized\n");
#endif DEBUG
    window_fit(base_frame);

#ifdef DEBUG
printf("Begin window main loop\n");
#endif DEBUG
    window_main_loop(base_frame);

    exit(0);
}

static void
call_annotate()
{
    char walker_string_buffer[81];
    sprintf(walker_string_buffer, "a %s\n",
        panel_get_value(walker_input_string));
    if (walker_run == 1) {
    ttysw_input(tty_sw, walker_string_buffer, strlen(
        walker_string_buffer));
    }
        panel_set_value(walker_input_string, "");
}

static void
call_change()
{
    char walker_string_buffer[81];
    sprintf(walker_string_buffer, "c %s\n",
        panel_get_value(walker_input_string));
    if (walker_run == 1) {
    ttysw_input(tty_sw, walker_string_buffer, strlen(
        walker_string_buffer));
```

```
    }
    panel_set_value(walker_input_string, "");
}

static void
call_join()
{
    char walker_string_buffer[81];
    sprintf(walker_string_buffer, "j %s\n",
        panel_get_value(walker_input_string));
    if (walker_run == 1) {
    ttysw_input(tty_sw, walker_string_buffer, strlen(
        walker_string_buffer));
    }
    panel_set_value(walker_input_string, "");
}

static void
call_goto()
{char walker_in_value[81];
    char walker_string_buffer[81];
    strcpy(walker_in_value, panel_get_value(walker_input_string));
    sprintf(walker_string_buffer, "g %s\n",walker_in_value);
    curblock=findahbk(walker_in_value);
    if ((walker_run == 1) && (curblock!= NULL)) {
    ttysw_input(tty_sw, walker_string_buffer, strlen(
        walker_string_buffer));
    }
    else { curblock = *(ahprocs+ahlen-1);
      /* eventually, we'll put a pop-up error message here */
      }
   drawgraph(curblock->abkgrph);
   panel_set_value(walker_input_string, "");
}

static void
call_prior()
{
    char walker_string_buffer[81];
    sprintf(walker_string_buffer, "p\n",
        panel_get_value(walker_input_string));
    if (walker_run == 1) {
    ttysw_input(tty_sw, walker_string_buffer, strlen(
```

```
        walker_string_buffer));
    }
    panel_set_value(walker_input_string, "");
}

static void
call_left()
{
    char walker_string_buffer[81];
    sprintf(walker_string_buffer, "l\n",
        panel_get_value(walker_input_string));
    if (walker_run == 1) {
    ttysw_input(tty_sw, walker_string_buffer, strlen(
        walker_string_buffer));
    }
    panel_set_value(walker_input_string, "");
}

static void
call_right()
{
    char walker_string_buffer[81];
    sprintf(walker_string_buffer, "r\n",
        panel_get_value(walker_input_string));
    if (walker_run== 1) {
    ttysw_input(tty_sw, walker_string_buffer, strlen(
        walker_string_buffer));
    }
    panel_set_value(walker_input_string, "");
}

static void
call_node()
{
    char walker_string_buffer[81];
    sprintf(walker_string_buffer, "n %s\n",
        panel_get_value(walker_input_string));
    if (walker_run== 1) {
    ttysw_input(tty_sw, walker_string_buffer, strlen(
        walker_string_buffer));
    }
    panel_set_value(walker_input_string, "");
}
```

```
static void
call_type()
{
    char walker_string_buffer[81];
    sprintf(walker_string_buffer, "t %s\n",
        panel_get_value(walker_input_string));
    if (walker_run== 1) {
    ttysw_input(tty_sw, walker_string_buffer, strlen(
        walker_string_buffer));
    }
    panel_set_value(walker_input_string, "");
}

static void
walker_proc(item, value, event)
    Panel_item item;
    int value;
    Event *event;
{
    char walker_string_buffer[81];
    if(event_action(event) ==
        MS_RIGHT && event_is_down(event)) {
        switch(value) {

            case 0:
                sprintf(walker_string_buffer, "walker -r reacher_out\n",
                 panel_get_value(walker_input_string));
                if (walker_run == 0) {
                    ttysw_input(tty_sw, walker_string_buffer,
                     strlen(
                     walker_string_buffer));
                    walker_run = 1;
                 }
                panel_set_value(walker_input_string, "");
                break;

            case 1:
                sprintf(walker_string_buffer, "h\n",
                 panel_get_value(walker_input_string));
                if (walker_run == 1) {
                    ttysw_input(tty_sw, walker_string_buffer,
                     strlen(walker_string_buffer));
                }
```

```
                panel_set_value(walker_input_string, "");
                break;

            case 2:
                sprintf(walker_string_buffer, "s %s\n",
                    panel_get_value(walker_input_string));
                if (walker_run == 1) {
                    ttysw_input(tty_sw, walker_string_buffer,
                     strlen(walker_string_buffer));
                }
                panel_set_value(walker_input_string, "");
                break;

            case 3:
                sprintf(walker_string_buffer, "q\n",
                    panel_get_value(walker_input_string));
                if (walker_run == 1) {
                    ttysw_input(tty_sw, walker_string_buffer,
                     strlen(walker_string_buffer));
                walker_run = 0;
                panel_set_value(walker_input_string, "");
                sprintf(walker_string_buffer, "clear\n",
                 panel_get_value(walker_input_string));
                ttysw_input(tty_sw, walker_string_buffer, strlen(
                 walker_string_buffer));
                }
                break;
        }
        }
}

static void
lister_proc()
{
    char walker_string_buffer[81];
    sprintf(walker_string_buffer, "lister -o %s %s\n",
     panel_get_value(lister_output_string),
     panel_get_value(lister_input_string));
    ttysw_input(tty_sw, walker_string_buffer, strlen(
     walker_string_buffer));
    panel_set_value(lister_input_string, "");
    panel_set_value(lister_output_string, "");
}
```

```
#ifdef ACFGGRAPH
static void
drawline(curx,cury,finalx,finaly)
int curx, cury, finalx,finaly;
{
#ifdef DEBUG
printf("Begin Drawline Function\n");
printf("%d %d %d %d\n", curx, cury, finalx, finaly);
#endif DEBUG

    pw_vector(pw, curx, cury, finalx, finaly, PIX_SRC, 1);

#ifdef DEBUG
printf("Leaving drawline function\n");
#endif DEBUG

}

static void
drawcircle(curx, cury)
int curx;
int cury;
{
    static int npts[1]= {NUM_POINTS};
#ifdef DEBUG
printf("Begin drawing Circle\n");
#endif DEBUG

    pw_polygon_2(pw, curx, cury, 1, npts, vlist0, PIX_CLR, NULLPR, 0,
     0);

    for(i = 0; i<(NUM_POINTS - 1); i++) {
        pw_vector(pw, vlist0[i].x + curx, vlist0[i].y + cury,
         vlist0[i+1].x + curx, vlist0[i+1].y + cury, PIX_SRC, 1);
    }
#ifdef DEBUG
printf("Leaving circle function\n");
#endif DEBUG
}
#endif ACFGGRAPH
```

```
/* walker.h - shared global types/defines for walker */
/* T. Shimeall July 1990 */

/*guard against multiple expansions*/
#ifndef WALKER

#ifndef EXTERN
#ifdef MAIN
#define EXTERN
#define INIT(Value) = Value
#else
#define EXTERN              extern
#define INIT(Value)
#endif MAIN
#endif EXTERN

#define NO_STMT             0
#define ASSIGN_STMT         1
#define CALL_STMT           2
#define IF_STMT             3
#define LOOP_STMT           4
#define CASE_STMT           5
#define WITH_STMT           6
#define BEGIN_END           7
#define GOTO_STMT           8
#define OTHER_STMT          9
/* below here -- extentions from REACHER tech report by */
/* R. Griffin */
#define EMPTY_STMT          10
#define FCALL_STMT          11
#define IFELSE_STMT         12
#define FLOOP_STMT          13
#define RLOOP_STMT          14
#define CASEIF_STMT         15
#define CASEEXIT_STMT       16
#define UNTIL_STMT          17

#define MAX_NAME            40
#define MAX_COND            512
#define BUFLEN              1024

typedef char condition[MAX_COND];
typedef struct grphrec {
```

```
    int acfgnum;
    int acfglnum;
    short marked;
    struct grphrec *acfglft;
    struct grphrec *acfgrt;
    condition acfglcnd;
    condition acfgrcnd;
    int acfgncalls;
    struct ahbkrec **acfgcalls;
    char acfgtext[BUFLEN];
    char acfgsumm[BUFLEN];
    int acfgtype;
} acfg;

typedef struct ahbkrec {
    char abkname[MAX_NAME];
    int abknumret;
    int abkcurret;
    acfg **abkret;
    condition abkreach;
    acfg *abkgrph;
    int abknsubs;
    struct ahbkrec **abksubs;
    char *abkdecl;
} ahbkhdr;

typedef ahbkhdr *ahbkref;

EXTERN char ahprgm[MAX_NAME] INIT("                                    ");
EXTERN int ahlen INIT(0);
EXTERN ahbkhdr **ahprocs INIT(0);
EXTERN char flterr[MAX_NAME]  INIT("                                  ");
char *malloc();
char *realloc();
#define P2CP(Obj) ((char *) Obj)
#define INT(Obj) ((int) Obj)
#define MAKEPOS(Size) ((INT(Size)>0) ? INT(Size) : 1)
#define MYALLOC(Size) malloc(MAKEPOS(Size))
#define MYREALLOC(Obj,Size) ((P2CP(Obj)==P2CP(NULL)) ? MYALLOC(Size)\
                                : realloc(Obj,MAKEPOS(Size)))
#define MYFREE(Obj) (P2CP(Obj) == P2CP(NULL) ? NULL : free(Obj))
#define WALKER
#endif WALKER
```

```
/* ablock.h  -- extern declarations for block header */
/* manipulations */
/* T. Shimeall July 1990 */

#ifndef ABLOCK
extern void initahbk();          /* initialize block */
extern ahbkref allocahbk();      /* allocate new block */
extern void remahbk();           /* delete named block */
extern int readreacher();        /* read data from reacher*/
extern void writereacher();      /* write falter save file*/
extern ahbkref findahbk();       /* find named block */
extern ahbkref findgrph();       /* find block containing indicated node */
#endif
```

```
/* acfg.h -- external declarations of public functions in acfg.c */
/* T. Shimeall October 1989 */
#ifndef ACFG
extern acfg *findacfg();        /* find or allocate acfg node */
extern acfg *allocacfg();       /* allocate new acfg node */
extern acfg *findnode();        /* find acfg node in graph */
extern void clearnode();        /* remove marking after acfg graph traversal */
extern void freeacfg();         /* add acfgnode to free list */
#endif
```

```
/* ablock.c -- block header manipulation routines */
/* T. Shimeall July 1990          */
#include <stdio.h>
#undef MAIN
#define ABLOCK
#include "walker.h"
#include "ablock.h"
#include "acfg.h"


typedef struct ahbkkprec {
    ahbkref cur; /* cur points to a block of 10 header records */
    struct ahbkkprec *next;
    struct ahbkkprec *last;
} ahbkeep;

static ahbkeep *bkalloc = NULL;      /* allocated header blocks (includes free) */
static ahbkeep *bkcur = NULL;  /* last entry of header blocks  list */
static ahbkref bkfree = NULL;  /* free header blocks */
static char *bkname = NULL;    /* name of next block, found while reading */
                               /* acfg nodes */
static int exitid = 999999;    /* initial id for single-exit nodes */


void initahbk(hdr)             /* initialize header values to zero values */
ahbkref hdr;
{   int i;
    for (i=0; i<MAX_NAME; i++)
    hdr->abkname[i] = '\0';
    hdr->abknumret = 0;
    hdr->abkcurret = 0;
    hdr->abkret = (acfg **) 0;
    strcpy(hdr->abkreach, "true");    /* default: routine always gets called */
    hdr->abkgrph = (acfg *) 0;
    hdr->abknsubs = 0;
    hdr->abksubs = (ahbkhdr **) 0;
    hdr->abkdecl = (char *) 0;
}

ahbkeep *allocahkeep()         /* allocate entry of bkalloc */
{   ahbkeep *ahbcur;
    ahbkref cur;
    ahbcur = (ahbkeep *) MYALLOC(sizeof(ahbkeep));
```

```
    ahbcur->next = NULL;
    ahbcur->last = NULL;
    ahbcur->cur = (ahbkhdr *) MYALLOC(10*sizeof(ahbkhdr));
    for (cur = ahbcur->cur ; cur < (ahbcur->cur + 10); cur++){
        initahbk(cur);
        cur->abksubs = (ahbkhdr **) (cur+1);
    }
    (ahbcur->cur+9)->abksubs = NULL;
    bkfree = ahbcur->cur;
    return ahbcur;
}

ahbkref allocahbk()          /* allocate new header block, using free list */
{   ahbkref cur;
    if (bkalloc == NULL) {
        bkalloc = allocahkeep();
        bkcur = bkalloc;
    }
    if (bkfree == NULL) {
        bkcur->next = allocahkeep();
        (bkcur->next)->last = bkcur;
        bkcur = bkcur->next;
    }
    if (bkcur == NULL) {
        printf("Major problem: bkcur null -- trying to recover\n");
        bkcur = bkalloc;
    }
    cur = bkfree;
    bkfree = (ahbkhdr *) bkfree->abksubs;
    cur->abksubs = (ahbkhdr **) 0;
    return cur;
}

ahbkref findahbk(name)          /* find named block */
char *name;
{   ahbkeep *kp;
    ahbkref cur;
    if (name == NULL) return NULL;
#ifdef DEBUG
if (bkcur == NULL) printf("Null block list\n");
#endif
```

```
    kp = bkcur;          /* search back to front (closer to Pascal scoping rules) */
    while (kp != NULL) {
        for (cur = kp->cur; cur < (kp->cur+10); cur++)

#ifdef DEBUG
{
    printf("find: looking for %s checking %s\n",name,cur->abkname);
    if (!strcmp(cur->abkname,name)) return cur;
}
#else
    if (!strcmp(cur->abkname,name)) return cur;
#endif
    kp = kp->last;
    }
 return NULL;
}

ahbkref findgrph(nodeid,cur)
/* find block containing node with acfgnum==nodeid */
int nodeid;                  /* node searched for */
ahbkref cur;                     /* current search location */
{   int i;
    ahbkref temp = NULL;
    if (nodeid== -1) return NULL;
    if (cur == NULL) return NULL;
    if (findnode(nodeid,cur->abkgrph) != NULL) {
        clearnode(cur->abkgrph);
        return cur;
    }
    for (i=0; (i < cur->abknsubs) && ((temp = findgrph(nodeid, cur->abksubs[i]))
==
        NULL); i++) ;
    clearnode(cur->abkgrph);          /* clean up after search */
    return temp; /* set by for loop either to NULL or to appropriate block */
}
```

```
void remahbk(cur)    /* deallocate header block, putting it on the free list */
ahbkref *cur;        /* double indirection so we can set it to null */
{   if (cur == NULL) return;        /* bad reference*/
    if ((*cur) == NULL) return;     /* nothing to deallocate */
    initahbk(*cur);
    (*cur)->abksubs = (ahbkhdr **) bkfree;
    bkfree = (*cur);
    (*cur) = NULL;
}

int grabint(f,inbuf)
FILE *f;
char *inbuf;
{   int temp;
    temp=-1;
    fgets(inbuf,255,f); inbuf[strlen(inbuf)-1] = '\0';
    if (sscanf(inbuf,"%d",&temp) != 1 ) temp = -1;
    return temp;
}



void grabstr(f,s,n,inbuf)
FILE    *f;
char    *s;
int     n;
char    *inbuf;
{   fgets(inbuf,255,f); inbuf[strlen(inbuf)-1] = '\0';
    strncpy(s,inbuf,n);
}
```

```
int numexit(root)
acfg     *root;
{    int excnt;
     if (root==NULL) return 1;
     if (root->marked) return 0;
     root->marked = 1;
     if (root->acfglft == NULL && root->acfgrt == NULL) return 1;
     excnt = 0;
     if (root->acfglft != NULL)
     excnt = numexit(root->acfglft);
     if (root->acfgrt != NULL)
     excnt += numexit(root->acfgrt);
     return excnt;
}

void makeexit(node)
acfg     *node;
{
     node->acfgnum = exitid; exitid--;
     node->acfglnum = 1;
     strcpy(node->acfgtext, "");
     strcpy(node->acfgsumm,"WALKER added exit node");
     strcpy(node->acfglcnd,"false");
     strcpy(node->acfgrcnd."false");
     node->acfglft = NULL;
     node->acfgrt = NULL;
     node->acfgncalls = 0;
     node->acfgcalls = 0;
     node->marked = 1;
     node->acfgtype = EMPTY_STMT;
}
```

```
void setexit(root,exnode)
acfg    *root;
acfg    *exnode;
{   if (root==NULL || exnode == NULL) return;
    if (root->marked == 0) return;
    root->marked = 0;
    if (root->acfglft == NULL && root->acfgrt == NULL && root != exnode)
    root->acfglft = exnode;
    else {
        setexit(root->acfglft,exnode);
        setexit(root->acfgrt,exnode);
    }
}

void singleexit(root)
acfg    **root;
{   acfg *exnode;
    if (numexit(*root) > 1) {
    exnode=allocacfg();
    makeexit(exnode);
    setexit(*root,exnode);
    }
    clearnode(*root);
}

#ifdef DEBUG
#define ENDTEST(Fileref,Msg) if (feof(Fileref)){printf(Msg); return -1;}\
                             else strcpy(flterr,"")
#else
#define ENDTEST(Fileref,Msg) if (feof(Fileref)){strcpy(flterr,Msg); return -1;}\
                             else strcpy(flterr,"")
#endif

int     readacfg(f,hdptr)
FILE    *f;
acfg    **hdptr;
{   int tmpindx;
    acfg *node;
    char inbuf[256];
    bkname=NULL;
    do {    tmpindx = grabint(f,inbuf);
```

```
#ifdef DEBUG
    printf("Read Node %d\n",tmpindx);
#endif
        if (tmpindx == -1) {
            /* we've found the start of the next block */
            bkname = MYALLOC(strlen(inbuf)+3),  /* 3 for insurance against */
                                               /* overflow */
            strcpy(bkname,inbuf);
        }
        else {
            if ((( (int) *hdptr) == 0) && !feof(f)) {
                *hdptr = findacfg(tmpindx);
                node = *hdptr;
            }
            else node = findacfg(tmpindx);
            if (node == NULL) return 0;
            ENDTEST(f,"End of file when expecting line number");
            node->acfglnum = grabint(f,inbuf);
            ENDTEST(f,"End of file when expecting lptr"); tmpindx =
            grabint(f,inbuf);
            if (tmpindx == -1) node->acfglft = NULL;
            else node->acfglft = findacfg(tmpindx);
            ENDTEST(f,"End of file when expecting rptr"); tmpindx =
            grabint(f,inbuf);
            if (tmpindx == -1) node->acfgrt = NULL;
            else node->acfgrt = findacfg(tmpindx);
            ENDTEST(f,"End of file when expecting lcnd");
            grabstr(f,node->acfglcnd,MAX_COND,inbuf);
            ENDTEST(f,"End of file when expecting rcnd");
            grabstr(f,node->acfgrcnd,MAX_COND,inbuf);
            ENDTEST(f,"End of file when expecting ncalls");
            node->acfgncalls = grabint(f,inbuf);
            if (node->acfgncalls >0) {
                node->acfgcalls = (struct ahbkrec **)
                MYALLOC(node->acfgncalls * sizeof(ahbkref));
                for (tmpindx = 0; tmpindx < node->acfgncalls; tmpindx++) {
                    ahbkref tmp;
                    ENDTEST(f,"End of file when expecting call name");
                    grabstr(f,inbuf,MAX_NAME,inbuf);
                    tmp = node->acfgcalls[tmpindx] = findahbk(inbuf);
                    if (tmp == NULL)
```

```
#ifdef DEBUG
                        {printf("can't find routine %s\n",inbuf); return 1;
                    }
#else
                        {strcpy(flterr,"can't find routine"); return 1;
                    }
#endif

    /* above line shouldn't happen */
    /* allocate temporary storage for return pointers -- be generous */
    /*  -- reduce to actual later */
                    if (tmp->abkret==NULL)
                        tmp->abkret = (acfg **) MYALLOC(50*sizeof(acfg *));
                    if ((tmp->abkcurret % 50 == 0) && (tmp->abkcurret>0))
                        tmp->abkret = (acfg **)
                    MYREALLOC(tmp->abkret,(50+tmp->abkcurret)*sizeof(acfg
*));
                    tmp->abkret[tmp->abkcurret++] = node;
                }
            }
            else node->acfgcalls = NULL;
            ENDTEST(f,"End of file when expecting text");
            fgets(inbuf,255,f);
            inbuf[strlen(inbuf)-1]='\0';
            strcpy(node->acfgtext, inbuf);
            ENDTEST(f,"End of file when expecting summary");
            fgets(inbuf,255,f);
            inbuf[strlen(inbuf)-1]='\0';
            strcpy(node->acfgsumm, inbuf);
            ENDTEST(f,"End of file when expecting node type");
            node->acfgtype = grabint(f,inbuf);

#ifdef DEBUG
            printf("@(%d)'%s'\nL:%d R:%d\n",node->acfglnum,node->acfgtext,
            ((node->acfglft==NULL) ? -1 : node->acfglft->acfgnum),
            ((node->acfgrt==NULL) ? -1 : node->acfgrt->acfgnum));
#endif
        }
    } while ((!feof(f)) && (bkname == NULL));
    singleexit(hdptr);
    return 0;
}
```

```
int readfixed(f,hdr)
FILE    *f;
ahbkref hdr;
{   int lenread;
    int lendecl;
    int readindx;
    int tmpindx;
    char inbuf[256];
    if (bkname == NULL) grabstr(f, hdr->abkname, MAX_NAME,inbuf);
    else {
        strcpy(hdr->abkname, bkname);
        MYFREE(bkname);
        bkname = NULL;
    }
        ENDTEST(f,"read of name");
        hdr->abknumret = grabint(f,inbuf);
        ENDTEST(f,"read of return");
        grabstr(f,hdr->abkreach,MAX_COND,inbuf);
        ENDTEST(f,"read of cond");
        hdr->abknsubs = grabint(f,inbuf);
        ENDTEST(f,"read of nsubs");
        lendecl = grabint(f,inbuf);
        /* read in decl lines */
        lenread = 0;
        if (lendecl>0) {
            hdr->abkdecl = (char *) MYALLOC(lendecl*160);
            /*too much, will trim later*/
            for (readindx=1; readindx<=lendecl; readindx++) {
                ENDTEST(f,"read of decl line");
                fgets(inbuf,255,f);
                for (tmpindx=0; tmpindx<strlen(inbuf); tmpindx++)
                    hdr->abkdecl[lenread++] = inbuf[tmpindx];
                    /*include \n at end of line*/
                if (lenread>(lendecl*160 - 40)) return -2;  /* lines too long */
            }
            hdr->abkdecl = (char *)MYREALLOC(hdr->abkdecl,lenread+1);
            /*trim storage*/
        }
        else hdr->abkdecl = (char *) MYALLOC(1);
        hdr->abkdecl[lenread++] = '\0';
        /* allocate return location storage */
        if (hdr->abknumret>0) {
```

```
            if (hdr->abkret == NULL) /* no calls from parent */
                hdr->abkret = (acfg **) MYALLOC((hdr->abknumret)*sizeof(
                    acfg *));
            else  /* trim space from overestimate parent uses to actual needs */
                hdr->abkret = (acfg **)
            MYREALLOC(hdr->abkret,(hdr->abknumret)*sizeof(acfg *));
            /* abkcurret is initialized to 0 and incremented on each call */
            if (hdr->abkcurret > hdr->abknumret) return 4; /* too many calls */
            for (tmpindx=hdr->abkcurret; tmpindx<hdr->abknumret; tmpindx++)
                hdr->abkret[tmpindx] = NULL;
        }
        else hdr->abkret = NULL;
        /* allocate submodule storage */
        if (hdr->abknsubs > 0) {
            hdr->abksubs = (ahbkhdr **) MYALLOC(hdr->abknsubs *
                sizeof(ahbkref));
            for (tmpindx = 0 ; tmpindx < hdr->abknsubs; tmpindx++)
                *(hdr->abksubs+tmpindx) = allocahbk();
        }
        /* read in submodule names */
        for (tmpindx = 0; tmpindx < hdr->abknsubs; tmpindx++){
            ENDTEST(f,"read of sub name");
            grabstr(f,(*(hdr->abksubs+tmpindx))->abkname,MAX_NAME,inbuf);
#ifdef DEBUG
    printf("Read in submodule ref of %s\n",(*(hdr->abksubs+tmpindx))->abkname);
#endif
        }
        return 0;
}
```

```
int readreacher(f,hdr)
/* Read reacher-generated file f to get data for block header hdr; */
/* return -1 if problem with read, return 0 if read is successful  */
FILE *f;                /* assumed to already be opened */
ahbkref hdr;                /* assumed to already be initialized */
{   char inbuf[80];
    int tmpindx;
#ifdef DEBUG
    acfg *node;
#endif

    if (f == NULL) return -1;   /* no file to read from */
    if (hdr == NULL) return -1; /* no place to copy to */
    ENDTEST(f,"test");
    /* read in fixed data: name, numret, reach, nsubs, number of decl lines;*/
    /* allocate return storage and submodule storage; */
    /*  read in decl lines and subroutine names */
    if (readfixed(f,hdr)!=0)
#ifdef DEBUG
    {printf("readfixed ended abnormally\n");
                                return 3;}
#else
    return 3;
#endif
#ifdef DEBUG
    printf("Read block %s\n",hdr->abkname);
#endif
    /* read in module acfg data */
#ifdef DEBUG
    if (readacfg(f,&(hdr->abkgrph))!=0) {
        printf("readacfg ended abnormally\n");
        return 3;
    }
    node=hdr->abkgrph;
    printf("readacfg ended normally\nStart of acfg:\n");
    printf("#%d @(%d)'%s'\nL:%d R:%d\n",node->acfgnum,
    node->acfglnum,node->acfgtext,
    ((node->acfglft==NULL) ? -1 : node->acfglft->acfgnum),
    ((node->acfgrt==NULL) ? -1 : node->acfgrt->acfgnum));
    node = ((node->acfglft!=NULL) ? node->acfglft:
    (node->acfgrt!=NULL) ? node->acfglft:NULL);
```

```
    if (node!=NULL)
        printf("#%d @(%d)'%s'\nL:%d R:%d\n",node->acfgnum,
    node->acfglnum,node->acfgtext,
    ((node->acfglft==NULL) ? -1 : node->acfglft->acfgnum),
    ((node->acfgrt==NULL) ? -1 : node->acfgrt->acfgnum));
#else
    if (readacfg(f,&(hdr->abkgrph))!=0) return 3;
#endif
    /* read in submodules*/
    if (hdr->abknsubs>0)
        for (tmpindx=0; tmpindx < hdr->abknsubs; tmpindx++) {
            ENDTEST(f,"test");
            if (readreacher(f, hdr->abksubs[tmpindx]) != 0)
#ifdef DEBUG
                {printf("readreacher ended abnormally\n");
                return 3;
            }
#else
                return 3;
#endif
        }
    else fgets(inbuf,80,f);          /* comment line */
    return 0;                        /* successfully read */
}

#undef ENDTEST
static acfg *terminal = NULL;

void dumpacfginfo(f,root)
FILE    *f;
acfg    *root;
{   int tmpindx;
    if (root == NULL) return;
    fprintf(f,"%d\n%d\n", root->acfgnum, root->acfglnum);
    root->marked =1;
    if (root->acfglft == NULL) fputs("-1\n",f);
    else fprintf(f,"%d\n",root->acfglft->acfgnum);
    if (root->acfgrt == NULL) fputs("-1\n",f);
    else fprintf(f,"%d\n",root->acfgrt->acfgnum);
    fputs(root->acfglcnd,f); fputc('\n',f);
    fputs(root->acfgrcnd,f);  fputc('\n',f);
    fprintf(f,"%d\n",root->acfgncalls);
```

```
    for (tmpindx=0; tmpindx < root->acfgncalls; tmpindx++) {
        fputs(root->acfgcalls[tmpindx]->abkname,f);
        fputc('\n',f);
    }
    fputs(root->acfgtext,f); fputc('\n',f);
    fputs(root->acfgsumm,f); fputc('\n',f);
    fprintf(f,"%d\n",root->acfgtype);
}

void writeacfg(f, root)
FILE    *f;
acfg    *root;
{   if (root == NULL) return; /* no data to write */
    if (root->marked) return; /* loop back, we've already done this node*/
    if (root->acfglft == NULL && root->acfgrt == NULL) {
        if (terminal != NULL && root != terminal) {
            fprintf(stderr,"Illegal graph in writeacfg (two terminal nodes)\n");
            exit(1);
        }
        terminal = root; /* save for output at end */
        return;
    }
    dumpacfginfo(f,root);
    fflush(f);
    if (root->acfglft!=NULL) writeacfg(f,root->acfglft);
    if (root->acfgrt!=NULL) writeacfg(f,root->acfgrt);
}

void writereacher(f, proc)
FILE    *f;
ahbkref proc;
{   int tmpctr, tmpidx;
    fputs(proc->abkname,f); fputc('\n',f);
    fprintf(f,"%d\n",proc->abknumret);
    fputs(proc->abkreach,f); fputc('\n',f);
    fprintf(f,"%d\n",proc->abknsubs);
    tmpctr = 0;
    if (proc->abkdecl!=NULL){
        for (tmpidx=0; tmpidx<strlen(proc->abkdecl); tmpidx++)
        if (proc->abkdecl[tmpidx] == '\n') tmpctr++;
        fprintf(f,"%d\n",tmpctr);
        fprintf(f,"%s",proc->abkdecl);
```

```
        for (tmpidx=0; tmpidx<proc->abknsubs; tmpidx++) {
            fputs(proc->abksubs[tmpidx]->abkname,f);
            fputc('\n',f);
        }
    }
    else fputs("0\n",f);
    fflush(f);
    terminal = NULL; /* clear any prior traversal */
    writeacfg(f,proc->abkgrph);
    dumpacfginfo(f,terminal); /* dump sink node last */
    fflush(f);
    clearnode(proc->abkgrph);
    for (tmpidx=0; tmpidx<proc->abknsubs; tmpidx++)
    writereacher(f,proc->abksubs[tmpidx]);
}
```

```
/* acfg.c -- annotated control-flow graph management functions for walker */
/* T. Shimeall July 1990 (from falter) */
#define ACFG
#include <stdio.h>
#include "walker.h"

typedef struct acfgkrec {
    acfg *cur;
    struct acfgkrec *next;
    struct acfgkrec *last;
} acfgkeep;

static acfgkeep *ndalloc = NULL;     /* allocated acfg nodes */
static acfgkeep *ndcur = NULL;  /* last entry of nodes list */
static acfg *acfgfree = NULL;     /* free node blocks */


acfgkeep *allocndkeep()              /* allocate entry of ndalloc */
{
    acfgkeep *acgcur;
    acfg *cur;
    acgcur = (acfgkeep *) MYALLOC(sizeof(acfgkeep));
    acgcur->next = NULL;
    acgcur->last = NULL;
    acgcur->cur = (acfg *) MYALLOC(10*sizeof(acfg));
    for (cur = acgcur->cur ; cur < (acgcur->cur + 10); cur++){
        cur->acfglft = (acfg *) (cur+1);
    }
    (acgcur->cur+9)->acfglft = NULL;
    acfgfree = acgcur->cur;
    return acgcur;
}


void freeacfg(node)              /* put node on free list */
acfg **node;
{
    (*node)->acfglft = acfgfree;
    (*node)->acfgrt = NULL;
    acfgfree =  *node;
    acfgfree->acfgncalls = 0;
    *node = NULL;
}
```

```
acfg *allocacfg()              /* allocate new acfg node, using free list */

{
    acfg *cur;
    if (ndalloc == NULL) {
        ndcur = ndalloc = allocndkeep();
    }

    if (acfgfree == NULL) {
        ndcur->next = allocndkeep();
        (ndcur->next)->last = ndcur;
        ndcur = ndcur->next;
    }

    cur = acfgfree;
    acfgfree = (acfg *) acfgfree->acfglft;
    cur->acfglft = (acfg *) 0;
    cur->marked = 0;
    return cur;
}

/* find (or create) acfg node with id given parametrically */
acfg *findacfg(id)
int id;

{
    acfg *cur;
    acfgkeep *kp;

    if (id == -1) return NULL;
    if (ndalloc != NULL) {
        kp = ndalloc;
        while (kp != NULL) {
            for (cur = kp->cur; cur <= (kp->cur + 9); cur++)
            if (cur->acfgnum == id) return cur;
            kp = kp->next;
        }
    }

    cur = allocacfg();
    cur->acfgnum = id;
    return cur;
}
```

```
acfg *findnode(nodeid, root)        /* find node in graph */
int nodeid;
acfg *root;
{
    acfg *lftret;
    if (root == NULL) return NULL;
    if (nodeid == -1) return NULL;
    if (root->marked) return NULL;  /* already checked this */
    if (root->acfgnum == nodeid) return root;
    root->marked = 1;
    if ((lftret = findnode(nodeid, root->acfglft))!=NULL) return lftret;
    return findnode(nodeid,root->acfgrt);
}

void clearnode(root)                /* clear marking after graph traversal */
acfg *root;
{
    if (root == NULL) return;
    if (!root->marked) return;
    root->marked = 0;
    clearnode(root->acfglft);
    clearnode(root->acfgrt);
}
```

```
/* TITLE          : faltwin.c
 * AUTHORS        : Vicki Abel and Medio Monti
 * DATE           : 20 October 1990
 * REVISED        : 12 December 1990
 * SYSTEM         : NPS SUN TAURUS
 * LANGUAGE       : SunView and C
 * COMPILER       : Unix cc
 * DESCRIPTION    : This program is the interface for the failure region
 * testing tool FALTER.  It is similar in design and construction to the WALKER
 * interface but functions to serve FALTER's needs.
 */

/* These are the SunView header files needed for this program. */
#define MAIN   /* Necessary for the acfg graph */
#include <suntool/sunview.h>
#include <suntool/panel.h>
#include <suntool/textsw.h>
#include <suntool/tty.h>
#include <suntool/canvas.h>
#include <suntool/icon.h>
#include <suntool/scrollbar.h>
#include <pixrect/pr_line.h>
#include <stdio.h>
#include <math.h>

/* These macros contain declarations necessary for drawing the
 * acfg graph.  These header files are from Dr. Shimeall's testing
 * tools.
 */

#ifdef ACFGGRAPH
#include "walker.h"
#include "ablock.h"
#include "acfg.h"
#endif ACFGGRAPH

#define NUM_POINTS  1000 /* for President Bush */
#define XSTART      200
#define YSTART      100
#define XDONE       200
#define YDONE       200
#define POSMAX      200
```

```
/* Presently necessary for defining a high resolution screen */
#ifdef HIGHRES
#define STARTX              200
#define STARTY              50
#define RADIUS              10
#define BASEFRAMEX          525
#define BASEFRAMEY          205
#define ACFGHEIGHT          794
#define ACFGWIDTH           500
#define CANVASHEIGHT        1400
#define CANVASWIDTH         900
#define TEXTWINHEIGHT       394
#define TEXTWINWIDTH        495
#define TTYWINHEIGHT        394
#define TTYWINWIDTH         495
#define PANELWINHEIGHT      100
#define PANELWINWIDTH       1000
#define PANELXGAP           10
#define FONT                1
#define XDISP               25
#define YDISP               25
#endif HIGHRES

#ifdef LOWRES
#define STARTX              150
#define STARTY              25
#define RADIUS              15
#define BASEFRAMEX          365
#define BASEFRAMEY          163
#define ACFGHEIGHT          476
#define ACFGWIDTH           350
#define CANVASHEIGHT        540
#define CANVASWIDTH         890
#define TEXTWINHEIGHT       236
#define TEXTWINWIDTH        347
#define TTYWINHEIGHT        236
#define TTYWINWIDTH         347
#define PANELWINHEIGHT      100
#define PANELWINWIDTH       702
#define PANELXGAP           5
#define FONT                0
#define XDISP               15
#define YDISP               12
```

```
#endif LOWRES

/* these define pixrect ptrs */
#define NULLPR  ((Pixrect *) 0)

/* vlist is an array of structures pr_pos  for the circle */
static struct pr_pos vlist0[NUM_POINTS];

/* type declarations */
Frame     base_frame;
Panel     control_panel;
Canvas        acfg_canvas;
Textsw        text_sw;
Tty           tty_sw;
Panel_item    falter_menu,
          falter_string_item;


Pixfont *bold;
/*pixwin object */
Pixwin *pw;

int i, j;
int startxval = XSTART;
int startyval = YSTART;
int finishxval = XDONE;
int finishyval = YDONE;
float angle; /* for circle computations */
float increment; /* for circle computations */
int counter = 0; /* for circle computations */
int falter_run = 0;

/* name of file to load into text subwindow */
char *file_name;
#ifdef ACFGGRAPH
FILE     *infile;      /* Logical file name for acfg graph */
#endif ACFGGRAPH

#ifdef ACFGGRAPH
/* Graph object declarations */
int       curx;
int       cury;
int       finalx;
```

```
int       finaly;
int       rootid;
/* structure for an (x,y) coordinate */
typedef struct
        {
                int x;
                int y;
        }POINT;
/* Array of the above structures */
POINT     position[200];
int       TOS = 0;  /* Top of Stack */
acfg      *curnode;          /*ptr to acfg */
ahbkref   curblock;          /*ptr to hdr structure */
acfg      *stack[200];       /*stack of acfg structures */
#endif ACFGGRAPH

Icon falter_icon;
Icon falter_menu_icon;

/* These procedures are notified by the buttons. */
static void call_add();
static void call_annotate();
static void call_error();
static void call_error_loc();
static void call_get_error();
static void call_implication();
static void call_left();
static void call_goto();
static void call_node();
static void call_prior();
static void call_right();
static void call_set_value();
static void call_error_type();
static void call_violation_set();

#ifdef ACFGGRAPH
static void drawline();       /* For drawing arcs between nodes */
static void drawcircle();     /* For drawing the nodes */
#endif ACFGGRAPH

static void falter_proc(); /* process for falter menu */

static short icon_image[] = {
```

```
#include "falter.icon"
};

mpr_static(icon_pixrect, 64, 64, 1, icon_image);

static short falter_array[] ={
#include "falter_menu.icon"
};

mpr_static(falter_pixrect, 64, 64, 1, falter_array);
#define ADDED_EXIT_NODE 999999
int shifts[POSMAX];

void subtreeshift(root,disp)
acfg *root;
int disp;
{if (root == NULL) return;
 if (root->acfgnum == ADDED_EXIT_NODE) return;
 shifts[root->acfgnum - rootid] += disp;
 subtreeshift(root->acfglft, disp);
 subtreeshift(root->acfgrt,disp);
}

int calcoffsets(root)
acfg *root;
{int leftval, rightval;
    if (root==NULL) return 0;
    if (root->marked == 1) return 0; /* crossovers possible */
    root->marked = 1;
    if (root->acfgnum == ADDED_EXIT_NODE) {
        shifts[POSMAX-1]=0;
    }
    else {
        shifts[root->acfgnum - rootid] = 0;
    }
    if (root->acfglft == NULL) return 0;
    if (root->acfgrt == NULL) return calcoffsets(root->acfglft);
    leftval = calcoffsets(root->acfglft);
    rightval = calcoffsets(root->acfgrt);
    if (leftval>0)
        if (rightval>0) {
            subtreeshift(root->acfglft, - (leftval+rightval+1)/2);
            subtreeshift(root->acfgrt, (leftval+rightval+1)/2);
```

```
        }
        else {
            subtreeshift(root->acfgrt, 1 + leftval/2);
            subtreeshift(root->acfglft, -(1+leftval/2));
        }
    else {
        subtreeshift(root->acfglft, - (rightval + 2)/2);
        subtreeshift(root->acfgrt, (rightval+2)/2);
    }
    return (int) 1.5*(leftval+rightval)+1;
}

void drawgraph(root)
acfg *root;
{ int done_draw = 0;
    curnode = root;
    rootid = root->acfgnum;
    curx = STARTX;
    cury = STARTY;

    calcoffsets(root);
    clearnode(root);

    pw_writebackground(pw,0,0,
        (pw->pw_pixrect)->pr_size.x,(pw->pw_pixrect)->pr_size.y,PIX_CLR);

        do {

#ifdef DEBUG
printf(" Drawing node %d (%d) into pos %d\n",curnode->acfgnum,
      curnode,curnode->acfgnum-rootid);
#endif DEBUG

            if (curnode == NULL) {
                printf(" ran off edge of graph\n");
                break;
            }

            if(curnode->acfgrt != NULL) {

#ifdef DEBUG
printf(" Pushing node %d (%d) into stack %d\n",curnode->acfgnum,curnode,TOS);
#endif DEBUG
```

```
                stack[TOS] = curnode;
                TOS++;
                finalx = curx - XDISP +
                    ((curnode->acfglft->marked == 1) ? 0 :
                    ((curnode->acfglft->acfgnum == ADDED_EXIT_NODE) ? 0 :
                    shifts[curnode->acfglft->acfgnum-rootid]));
                finaly = cury + YDISP;
                drawline(curx, cury, finalx, finaly);
                drawcircle(curx, cury);
                if (curnode->acfgnum == ADDED_EXIT_NODE) {
                    position[POSMAX-1].x = curx;
                    position[POSMAX-1].y = cury;
                }
                else {
                    position[curnode->acfgnum-rootid].x = curx;
                    position[curnode->acfgnum-rootid].y = cury;
                    curx = finalx;
                    cury = finaly;
                    curnode->marked = 1;
#ifdef DEBUG
printf(" Advancing left to node %d\n",curnode->acfglft);
#endif
                    curnode = curnode->acfglft;
                }
            }

            else if(curnode->acfglft == NULL && TOS>0) {
#ifdef DEBUG
printf(" Popping node %d (%d) from stack %d\n",stack[TOS-1]->acfgnum,
      stack[TOS-1],TOS-1);
#endif DEBUG
                TOS--;
                curnode = stack[TOS];
                finalx = curx + XDISP +
                    ((curnode->acfgrt->marked == 1) ? 0 :
                    ((curnode->acfgrt->acfgnum == ADDED_EXIT_NODE) ? 0 :
                    shifts[curnode->acfgrt->acfgnum-rootid]));
                finaly = cury + YDISP;
                if (curnode->acfgnum == ADDED_EXIT_NODE) {
                    position[POSMAX-1].x = curx;
                    position[POSMAX-1].y = cury;
                    drawcircle(curx, cury);
                }
```

```
                else {
                    position[curnode->acfgnum-rootid].x = curx;
                    position[curnode->acfgnum-rootid].y = cury;
                    drawline(curx, cury, finalx, finaly);
                    drawcircle(curx, cury);
                    curx = finalx;
                    cury = finaly;
#ifdef DEBUG
printf(" Advancing right to node %d\n",curnode->acfgrt);
#endif
                    curnode = curnode->acfgrt;
                }
            }
            else  if (curnode->acfglft != NULL) {
                if (curnode->acfgnum != ADDED_EXIT_NODE) {

#ifdef DEBUG
printf(" Advancing down to node %d\n",curnode->acfglft);
#endif

                    curnode = curnode->acfglft;
                    finalx = curx;
                    finaly = cury + YDISP;
                    if (curnode->acfgnum == ADDED_EXIT_NODE) {
                        position[POSMAX-1].x = curx;
                        position[POSMAX-1].y = cury;
                        drawcircle(curx, cury);
                    }
                    else {
                        position[curnode->acfgnum-rootid].x = curx;
                        position[curnode->acfgnum-rootid].y = cury;
                        drawline(curx, cury, finalx, finaly);
                        drawcircle(curx, cury);
                        cury = finaly;
                    }
                }
                else {done_draw = 1;}
            }
            if (curnode!=NULL)
            while((!done_draw) &&
                ((curnode->acfgnum == ADDED_EXIT_NODE)||(curnode->marked
== 1))
                && (TOS > 0)) {
```

```
#ifdef DEBUG
printf(" Popping node %d (%d) from stack %d\n",stack[TOS-1]->acfgnum,
      stack[TOS-1],TOS-1);
#endif DEBUG
                curnode = stack[TOS-1];
                if (curnode == NULL) {done_draw=1; curnode=root; break;}
                if (curnode->acfgnum == ADDED_EXIT_NODE) {
                    curx = position[POSMAX].x;
                    cury = position[POSMAX].y;
                }
                else {
                    curx = position[curnode->acfgnum-rootid].x;
                    cury = position[curnode->acfgnum-rootid].y;
                }
                if (curnode->acfgrt == NULL) {done_draw=1; break;}
                finalx = curx + XDISP +
                    (curnode->acfgrt->marked == 1) ? 0 :
                    ((curnode->acfgrt->acfgnum == ADDED_EXIT_NODE) ? 0 :
                    shifts[curnode->acfgrt->acfgnum-rootid]));
                finaly = cury + YDISP;
                drawline(curx, cury, finalx, finaly);
                curx = finalx;
                cury = finaly;
#ifdef DEBUG
printf(" Advancing remaining link to node %d\n",curnode->acfgrt);
#endif
                curnode = curnode->acfgrt;
                TOS--;
            }
            else {done_draw = 1;
                curnode = root;
            }

        } while (!done_draw && ((curnode->acfglft != NULL) ||
        (curnode->acfgrt != NULL) ||
        (TOS > 0)));

    if (!done_draw) drawcircle(curx, cury);
}

/* ----------------------- MAIN PROGRAM ----------------------*/

main(argc, argv)
```

```
int argc;
char *argv[];
{
static int npts[1] = {NUM_POINTS};

/* assign second item of input line to filename to display it */
/* in the text subwindow */
file_name = argv[1];

falter_icon = icon_create(ICON_IMAGE, &icon_pixrect, 0);

/* this is the base frame for the application */
    base_frame = window_create(NULL, FRAME,
        WIN_X,  BASEFRAMEX, /* sets x position relative to owner */
        WIN_Y,  BASEFRAMEY, /* sets y position relative to owner */
        FRAME_LABEL, "Falter Window 1.0",  /* frame label */
        FRAME_ICON, falter_icon,    /* icon used */
        FRAME_ARGS, argc, argv,        /* main args */
        0);

/* this is a graphics subwindow */
    acfg_canvas = window_create(base_frame, CANVAS,
        WIN_HEIGHT, ACFGHEIGHT,
        WIN_WIDTH, ACFGWIDTH,
        CANVAS_AUTO_EXPAND,   FALSE,
        CANVAS_AUTO_SHRINK,   FALSE,
        CANVAS_WIDTH,  CANVASWIDTH,
        CANVAS_HEIGHT,    CANVASHEIGHT,
        WIN_VERTICAL_SCROLLBAR, scrollbar_create(
            SCROLL_PLACEMENT,   SCROLL_WEST,
            SCROLL_PAGE_BUTTONS,   FALSE,
            0),
        WIN_HORIZONTAL_SCROLLBAR, scrollbar_create(
            SCROLL_PLACEMENT,   SCROLL_SOUTH,
            SCROLL_PAGE_BUTTONS,   FALSE,
            0),
        0);
#ifdef ACFGGRAPH
#ifdef DEBUG
printf("Ready to start graph.c\n");
#endif DEBUG
    /*---------------graph.c main code----------------------*/
    /* reacher_out is where we find the textual representation of the
```

```
     * control flow graph.  We must put it into the logical file to use
     * use it.
     */
    infile = fopen("walker_out", "r");
    if (infile == NULL) infile = fopen("reacher_out","r");
    if (infile == NULL) {
        fprintf(stderr,"No walker_out or reacher_out file\n");
        exit(1);
    }
    ahlen = 0;
    /* Dynamically allocate memory for the logical file and put it into
     * memory.  The function readreacher is part of the failure region
     * testing tools.  We used it with our program with Dr. Shimeall's
     * permission.
     */
    while(!feof(infile)) {
int i;
        ahlen++;
        i = sizeof(ahbkref);

        if(ahlen > 1) {
            ahprocs = (ahbkhdr **)MYREALLOC(ahprocs, ahlen*i);
        }

        else {
            ahprocs = (ahbkhdr **) MYALLOC(i);
        }

        *(ahprocs + ahlen-1) = allocahbk();

        readreacher(infile, *(ahprocs + ahlen-1));
    }

#ifdef DEBUG
printf("Ready to draw graph\n");
#endif DEBUG
    /* initialize */
 curblock = *(ahprocs + ahlen-1);
 curnode = curblock->abkgrph;

    if (curnode == NULL) {
        fprintf(stderr,"No nodes in walker_out main block\n");
        exit(1);
```

```
    }

    /* Define pw and set up circle coordinates */
    pw = canvas_pixwin(acfg_canvas);
    increment = 2 * M_PI / NUM_POINTS;
    for(angle = 0; angle < (2 * M_PI); angle += increment) {
        vlist0[counter].x = RADIUS * cos(angle);
        vlist0[counter].y = RADIUS * sin(angle);
        counter++;
    }

   drawgraph(curblock->abkgrph);
#ifdef DEBUG
printf("Ready to initialize graphics panel\n");
#endif DEBUG

#endif ACFGGRAPH


/* this is a text subwindow */
    text_sw = window_create(base_frame, TEXTSW,
        WIN_RIGHT_OF, acfg_canvas,
        WIN_HEIGHT, TEXTWINHEIGHT,
        WIN_WIDTH,  TEXTWINWIDTH,
        /* word wrap vice default */
        TEXTSW_LINE_BREAK_ACTION, TEXTSW_WRAP_AT_WORD,
        0);

/* this is a tty subwindow */
    tty_sw = window_create(base_frame, TTY,
        WIN_BELOW, text_sw,
        WIN_RIGHT_OF, acfg_canvas,
        WIN_HEIGHT, TTYWINHEIGHT,
        WIN_WIDTH, TTYWINWIDTH,
        0);

/* this is a button subwindow */
    control_panel = window_create(base_frame, PANEL,
        WIN_BELOW, acfg_canvas,
        WIN_HEIGHT,      PANELWINHEIGHT,
        WIN_WIDTH,       PANELWINWIDTH,
        WIN_X,   0,
        PANEL_ITEM_X_GAP,    PANELXGAP,
```

```
        0);

/*  puts file in the text_window */
    window_set(text_sw, TEXTSW_FILE, file_name, 0);


/* Falter buttons and input */

    falter_menu = panel_create_item(control_panel,
        PANEL_CHOICE, PANEL_CHOICE_STRINGS,
            "Run",
            "Help",
            "Save",
            "Quit",
            0,
        PANEL_DISPLAY_LEVEL,    PANEL_NONE,
        PANEL_LABEL_IMAGE,      &falter_pixrect,
        PANEL_NOTIFY_PROC,      falter_proc,
        0);

/* Add Button */
    bold = pf_open("/usr/lib/fonts/fixedwidthfonts/screen.b.16");
    panel_create_item(control_panel, PANEL_BUTTON,
        PANEL_NOTIFY_PROC,   call_add,
        PANEL_LABEL_IMAGE,   panel_button_image(
            control_panel,     "Add Fault", 11, bold),
        0);

/* Annotate Button */

    panel_create_item(control_panel, PANEL_BUTTON,
        PANEL_NOTIFY_PROC, call_annotate,
        PANEL_LABEL_IMAGE, panel_button_image(
            control_panel, "Annotate", 11, bold),
        0);

/* Error Button */

    panel_create_item(control_panel, PANEL_BUTTON,
        PANEL_NOTIFY_PROC, call_error,
        PANEL_LABEL_IMAGE, panel_button_image(
            control_panel, "Error", 11, bold),
        0);
```

```
/* FLOC to LocCond Button */

    panel_create_item(control_panel, PANEL_BUTTON,
        PANEL_NOTIFY_PROC, call_error_loc,
        PANEL_LABEL_IMAGE, panel_button_image(
            control_panel, "Error Loc.", 11, bold),
        0);

/* using entry indicated by ErrNum */

    panel_create_item(control_panel, PANEL_BUTTON,
        PANEL_NOTIFY_PROC, call_get_error,
        PANEL_LABEL_IMAGE, panel_button_image(
            control_panel, "Get Error", 11, bold),
        0);

/* Implication Button */

    panel_create_item(control_panel, PANEL_BUTTON,
        PANEL_NOTIFY_PROC, call_implication,
        PANEL_LABEL_IMAGE, panel_button_image(
            control_panel, "Implication", 11, bold),
        0);

/* Left Button */

    panel_create_item(control_panel, PANEL_BUTTON,
        PANEL_NOTIFY_PROC, call_left,
        PANEL_LABEL_IMAGE, panel_button_image(
            control_panel, "Left", 11, bold),
        0);

/* Module Button */

    panel_create_item(control_panel, PANEL_BUTTON,
        PANEL_NOTIFY_PROC, call_goto,
        PANEL_LABEL_IMAGE, panel_button_image(
            control_panel, "Goto", 11, bold),
        PANEL_ITEM_X,   77,
        PANEL_ITEM_Y,   35,
        0);

/* Node Button */
```

```
    panel_create_item(control_panel, PANEL_BUTTON,
        PANEL_NOTIFY_PROC, call_node,
        PANEL_LABEL_IMAGE, panel_button_image(
            control_panel, "Node", 11, bold),
        0);

/* Prior Button */

    panel_create_item(control_panel, PANEL_BUTTON,
        PANEL_NOTIFY_PROC, call_prior,
        PANEL_LABEL_IMAGE, panel_button_image(
            control_panel, "Prior", 11, bold),
        0);

/* Right Button */

    panel_create_item(control_panel, PANEL_BUTTON,
        PANEL_NOTIFY_PROC, call_right,
        PANEL_LABEL_IMAGE, panel_button_image(
            control_panel, "Right", 11, bold),
        0);

/* Set Value Button */

    panel_create_item(control_panel, PANEL_BUTTON,
        PANEL_NOTIFY_PROC, call_set_value,
        PANEL_LABEL_IMAGE, panel_button_image(
            control_panel, "Set Value", 11, bold),
        0);

/* Type Button */

    panel_create_item(control_panel, PANEL_BUTTON,
        PANEL_NOTIFY_PROC, call_error_type,
        PANEL_LABEL_IMAGE, panel_button_image(
            control_panel, "Error Type", 11, bold),
        0);

/* Violation Button */

    panel_create_item(control_panel, PANEL_BUTTON,
        PANEL_NOTIFY_PROC, call_violation_set,
        PANEL_LABEL_IMAGE, panel_button_image(
```

```
            control_panel, "Violation", 11, bold),
        0);


/* This is the input string code for falter */
    falter_string_item = panel_create_item(control_panel,
        PANEL_TEXT, PANEL_LABEL_STRING,
        "Falter input string:",
        PANEL_VALUE_DISPLAY_LENGTH, 50,
        PANEL_ITEM_X,   94,
        PANEL_ITEM_Y,   70,
        0);

    window_fit(base_frame);

    window_main_loop(base_frame);

    exit(0);
}

static void
call_add()
{
    char falter_string_buffer[81];
    sprintf(falter_string_buffer, "a\n",
        panel_get_value(falter_string_item));
    if (falter_run == 1) {
    ttysw_input(tty_sw, falter_string_buffer, strlen(
        falter_string_buffer));
    }
        panel_set_value(falter_string_item, "");
}

static void
call_annotate()
{
    char falter_string_buffer[81];
    sprintf(falter_string_buffer, "c %s\n",
        panel_get_value(falter_string_item));
    if (falter_run == 1) {
    ttysw_input(tty_sw, falter_string_buffer, strlen(
        falter_string_buffer));
    }
```

```
    panel_set_value(falter_string_item, "");
}

static void
call_error()
{
    char falter_string_buffer[81];
    sprintf(falter_string_buffer, "e %s\n",
        panel_get_value(falter_string_item));
    if (falter_run == 1) {
    ttysw_input(tty_sw, falter_string_buffer, strlen(
        falter_string_buffer));
    }
    panel_set_value(falter_string_item, "");
}

static void
call_error_loc()
{
    char falter_string_buffer[81];
    sprintf(falter_string_buffer, "f %s\n",
        panel_get_value(falter_string_item));
    if (falter_run == 1) {
    ttysw_input(tty_sw, falter_string_buffer, strlen(
        falter_string_buffer));
    }
    panel_set_value(falter_string_item, "");
}

static void
call_get_error()
{
    char falter_string_buffer[81];
    sprintf(falter_string_buffer, "g %s\n",
        panel_get_value(falter_string_item));
    if (falter_run == 1) {
    ttysw_input(tty_sw, falter_string_buffer, strlen(
        falter_string_buffer));
    }
    panel_set_value(falter_string_item, "");
}

static void
```

```
call_implication()
{
    char falter_string_buffer[81];
    sprintf(falter_string_buffer, "i %s\n",
        panel_get_value(falter_string_item));
    if (falter_run == 1) {
    ttysw_input(tty_sw, falter_string_buffer, strlen(
        falter_string_buffer));
    }
    panel_set_value(falter_string_item, "");
}

static void
call_left()
{
    char falter_string_buffer[81];
    sprintf(falter_string_buffer, "l\n",
        panel_get_value(falter_string_item));
    if (falter_run == 1) {
    ttysw_input(tty_sw, falter_string_buffer, strlen(
        falter_string_buffer));
    }
    panel_set_value(falter_string_item, "");
}

static void
call_goto()
{
    char falter_in_value[81];
    char falter_string_buffer[81];
    strcpy(falter_in_value,panel_get_value(falter_string_item));
    sprintf(falter_string_buffer, "m %s\n",falter_in_value);
    curblock = findahbk(falter_in_value);
    if (falter_run == 1 && curblock != NULL) {
    ttysw_input(tty_sw, falter_string_buffer, strlen(
        falter_string_buffer));
    }
    else {curblock = *(ahprocs+ahlen-1);
      /* eventually, we'll put a pop-up error message here */
      }
    drawgraph(curblock->abkgrph);
    panel_set_value(falter_string_item, "");
}
```

```
static void
call_node()
{
    char falter_string_buffer[81];
    sprintf(falter_string_buffer, "n %s\n",
        panel_get_value(falter_string_item));
    if (falter_run == 1) {
    ttysw_input(tty_sw, falter_string_buffer, strlen(
        falter_string_buffer));
    }
    panel_set_value(falter_string_item, "");
}

static void
call_prior()
{
    char falter_string_buffer[81];
    sprintf(falter_string_buffer, "p\n",
        panel_get_value(falter_string_item));
    if (falter_run == 1) {
    ttysw_input(tty_sw, falter_string_buffer, strlen(
        falter_string_buffer));
    }
    panel_set_value(falter_string_item, "");
}

static void
call_right()
{
    char falter_string_buffer[81];
    sprintf(falter_string_buffer, "r\n",
        panel_get_value(falter_string_item));
    if (falter_run == 1) {
    ttysw_input(tty_sw, falter_string_buffer, strlen(
        falter_string_buffer));
    }
    panel_set_value(falter_string_item, "");
}

static void
call_set_value()
{
    char falter_string_buffer[81];
```

```
    sprintf(falter_string_buffer, "s\n",
        panel_get_value(falter_string_item));
    if (falter_run == 1) {
    ttysw_input(tty_sw, falter_string_buffer, strlen(
        falter_string_buffer));
    }
    panel_set_value(falter_string_item, "");
}

static void
call_error_type()
{
    char falter_string_buffer[81];
    sprintf(falter_string_buffer, "t %s\n",
        panel_get_value(falter_string_item));
    if (falter_run == 1) {
    ttysw_input(tty_sw, falter_string_buffer, strlen(
        falter_string_buffer));
    }
    panel_set_value(falter_string_item, "");
}

static void
call_violation_set()
{
    char falter_string_buffer[81];
    sprintf(falter_string_buffer, "v %s\n",
        panel_get_value(falter_string_item));
    if (falter_run == 1) {
    ttysw_input(tty_sw, falter_string_buffer, strlen(
        falter_string_buffer));
    }
    panel_set_value(falter_string_item, "");
}


static void
falter_proc(item, value, event)
    Panel_item item;
    int value;
    Event *event;
{
    char falter_string_buffer[81];
```

```
if(event_action(event) ==
    MS_RIGHT && event_is_down(event)) {
    switch(value) {
        case 0:
            sprintf(falter_string_buffer, "falter %s\n",
                panel_get_value(falter_string_item));
            if (falter_run == 0) {
            ttysw_input(tty_sw, falter_string_buffer, strlen(
                falter_string_buffer));
            falter_run = 1;
            }
            panel_set_value(falter_string_item, "");
            break;
        case 1:
            sprintf(falter_string_buffer, "h\n",
                panel_get_value(falter_string_item));
            if (falter_run == 1) {
            ttysw_input(tty_sw, falter_string_buffer, strlen(
                falter_string_buffer));
            }
            panel_set_value(falter_string_item, "");
            break;
        case 2:
            sprintf(falter_string_buffer, "w %s\n",
                panel_get_value(falter_string_item));
            if (falter_run == 1) {
            ttysw_input(tty_sw, falter_string_buffer, strlen(
                falter_string_buffer));
            }
            panel_set_value(falter_string_item, "");
            break;
        case 3:
            sprintf(falter_string_buffer, "q\n",
                panel_get_value(falter_string_item));
            if (falter_run == 1) {
            ttysw_input(tty_sw, falter_string_buffer, strlen(
                falter_string_buffer));
            falter_run = 0;
            }
            panel_set_value(falter_string_item, "");
            sprintf(falter_string_buffer, "clear\n",
                panel_get_value(falter_string_item));
            ttysw_input(tty_sw, falter_string_buffer, strlen(
```

```
                falter_string_buffer));
            break;
        }
        }
}
#ifdef ACFGGRAPH
static void
drawline(curx,cury,finalx,finaly)
int curx, cury, finalx,finaly;
{
#ifdef DEBUG
printf("Begin Drawline Function\n");
printf("%d %d %d %d\n", curx, cury, finalx, finaly);
#endif DEBUG

    pw_vector(pw, curx, cury, finalx, finaly, PIX_SRC, 1);

#ifdef DEBUG
printf("Leaving drawline function\n");
#endif DEBUG

}


static void
drawcircle(curx, cury)
int curx;
int cury;
{
    static int npts[1]= {NUM_POINTS};
#ifdef DEBUG
printf("Begin drawing Circle\n");
#endif DEBUG

    pw_polygon_2(pw, curx, cury, 1, npts, vlist0, PIX_CLR, NULLPR, 0,
     0);

    for(i = 0; i<(NUM_POINTS - 1); i++) {
        pw_vector(pw, vlist0[i].x + curx, vlist0[i].y + cury,
         vlist0[i+1].x + curx, vlist0[i+1].y + cury, PIX_SRC, 1);
    }
```

```
#ifdef DEBUG
printf("Leaving circle function\n");
#endif DEBUG
}
#endif ACFGGRAPH
```

```
/* TITLE          : spacewin.c
 * AUTHORS        : Vicki Abel and Medio Monti
 * DATE           : 15 November 1990
 * REVISED        : 30 November 1990
 * SYSTEM         : NPS SUN TAURUS
 * LANGUAGE       : SunView and C
 * COMPILER       : Unix cc
 * DESCRIPTION    : This program creates the spacer frame and the subwindows
 * associated with it.
 */

/* SunView header files needed in this program. */
#include <suntool/sunview.h>
#include <suntool/panel.h>
#include <suntool/tty.h>
#include <suntool/icon.h>
#include <suntool/scrollbar.h>
#include <pixrect/pr_line.h>
#include <stdio.h>
#include <math.h>

#define BASEFRAMEWINX      525
#define BASEFRAMEWINY      205
#define TTYWINHEIGHT       794
#define TTYWINWIDTH        1000
#define PANELWINHEIGHT     100
#define PANELWINWIDTH      1000
#define PANELWINX          0
#define PANELXGAP          27

/* type declarations */
Frame       base_frame;
Panel       control_panel;
Panel_item  menu;
Panel_item  input_string;

Tty         tty_sw;
Icon        spacer_icon;

/* Procedures used by buttons and menus */
static void call_right();
static void spacer_proc();
```

```
/* This is the icon for the frame */
static short icon_image[] = {
#include "spacer.icon"
};

mpr_static(spacer_pixrect, 64, 64, 1, icon_image);

main(argc, argv)
int argc;
char *argv[];
{
    spacer_icon = icon_create(ICON_IMAGE, &spacer_pixrect, 0);

    /* this is the base frame for the application */
    base_frame = window_create(NULL, FRAME,
        WIN_X,          BASEFRAMEWINX, /* sets x position */
        WIN_Y,          BASEFRAMEWINY, /* sets y position */
        FRAME_LABEL,    "Spacer Window 1.0",/* frame label */
        FRAME_ICON,     spacer_icon, /* icon used */
        FRAME_ARGS,     argc, argv,  /* main args */
    0);

    /* this is a tty subwindow */
    tty_sw = window_create(base_frame, TTY,
        WIN_HEIGHT, TTYWINHEIGHT,
        WIN_WIDTH,  TTYWINWIDTH,
    0);

    /* this is a panel subwindow */
    control_panel = window_create(base_frame, PANEL,
        WIN_BELOW,          tty_sw,
        WIN_HEIGHT,         PANELWINHEIGHT,
        WIN_WIDTH,          PANELWINWIDTH,
        WIN_X,              PANELWINX,
        PANEL_ITEM_X_GAP,   PANELXGAP,
    0);

    window_fit(base_frame);
    window_main_loop(base_frame);

    exit(0);
}
```

APPENDIX E

This appendix contains the source code for "getangle.p". Italicized text indicates the corresponding acfg node numbers.

```
program getangle (input,output);
const Pi = 3.1415926536;
var ang,XO,YO,XT,YT,W,L:real;

function Angle(XO,YO,XT,YT,WT,LT:real):real;
var SA,SB,SC,SD,AX,AY, BX,BY, CX,CY, DX,DY: real;
HalfWidth, HalfLength: real;

function Slope (X1,Y1,X2,Y2:real): real;
begin                                          node 1
    Slope := (Y2 - Y1) / (X2 - X1);            node 2
end;
begin                                          node 4
    HalfWidth := WT / 2;                       node 5
    HalfLength := LT / 2;                      node 6
    AX := XT - HalfWidth;                      node 7
    AY := YT + HalfLength;                     node 8
    DX := XT + HalfWidth;                      node 9
    DY := YT - HalfLength;                     node 10
    BX := DX;                                  node 11
    BY := AY;                                  node 12
    CX := AX;                                  node 13
    CY := DY;                                  node 14
    SA := Slope(XO,YO,AX,AY);                  nodes 15 & 16
    SA := arctan(SA);                          nodes 17 & 18
    SD := Slope(XO,YO,DX,DY);                  nodes 19 & 20
    SD := arctan(SD);                          nodes 21 & 22
    SC := Slope(XO,YO,CX,CY);                  nodes 23 & 24
    SC := arctan(SC);                          nodes 25 & 26
    SB := Slope(XO,YO,BX,BY);                  nodes 27 & 28
    SB := arctan(SB);                          nodes 29 & 30
    if ((YO > BY) and (XO > BX)) then          node 31
        Angle := abs(SD - SA)                  nodes 32 & 33
    else if ((XO < CX) and (YO < CY)) then     node 34
        Angle := abs(SA - SD)                  nodes 35 & 36
    else if ((YO > AY) and (XO < AX)) then     node 37
        Angle := abs(SC - SB)                  nodes 38 & 39
    else if ((YO > DY) and (XO > DX)) then     node 40
```

```
        Angle := abs(SB - SC)                    nodes 41 & 42
    else if XO > BX then                         node 43
        Angle := abs(SD + SB)                    nodes 44 & 45
    else if XO < AX then                         node 46
        Angle := abs(SA + SC)                    nodes 47 & 48
    else if (YO >= AY) then                      node 49
        if (XO = AX) then                        node 50
            Angle := Pi/2 - abs(SB)              nodes 51 & 52
        else if (XO < BX) then                   node 53
            Angle := Pi - abs(SA) - abs(SB)      nodes 54, 55 & 56
        else
            Angle := Pi/2 - abs(SA)              nodes 57 & 58
    else if (YO <= CY) then                      node 59
        if (XO = DX) then                        node 60
            Angle := Pi/2 - abs(SC)              nodes 61 & 62
        else if (XO > CX) then                   node 63
            Angle := Pi - abs(SD) - abs(SC)      nodes 64, 65 & 66
        else
            Angle := Pi/2 - abs(SD)              nodes 67 & 68
    else
        Angle := 2 * Pi;                         node 69
end;

begin                                            node 71
    readln(XO,YO,XT,YT,W,L);                     node 72
    ang:= Angle(XO,YO,XT,YT,W,L);                nodes 73 & 74
    writeln('Observed angle is ',ang);           node 75
end.
```

APPENDIX F

This appendix contains a copy of the file "reacher_out".

```
getangle
0
true
1
3
program getangle (input,output);
const Pi = 3.1415926536;
var ang,XO,YO,XT,YT,W,L:real;
Angle
71
70
72
-1
true
false
0


7
72
71
73
-1
true
false
0
  readln(XO,YO,XT,YT,W,L)

2
73
72
74
-1
true
```

```
false
1
Angle
Angle(XO,YO,XT,YT,W,L)

11
74
72
75
-1
true
false
0
  ang:= Angle(XO,YO,XT,YT,W,L);

1
75
73
-1
-1
true
false
0
  writeln('Observed angle is ',ang)

2
Angle
1
true
1
3
function Angle(XO,YO,XT,YT,WT,LT:real):real;
var S^ ^B,SC,SD,AX,AY, BX,BY, CX,CY, DX,DY:
HalfW...l, HalfLength: real;
Slope
4
13
5
-1
true
false
0
```

```
7
5
14
6
-1
true
false
0
  HalfWidth := WT / 2;

1
6
15
7
-1
true
false
0
  HalfLength := LT / 2;

1
7
16
8
-1
true
false
0
  AX := XT - HalfWidth;

1
8
17
9
-1
true
false
0
  AY := YT + HalfLength;

1
9
18
```

```
10
-1
true
false
0
  DX := XT + HalfWidth;

1
10
19
11
-1
true
false
0
  DY := YT - HalfLength;

1
11
20
12
-1
true
false
0
  BX := DX;

1
12
21
13
-1
true
false
0
  BY := AY;

1
13
22
14
-1
true
```

```
false
0
  CX := AX;

1
14
23
15
-1
true
false
0
  CY := DY;

1
15
24
16
-1
true
false
1
Slope
Slope(XO,YO,AX,AY)

11
16
24
17
-1
true
false
0
  SA := Slope(XO,YO,AX,AY);

1
17
25
18
-1
true
false
0
```

```
arctan(SA)

11
18
25
19
-1
true
false
0
  SA := arctan(SA);

1
19
26
20
-1
true
false
1
Slope
Slope(XO,YO,DX,DY)

11
20
26
21
-1
true
false
0
  SD := Slope(XO,YO,DX,DY);

1
21
27
22
-1
true
false
0
arctan(SD)
```

```
11
22
27
23
-1
true
false
0
  SD := arctan(SD);

1
23
28
24
-1
true
false
1
Slope
Slope(XO,YO,CX,CY)

11
24
28
25
-1
true
false
0
  SC := Slope(XO,YO,CX,CY);

1
25
29
26
-1
true
false
0
arctan(SC)

11
26
```

```
29
27
-1
true
false
0
  SC := arctan(SC);

1
27
30
28
-1
true
false
1
Slope
Slope(XO,YO,BX,BY)

11
28
30
29
-1
true
false
0
  SB := Slope(XO,YO,BX,BY);

1
29
31
30
-1
true
false
0
arctan(SB)

11
30
31
31
```

```
-1
true
false
0
  SB := arctan(SB);

1
31
32
32
34
 ((YO > BY) and (XO > BX))
 not ( ((YO > BY) and (XO > BX)) )
0
if  ((YO > BY) and (XO > BX))  then

12
32
33
33
-1
true
false
0
abs(SD - SA)

11
33
34
-1
-1
true
false
0
Angle := abs(SD - SA)

1
34
35
35
37
 ((XO < CX) and (YO < CY))
 not ( ((XO < CX) and (YO < CY)) )
```

```
0
if  ((XO < CX) and (YO < CY))  then

12
35
36
36
-1
true
false
0
abs(SA - SD)

11
36
37
-1
-1
true
false
0
Angle := abs(SA - SD)

1
37
38
38
40
 ((YO > AY) and (XO < AX))
 not ( ((YO > AY) and (XO < AX)) )
0
if  ((YO > AY) and (XO < AX))  then

12
38
39
39
-1
true
false
0
abs(SC - SB)
```

```
11
39
40
-1
-1
true
false
0
Angle := abs(SC - SB)

1
40
41
41
43
 ((YO > DY) and (XO > DX))
 not ( ((YO > DY) and (XO > DX)) )
0
if  ((YO > DY) and (XO > DX))  then

12
41
42
42
-1
true
false
0
abs(SB - SC)

11
42
43
-1
-1
true
false
0
Angle := abs(SB - SC)

1
43
44
```

```
44
46
 XO > BX
 not ( XO > BX )
0
if  XO > BX  then

12
44
45
45
-1
true
false
0
abs(SD + SB)

11
45
46
-1
-1
true
false
0
Angle := abs(SD + SB)

1
46
47
47
49
 XO < AX
 not ( XO < AX )
0
if  XO < AX  then

12
47
48
48
-1
true
```

```
false
0
abs(SA + SC)

11
48
49
-1
-1
true
false
0
Angle := abs(SA + SC)

1
49
50
50
59
 (YO >= AY)
 not ( (YO >= AY) )
0
if  (YO >= AY)  then

12
50
51
51
53
 (XO = AX)
 not ( (XO = AX) )
0
if  (XO = AX)  then

12
51
52
52
-1
true
false
0
abs(SB)
```

```
11
52
53
-1
-1
true
false
0
Angle := Pi/2 - abs(SB)

1
53
54
54
57
 (XO < BX)
 not ( (XO < BX) )
0
if  (XO < BX)  then

12
54
55
55
-1
true
false
0
abs(SA)

11
55
55
56
-1
true
false
0
abs(SB)

11
56
56
```

```
-1
-1
true
false
0
Angle := Pi - abs(SA) - abs(SB)

1
57
57
58
-1
true
false
0
abs(SA)

11
58
58
-1
-1
true
false
0
Angle := Pi/2 - abs(SA)

1
59
59
60
69
 (YO <= CY)
 not ( (YO <= CY) )
0
if  (YO <= CY)  then

12
60
60
61
63
 (XO = DX)
```

```
 not ( (XO = DX) )
0
if  (XO = DX)  then

12
61
61
62
-1
true
false
0
abs(SC)

11
62
62
-1
-1
true
false
0
Angle := Pi/2 - abs(SC)

1
63
62
64
67
 (XO > CX)
 not ( (XO > CX) )
0
if  (XO > CX)  then

12
64
63
65
-1
true
false
0
abs(SD)
```

```
11
65
63
66
-1
true
false
0
abs(SC)

11
66
64
-1
-1
true
false
0
Angle := Pi - abs(SD) - abs(SC)

1
67
65
68
-1
true
false
0
abs(SD)

11
68
66
-1
-1
true
false
0
Angle := Pi/2 - abs(SD)

1
69
67
```

```
-1
-1
true
false
0
    Angle := 2 * Pi;

1
Slope
4
true
0
1
function Slope (X1,Y1,X2,Y2:real): real;
1
10
2
-1
true
false
0


7
2
11
-1
-1
true
false
0
     Slope := (Y2 - Y1) / (X2 - X1);

1
```

APPENDIX G

This appendix contains a copy of the file "walker_out".

```
getangle
0
true
1
3
program getangle (input,output);
const Pi = 3.1415926536;
var ang,XO,YO,XT,YT,W,L:real;
Angle
72
70
74
-1
true
false
0
  readln(XO,YO,XT,YT,W,L)

2
74
72
75
-1
true
false
1
Angle
  ang:= Angle(XO,YO,XT,YT,W,L);

1
75
73
-1
-1
```

```
true
false
0
  writeln('Observed angle is ',ang)

2
Angle
1
true
1
3
function Angle(XO,YO,XT,YT,WT,LT:real):real;
var SA,SB,SC,SD,AX,AY, BX,BY, CX,CY, DX,DY:
HalfWidth, HalfLength: real;
Slope
5
13
6
-1
true
false
0
  HalfWidth := WT / 2;

1
6
15
7
-1
true
false
0
  HalfLength := LT / 2;

1
7
16
8
-1
true
false
0
  AX := XT - HalfWidth;
```

```
1
8
17
9
-1
true
false
0
  AY := YT + HalfLength;

1
9
18
10
-1
true
false
0
  DX := XT + HalfWidth;

1
10
19
11
-1
true
false
0
  DY := YT - HalfLength;

1
11
20
12
-1
true
false
0
  BX := DX;

1
12
21
```

```
13
-1
true
false
0
  BY := AY;

1
13
22
14
-1
true
false
0
  CX := AX;

1
14
23
16
-1
true
false
0
  CY := DY;

1
16
24
18
-1
true
false
1
Slope
  SA := Slope(XO,YO,AX,AY);

1
18
25
20
-1
```

```
true
false
0
  SA := arctan(SA);

1
20
26
22
-1
true
false
1
Slope
  SD := Slope(XO,YO,DX,DY);

1
22
27
24
-1
true
false
0
  SD := arctan(SD);

1
24
28
26
-1
true
false
1
Slope
  SC := Slope(XO,YO,CX,CY);

1
26
29
28
-1
true
```

```
false
0
  SC := arctan(SC);

1
28
30
30
-1
true
false
1
Slope
  SB := Slope(XO,YO,BX,BY);

1
30
31
31
-1
true
false
0
  SB := arctan(SB);

1
31
32
33
34
 ((YO > BY) and (XO > BX))
 not ( ((YO > BY) and (XO > BX)) ) and  ((YO > BY) and (XO > BX))
0
if  ((YO > BY) and (XO > BX))  then

12
33
33
999999
-1
 ((YO > BY) and (XO > BX))
false
0
```

```
Angle := abs(SD - SA)

1
34
35
36
37
 ((XO < CX) and (YO < CY))  and  ((XO < CX) and (YO < CY))
 not ( ((XO < CX) and (YO < CY)) ) and not ( ((YO > BY) and (XO > BX)) )
0
if  ((XO < CX) and (YO < CY))  then

12
36
36
999999
-1
((XO < CX) and (YO < CY))
false
0
Angle := abs(SA - SD)

1
37
38
39
40
 ((YO > AY) and (XO < AX))  and ((YO > AY) and (XO < AX))
 not ( ((YO > AY) and (XO < AX)) ) and ((YO > AY) and (XO < AX))
0
if  ((YO > AY) and (XO < AX))  then

12
39
39
999999
-1
((YO > AY) and (XO < AX))
false
0
Angle := abs(SC - SB)
```

```
1
40
41
42
43
 ((YO > DY) and (XO > DX))  and ((YO > DY) and (XO > DX))
 not ( ((YO > DY) and (XO > DX)) )
0
if  ((YO > DY) and (XO > DX))  then

12
42
42
999999
-1
((YO > DY) and (XO > DX))
false
0
Angle := abs(SB - SC)

1
43
44
45
46
 XO > BX  and XO > BX and YO >= DY and YO <= BY and YO >= DY and
YO <= BY
 not ( XO > BX ) and c YO >= DY and YO <= BY and YO <= BY and YO >=
DY
0
if  XO > BX  then

12
45
45
999999
-1
XO > BX and YO >= DY and YO <= BY
false
0
Angle := abs(SD + SB)

1
```

```
46
47
48
49
 XO < AX  and YO >= CY and YO <= AY
 not ( XO < AX ) and  XO < AX  and YO >= CY and YO <= AY
0
if  XO < AX  then

12
48
48
999999
-1
 XO < AX  and YO >= CY and YO <= AY
false
0
Angle := abs(SA + SC)

1
49
50
50
59
 (YO >= AY)
 not ( (YO >= AY) )
0
if  (YO >= AY)  then

12
50
51
52
53
 (XO = AX)  and  (YO >= AY)
 not ( (XO = AX) ) and (YO >= AY)
0
if  (XO = AX)  then

12
52
52
999999
```

```
-1
 (XO = AX)  and  (YO >= AY)
false
0
Angle := Pi/2 - abs(SB)

1
53
54
56
58
 (XO < BX)  and (XO > AX) and (YO >= AY)
 not ( (XO < BX) ) and (XO > AX) and (YO >= AY)
0
if  (XO < BX)  then

12
56
55
999999
-1
 (XO < BX)  and (XO > AX) and (YO >= AY)
false
0
Angle := Pi - abs(SA) - abs(SB)

1
58
57
999999
-1
(XO=BX) and (YO >= AY)
false
0
Angle := Pi/2 - abs(SA)

1
59
59
60
69
 (YO <= CY)
 not ( (YO <= CY) ) and (YO < AY) and (XO >= AX) and (XO <= BX)
```

```
0
if  (YO < = CY)  then

12
60
60
62
63
 (XO = DX)  and  (YO < = CY)
 not ( (XO = DX) ) and YO < = CY
0
if  (XO = DX)  then

12
62
61
999999
-1
 (XO = DX)  and  (YO < = CY)
false
0
Angle := Pi/2 - abs(SC)

1
63
62
66
68
 (XO > CX)  and (YO < = CY) and (XO < DX)
 not ( (XO > CX) ) and (YO < = CY) and (XO < DX)
0
if  (XO > CX)  then

12
66
63
999999
-1
 (XO > CX)  and (YO < = CY) and (XO < DX)
false
0
Angle := Pi - abs(SD) - abs(SC)
```

```
1
68
65
999999
-1
 not ( (XO > CX) ) and (YO <= CY) and (XO < DX)
false
0
Angle := Pi/2 - abs(SD)

1
69
67
999999
-1
 not ( 'YO <= CY) ) and (YO < AY) and (XO >= AX) and (XO <= BX)
false
0
    Angle := 2 * Pi;

1
999999
1
-1
-1
false
false
0

WALKER added exit node
10
Slope
4
true
0
1
function Slope (X1,Y1,X2,Y2:real): real;
2
10
-1
-1
true
false
```

```
    Slope := (Y2 - Y1) / (X2 - X1);
```

1

APPENDIX H

This appendix contains a copy of the file "getangle.list".

```
program getangle (input,output);
const Pi = 3.1415926536;
var ang,XO,YO,XT,YT,W,L:real;

function Angle(XO,YO,XT,YT,WT,LT:real):real;
var SA,SB,SC,SD,AX,AY, BX,BY, CX,CY, DX,DY:
HalfWidth, HalfLength: real;

function Slope (X1,Y1,X2,Y2:real): real;

begin {Slope when true}
1:
2:      Slope := (Y2 - Y1) / (X2 - X1);
end {Slope}

begin {Angle when true}
4:
5:  HalfWidth := WT / 2;
6:  HalfLength := LT / 2;
7:  AX := XT - HalfWidth;
8:  AY := YT + HalfLength;
9:  DX := XT + HalfWidth;
10:  DY := YT - HalfLength;
11:  BX := DX;
12:  BY := AY;
13:  CX := AX;
14:  CY := DY;
15: Slope(XO,YO,AX,AY)
16:  SA := Slope(XO,YO,AX,AY);
17: arctan(SA)
18:  SA := arctan(SA);
19: Slope(XO,YO,DX,DY)
20:  SD := Slope(XO,YO,DX,DY);
21: arctan(SD)
```

```
22:   SD := arctan(SD);
23: Slope(XO,YO,CX,CY)
24:   SC := Slope(XO,YO,CX,CY);
25: arctan(SC)
26:   SC := arctan(SC);
27: Slope(XO,YO,BX,BY)
28:   SB := Slope(XO,YO,BX,BY);
29: arctan(SB)
30:   SB := arctan(SB);
31: if  ((YO > BY) and (XO > BX))  then
32: abs(SD - SA)
33: Angle := abs(SD - SA)
else
34: if  ((XO < CX) and (YO < CY))  then
35: abs(SA - SD)
36: Angle := abs(SA - SD)
else
37: if  ((YO > AY) and (XO < AX))  then
38: abs(SC - SB)
39: Angle := abs(SC - SB)
else
40: if  ((YO > DY) and (XO > DX))  then
41: abs(SB - SC)
42: Angle := abs(SB - SC)
else
43: if  XO > BX  then
44: abs(SD + SB)
45: Angle := abs(SD + SB)
else
46: if  XO < AX  then
47: abs(SA + SC)
48: Angle := abs(SA + SC)
else
49: if  (YO >= AY)  then
50: if  (XO = AX)  then
51: abs(SB)
52: Angle := Pi/2 - abs(SB)
else
53: if  (XO < BX)  then
54: abs(SA)
55: abs(SB)
56: Angle := Pi - abs(SA) - abs(SB)
else
```

```
57: abs(SA)
58: Angle := Pi/2 - abs(SA)
else
59: if (YO <= CY) then
60: if (XO = DX) then
61: abs(SC)
62: Angle := Pi/2 - abs(SC)
else
63: if (XO > CX) then
64: abs(SD)
65: abs(SC)
66: Angle := Pi - abs(SD) - abs(SC)
else
67: abs(SD)
68: Angle := Pi/2 - abs(SD)
else
69:     Angle := 2 * Pi;
end {Angle}

begin {getangle when true}
71:
72:   readln(XO,YO,XT,YT,W,L)
73: Angle(XO,YO,XT,YT,W,L)
74:   ang:= Angle(XO,YO,XT,YT,W,L);
75:   writeln('Observed angle is ',ang)
end {getangle}
```

APPENDIX I

This appendix contains a copy of the file "spacer_in.l".

```
(setq progname '_getangle_)
(defun _getangle_ (&rest params)
(prog ()
(startup)
(setq routine_names (list 'getangle))
(setq arg_list nil)
(setq _t_arglist nil)
(setq _t_retval nil)
(newvlist 0)
(setq typelist (cons nil typelist))
(setq FAULT_LIST nil)
(setq FAULT_LIST (append FAULT_LIST '((0 line_42 "mis-substitution of SC for SD
in this assignment" Angle1 13 (NE (varval varlist '(W)) 0) (uni_AND (uni_AND (GT
(varval varlist '(YO)) (varval varlist '(DY))) (GT (varval varlist '(XO)) (varval varlist
'(DX)))) (uni_AND (GT (varval varlist '(YO)) (varval varlist '(DY))) (GT (varval varlist
'(XO)) (varval varlist '(DX))))) (varval varlist '(Angle)) "Obs Cond 1") )))
(setq Angle1 nil)
(cond (FAULT_LIST (getfault FAULT_LIST)))
(patom "initialize debug options")
(terpr)
(setq linenum 1) (exec1) (cond (swln (go _t_switch)))
getangle
line_1   (exec 1) (cond (swln (go _t_switch)))
(setq entry_asserts (cons '(t) entry_asserts))
(setq exit_asserts (cons '(t) exit_asserts))
(addvar 'L 'real)
(addvar 'W 'real)
(addvar 'YT 'real)
(addvar 'XT 'real)
(addvar 'YO 'real)
(addvar 'XO 'real)
(addvar 'ang 'real)
(cond (in_verif_mode (initialize_variables t))
    (t (global_init)))
```

```
(cond (static_chain
    (let ((next (car static_chain)))
        (setq static_chain (cdr static_chain))
        (setq swln next)
        (go _t_switch))))
(go getangle.main)
Angle.static_chain
(setq static_chain '( Angle))
(go getangle)
Angle
(set_up_proc 1)

;top of function Angle

(addvar 'LT 'real)
(addvar 'WT 'real)
(addvar 'YT 'real)
(addvar 'XT 'real)
(addvar 'YO 'real)
(addvar 'XO 'real)
(setargs) (setq arg_list nil)
(addvar 'Angle 'real)
(cond ((not at_top_level) (changevar '(Angle) '@Angle)))
(addvar 'DY 'HalfWidth)
(addvar 'DX 'HalfWidth)
(addvar 'CY 'HalfWidth)
(addvar 'CX 'HalfWidth)
(addvar 'BY 'HalfWidth)
(addvar 'BX 'HalfWidth)
(addvar 'AY 'HalfWidth)
(addvar 'AX 'HalfWidth)
(addvar 'SD 'HalfWidth)
(addvar 'SC 'HalfWidth)
(addvar 'SB 'HalfWidth)
(addvar 'SA 'HalfWidth)
(addvar 'HalfLength 'real)
(cond ((not at_top_level) (save_init_values)(do_entry_assertion)
    (init_local_params) (go Angle.exit)))
(cond (in_verif_mode (initialize_variables t)))
(cond (static_chain
    (let ((next (car static_chain)))
        (setq static_chain (cdr static_chain))
        (setq swln next)
```

```
        (go _t_switch))))
(go Angle.main)
(setq entry_asserts (cons '(t) entry_asserts))
(setq exit_asserts (cons '(t) exit_asserts))
Slope.static_chain
(setq static_chain '( Angle Slope))
(go getangle)
Slope
(set_up_proc 2)

;top of function Slope

(addvar 'Y2 'real)
(addvar 'X2 'real)
(addvar 'Y1 'real)
(addvar 'X1 'real)
(setargs) (setq arg_list nil)
(addvar 'Slope 'real)
(cond ((not at_top_level) (changevar '(Slope) '@Slope)))
(cond ((not at_top_level) (save_init_values)(do_entry_assertion)
    (init_local_params) (go Slope.exit)))
(cond (in_verif_mode (initialize_variables t)))
(cond (static_chain
    (let ((next (car static_chain)))
        (setq static_chain (cdr static_chain))
        (setq swln next)
        (go _t_switch))))
(go Slope.main)
(setq entry_asserts (cons '(t) entry_asserts))
(setq exit_asserts (cons '(t) exit_asserts))
Slope.main
(save_init_values)
(do_entry_assertion)
(cond (swln (go _t_switch)))
;begin
line_10 (exec 10) (cond (swln (go _t_switch)))
(changevar '(Slope) '(uni/ (uni- (varval varlist '(Y2)) (varval varlist '(Y1))) (uni- (varval
varlist '(X2)) (varval varlist '(X1)))))
(go Slope.exit)
;end
Slope.exit
(do_exit_assertion)    (cond (swln (go _t_switch)))
(check_routine_exit)(cond (swln (go _t_switch)))
```

```
(set_ret_val '(varval varlist '(Slope)))
(closeproc)
(setq swln (car temp))
(go _t_switch)

Angle.main
(save_init_values)
(do_entry_assertion)
(cond (swln (go _t_switch)))
;begin
line_13 (exec 13) (cond (swln (go _t_switch)))
(changevar '(HalfWidth) '(uni/ (varval varlist '(WT)) 2))
line_15 (exec 15) (cond (swln (go _t_switch)))
(changevar '(HalfLength) '(uni/ (varval varlist '(LT)) 2))
line_16 (exec 16) (cond (swln (go _t_switch)))
(changevar '(AX) '(uni- (varval varlist '(XT)) (varval varlist '(HalfWidth))))
line_17 (exec 17) (cond (swln (go _t_switch)))
(changevar '(AY) '(uni+ (varval varlist '(YT)) (varval varlist '(HalfLength))))
line_18 (exec 18) (cond (swln (go _t_switch)))
(changevar '(DX) '(uni+ (varval varlist '(XT)) (varval varlist '(HalfWidth))))
line_19 (exec 19) (cond (swln (go _t_switch)))
(changevar '(DY) '(uni- (varval varlist '(YT)) (varval varlist '(HalfLength))))
line_20 (exec 20) (cond (swln (go _t_switch)))
(changevar '(BX) '(varval varlist '(DX)))
line_21 (exec 21) (cond (swln (go _t_switch)))
(changevar '(BY) '(varval varlist '(AY)))
line_22 (exec 22) (cond (swln (go _t_switch)))
(changevar '(CX) '(varval varlist '(AX)))
line_23 (exec 23) (cond (swln (go _t_switch)))
(changevar '(CY) '(varval varlist '(DY)))
line_24 (exec 24) (cond (swln (go _t_switch)))
(cond (arg_list (setq _t_arglist arg_list) (setq arg_list nil)))

(setq arg_list (cons '(VAR_REF XO) arg_list))
(setq arg_list (cons '(VAR_REF YO) arg_list))
(setq arg_list (cons '(VAR_REF AX) arg_list))
(setq arg_list (cons '(VAR_REF AY) arg_list))
(setq arg_list (reverse arg_list))
(setq routine_names (cons 'Slope routine_names))
(setq proclist (cons 'f_ret_0 proclist))
(cond (in_verif_mode (setq at_top_level nil)))
(go Slope) f_ret_0
(cond (_t_arglist (setq arg_list _t_arglist) (setq _t_arglist nil))
```

```
    (t (setq arg_list nil)))
(changevar '(SA) '(get_ret_val))
line_25 (exec 25) (cond (swln (go _t_switch)))
(cond (arg_list (setq _t_arglist arg_list) (setq arg_list nil)))

(setq arg_list (cons '(VAR_REF SA) arg_list))
(setq arg_list (reverse arg_list))
(set_ret_val (u_arctan arg_list))
(cond (_t_arglist (setq arg_list _t_arglist) (setq _t_arglist nil))
    (t (setq arg_list nil)))
(changevar '(SA) '(get_ret_val))
line_26 (exec 26) (cond (swln (go _t_switch)))
(cond (arg_list (setq _t_arglist arg_list) (setq arg_list nil)))

(setq arg_list (cons '(VAR_REF XO) arg_list))
(setq arg_list (cons '(VAR_REF YO) arg_list))
(setq arg_list (cons '(VAR_REF DX) arg_list))
(setq arg_list (cons '(VAR_REF DY) arg_list))
(setq arg_list (reverse arg_list))
(setq routine_names (cons 'Slope routine_names))
(setq proclist (cons 'f_ret_1 proclist))
(cond (in_verif_mode (setq at_top_level nil)))
(go Slope) f_ret_1
(cond (_t_arglist (setq arg_list _t_arglist) (setq _t_arglist nil))
    (t (setq arg_list nil)))
(changevar '(SD) '(get_ret_val))
line_27 (exec 27) (cond (swln (go _t_switch)))
(cond (arg_list (setq _t_arglist arg_list) (setq arg_list nil)))

(setq arg_list (cons '(VAR_REF SD) arg_list))
(setq arg_list (reverse arg_list))
(set_ret_val (u_arctan arg_list))
(cond (_t_arglist (setq arg_list _t_arglist) (setq _t_arglist nil))
    (t (setq arg_list nil)))
(changevar '(SD) '(get_ret_val))
line_28 (exec 28) (cond (swln (go _t_switch)))
(cond (arg_list (setq _t_arglist arg_list) (setq arg_list nil)))

(setq arg_list (cons '(VAR_REF XO) arg_list))
(setq arg_list (cons '(VAR_REF YO) arg_list))
(setq arg_list (cons '(VAR_REF CX) arg_list))
(setq arg_list (cons '(VAR_REF CY) arg_list))
(setq arg_list (reverse arg_list))
```

```
(setq routine_names (cons 'Slope routine_names))
(setq proclist (cons 'f_ret_2 proclist))
(cond (in_verif_mode (setq at_top_level nil)))
(go Slope) f_ret_2
(cond (_t_arglist (setq arg_list _t_arglist) (setq _t_arglist nil))
      (t (setq arg_list nil)))
(changevar '(SC) '(get_ret_val))
line_29 (exec 29) (cond (swln (go _t_switch)))
(cond (arg_list (setq _t_arglist arg_list) (setq arg_list nil)))

(setq arg_list (cons '(VAR_REF SC) arg_list))
(setq arg_list (reverse arg_list))
(set_ret_val (u_arctan arg_list))
(cond (_t_arglist (setq arg_list _t_arglist) (setq _t_arglist nil))
      (t (setq arg_list nil)))
(changevar '(SC) '(get_ret_val))
line_30 (exec 30) (cond (swln (go _t_switch)))
(cond (arg_list (setq _t_arglist arg_list) (setq arg_list nil)))

(setq arg_list (cons '(VAR_REF XO) arg_list))
(setq arg_list (cons '(VAR_REF YO) arg_list))
(setq arg_list (cons '(VAR_REF BX) arg_list))
(setq arg_list (cons '(VAR_REF BY) arg_list))
(setq arg_list (reverse arg_list))
(setq routine_names (cons 'Slope routine_names))
(setq proclist (cons 'f_ret_3 proclist))
(cond (in_verif_mode (setq at_top_level nil)))
(go Slope) f_ret_3
(cond (_t_arglist (setq arg_list _t_arglist) (setq _t_arglist nil))
      (t (setq arg_list nil)))
(changevar '(SB) '(get_ret_val))
line_31 (exec 31) (cond (swln (go _t_switch)))
(cond (arg_list (setq _t_arglist arg_list) (setq arg_list nil)))

(setq arg_list (cons '(VAR_REF SB) arg_list))
(setq arg_list (reverse arg_list))
(set_ret_val (u_arctan arg_list))
(cond (_t_arglist (setq arg_list _t_arglist) (setq _t_arglist nil))
      (t (setq arg_list nil)))
(changevar '(SB) '(get_ret_val))
line_32 (exec 32) (cond (swln (go _t_switch)))
;top of if
```

```
(cond ((not (predval (uni_AND (GT (varval varlist '(YO)) (varval varlist '(BY))) (GT
(varval varlist '(XO)) (varval varlist '(BX)))))) (go line_35))
(swln (go _t_switch)))
line_33 (exec 33) (cond (swln (go _t_switch)))
(cond (arg_list (setq _t_arglist arg_list) (setq arg_list nil)))

(setq arg_list (cons '(uni- (varval varlist '(SD)) (varval varlist '(SA))) arg_list))
(setq arg_list (reverse arg_list))
(set_ret_val (u_fabs arg_list))
(cond (_t_arglist (setq arg_list _t_arglist) (setq _t_arglist nil))
    (t (setq arg_list nil)))
(changevar '(Angle) '(get_ret_val))
line_1 (exec 1) (cond (swln (go _t_switch)))
(go Angle.exit)
line_35 (exec 35) (cond (swln (go _t_switch)))
;top of if
(cond ((not (predval (uni_AND (LT (varval varlist '(XO)) (varval varlist '(CX))) (LT
(varval varlist '(YO)) (varval varlist '(CY)))))) (go line_38))
(swln (go _t_switch)))
line_36 (exec 36) (cond (swln (go _t_switch)))
(cond (arg_list (setq _t_arglist arg_list) (setq arg_list nil)))

(setq arg_list (cons '(uni- (varval varlist '(SA)) (varval varlist '(SD))) arg_list))
(setq arg_list (reverse arg_list))
(set_ret_val (u_fabs arg_list))
(cond (_t_arglist (setq arg_list _t_arglist) (setq _t_arglist nil))
    (t (setq arg_list nil)))
(changevar '(Angle) '(get_ret_val))
(go line_1)
line_38 (exec 38) (cond (swln (go _t_switch)))
;top of if
(cond ((not (predval (uni_AND (GT (varval varlist '(YO)) (varval varlist '(AY))) (LT
(varval varlist '(XO)) (varval varlist '(AX)))))) (go line_41))
(swln (go _t_switch)))
line_39 (exec 39) (cond (swln (go _t_switch)))
(cond (arg_list (setq _t_arglist arg_list) (setq arg_list nil)))

(setq arg_list (cons '(uni- (varval varlist '(SC)) (varval varlist '(SB))) arg_list))
(setq arg_list (reverse arg_list))
(set_ret_val (u_fabs arg_list))
(cond (_t_arglist (setq arg_list _t_arglist) (setq _t_arglist nil))
    (t (setq arg_list nil)))
(changevar '(Angle) '(get_ret_val))
```

```
(go line_1)
line_41 (exec 41) (cond (swln (go _t_switch)))
;top of if
(cond ((not (predval (uni_AND (GT (varval varlist '(YO)) (varval varlist '(DY))) (GT
(varval varlist '(XO)) (varval varlist '(DX)))))) (go line_44))
(swln (go _t_switch)))
line_42 (exec 42) (cond (swln (go _t_switch)))
(cond (Angle1
      (setq pc (list 'uni_and pc '(uni_AND (uni_AND (GT (varval varlist '(YO)) (varval
varlist '(DY))) (GT (varval varlist '(XO)) (varval varlist '(DX)))) (uni_AND (GT (varval
varlist '(YO)) (varval varlist '(DY))) (GT (varval varlist '(XO)) (varval varlist '(DX)))))))
      (setq contamlist '((Angle (uni_and (NE (varval varlist '(W)) 0) (uni_AND
(uni_AND (GT (varval varlist '(YO)) (varval varlist '(DY))) (GT (varval varlist '(XO))
(varval varlist '(DX)))) (uni_AND (GT (varval varlist '(YO)) (varval varlist '(DY))) (GT
(varval varlist '(XO)) (varval varlist '(DX)))))))))))
(cond (arg_list (setq _t_arglist arg_list) (setq arg_list nil)))

(setq arg_list (cons '(uni- (varval varlist '(SB)) (varval varlist '(SC))) arg_list))
(setq arg_list (reverse arg_list))
(set_ret_val (u_fabs arg_list))
(cond (_t_arglist (setq arg_list _t_arglist) (setq _t_arglist nil))
      (t (setq arg_list nil)))
(changevar '(Angle) '(get_ret_val))
(go line_1)
line_44 (exec 44) (cond (swln (go _t_switch)))
;top of if
(cond ((not (predval (GT (varval varlist '(XO)) (varval varlist '(BX))))) (go line_47))
(swln (go _t_switch)))
line_45 (exec 45) (cond (swln (go _t_switch)))
(cond (arg_list (setq _t_arglist arg_list) (setq arg_list nil)))

(setq arg_list (cons '(uni+ (varval varlist '(SD)) (varval varlist '(SB))) arg_list))
(setq arg_list (reverse arg_list))
(set_ret_val (u_fabs arg_list))
(cond (_t_arglist (setq arg_list _t_arglist) (setq _t_arglist nil))
      (t (setq arg_list nil)))
(changevar '(Angle) '(get_ret_val))
(go line_1)
line_47 (exec 47) (cond (swln (go _t_switch)))
;top of if
(cond ((not (predval (LT (varval varlist '(XO)) (varval varlist '(AX))))) (go line_50))
(swln (go _t_switch)))
line_48 (exec 48) (cond (swln (go _t_switch)))
```

```
(cond (arg_list (setq _t_arglist arg_list) (setq arg_list nil)))

(setq arg_list (cons '(uni+ (varval varlist '(SA)) (varval varlist '(SC))) arg_list))
(setq arg_list (reverse arg_list))
(set_ret_val (u_fabs arg_list))
(cond (_t_arglist (setq arg_list _t_arglist) (setq _t_arglist nil))
      (t (setq arg_list nil)))
(changevar '(Angle) '(get_ret_val))
(go line_1)
line_50 (exec 50) (cond (swln (go _t_switch)))
;top of if
(cond ((not (predval (GE (varval varlist '(YO)) (varval varlist '(AY))))) (go line_59))
(swln (go _t_switch)))
line_51 (exec 51) (cond (swln (go _t_switch)))
;top of if
(cond ((not (predval (uni_EQ (varval varlist '(XO)) (varval varlist '(AX))))) (go line_54))
(swln (go _t_switch)))
line_52 (exec 52) (cond (swln (go _t_switch)))
(cond (arg_list (setq _t_arglist arg_list) (setq arg_list nil)))

(setq arg_list (cons '(VAR_REF SB) arg_list))
(setq arg_list (reverse arg_list))
(set_ret_val (u_fabs arg_list))
(cond (_t_arglist (setq arg_list _t_arglist) (setq _t_arglist nil))
      (t (setq arg_list nil)))
(changevar '(Angle) '(uni- (uni/ 3.1415926536 2) (get_ret_val)))
(go line_1)
line_54 (exec 54) (cond (swln (go _t_switch)))
;top of if
(cond ((not (predval (LT (varval varlist '(XO)) (varval varlist '(BX))))) (go line_57))
(swln (go _t_switch)))
line_55 (exec 55) (cond (swln (go _t_switch)))
(cond (arg_list (setq _t_arglist arg_list) (setq arg_list nil)))

(setq arg_list (cons '(VAR_REF SA) arg_list))
(setq arg_list (reverse arg_list))
(set_ret_val (u_fabs arg_list))
(cond (_t_arglist (setq arg_list _t_arglist) (setq _t_arglist nil))
      (t (setq arg_list nil)))
(changevar '(Angle) '(uni- 3.1415926536 (get_ret_val)))
(go line_1)
line_57 (exec 57) (cond (swln (go _t_switch)))
(cond (arg_list (setq _t_arglist arg_list) (setq arg_list nil)))
```

```
(setq arg_list (cons '(VAR_REF SA) arg_list))
(setq arg_list (reverse arg_list))
(set_ret_val (u_fabs arg_list))
(cond (_t_arglist (setq arg_list _t_arglist) (setq _t_arglist nil))
      (t (setq arg_list nil)))
(changevar '(Angle) '(uni- (uni/ 3.1415926536 2) (get_ret_val)))
(go line_1)
line_59 (exec 59) (cond (swln (go _t_switch)))
;top of if
(cond ((not (predval (LE (varval varlist '(YO)) (varval varlist '(CY))))) (go line_67))
(swln (go _t_switch)))
line_60 (exec 60) (cond (swln (go _t_switch)))
;top of if
(cond ((not (predval (uni_EQ (varval varlist '(XO)) (varval varlist '(DX))))) (go line_62))
(swln (go _t_switch)))
line_61 (exec 61) (cond (swln (go _t_switch)))
(cond (arg_list (setq _t_arglist arg_list) (setq arg_list nil)))

(setq arg_list (cons '(VAR_REF SC) arg_list))
(setq arg_list (reverse arg_list))
(set_ret_val (u_fabs arg_list))
(cond (_t_arglist (setq arg_list _t_arglist) (setq _t_arglist nil))
      (t (setq arg_list nil)))
(changevar '(Angle) '(uni- (uni/ 3.1415926536 2) (get_ret_val)))
(go line_1)
line_62 (exec 62) (cond (swln (go _t_switch)))
;top of if
(cond ((not (predval (GT (varval varlist '(XO)) (varval varlist '(CX))))) (go line_65))
(swln (go _t_switch)))
line_63 (exec 63) (cond (swln (go _t_switch)))
(cond (arg_list (setq _t_arglist arg_list) (setq arg_list nil)))

(setq arg_list (cons '(VAR_REF SD) arg_list))
(setq arg_list (reverse arg_list))
(set_ret_val (u_fabs arg_list))
(cond (_t_arglist (setq arg_list _t_arglist) (setq _t_arglist nil))
      (t (setq arg_list nil)))
(changevar '(Angle) '(uni- 3.1415926536 (get_ret_val)))
(go line_1)
line_65 (exec 65) (cond (swln (go _t_switch)))
(cond (arg_list (setq _t_arglist arg_list) (setq arg_list nil)))

(setq arg_list (cons '(VAR_REF SD) arg_list))
```

```
(setq arg_list (reverse arg_list))
(set_ret_val (u_fabs arg_list))
(cond (_t_arglist (setq arg_list _t_arglist) (setq _t_arglist nil))
      (t (setq arg_list nil)))
(changevar '(Angle) '(uni- (uni/ 3.1415926536 2) (get_ret_val)))
(go line_1)
line_67 (exec 67) (cond (swln (go _t_switch)))
(changevar '(Angle) '(uni* 2 3.1415926536))
(go line_1)
;end
Angle.exit
(do_exit_assertion)    (cond (swln (go _t_switch)))
(check_routine_exit)(cond (swln (go _t_switch)))
(set_ret_val '(varval varlist '(Angle)))
(closeproc)
(setq swln (car temp))
(go _t_switch)

getangle.main
(save_init_values)
(do_entry_assertion)
(cond (swln (go _t_switch)))
;begin
line_70 (exec 70) (cond (swln (go _t_switch)))
(cond ((eq (tyipeek) NEWLINE) (readc)))
(unisex_read '(XO))
(unisex_read '(YO))
(unisex_read '(XT))
(unisex_read '(YT))
(unisex_read '(W))
(unisex_read '(L))
(cond ((eq (tyipeek) NEWLINE) (readc)))
line_72 (exec 72) (cond (swln (go _t_switch)))
(cond (arg_list (setq _t_arglist arg_list) (setq arg_list nil)))

(setq arg_list (cons '(VAR_REF XO) arg_list))
(setq arg_list (cons '(VAR_REF YO) arg_list))
(setq arg_list (cons '(VAR_REF XT) arg_list))
(setq arg_list (cons '(VAR_REF YT) arg_list))
(setq arg_list (cons '(VAR_REF W) arg_list))
(setq arg_list (cons '(VAR_REF L) arg_list))
(setq arg_list (reverse arg_list))
(setq routine_names (cons 'Angle routine_names))
```

```
(setq proclist (cons 'f_ret_4 proclist))
(cond (in_verif_mode (setq at_top_level nil)))
(go Angle) f_ret_4
(cond (_t_arglist (setq arg_list _t_arglist) (setq _t_arglist nil))
      (t (setq arg_list nil)))
(changevar '(ang) '(get_ret_val))
line_73 (exec 73) (cond (swln (go _t_switch)))
(unisex_write "Observed angle is")
(unisex_write (varval varlist '(ang)))
(terpri)
(go getangle.exit)
;end
getangle.exit
(do_exit_assertion) (cond (swln (go _t_switch)))
(exec 'end) (cond (swln (go _t_switch))
          (t (return)))
toplevelreturn (toplevelrestore)(go _t_switch)
_t_switch (setq _switch swln) (setq swln nil)
(cond ((or (equal _switch 'getangle) (equal _switch "getangle")) (go getangle))
((or (equal _switch 'toplevelreturn) (equal _switch "toplevelreturn")) (go toplevelreturn))
((or (equal _switch 'Angle) (equal _switch "Angle")) (go Angle))
((or (equal _switch 'Angle.static_chain) (equal _switch "Angle.static_chain")) (go
Angle.static_chain))
((or (equal _switch 'Slope) (equal _switch "Slope")) (go Slope))
((or (equal _switch 'Slope.static_chain) (equal _switch "Slope.static_chain")) (go
Slope.static_chain))
((or (equal _switch 'Slope.main) (equal _switch "Slope.main")) (go Slope.main))
((or (equal _switch 'line_10) (equal _switch "line_10")) (go line_10))
((or (equal _switch 'Slope.exit) (equal _switch "Slope.exit")) (go Slope.exit))
((or (equal _switch 'Angle.main) (equal _switch "Angle.main")) (go Angle.main))
((or (equal _switch 'line_13) (equal _switch "line_13")) (go line_13))
((or (equal _switch 'line_15) (equal _switch "line_15")) (go line_15))
((or (equal _switch 'line_16) (equal _switch "line_16")) (go line_16))
((or (equal _switch 'line_17) (equal _switch "line_17")) (go line_17))
((or (equal _switch 'line_18) (equal _switch "line_18")) (go line_18))
((or (equal _switch 'line_19) (equal _switch "line_19")) (go line_19))
((or (equal _switch 'line_20) (equal _switch "line_20")) (go line_20))
((or (equal _switch 'line_21) (equal _switch "line_21")) (go line_21))
((or (equal _switch 'line_22) (equal _switch "line_22")) (go line_22))
((or (equal _switch 'line_23) (equal _switch "line_23")) (go line_23))
((or (equal _switch 'line_24) (equal _switch "line_24")) (go line_24))
((or (equal _switch 'f_ret_0) (equal _switch "f_ret_0")) (go f_ret_0))
((or (equal _switch 'line_25) (equal _switch "line_25")) (go line_25))
```

```
((or (equal _switch 'line_26) (equal _switch "line_26")) (go line_26))
((or (equal _switch 'f_ret_1) (equal _switch "f_ret_1")) (go f_ret_1))
((or (equal _switch 'line_27) (equal _switch "line_27")) (go line_27))
((or (equal _switch 'line_28) (equal _switch "line_28")) (go line_28))
((or (equal _switch 'f_ret_2) (equal _switch "f_ret_2")) (go f_ret_2))
((or (equal _switch 'line_29) (equal _switch "line_29")) (go line_29))
((or (equal _switch 'line_30) (equal _switch "line_30")) (go line_30))
((or (equal _switch 'f_ret_3) (equal _switch "f_ret_3")) (go f_ret_3))
((or (equal _switch 'line_31) (equal _switch "line_31")) (go line_31))
((or (equal _switch 'line_32) (equal _switch "line_32")) (go line_32))
((or (equal _switch 'line_33) (equal _switch "line_33")) (go line_33))
((or (equal _switch 'line_1) (equal _switch "line_1")) (go line_1))
((or (equal _switch 'line_35) (equal _switch "line_35")) (go line_35))
((or (equal _switch 'line_36) (equal _switch "line_36")) (go line_36))
((or (equal _switch 'line_38) (equal _switch "line_38")) (go line_38))
((or (equal _switch 'line_39) (equal _switch "line_39")) (go line_39))
((or (equal _switch 'line_41) (equal _switch "line_41")) (go line_41))
((or (equal _switch 'line_42) (equal _switch "line_42")) (go line_42))
((or (equal _switch 'line_44) (equal _switch "line_44")) (go line_44))
((or (equal _switch 'line_45) (equal _switch "line_45")) (go line_45))
((or (equal _switch 'line_47) (equal _switch "line_47")) (go line_47))
((or (equal _switch 'line_48) (equal _switch "line_48")) (go line_48))
((or (equal _switch 'line_50) (equal _switch "line_50")) (go line_50))
((or (equal _switch 'line_51) (equal _switch "line_51")) (go line_51))
((or (equal _switch 'line_52) (equal _switch "line_52")) (go line_52))
((or (equal _switch 'line_54) (equal _switch "line_54")) (go line_54))
((or (equal _switch 'line_55) (equal _switch "line_55")) (go line_55))
((or (equal _switch 'line_57) (equal _switch "line_57")) (go line_57))
((or (equal _switch 'line_59) (equal _switch "line_59")) (go line_59))
((or (equal _switch 'line_60) (equal _switch "line_60")) (go line_60))
((or (equal _switch 'line_61) (equal _switch "line_61")) (go line_61))
((or (equal _switch 'line_62) (equal _switch "line_62")) (go line_62))
((or (equal _switch 'line_63) (equal _switch "line_63")) (go line_63))
((or (equal _switch 'line_65) (equal _switch "line_65")) (go line_65))
((or (equal _switch 'line_67) (equal _switch "line_67")) (go line_67))
((or (equal _switch 'Angle.exit) (equal _switch "Angle.exit")) (go Angle.exit))
((or (equal _switch 'getangle.main) (equal _switch "getangle.main")) (go getangle.main))
((or (equal _switch 'line_70) (equal _switch "line_70")) (go line_70))
((or (equal _switch 'line_72) (equal _switch "line_72")) (go line_72))
((or (equal _switch 'f_ret_4) (equal _switch "f_ret_4")) (go f_ret_4))
((or (equal _switch 'line_73) (equal _switch "line_73")) (go line_73))
((or (equal _switch 'getangle.exit) (equal _switch "getangle.exit")) (go getangle.exit))
(t (print "can't find target for goto")
```

```
 (print _t_switch)
(terpri)))
)
)
(setq all_routines (list 'getangle 'Angle 'Slope))
(setq NO_REALS nil)
```

LIST OF REFERENCES

Adrion, R. W., Branstad, M. A., and Cherniavsky, J. C., "Validation, Verification, and Testing of Computer Software," *ACM Computing Surveys*, June 1982, pp. 159-192.

Beizer, B., *Software Testing Techniques*, Van Nostrand Reinhold, 1983.

Beizer, B., *Software System Testing and Quality Assurance*, Van Nostrand Reinhold, 1984.

Beer, S., and others, *A Sun User's Guide*, MacMillan Education Ltd, 1989.

Bolchoz, J. M., *The Identification of Software Failure Regions*, Master's Thesis, Naval Postgraduate School, Monterey, CA, June 1990.

Brooke, J.B. and Duncan, K.D., "Experimental Studies of Flowchart Use at Different Stages of Program Debugging" (*Ergonomics*, Vol 23, No 11, 1980, pages 1057 - 1091), *Human Factors in Software Development*, IEEE Computer Society Press, 1981, pp. 328-357.

Brown, J. R. and Cunningham, S., *Programming the User Interface: Principles and Examples*, John Wiley and Sons, 1989.

Curtis, B. (editor), *Human Factors in Software Development*, IEEE Computer Society Press, 1981.

Fischer, G., "Human-Computer Interaction Software: Lessons Learned, Challenges Ahead," *IEEE Software*, January 1989, pp. 44-52.

Fisher, A. S., *CASE Using Software Development Tools*, John Wiley & Sons, Inc., 1988.

Fitter, M. and Green, T.R.G., "When Do Diagrams Make Good Computer Languages?" (*International Journal of Man-Machine Studies*, Vol 11, 1979, pages 235 - 261), *Human Factors in Software Development*, IEEE Computer Society Press, 1981, pp. 358-379.

Frankl, P., *ASSET Reference Manual*, New York University, 1987.

Green, T.R.G., Sims, M.E., and Fitter, M.J., "The Problems the Programmer Faces" (*Ergonomics*, Vol 23, No 9, 1980, pages 893 - 907), *Human Factors in Software Development*, IEEE Computer Society Press, 1981, pp. 125-137.

Lewis, T. G. and Oman, P. W., "The Challenge of Software Development," *IEEE Software*, November 1990, pp. 9-12.

Lutz, M., "Testing Tools," *IEEE Software*, May 1990, pp. 53-57.

Ramamoorthy, C.V., and Ho, S.F., "Testing Large Software With Automated Software Evaluation Systems," *IEEE Transactions on Software Engineering*, Vol. SE-1, No. 1, March 1975, pp. 46-58.

Schneiderman, B., *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, Addison-Wesley Publishing Company, 1987.

Shimeall, T., "FALTER - A Fault Annotation Tool," Technical Report NPS52-89-051, Naval Postgraduate School, Monterey, CA, September 1989.

Shimeall, T., "REACHER - A Reachability Condition Derivation Tool," Technical Report NPS52-89-050, Naval Postgraduate School, Monterey, CA, September 1989.

Shimeall, T., "VIEWER - A User Interface for Failure Region Analysis," Naval Postgraduate School, Monterey, CA, 27 September 1989.

SunView 1 Programmer's Guide, Sun Microsystems, Inc., Revision A of 9 May 1988.

BIBLIOGRAPHY

Adrion, R. W., Branstad, M. A., and Cherniavsky, J. C., "Validation, Verification, and Testing of Computer Software," *ACM Computing Surveys*, June 1982, pp. 159-192.

Beer, S., and others, *A Sun User's Guide*, MacMillan Education Ltd, 1989.

Beizer, B., *Software Testing Techniques*, Van Nostrand Reinhold, 1983.

Beizer, B., *Software System Testing and Quality Assurance*, Van Nostrand Reinhold, 1984.

Bolchoz, J.M. *The Identification of Software Failure Regions*, Master's Thesis, Naval Postgraduate School, Monterey, CA, June 1990.

Brooke, J.B. and Duncan, K.D., "Experimental Studies of Flowchart Use at Different Stages of Program Debugging", (*Ergonomics*, Vol 23, No 11, 1980, pages 1057 - 1091), *Human Factors in Software Development*, IEEE Computer Society Press, 1981, pp. 328-357.

Brown, J. R. and Cunningham, S., *Programming the User Interface: Principles and Examples*, John Wiley and Sons, 1989.

Curtis, B. (editor), *Human Factors in Software Development*, IEEE Computer Society Press, 1981.

DeMillo, R.A., Guindi, D.S., McCracken, W.M., Offutt, A.J., and King, K.N., "An Extended Overview of the Mothra Software Testing Environment," *Proceedings Second Workshop on Software Testing, Verification, and Analysis*, July 1988.

Fischer, G., "Human-Computer Interaction Software: Lessons Learned, Challenges Ahead," *IEEE Software*, January 1989, pp. 44-52.

Fisher, A. S., *CASE Using Software Development Tools*, John Wiley & Sons, Inc., 1988.

Fitter, M. and Green, T.R.G., "When Do Diagrams Make Good Computer Languages?" (*International Journal of Man-Machine Studies*, Vol 11, 1979, pages 235 - 261), *Human Factors in Software Development*, IEEE Computer Society Press, 1981, pp. 358-379.

Frankl, P., *ASSET Reference Manual*, New York University, 1987.

Green, T.R.G., Sims, M.E., and Fitter, M.J., "The Problems the Programmer Faces" (*Ergonomics*, Vol 23, No 9, 1980, pages 893 - 907), *Human Factors in Software Development*, IEEE Computer Society Press, 1981, pp. 125-137.

Heller, M., "Graphical User Interfaces. Hype No Longer," *Computer Shopper*, September 1990, pp. 121-158.

King, J. C., "A New Approach to Program Testing," *Proceedings of the International Conference on Reliable Software*, Vol.10, No.6, 1975, pp. 228-233.

Lutz, M., "Testing Tools," *IEEE Software*, May 1990, pp. 53-57.

Ramamoorthy, C.V., and Ho, S.F., "Testing Large Software With Automated Software Evaluation Systems," *IEEE Transactions on Software Engineering*, Vol. SE-1, No. 1, March 1975, pp. 46-58.

Schneiderman, B., *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, Addison-Wesley Publishing Company, 1987.

Shimeall, T., "FALTER - A Fault Annotation Tool," Technical Report NPS52-89-051, Naval Postgraduate School, Monterey, CA, September 1989.

Shimeall, T., "REACHER - A Reachability Condition Derivation Tool," Technical Report NPS52-89-050, Naval Postgraduate School, Monterey, CA, September 1989.

Shimeall, T., "VIEWER - A User Interface for Failure Region Analysis," Naval Postgraduate School, Monterey, CA, 27 September 1989.

SunView 1 Programmer's Guide, Sun Microsystems, Inc., Revision A of 9 May 1988.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	LCDR Vicki Sue Abel c/o W. M. Abel, Jr. 509 Hartwood Road Hartwood, VA 22405-4104	2
4.	CAPT Medio Monti 178 Baker Drive Pittsburgh, PA 15237	2
5.	Timothy Shimeall Naval Postgraduate School Code CS/Sm, Department of Computer Science Monterey, CA 93943-5100	3
6.	LCDR Rachel Griffin Naval Postgraduate School Code CS/Gr, Department of Computer Science Monterey, CA 93943-5100	1
7.	Robert B. McGhee Chairman, Department of Computer Science Naval Postgraduate School Monterey, CA 93943-5100	1
8.	CDR Thomas J. Hoskins Code 37, Curricular Officer, Computer Technology Curriculum Naval Postgraduate School Monterey, CA 93943-5100	1

9. Commandant of the Marine Corps 1
Code TE 06
Headquarters, U.S. Marine Corps
Washington, D.C. 20380-0001