View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Calhoun, Institutional Archive of the Naval Postgraduate School

NPS52-89-050

NAVAL POSTGRADUATE SCHOOL

Monterey, California

REACHER -- A Reachability Condition Derivation Tool

Timothy J. Shimeall

September 1989

Approved for public release; distribution is unlimited.

Prepared for:

Naval Postgraduate School
Monterey, California 93943

FedDocs
D 208.14/2
NPS-52-89-050

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA 93943-5002

Technical
D 202.14/2:
NPS-52-87-050

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral R. W. West, Jr.
Superintendent

Harrison Shull
Provost

This report was prepared in conjunction with research funded by the Naval Postgraduate School Research Council.

Reproduction of all or part of this report is authorized.

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

REPORT DOCUMENTATION PAGE

REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited
DECLASSIFICATION/DOWNGRADING SCHEDULE	
ERFORMING ORGANIZATION REPORT NUMBER(S): NPS52-89-050	5. MONITORING ORGANIZATION REPORT NUMBER(S)
NAME OF PERFORMING ORGANIZATION: mputer Science Dept. val Postgraduate School; 6b. OFFICE SYMBOL (if applicable): 52	7a. NAME OF MONITORING ORGANIZATION: Naval Postgraduate School
ADDRESS (City, State, and ZIP Code): nterey, CA 93943	7b. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943
NAME OF FUNDING/SPONSORING ORGANIZATION: val Postgraduate School; 8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: O&MN direct funding
ADDRESS (City, State, and ZIP Code): nterey, CA 93943	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

TITLE (Include Security Classification)
ACHER -- A Reachability Condition Derivation Tool (U)

PERSONAL AUTHOR(S)
IMEALL, Timothy J.

TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
rogress	FROM 9/88 TO 9/89	September 1989	14

SUPPLEMENTARY NOTATION

COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
IELD	GROUP	SUB-GROUP	Software Testing, Statement Coverage, Reachability Analysis, Failure Regions

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

ACHER is a tool that derives the conditions under which each program block in a Pascal program, ocedure or function may be executed (i.e., the reachability conditions for each subprocedure, subfunction and gin-end block). The tool shall accept compilable Pascal program source code and shall produce both an notated listing and an augmented control flow graph.

ACHER is one of a series of four tools that work in an integrated fashion to analyze Pascal programs to termine the failure regions associated with identified faults in the programs. The augmented control flow aph produced by REACHER will used as input by the programs FALTER and SPACER, and shall be stomized for such usage. The annotated source listing provides includes a correspondence between Pascal tements and control flow graph nodes. The users may access REACHER, FALTER and SPACER through screen-oriented user interface called VIEWER. This document describes the operation of REACHER and direct user interface.

DISTRIBUTION/AVAILABILITY OF ABSTRACT: ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED
NAME OF RESPONSIBLE INDIVIDUAL: Shimeall, Timothy J.	22b. TELEPHONE (Include Area Code): (408) 646-2509; 22c. OFFICE SYMBOL: 52Sm

Environment for Failure Region Analysis:

REACHER -- A Reachability Condition Derivation Tool

Timothy Shimeall

Computer Science Department (Code 52Sm)
Naval Postgraduate School
Monterey CA 93943

25 September 1989

Table of Contents

List of Tables

List of Figures

1.0 Introduction: REACHER -- A Reachability Condition Derivation Tool

REACHER is a tool that derives the conditions under which each program block in a Pascal program, procedure or function may be executed (i.e., the reachability conditions for each subprocedure, subfunction and begin-end block). The tool shall accept compilable Pascal program source code and shall produce both an annotated listing and an augmented control flow graph.

The intended users of REACHER are experienced research personnel familiar with the theory of reachability conditions and their derivation. REACHER shall support interaction with the user to guide the process of derivation of the reachability conditions. The user shall be asked to perform those parts of the derivation for which automation is difficult. To support the user participation, REACHER shall include functionality to provide for annotation of the code with summary comments.

REACHER is one of a series of four tools that work in an integrated fashion to analyze Pascal programs to determine the failure regions associated with identified faults in the programs. The augmented control flow graph produced by REACHER will used as input by the programs FALTER and SPACER, and shall be customized for such usage. The annotated source listing provides includes a correspondence between Pascal statements and control flow graph nodes. The users may access REACHER, FALTER and SPACER through a screen-oriented user interface called VIEWER. Since the process of reachability condition generation may take substantial time, REACHER shall support saving and recovering of partial derivations. Figure 1 provides a context diagram for this use of REACHER.

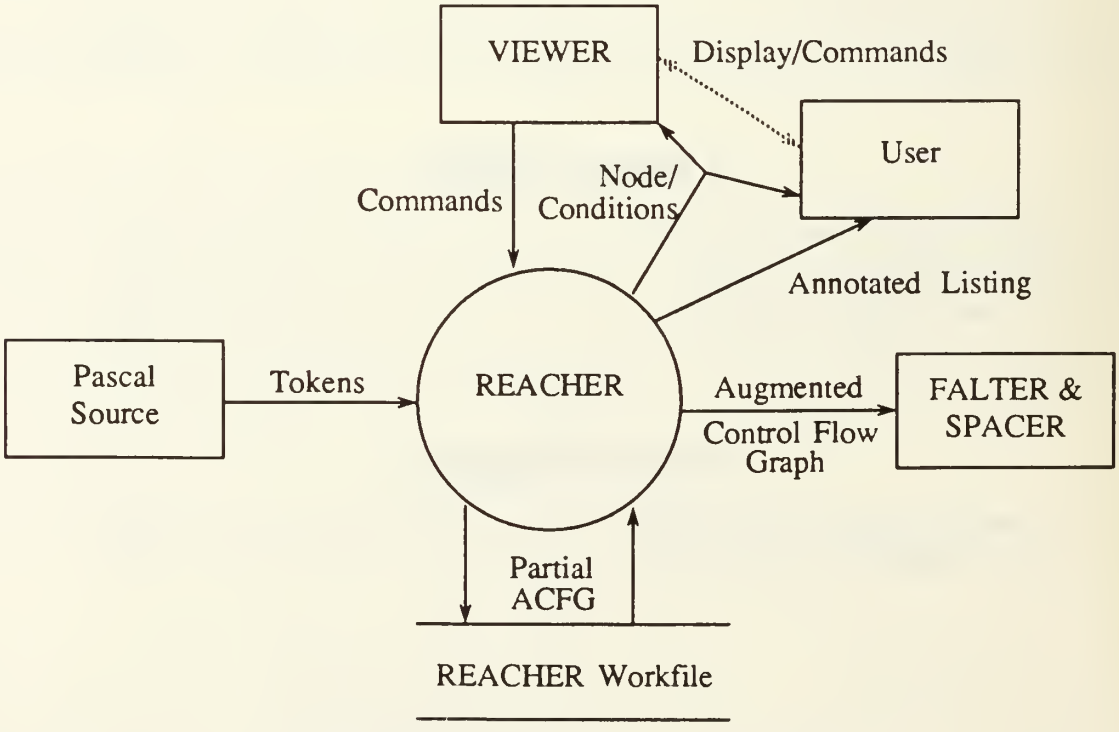

Figure 1: Context Diagram for REACHER

REACHER may also be independently used to support techniques that employ reachability conditions of selected parts of a piece of software, such as code reading, structural test planning and static analysis.

REACHER shall be written in C for use under UNIX 4.3 BSD. Future versions may be transported to other operating systems and versions of BSD. Future versions may also be constructed that deal with other input languages, in particular Ada (trademark, DoD AJPO).

This document contains all requirements for REACHER. Section 2 is a description of the input and output data for REACHER. The output data is described using two conventions. The annotations for the source listing are described using a BNF-style description, with non-terminals in *italics*, terminals in **bold**, explanations of non-terminals in normal print and alternatives definitions are indicated by the vertical bar '|'. The augmented control flow graph and node/condition output data are described in a record-style format.

Section 3 is a list of all of the functional requirements, including a description of the response to each possible program input. Terms found in the Glossary are !delimited by exclamation points!. /Input variables/ are delimited by slashes. //Output variables// or portions thereof are delimited by doubled slashes. $Symbolic Value References$ are delimited by dollar signs. In this section, the verb "shall" is used to indicate required behaviors for REACHER. The verbs "will" or "is" is used to indicate necessary or desirable actions that occur beyond the control of REACHER (e.g., user actions). The verb "may" is used to indicate optional or alternative actions.

Section 4 identifies all acceptable subsets and forseen supersets(extentions) to the basic functionality described in sections 2 and 3.

Section 5 identifies the forseen undesired events that may occur during REACHER's execution and describes responses to these undesired events. Omitted from this section are events that may occur during REACHER's execution, but that REACHER cannot respond to. Duplicatively included in this section are all error messages produced by REACHER and the conditions under which REACHER will generate these messages.

Section 6 is a glossary of defined terms used in this document. In the text of this document, each defined term appears delimited by exclamation points. These defined terms may be looked upon as text macros, and these terms should be read in context.

2.0 Data Descriptions

Input

Pascal source code (c.f. Jensen & Wirth, *Pascal User Manual and Report*)

Output

1. Annotated program source
   Annotations:
   - Graph Node Indicators (//NodeNum//) ::= {**N** *Number* }
   - Reachability condition (//ReachCond//) ::= {**RC** *ReachCond*}

     ReachCond ::= *Calling-cond* | *Calling-cond* **or** *ReachCond*

     Calling-cond ::= *Var-cond* | *Var-cond* **and** *Calling-cond*

     Var-cond ::= **true** | **false** | *v* = *Value* | *v* **in** [*Value-list*] | *v* < *Value* | *v* > *Value*

     Value-list ::= *List-element* , *Value-list* | *List-element*

     List-element ::= *Value* | *Value* .. *Value*

     Value ::= *d* | (*Expression*) | *Literal*

     v, *d* ::= Pascal variable reference

     Expression ::= Pascal expression

     Literal ::= Pascal literal of type corresponding to *v*
   - Summary Annotation (//Summary//) ::= {S *comment* }

     comment ::= Pascal comment without delimters

2. Augmented Control Flow Graph (ACFG)

   1. ACFG Header Info (//ACFGHDR//)

Field	Acronym	Value
Number of Graphs	//AHLEN//	Integer
Graph Data	//AHPROCS//	List of //ABKHDR//
Program Name	//AHPGRM//	String

   2. ACFG Block Header Info (//ABKHDR//)

Field	Acronym	Value
Block Name	//ABKNAME//	String
Number of Return Locations	//ABKNUMRET//	Integer
Return Locations	//ABKRET//	List of //ACFG//
Entry Conditions	//ABKREACH//	//ReachCond//
Block Nodes	//ABKGRPH//	//ACFG//
Number of Subsidiary Blocks	//ABKNSUBS//	Integer
Subsidiary Blocks	//ABKSUBS//	List of //ABKHDR//
Declaration Text	//ABKDECL//	String

3. Augnmentd Control Flow Graph Nodes (//ACFG//)

Field	Acronym	Value
Node Number	//ACFGNUM//	Integer
Left Child	//ACFGLFT//	//ACFG//
Right Child	//ACFGRT//	//ACFG//
Left Condition	//ACFGLCND//	//ReachCond//
Right Condition	//ACFGRCND//	//ReachCond//
Number of Proc/Funct calls	//ACFGNCALLS//	Integer
Procedure/Function calls	//ACFGCALLS//	List of //ABKHDR//
Line Text	//ACFGTEXT//	String
Comment Text	//ACFGSUMM//	String
Node Type	//ACFGTYPE//	$None$, $Assign$, $Call$, If, $Loop$, $Case$, $With$, $BeginEnd$, $Goto$, $Other$

3. Partial ACFG -- //ACFGHDR// extended to include Integer //ACFGCUR//, with value encoded:

Value	Meaning
0	Link conditions are branch conditions
1..//AHLEN//	Reachability conditions derived for ACFG entries 1 through //ACFGCUR// (partial derivation)
> //AHLEN//	Derivation complete

4. Node/Condition Prompts (//NodeCond//)

Field	Acronym	Value
Node Identification	//NCNum//	Integer
Node Reachability Condition	//NCCond//	//ReachCond//
Node Branch Condition	//NCBranch//	//ReachCond//
Node Statement text	//NCText//	String

3.0 Functional Requirements

3.1 Initial Processing

3.1.1 Overview

On program initialization, REACHER shall expect the name of a file (/InFile/) to be passed as an argument, along zero or more execution options. REACHER's response to the options and use of /InFile/ are described in Table 1 below. Should the file named by /InFile/ not exit or not be readable by REACHER, then REACHER shall display the message: File not fond and exit.

Option String	Response
r	!ReadWorkFile!
o /OutFile/	//ResultFile// is set to /OutFile/
not r	!GraphGen!
not o	//ResultFile// is set to /InFile/

Table 1 -- REACHER Option Processing

In overview, these options shall lead REACHER to parse a Pascal source file into an initial ACFG, or to read in a previously generated partial ACFG, and to set the name to be used for the result files generated by REACHER. Once the current (inital or partial) ACFG is set up, REACHER shall traverse the graph, interactively constructing the reachability conditions for each block as described in section 3.2.

3.1.2 !GraphGen! -- Graph Generation

To generate the intial ACFG, REACHER shall parse the Pascal program, procedure or function located in the file named by /InFile/. REACHER shall respond to each section of the input as described in Table 2: REACHER does not require information on label, const, type or var declarations for its processing, but since REACHER must produce an annotated listing of its input, these unused portions of each block are saved in the //ABKDECLS// string. Note that the actions specified in this table imply that REACHER shall handle files of multiple independent procedures and functions without an enclosing program. However, no label, const, type or var declarations global to any such procedures or functions are permittted. If such declarations are detected, REACHER shall display the message: Invalid global declaration and exit. If the input is a set of procedures and functions as opposed to a program, then all references to the 'main program' in the document that follows should be interpreted by the reader as indicating the first non-nested procedure or function in the input. Should the input file not contain at least one complete scope, REACHER shall display the message: Incomplete input file and exit.

3.1.3 !ReadWorkFile! -- Graph Restoration

To restore a saved partial ACFG, REACHER shall read the workfile named by /Infile/. The format of this workfile is given in section 3.3. Should the file not be a complete

Input Structure	Response
Start of Input File	//AHLEN// shall be set to 1, //AHPGRM// shall be set to /InFile/, //AHPROCS// shall be set to a newly allocated //ABKHDR//, in that //ABKHDR//, !NewABK!, /NodeCnt/ shall be set to 1 and //ABKNAME// shall be set to an empty string.
Program header	//ABKNAME// shall be set to the program name, //ABKDECL// shall be set to contain the text of the Program header and //ABKGRPH// shall be set to a newly allocated //ACFG// and in the new //ACFG// !NewACFG!.
Label, Const, Type, or Var Declaration	If //ABKDECL// is empty then //ABKDECL// shall be set to contain the text of the declaration, otherwise the declaration shall be appended to //ABKDECL//.
Procedure or Function Declaration	If the first (or only) //ABKNAME// in //AHPROCS// is empty then !SetABK!. If the procedure or function is nested with the scope named by //ABKNAME// in any entry of //AHPROCS// then for that entry of //AHPROCS// //ABKNSUBS// shall be incremented, a newly allocated //ABKHDR// shall be linked into //ABKSUBS// and in the new //ABKHDR// !NewABK! and !SetABK!. If the procedure or function is not nested within the scope named by //ABKNAME// in any entry of //AHPROCS// then //AHLEN// shall be incremented, a newly allocated //ABKHDR// shall be linked into //AHPROCS// aand in the new //ABKHDR// !NewABK! and !SetABK!.
Begin	If //ABKGRPH// for the current block contains only one node, then no change shall be made to //ABKGRPH// and parsing shall continue. Otherwise a newly allocated //ACFG// shall be linked into //ABKGRPH// at /CurNode/ and in the new //ACFG// !NewACFG!.
If	!MakeNode(If)!.
Else	/CurNode/ shall be set to reference the Right child of the if node allocated most recently at the current nesting level.
While	!MakeNode($Loop$)!.
For	!MakeNode($Loop$)!.
Repeat	!MakeNode($Loop$)!.
Until	!MakeNode($Loop$)!, and set //ACFGRCHILD// to reference the most recently parsed Repeat at the current nesting level.

Table 2 -- REACHER Parsing Actions

Input Structure	Response
Case	!MakeNode($Case$)!
With	!MakeNode($With$)!
:=	!MakeNode($Assign$)!
Goto	Complain and exit.
Procedure call	!MakeNode($Call$)!, Increment //ACFGNCALLS// and add in //ACFGCALLS// a reference to the //ABKHDR// corresponding to the procedure being called.
Function call	Increment //ACFGNCALLS// in the most recently allocated //ACFG// and add in //ACFGCALLS// a reference to the //ABKHDR// corresponding to the function being called.
All other tokens	No change shall be made to //ACFG// and parsing shall continue.
End of Input File	/CurNode/ shall be set to the first //ACFG// in the //ABKGRPH// of the //ABKHDR// of the main program.

Table 2 -- REACHER Parsing Actions (Continued)

and consistent set of headers and //ACFG// REACHER shall display the message: Invalid workfile format and prompt for a Pascal source file to regenerate the workfile. Once the data is read in, /CurNode/ shall be set to the first //ACFG// in the //ABKGRPH// of the entry of //AHPROCS// corresponding to //ACFGCUR//. If no such //ACFG// exists, REACHER shall display the message: Null workfile and exit.

3.2 Program Traversal

3.2.1 Overview

Starting with the program main body, REACHER shall traverse the input in a depth-first manner, annotating each procedure and function with the conditions under which that procedure or function may be called (i.e.,. reachability conditions, henceforth //ReachCond//). For each procedure or function, the //ReachCond// shall be the //ReachCond// of its caller, combined with additional conditions due to control structure of the caller using a logical **and**. See Table 3 for a description of the conditions associated with each Pascal control structure. If there is more than one caller, the partial //ReachCond// clauses from each shall be combined with a logical **or** to produce the full //ReachCond//. The initial //ReachCond// (for the main body) is **true**. Should a procedure be called that is not part of the input file supplied to REACHER, the tool will display the message: Missing procedure *name* and procede with processing.

3.2.2 User Commands

During the program traversal the user shall be prompted with the //NodeCond// information for each //ACFG// as REACHER processes it. When prompted, the user may enter any of the commands described in table 4. REACHER shall perform the action described as a response to each command as it is entered. Shold the user enter a command that is not listed in table 4, REACHER shall display the message: No such command and prompt the user again. Should the user enter a command listed in table 4 without the listed arguments, REACHER shall display the message: Missing command arguments and prompt the user again, ignoring the partial command. Should the user enter a command with more arguments than those listed in table 4, REACHER shall display the message Ignoring *string* at end of command, where *string* is a list of the extra arguments, and proceed to follow the command, ignoring the extra arguments. Should the user enter a command with arguments that are not of the appropriate type as listed in table 4, REACHER shall display the message: Invalid arguments to command and prompt the user again, ignoring the attempted command.

Structure	Affect on //ReachCond//
CALL:	In the called procedure/function //ReachCond// shall be set to called-//ReachCond// **or** (calling-//ReachCond// **and** !parameter assignments!).
IF:	(//ReachCond// **and** !if condition!) shall be used as the initial //ReachCond// to process the **then** code. *Result1* will represent the result of processing the **then** code, (//ReachCond// **and not** !f condition!) shall be used as the initial //ReachCond// to process the **else** code (if any). *Result2* will represent the result of processing the **else** code. (*Result1* **or** *Result2*) shall be used for processing after the **if** statement.
CASE:	(//ReachCond// **and** !case condition!) shall be used to process each case body. A condtion formed by the logical **or** of all case body results shall be used for further processing.
LOOPS:	(//ReachCond// **and** !loop invariant!) shall be used to process the loop body, getting !loop invariant! from the user. (//ReachCond// **and** !oop invarian! **and not** !loop condition!) shall be used for further processing.
BEGIN-END:	Each enclosed statment shall be processed in turn, with the results of the processing combind by **and** with the current //ReachCond// for further processing.
ASSIGNMENT:	All occurences in the //ReachCond// of the expression on the right-hand side shall be replaced with the variable on the left-hand side. If no occurences, (//ReachCond//.**and** !assign condition!) shall be used for further processing.
WITH:	No effect on //ReachCond//
GOTO:	REACHER shall complain to the user and exit

Table 3 -- Control Structure Conditions

Command	Response
!Return!, l or n	REACHER shall process the left child of the current node. If the left child is null, the right child shall be processed. (left or next)
a *sting*	REACHER shall set //ACFGSUMM// of /CurNode/ to *string*. (annotate)
c //ReachCond//	REACHER shall replace the //ReachCond// for the arc traversed to reach this node with the //ReachCond// entered by the user. (change)
g *procname*	REACHER shall continue processing with the //ABKGRPH// in the //ABKHDR// corresponding to the scope named *procname*, which must be visible at the current scope or this command shall be ignored. (goto)
j *node1 node2*	REACHER shall merge the nodes in //ABKGRPH// with node numbers *node1* and *node2*. The children of the joined node shall be the children of the node indicated by *node2*. The //ACFGTEXT// of the joined node shall be the concatenation of the //ACFGTEXT//s of *node1* and *node2*. The //ACFGNUM// of the joined node shall be *node2*. The //ACFGSUMM// of the joined node shall be the concatenation of //ACFGSUMM// for *node1*, the string 'and' and //ACFGSUMM// for *node2*. //ACFGNCALLS// of the joined node shall be the sum of the //ACFGNCALLS// values for the two nodes. //ACFGCALLS// for the joined node shall be the concatenation of the //ACFGCALLS// for the two nodes. //ACFGTYPE// for the two nodes shall be $Other$.(join)
p	REACHER shall set /CurNode/ to the node processed immediately prior to the current node and continue processing. (previous)
q	REACHER shall request confirmation, and if affirmation is given REACHER shall terminate processing without saving //ACFGHDR//. If the immediately prior command was 's' then the confirmation step shall be omitted. (quit)
r	REACHER shall process the right child of the current node. If the right child is null, the left child shall be processed(right)
s *filename*	REACHER shall store //ACFGHDR// and all its subordinates in the file named *failename*. This stored //ACFGHDR// shall be stored as a Partial ACFG. If this file may not be written on, REACHER shall display the message: Cannot write save file. (save)
u	REACHER shall undo the latest change. (undo)

Table 4 -- User Commands

The flow of REACHER execution is diagrammed by figure 2

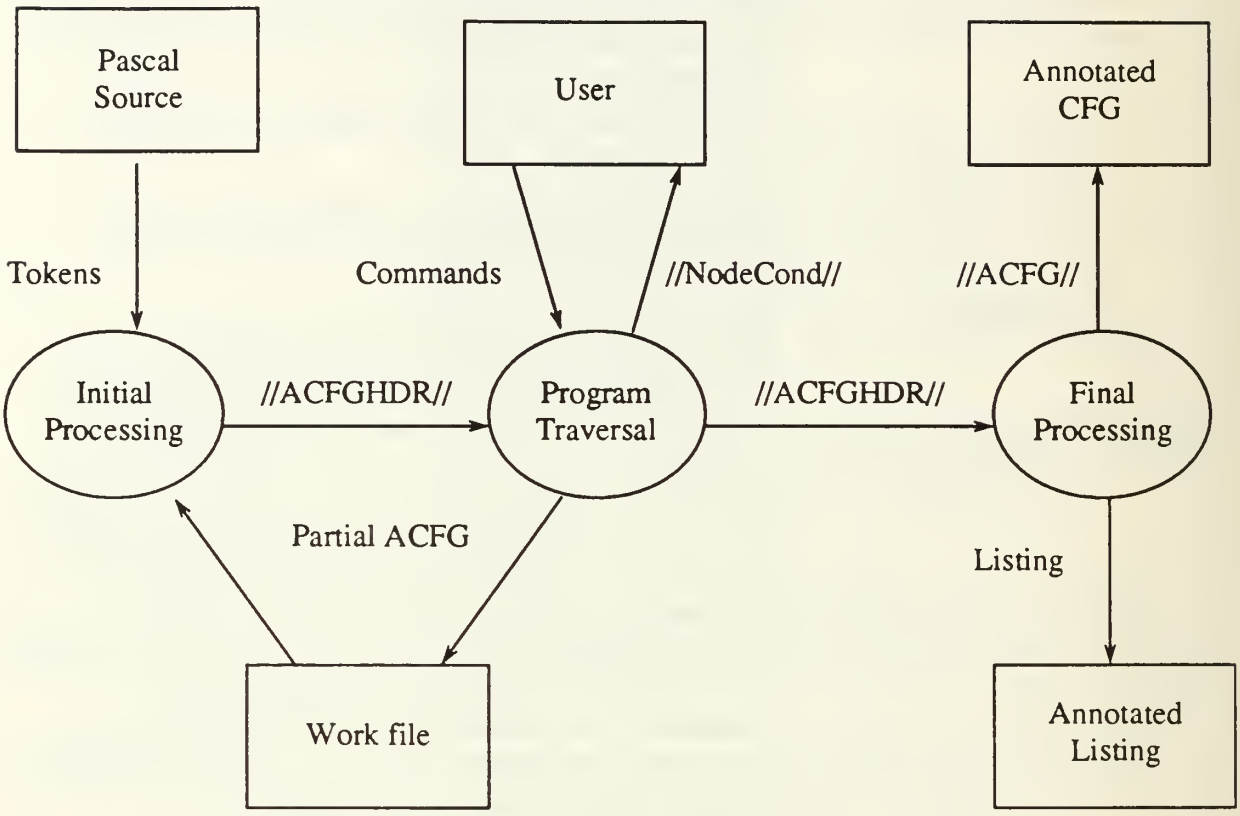

Figure 2: REACHER Flow of Execution

3.3 Final Processing

When all nodes in each //ACFG// in //ACFGHDR// have been processed, REACHER shall generate two output files, one containing the annotated program souce described in section 2, and the other containing //ACFGHDR// and all of its subordinates. The names of these files shall correspond to those specified in section 3.1. Should either of these files not be accessible for wirte, REACHER shall display the message: Cannot write result file and prompt the user for a new file name.

The second file shall be a normal UNIX character file, formatted such that all of the fields of //ACFGHDR// appear first, then all of the fields of each //ABKHDR// in the order that they appear in //ACFGHDR//, then all of the fields of each //ACFG// in each //ABKHDR//, with node numbers acting as links between nodes of the //ACFG//. This format will be expected by FALTER as its input format and shall not be modified without appropriate modifications to FALTER and its specifications.

4.0 Subsets and Supersets

4.1 Subsets

1. Eliminate the 'join' command
1. Eliminate the 'previous' command

4.2 Supersets

1. Expand the 'undo' command to beyond the last change
2. Implement a node information editting command
3. Make the 'join' response more sophisticated, particularly in the handling of child pointers.

5.0 Undesired Event Handling

Error Messages:

Message	Conditions of generation
Cannot write result file	Result file(s) is inaccessible for write.
Cannot write save file	File for save command is inaccessible for write.
File not found	Missing or inaccessible input file.
Ignoring *string* at end of command	Extra arguments on command entered by user.
Incomplete Input File	Pascal input file does nto contain at least one complete scope.
Invalid arguments to command	Command entered with arguments of wrong type.
Invalid global declaration	Pascal input file contains multiple scopes and global declarations.
Invalid workfile format	Workfile is of wrong format for restoration, or data in workfile is incomplete or inconsistent.
Missing command arguments	Command entered by user without needed arguments.
Missing procedure *name*	Procedure called that is not part of the input file.
No such command	Unrecognized command entered by user.
Null workfile	No //ACFG// nodes in workfile.

6.0 Glossary

!assign condition! left hand side of assignment statement = right hand side of assignment statement.

!case condition! case variable = case label

!GenGraph! See section 3.1.2

!if condition! The series of tokens appearing between **if** and **then**.

!loop condition! The condition that must be satisfied if the loop is to continue.

!loop invariant! The condition that expresses the semantics of the loop. True on every execution of the loop.

!MakeNode(T)! A newly allocated //ACFG// shall be linked into //ABKGRPH// at /CurNode//. In that //ACFG// !SetNum!, //ACFGLFT// shall be set to nil, //ACFGRT// shall be set to nil, //ACFGLCND// shall be set to reflect the condition in the current statement, //ACFGRCND// shall be set to refelect the logical inverse of that condition, !NoCalls!, //ACFGTEXT// shall be set to the text of the statement (excluding any substatements), /./ACFGSUMM// shall be set to an empty string, and //ACFGTYPE// shall be set to T and /CurNode/ shall be set to reference the left child of this //ACFG//.

!NewABK! //ABKNUMRET// shall be set to 0, //ABKRET// shall be set to an empty list, //ABKREACH// shall be set to **true**, //ABKGRPH// shall be set to nil, //ABKNSUBS// shall be set to 0, //ABKSUBS// shall be set to an empty list and //ABKDECL// shall be set to an empty string

!NewACFG! !SetNum!, //ACFGLFT// shall be set to nil, //ACFGRT// shall be set to nil, //ACFGLCND// shall be set to true, //ACFGRCND// shall be set to true, !NoCalls!, //ACFGTYPE// shall be set to $BeginEnd$ and //ACFGTEXT// shall be set to the empty string. An indicator /CurNode/ shall be set to reference the left child of this //ACFG//

!NoCalls! //ACFGNCALLS// shall be set to 0, //ACFGCALLS// shall be set to an empty list

!parameter-assignments! an expression in one of three forms: **true**, (*v* = *expression*), or ((*v* = *expression*) **and** !parameter-assignments!)

!ReadWork! See section 4.1.3

!SetABK! //ABKGRPH// shall be set to a newly allocated //ACFG// and in that //ACFG// !NewACFG!, //ABKNAME// shall be set to the procedure or function name and //ABKDECL// shall be set to contain the text of the procedure or function header

!SetNum! //ACFGNUM// shall be set to /NodeCnt/, /NodeCnt/ shall be incremented

Distribution List

Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Library, Code 0142 Naval Postgraduate School Monterey, CA 93943	2
Center for Naval Analyses 4401 Ford Ave. Alexandria, VA 22302-0268	1
Director of Research Administration Code 012 Naval Postgraduate School Monterey, CA 93943	1
Chairman, Computer Science Department Code 52 Naval Postgraduate School Monterey, CA 93943	1
Shimeall, Timothy J Code 52Sm Naval Postgraduate School Monterey, CA 93943	20

DUDLEY KNOX LIBRARY
3 2768 00338344 9