THE SAFETY OF LOGICAL STRUCTURES

by

KHALED ALI DAWOUD AL-ALI

B.S., The University of Wyoming, 1978

A THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

COMPUTER SCIENCE

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1989

Approved by:

[illegible]
Major Professor

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENT

Many thanks to Dr. David A. Gustafson for his guidance and contributions throughout this thesis. I am greatful to my wife Lynn and my children Dawoud, Sarah, and Donna for their patience and support during my stay at Kansas State University.

CHAPTER 1

INTRODUCTION

1.0 TESTING METHODOLOGIES

Commonly, two methods are used to confirm a software system's correctness with respect to its design specifications: program proving (verification) and program testing. Although program proving may provide the assurance necessary for determining the correctness of a program, such a procedure is often time consuming, complex, and unfeasible. Testing is usually the only practical and affordable alternative to give some confidence in the reliability and correctness of software.

In general, guaranteeing correctness is not a realistic goal of program testing [HOWD76]. Hence, the goals of testing are reduced to improving the quality and usefulness of the software at a reasonable cost [NTAF84] and to indicating that a program performs all its intended functions [RAPP85]. Testing accomplishes its task by locating and eliminating errors in programs. Test sets are generated according to a testing methodology, and the outcomes of the tests when applied to the program as inputs are compared with the expected outputs. This comparison is used to predict the accuracy of the program at any stage of software testing.

Usually, an oracle is assumed to exist during testing that would indicate the correctness of the produced output [RAPP85]. An oracle refers to a source of knowledge or information outside the software being tested [HOWD85]. One kind of oracle is an input-output oracle, which indicates for a given input if the actual output from testing is correct [HOWD86].

Several testing strategies have evolved and are extensively covered in the literature. Some are proven to be more practical than others in terms of the ease in selecting test data and in detecting program errors. The most recognized are structural (White Box) and functional (Black Box) testing methods. Structural testing uses the code of a program as the basis for testing. The main three strategies of the structural testing are node, branch, and path testing. In node testing, each node in the program is visited at least once while testing. A node is a sequence of statements that are executed together. This method is often referred to as C0 coverage. In branch testing, each branch in the program is visited at least once. A branch is a control flow transition between two nodes. This method is often referred to as C1 coverage. C1 coverage is considered by many as the minimum acceptable coverage criterion. In path testing, every path in the program must be visited. A path is an execution sequence of statements from an entry point to an exit point of the program. This method is often impractical even for simple programs due to the large number of possible paths [NTAF88]. Functional testing uses the specifications of the program to identify possible test cases for testing all recognizable output values of program functions [HOWD86].

Despite the variety of available testing methods, many limitations persist. Two obstacles are the diversity of the practical and theoretical approaches and their inability to draw from each other's strengths and eliminate common weaknesses. Another problem is knowing the amount of testing necessary to declare that a program is correct and ready for use. This is not a criticism of these methods, without which, testing would have no clear guidelines but is merely to state that there are many inherent difficulties within testing that must be overcome. These difficulties may include incorrect control structures, ambiguous and incomplete specifications, and the overwhelming number of test cases required to meet the coverage of some methods for even the simplest of programs. Despite their difficulties, structural testing strategies offer practical and effective means for program testing due to the simplicity and clarity of their approach [NTAF88]. C0 and C1 coverages are being accepted in parts of the software industry as a measure of the quality of the software, with many believing that the program structure often dictates its testability (the ease or difficulty of its testing). Thus, any approach to improving the testing process must take into consideration the program structure. Consequently, functional testing of a program without the examination of its code is considered incomplete.

The limitations of testing have lead to the exploration of other concepts to improve the testing process. One important concept is called software reliability.

1.1 SOFTWARE RELIABILITY CONCEPT

The classical definition of reliability is the probability of not failing while testing a program for a fixed time. The reliability is determined by estimating the number of errors left in the program at any stage of testing. Software reliability is an important but difficult concept to measure in the frame of testing theory. It is used as an indicator of the possibility of the correct execution of a program over a given set of test data. Intuitively, after several successful tests of a program, the confidence in its correctness and usefulness is considerably higher [GOUR83]. This has lead many to link the number of actual or predicted errors in the software to its reliability.

Several theoretical views exist for determining software reliability. The best known model was introduced by Jelinski and Moranda, which estimates the number of errors in the software and the mean time to failure by using assumptions on the distribution of faults and the software execution profile [DOWN86]. The criticism of this view of reliability stems from the fact that the estimations are based on errors no longer present in the program, and not in the corrected program [HAML81]. Another approach views reliability from an operational perspective, where program reliability is determined by the probability of correctly responding to the users' requests at a particular environment. Software reliability is expressed as a function of component reliability and the utilization profile of these components [CHEU80]. The problem with this concept is that program reliability is solely dependent on the user's environment. Probability predictions made in one environment offer no assistance in the prediction of software reliability in other environments. When a software system is transferred to a different environment, a large increase in its failure rate often occurs due to previously unexercised functions at a particular phase or environment [DOWN85]. These different views of software reliability indicate that it is often a judgmental concept. Many software systems contain errors which unless encountered will not prevent the software from providing a reasonable service.

Reliability estimations are often built on the assumption that a program is fully testable. Such an assumption usually will not hold unless the structure of the program is examined, since the structure will often determine the program testability.

Even accepted methods, such as structural testing or J.M. reliability model, cannot provide answers to some legitimate questions facing testing. The first question is " what effect, if any, does the program structure have on its testability ? ". The second question is " will a program have a higher probability of correctly evaluating test data based on the logical structure it uses?". The final question is " can the occurrence of major problems, such as coincidental correctness or missing path errors, be predicted using the program code structure?". Coincidental correctness occurs if a test data causes the wrong path to be traversed, and yet attains the same result as when a test data traverses the correct path. A missing path error occurs when a logical condition "predicate" is not present in the program to be tested [WHIT80]. Howden stated that existing path testing methods are not able to determine such an occurrence [HOWD76].

We believe that the answers to these questions would require the combining of probability analysis common to all the reliability models and structural evaluation of code used in most code-based testing strategies. The resultant methodology will provide a measure of the testability of programs and a view of the probable program behavior to aid in the assignment of required coverage for structural based testing methods.

1.3 RESEARCH OBJECTIVES.

With these questions in mind, a method is introduced to examine the effect that the logical structure of the program may have on its testability in terms of a new concept called safety. Program safety is defined as the probability of correctly evaluating all examined decisions or detecting errors in the examined code. The method uses the probable outcome of logical conditions evaluated in the program structure to determime the safety of its components. Using this method, the safety of equivalent programs with different logical structures can be examined to determine the ones with higher probability of correct test data evaluation. This measure of safety will provide a numerical comparison between different logical structures used in programs and will give testers a guideline to determine which expressions may have a higher degree of unsafety and thus require a closer look during testing. Questions, such as whether the restructuring of conditional statements within a program would reduce or increase its safety, or whether the use of different, but equivalent, logical configurations would influence software safety, or whether the situations of coincidental correctness and missing path errors can be predicted based on the evaluation of logical structure, could be answered.

CHAPTER 2

STRUCTURAL SAFETY

2.0 INTRODUCTION

In our analysis, the program structure is represented by a directed graph with a single entry and a single exit point. A simple node contains one or more statements evaluated in sequence. Possible branches between nodes are the arcs of the graph. We will represent a program as a unique logical configuration of nodes and arcs. Our proposed testing method incorporates the logical structure with the probable outcome of evaluating decision statements to predict the program behavior under erroneous and under correct conditions. The result of the evaluation provides an indication of program testability and suggests to paths where errors may or may not be detected. Testability is defined as the probability of uncovering errors in the program over a given set of test data and, therefore, provides an estimate of the successful testing of programs.

Howden [HOWD76] identified two types of common errors in programs. The first is a domain error in which an error occurs in the control structure of the program, effecting the evaluation of its decision statements. The second is a computational error in which an error occurs in an assignment statement of the program, causing the wrong result for some functions. We will restrict our study to the first type, as computational errors are often detected as a result of encountering domain errors [WHIT80]. Since the absence or existence of program errors cannot be predicted with absolute certainty, a method capable of determining the probabilities of correct or incorrect evaluations of the test data by a program provides a needed estimate of its probable behavior. Inferences drawn from these measures could be used to predict the quality and the reliability of the program.

We will assume the existence of an input-output oracle. Estimation of program behavior would require an oracle to determine the correctness of an output for any given input test data. This assumption is widely used in many testing methodologies [WHIT80].

2.1 DEFINITIONS OF PROGRAM COMPONENTS

In order to clearly define the concept of safety for programs, a few relevant terms must be defined first.

Definition 2.1

A simple expression is a boolean condition that contains one of the relational operators, { >, <, ==, >=, <=, != }, or a boolean variable with a pre-assigned logical value, which evaluates to true or false when the decision statement is executed.

Definition 2.2

A compound expression is a boolean condition that contains two or more simple expressions separated by the logical operators {AND, OR}.

Definition 2.3

An expression, either simple or compound, is a boolean condition which evaluates to true or false when the decision is evaluated. An expression determines the outcome of the branching of a decision statement.

Many researchers have concluded that most constructs used in programming are simple ones, and about ninety-eight percent of all the expressions used in programs contain two or less operators [WHIT80]. We will use this conclusion to limit the size of expressions to ones with two or less logical operators. The results can be easily expanded to fit expressions with more than two operators.

Definition 2.4

A simple decision causes a two way branching in a program, such as IF-THEN-ELSE statement. The branching path is determined by the expression.

Definition 2.5

A nested decision is a decision with at least one branch containing a simple decision.

Defination 2.6

A decision node, either simple or nested, is a block of code that has a decision at the entrance and a single exit point.

Definition 2.7

A program is usually a node with a single entry and a single exit point. The body of the program usually contains several decisions nodes performing varied tasks.

Next we will develop the definitions for the safety of the program and its componants.

2.2 DEFINITIONS OF SAFETY

Intuitively, program safety is a measure of the probability of the correct evaluation of all the decisions within the program over a given set of test data or the detection of any errors in all examined conditions. The following are definitions used to establish a concrete definition for safety.

Definition 2.8

Correctly evaluating a simple expression means getting the correct result, either TRUE or FALSE as stated by the oracle. The probability of correctly evaluating a simple expression will be refered to by Ptc() and Pfc().

Defination 2.9

Correctly evaluating a compound expression means evaluating the expression in the correct sequence of simple expressions and correctly evaluating each evaluated simple expression.

Definition 2.10

The correctness of an expression is the probability of correctly evaluating the expression. This will be expressed in terms of the probabilities of correctly evaluating the simple expressions. We will refer to this as Pc(expr).

Definition 2.11

Correctly evaluating a simple decision means correctly evaluating the expression within the decision.

Definition 2.12

Correctly evaluating a nested decision means correctly evaluating the nested decision in the correct sequence (path traversal) between simple decisions and correctly evaluating each examined simple decision.

Definition 2.13

The correctness of a decision node is the probability of correctly evaluating the decision. This will be expressed in terms of the probabilities of correctly evaluating the simple decisions.

Definition 2.14

Correctly evaluating a program means correctly evaluating the program in the correct sequence of nodes and correctly evaluating each examined decision.

Definition 2.15

The correctness of a program is the probability of correctly evaluating the program.

Definition 2.16

Detecting an error in a simple expression means getting an answer different from the oracle. The probability of detecting an error in a simple expression will be referred to by Ptwd() and Pfwd().

Definition 2.17

Detecting an error in a compound expression means incorrectly evaluating examined simple expressions, resulting in an incorrect sequence of evaluation, wrong results, and error detection.

Definition 2.18

The detectability of an expression is the probability of detecting an error in the expression. We will refer to this as Pwd(expr).

Definition 2.19

Detecting an error in a simple decision means detecting an error in the expression within the decision.

Definition 2.20

Detecting an error in a nested decision means incorrectly evaluating examined simple decisions, resulting in an incorrect sequence of evaluations, wrong results, and error detection.

Definition 2.21

The detectability of a decision node is the probability of detecting an error in the decision node.

Definition 2.22

Detecting an error in a program means incorrectly evaluating decisions, resulting in an incorrect sequence of evaluation, the wrong result, and errors being detected.

Definition 2.23

The detectability of a program is the probability of detecting errors in the program.

Definition 2.24

Not detecting an error in a simple expression means getting the correct answer (as stated by the oracle), but with an incorrect evaluation of the boolean condition due to the error existence. We will refer to this as Ptwu(),Pfwu().

Definition 2.25

Not detecting an error in a compound expression means getting the correct answer, but with an incorrect evaluation of examined simple expressions within and/or an incorrect sequence of evaluations between examined expressions.

Definition 2.26

The undetectability of an expression is the probability of not detecting an error in the expression. We will refer to this as Pwu(expr).

Definition 2.27

Not detecting an error in a simple decision means not detecting an error in the expression within the decision.

Definition 2.28

Not detecting an error in a nested decision means getting the correct answer, but with the incorrect sequence of evaluation and/or an incorrect evaluation of examined simple decisions within.

Definition 2.29

Undetectability of a decision is the probability of not detecting the error in the decision.

Definition 2.30

Not detecting an error in the program means getting the correct result, but with incorrect evaluation of examined decisions within and/or incorrect sequence of evaluation between examined decisions.

Definition 2.31

The undetectability of a program is the probability of not detecting errors in the program.

Definition 2.32

Incorrectly evaluating an expression means the incorrect evaluation of examined simple expressions within and/or the incorrect sequence of evaluations between examined expressions.

Definition 2.33

The incorrectness of an expression is the probability of incorrectly evaluating the expression. It is the sum of the probabilities of detectability Pwd(expr) and undetectability Pwu(expr) of an expression. It is referred to as Pw(expr).

Definition 2.34

Incorrectly evaluating a decision means the incorrect evaluation of examined simple decisions within and/or the incorrect sequence of evaluation between examined decisions.

Definition 2.35

The incorrectness of a decision node is the probability of incorrectly evaluating the decision.

Definition 2.36

Incorrectly evaluating a program means the incorrect evaluation of all examined decisions within and/or the incorrect sequence of evaluation between evaluated decisions.

Definition 2.37

The incorrectness of a program is the probability of incorrectly evaluating the program.

Definition 2.38

The safety of an expression is the probability that all evaluated simple expressions within the expression are correctly evaluated or that all errors in the evaluated expressions are detected. It is the sum of Pc(expr) and Pwd(expr), and will be referred to as Ps(expr).

Definition 2.39

The safety of a decision is the probability that all evaluated simple decisions within the decision are correctly evaluated or that errors in the evaluated decisions are detected.

Definition 2.40

The safety of a program is the probability that all evaluated decisions within the program are correctly evaluated or that errors in the evaluated decisions are detected.

Definition 2.41

The unsafety of an expression is the undetectability of an expression. It will be referred to as Pus(expr).

Definition 2.42

The unsafety of a decision node is the undetectability of a decision node.

Definition 2.43

The unsafety of a program is the undetectability of a program.

Unevaluated conditions during an expression's evaluation have no effect on the outcome, and are irrelevant to the determination of the correctness or the wrongfulness of an expression. Such conditions are not included in our safety analysis.

In the next chapter examples are presented to show different safety estimations for logical structures.

CHAPTER 3

SAFETY ESTIMATIONS OF CODE SEGMENTS

3.0 TERMINOLOGIES

Consider the following code segment.

```
IF (expr) THEN
    S1
ENDIF;
S2;
```

When the logical expression "expr" is evaluated during the execution of the segment, four different outcomes are possible. First, correct evaluation of "expr" producing logically true; let Ptc(expr) be the probability of such an occurrence. Second, correct evaluation of "expr" producing logically false; let Pfc(expr) be the probability of such an occurrence. Third, the wrong evaluation of "expr" producing logically true; let Ptw(expr) be the probability of such an occurrence. Fourth, the wrong evaluation of "expr" producing logically false; let Pfw(expr) be the probability of such an occurrence. Using the above defined probabilities and the logical structure of the segment, we can estimate the probabilities of correctness, incorrectness, safety, and unsafety for the code segment. We will refer to the probabilities of correctness, incorrectness, safety, and unsafety as safety estimation probabilities or SE probabilities for short.

3.1 ANALYSIS OF COMPOUND EXPRESSIONS

To illustrate the different effects that compound expressions may have on the SE probabilities of a code segment, some examples are introduced using expressions with different combinations of logical operators. Also, an equivalent structure to each expression, using the decision statement IF-THEN-ELSE, is analyzed for comparison.

3.1.1 EXPRESSIONS WITH A SINGLE LOGICAL OPERATOR

Example 1.

Given the code segment in figure 3.1, we can calculate the SE probabilities of the code segment.

```
IF (a AND b) THEN
 S1
ENDIF;
S2;
```

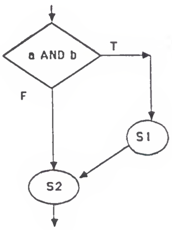

FIGURE 3.1 A code segment with the AND operator.

The four SE probabilities can be expressed as follows:

(a) The correctness of the segment

Pc(segment) = Ptc(expr) + Pfc(expr)

where,

Ptc(expr) = Ptc(a AND b)

Pfc(expr) = Pfc(a AND b)

(b) The incorrectness of the segment

Pw(segment) = Ptw(expr) + Pfw(expr)

where,

Ptw(expr) = Ptwd(a AND b) + Ptwu(a AND b)

Pfw(expr) = Pfwd(a AND b) + Pfwu(a AND b)

Here, wd and wu refer to wrong detected and wrong undetected.

(c) The safety of the segment

Ps(segment) = Pts(expr) + Pfs(expr)

where,

Pts(expr) = Ptc(a AND b) + Ptwd(a AND b)

Pfs(expr) = Pfc(a AND b) + Pfwd(a AND b)

(d) The unsafety of the segment

Pus(segment) = Ptwu(expr) + Pfwu(expr)

where,

Ptwu(expr) = Ptwu(a AND b)

Pfwu(expr) = Pfwu(a AND b)

We have devised a table showing all combinations possible for the probabilities of the boolean variables "a" and "b" in terms of TRUE, FALSE, correct, and wrong. The resultant probabilities of the evaluation of the expressions "a AND b" at each combination are also given. The SE probabilities are determined from the table in figure 3.2 by adding all the appropriate probabilities.

a	b	correct-AND	P(a)	P(b)	actual	P(aANDb)
			tc	tc	T	c
			tc	fw	F	wd
T	T	T	fw	tc	F	wd
			fw	fw	F	wd
			tc	tw	T	wd
			tc	fc	F	c
T	F	F	fw	tw	F	wu
			fw	fc	F	wu
			tw	tc	T	wd
			tw	fw	F	wu
F	T	F	fc	tc	F	c
			fc	fw	F	wu
			tw	tw	T	wd
			tw	fc	F	wu
F	F	F	fc	tw	F	wu
			fc	fc	F	c

FIGURE 3.2 A table for the Resultant Probabilities of Evaluating the AND operator.

(a) The correctness of "a AND b" is determined by adding all the combinations for which the probability of evaluation is correct. These are the combinations from the table in figure 3.2 where P(aANDb) is correct.

$$Pc(aANDb) = Ptc(a)*Ptc(b) + Ptc(a)*Pfc(b) + Pfc(a)*Ptc(b) + Pfc(a)*Pfc(b) \quad ....(1)$$

Next, the terms in the right-hand side of equation 1 are combined using simple algebraic rules.

$$Ptc(a)*Ptc(b) + Ptc(a)*Pfc(b) = Ptc(a)*Pc(b)$$

$$Pfc(a)*Ptc(b) + Pfc(a)*Pfc(b) = Pfc(a)*Pc(b)$$

Similarly,

$$Pfc(a)*Pc(b) + Ptc(a)*Pc(b) = Pc(a)*Pc(b)$$

Finally, this value is substituted in the right-hand side of equation 1 to get the result.

$$Pc(aANDb) = Pc(a)*Pc(b)$$

(b) $Pw(aANDb) = Pw(a) + Pc(a)*Pw(b)$

$= 1 - Pc(a)*Pc(b)$

$= 1 - Pc(aANDb)$

(c) $Ps(aANDb) = Pc(aANDb) + Pwd(aANDb)$

$Pwd(aANDb) = Ptc(a)*Pw(b) + Pw(a)*Ptc(b) + Pfw(a)*Pfw(b) + Ptw(a)*Ptw(b)$

$Pc(aANDb) = Ptc(a)*Pc(b) + Pfc(a)*Pc(b)$

$Ps(aANDb) = Ptc(a) + Pfc(a)*Pc(b) + Pw(a)*Ptc(b) + Pfw(a)*Pfw(b) + Ptw(a)*Ptw(b)$

(d) $Pus(aANDb) = Pfc(a)*Pw(b) + Pw(a)*Pfc(b) + Pfw(a)*Ptw(b) + Ptw(a)*Pfw(b)$

$= 1 - Ps(aANDb)$

Example 2.

Consider the program segment in figure 3.3 which is a functionally equivalent but structurally different program of example 1; we can calculate the SE probabilities of incorrectness and unsafety for a comparison between the two segments.

```
IF a THEN
   IF b THEN
      S1
   ENDIF;
ENDIF;
S2;
```

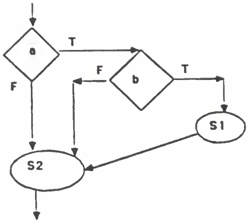

FIGURE 3.3 An equivalent logical representation of the expression (a AND b).

a	b	correct	P(a)	P(b)	actual	P(segment)
			tc	tc	T	c
			tc	fw	F	wd
T	T	T	fw	tc	F	wd
			fw	fw	F	wd
			tc	tw	T	wd
			tc	fc	F	c
T	F	F	fw	tw	F	wu
			fw	fc	F	wu
			tw	tc	T	wd
			tw	fw	F	wu
F	T	F	fc	tc	F	c
			fc	fw	F	c
			tw	tw	T	wd
			tw	fc	F	wu
F	F	F	fc	tw	F	c
			fc	fc	F	c

FIGURE 3.4 A table for the segment in figure 3.3

The probability of incorrectness in this case can be determined directly from the control flow diagram shown in figure 3.3 :

$$Pw(segment) = Pw(a) + Ptc(a)*Pw(b)$$

The SE probabilities for the segment using the table in figure 3.4:

(a) $Pc(segment) = Ptc(segment) + Pfc(segemnt)$
$\qquad = Pfc(a) + Ptc(a)*Pc(b)$

(b) $Pw(segment) = Ptw(segement) + Pfw(segment)$
$\qquad = Pw(a) + Ptc(a)*Pw(b)$

(c) $Ps(segment) = Pc(segment) + Pwd(segment)$
$\qquad = Pc(a) + Pw(a)*Ptc(b) +$
$\qquad Ptw(a)*Ptw(b) + Pfw(a)*Pfw(b)$

(d) $Pus(segment) = Ptwu(segment) + Pfwu(segment)$
$\qquad = Pw(a)*Pfc(b) + Ptw(a)*Pfw(b) +$
$\qquad Pfw(a)*Ptw(b)$

In examining the results of example 1 and 2, we found the probabilities of safety and correctness to be higher for example 2. The difference between the two results is in the treatment of unevaluated conditions. In example 2, only the errors occurring on the path of evaluation are considered in the unsafety calculations.

In example 1, an error occurring anywhere in the expression is considered in the unsafety calculations. The reason for the latter is to ensure that error occurrences in all the expressions examined are accounted for.

Next, we will examine a code segment with the logical operator OR and the resultant SE probabilities.

Example 3.

Consider the code segment in figure 3.5; we can calculate the SE probabilities for the segment.

```
IF (a OR b) THEN
  S1
ENDIF;
S2;
```

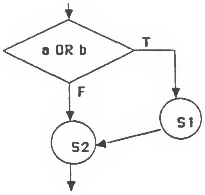

FIGURE 3.5 A code Segment with the OR operator.

We have devised a table, shown in figure 3.6 for determining the SE probabilities.

a	b	correct- OR	P(a)	P(b)	actual	P(aORb)
			tc	tc	T	c
			tc	fw	T	wu
T	T	T	fw	tc	T	wu
			fw	fw	F	wd
			tc	tw	T	wu
			tc	fc	T	c
T	F	T	fw	tw	T	wu
			fw	fc	F	wd
			tw	tc	T	wu
			tw	fw	T	wu
F	T	T	fc	tc	T	c
			fc	fw	F	wd
			tw	tw	T	wd
			tw	fc	T	wd
F	F	F	fc	tw	T	wd
			fc	fc	F	c

FIGURE 3.6 A table for evaluating the OR operator.

The SE probabilities for the segment, using the table in figure 3.6 are:

(a) Pc(aORb) = Pc(a)*Pc(b)

(b) Pw(aORb) = 1 - Pc(a)*Pc(b)

(c) Ps(aORb) = Pfc(a) + Ptc(a)*Pc(b) + Pw(a)*Pfc(b) +
Ptw(a)*Ptw(b) + Pfw(a)*Pfw(b)

(d) Pus(aORb)= 1 - Ps(aORb)

Example 4.

Another equivalent structure for the one in example 3 is shown in figure 3.7; we can calculate the SE probabilities for this segment for a comparison.

```
IF a THEN
  S1
ELSE
  IF b THEN
    S1
  ENDIF;
ENDIF;
S2;
```

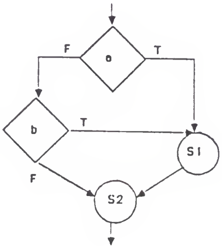

FIGURE 3.7 An equivalent logical representation of the expression (a OR b).

The probability of incorrectness for the code segment can be determined directly from the control flow diagram shown in figure 3.7;

Pw(code segment) = Pw(a) + Pfc(a)*Pw(b)

We have devised a table, shown in figure 3.8, for determining the SE probabilities for the segment.

a	b	correct	P(a)	P(b)	actual	P(segment)
			tc	tc	T	c
			tc	fw	T	c
T	T	T	fw	tc	T	wu
			fw	fw	F	wd
			tc	tw	T	c
			tc	fc	T	c
T	F	T	fw	tw	T	wu
			fw	fc	F	wd
			tw	tc	T	wu
			tw	fw	T	wu
F	T	T	fc	tc	T	c
			fc	fw	F	wd
			tw	tw	T	wd
			tw	fc	T	wd
F	F	F	fc	tw	T	wd
			fc	fc	F	c

FIGURE 3.8 A table for the segment in figure 3.7

The SE probabilities of code segment are:

(a) Pw(code segment) = Pw(a) + Pfc(a)*Pw(b)

(b) Pc(code segment) = Ptc(a) + Pfc(a)*Pc(b)

(c) Ps(code segment) = Pc(a) + Pw(a)*Pfc(b) +
Ptw(a)*Ptw(b) + Pfw(a)*Pfw(b)

(d) Pus(code segment) = Ptwu(segment) + Pfwu(segment)

Ptwu(segment) = Pw(a)*Ptc(b) + Ptw(a)*Pfw(b) +
Pfw(a)*Ptw(b)

Pfwu(segment) = 0

Pus(code segment) = Pw(a)*Ptc(b) + Ptw(a)*Pfw(b) +
Pfw(a)*Ptw(b)

By examining the results of example 3 and 4, we found the probabilities of safety and correctness to be higher for example 4. The difference between the two results lies in the treatment of all unevaluated conditions. In example 4, only the errors occurring on the path of evaluation are considered a part of the unsafety calculations. In example 3, an error occurring in any part of the expression is added to the unsafety calculations.

3.1.2 EXPRESSIONS WITH TWO LOGICAL OPERATORS

Example 5.

Consider the code segment in figure 3.9; we can determine the SE probabilities for this code segment.

```
IF (aANDbANDc) THEN
  S1
ENDIF;
S2;
```

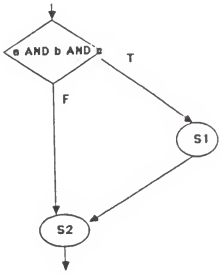

FIGURE 3.9 A code segment using the compound logical expression (a AND b AND c).

We have devised a table, shown in figure 3.10, that accomplishes the task.

a	b	c	correct-AND	P(a)	P(b)	P(c)	actual	P(aANDbANDc)
				tc	tc	tc	T	c
				tc	tc	fw	F	wd
				tc	fw	tc	F	wd
				tc	fw	fw	F	wd
T	T	T	T	fw	tc	tc	F	wd
				fw	tc	fw	F	wd
				fw	fw	tc	F	wd
				fw	fw	fw	F	wd
				tc	tc	tw	T	wd
				tc	tc	fc	F	c
				tc	fw	tw	F	wu
				tc	fw	fc	F	wu
T	T	F	F	fw	tc	tw	F	wu
				fw	tc	fc	F	wu
				fw	fw	tw	F	wu
				fw	fw	fc	F	wu
				tc	tw	tc	T	wd
				tc	tw	fw	F	wu
				tc	fc	tc	F	c
				tc	fc	fw	F	wu
T	F	T	F	fw	tw	tc	F	wu
				fw	tw	fw	F	wu
				fw	fc	tc	F	wu
				fw	fc	fw	F	wu
				tc	tw	tw	T	wd
				tc	tw	fc	F	wu
				tc	fc	tw	F	wu
				tc	fc	fc	F	c
T	F	F	F	fw	tw	tw	F	wu
				fw	tw	fc	F	wu
				fw	fc	tw	F	wu
				fw	fc	fc	F	wu
				tw	tc	tc	T	wd
				tw	tc	fw	F	wu
				tw	fw	tc	F	wu
				tw	fw	fw	F	wu
F	T	T	F	fc	tc	tc	F	c
				fc	tc	fw	F	wu
				fc	fw	tc	F	wu
				fc	fw	fw	F	wu
				tw	tc	tw	T	wd
				tw	tc	fc	F	wu
				tw	fw	tw	F	wu
				tw	fw	fc	F	wu
F	T	F	F	fc	tc	tw	F	wu
				fc	tc	fc	F	c
				fc	fw	tw	F	wu
				fc	fw	fc	F	wu
				tw	tw	tc	T	wd
				tw	tw	fw	F	wu
				tw	fc	tc	F	wu
				tw	fc	fw	F	wu
F	F	T	F	fc	tw	tc	F	wu
				fc	tw	fw	F	wu
				fc	fc	tc	F	c
				fc	fc	fw	F	wu
				tw	tw	tw	T	wd
				tw	tw	fc	F	wu
				tw	fc	tw	F	wu
				tw	fc	fc	F	wu
F	F	F	F	fc	tw	tw	F	wu
				fc	tw	fc	F	wu
				fc	fc	tw	F	wu
				fc	fc	fc	F	c

FIGURE 3.10 Resultant probabilities for figure 3.9

The SE probabilities for example 5 are:

(a) The correctness is

$$Pc(aANDbANDc) = Pc(a)*Pc(b)*Pc(c)$$

(b) The incorrectness is

$$Pw(aANDbANDc) = 1 - Pc(aANDbANDc)$$

(c) The safety is

$$\begin{aligned} Ps(aANDbANDc) = {} & Pc(a)*Pc(b)*Pc(c) + \\ & Ptc(a)*Ptw(b)*Ptw(c) + Ptc(a)*Ptc(b)*Pw(c) + \\ & Ptc(a)*Pw(b)*Ptc(c) + Ptc(a)*Pfw(b)*Pfw(c) + \\ & Pw(a)*Ptc(b)*Ptc(c) + Ptw(a)*Ptc(b)*Ptw(c) + \\ & Ptw(a)*Ptw(b)*Ptc(c) + Ptw(a)*Ptw(b)*Ptw(c) + \\ & Pfw(a)*Ptc(b)*Pfw(c) + Pfw(a)*Pfw(b)*Ptc(c) + \\ & Pfw(a)*Pfw(b)*Pfw(c) \end{aligned}$$

(d) The unsafety is

$$Pus(aANDbANDc) = 1 -Ps(aANDbANDc)$$

Next, an example is analyzed using an equivalent structure to the one in example 5 for comparison.

Example 6.

Given the code segment in figure 3.11, we can determine the SE probabilities for the segment.

```
IF a THEN
  IF b THEN
    IF c THEN
      S1
    ENDIF;
  ENDIF;
ENDIF;
S2;
```

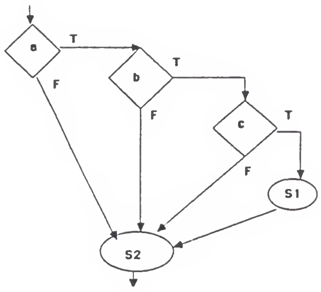

FIGURE 3.11 A code segment with a structure equivalent to the expression (a AND b AND c).

The probability of incorrectness, Pw(segment), can be determined from figure 3.11 directly;

$$Pw(segment) = Pw(a) + Ptc(a)*Pw(b) + Ptc(a)*Ptc(b)*Pw(c)$$

We have devised a table shown in figure 3.12 to determine the SE probabilities of example 3.6.

a	b	c	correct	P(a)	P(b)	P(c)	actual	P(segment)
				tc	tc	tc	T	c
				tc	tc	fw	F	wd
				tc	fw	tc	P	wd
				tc	fw	fw	F	wd
T	T	T	T	fw	tc	tc	F	wd
				fw	tc	fw	F	wd
				fw	fw	tc	F	wd
				fw	fw	fw	F	wd
				tc	tc	tw	T	wd
				tc	tc	fc	F	c
				tc	fw	tw	F	wu
				tc	fw	fc	F	wu
T	T	F	F	fw	tc	tw	F	wu
				fw	tc	fc	F	wu
				fw	fw	tw	F	wu
				fw	fw	fc	F	wu
				tc	tw	tc	T	wd
				tc	tw	fw	F	wu
				tc	fc	tc	F	c
				tc	fc	fw	F	c
T	F	T	F	fw	tw	tc	F	wu
				fw	tw	fw	F	wu
				fw	fc	tc	F	wu
				fw	fc	fw	F	wu
				tc	tw	tw	T	wd
				tc	tw	fc	P	wu
				tc	fc	tw	F	c
				tc	fc	fc	F	c
T	F	F	F	fw	tw	tw	F	wu
				fw	tw	fc	P	wu
				fw	fc	tw	F	wu
				fw	fc	fc	F	wu
				tw	tc	tc	T	wd
				tw	tc	fw	F	wu
				tw	fw	tc	F	wu
				tw	fw	fw	F	wu
F	T	T	F	fc	tc	tc	F	c
				fc	tc	fw	F	c
				fc	fw	tc	F	c
				fc	fw	fw	F	c
				tw	tc	tw	T	wd
				tw	tc	fc	P	wu
				tw	fw	tw	F	wu
				tw	fw	fc	F	wu
F	T	F	F	fc	tc	tw	F	c
				fc	tc	fc	F	c
				fc	fw	tw	F	c
				fc	fw	fc	F	c
				tw	tw	tc	T	wd
				tw	tw	fw	F	wu
				tw	fc	tc	F	wu
				tw	fc	fw	F	wu
P	F	T	P	fc	tw	tc	F	c
				fc	tw	fw	F	c
				fc	fc	tc	F	c
				fc	fc	fw	F	c
				tw	tw	tw	T	wd
				tw	tw	fc	F	wu
				tw	fc	tw	F	wu
				tw	fc	fc	F	wu
F	F	F	F	fc	tw	tw	F	c
				fc	tw	fc	P	c
				fc	fc	tw	F	c
				fc	fc	fc	F	c

FIGURE 3.12 A table for the segment in example 6.

Using the table in figure 3.12;

(a) The correctness is

Pc(segment) = Pfc(a) + Ptc(a)*Pfc(b) +
Ptc(a)*Ptc(b)*Pc(c)

(b) The incorrectness is

Pw(segment) = Pw(a) + Ptc(a)*Pw(b) +
Ptc(a)*Ptc(b)*Pw(c)

(c) The safety is

Ps(segment) = Pfc(a) + Ptc(a)*Pfc(b) +
Ptc(a)*Ptc(b)*Pc(c) + Ptc(a)*Ptw(b)*Ptw(c) +
Ptc(a)*Ptc(b)*Pw(c) + Ptc(a)*Pw(b)*Ptc(c) +
Ptc(a)*Pfw(b)*Pfw(c) + Pw(a)*Ptc(b)*Ptc(c) +
Ptw(a)*Ptc(b)*Ptw(c) + Ptw(a)*Ptw(b)*Ptc(c) +
Ptw(a)*Ptw(b)*Ptw(c) + Pfw(a)*Ptc(b)*Pfw(c) +
Pfw(a)*Pfw(b)*Ptc(c) + Pfw(a)*Pfw(b)*Pfw(c)

(d) The unsafety is

Pus(segment) = Pw(a)*Pfc(b) + Ptw(a)*Pfw(b) +
Pfw(a)*Ptw(b) + Pw(a)*Ptc(b)*Pfc(c) +
Ptc(a)*Pw(b)*Pfc(c) + Ptc(a)*Ptw(b)*Pfw(c) +
Ptc(a)*Pfw(b)*Ptw(c) + Ptw(a)*Ptw(b)*Pfc(c) +
Ptw(a)*Ptw(b)*Pfw(c) + Ptw(a)*Ptc(b)*Pfw(c) +
Pfw(a)*Ptc(b)*Ptw(c) + Pfw(a)*Pfw(b)*Ptw(c) +
Pfw(a)*Pfw(b)*Pfc(c)

In examining the results of example 5 and 6, the probability of safety and the probability correctness is higher in example 6. The difference between the two results is in the treatment of unevaluated conditions. In example 6, only the errors occurring on the path of evaluation are considered in the unsafety calculations. In example 5, an error occurring anywhere in the expression is a part of the unsafety calculations because of the complete evaluation usage.

Example 7.

Given the code segment in figure 3.13, we can determine the SE probabilities for the segment.

```
IF (aORbORc) THEN
   S1
ENDIF;
S2;
```

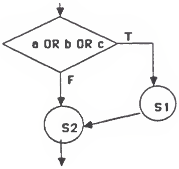

FIGURE 3.13 A code segment using the compound expression (a OR b OR c).

The table in figure 3.14 will accomplish the task.

a	b	c	correct- OR	P(a)	P(b)	P(c)	actual	P(aORbORc)
				tc	tc	tc	T	c
				tc	tc	fw	T	wu
				tc	fw	tc	T	wu
				tc	fw	fw	T	wu
T	T	T	T	fw	tc	tc	T	wu
				fw	tc	fw	T	wu
				fw	fw	tc	T	wu
				fw	fw	fw	F	wd
				tc	tc	tw	T	wu
				tc	tc	fc	T	c
				tc	fw	tw	T	wu
				tc	fw	fc	T	wu
T	T	F	T	fw	tc	tw	T	wu
				fw	tc	fc	T	wu
				fw	fw	tw	T	wu
				fw	fw	fc	F	wd
				tc	tw	tc	T	wu
				tc	tw	fw	T	wu
				tc	fc	tc	T	c
				tc	fc	fw	T	wu
T	F	T	T	fw	tw	tc	T	wu
				fw	tw	fw	T	wu
				fw	fc	tc	T	wu
				fw	fc	fw	F	wd
				tc	tw	tw	T	wu
				tc	tw	fc	T	wu
				tc	fc	tw	T	wu
				tc	fc	fc	T	c
T	F	F	T	fw	tw	tw	T	wu
				fw	tw	fc	T	wu
				fw	fc	tw	T	wu
				fw	fc	fc	F	wd
				tw	tc	tc	T	wu
				tw	tc	fw	T	wu
				tw	fw	tc	T	wu
				tw	fw	fw	T	wu
F	T	T	T	fc	tc	tc	T	c
				fc	tc	fw	T	wu
				fc	fw	tc	T	wu
				fc	fw	fw	F	wd
				tw	tc	tw	T	wu
				tw	tc	fc	T	wu
				tw	fw	tw	T	wu
				tw	fw	fc	T	wu
F	T	F	T	fc	tc	tw	T	wu
				fc	tc	fc	T	c
				fc	fw	tw	T	wu
				fc	fw	fc	F	wd
				tw	tw	tc	T	wu
				tw	tw	fw	T	wu
				tw	fc	tc	T	wu
				tw	fc	fw	T	wu
F	F	T	T	fc	tw	tc	T	wu
				fc	tw	fw	T	wu
				fc	fc	tc	T	c
				fc	fc	fw	F	wd
				tw	tw	tw	T	wd
				tw	tw	fc	T	wd
				tw	fc	tw	T	wd
				tw	fc	fc	T	wd
F	F	F	F	fc	tw	tw	T	wd
				fc	tw	fc	T	wd
				fc	fc	tw	T	wd
				fc	fc	fc	F	c

FIGURE 3.14 A table for the expression in example 7.

(a) The correctness is

$Pc(aORbORc) = Pc(a)*Pc(b)*Pc(c)$

(b) The incorrectness is

$Pw(aORbORc) = 1 - Pc(a)*Pc(b)*Pc(c)$

(c) The safety is

$$\begin{aligned}
Ps(aORbORc) = {} & Pc(a)*P(b)*P(c) + Pfc(a)*Ptw(b)*Ptw(c) + \\
& Pfc(a)*Pfc(b)*Pw(c) + Pfc(a)*Pw(b)*Pfc(c) + \\
& Pfc(a)*Pfw(b)*Pfw(c) + Pw(a)*Pfc(b)*Pfc(c) + \\
& Ptw(a)*Pfc(b)*Ptw(c) + Ptw(a)*Ptw(b)*Pfc(c) + \\
& Ptw(a)*Ptw(b)*Ptw(c) + Pfw(a)*Pfc(b)*Pfw(c) + \\
& Pfw(a)*Pfw(b)*Pfc(c) + Pfw(a)*Pfw(b)*Pfw(c)
\end{aligned}$$

(d) The unsafety is

$Pus(aORbORc) = 1 - Ps(aORbORc)$

Next, a segment with an equivalent structure to the one in example 7 is analyzed for comparison.

Example 8.

Consider the code segment in figure 3.15, we can determine the SE probabilities for the segment.

```
IF s THEN
  S1
ELSE
  IF b THEN
    S1
  ELSE
    IF c THEN
      S1
    ENDIF;
  ENDIF;
ENDIF;
S2;
```

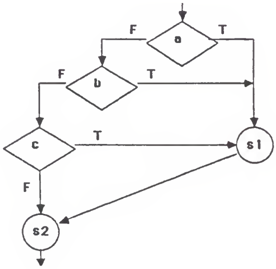

FIGURE 3.15 A structure that is equivalent to the expression (a OR b OR c).

The probability of incorrectness, Pw(segment), can be determined directly from the flow graph in figure 3.15.

$$Pw(segment) = Pw(a) + Pfc(a)*Pw(b) + Pfc(a)*Pfc(b)*Pw(c)$$

We have devised a table, in figure 3.16 on the next page, that accomplishes the task.

a	b	c	correct	P(a)	P(b)	P(c)	actual	P(segment)
				tc	tc	tc	T	c
				tc	tc	fw	T	c
				tc	fw	tc	T	c
				tc	fw	fw	T	c
T	T	T	T	fw	tc	tc	T	wu
				fw	tc	fw	T	wu
				fw	fw	tc	T	wu
				fw	fw	fw	F	wd
				tc	tc	tw	T	c
				tc	tc	fc	T	c
				tc	fw	tw	T	c
				tc	fw	fc	T	c
T	T	F	T	fw	tc	tw	T	wu
				fw	tc	fc	T	wu
				fw	fw	tw	T	wu
				fw	fw	fc	F	wd
				tc	tw	tc	T	c
				tc	tw	fw	T	c
				tc	fc	tc	T	c
				tc	fc	fw	T	c
T	F	T	T	fw	tw	tc	T	wu
				fw	tw	fw	T	wu
				fw	fc	tc	T	wu
				fw	fc	fw	F	wd
				tc	tw	tw	T	c
				tc	tw	fc	T	c
				tc	fc	tw	T	c
				tc	fc	fc	T	c
T	F	F	T	fw	tw	tw	T	wu
				fw	tw	fc	T	wu
				fw	fc	tw	T	wu
				fw	fc	fc	F	wd
				tw	tc	tc	T	wu
				tw	tc	fw	T	wu
				tw	fw	tc	T	wu
				tw	fw	fw	T	wu
F	T	T	T	fc	tc	tc	T	c
				fc	tc	fw	T	c
				fc	fw	tc	T	wu
				fc	fw	fw	F	wd
				tw	tc	tw	T	wu
				tw	tc	fc	T	wu
				tw	fw	tw	T	wu
				tw	fw	fc	T	wu
F	T	F	T	fc	tc	tw	T	c
				fc	tc	fc	T	c
				fc	fw	tw	T	wu
				fc	fw	fc	F	wd
				tw	tw	tc	T	wu
				tw	tw	fw	T	wu
				tw	fc	tc	T	wu
				tw	fc	fw	T	wu
F	F	T	T	fc	tw	tc	T	wu
				fc	tw	fw	T	wu
				fc	fc	tc	T	c
				fc	fc	fw	F	wd
				tw	tw	tw	T	wd
				tw	tw	fc	T	wd
				tw	fc	tw	T	wd
				tw	fc	fc	T	wd
F	F	F	F	fc	tw	tw	T	wd
				fc	tw	fc	T	wd
				fc	fc	tw	T	wd
				fc	fc	fc	F	c

FIGURE 3.16 A table for the segment in example 8.

Using the table in figure 3.16;

(a) The correctness is

Pc(segment) = Ptc(a) + Pfc(a)*Ptc(b) +
Pfc(a)*Pfc(b)*Pc(c)

(b) The incorrectness is

Pw(segment) = Pw(a) + Pfc(a)*Pw(b) +
Pfc(a)*Pfc(b)*Pw(c)

(c) The safety is

Ps(segment) = Ptc(a) + Pfc(a)*Ptc(b) +
Pfc(a)*Pfc(b)*Pc(c) + Pfc(a)*Ptw(b)*Ptw(c) +
Pfc(a)*Pfc(b)*Pw(c) + Pfc(a)*Pw(b)*Pfc(c) +
Pfc(a)*Pfw(b)*Pfw(c) + Pw(a)*Pfc(b)*Pfc(c) +
Ptw(a)*Pfc(b)*Ptw(c) + Ptw(a)*Ptw(b)*Pfc(c) +
Ptw(a)*Ptw(b)*Ptw(c) + Pfw(a)*Pfc(b)*Pfw(c) +
Pfw(a)*Pfw(b)*Pfc(c) + Pfw(a)*Pfw(b)*Pfw(c)

(d) The unsafety is

Pus(segment) = Pw(a)*Ptc(b) + Ptw(a)*Pfw(b) +
Pfw(a)*Ptw(b) + Pw(a)*Pfc(b)*Ptc(c) +
Pfc(a)*Ptw(b)*Ptc(c) + Pfc(a)*Ptw(b)*Pfw(c) +
Pfc(a)*Pfw(b)*Ptw(c) + Ptw(a)*Ptw(b)*Ptc(c) +
Ptw(a)*Ptw(b)*Pfw(c) + Ptw(a)*Pfc(b)*Pfw(c) +
Pfw(a)*Pfc(b)*Ptw(c) + Pfw(a)*Pfw(b)*Ptw(c) +
Pfw(a)*Pfw(b)*Ptc(c)

In examining the results of example 7 and 8, we found the probabilities of safety and correctness to be higher for example 8. The difference between the two results is in the treatment of unevaluated conditions.

Example 9.

Given the segment in figure 3.17, we can determine the SE probabilities for the segment.

```
IF (aORbANDc) THEN
  S1
ENDIF;
S2;
```

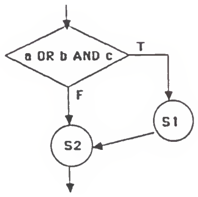

FIGURE 3.17 A code segment using the compound logical expression (a OR b AND c).

The condition "b AND c" will be evaluated first according to the precedence of logical operators, since the order of evaluation is important for determining the SE probabilities. A table is given in figure 3.18 to aid in finding the SE probabilities for the segment.

a	b	c	correct	P(a)	P(b)	P(c)	actual	P(aORbANDc)
				tc	tc	tc	T	c
				tc	tc	fw	T	wu
				tc	fw	tc	T	wu
				tc	fw	fw	T	wu
T	T	T	T	fw	tc	tc	T	wu
				fw	tc	fw	F	wd
				fw	fw	tc	F	wd
				fw	fw	fw	F	wd
				tc	tc	tw	T	wu
				tc	tc	fc	T	c
				tc	fw	tw	T	wu
				tc	fw	fc	T	wu
T	T	F	T	fw	tc	tw	T	wu
				fw	tc	fc	F	wd
				fw	fw	tw	F	wd
				fw	fw	fc	F	wd
				tc	tw	tc	T	wu
				tc	tw	fw	T	wu
				tc	fc	tc	T	c
				tc	fc	fw	T	wu
T	F	T	T	fw	tw	tc	T	wu
				fw	tw	fw	F	wd
				fw	fc	tc	F	wd
				fw	fc	fw	F	wd
				tc	tw	tw	T	wu
				tc	tw	fc	T	wu
				tc	fc	tw	T	wu
				tc	fc	fc	T	c
T	F	F	T	fw	tw	tw	T	wu
				fw	tw	fc	F	wd
				fw	fc	tw	F	wd
				fw	fc	fc	F	wd
				tw	tc	tc	T	wu
				tw	tc	fw	T	wu
				tw	fw	tc	T	wu
				tw	fw	fw	T	wu
F	T	T	T	fc	tc	tc	T	c
				fc	tc	fw	F	wd
				fc	fw	tc	F	wd
				fc	fw	fw	F	wd
				tw	tc	tw	T	wd
				tw	tc	fc	T	wd
				tw	fw	tw	T	wd
				tw	fw	fc	T	wd
F	T	F	F	fc	tc	tw	T	wd
				fc	tc	fc	F	c
				fc	fw	tw	F	wu
				fc	fw	fc	F	wu
				tw	tw	tc	T	wd
				tw	tw	fw	T	wd
				tw	fc	tc	T	wd
				tw	fc	fw	T	wd
F	F	T	F	fc	tw	tc	T	wd
				fc	tw	fw	F	wu
				fc	fc	tc	F	c
				fc	fc	fw	F	wu
				tw	tw	tw	T	wd
				tw	tw	fc	T	wd
				tw	fc	tw	T	wd
				tw	fc	fc	T	wd
F	F	F	F	fc	tw	tw	T	wd
				fc	tw	fc	F	wu
				fc	fc	tw	F	wu
				fc	fc	fc	F	c

FIGURE 3.18 A table for the expression (a OR b AND c).

Using the table in figure 3.18;

(a) The correctness is

Pc(aORbANDc) = Pc(a)*Pc(b)*Pc(c)

(b) The incorrectness is

Pw(aORbANDc) = 1 - Pc(a)*Pc(b)*Pc(c)

(c) The unsafety is

Pus(aORbANDc) = Pwu(aORbANDc)

Pwu(aORbANDc) = Pt(a)*Pfw(b)*(Pfw(c) + Ptc(c)) +

Pt(a)*Ptc(b)*Pfw(c) + Pw(a)*Ptc(b)*Ptc(c) +

(Pfw(a) + Ptc(a))*(Pt(b)*Ptw(c) + Ptw(b)*Ptc(c)) +

Pc(a)*(Pw(b)*Pfc(c) + Pfc(b)*Pw(c)) +

Pc(a)*(Ptw(b)*Pfw(c) + Pfw(b)*Ptw(c))

(d) The safety is

Ps(aORbANDc) = 1 - Pus(aORbANDc)

Next, an equivalent structure is analyzed for a comparison with this example.

Example 10.

Using the segment in figure 3.19, we can determine the SE probabilities.

```
IF b THEN
    IF c THEN
       S1
    ELSE
      IF a THEN
         S1
      ENDIF;
    ENDIF;
ELSE
  IF a THEN
    S1
  ENDIF;
ENDIF;
S2;
```

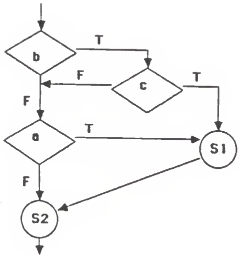

FIGURE 3.19 A structure that is equivalent to the expression (a OR b AND c).

We can determine the probability of incorrectness directly from the flow graph in figure 3.19;

$$Pw(segment) = Pw(b) + Pw(a)*Pfc(b) + Ptc(b)*Pw(c) + Pw(a)*Ptc(b)*Pfc(c)$$

The SE probabilities are obtained using the table in figure 3.20 on the next page.

a	b	c	correct	P(a)	P(b)	P(c)	actual	P(segment)
				tc	tc	tc	T	c
				tc	tc	fw	T	wu
				tc	fw	tc	T	wu
				tc	fw	fw	T	wu
T	T	T	T	fw	tc	tc	T	c
				fw	tc	fw	F	wd
				fw	fw	tc	F	wd
				fw	fw	fw	F	wd
				tc	tc	tw	T	wu
				tc	tc	fc	T	c
				tc	fw	tw	T	wu
				tc	fw	fc	T	wu
T	T	F	T	fw	tc	tw	T	wu
				fw	tc	fc	F	wd
				fw	fw	tw	F	wd
				fw	fw	fc	F	wd
				tc	tw	tc	T	wu
				tc	tw	fw	T	wu
				tc	fc	tc	T	c
				tc	fc	fw	T	c
T	F	T	T	fw	tw	tc	T	wu
				fw	tw	fw	F	wd
				fw	fc	tc	F	wd
				fw	fc	fw	F	wd
				tc	tw	tw	T	wu
				tc	tw	fc	T	wu
				tc	fc	tw	T	c
				tc	fc	fc	T	c
T	F	F	T	fw	tw	tw	T	wu
				fw	tw	fc	F	wd
				fw	fc	tw	F	wd
				fw	fc	fc	F	wd
				tw	tc	tc	T	c
				tw	tc	fw	T	wu
				tw	fw	tc	T	wu
				tw	fw	fw	T	wu
F	T	T	T	fc	tc	tc	T	c
				fc	tc	fw	F	wd
				fc	fw	tc	F	wd
				fc	fw	fw	F	wd
				tw	tc	tw	T	wd
				tw	tc	fc	T	wd
				tw	fw	tw	T	wd
				tw	fw	fc	T	wd
F	T	F	F	fc	tc	tw	T	wd
				fc	tc	fc	F	c
				fc	fw	tw	F	wu
				fc	fw	fc	F	wu
				tw	tw	tc	T	wd
				tw	tw	fw	T	wd
				tw	fc	tc	T	wd
				tw	fc	fw	T	wd
F	F	T	F	fc	tw	tc	T	wd
				fc	tw	fw	F	wu
				fc	fc	tc	F	c
				fc	fc	fw	F	c
				tw	tw	tw	T	wd
				tw	tw	fc	T	wd
				tw	fc	tw	T	wd
				tw	fc	fc	T	wd
F	F	F	F	fc	tw	tw	T	wd
				fc	tw	fc	F	wu
				fc	fc	tw	F	c
				fc	fc	fc	F	c

FIGURE 3.20 A table for the segment in example 10.

using the table in figure 3.20;

(a) The correctness is

Pc(segment) = Pc(a)*Pfc(b)*Pw(c) + Pc(a)*Pc(b)*Pc(c) +
Pw(a)*Ptc(b)*Ptc(c)

(b) The incorrectness is

Pw(segment) = 1 - Pc(segment)

(c) The safety is

Ps(segment) = Pc(segment) + Pwd(segment)
= 1 - Pus(segment)

(d) The unsafety is

Pus(segment) = Ptc(a)*Pfw(b)*(Pfw(c) + Ptc(c)) +
Pt(a)*Ptc(b)*Pfw(c) + Pc(a)*Pw(b)*Pfc(c) +
Pc(a)*(Ptw(b)*Pfw(c) + Pfw(b)*Ptw(c)) +
(Pfw(a) + Ptc(a))*(Pt(b)*Ptw(c) + Ptw(b)*Ptc(c))

In examining the results of example 9 and 10, we found the probabilities of safety and correctness to be higher for example 10. The difference between the two results is in the treatment of unevaluated conditions. In example 10, only errors occurring on the path of evaluation are a part of the unsafety calculations.

3.2 ANALYSIS SUMMARY

Usually, it is not possible to determine whether an error occurrence in unexamined conditions is detectable or not. If the existence of errors in unevaluated conditions is not a part of the safety calculations, then this approach of evaluation is referred to as the short circuit approach. We used this approach in analyzing examples 2, 4, 6, and 8. In evaluating the expressions themselves, all error occurrences are added to the unsafety calculations. This method of evaluation is called the complete evaluation approach. The complete approach is used to account for all errors in examined expressions. The resultant safety and correctness probabilities are higher in the examples using the short circuit approach. This is due to the extra number of correct probability combinations that are considered wrong and undetected by examples 1, 3, 5, and 7 which use the complete evaluation approach. The over estimation of unsafety, in the complete approach, is used as a guard against the problem of missing path errors, and also used to make the evaluation of expressions as machine independent as possible. The results of examples 1 through 10 will be used as the basis for evaluating the SE probabilities of programs, covered in chapter 4.

CHAPTER 4

SAFETY ESTIMATION OF PROGRAMS

4.0 ANALYSIS METHODOLOGY

To determine the SE probabilities for programs, was devised two algorithms. In algorithm 1, a table, similer to the ones in chapter 3, is developed for the program as a whole. The table containes every decision in the program. The SE probability formulas for the program are found from this table. In algorithm 2, The program flow graph is reduced to simplify the calculation of the SE probabilities. A table is then devised for the reduced graph. This alogrithm is meant for use on programs with a high number of decision statements.

Algorithm 1.

The following is the procedure used to determine the SE probabilities.

Step 1. Develop a flow graph of the program code.

Step 2. Represent all the decisions in the program by arbitrary symbols.

Step 3. Develop a table to determine the SE probabilities for the flow graph.

Step 4. Represent all expressions in the program by arbitrary symbols.

Step 5. Determine the SE probabilities for all expressions.

Step 6. Substitute the values of SE probabilities in step 5 into the ones in step 3.

In step 1 of the procedure a control flow graph representation of the program code is developed.

In step 2, during the evaluations of the flow graph, all the decisions in a program are represented by arbitrary symbols.

In step 3, now the SE probabilities for the program can be evaluated directly with each symbol treated as a simple expression. The table for the program can be developed in the same manner as shown in chapter 3.

In step 4, all the expressions in the program, whether compound or simple, are treated as simple expressions during the SE probabilities evaluation. For example, if a program contained the expression " a OR b AND c " in one of its decision statements, then a single symbol "R" can be used to denote the same expression during the calculations.

In step 5, the SE probabilities for each symbol in the program are determined using the table method shown in chapter 3. If a symbol is a simple expression, a table need not be devised.

In step 6. Substitute the SE probabilities for each symbol from step 5 into the formulas of the program's SE probabilities attained in step 3. The SE probabilities for the program are now expressed in terms of each simple expression in the program.

Algorithm 2.

The following procedure is used to find the SE probabilities.

Step 1. Develop a flow graph of the program code.

Step 2. Reduce the flow graph into a main graph and some subgraphs.

Step 3. Represent all the decisions in the program by arbitrary symbols.

Step 4. Determine the SE probabilities for the reduced main graph and the subgraphs.

step 5. substitute the SE probabilities of the subgraphs into the reduced main graph.

Step 6. Represent all expressions in the program by arbitrary symbols.

Step 7. Determine the SE probabilities for all expressions.

Step 8. Substitute the values of SE probabilities in step 7 into the ones in step 5.

If a program has a high number of decision statements in its structure, then the control structure can be reduced by representing decision nodes on the main branches of the structure as single decision nodes. To illustrate the procedure for algorithm 2, consider the flow graph in figure 4.1. The flow graph contains seven decision statements, although not all in sequence. Such a graph is reduced in size in figure 4.2 to simplify the calculations needed for determining the SE probabilities of the structure. The reduction in this case is accomplished by expressing the three decisions b, c, and d by the single decision X. The decisions e, g, and f are expressed by the decision Y. The reduced graph for the structure has only the decisions a, X, and Y. Further reductions may be necessary for X and Y. After the reduction of the structure, the SE probabilities table can be easily developed. Tables can also be developed for X and Y as was shown in chapter 3. The SE probabilities for the main structure are expressed in terms of a, X, and Y. The SE probabilities for X and Y are produced by evaluating the substructures. For Y, this is done by evaluating b and V and substituting their SE probabilities into the expression for Y. For X, this is accomplished by doing the same for U and W. Finally, the values for X and Y are substituted back into the SE probabilities for the main structure. The substitution of the SE probabilities are carried out from the bottom up, until the SE probabilities formulas for the main flow graph contain all the original decisions. In the case of figure 4.1, the formulas attained will contain the decisions a, b, c, d, e, f, and g.

We have found this algorithm to be error prone. It seems to lose evaluation semantics during the reduction process. Algorithm 2 should be limited to the situations in which algorithm 1 is unfeasible.

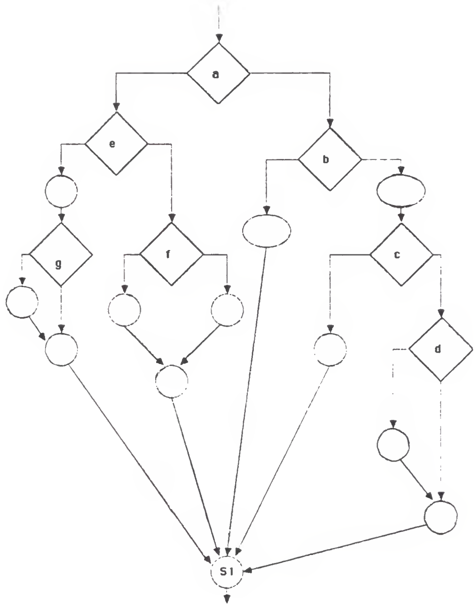

FIGURE 4.1 Sample flow graph for reduction.

The reduced flow graph of figure 4.1 is in figure 4.2.

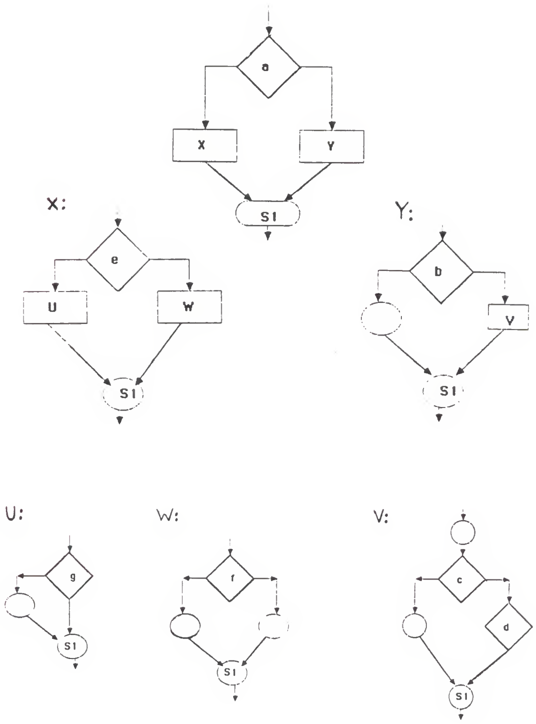

FIGURE 4.2 Reduced flow graph of figure 4.1

Once the use of Algorithm 1 or 2 is finished, the symbolic evaluation phase is completed. Numerical values for the SE probabilities of expressions are now used to formulate the values of the SE probabilities of the program. Hence, we substitute a numerical value for Pt, Pf, Pw, Pc, Ptw, Pfw, Ptc, and Pfc in the formulas attained at the end of applying the algorithms. The values used for the probability of error occurrences in an expression, Pw(expression), are based on the industrial average value of the quality estimations of software. The value of 70 errors per 1000 lines of code tested is considered an average for the software industry. The figure was used in a published report by Linger and Mills [LING88]. Although the figure may not be precise, it is adequate for the purpose of comparing equivalent programs. The average of 70 errors per 1000 lines of code (70 errors/KLOC), includes domain as well as computational errors. Domain errors are the ones of concern to our analysis. Hence, we will use the values Of 100, 10, and 1 error per KLOC to cover the range of the probability of error occurrence in an expression. A value for the probability of an expression being TRUE or FALSE will vary depending on the logical configuration used in the expression. The probability of a FALSE evaluation, Pf, is higher for an expression using an AND condition, "a AND b", than an expression using an OR condition, "a OR b". Therefore, we use the values of 0.2, 0.5, 0.8 to cover the spectrum for the probability of FALSE evaluation for an expression. Once Pf and Pw is known, the other SE probabilities may be attianed as follows:

$Pt = 1 - Pf$

$Pc = 1 - Pw$

$Ptc = Pt * Pc$

$Ptw = Pt * Pw$

$Pfc = Pf * Pc$

$Pfw = PF * Pw$

Next, we will demonstrate the use of algorithm number 1, by comparing two versions of the triangle problem. The triangle problem is widely used in many articles dealing with software testing. In these versions, three integer numbers (bigger than zero) are the inputs for the program. The program produces the type of triangle as an output. The outputs are illegal input, scalene, isosceles, and equilateral. Not a triangle is dropped as an output for simplicity.

4.1 THE TRIANGLE PROBLEM

Example 1.

The flow graph used here is a modified version of the triangle problem shown in [WEYU80]. We can calculate the SE probabilities for the example.

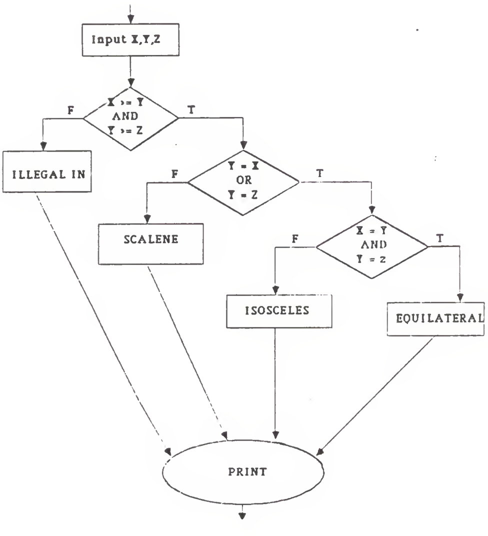

FIGURE 4.3 A flow graph of the triangle problem of example 1.

To determine the SE probabilities for this graph, we devised a table shown in figure 4.4.

R	S	Q	correct	P(R)	P(S)	F(Q)	actual	F(program)
				tc	tc	tc	equilateral	c
				tc	tc	fw	isosceles	wd
				tc	fw	tc	scalene	wd
				tc	fw	fw	scalene	wd
T	T	T	equilateral	fw	tc	tc	illegal in	wd
				fw	tc	fw	illegal in	wd
				fw	fw	tc	illegal in	wd
				fw	fw	fw	illegal in	wd
				tc	tc	tw	equilateral	wd
				tc	tc	fc	isosceles	c
				tc	fw	tw	scalene	wd
				tc	fw	fc	scalene	wd
T	T	F	isosceles	fw	tc	tw	illegal in	wd
				fw	tc	fc	illegal in	wd
				fw	fw	tw	illegal in	wd
				fw	fw	fc	illegal in	wd
T	F	T	NOT POSSIBLE					
				tc	tw	tw	equilateral	wd
				tc	tw	fc	isosceles	wd
				tc	fc	tw	scalene	c
				tc	fc	fc	scalene	c
T	F	F	scalene	fw	tw	tw	illegal in	wd
				fw	tw	fc	illegal in	wd
				fw	fc	tw	illegal in	wd
				fw	fc	fc	illegal in	wd
F	T	T	NOT POSSIBLE					
				tw	tc	tw	equilateral	wd
				tw	tc	fc	isosceles	wd
				tw	fw	tw	scalene	wd
				tw	fw	fc	scalene	wd
F	T	F	illegal in	fc	tc	tw	illegal in	c
				fc	tc	fc	illegal in	c
				fc	fw	tw	illegal in	c
				fc	fw	fc	illegal in	c
F	F	T	NOT POSSIBLE					
				tw	tw	tw	equilateral	wd
				tw	tw	fc	isosceles	wd
				tw	fc	tw	scalene	wd
				tw	fc	fc	scalene	wd
F	F	F	illegal in	fc	tw	tw	illegal in	c
				fc	tw	fc	illegal in	c
				fc	fc	tw	illegal in	c
				fc	fc	fc	illegal in	c

FIGURE 4.4 A table for the flow graph in example 1.

We start by determining the SE probabilities for the program from the table in figure 4.4. In the table, "R" represents "X >= Y AND Y >=Z", "S" represents "Y = X OR Y = Z", and "Q" represents "X = Y AND Y = Z". The SE probabilities are:

(a) The correctness is

Pc(program) = Pfc(R)*Pw(S)*Pfc(Q) + Pc(R)*Pfc(S)*Pfc(Q) + Pfc(R)*Pfw(S)*Ptw(Q) + Pc(R)*Pfc(S)*Ptw(Q) + Pc(R)*Ptc(S)*Pfc(Q) + Pfc(R)*Ptc(S)*Ptw(Q) + Ptc(R)*Ptc(S)*Ptc(Q) + Pfc(R)*Ptw(S)*Ptw(Q)

(b) The incorrectness is

Pw(program) = 1 - Pc(program)

(c) The unsafety is

Pus(program) = 0

(d) The safety is

Ps(program) = 1

Next the SE probabilities for the expressions "R", "S", and "Q" need to be determined. Let the expression "R" be "a AND b", where "a" is "X >= Y" and "b" is "Y >= Z". Let "S" be "i OR j", where "i" is "Y = X" and "j" is "Y = Z". Let "Q" be "M AND K", where "M" is "X = Y" and "K" is "Y = Z".

The table for the operators "AND" in figure 3.2 and the table for "OR" in figure 3.6 are used to determine the SE probabilities for the expressions.

(a) For the expression "R",

```
Ptc(R) = Ptc(a)*Ptc(b)
Pfc(R) = Pfc(a)*Pc(b) + Ptc(a)*Pfc(b)
Ptw(R) = Ptc(a)*Ptw(b) + Ptw(a)*Pt(b)
Pfw(R) = Pfw(a) + Ptw(a)*Pf(b) +
         Ptc(a)*Pfw(b) + Pfc(a)*Pw(b)
Pw(R)  = Pw(a) + Pc(a)*Pw(b)
Pc(R)  = Pc(a)*Pc(b)
```

(b) For the expression "S",

```
Ptc(S)  = Ptc(i)*Pc(j) + Pfc(i)*Ptc(j)
Pfc(S)  = Pfc(i)*Pfc(j)
Ptw(S)  = Ptw(i) + Pfw(i)*Pt(j) +
          Pfc(i)*Ptw(j) + Ptc(i)*Pw(j)
Pfw(S)  = Pfw(i)*Pf(j) + Pfc(i)*Pfw(j)
Pw(S)   = Pw(i) + Pc(i)*Pw(j)
Pc(S)   = Pc(i)*Pc(j)
```

(c) For the expression "Q",

```
Ptc(Q)  = Ptc(m)*Ptc(k)
Pfc(Q)  = Pfc(m)*Pc(k) + Ptc(m)*Pfc(k)
Ptw(Q)  = Ptc(m)*Ptw(k) + Ptw(m)*Pt(k)
```

$$Pfw(Q) = Pfw(m) + Ptw(m)*Pf(k) + Ptc(m)*Pfw(k) + Pfc(m)*Pw(k)$$

$$Pw(Q) = Pw(m) + Pc(m)*Pw(k)$$

$$Pc(Q) = Pc(m)*Pc(k)$$

Finally, by substituting the formulas for the expressions "R", "S", and "Q" in the SE probabilities for the program, the resultant probabilities for the program follow.

(a) The correctness is

$$\begin{aligned}
&Pc(program) = \{[(Pfc(a)*Pc(b)+Ptc(a)*Pfc(b))* \\
&(Pw(i)+Pc(i)*Pw(j))*(Pfc(m)*Pc(k)+Ptc(m)*Pfc(k))]+ \\
&[(Pc(a)*Pc(b))*(Pfc(i)*Pfc(j))*(Pfc(m)*Pc(k)+ \\
&Ptc(m)*Pfc(k))]+[(Pfc(a)*Pc(b)+Ptc(a)*Pfc(b))* \\
&(Pfw(i)*Pfw(j)+Pfc(i)*Pfw(j))*(Ptc(m)*Pw(k)+ \\
&Ptw(m)*Pt(k))]+[(Pc(a)*Pc(b))*(Pfc(i)*Pfc(j))* \\
&(Ptc(m)*Ptw(k)+Ptw(m)*Pt(k))]+[(Pc(a)*Pc(b))* \\
&(Ptc(i)*Pc(j)+Pfc(i)*Ptc(j))*(Pfc(m)*Pc(k)+ \\
&Ptc(m)*Pfc(k))]+[(Pfc(a)*Pc(b)+Ptc(a)*Pfc(b))* \\
&(Ptc(i)*Pc(j)+Pfc(i)*Ptc(j))*(Ptc(m)*Ptw(k)+ \\
&Ptw(m),Pt(k))]+[(Ptc(a)*Ptc(b))(Ptc(i)*Pc(j)+ \\
&Pfc(i)*Ptc(j))*(Ptc(m)*Ptc(k))]+[(Pfc(a)*Pc(b)+ \\
&Ptc(a)*Pfc(b))*(Ptw(i)+Pfw(i)*Pt(j)+Pfc(i)*Ptw(j)+ \\
&Ptc(i)*Pw(j))(Ptc(m)*Ptw(k)+Ptw(m)*Pt(k))]\}
\end{aligned}$$

(b) The incorrectness is

Pw(program) = 1 - Pc(program)

(c) The unsafety is

Pus(program) = 0

(d) The safety is

Ps(program) = 1

Values for the probability of correctness and incorectness for the program are obtained by using the values recommended at the end of the procedure in section 4.0. The probabilities of safety and unsafety for the program are known. The results of probabilities of correctness and incorrectness,

(a) When Pw = 0.1 and Pf = 0.2 ,

Pc = 0.454

Pw = 0.546

(b) When Pw = 0.1 and Pf = 0.5 ,

Pc = 0.524

Pw = 0.476

(c) when Pw = 0.1 and Pf = 0.8 ,

Pc = 0.631

Pw = 0.369

(d) when $P_w = 0.01$ and $P_f = 0.2$,

$P_c = 0.716$

$P_w = 0.284$

(e) when $P_w = 0.01$ and $P_f = 0.5$,

$P_c = 0.765$

$P_w = 0.235$

(f) when $P_w = 0.01$ and $P_f = 0.8$,

$P_c = 0.923$

$P_w = 0.077$

(g) when $P_w = 0.001$ and $P_f = 0.2$,

$P_c = 0.749$

$P_w = 0.251$

(h) when $P_w = 0.001$ and $P_f = 0.5$,

$P_c = 0.794$

$P_w = 0.206$

(i) when $P_w = 0.001$ and $P_f = 0.8$,

$P_c = 0.957$

$P_w = 0.043$

Example 2.

This is the second version of the triangle problem. The structure used is different from that given in example 1. All of the decision statements in example 2 are arranged in sequence, while the decisions statements in example 1 are nested . The flow graph used in example 2 was developed by Dr. David A. Gustafson.

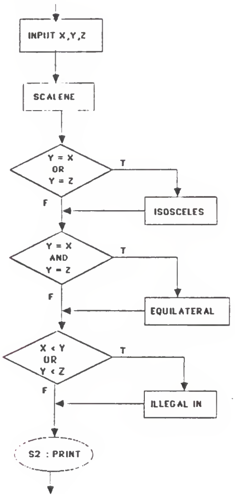

FIGURE 4.5 A flow graph of the triangle problem

To determine the SE probabilities for this graph, we devised the table in figure 4.6.

R	S	Q	correct	F(R)	F(S)	F(Q)	actual	F(program)
T	T	T	NOT POSSIBLE					
				tc	tc	tw	illegal in	wd
				tc	tc	fc	equilateral	c
				tc	fw	tw	illegal in	wd
				tc	fw	fc	isosceles	wd
T	T	F	equilateral	fw	tc	tw	illegal in	wd
				fw	tc	fc	equilateral	wu
				fw	fw	tw	illegal in	wd
				fw	fw	fc	scalene	wd
				tc	tw	tc	illegal in	wu
				tc	tw	fw	equilateral	wd
				tc	fc	tc	illegal in	c
				tc	fc	fw	isosceles	wd
T	F	T	illegal in	fw	tw	tc	illegal in	wu
				fw	tw	fw	equilateral	wd
				fw	fc	tc	illegal in	wu
				fw	fc	fw	scalene	wd
				tc	tw	tw	illegal in	wd
				tc	tw	fc	equilateral	wd
				tc	fc	tw	illegal in	wd
				tc	fc	fc	isosceles	c
T	F	F	isosceles	fw	tw	tw	illegal in	wd
				fw	tw	fc	equilateral	wd
				fw	fc	tw	illegal in	wd
				fw	fc	fc	scalene	wd
F	T	T	NOT POSSIBLE					
F	T	F	NOT POSSIBLE					
				tw	tw	tc	illegal in	wu
				tw	tw	fw	equilateral	wd
				tw	fc	tc	illegal in	wu
				tw	fc	fw	isosceles	wd
F	F	T	illegal in	fc	tw	tc	illegal in	wu
				fc	tw	fw	equilateral	wd
				fc	fc	tc	illegal in	c
				fc	fc	fw	scalene	wd
				tw	tw	tw	illegal in	wd
				tw	tw	fc	equilateral	wd
				tw	fc	tw	illegal in	wd
				tw	fc	fc	isosceles	wd
F	F	F	scalene	fc	tw	tw	illegal in	wd
				fc	tw	fc	equilateral	wd
				fc	fc	tw	illegal in	wd
				fc	fc	fc	scalene	c

FIGURE 4.6 A table for the flow graph in example 2.

We start by determining the SE probabilities for the program from the table in figure 4.6. In the table "R" represents "X = Y OR Y =Z", also the "S" represents "Y = X AND Y = Z", and "Q" is to represent "X < Y OR Y < Z". The probabilities are:

(a) The correctness is

Pc(program) = Ptc(R)*Ptc(S)*Pfc(Q) +
Pc(R)*Pfc(S)*Pc(Q)

(b) The incorrectness is

Pw(program) = 1 - Pc(program)

(c) The unsafety is

Pus(program) = Pwu(program)
= Ptw(S)*Ptc(Q) + Pw(R)*Pfc(S)*Ptc(Q) +
Pfw(R)*Ptc(S)*Pfc(Q)

(d) The safety is

Ps(program) = 1 - Pus(program)

Next, the SE probabilities for the expressions "R", "S", and "Q" need to be determined. Let the expression "R" be "a OR b", where "a" is "X = Y" and "b" is "Y = Z". Let "S" be "i AND j", where "i" is "Y = X" and "j" is "Y = Z". Let "Q" be "M

OR K", where "M" being "X < Y" and "K being "Y < Z". The table for the logical operators "AND" in figure 3.2 and the table for "OR" in figure 3.6 are used to determine the SE probabilities for the expressions. The results are,

(a) For the expression "R",

Ptc(R) = Ptc(a)*Pc(b) + Pfc(a)*Ptc(b)

Pfc(R) = Pfc(a)*Pfc(b)

Ptw(R) = Ptw(a) + Pfw(a)*Pt(b) +
Pfc(a)*Ptw(b) + Ptc(a)*Pw(b)

Pfw(R) = Pfw(a)*Pf(b) + Pfc(a)*Pfw(b)

Pw(R) = Pw(a) + Pc(a)*Pw(b)

Pc(R) = Pc(a)*Pc(b)

(b) For the expression "S",

Ptc(S) = Ptc(i)*Ptc(j)

Pfc(S) = Pfc(i)*Pc(j) + Ptc(i)*Pfc(j)

Ptw(S) = Ptc(i)*Ptw(j) + Ptw(i)*Pt(j)

Pfw(S) = Pfw(i) + Ptw(i)*Pf(j) +
Ptc(i)*Pfw(j) + Pfc(i)*Pw(j)

Pw(S) = Pw(i) + Pc(i)*Pw(j)

Pc(S) = Pc(i)*Pc(j)

(c) For the expression "Q",

Ptc(Q) = Ptc(m)*Pc(k) + Pfc(m)*Ptc(k)

Pfc(Q) = Pfc(m)*Pfc(k)

Ptw(Q) = Ptw(m) + Pfw(m)*Pt(k) +
Pfc(m)*Ptw(k) + Ptc(m)*Pw(k)

Pfw(Q) = Pfw(m)*Pf(k) + Pfc(m)*Pfw(k)

Pw(Q) = Pw(m) + Pc(m)*Pw(k)

Pc(Q) = Pc(m)*Pc(k)

Finally, by substituting the formulas for the expressions "R", "S", and "Q" in the SE probabilities for the program, the resultant probabilities for the program are obtained.

(a) The correctness is

Pc(program) = {[(Ptc(a)*Pc(b)+Pfc(a)*Ptc(b))*
(Ptc(i)*Ptc(j))*(Pfc(m)*Pfc(k))] +
[(Pc(a)*Pc(b))(Pfc(i)*Pc(j)+Ptc(i)*Pfc(j))*
(Pc(m)*Pc(k))]}

(b) The incorrectness is

Pw(program) = 1 - Pc(program)

(c) The unsafety is

```
Pus(program) = {[(Ptc(i)*Ptw(j)+Ptw(i)*Pt(j))*
               (Ptc(m)*Pc(k)+Pfc(m)*Ptc(k))]+
               [(Pw(a)+Pc(a)*Pw(b))(Pfc(i)*Pc(j)+
               Ptc(i)*Pfc(j))*(Ptc(m)*Pc(k)+
               Pfc(m)*Ptc(k))]+[(Pfw(a)*Pf(b)+
               Pfc(a)*Pfw(b))*(Ptc(i)>Ptc(j))*
               (fc(m)*Pfc(k))]}
```

(d) The safety is

```
Ps(program) = 1 - Pus(program)
```

Values for the probability of correctness and incorectness for the program are obtained by using the values recommended at the end of the procedure in section 4.0. The probability results are,

(a) When Pw = 0.1 and Pf = 0.2 ,

Pc = 0.204

Pw = 0.796

Ps = 0.862

Pus = 0.138

(b) When Pw = 0.1 and Pf = 0.5 ,

Pc = 0.423

Pw = 0.577

Ps = 0.899

Pus = 0.101

(c) When Pw = 0.1 and Pf = 0.8 ,

Pc = 0.515

Pw = 0.485

Ps = 0.953

Pus = 0.047

(d) When Pw = 0.01 and Pf = 0.2 ,

Pc = 0.362

Pw = 0.638

Ps = 0.981

Pus = 0.019

(e) When Pw = 0.01 and Pf = 0.5 ,

Pc = 0.750

Pw = 0.250

Ps = 0.985

Pus = 0.015

(f) When $Pw = 0.01$ and $Pf = 0.8$,

$Pc = 0.912$

$Pw = 0.088$

$Ps = 0.993$

$Pus = 0.007$

(g) When $Pw = 0.001$ and $Pf = 0.2$,

$Pc = 0.382$

$Pw = 0.618$

$Ps = 0.998$

$Pus = 0.002$

(h) When $Pw = 0.001$ and $Pf = 0.5$,

$Pc = 0.792$

$Pw = 0.208$

$Ps = 0.998$

$Pus = 0.002$

(i) When $Pw = 0.001$ and $Pf = 0.8$,

$Pc = 0.963$

$Pw = 0.037$

$Ps = 0.999$

$Pus = 0.001$

4.2 RESULTS

In analyzing the resultant values of the SE probabilities of example 1 and 2 shown in figures 4.7 and 4.8, the following was observed.

PROBABILITY OF FALSE

PROBABILITY OF WRONG			0.2	0.5	0.8
0.1	figure 4.3	S	1.0	1.0	1.0
		US	0.0	0.0	0.0
	figure 4.5	S	0.862	0.899	0.953
		US	0.138	0.101	0.047
0.01	figure 4.3	S	1.0	1.0	1.0
		US	0.0	0.0	0.0
	figure 4.5	S	0.981	0.985	0.993
		US	0.019	0.015	0.007
0.001	figure 4.3	S	1.0	1.0	1.0
		US	0.0	0.0	0.0
	figure 4.5	S	0.998	0.998	0.999
		US	0.002	0.002	0.001

FIGURE 4.7 A table for safety and unsafety of example 1 and 2.

PROBABILITY OF FALSE

PROBABILITY OF WRONG			0.2	0.5	0.8
0.1	figure 4.3	C	0.454	0.524	0.631
		V	0.546	0.476	0.369
	figure 4.5	C	0.204	0.423	0.515
		V	0.796	0.577	0.485
0.01	figure 4.3	C	0.716	0.765	0.923
		V	0.284	0.235	0.077
	figure 4.5	C	0.362	0.750	0.912
		V	0.638	0.250	0.088
0.001	figure 4.3	C	0.749	0.794	0.957
		V	0.251	0.206	0.043
	figure 4.5	C	0.382	0.792	0.963
		V	0.618	0.208	0.037

FIGURE 4.8 A table for correctness and incorrectness of example 1 and 2.

(a) The logical structure of example 1 is a better structure from a testing stand point. This is due to its ability to detect possible error occurrence in all the examined conditions. The nesting of the decisions in example 1 has improved the testability of the structure. Nesting does not always improve testability, especially when the structure has decision branches leading directly to an exit node, as errors may go undetected.

It is clear from the results of example 1 and 2 that the logical structure of a program effects its safety and testability. We have attained different values for the SE probabilities of the same problem by expressing it in two diffenent logical structures. To illustrate the difference between the two examples in the detection of errors we have introduced the same error in the condition "Y = X OR Y = Z", which is the same in both examples, to make it say " Y < X OR Y < Z". We also applied the same test point of (5, 5, 5) to both examples. In example 1, the expected result is a triangle of type equilateral and the result of applying the test point to the program is a triangle of type scalene. This makes the error in the condition detectable, since the two triangle types in the actual and expected results are different. The actual triangle type of applying the test point to the version in example 2 is equilateral and expected triangle type is equilateral, which makes the error in the condition undetectable. Errors in conditions not examined in both examples will not be detected from the comparison of actual to expected results.

(b) In both examples, as Pw(expr) decreased to a minimum, the Pc(program) increased. The increase was expected, because Pc(program) is dependent on the SE probabilities of its components. As the Pw(expr) decreased so should Pw(program). Normally we desire an increase of Pc(program), which leads to an increase of program safety. In example 1 of this chapter an increase in Pc(program) is not crucial due to Pwu(program) being equal to zero. In example 2 this point is apparent from the results. As the Pw(expr) decreased, the Pc(program) and Ps(program) increased. The safety and correctness of a program increase when the probability of error decreases.

(c) In both examples, as Pf(expr) increased to a maximum, the Pc(program) increased. In figure 4.3, most of the combinations in which the program is correctly evaluated, i.e. Pc(program), occur when the condition's logical configration is FTF or FFF. Therefore, the Pc(program) should increase with an increase in Pf(expr). As Pw(expr) increases the Pc(program) should decrease at a higher rate for smaller values of Pf(expr). The highest possible value for Pc(program) is attained when Pw(expr) is low (0.001) and Pf(expr) is high (0.8). The results from the example seem to agree. when the program in figure 4.5 is correctly evaluated, there are more terms with Pf(expr) than there are with Pt(expr). Hence, the Pc(program) should increase as the Pf(expr) increases. The results from the example agree. The ratio of Pt(expr) to Pf(expr) in any one combination depends on the logical configuration of the expression under examination. This ratio influences the safety and correctness of the program.

(d) Pc(program) for figure 4.3 is higher than that of figure 4.5 in all the combinations of different values of Pw(expr) and Pf(expr), except for the case in which Pw(expr) is equal to 0.001 and Pf(expr) is equal to 0.8. The difference in value of Pc(program) between the two versions is due to the number of combinations in which the program is evaluated correctly. Thus, structure nesting is clearly a factor that influences correctness.

CHAPTER 5

CONCLUSIONS AND AREAS OF FUTURE RESEARCH

5.0 CONCLUSIONS

We have proposed a new method for calculating the safety (a measure of testability) of expressions, segments, and programs. The method is based on evaluating the logical structure used, and on determining the probable outcome of every tested condition in the program. The testability of functionally equivalent but logically different structures can be compared using this method, and the effect of the program structure on testability can be determined.

The testability of a program is expressed in terms of the safety of its components and not in terms of the number of errors detected or removed from the program. Simulations of program behavior during testing are made possible, and program unsafety, in which errors are not revealed under testing, may be detected.

The practicality of the proposed method lies in its ability to provide a numerical measure of testability based solely on the structure used. This measure will aid in the selection of logical structures in which error existence has a high probability of being detected. This measure also provides guidance into what conditions and paths need closer examination during testing. When estimating the program unsafety, the method compensates for the situations in which coincidental correctness and missing path errors may occur. Large values of unsafety should warrant the selection of an alternative structure to improve the quality of testing. We used our method to analyze two versions of the triangle problem. We found the method helpful in identifying the more testable structure and in predicting situations in which errors may go undetected.

5.1 AREAS OF FUTURE RESEARCH

Throughout the development of our methodology, many issues were encountered that are worthy of further research and investigation.

1. The value for the probability of error existence in an expression Pw(expr), in chapter 4, was based on the average of all errors occurring per line of code regardless of the error type. Although this average may be adequate for comparing programs with equivalent structures, a more accurate value is required for analyzing programs with different logical structures. An average value of error occurrences in expressions needs to be established through research.

2. Two different approaches are used for evaluating the SE probabilities. The short circuit approach is the one used to evaluate the logical structure of the program itself, and the complete evaluation approach is used for the expressions within conditional statements. The difference between the two is in the treatment of unevaluated conditions. Further investigation is needed to determine the more effective approach in determining the SE probabilities.

3. The problem of coincidental correctness is compensated for in the safety estimations of all structures. The problem of missing path errors is compensated for in the safety estimations of expressions. A study of the effectiveness of our

approach in the treatment of the two problems is needed to establish the usefulness of the method.

4. We linked our concept of safety to program testability. Further research is needed to determine if a relationship exists between our concept of safety and the traditional concepts of program reliability.

5. Our results indicate that by changing the program structure we can change its safety. An important question to be answered is " should the testing of unsafe structures take precedence over the testing of safe structures ?". Further investigations are needed to answer this question.

REFERENCES

[CHEU80] R.C. Cheung. "A User-Oriented Software Reliability Model", IEEE Trans. Software Eng., Vol.SE-6, No.2, pp. 118-125, March 1980.

[CLAR82] L.A. Clarke, J. Hassell, and D.J. Richardson. "A Close Look at Domain Testing", IEEE. Trans Software Eng., Vol. SE-8, No.4, pp. 380-390, July 1982.

[DOWN85] T. Downs. "An Approach to the Modeling of Software Testing with Some Applications", IEEE Trans. Software Eng., Vol. SE-11, No.4, pp. 375-386, April 1985.

[DOWN86] T. Downs. "Extensions to an Approach to the Modeling of Software Testing with Some Performance Comparisons", IEEE Trans. Software Eng., Vol. SE-12, No.9, pp. 979-987, Sept. 1986.

[DURA84] J. Duran and S. Ntafos. "An Evaluation of Random Testing", IEEE Trans. Software Eng., Vol. SE-10, No.4, pp. 438-444, July 1984.

[GELP88] D. Gelperin and B. Hetzel. "The Growth of Software Testing", Comm. of ACM, Vol. 31, No.6, pp. 687-695, June 1988.

[GOOD75] J.B. Goodenough and S.L. Gerhart. "Toward a Theory of Test Data Selection", IEEE Trans. Software Eng., Vol. SE-1, No.2, pp. 156-173, June 1975.

[GOUR83] J.S. Gourlay. "A Mathematical Framework for the Investigation of Testing", IEEE Trans. Software Eng., Vol. SE-19, No. 6, pp. 686-704, Nov. 1983.

[HAML81] R. Hamlet. "Reliability Theory of Program Testing", Actu Informatica, 16, pp. 31-43, 1981.

[HAML88] R. Hamlet. "Special Section on Software Testing", Comm. of ACM, Vol. 31, No.6, pp. 662-667, June 1988.

[HOWD76] W.E. Howden. "Reliability of the Path Analysis Testing Strategy", IEEE Trans. Software Eng., Vol. SE-2, pp. 208-214, Sept. 1976.

[HOWD85] W.E. Howden. "The Theory and Practice of Functional Testing", IEEE Software, pp. 6-17, Sept. 1985.

[HOWD86] W.E. Howden. "A Functional Approach to Program Testing and Analysis", IEEE Trans. Software Eng., Vol. SE-12, No. 10, pp. 997-1005, Oct. 1986.

[LING88] R.C. Linger and H.D. Mills. "A Case Study in Cleanroom Software Engineering: The IBM COBOL Structuring Facility", Computer Society Press of the IEEE, 5-7, pp. 10-17, Oct. 1988.

[NTAF88] S. Ntafos. "A Comparison of Some Structural Testing Strategies", IEEE Trans. Software Eng., Vol. SE-14, No.6, pp. 868-874, June 1988.

[NTAF84] S.C. Ntafos. "On Required Element Testing", IEEE Trans. Software Eng., Vol. SE-10, pp. 795-803, Nov. 1984.

[RAPP85] S. Rapps and E. Weyuker. "Selecting Software Test Data Using Data Flow Information", IEEE Trans. Software Eng., Vol. SE-11, No.4, pp. 367-375, April 1985.

[ROSS85] S.M. Ross. "Software Reliability: The Stopping Rule Problem", IEEE Trans. Software Eng., Vol. SE-11, No.12, pp. 1472-1476, Dec. 1985.

[WEYU80] E. Weyuker and T. Ostrand. "Theories of Program Testing and the Application of Revealing Subdomains", IEEE Trans. Software Eng., Vol. SE-6, No.3, pp.236-246, May 1980.

[WHIT80] L. White and E. Cohen. "A Domain Strategy for Computer Program Testing", IEEE Trans. Software Eng., Vol. SE-6, No.3, pp. 247-257, May 1980.

THE SAFETY OF LOGICAL STRUCTURES

by

KHALED ALI DAWOUD AL-ALI

B.S., The University of Wyoming, 1978

AN ABSTRACT OF A THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

COMPUTER SCIENCE

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1989

ABSTRACT

This thesis proposes a method for estimating the testability of a program based on its logical structure. A new concept called safety is introduced as a measure of testability. The safety of a program is defined as the probability of correctly evaluating the structure of a program, with the ability to detect errors in all evaluated conditions. Using this concept, inference about the behavior of the program under testing is possible. Equivalent logical structures could be compared for their testability. Conditions and paths requiring closer examination during testing are identified to improve the testing of programs.