Abstract: Choosing effective tests for complex logical expressions can be difficult. This paper describes a black-box strategy for choosing such tests. The strategy is shown to be a natural generalization of current methods for choosing tests for simpler expressions. The strategy chooses combinations that are not only effective at revealing implementation defects but, perhaps more important, are effective at helping detect defects in the requirements themselves.

In addition, if there are dependencies between variables in the required logic, a variant of the strategy excludes those truth-value combinations that are "impossible" because of the dependencies. It extends the required expression to include dependency information and then derives tests from this extension. This strategy is the only test selection method that smoothly incorporates dependency information. It also permits the dependency information itself to be partially validated through analysis of the "impossible" cases developed by the basic strategy, but excluded by the variant.

Prerequisite Key Words: Boolean Algebra

Topic Descriptors: Requirements-based Testing, Black-box Testing, Logic/Decision Testing, Normal Form Logical Expressions, Requirements Validation, Impossible Test Cases

1 Introduction

Consider a process control program that sends commands to a valve controller when the level of liquid in a storage container gets "too deep" (condition A) or the temperature of the liquid gets "too high" (condition B) or optical characteristics of the liquid change "too much" (condition C). Thus the program must correctly implement the Boolean expression in If (A or B or C), send commands Endif.

Choosing effective tests for complex logic can be difficult. The difficulty is dependent on the number and arrangement of Boolean variables in the required logical expressions. If the number is "small", the best thing to do is test all combinations. For the example above, this would mean testing the 8 different truth-value assignments.

When the number of truth-value combinations is unmanageable (either psychologically or economically), one attempts to choose an effective subset. Expressions of six or more Boolean variables appearing in financial, insurance, and aerospace applications can be a challenge to understand due to the arrangement of their variables (e.g., (A & (B' or C' or BCD'E)) or F). Expressions of 10 or more Boolean variables have been reported [LH94] in aerospace applications (i.e., more than a thousand truth-value combinations).

This paper describes two black-box strategies for choosing test sets. The strategies are specified in detail and logical justification is provided for their use. While the logical case is strong, experimental
investigation of their scope of critical application and their defect detection efficiency compared to competitive techniques remains to be done.

2 Previous Work

Various strategies, both black-box and code-based, have been proposed for testing complex logic [Be90], [CM94], [Fo84], [Ma94], [Ta90], [WG94]. Only recently has a comparison of some of the techniques been attempted [WG94]. We note that one method is required for testing critical aerospace software [CM94], [Rt92] without any published comparison data to justify the choice. Verifiable comparative studies [He95] need to be done.

3 Normal Forms

To understand the test strategies, the reader must first understand the process in Boolean Algebra [Ko78] of "normalizing" a logical expression because the strategies will select truth-value combinations based on these normalized expressions.

We will use (A & (B' or C')) or C'D as our sample expression. We call A, B, C, and D the Boolean variables (i.e., variables that only take values TRUE or FALSE). We call either a Boolean variable or its negation (e.g., C or C') a literal. Note that for the expression AB is equivalent to A & B i.e., the “and” is implied.

Every Boolean expression has both a unique disjunctive (ored) and a unique conjunctive (anded) normal form, both of which are truth value equivalent to the original (i.e., they are both TRUE and FALSE at exactly the same times). The sample expression (A & (B' or C)) or C'D is equivalent to disjunctive normal form:

$$\text{AB'CD or AB'CD' or AB'C'D or ABCD or ABCD' or ABC'D or A'BC'D or A'B'C'D}$$

Note that this normal form expression has nine disjunctive terms (i.e., series of anded literals). Together they represent the complete set of truth-value assignments that make this function TRUE. Since A'BC'D is a term (the 8th), the assignment f t f t will make the original expression TRUE.

The remaining seven truth-value assignments are the conjunctive terms found in the conjunction normal form expression

$$(\text{A' or B' or C or D}) \& (\text{A or B or C' or D}) \& (\text{A or B or C' or D'}) \& (\text{A' or B' or C' or D}) \& (\text{A or B' or C' or D'}) \& (\text{A or B or C or D}) \& (\text{A or B' or C or D}),$$

Conjunctive terms represent truth-value assignments that make the expression FALSE. Since (A or B or C' or D') is a term (the 4th), assignment f t t t will make the original expression FALSE.

We use normal terms to refer to either disjunctive or conjunctive terms. We define minimality of a Boolean normal form expression as having minimal numbers of normal terms and then literals within those terms. Using logical equivalencies such as AB or AB' = A, a "reducible" normal form
expression can be transformed into a minimal expression that is truth value equivalent. For the example above, the minimal disjunctive normal form expression is \( AB' \) or \( AC \) or \( C'D \).

### 4 Basic Unique Cause Strategy

The basic strategy for test selection based on a required Boolean expression has the following 4 steps:

[1] transform the required logical expression into a minimal disjunctive normal form,

[2] transform the required expression into a minimal conjunctive normal form,

[3] for each disjunctive term, choose a combination of truth values that makes that term (i.e., the causative term) TRUE, all others FALSE, and a maximum number of literals in the remaining terms FALSE,

[4] for each conjunctive term, choose a combination of truth-values that makes that term (i.e., the causative term) FALSE, all others TRUE, and a maximum number of literals in the remaining terms TRUE.

For \( A \lor B \lor C \), the minimal disjunctive normal form is \( A \lor B \lor C \), the minimal conjunctive normal form is \( (A \lor B \lor C) \), and the chosen truth value combinations are \( tff, tft, fff \). For \( A \land B \land C \), the minimal disjunctive normal form is \( ABC \), the minimal conjunctive normal form is \( (A) \land (B) \land (C) \), and the chosen truth value combinations are \( ttt, ftt, tft, tff \). Thus the \( n+1 \) test cases usually suggested for such expressions can be seen to result from applying the Basic Unique Cause Strategy.

For sample expression \( (A \land (B' \lor C')) \lor C'D \), the basic strategy yields:

[1] \( AB' \) or \( AC \) or \( C'D \)

[2] \( (B' \lor C \lor D) \) (\( A \lor C' \) (A or D))

[3] \( tff \) Test Case BT1 - Basic True I
   \( ttt \) Test Case BT2
   \( ftf \) Test Case BT3

[4] \( tff \) Test Case BF1
   \( ftt \) Test Case BF2
   \( fff \) Test Case BF3

Each chosen truth value combination represents either a minimum number of TRUE literals resulting in a TRUE logical expression (step 3) or a minimum number of FALSE literals resulting in a FALSE logical expression (step 4). Note that if the logical expression is TRUE for a test case (from step 3), one and only one of its terms is TRUE, and that if the expression is FALSE (from step 4), one and only one of its terms is FALSE. It is this characteristic that gives the strategy its name. The program's behavior is uniquely caused by a minimal subset of truth-values.
Each step in the strategy may have alternative results and thus several different test sets may be possible. However, each of the resulting test sets will have the properties described above. While steps 1 and 2 are non-deterministic because reduced normal forms may not be unique [Ko78], steps 3 and 4 are non-deterministic when there are pairs of complementary literals (e.g., D and D') among the non-causative terms.

5 Validating the Required Expressions

A major concern, when implementing complex decision logic, is whether the specified logical expression is correct (i.e., does it accurately and completely capture all aspects of the conditional requirement). The more complex the logic, the greater the concern. The well-known limitations of the human mind when "understanding" and accurately assessing the correctness of complex expressions motivate this concern.

To assess correctness with a reasonable degree of confidence requires "divide and conquer" mechanisms. One such mechanism is the use of a hierarchy of "abstract" Boolean variables to specify the required expression. Each abstract variable encodes the meaning of a smaller Boolean expression in minimal normal form (e.g., $A \lor B$, where $A = (C \lor D) \land (C \lor E)$, $B = F \land (G \lor H)$, and $C = P \land Q$). $A$ might "mean" that the 'runway is adequate' and $B$ might mean that 'aircraft is critically low on fuel'.

A second mechanism is the use of "small" examples to analyze the behavior of the specified logic. Test cases developed using a Unique Cause strategy contain the smallest possible examples, by design, and each case contains an important example. Only the causative term values in each case need to be considered. At a minimum, each of these Unique Cause examples should be carefully reviewed with a subject matter expert to assure that the specified logical expression behaves correctly for these causative values. In [LH94], Leveson reports on the specification value of disjunctive normal form. We contend that conjunctive normal form has psychological verification value as well.

6 Incorporating Dedendency Information

If there are dependencies between variables in the required logical expression, the Dependent Unique Cause Strategy is used. This strategy excludes those input combinations that are "impossible" because of dependencies. It extends the required expression to include dependency information and then normalizes this extension.

The dependent UC strategy has the following 6 steps:

[1] transform the required logical expression into a minimal disjunctive normal form,

[2] AND disjunctive forms of each dependency onto the required expression, (This forces all impossible combinations to be FALSE.) and transform this modified expression into a minimal disjunctive normal form,
[3] for each causative term in the modified expression, choose a combination of truth values that makes that term TRUE, all possible terms in the original expression FALSE, and a maximum number of literals in terms of the original expression FALSE,

[4] transform the required expression into a minimal conjunctive normal form,

[5] OR conjunctive forms of the negation of each dependency onto the required expression, (This forces all impossible combinations to be TRUE), and transform this modified expression into a minimal conjunctive normal form,

[6] for each causative term in the modified expression, choose a combination of truth values that makes that term FALSE, all possible terms in the original expression TRUE, and a maximum number of literals in terms of the original expression TRUE.

For sample expression \((A \& (B' \lor C')) \lor C'D\) when \(A\) implies \(D\), the Dependent Unique Cause Strategy yields:

\[\begin{align*}
\text{(1)} & \quad (AB') \lor (AC) \lor (C'D) \\
\text{(2)} & \quad [(A \& (B' \lor C')) \lor C'D] [A' \lor D] \\
& \quad (AD) \lor (C'D) \\
\text{(31)} & \quad t \ t \ f \ t \quad \text{Test Case DT1 - Dependent True 1} \\
& \quad f \ t \ f \ t \quad \text{Test Case DT2} \\
\text{(4)} & \quad (A' \lor B' \lor C \lor D) (A \lor C') (A \lor D) \\
\text{(5)} & \quad (A \& (B' \lor C')) \lor C'D \lor AD' \\
& \quad (A \lor C') (A \lor D) \\
\text{(61)} & \quad f \ f \ t \ t \quad \text{Test Case DF1} \\
& \quad f \ f \ f \ f \quad \text{Test Case DF2}
\end{align*}\]

7 Validating Dependency Information

By developing tests using both Basic and Dependent Unique Cause, the quality of the dependency information can be partially evaluated. All test cases that are in the Basic set, but not in the Dependent set are "impossible" based on dependency information. To check the completeness and accuracy of this information do the following:

1) Check "possibility" of Dependent cases. Cases that are not possible indicate the dependency information is incomplete and must be updated.
2) Check "impossibility" of excluded Basic cases. Cases that turn out to be possible indicate the dependency information is inaccurate and must be corrected. For sample expression \((A \& (B' \text{ or } C'))\) or \(C'D\), when \(A\) implies \(D\), the excluded cases are:

<table>
<thead>
<tr>
<th>t f f f</th>
<th>Test Case BT1</th>
</tr>
</thead>
<tbody>
<tr>
<td>t t f f</td>
<td>Test Case BT2</td>
</tr>
<tr>
<td>t t f f</td>
<td>Test Case BF1</td>
</tr>
</tbody>
</table>

8 Limitations of Unique Cause Strategies

Too Many Tests -- In some cases, the (basic or dependent) unique cause strategy will require too many tests. For a worst case expression containing \(N\) distinct variables and complementary literals (i.e., \(A\) and \(A'\)), the strategy can require \(2^N\) tests.

Automation Required -- Effective use of unique cause strategies requires automation. At a minimum, accurate normalization of all but the simpler Boolean expressions will require a software tool. Manual application of the strategies will only be effective for simpler expressions. This is true for the effective use of any test strategy for complex expressions.

Omission Is Risky -- We know that omitting even one logical combination entails a risk that a detectable logic problem will be missed. For 6 or more variables, there are lots of distinct ways to code a wrong program. For the simple condition \((A \& B \& C)\), there are 255 wrong functions. Passing (any) 4 tests, means 15 wrong functions remain untested.

While we know that omission involves risk, we do not know the extent of this risk. Risk in using any version of Unique Cause (or any non-exhaustive strategy) depends on the likelihood of undetected defects and the failure impact of these defects.

We note that the Unique Cause strategy will not detect that an "inclusive OR" was implemented as an "exclusive OR" [i.e., \((A \text{ or } B)\) was implemented as \((AB' \text{ or } A'B')\)]. \((AB' \text{ or } A'B')\) is logically equivalent to \((A \text{ or } B)(A' \text{ or } B')\). For each of the three unique cause cases \((t f, f t, ff)\), the correct expression is anded with an expression that is TRUE during testing. In this situation, running \(tt\) would detect the difference.

All Combinations Is Not Enough -- Observe that no black box test strategy can assure detection of defective logic when the required logical expression is anded with a TRUE expression or ored with a FALSE one. For example, if the required expression is \(AB\) and the incorrect implementation is \(ABC\) where \(C\) is TRUE during testing for all truth value combinations of \(A\) and \(B\), this defect is not black box detectable. We assume that \(C\) may be FALSE in production use.

This means that unless the code is reliably (automatically) generated from the required expression, even testing with all logical combinations must be coupled with some form of code analysis technique. Some of these "stuck at" defects are detectable by noting that implemented literals do not take on both TRUE and FALSE values during execution of the unique cause tests.
9 Balance Sheet on Unique Cause

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tests provide effective examples for validating required expressions</td>
<td>Requires understanding of Boolean Algebra</td>
</tr>
<tr>
<td>Smoothly incorporates dependency information and automatically avoids “impossible” interpretations</td>
<td>Requires automated support</td>
</tr>
<tr>
<td>Provides a mechanism for verifying dependency information</td>
<td>Requires learning and experience</td>
</tr>
</tbody>
</table>

References


