International Journal Of Computer Science And Applications Vol. 1, No. 1, June 2008
ISSN 0974-1003

Detection Of Associative Shift Fault In Boolean Expression

R.K. Singh
CDAC, Noida
B-30, Institutional Area, Sector 62
Noida, UP 201 307
+91 120 3063377
rksingh@cdacnoida.in

Pravin Chandra
USIT, GGSIP University
Kashmere Gate
Delhi – 110 006
+91 11 32900318
pc_ipu@yahoo.com

Yogesh Singh
USIT, GGSIP University
Kashmere Gate
Delhi – 110 006
+91 11 32900308
ys66@rediffmail.com

ABSTRACT

The various faults that can affect a Boolean expression have been classified into various fault models. A number of techniques for testing Boolean expressions use these fault models for detecting the errors. However, the fault detection criterion for Associative Shift Fault for Boolean expression in Disjunctive Normal Form (DNF) has not been reported. In this paper we, propose a criteria for detecting Associative Shift Fault in Boolean expression in DNF.

Categories and Subject Descriptors

D.2.4 [Software Engineering]: Software/Program Verification; D.2.1 [Software Engineering]: Requirements/Specifications; D.2.5 [Software Engineering]: Testing and Debugging

General Terms

Verification

Keywords

Boolean Specification, Fault Classes, Mutation Analysis, Fault Detection Criteria.

1 INTRODUCTION

In high integrity/safety critical system, every decision has to be handled specially as any failure to do so can result in unacceptable loss. Each decision may partition the problem into two branches and each branch has to be handled separately to achieve the desired results. A fault in a branch statement of a program can lead to a fault in the program's execution behavior. Boolean expressions in Disjunctive Normal Form (DNF) have been widely used to represent a decision/predicate and a number of branch testing techniques have been reported in the literature [2,3,5,7,8,9]. Some of the most widely used branch testing techniques for testing Boolean expression are MCDC [3], RCDC [12], Basic Impact Strategy and its variants [13], BOR Strategy [10], Elmendorf Strategy [4] etc.

In general, if *n*-variables are present in a given Boolean expression, then one can construct $(2^{2^n}-1)$ expressions that are not equivalent to the given statement. If the aim of a testing mechanism/criterion is to establish the non-equivalence of all these $(2^{2^n}-1)$ expressions to the given expression, then 2^n test cases corresponding to the truth table of the given Boolean expression are required [7,13]. That is, exhaustive testing is required for testing for all non-equivalent expression. Even for moderate *n*'s the number of these test cases can become prohibitively high. Therefore, detection of difference between the intended function and all other inequivalent expression is not a feasible approach for testing of Boolean expression. A feasible approach is to develop models for "realistic" (common) faults that can occur during implementation and to restrict the attention of testing to the detection of the faults as represented by expression that differ from the intended expression due to these fault (model) based errors. In the software testing literature various fault model have been reported and are described in Section 3.

The fault detection criteria for the various faults for Boolean expression in DNF have been proposed by Kuhn [6], Tsuchiya and Kikuno [11], Lau and Yu [7]. Out of the fault models enumerated, ASF were not considered. In this paper, we propose fault detection criterion for ASF.

This paper is organized as follows: Section 2 discusses the notations and terminology. Section 3 discusses the fault models and Associative Shift Fault has been discussed in Section 4. Section 5 enumerates the fault detection criteria for the ASF and conclusions are given in Section 6.

2 NOTATIONS AND TERMINOLOGY

Boolean expression are written using Boolean variables and Boolean operators such as AND, OR and NOT are represented by '.', '+', '¯' respectively. A Boolean expression in DNF may consist of one or more product terms (*sop* term) represented as

$$S = p_1 + p_2 + \cdots + p_i + p_{i+1} + \cdots + p_j + \cdots + p_m \quad \text{or} \quad S = \sum_{i=1}^{m} p_i$$

where p_i represents one product term (*sop* term) of the Boolean expression and $1 \le i \le m$.

Published by Research Publications, Chikhli, India

Let I denotes the implementation of *S*. A *sop* term p_i may consist of one or more literal e.g. $p_i = l_1^i l_2^i \cdots l_{j-1}^i + l_j^i + l_{j+1}^i \cdots l_k^i$. The notation l_1^i is the 1st literal of the p_i^{th} *sop* term, l_j^i represents the j^{th} literal $(1 \le j \le k)$ of the p_i^{th} *sop* term where the total *number* of literal in that *sop* term are k.

The difference between Specification '*S*' and Implementation '*I*' can be represented as $S \oplus I$, where symbol $\oplus$ represent Boolean eXclusive-OR(XOR) [6]. For example, $S \oplus I_{LNF}^{l_1^i \to \bar{l}_1^i}$, computes the Boolean difference that will determine the conditions under which Literal Negation Fault in literal l_1^i will cause a failure. For simplicity, we have grouped *sop* terms except the term(s) affected by the fault as β. For example, a Boolean expression is represented as $S = p_j + \beta$ (if the fault effects the p_j^{th} *sop* term only). If a fault occurs in the literal l_i^j of the p_j^{th} *sop* term, it is rearranged such that l_i^j becomes the first literal l_1^j and α represent the remaining literals $\left(l_2^j \cdots l_k^j\right)$ of the *sop* term then $p_j = l_1^j \alpha$.

3 FAULT MODEL

The various kinds of faults [1,5,6,7,9,11,13] that can affect any expression are classified into the following categories :

- Operator Reference Fault (ORF) : In this class of fault, a binary logical operator '**.**' is replaced by '+' or vice versa.
- Expression Negation Fault (ENF) : A sub-expression in the statement is replaced by its negation .
- Term Negation Fault (TNF): A *sop* term in a expression is replaced by its negation.
- Variable Negation Fault (VNF) : An atomic Boolean literal 'a' is replaced by its negation ($\bar{a}$). This fault is also called as Literal Negation Fault (LNF).
- Associative Shift Fault (ASF): This fault occurs when an association among conditions is incorrectly implemented due to misunderstanding about operator evaluation properties.
- Missing Variable Fault (MVF) : A condition in the expression is missing with respect to original expression.
- Variable Reference Fault (VRF): A condition is replaced by another input which exist in the statement.
- Clause Conjunction Fault (CCF): A condition a in expression is replaced with $a.b$, where both inputs a and b appear in the function.
- Clause Disjunction Fault (CDF): A condition a in expression is replaced with $a+b$, where both inputs a and b appear in the function.
- Stuck at 0: A condition a is replaced with 0 in the function.
- Stuck at 1: A condition a is replaced with 1 in the function.

Faults and their brief illustration are given in Table 1.

Fault Type	**Effect of Fault on Boolean Expression** $S = (ab + cde + ghij)$
ORF$^{(+\to\bullet)}$	$(ab.cde + ghij)$
ORF$^{(\bullet\to+)}$	$(a + b + cde + ghij)$
ENF	$(\overline{ab + cde} + ghij)$
VNF	$(\bar{a}b + cde + ghij)$
TNF	$(ab + \overline{cde} + ghij)$
ASF	$a(b + cde + ghij)$
MVF	$(b + cde + ghij)$
VRF	$(cb + cde + ghij)$
CCF	$(abc + cde + ghij)$
CDF	$(ab + g + cde + ghij)$
SA0	$(cde + ghij)$
SA1	$(b + cde + ghij)$

Table 1 : Fault Class and Mutant example(s) for the Boolean Expression **:** $S = (ab + cde + ghij)$ **.**

4 ASSOCIATIVE SHIFT FAULT (ASF)

Associative Shift Fault in an expression occurs due to the incorrect placement of the parentheses. ASF changes the associativity between two or more *sop* terms resulting into the change in the output of the Boolean expression from TRUE(FALSE) to FALSE(TRUE) for some input configuration. While writing the Boolean expression in DNF form, the opening parenthesis is placed in the beginning of a *sop* term and corresponding parenthesis is placed at the end of another *sop* term. For example, $S = \begin{pmatrix} p_1 + p_2 + \cdots + p_{i-1} + l_1^i l_2^i \cdots l_j^i \cdots l_k^i + p_{i+1} + \\ + l_1^{i+2} l_2^{i+2} \cdots l_j^{i+2} \cdots l_k^{i+2} + \cdots + p_m \end{pmatrix}$. ASF may change the location of the opening parentheses, closing parentheses, or both in a Boolean expression. In expression

$$I_{ASF}^{\left(l_2^i\right.} = p_1 + p_2 + \cdots + p_{i-1} + \\ + l_1^i \left(l_2^i \cdots l_j^i \cdots l_k^i + p_{i+1} + l_1^{i+2} l_2^{i+2} \cdots l_j^{i+2} \cdots l_k^{i+2} + \cdots + p_m \right)'$$

opening parentheses is wrongly placed before literal l_2^i in term p_i. If a closing parentheses is wrongly placed before l_k^{i+2} of *sop* term p_{i+2}, it can be written as

$$I^{)l_k^{i+2}}_{ASF} = \begin{pmatrix} p_1+p_2+\cdots+p_{i-1}+l_1^i l_2^i \cdots l_j^i \cdots l_k^i + p_{1+1} + \\ + l_1^{i+2} l_2^{i+2} \cdots l_j^{i+2} \cdots \end{pmatrix} l_k^{i+2} + \cdots + p_m$$

In case a Boolean expression *S* suffers from both opening and closing parentheses fault, it can be shown as $I^{\left(l_2^i)l_k^{i+2}}_{ASF} = p_1+p_2+\cdots+p_{i-1}+l_1^i\left(l_2^i \cdots l_j^i \cdots l_k^i + p_{i+1} + l_1^{i+2} l_2^{i+2} \cdots l_j^{i+2} \cdots\right) l_k^{i+2} + \cdots + p_m$ In this case, the opening parenthesis is wrongly placed before the literal l_2^i of *sop* term p_i and the closing parentheses is placed before literal l_k^{i+2} of *sop* term p_{i+2}. For example, if $S = ab + cde + ghij$, then opening bracket fault $I^{\left(l_2^2}_{ASF} = ab + c(de + ghij)$, closing bracket fault $I^{)l_3^3}_{ASF} = (ab + cde + gh)ij$ and opening and closing bracket fault $I^{\left(l_2^2)l_3^3}_{ASF} = ab + c(de + gh)ij$.

5 FAULT DETECTION CRITERIA FOR ASF

Associative Shift Fault in an expression occurs due to the incorrect placement of the opening bracket, closing bracket or both, in a Boolean expression resulting in a change of the associativity amongst the *sop* terms [5,9].

5.1 Opening Bracket Fault

If opening bracket fault occurs in the p_j^{th} *sop* term, it can be written as $I^{\alpha_x(}_{ASF} = \alpha_x(\alpha_y + \tau) + \beta$ where $p_j = \alpha_x\alpha_y$ with the opening fault partitioning the p_j^{th} *sop* term while τ is a sub-expression of the Boolean expression such that $|\tau| \geq 1$ (number of terms present in τ is greater than or equal to 1) and β is the sum of the rest of the term in the expression i.e. $S = p_j + \tau + \beta$. The fault detection criteria for the opening bracket fault can be represented as $S \oplus I^{\alpha_x\alpha_y \to \alpha_x(\alpha_y}_{ASF} = (\alpha_x\alpha_y + \tau) + \beta \oplus \alpha_x(\alpha_y + \tau) + \beta$. Therefore, the fault detection criteria for the opening bracket fault can be written as:

$$S \oplus I^{\alpha_x\alpha_y \to \alpha_x(\alpha_y}_{ASF} = \overline{\alpha}_x \tau \overline{\beta} \qquad (1)$$

5.2 Closing Bracket Fault

If closing bracket fault occurs at α_y partition then it can be represented as $S \oplus I^{\alpha_y)\alpha_x}_{ASF} = (\tau + \alpha_y)\alpha_x + \beta$ and this can be written as $\alpha_x(\alpha_y + \tau) + \beta$. That is, the closing bracket fault is of the same form as the opening bracket fault. Hence, the fault detection criteria in this case would be the same as opening bracket fault. That is, if τ and the splitting of the term $\alpha_x\alpha_y$ due to bracket error in the two cases is same, their detection criteria is also the same.

5.3 Opening and Closing Bracket Fault

In this case, the opening and closing bracket fault occur simultaneously in the expression. Then it can easily be seen that the opening bracket fault detection criterion will also detect this type of fault provided the *sop* term being partitioned by the closing bracket is considered as a part of τ. Thus, eq. (1) is the detection criterion for all types of ASF. If we choose one p_k *sop* term in sub-expression τ to be TRUE and the rest of the term in τ to evaluate to FALSE then also the criteria is met. That is, we may write the detection criteria as:

$$= \overline{\alpha}_x\, p_k\, \overline{\hat{\beta}} \qquad (2)$$

where p_k is a term in τ, by choice of p_k, $\hat{\beta} = \sum_{\substack{l=1 \\ l \neq j,k}} p_l$ and each p_l is FALSE for eq. (2) to evaluate to TRUE and p_j evaluates to FALSE in eq. (2). It should be noted that α_x must be FALSE that is even p_j evaluates to FALSE.

6 CONCLUSIONS

In this paper we have presented the fault detection criteria for the Associative Shift Fault that is considered to be the strongest fault in the fault models. The proposed fault detection criteria guarantee the detection of the ASF fault.

7 REFERENCES

[01] Chen T.Y and M.F Lau. 2001. Test Case Selection Strategies based on Boolean Specifications. Journal of Software Testing, Verification and Reliability. 11: 165-180.

[02] Chen, T.Y, M.F Lau, and Y.T. Yu. 1999. MUMCUT: A Fault-based Strategy for Testing Boolean specification, 6th Asia Pacific Software Engineering Conference (APSEC'99): 606-613.

[03] Chilenski J.J. and S. Miller. 1994. Applicability of Modified Condition/ Decision Coverage to Software Testing. Software Engineering Journal, 9(5):193-200.

[04] Elmendorf W. R. 1973. Cause-Effect Graphs on Functional Testing. TR-00.2487, IBM Systems Development Division, Poughkeepsie, NY.

[05] Kapoor K. 2004. Stability of Test Criteria and Fault Hierarchies in Software Testing. PhD Thesis, London South Bank University.

[06] Kuhn D. Richard. 1999. Fault Classes and Error Detection Capability of Specification-based Testing. ACM Transactions on Software Engineering and Methodology 8, 411-424.

[07] Lau M.F. and Y.T Yu. 2005. An Extended Fault Class Hierarchy for Specification-Based Testing. ACM Transactions on software Engineering and Methodology 14 No. 3 : 247-276

[08] Lau M.F and Y.T. Yu. 2001. On the Relationships of Faults for Boolean Specification Based Testing. Proceedings of Software Engineering Conference : 21-28

[09] Singh R.K., Pravin Chandra, Yogesh Singh. 2006. An Evaluation of Boolean Expression Testing Techniques. ACM Software Engineering Notes 31 No. 5:1-6

[10] Tai K.C, M.A. Vouk, A. Paradkar, and P Lu. 1994. Evaluation of a predicate-based software testing strategy. IBM Systems Journal 33, No.3: 445-457.

[11] Tsuchiya T. and Kikuno T. 2002. On Fault Classes and Error Detection Capability of Specification–Based Testing, Transactions on Software Engineering and Methodology 11, 58-62.

[12] Vilkomir S. A. and J. P. Bowen. 2002. Reinforced Condition/Decision Coverage (RC/DC): A New Criterion for Software Testing. 2nd International Conference, Formal Specification and Development in Z and B, Springer-Verlag Lecture Notes in Computer Science 2272 : 295–313.

[13] Weyuker E.J, T. Gorodia., and A. Singh. 1994. Automatically Generating Test Data from a Boolean Specification. IEEE Transactions on Software Engineering 20 No. 5:353–363.