Analyzing and Extending MUMCUT for Fault-based Testing of General Boolean Expressions

Chang-ai Sun[1], Yunwei Dong[2], R. Lai[3], K.Y. Sim[4], T.Y. Chen[5]

[1,2,5] *Faculty of Information and Communication Technologies, Swinburne University of Technology, Victoria 3122, Australia*
[3] *Department of Computer Science and Computer Engineering, La Trobe University, Victoria 3086, Australia*
[4] *School of Engineering, Swinburne University of Technology, Kuching, Sarawak, Malaysia*

Abstract

Boolean expressions are widely used to model decisions or conditions of a specification or source program. The MUMCUT, which is designed to detect seven common faults where Boolean expressions under test are assumed to be in Irredundant Disjunctive Normal Form (IDNF), is an efficient fault-based test case selection strategy in terms of the fault-detection capacity and the size of selected test suite. Following up our previous work that reported the fault-detection capacity of the MUMCUT when it is applied to general form Boolean expressions, in this paper we present the characteristic of the types of single faults committed in general Boolean expressions that a MUMCUT test suite fails to detect, analyze the certainty why a MUMCUT test suite fails to detect these types of undetected faults, and provide some extensions to enhance the detection capacity of the MUMCUT for these types of undetected faults.

1. Introduction

Boolean expressions are widely used to model or specify decisions and conditions in a specification or source program. When a decision or condition in a specification or program source is incorrectly implemented, its execution may result in a faulty output. Given a Boolean expression in some restricted form, such as *Irredundant Disjunctive Normal Form (IDNF)* [1][10], it is possible to enumerate some typical faults that may be introduced during the programming process.

Several fault-based testing strategies have been developed to detect some typical faults of Boolean expressions in some restricted form [1] [9] [10]. For example, Tai et al.[9] proposed a Boolean operator (BOR) test case generation strategy for *singular* Boolean expressions where each variable can occur once. Weyuker et al. [10] proposed a family of meaningful impact strategies (MI) for testing Boolean expressions in *IDNF*. Chen et al.[1] proposed a set of more efficient test case generation strategies (MUMCUT) for Boolean expressions in *IDNF*. All these fault-based strategies only select a subset of an exhaustive test set and in the meantime have a high fault-detection capability, thus they can save the cost of software testing when they are employed in practice. For example, the MIN strategy uses on average 5.8% of the size of an exhaustive test set while achieves from a low average mutation score of 97.9 to a high average mutation score of 99.7 [10]. The experiments show that compared with the MIN strategy, MUMCUT may achieve higher fault detection capability while using smaller size of test cases [11].

However, Boolean expressions in a realistic program or specification are often not in some restricted forms, say IDNF or DNF. This means that during a designer or programmer implements Boolean expressions, faults are introduced in the context of general form. A single fault in a general form Boolean expression may give rise to more than one fault in the corresponding equivalent restricted form. Since both the meaningful impact strategies [10] and the MUMCUT [1] are evaluated using relevant "simple faults" operator to obtain the mutation of Boolean expressions under test, it is interesting to evaluate or enhance IDNF-oriented fault-based testing strategies to detect faults in general form Boolean expressions.

We have reported in our previous experiments that the MUMCUT did very well when it was applied to general Boolean expressions [2] [8]. In this paper, we present the characteristic of undetected faults and their certainty of being detected by the MUMCUT, and

Proceedings of The Sixth IEEE International Conference on Computer and Information Technology (CIT'06)
0-7695-2687-X/06 $20.00 © 2006 IEEE

provide some extensions to the MUMCUT to enhance its fault detection capacity for general form Boolean expressions.

2. Notation and Terminology

In this section, we introduce some basic concepts and notation related to the MUMCUT and the empirical study later in the paper.

2.1 Boolean expressions and their test data

Boolean values will be written as *T* and *F* or *1* and *0*. Boolean expressions are built from Boolean values, variables, the unary operator NOT, and binary operators AND, OR, XOR, = and !=. The parentheses are used to change the precedence of operators or association of Boolean variables. Given a Boolean expression, it may be represented in several forms. A Boolean expression in Disjunctive Normal Form (DNF) is said to be in Irredundant DNF (IDNF) when none of the Boolean's literals or terms can be deleted without altering the Boolean expression's value for some inputs [1] [10].

For a Boolean expression with m Boolean variables, a *m*-dimensional Boolean vector where each Boolean variable is assigned to a Boolean value is called a test case *t*. Furthermore, for a Boolean expression *B* in IDNF that can be written as

$$B = T_1 + T_2 + ... + T_n$$

where T_i (i=1,…,n) is a term of B, then we can classify the input domain into the following disjoint set of points:

TP (True Point) ={t | *B*(t)=T} ,

FP (False Point) ={t | *B*(t)=F},

UTP (Unique True Point) = $\bigcup_{i=1}^{n} TP_i$,

$$TP_i = \{t \mid T_i(t) = T \wedge (\forall j(j \neq i \wedge T_j(t) = F)\} ,$$

OTP (Overlapping True Point)= TP\UTP,

NFP (Near False Point)= $\bigcup_{i=1}^{n}\bigcup_{j=1}^{m} FP_{i,j}$, where

$$FP_{i,j} = \{t \mid t \in FP_i \wedge FP_i(t_{i,j} / \overline{t_{i,j}})(t) = T\} , \text{ and}$$

$$FP_i = \{t \mid T_i(t) = F\} ,$$

RFP (Remain False Point) = FP\NFP.

2.2 Fault types

Various faults related to Boolean expressions have been reported in literature [4][5][6][9][10]. The terminologies within this paper will follow those discussed in [2][6]. In this study, we investigate ten types of faults in general form Boolean expressions, including Expression Negation Fault (*ENF*), Term Negation Fault (*TNF*), Term Omission Fault (*TOF*), Literal Negation Fault *(LNF)*, Literal Omission Fault (*LOF*), Literal Insert Fault (*LIF*), Literal Reference Fault (*LRF*), Operator Reference Fault (*ORF*), Parenthesis Omission Fault (*POF*) and Parenthesis Insertion Fault (*PIF*).

2.3 MUMCUT

Fault-based testing in nature is intended to detect some special types of faults using a subset of the exhaustive test set. A fault-based test case generation strategy focuses on where and how to select the meaningful points. The MUMCUT [1] is designed to detect eight types of single fault in Boolean expressions that are assumed to be in IDNF. The MUMCUT is the integration of MUTP, CUTPNFP and NFP. The MUMCUT only selects some UTP and NFP to form a test suite. The MUMCUT can greatly reduce the size of the resulting test suite because UTP and NFP are a part of an exhaustive set, and in the meantime a MUMCUT test suite can guarantee to detect single faults including *ENF*, *TNF*, *TOF*, *LNF*, *LOF*, *LIF*, *LRF* and *ORF*.

2.4 Mutation

The mutation analysis [3] is widely used to verify the adequacy of a test suite based on some specific testing criteria. Given a Boolean expression *B*, a derivation *M* is obtained by seeding faults into *B*. *M* is called a *mutant* of *B*, and the process to obtain *M* from *B* is called *mutation*. In this study, the mutation technique is used to derive mutants of general form Boolean expressions. The ten types of faults discussed in Section 2.2 have been simulated by the mutation technique. Note that in our study a derived mutant contains only one single syntactic change when it is compared with the original Boolean expression.

3. Patterns of the Undetected Faults

In this section, we analyze the characteristics of faults/mutations that will escape from being detected by a MUMCUT test suite.

3.1 Empirical studies

When applying the MUMCUT to a general form Boolean expression *GB*, we need to transform *GB* into a Boolean expression *IB* in IDNF, then the MUMCUT

Proceedings of The Sixth IEEE International Conference on Computer and Information Technology (CIT'06)
0-7695-2687-X/06 $20.00 © 2006 IEEE

is used to generate test suite from *IB* or its mutant *IM* that is transformed from a mutant *GM* of *GB*. In this paper, we refer to these two applications of MUMCUT as *forward MUMCUT* and *reverse MUMCUT*, which basically simulate the situations where test cases are generated from a specification (that is, *IB*) or source program (that is, *IM*), respectively.

Our experiments have reported that when the MUMCUT is applied to ten types of fault in general form Boolean expressions, the forward MUMCUT has a mutation score of 99.4, while the reverse MUMCUT has a mutation score of 97.9 [2]. There are totally 22 undetected mutants for the forward MUMCUT and 86 for the reverse MUMCUT, respectively. We have reported the five kinds of patterns which describe the characteristics of 55 undetected mutants in the reverse MUMCUT [8]. We examine and summarize below the characteristic of 108 undetected mutants, which will be used as the basis of certainty analysis of detection failure and extension to the MUMCUT.

3.2 Patterns of undetected mutations

For briefness and simplicity, we use the concept of pattern to abstract the characteristics of an undetected mutant. An undetected mutation pattern describes the causes that result in the failure of detection by the MUMCUT. An undetected mutation pattern is also a function that defines the substantial difference between the source and the target. Under the circumstances of mutation, the source of an undetected mutation pattern corresponds to an original Boolean expression in IDNF, while the target corresponds to its mutants in IDNF. The forward MUMCUT generates test cases from the source, while the reverse MUMCUT from the target.

For all undetected mutants in the forward MUMCUT, we discover one common characteristic by comparing the source and the target, namely, the insertion of terms consisting of more than one literal. Furthermore, for a large part of the undetected mutation patterns, some Boolean variables occur at the target while disappear at the source. Similarly, among the undetected mutation patterns in the reverse MUMCUT, one common characteristic is the omission of terms consisting of more than one literal. For a large part of them, some Boolean variables occur at the source while disappear at the target.

We summarize several undetected mutation patterns by examining all undetected mutants from both the forward MUMCUT and the reverse MUMCUT. We only outline these patterns in terms of the forward MUMCUT. It is easy to derive their representations for the reverse MUMCUT due to symmetry. For example, a pattern "p:a→b" in the forward MUMCUT will be represented as "p:b→a" in the reverse MUMCUT; a test suite generated from *a* will not detect *b* in the forward MUMCUT; correspondingly, a test suite generated from *a* will not detect *b* in the reverse MUMCUT. To address more compact and substantial characteristics, we further extract core patterns and their extensions. It is noted that the occurrence of both core patterns and extension patterns will result in the failure of detection by the MUMCUT.

Pattern 1 Core Pattern: $abc \rightarrow abc + \overline{ab}$, and
Extension Patterns:
$abcS \rightarrow abcS + \overline{ab}$;
$abcS \rightarrow (abc + \overline{ab})S$. Here, S is a null term or a term without the occurrence of *a*, *b* or *c*.

Pattern 2 Core Pattern:
$ab + c \rightarrow ab + bc + ac = ab + c(a + b)$ and
Extension Patterns:
$ab + cS \rightarrow ab + c(a + b)S$;
$(ab + c)S \rightarrow (ab + c(a + b))S$. Here, S is a null term or a term without the occurrence of *a*, *b* or *c*.

Pattern 3 Core Pattern: $acde + bc \rightarrow acde + bc + ab\overline{d}$ and Extension Patterns:
$acdeS + bc \rightarrow acdeS + bc + ab\overline{d}$;
$acde + bc \rightarrow acde + bc + ab\overline{d}S$;
$(acde + bc)S \rightarrow (acde + bc + ab\overline{d})S$. Here, S is a null term or a term without the occurrence of *a*, *b*, *c*, *d* and *e*.

Pattern 4 Core Pattern: $ab + ac \rightarrow ab\overline{c} + a\overline{b}c + \overline{a}bc$ and
Extension Pattern:
$(ab + ac)S \rightarrow (ab\overline{c} + a\overline{b}c + \overline{a}bc)S$. Here, S is a null term or a term without the occurrence of *a*, *b* or *c*.

Pattern 5 Core Pattern: $ab \rightarrow ab + bc$ and
Extension Patterns:
$abS \rightarrow abS + bc$;
$ab \rightarrow ab + bcS$;
$abS \rightarrow (ab + bc)S$. Here, S is a null term or a term without the occurrence of *a*, *b* or *c*.

3.3 Undetected mutation patterns

Tables 1 and 2 report the distribution of undetected faults in the forward MUMCUT and the reverse MUMCUT, respectively.

Proceedings of The Sixth IEEE International Conference on Computer and Information Technology (CIT'06)
0-7695-2687-X/06 $20.00 © 2006 IEEE

Table 1. The distribution of undetected mutation patterns in the forward MUMCUT

Mutation Type	P1	P2	P3	P4	P5	Total
ENF	0	0	0	0	0	0
LNF	0	2	0	0	5	7
TOF	0	0	0	1	0	1
TNF	2	0	0	0	0	2
LOF	1	0	0	0	1	2
LIF	0	0	0	0	1	1
LRF	0	1	0	0	2	3
ORF	0	1	0	0	0	1
POF	0	0	0	0	0	0
PIF	1	0	0	0	4	5
Total (%)	4 (18.2)	4 (18.2)	0 (0)	1 (4.5)	13 (59.1)	22 (100)

Table 2. The distribution of undetected mutation patterns in the reverse MUMCUT

Mutation Type	P1	P2	P3	P4	P5	Total
ENF	0	0	0	0	2	2
LNF	0	2	0	1	3	6
TOF	3	4	3	0	15	25
TNF	0	0	0	0	5	5
LOF	0	1	0	0	1	2
LIF	2	3	0	0	1	6
LRF	1	2	0	0	1	4
ORF	2	1	1	0	4	8
POF	0	0	0	0	0	0
PIF	3	2	0	2	21	28
Total (%)	11 (12.8)	15 (17.4)	4 (4.7)	3 (3.5)	53 (61.6)	8 6 (100)

From Tables 1 and 2, single faults from general form Boolean expressions may result in the occurrence of various undetected mutation patterns. On the other hand, our experimental results demonstrate the omission of some undetected mutation patterns for some mutation types. For example, no undetected LRF mutation in the forward MUMCUT belongs to Pattern 1, and no undetected LOF mutation in the reverse MUMCUT belongs to Pattern 1. Intuitively speaking, for each type of faults in general form Boolean expressions, there should be a possibility of occurrence of these undetected mutation patterns.

From Tables 1 and 2, we discover that in both the forward MUMCUT and the reverse MUMCUT, nearly 60% of the undetected mutants fall in Pattern 5; only a very small percentage of the undetected mutants fall in Patterns 3 and 4. The distribution of undetected mutation pattern in our experiments provides us information about on which aspects we should put emphasis to analyze and enhance the MUMCUT.

4.Certainty Analysis of Detection Failures

We analyze below the reason why those faults or mutations that satisfy the characteristics of one of five patterns or their combination can escape from being detected by a MUMCUT test suite. Further more, we will concentrate our analysis on the core patterns of undetected mutations individually. In the following, we use a_t to represent the TRUE assignment of Boolean variable a, and a_f for FALSE assignment.

There are 15 undetected mutations of *Patten 1* in our experiments. When the MUMCUT is used to generate test cases from abc, the resulting test suite is $\{\ a_tb_tc_t, a_fb_tc_t, a_tb_fc_t, a_tb_tc_f\ \}$. The outputs of $abc+\overline{ab}$ and abc are always the same with test cases from the resulting test suite. If one test point $a_fb_fc_t$ is appended to the test suite, the mutant can be killed. The point $a_fb_fc_t$ however is included in neither UTP nor NFP, so this is a fault that cannot be detected by a MUMCUT test suite.

There are 19 undetected mutations of Patten 2 in our experiments. Similarly, a MUMCUT test suite $\{\ a_tb_tc_f\ ,\ a_tb_fc_t\ ,\ a_fb_tc_t\ ,\ a_fb_tc_f\ ,\ a_tb_fc_f\ \}$ is generated from $ab+c$. When an extra point $a_fb_fc_t$ is appended, the mutant can be killed. This extra point $a_fb_fc_t$ is included in UTP, while not in the MUMCUT test suite. We further discover that it is a fault that may or may not be detected by a MUMCUT test suite since the selection of point in the MUMCUT is nondeterministic.

There are 4 undetected mutations of *Pattern 3* in our experiments. A MUMCUT test suite generated from $acde+bc$ does not include points $a_tb_tc_fd_fe_f$ and $a_tb_tc_fd_fe_t$. If either of these two points is appended, the mutant can be killed. Since both $a_tb_tc_fd_fe_f$ and $a_tb_tc_fd_fe_t$ are included in neither UTP nor NFP, therefore it is a certain fault that cannot be detected by a MUMCUT test suite.

There are 4 undetected mutants of *Pattern 4* in our experiments. A MUMCUT test suite $\{a_tb_tc_f, a_tb_fc_t, a_fb_tc_f, a_tb_fc_f,\ a_fb_fc_t, a_fb_tc_t\}$ is generated from $ab+ac$. Only when the test point $a_tb_tc_t$ is appended, can the mutant be killed. Since $a_tb_tc_t$ is included in neither UTP nor NFP, it is also a certain fault that cannot be detected by a MUMCUT test suite.

There are 66 undetected mutations of *Pattern 5* in our experiments. A MUMCUT test suite $\{\ a_tb_t\ ,\ a_tb_f\ ,\ a_fb_t\ \}$ is generated from ab. With

Proceedings of The Sixth IEEE International Conference on Computer and Information Technology (CIT'06)
0-7695-2687-X/06 $20.00 © 2006 IEEE

$a_t b_t$ and $a_t b_f$ as inputs, two Boolean expressions produce the same outputs. If $a_f b_t$ is used as an input, the output of $ab + bc$ depends on the value of c and the output of ab is False. Therefore, this fault is also uncertain for the MUMCUT strategy. Unlike Pattern 2, this undetected mutation pattern is beyond the design of the MUMCUT because the relevant Boolean variables do not exist in the mutants. However, we notice that this type of faults can result from one single fault when general form Boolean expressions are concerned. For example, we assume that an LRF (replacing d with b) happens to the original Boolean expression $(a+b)(c+d)+bd$, a mutant $(a+b)(c+b)+bd$ is obtained. When the mutant is transformed to one in IDNF, namely $ac + b$, Boolean variable d has disappeared. Obviously, when the reverse MUMCUT is applied, the detection of this fault is similar to the core pattern discussed here.

5. Extensions to MUMCUT

In this section, we discuss how to extend a MUMCUT test suite to detect those undetected mutations reported in our experiments. We will firstly propose the extension of test suite for undetected mutation pattern individually, and then proposed a more general enhancement to the MUMCUT.

5.1 Specific extensions for undetected mutation patterns

The characteristic of *Pattern 1* lies in the omission of one term consisting of two or more complementary literals. Recall all existing IDNF-oriented fault-based testing strategies divide the input space into four disjoint sets, and NFP is designed to demonstrate the effect of each literal in one term on the output. We can further generalize the NFP to n-NFP, thus the False Point Set is further classified into 1-NFP, 2-NFP, …, n-NFP and RFP. 1-NFP here is equal to NFP mentioned in Section 2.1. As an illustration, we define 2-NFP as follows.

$$\text{2-NFP} = \bigcup_{i=1}^{n} \bigcup_{j_1, j_2} FP_{i,j_1 j_2}, \text{ where}$$

$$FP_{i,j_1 j_2} = \{t \mid t \in FP_i \wedge FP_i(t_{i,j_1} / \overline{t_{i,j_1}}, t_{i,j_2} / \overline{t_{i,j_2}})(t) = T \wedge (|T_i| > 1 \wedge j_1 \neq j_2)\}.$$

$FP_i = \{t \mid T_i(t) = F\}$.

$FP_i(t_{i,j_1} / \overline{t_{i,j_1}}, t_{i,j_2} / \overline{t_{i,j_2}})$ is one Boolean expression in IDNF B' derived from the original Boolean expression in IDNF B through the following replacements: the $j_1{}^{th}$ literal in the i^{th} term in B is replaced with its complementary, and the $j_2{}^{th}$ literal in the i^{th} term in B is replaced with its complementary.

Based on the n-NFP (for example, 2-NFP), we extend MNFP in MUMCUT as follows: "Given each term T_i and $FP_{i,j}$, $FP_{i,j_1 j_2}$, the selected test points should cover (1) all possible truth-values of variables not occurring in $FP_{i,j}$, and (2) all possible combination of any two variables occurring in $FP_{i,j_1 j_2}$". Still take "$abc \rightarrow abc + \overline{ab}$" as an example, test point $a_f b_f c_t$ is included in the test suite of the extended strategy, and this fault can be detected by it.

Pattern 2 is an uncertain undetected mutation pattern for MUMCUT. The exact reason that a MUMCUT test suite fails to detect such a fault pattern lies in the MUMCUT's nondeterministic selection of UTP in MUTP. To detect this uncertain fault, a feasible method is to increase the number of UTP samples.

Pattern 5 cannot be detected by a MUMCUT test suite because new Boolean variables are introduced. This problem is outstanding when we apply MUMCUT, which is designed to detect faults in Boolean expressions in IDNF, to detect the faults in general form Boolean expressions. To effectively detect this type of fault, a test suite TS_o generated from an original Boolean expression should be extended to contain the assignments to new Boolean variables. One simplest and safest extension is described as " for new Boolean variables, an exhaustive test set TS_n is generated. The ultimate test suite is the combinational concatenation of TS_n and TS_o ". For example, a MUMCUT test suite for ab is $\{a_t b_t, a_t b_f, a_f b_t\}$, and the exhaustive test points for c are $\{c_f, c_t\}$, then an extension test suite is $\{a_t b_t c_t, a_t b_f c_t, a_f b_t c_t, a_t b_t c_f, a_t b_f c_f, a_f b_t c_f\}$. Obviously, the test point $a_f b_t c_t$ can be used to kill this fault.

Pattern 3 cannot be detected because of the omission of terms that consists of multiple coupling literals occurring in other terms. *Pattern 4* cannot be detected because of the coupling changes in multiple terms. Both *Pattern 3* and *Pattern 4* are certain undetected faults for MUMCUT. Up to now, we cannot figure out a specific extension to *Patterns* 3 and 4. Fortunately, there are only 8 such types of undetected mutants from totally 4122 mutants in our experiments.

Proceedings of The Sixth IEEE International Conference on Computer and Information Technology (CIT'06)
0-7695-2687-X/06 $20.00 © 2006 IEEE

5.2 A general extension to MUMCUT

We describe here a unified extension strategy for test case generation for all undetected mutation patterns. Given a Boolean specification ***B***, we employ the MUMCUT to generate a test set ***TS*** from the equivalent ***BI*** in IDNF. To increase the fault detection capacity (***BI*** may contain one of the five undetected mutation patterns or their combinations), we can extend the test set ***TS*** using the following procedure:

1) Append a full-false point and a full-true point to ***TS***; and
2) If $x > 2$, then append x-2 test cases from the remaining test cases to ***TS*** $\{ts_1, \ldots, ts_m\}$, where the number of an exhaustive test set for ***BI*** is 2^x (x is the number of Boolean variables in ***BI***). When one candidate t_i is chosen from the remaining test cases $\{t_1, \ldots, t_n\}$, it should satisfy the condition $\underset{j=1..m}{Max}(|t_i - ts_j|) \geq \underset{k=1..n, k\neq i}{Max}(\underset{j=1..m}{Max}(|t_k - ts_j|))$, where $|t_i - t_j|$ is the number of different bits between test cases t_i and t_j.

The intuition behind our extension is that those test points that are largely different from the selected ones should be selected for execution with a bigger chance to detect faults. As a very preliminary study, we experimented this extension to the current MUMCUT for five core patterns of undetected mutations. Experimental results show that the extended MUMCUT test suite may detect all faults which satisfy one of the five core patterns of undetected mutations. We illustrate this extension by *Pattern 1*. We apply the forward MUMCUT to generate a test suite $\{a_t b_t c_t, a_f b_t c_t, a_t b_f c_t, a_t b_t c_f\}$ from abc, and the extended MUMCUT test suite is $\{\underline{a_t b_t c_t}, a_f b_t c_t, a_t b_f c_t, a_t b_t c_f, \underline{a_f b_f c_f}, \underline{a_f b_f c_t}\}$. The latter can detect the fault while the former cannot. It is noted that for *Pattern 5*, we need to combine this general extension strategy with specific extensions discussed as above.

6. Conclusion

We have studied the extension of the MUMCUT, which is originally designed to detect several common single faults in Boolean expressions that are assumed to be in *IDNF*, for one single fault in general form Boolean expressions. We have conducted a family of experiments to examine the issues arising when the MUMCUT is applied to test general form Boolean expressions. We have identified five undetected mutation patterns by comparing all undetected mutants in the forward/reverse MUMCUT experiments. We further analyze why a MUMCUT test suite fails to detect those faults that satisfy these undetected mutation patterns, and propose specific and general extensions to the current MUMCUT to detect those undetected faults.

In future work, we need to conduct the empirical evaluation on the extension to MUMCUT discussed in this paper using other Boolean expression samples.

7. Acknowledgements

This research is partially supported by the ARC Discovery Grant of Project No. DP0345147.

8. References

[1] T.Y. Chen, M.F. Lau. Test case selection strategies based on Boolean Specification, Software Testing, Verification and Reliability, Vol. 11,No.3, 2001, pp165-180

[2] T.Y. Chen, M.F. Lau, C.A. Sun, K.Y. Sim, On detecting faults for Boolean Expressions, submitted for publication, 2006

[3] R.A. DeMillo, R.J. Lipton, F.G. Sayward, Hints on test data selection: Help for the practicing programmer, Computer, Vol. 11, No.4, 1978, pp34-41

[4] K. Foster, Sensitive test data for logical expressions, SIGSOFT Software Engineering Notes, Vol. 9, No.2, 1984, pp120-125

[5] K. R. Kuhn, Fault Classes and Error Detection Capability of Specification-based Testing, ACM Transactions on Software Engineering and Methodology, vol.8, No.4, October 1999 pp411-424

[6] M.F. Lau and Y. T. Yu, An Extended Fault Class Hierarchy for Specification-Based Testing, ACM Transaction on Software Engineering and Methodology, Vol. 14, No.3, 2005, pp247-276

[7] C.A. Sun and K.Y. Sim, An FSM-based parameterized generator for Boolean expressions, Proceedings of ICENCO-2004, Dec. 27-30 2004, Cairo, Egypt, pp119-126

[8] C.A. Sun and K.Y. Sim, T.H. Tse, T.Y. Chen, An empirical evaluation and analysis of the fault-detection capability of MUMCUT for general Boolean expressions, Proceedings of International Computer Symposium 2004, Dec.14-17, Taipei, Taiwan, pp926-932

[9] K.C. Tai, M.A.Vouk, A. Paradkar, P.Lu, Evaluation of a Predicate –Based Software Testing Strategy. IBM System Journal, Vol.33, no.3, Oct. 1994, pp445-457

[10] E. Weyuker, T. Goradia, A. Singh, Automatically generating test data from a Boolean specification, IEEE Transactions on Software Engineering, Vol. 20, No.5, 1994 pp353-363

[11]Y.T. Yu, M.F. Lau, and T.Y. Chen. Automatic generation of test cases from Boolean specifications using the MUMCUT strategy, Journal of Systems and Software, Vol. 79, No.6, 2006, pp820-840

Proceedings of The Sixth IEEE International Conference on Computer and Information Technology (CIT'06)
0-7695-2687-X/06 $20.00 © 2006 IEEE