IJCSE International Journal of Computer Sciences and Engineering Open Access
Survey Paper Vol-6, Special Issue-8, Oct 2018 E-ISSN: 2347-2693

A Brief Survey Onboolean Expressions in Fault Based Techniques

[1*]M. Sivaranjani, [2]D. Gayathri Devi

[1,2]Dept. of Computer Science Sri Ramakrishna College of Arts and Science for Women Coimbatore, India

Available online at: www.ijcseonline.org

Abstract- Boolean expressions are major focus of specifications and they are very much prone to introduction of faults, this survey presents various fault based testing techniques.It recognizes that the methods differ in their fault detection capabilities and creation of test suite. The various techniques like Dealing with Constraints in Boolean Expression, Minimal Fault Detecting Test Suites, Reducing logic test set size, A logic mutation approach, SAT and SMT Solvers for Test Generation and Boolean Expressions by Cell Covering has been considered. This survey describes the fundamental algorithms and fault categories used by these strategies for evaluating their performance. Finally, it contains short summaries of the papers that use Boolean expressions used to specify the requirements for detecting faults. These techniques have been empirically evaluated by various researchers on a simplified safety related real time conditionals system.

Keywords-Boolean Expression, BOR, Test suite, MBT.

I. INTRODUCTION

Software dimension and complexity is increasing that has made software testing a challenging exercise. The objective of testing is to determine error, which requires dynamic execution of test cases that consumes significant amount of time so it is important to investigate ways of increasing the efficiency and effectiveness of test cases.

Test case designing is one of the important factors that influence cost and coverage of testing. The cost depends on size of test suit and coverage on fault detection capabilities. Much research has been aimed at achieving high efficacy and reduced cost of testing by selecting appropriate test cases. Boolean expressions can be used to specify the requirements of safety-critical software like avionics, medical and other control software. These expressions can describe certain conditions of specifications, to model predicates and logical expressions. Test cases are generated on Boolean expressions which are capable of revealing faults in programs that are developed based on such specifications.

Many testing techniques have been proposed by various researchers to select test cases based on Boolean specifications; moreover test case generated by these methodologies can guarantee to detect certain type of faults.

Boolean expressions, i.e. terms that evaluate to true or false, are frequently found in logical predicates inside programs to model complex conditions under which some code is executed. They are commonly used as guards for conditional instructions and cycles. Also in model based testing, Boolean conditions play a very important role because they can be found as guards of transitions and actions. They constitute a critical part also because many typical programmer and designer errors result in faults in Boolean expressions.

In Boolean expression, Boolean Operator testing Strategy (BOR) is a technique suitable for test generation for singular Boolean expression. It guarantees the detection of Boolean operator faults, including incorrect AND/OR operators and missing or extra Not operators.[1-3] showed that a BOR test set for a Boolean expression is effective in detecting various types of Boolean expression faults, including Boolean operator faults, incorrect Boolean variables and parentheses and their combinations.

Infeasible test requirements are demands for tests that simply do not exist. They are an unfortunate fact of life in software testing. They confound test engineers, who must decide if a given test requirement really is infeasible or if a more diligent search for a suitable input is in order. They also confound attempts by researchers to relate coverage criteria. By definition, an infeasible test requirement for a given criterion does not result in a test. If the corresponding test requirement for a 'weaker' criterion happens to be feasible, the infeasibility can cause an apparently 'stronger' criterion to fail to subsume the 'weaker' one. Many well-known cases of this phenomenon pervade the testing literature.

This paper explores strategies for Test Case Generation for Boolean Expressions by Cell Covering models. Boolean expressions are found in logical predicates inside programs and specifications which model complex conditions. This paper, various approaches has been surveyed in which test cases are generated from Boolean expressions that target specific fault classes and test suites is reduced with respect to exhaustive testing. In this article, it is assumed that readers are familiar with notations and terminologies of Boolean expressions.

II. TYPES OF FAULTY DETECTION

There are various types of faulty techniques namely,

1. Cause effect graph
2. Branch Operator Strategy (BOR)
3. BOR+MI
4. MUMCUT
5. Modified Condition/Decision Coverage (MCDC)

- **Cause effect graph:** It focuses on modelling dependency relationships among program input conditions known as causes, and output conditions known as, effects.
  - ❖ **Advantages:**It helps us to determine the root causes of a problem or quality using a structured approach.
  - ❖ **Disadvantages:**The process of creating decision table is inconsistent and ambiguous.
- **BOR:** It guarantees the detection of Boolean operator faults, including incorrect AND/OR operators and missing or extra Not operators.
  - ❖ **Advantages:**It'sincluding Boolean operator faults, incorrect Boolean variables and parentheses and their combinations.
  - ❖ **Disadvantages:**BOR strategy is not suitable for non singular expressions.
- **BOR+MI:** Meaningful Impact (MI) testing strategycombines BOR.This is hybrid algorithm partitions an input Boolean expression in to components such that BOR strategy can be applied to some and MI strategy to remaining components.
  - ❖ **Advantages:**The test constraints for individual components are combined using BOR strategy.
  - ❖ **Disadvantages:**It produces a smaller test constraint set for Boolean expression.
- **MUMCUT:** Itintegrates the Multiple Unique True Point (MUTP), Multiple Near False Point (MNFP) and Corresponding Unique True Point and Near False Point Pair (CUTPNFP).
  - ❖ **Advantages:**It guarantees the detection of certain faults in logical decisions in disjunctive normal.
  - ❖ **Disadvantages:**It may still miss some faults that can almost always be detected by test sets.
- **MCDC:**The pair for a condition is one that changes the output on varying the input from "*f*" to "*t*" while keeping the other conditions fixed.
  - ❖ **Advantages:**MC/DC test sets are effective.
  - ❖ **Disadvantages:**It cannot be broken down into simpler Boolean expressions.

According to the types of faulty detection techniques, Performance of MCDC is much better than BOR for all kinds of Faults. The size of the test suite is also comparable to BOR.MUMCUT detects all faults detected by MI and the test generated is a subset of test sets generated by MI and the size of test suit is much smaller.

III. RELATED WORK

***A. Gargantini* (2011)**[4]proposedWhen testing a Boolean expression, one should consider also the constraints among the variables contained in it. Constraints model interdependence among the conditions in the expressions. The author presented three ways to deal with such constraints: (1) ignoring them during test generation and removing invalid tests later, (2) including them in the expression as conjoint and again removing invalid tests later, and (3) considering them during the test generation process in order to generate only valid tests from the start. Meanwhile, introduced a general framework in which the three policies are implemented and compared over a set of Boolean expressions commonly used as benchmarks. Although the third policy requires a constraints solving technique for actual test generation, it presents several benefits: it generates smaller test suites and it may require less time for tests generation.

***G. Fraser and A. Gargantini*(2011)**[5] described a method that generates test cases directly from an expression's possible faults, guaranteeing that faults of any chosen class will be detected. In contrast to many previous criteria, this approach does not require the Boolean expressions to be in disjunctive normal form (DNF), but allows expressions in any format, using any deliberate fault classes.

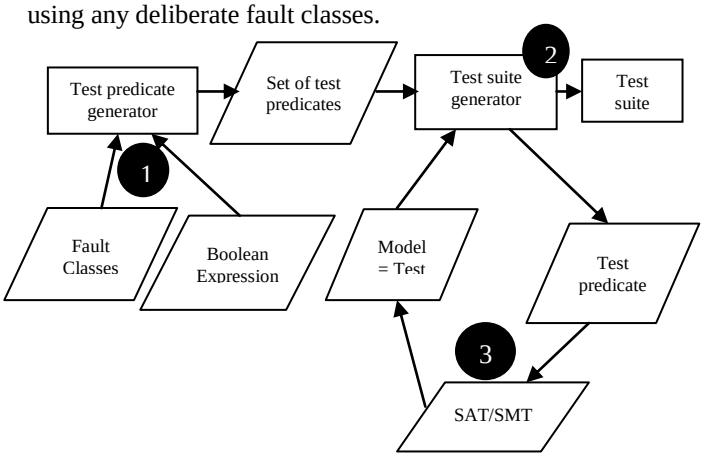
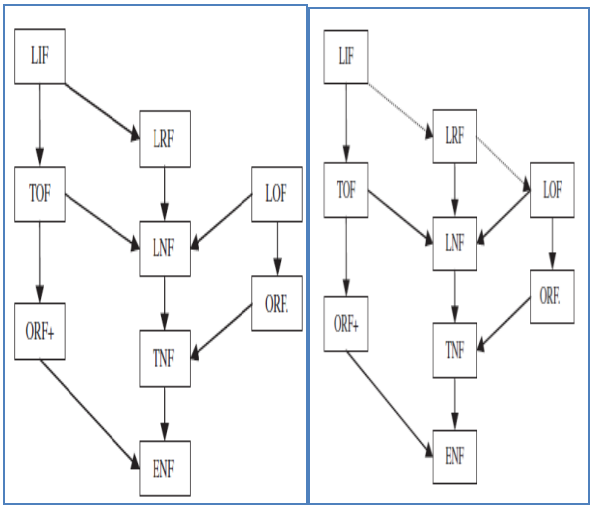


Fig.1: The basic process of generating tests

In figure 1, given a set of fault classes and Boolean expressions, test predicates are generated (1). The test suite generator (2) uses a SAT or SMT (Satisfiability Modulo Theories) solver to look for a model for each of these test predicates; if a model exists, this can serve as a test case.

***G. Kaminski and P. Ammann*(2011)**[6] presentedminimal Conjunctive Normal Form (CNF), and general form Boolean expressions. With Minimal-MUMCUT, a determination is made of which constituent criteria are feasible, and hence necessary, at the level of individual literals and terms. An empirical study found that Minimal-MUMCUT reduces the test set size, without sacrificing fault detection, regardless of the predicate format.

In figure 2 (a), shows a solid arrow from a source fault to a destination fault indicates that if a test detects a source fault, it also detects a corresponding destination fault. Figure 2 (b) a solid arrows from a source faults to a destination faults again indicates that if a test detects a source fault, it also detects a corresponding destination fault.



(a) (b)

Fig.2: (a) Fault class hierarchy Modified; (b) fault class hierarchy.

***G. Kaminski, et.al.*(2011)**[7] explored the idea of addressing these issues by selectively generating only specially engineered subsuming higher order logic mutants. However, such an approach is only useful if a test set that kills all such mutants also kills a high percentage of general mutants. *Method:* An empirical study was conducted using a tool that generates only subsuming higher order logic mutants and tools that generate general mutants. Both Java code and SQL were used as the source under test.

***Godefroid, P., Levin, M. Y., and Molnar, D.*(2012)** [8] presented Whitebox fuzzing was first implemented in the tool SAGE, short for Scalable Automated Guided Execution. Because SAGE targets large applications where a single execution may contain hundreds of millions of instructions, symbolic execution is its slowest component. Therefore, SAGE implements a novel directed search algorithm—dubbed generational search—that maximizes the number of new input tests generated from each symbolic execution.

***Peleska, J.*(2013)**[9] presented a model-based testing (MBT) is considered as leading-edge technology in industry. The key factors for successful industrial-scale application of MBT are described, both from a scientific and a managerial point of view. With respect to the former view, to describe the techniques for automated test case, test data and test procedure generation for concurrent reactive real-time systems which are considered as the most important enablers for MBT in practice.

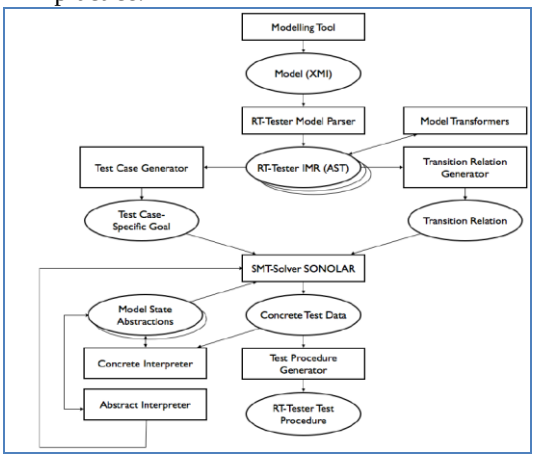


Fig.3: Components of the RT-Tester test case/test data generator

The starting point for MBT is a concrete test model describing the expected behaviour of the system under test (SUT) and, optionally, the behaviour of the operational environment to be simulated in test executions by the testing environment (TE) (see Fig. 3).

***P. Arcaini, A. Gargantini, and E. Riccobene*(2015)** [10] discussed the context of automatic test generation, the use of propositional satisfiability (SAT) and Satisfiability Modulo Theories (SMT) solvers is becoming an attractive alternative to traditional algorithmic test generation methods, especially when testing Boolean expressions. The main advantages are the capability to deal with constraints over the inputs, the generation of compact test suites, and the support for fault detecting test generation methods. However, these solvers normally require more time and a greater amount of memorythan classical test generation algorithms, making their applicability not always feasible in practice.

***Lian Yu and Wei-Tek Tsai*(2018)** [11] discussed characterizes Boolean expression faults as changes of the topological structures in terms of shrinking and/or expanding regions in K-map. A cell-covering is a set of cells (test cases) in K-map to cover the fault regions such that faults guarantee to be detected. Minimizing cell covering can be formulated as an Integer Linear Programming (ILP) problem. By analyzing the structures of the constraint coefficient matrix, the original problem can be decomposed into sub-programs that can be solved instead of the original problem, and this significantly reduces the time needed for ILP execution. An efficient approximate algorithm with a tight theoretical bound is used to address those complex Boolean expressions by corresponding the cell-covering problem to the setcovering problem. The optimal approach and the approximate approach are combined into a hybrid process to identify test cases based on the fraction analysis on the ILP relaxation.

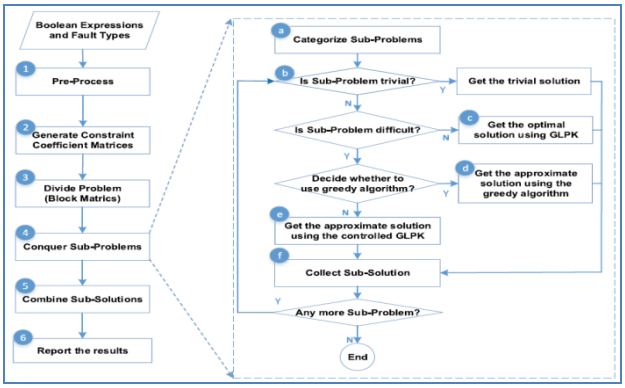


Fig.4: The process to generate test cases for Boolean expressions

IV. BOOLEAN EXPRESSION SOLVING PROCEDURE

A. *Boolean Operator testing Strategy*

A test set $T(E)$ is said to be a BOR test set for E if $T(E)$ satisfies the BOR testing strategy for E. If E is a simple Boolean expression then the minimum BOR test set for E is given by $\{(t),(f)\}$. If E is a compound Boolean expression, then E can be represented as E1 op E2, where op could be either or +, and E1, E2 are either simple or compound Boolean expressions.Seven test cases selected for N7 by applying
aproach{(t,f,t,t)(f,t,t,t)(f,f,t,t)(t,f,t,f)(t,f,f,t)(t,f,f,f)(f,f,f,f)}onfig ure5.

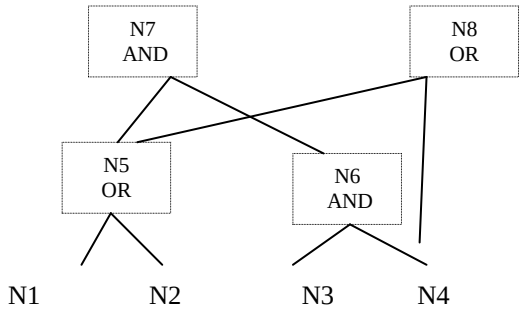


Fig. 5: A cause Effect Graph

Example: a two lowest test sets created for node N7 of figure 1 by applying BORstrategy

S_t (N7)={(t,f,t,t)(f,t,t,t)}
S_f(N7)={(f,f,t,t)(t,f,t,f) or {f,f,t,t)(f,t,t,f)(f,t,f,t)}

B. BOR+MI

The BOR+MI strategy and MI strategy have comparable fault detection capability.

Algorithm for BOR+MI

Step 1: separation Boolean expression *BE* into equally singular components.

Step 2: Create test cases using *BOR* for every singular component.

Step 3: Create test cases using *MI* for non singular components.

Step 4: Join the conditions generated above

Example: Let *BE*=a(bc+¬bd) the generated test cases by BOR+MI are

S^tBE={(t,t,t,f)(t,f,t,t)
S^fBE ={(f,t,t,f)(t,f,t,f)(t,t,f,t)

C. Minimal-MUMCUT

The minimal-MUMCUT selectsthe test points in Unique True Point UTP(i) such that every truth value of every missing variable is covered.

Example: Let $E= ab + cd$

Set of test cases generated using MUMCUT

By applying MUTP strategy{(t,t,f,t)(t,t,t,f)(f,t,t,t)(t,f,t,t)

By applying MNFP strategy{(f,t,f,t)(f,t,t,f)(t,f,f,t)(t,f,t,f)

Byapplying CUTPNFP strategy

{(t,t,f,t)(f,t,f,t)(t,f,f,t)(f,t,t,t)(f,t,f,t) f,,t,t,f)}

The efficiency of above revealed strategies is mostly assessed in terms of their ability in identifying faulty mutations.

V. COMPARISON ANALYSIS

This article, it is assumed that readers are familiar with notations and terminologies of Boolean expressions. This survey aims at presenting such techniques at one place and form a basis for comparison among these techniques.Many different approaches have been projected to assistBoolean Expressions, Fault based Techniques, which has mentioned in a body of literature that is spread over a wide variety of fields and periodicallocations. The comparison of this survey study has been evaluated and particularly in the software testing and software maintenance literature. The table 1 shows the comparative study of Boolean expressions.

Table 1: COMPARISON OF BOOLEAN EXPRESSIONS FAULT BASED METHODS

Title	Algorithm	Key-Idea	Techniques	Results	Performance
Using Logic Criterion Feasibility to Reduce Test Set Size While Guaranteeing Fault Detection(2009) [12]	Minimal-MUMCUT	To reduce test set size without sacrificing fault detection.	Disjunctive Normal Form.	The approach was examined on a sample of predicates (having from 5 to 13 unique literals) in	80% of these predicates were in minimal DNF.

				avionics software.	
Using Logic Criterion Feasibility to Reduce Test Set Size While Guaranteeing Double Fault Detection (2009) [13]	Minimal-MUMCUT and heuristicalgorithm.	To detect double faults and kill second-order mutants.	Logic Testing, Logic Criteria and Fault Coupling.	Parent criterion without sacrificing fault detection.	Double fault types 99.91% of the double faults were detected.
Applications of Optimization to Logic Testing(2010) [14]	Greedy algorithm	To minimize test set size subject to guaranteeing fault detection.	Software Logic Testing, Logic Criteria, MUMCUT, Disjunctive Normal Form.	Maximizing the number of faults detected subject to a test set size.	Fault detection percentage for the worst case (0.35%), random case (6.13%), and best case (15.97%) for a single test.
Dealing with Constraints in Boolean Expression Testing(2011) [4]	Test generation algorithm.	Dealing with Constraints variables.	Ignoring the constraints(IGN), Including the constraints (INC) and Generating only valid tests (VAL).	VAL policy presents several benefits: reduced test suite size, complete fault detection.	Increases the test size by 48%.
Generating Minimal Fault Detecting Test Suites for General Boolean Specifications (2011) [5]	Disjunctive normal form (DNF).	Test cases directly from an expression's possible faults, guaranteeing that faults of any chosen class will be detected.	Fault-based testing and SMT solvers.	Clearly improves over state of the art criteria for general form Boolean expressions.	Reduce the test suite size by 81%.
Reducing logic test set size while preserving fault detection (2011) [6]	Minimal-MUMCUT.	Reducing test set size without sacrificing single or double fault detection.	Minimal DNF and Minimal CNF.	The result in an equivalent fault in that no input can distinguish the original predicate from the faulty version.	Double fault type, 99.91% of the faults was detected.
A logic mutation approach to selective mutation for programs and queries (2011) [7]	Term Insertion Fault (TIF)/Literal Omission Fault (LOF) algorithm.	Addressing the specific miss detection issues.	Logic faults and mutation analysis.	Higher order logic mutation is an effective approach to selective mutation for programs and queries.	Mutation score of 84.57%.
Industrial-Strength Model-Based Testing - State of the Art and Current Challenges (2013) [9]	Model-based testing (MBT)	Techniques for automated test case, test data and test procedure generation.	Satisfiability modulo theory (SMT)	Performed by adding constraints identified during test observations.	70% reduced Model Coverage Test Cases.
How to Optimize the Use of SAT and SMT Solvers for Test Generation of Boolean	greedy, Propositional satisfiability (SAT) and	To deal with constraints over the inputs.	Test computation and Coverage evaluation.	The final results may strongly depend on the choices done at the	Test suite is maximum 10% bigger than the (optimal) smallest

Expressions (2015) [10]	Satisfiability Modulo Theories (SMT).			input level.	test suite
Test Case Generation for Boolean Expressions by Cell Covering (2018) [11]	Approximate algorithm.	Analyzing the structures of the constraint coefficient matrix.	Boolean expression testing, fault characterization.	Obtains optimal solutions quickly, and produces near-optimal solutions rapidly for those rare and complex expressions.	Time consumption is not more than 4 seconds.

Table 2: COMPARISON OF SPEED, COST AND QUALITY BOOLEAN EXPRESSIONS FAULT TECHNIQUES

TECHNIQUES	SPEED	COST	QUALITY
Disjunctive Normal Form (DNF)	Only 3% of these predicates contained five or more unique	High	Fault detection of 1.30% of the size needed if feasibility is not considered.
SMT	40% speed up the test case and test data generation process.	High computational.	Better quality.
Boolean expression testing (BET)	High	Low	High

In table 2 represents a comparison of Boolean expression techniques of speed, cost and quality measures. According to this comparison the Boolean expression by cell covering techniques performs better prediction than other existing techniques.

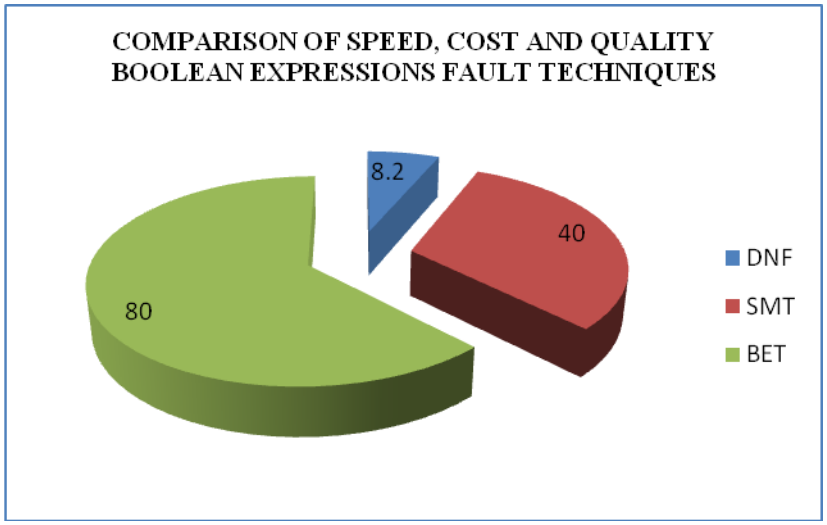


Fig.5: Comparison chart of Boolean expressions faulty techniques

VI. CONCLUSION

This paper presentsa brief survey about Test Case Generation for Boolean Expressions by Cell Covering discussed with the different categories.Much of the published research in fault class analysis was based on empirical evidences, an empirical evaluation of the Boolean expressions and fault detection capabilities based approach has been performed. Boolean expressions from literature various performance and effectiveness of the testing techniques based on fault based analysis. Logic Mutation expressions were generated from the given Boolean expressions by making syntactic change based on particular type of fault. The results were in favour of cell-covering Method for detection of all fault classes, but the size of test suite is large. Boolean expressionapproximate technique has been originally designed for the detection of missing/extra negation operators; therefore, it does not guarantee the detection of other faults.

The further work enhancedand expandedfor the Boolean expressions technique for Depth first Searching algorithm to detect the faults conditions analysis can make the more efficient algorithm.

REFERENCES

[1] K.C. Tai. Theory of Fault Based Predicate Testing for Computer Programs, IEEE Transactions of Software Engineering, vol 22, no 8, pp 552-562, 1996

[2] K.C Tai. M.A Vouk., A. Paradkar., Lu P. , "Predicate Based Testing," IBM Systems Journal, Vol 33 (3), p 445, 1994

[3] M. A. Vouk, K. C. Tai, and A. Paradkar. Empirical Studies of Predicate-based Software Testing. In 5th International Symposium on Software Reliability Engineering, pages 55–64. IEEE, 1994.

[4] A. Gargantini, "Dealing with constraints in boolean expression testing," in Proc. 3rd Workshop Constraints Softw. Testing Verification Anal., Mar. 25, 2011, pp. 322- -327.

[5] G. Fraser and A. Gargantini, "Generating minimal fault detecting test suites for boolean expressions," in Proc. 3rd Int. Conf. Softw. Testing Verification Validation Workshops, Apr. 2010, pp. 37–45.

[6] G. Kaminski and P. Ammann, "Reducing logic test set size while preserving fault detection," Softw. Testing, Verification Rel., vol. 21, pp. 155–193, 2011.

[7] 21] G. Kaminski, U. Praphamontripong, P. Ammann, and J. Offutt, "A logic mutation approach to selective mutation for programs and queries," Inf. Softw. Technol., vol. 53, pp. 1137–1152, 2011.

[8] Godefroid, P., Levin, M. Y., and Molnar, D. (2012) SAGE: Whitebox fuzzing for security testing. Commun. ACM, 55, 40-44.

[9] Peleska, J. (2013) Industrial-strength Model-Based Testing - state of the art and current challenges. In Petrenko, A. K. and Schlinglo, H. (eds.), Proceedings Eighth Workshop on Model-Based Testing, MBT 2013, Rome, Italy, 17th March 2013, EPTCS, 111, pp. 3-28.

[10] P. Arcaini, A. Gargantini, and E. Riccobene, "How to optimize the use of SAT and SMT solvers for test generation of boolean expressions," Comput. J., vol. 58, pp. 2900–2920, Jan. 21, 2015.

[11] Lian Yu and Wei-Tek Tsai, "Test Case Generation for Boolean Expressions by Cell Covering", IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 44, NO. 1, JANUARY 2018.

[12] Kaminski G, Ammann P. Using logic criterion feasibility to reduce test set size while guaranteeing fault detection. Proceedings of the 2nd International Conference on Software Testing, Verification and Validation, Denver, CO, April 2009; 167–176.

[13] Kaminski G, Ammann P. Using logic criterion feasibility to reduce test set size while guaranteeing double fault detection. Proceedings of the Mutation Workshop at the 2nd International Conference on Software Testing, Verification and Validation, Denver, CO, April 2009.

[14] G. Kaminski and P. Ammann, "Applications of optimization to logic testing," in Proc. Softw. Testing, Verification Validation Workshops, 2010, pp. 331–336.

