SAT Based Automated Test Case Generation For MUMCUT Coverage

Jun Yan[1,2] and Jian Zhang[1]
[1] Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences
[2] Graduate University, Chinese Academy of Sciences
{yanjun,zj}@ios.ac.cn

Abstract

MUMCUT is a criterion for testing Boolean specifications. The traditional test case generation methods for this problem are based on approximate approaches. The efficiency of these algorithms depend on the designers' experience on this coverage. In this paper, the authors apply SAT based method to solve this problem. Two SAT solvers, walksat and zChaff, are employed to generate approximate and optimal test case set respectively. The experimental results show that the two SAT methods are efficient.

1 The Problem

A typical problem in software testing is test case generation for SUT (System Under Test). A possible way to reduce the number of test cases is to find those cases satisfying some test criteria. Many criteria are coverage style, i.e. for each pattern, at least one of test cases cover it.

The sophisticated MUMCUT [3] coverage strategy is developed for testing software embedded with complex logical decisions adequately. The MUMCUT coverage criterion is a hybrid test coverage which integrates the MUTP, MNFP and CUTPNFP strategies for detecting the same types of fault for IDNF of boolean expressions. The IDNF (Irredundant Disjunctive Normal Form) is a DNF form in which no variable is redundant (Or it can not be reduced to an equivalent expression which has fewer literals). A test case is a set of values assigned to the variables of the IDNF.

For a decision $S = p_1 + \ldots + p_m$, a test case $\overrightarrow{t}$ is a true point (respectively a false point) iff $S(\overrightarrow{t}) = T$ (respectively $S(\overrightarrow{t}) = F$). If $p_i(\overrightarrow{t}) = T$ and $p_j(\overrightarrow{t}) = F$ for every $j \neq i$, then $\overrightarrow{t}$ is said to be a *unique true point* (UTP) for the i'th term p_i of S. Furthermore, let $p_i = x_1^i x_2^i \cdots x_{k_i}^i$, where x_j^i is the j'th literal in p_i. We use $p_{i,\bar{j}} = x_1^i \cdots \overline{x_j^i} \cdots x_{k_i}^i$ to denote the term obtained from p_i by negating its j'th literal x_j^i. A test case $\overrightarrow{f}$ is said to be a *near false point* (NFP) of the literal x_j^i of p_i of S if $S(\overrightarrow{f}) = F$ and $p_{i,\bar{j}}(\overrightarrow{f}) = T$. If a pair of tests, a UTP $\overrightarrow{t}$ and an NFP $\overrightarrow{f}$, differ only in the corresponding truth value of the j'th literal of p_i, we say this pair is a corresponding UN pair.

For every i, if the test set contains UTPs of i'th term p_i such that all possible truth values (that is, T and F) of every variable not occurring in p_i are covered, we say the test set satisfies the **MUTP** (Multiple Unique True Point) criterion. For every i and j, if the test set contains NFPs of j'th literal of the i'th term p_i such that all possible truth values of every variable not occurring in p_i are covered, we say the test set satisfies the **MNTP** (Multiple Near False Point) criterion. The **CUTPNFP** (Corresponding Unique True Point and Near False Point Pair) requires for every i and j, as far as possible, the test set contains a UTP $\overrightarrow{t}$ of p_i and a NFP $\overrightarrow{f}$ of the j'th literal of p_i such that $\overrightarrow{t}$ and $\overrightarrow{f}$ are UN pair. For more about the MUMCUT coverage, please refer to paper [3].

This criterion is proved to have stronger fault-detecting ability of test sets than the MC/DC and some other related coverage criteria for logical decisions [3]. Meanwhile, the constraints are more complicated than other related criteria. Due to the complexity of the criterion, test case generation is a difficult problem. Existing methods are mainly based on greedy or random methods. For such methods we do not know the least number of test cases that are needed to achieve MUMCUT strategy. In this paper, we describe a complete approach which can find minimal set of test cases.

2 The Approach

The SAT problem is the first proved NP-complete problem and well researched. There are two types of solving methods for SAT problems: heuristic local search and systematic search. The first method is approximate and may not definitely get a solution even for a satisfiable instance; The latter one is complete but may cost much time. We can translate the decision problem (i.e, for a given size N, can we find a test set satisfying MUMCUT criterion?) to SAT and employ the solvers to generate test set.

To speed up the systematic search process, a possible efficient improving method is to reduce the search space by adding SB (symmetry-breaking) constraints. Two solutions are isomorphic if one can be obtained from the other by permuting element names. Because of the isomorphism, one solution may be represented in many ways, which results in much redundancy in the search space. We say that a problem has symmetries if it has isomorphic solutions. We use a matrix to represent a test set. Each row denotes a test case and each column denotes a value for an IDNF variable. So there are two types of symmetries in our problem: row symmetries (the test cases are isomorphic such that we can permute each two rows of the matrix) and column symmetries (some parameters of the IDNF are isomorphic). We add the SB clauses to the SAT set during the SAT encoding to provide partial orders for these isomorphic elements. Also a symmetry-breaking tool Shatter is used to process the encoded SAT clause set.

Our goal is to find the optimal test set (e.g. the test set that has minimum number of test cases). Let C_N denote the SAT clause set of test set sized N, we can make use of the solvers to try to solve C_N with different N until C_N is satisfiable while C_{N-1} is not. We first use the local search tool walksat [2] to find the near-optimal value N_1. Then we use a complete solver zChaff [1] to try the test set near size N_1, until an optimal test set is generated.

3 Experimental Results and Conclusion

We implemented a tool to translate MUMCUT problem to SAT problem in C programming language. Our translator can also generate some SB clauses. Our translator, the SB tool Shatter and the SAT solvers are integrated into an automatic tool by Perl programs. Our tool can be used to find near-optimal test sets by walksat and the optimal ones by zChaff.

First we use a simple instance $ab + cde$ to show the improvement of the symmetry-breaking clauses for zChaff's processing time. The isomorphic parameters are (a, b) and (c, d, e). The optimal test size of MUMCUT test is 10 and the time cost with different strategies of size 9 and 10 are listed in Table 1. All the times are measured in seconds. We can see that all the techniques decrease the zChaff processing time greatly especially for the unsatisfiable instance.

Table 1. The Efficiency of Symmetry-breaking

Test Size	9	10
Satisfiable?	No	Yes
No SB clauses added	20.149	0.032
Row symmetries	0.008	0.016
Column symmetries	4.908	0.004
Shatter	10.007	0.032
All techniques	0.004	0.012

Then we run some benchmarks. The instances of Table 2 come from [4]. We solved the instances with test size not exceeding 80. The comparison between our two methods and the best results of that paper is listed in Table 2. Note the results of [4] are mean sizes.

Table 2. Some Small Instances

Instance	Number of Variables	Best size of [4]	Our Method	
			walksat	zChaff
T01	7	38.5	39	38
T04	5	11.7	11	11
T06	11	84.0	63	54
T08	8	36.0	43	36
T09	7	16.0	16	16
T13	12	36.0	30	27
T14	7	32.9	25	24
T15	9	58.2	68	43
T19	8	44.3	40	35
T20	7	24.0	24	24

For some instances the walksat performs better than the method of Yu et al. It can be used as a supplement for the complete search method. According to zChaff, for some instances, the results generated by the approximate method are far from the optimal one. This implies that there is still much work to be done to improve the test case generation algorithm for MUMCUT.

Our computational results show that our method is sound and efficient. The walksat method provides an approximate result and the zChaff method gets the optimal one. The benefits of SAT method for test case generation are two-fold: 1) This method can provide either approximate test sets rapidly or optimal ones on a long run by employing different type of solvers. 2) This method is a general approach that we use to can get a solution without deep study of the special problem. So it may be very useful for some newly developed test criteria.

References

[1] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th conference on Design automation table of contents*, pages 530–535, 2001.

[2] B. Selman, H. Kautz, and B. Cohen. Local search strategies for satisfiability testing. In *AAAI-92: Proceedings 10th National Conference on AI*, 1995.

[3] Y. T. Yu and M. F. Lau. A comparison of MC/DC, MUMCUT and several other coverage cirteria for logic decisions. *The Journal of System and Software*, 79(5):577–590, May 2006.

[4] Y. T. Yu, M. F. Lau, and T. Y. Chen. Automatic generation of test cases from Boolean specifications using the MUMCUT strategy. *The Journal of System and Software*, 79(6):820–840, June 2006.