Efficient Decision Ordering Techniques for SAT-based Test Generation*

Mingsong Chen mchen@cise.ufl.edu
Xiaoke Qin xqin@cise.ufl.edu
Prabhat Mishra prabhat@cise.ufl.edu

Department of Computer and Information Science and Engineering
University of Florida, Gainesville, FL 32611, USA.

Abstract

Model checking techniques are promising for automated generation of directed tests. However, due to the prohibitively large time and resource requirements, conventional model checking techniques do not scale well when checking complex designs. In SAT-based BMC, many variable ordering heuristics have been investigated to improve counterexample (test) generation involving only one property. This paper presents efficient decision ordering techniques that can improve the overall test generation time of a cluster of similar properties. Our method exploits the assignments of previously generated tests and incorporates it in the decision ordering heuristic for current test generation. Our experimental results using both software and hardware benchmarks demonstrate that our approach can drastically reduce the overall test generation time.

1 Introduction

Due to the increasing complexity and decreasing time-to-market requirement, functional validation of System-on-Chip (SOC) designs is becoming a major bottleneck. Conventional simulation based validation methods mainly adopt two kinds of tests: constrained-random and directed tests. Compared to constrained-random testing, directed testing uses fewer tests to achieve required functional coverage. Thus the overall validation effort can be reduced. However, most directed test generation methods need the expert knowledge of design under validation. The inevitable human intervention makes directed test generation laborious, time consuming and error-prone. So it is necessary to develop efficient techniques to automate the process of directed test generation.

Model checking is recognized as one of the most promising methods for automated directed test generation. In SOC functional validation, the design is transformed to a formal specification, and the negation of the coverage requirements are derived as safety properties in the form of temporal logic. During verification, the model checker exhaustively enumerates all the possible states. If one state contradicts the specified property, the model checker will report a counterexample. Such a counterexample is a sequence of variable assignments which can be refined to a test. Validation of SOC design using these tests can guarantee coverage requirement. To relieve the state space explosion problem when checking complex designs, Boolean Satisfiability (SAT) based Bounded Model Checking (BMC) [1] is proposed. By unrolling the design and the property k times, BMC converts the *k step* state search problem into a SAT problem. If the property fails within k steps, a SAT solver will report a satisfiable assignment (counterexample).

Decision ordering plays an important role during the search because different ordering implies different search path which strongly affects the search time. Most existing decision ordering methods focus on exploiting the useful information of general SAT problem with only one SAT instance. Generally for test generation, a design may have various properties and BMC will check each of them individually. Based on the same design, similar properties describe correlated functional scenarios. Therefore the respective counterexample are expected to have a significant overlap. This paper exploits the decision ordering information in the context of test generation involving one design and multiple properties. The contribution of this paper is the development of an algorithm to drastically reduce the overall directed test generation time by using our proposed heuristics on decision ordering. We integrated our method into the SAT solver zChaff [2]. Experimental studies show that our method can drastically reduce the overall test generation time.

The rest of the paper is organized as follows. Section 2 presents related work on SAT-based BMC techniques and decision ordering heuristics. Section 3 describes SAT-based BMC. Section 4 proposes our test generation methodology using efficient decision ordering techniques. Section 5 presents the experimental results. Finally, Section 6 concludes the paper.

2 Related Work

Model checking [3] techniques have been widely accepted as a promising method for automatic test generation. Due to the scalability issues of conventional Binary Decision Diagram (BDD) based methods, SAT based BMC is proposed as a complementary solution for large designs. Many studies in both software and hardware domain [4] show that BMC has better capacity and productivity over unbounded model checking for real designs. Currently, various techniques based on conflict clause forwarding and variable ordering [5] are proposed to further improve the efficiency of BMC based test generation.

Due to the incremental nature of BMC, SAT techniques [6, 7]

*This work was partially supported by NSF CAREER award 0746261.

try to exploit the commonality between SAT instances and reuse previously learned conflict clauses to prune current search tree. Strichman [7] found that when solving the SAT instance series of a property, some conflict clauses can be replicated and forwarded because of the symmetry of the transition part of the property. In [8], Mishra and Chen observed that during directed test generation using SAT-based BMC, similar properties can be clustered and solved together. For each cluster, they reused the learned knowledge of base property to solve other properties. Our method in this paper uses variable ordering instead of conflict clauses as learning knowledge to reduce the overall test generation time. The comparison between our method and [8] is shown in Section 5.

Different variable ordering will lead to different search trees, therefore branching heuristics can improve the SAT searching performance significantly [9]. As a popular SAT solver, zChaff uses the Variable State Independent Decaying Sum (VSIDS) heuristic [5]. This heuristic contains two parts: i) the static part collects the statistics of the Conjunctive Normal Form (CNF) literals prior to SAT solving and sets the initial decision ordering, and ii) during the SAT solving, the dynamic part periodically updates the priority based on conflict clauses. Although the above general-purpose heuristics are promising for propositional formulas, they neglect some unique information of BMC. In [10], Strichman exploited the characteristics of the BMC formulas for a variety of optimizations including decision ordering. When the bound is unknown, SAT-based BMC needs to increase the unrolling depth one-by-one until finding a counterexample. Wang et al. [11] analyzed the correlation among different SAT instances of a property. They used the *unsatisfiable core* of previously checked SAT instances to guide the variable ordering for the current SAT instance.

To the best of our knowledge, all the existing approaches exploit variable ordering to improve the SAT solving time involving only one property (one SAT instance or several correlated SAT instances with different bounds). Our approach is the first attempt to use decision ordering to reduce the test generation time for a cluster of similar properties.

3 Background

SAT-based BMC is very promising to locate the errors and report the counterexample for a faulty property when bound is known a priori. Given a model M, a safety property p, and a bound k, BMC will unroll the model k times and transform the problem into a Boolean formula as follows:

$$BMC(M,p,k) = I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i,s_{i+1}) \wedge \bigvee_{i=0}^{k} \neg p(s_i)$$

It consists of three parts: i) $I(s_0)$ presents the system initial state, ii) $T(s_i,s_{i+1})$ describes the state transition from state s_i to state s_{i+1}, and iii) $p(s_i)$ tests whether property p is true on state s_i. Then this formula will be transformed to CNF and solved by SAT solvers. Semantically, if there is a satisfiable assignment for this property, then the property is false, written $M \not\models_k p$. Otherwise, it means that the property holds for the design within bound k, written $M \models_k p$.

```
while (1){
    run_periodic_functions();
    if (decide_next_branch()) {
        while (deduce() == CONFLICT) {
            blevel = analyze_conflicts();
            if(blevel < 0)
                return UNSAT;
        }
    }
    else return SAT;
}
```

Figure 1. DLL search procedure of zChaff

Davis-Putnam-Logemann-Loveland (DPLL) algorithm [12] is widely used for SAT search. Figure 1 shows its implementation in zChaff. It contains three parts:

- Periodic function updates the SAT configuration triggered by some specified events, such as updating the scores of literals after a certain number of backtracks.
- Boolean Constraint Propagation (BCP) is implemented in *deduce*. It figures out all possible implications by previous decision assignment.
- Conflict analysis does a proper backtrack when encountering a conflict. It analyzes the reason for the conflict and make it as a conflict clause to avoid the same conflict in future processing.

Decision ordering plays an important role during the SAT search. In zChaff, each literal l is associated with a *zchaff_score(l)* which is used for decision ordering at *decide_next_branch()*. Initially the score is equal to the literal count in corresponding CNF file. During the SAT solving, the score will be updated in periodic function after a certain numbers of backtracks. The calculation of the new literal score is as follows:

$$chaff\_score(l) = chaff\_score(l)/2 + lits\_in\_new\_confs(l)$$

where *lits_in_new_confs(l)* is the number of newly added conflict clauses which contain literal l since last update.

Studies show that modern SAT solvers spend approximately 80% of time to carry out BCP. In addition, during the conflict analysis, long distance backtracks will increase the burden of SAT solvers. Our method tries to optimize both parts by using the learning from decision ordering. The learning can guide the SAT search so that it can drastically reduce the search time.

4 Test Generation using Decision Ordering

For model checking based directed test generation, each property is a negation of a desired system behavior. Thus it

is assumed that each property can produce a counterexample (test). The properties which are used to detect the similar functional errors have a large overlap and can be clustered. This section presents our decision ordering heuristic to reduce the overall test generation time for a cluster of similar properties. The remainder of this section is organized as follows. In Section 4.1, we present an overview of our approach using an illustrative example. Section 4.2 presents our decision ordering heuristic. Finally, Section 4.3 proposes an algorithm which incorporates our heuristic.

4.1 Overview

As discussed in Section 3, the most time consuming parts are BCP and long distance backtracking. They are indicated by implication number and conflict clause number which represent the successful decision ratio and backtrack number respectively. Ideally, a search method can get a satisfiable assignment by making the assignment for each variable only once. However, generally it is impossible to achieve such scenario. For a cluster of similar properties and pre-determined bounds, the objective of our method is to reduce the number of implications and conflict clauses of unchecked properties by incorporating the learned decision ordering knowledge from previously checked properties.

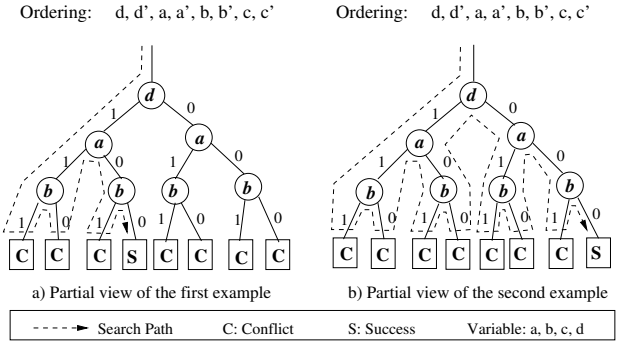

Figure 2. Two examples of SAT search

If we have two similar properties, both properties will have a large overlap on CNF clauses and counterexample assignments. Figure 2 shows the partial views of search trees and search paths of the two properties. The search paths are formed according to the decision ordering (shown on top of the search trees). For each variable v in the ordering, there are two literals (v means $v = 1$ and v' means $v = 0$). As shown in the Figure 2a, there are 3 conflicts encountered. The search stops after finding a satisfiable assignment $a = 0$, $b = 0$, $c = 1$, $d = 1$ in this scenario. In Figure 2b, the search will be successful only when $a = 0$, $b = 0$, $c = 1$, $d = 0$ after encountering 7 conflicts. Therefore the search of the second example will be more time-consuming because of more backtracks.

Because of the large overlap in the assignment of counterexamples, the result of previously checked properties can be used as a learning for unchecked properties. For example, in Figure 2, the result of first example strongly indicates the assignment of the second example because of the satisfiable assignment intersection $a = 0$, $b = 0$, $c = 1$. If the second example use the decision ordering based on the variable assignment in the intersection first, the searching time of the second example can be drastically reduced as shown in Figure 5.

4.2 Decision Ordering Heuristic

In this section we present our heuristic as well as its implementation to improve the overall test generation time. Our heuristic consists of both ordering of values as well as variables.

4.2.1 Bit Value Ordering

Similar properties generally have a large intersection on both corresponding CNF clauses and counterexample assignments. This indicates that the satisfiable assignment of checked SAT instances contain rich decision ordering knowledge for unchecked satisfiable SAT instance. In SAT search, incorrect value selection for each variable will cause conflicts which will result in backtracks to remove the reason of the conflicts. A good decision ordering can mostly avoid such faulty assignments. Unlike pruning the search tree using conflict clause forwarding [8], bit value ordering changes the *search path*. By setting the *bit priority* (choose 0 or 1 first) for each variable using the knowledge of previous property checking, the length of the search path can be reduced.

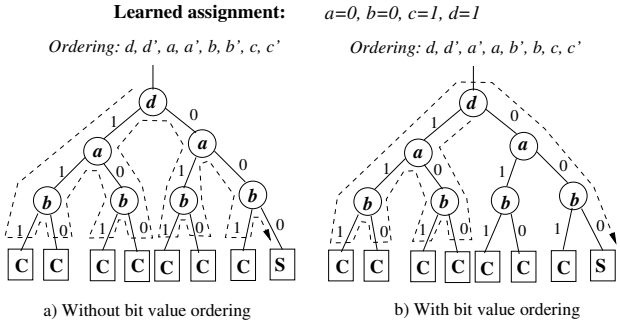

Figure 3. A scenario where bit value ordering works

Figure 3 shows an example where bit value ordering works. As shown in Figure 2a, we can get a satisfiable assignment $a = 0$, $b = 0$, $c = 1$ and $d = 1$. This assignment can be used to change the decision ordering of the second example. That means, when node a is encountered, the search chooses $a = 0$ first in its search path. The same rule also applies on other nodes. Applying such heuristic in Figure 3b, there are only 4 conflicts encountered compared to 7 conflicts in Figure 3a. In addition the search path is also shortened. Therefore the searching time is reduced.

It is important to note that the bit value ordering itself is not always helpful for the SAT searching. For example in Figure 4, $a = 1$, $b = 0$, $c = 1$, $d = 1$ is the only satisfiable assignment in the given scenario. The searching in Figure 4a without bit value ordering is faster than the searching in Figure 4b because of less conflicts. As another example, if the learning assignment in Figure 4 is a=0, b=0, c=1 and d=0, the searching performance

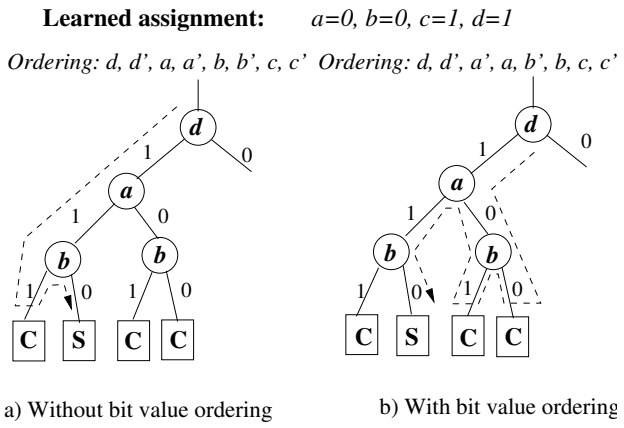

Figure 4. A scenario where bit value ordering fails

will be worse than the search in Figure 4b. Clearly, in the search tree, the high level variables (e.g. node d) strongly affect the performance of the searching if they are not consistent with learned bit value ordering.

4.2.2 Variable Ordering

Although bit value ordering is promising in general, there are still a lot of conflicts encountered during the search. According to the example shown in Figure 4, if high level nodes (e.g. node d) make the wrong decision, the search path will be lengthened due to the long distance backtrack. To reduce the searching time, it is necessary to restrict the conflict detection and reasoning in a small area.

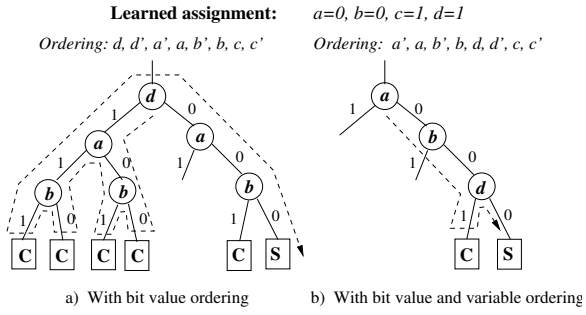

Figure 5. An example of bit value and variable ordering

Efficient combination of variable ordering and bit value ordering is very promising. As shown in Figure 5b, the search time is better than that in Figure 5a due to a shorter search path and less conflicts. The reason of this improvement is that we enhance the priority of literal a' and b'. Since d is the variable with different value between the two satisfiable assignments shown in Figure 2, lowering down the priority of such variables with the potential different value between two CNFs can efficiently avoid the long distance backtrack. Generally, before SAT solving, it is hard to figure out the difference between two satisfiable CNF variable assignments. However, based on the value assignment statistics of the checked properties, the variable ordering can be constructed. For a variable with the lower assignment value variation which indicates high chance of same value, we will enhance its priority by increasing the score of its two literals.

4.2.3 Heuristic Implementation

In our heuristic implementation, we predict the decision ordering based on the statistics collected from the checked properties. Let $varStat[sz][2]$ (sz is the largest variable number for CNFs) be a 2-dimensional array to keep the count of variable assignments. Initially, $varStat[i][0] = varStat[i][1] = 0$ ($0 < i \leq sz$). $varStat$ will be updated after checking each property. Assuming we are now checking property p_j, if the value of variable v_i in the assignment of the p_j is 0, then $varStat[i][0]$ will be increased by one; otherwise, $varStat[i][1]$ will be increased by one. This updated information of $varStat$ will be utilized when checking property p_{j+1}.

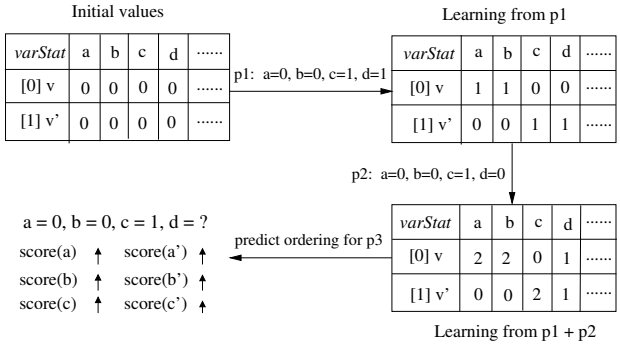

Figure 6. Statistics for two properties

For example, if we have three properties p_1, p_2 and p_3, the statistics after checking p_1 and p_2 are shown in Figure 6. When checking p_3, we can predict its decision ordering based on the collected information saved in $varStat$. The content of $varStat$ indicates that variables a and b are more likely to be 0, c is more likely to be 1 and d can be assigned any value. Furthermore, $varStat$ implies the assignments for variable a, b and c are more consistent than the assignment for variable d. Thus the score of variable a, b and c will be increased. In other words, they will be searched first as described in Section 4.2.2.

Assuming l_i is a literal of v_i, we use the following equation to predict the bit value assignment of v_i when checking p_{j+1}.

$$potential(l_i) = \begin{cases} 1 & (varStat[i][1] > varStat[i][0] \& l_i = v_i) \\ & or(varStat[i][1] < varStat[i][0] \& l_i = v_i') \\ 0 & otherwise \end{cases}$$

For example, $potential(l_i) = 0$ means that value of l_i is more likely to be 0 in the satisfiable assignment of p_{j+1}. For example, in Figure 6, $potential(a) = 0$ which means that a is more likely to be assigned with 0. Let

$$ratio(i) = \frac{max(varStat[i][0], varStat[i][1]) + 1}{min(varStat[i][0], varStat[i][1]) + 1}$$

indicates the assignment variance of variable v_i. The larger $ratio_i$ means the value assignments for variable v_i are more consistent. So it can be used for variable ordering.

Our decision heuristic is based on VSIDS. The only difference is that our method incorporates the statistics of previously checked properties. For each literal l_i, we use $score(l_i)$ to describe its priority. Initially, $score(l_i)$ is equal to the literal count of l_i. At the beginning of search as well as periodically decaying time, the literal score will be recalculated using the following equation where $max(v_i) = MAX(score(v_i), score(v_i')) + 1$.

$$score(l_i) = \begin{cases} max(v_i) * ratio(i) & pontential(l_i) = 1 \\ score(l_i) * ratio(i) & otherwise \end{cases}$$

4.3 Test Generation using Our Heuristics

In this paper, we assume that the bound is pre-determined and given as an input to our method. Determination of bound is hard in general. However, for directed test generation, the bound can be estimated by exploiting the structure of the design. Algorithm 1 describes our test generation methodology. The inputs of the algorithm are a formal model of the design and a cluster of similar properties. The first step initializes *varStat* which is used to keep statistics of the variable assignments. After generating a CNF for the base property p_1 in step 2, step 3 solves the CNF using $chaff\_score$ presented in Section 3 without any learning techniques. In step 4, *varStat* is updated after the satisfiable assignments of p_{i-1} are achieved. Step 5 generates CNFs for unchecked properties, and step 6 checks such properties using our decision ordering learning techniques. Finally, the algorithm reports all the generated counterexamples (tests).

Algorithm 1: *Test Generation using Decision Ordering*
Inputs: i) Formal model of the design, *D*
ii) A cluster of similar properties, *P*, with satisfiable bounds
Outputs: Test-suite
Begin
1. Initialize *varStat*;
2. Select the base property p_1 and generate CNF, CNF_1
3. ($assignment_1$, $test_1$) = SAT(CNF_1, $chaff\_score(CNF_1)$)
Test-suite = $\{test_1\}$
for *i* is from 2 to the size of cluster P
4. Update *varStat* using $assignment_{i-1}$
5. Generate CNF, $CNF_i = BMC(D, p_i, bound_i)$
6. ($assignment_i$, $test_i$) = SAT(CNF_i, $score(CNF_i)$)
Test-suite = *Test-suite* $\cup$ $test_i$
endfor
return *Test-suite*
End

5 Experiments

This section presents two case studies: a VLIW implementation of the MIPS architecture [13] and a stock exchange system. We used NuSMV [14] to generate the CNF clauses (in DIMACS format) and modified the zChaff [2] as our SAT solver. We implemented our decision ordering heuristic on top of VSDIS. The experimental results are obtained on a Linux PC using 2.0GHz Core 2 Duo CPU with 1 GB RAM.

5.1 A VLIW MIPS Processor

The MIPS processor consists of five pipeline stages: fetch, decode, execute, memory and writeback. We applied our methodology to generate the required directed tests for four pipeline paths in the execute stage (ALU, FADD, MUL, DIV). Due to the similarity, we cluster the properties of each path together to share the learning. There are 16 properties divided into 4 clusters. Each cluster has a base property. Table 1 shows the results. The first column indicates the properties used for test generation. The second column represents the test generation time using zChaff. The third column shows the result by forwarding conflict clauses among properties [8]. The fourth column indicates the improvement of the method in the third column over zChaff. Following the results of our method in the fifth column, the sixth column shows our improvement over the method proposed in [8]. As expected, there is no improvement for base properties since they are solved first without any learning opportunity. Compared to [8] which uses conflict clause forwarding, our method can get an average of 15.87X improvement using decision ordering.

Table 1. Test Generation Result for MIPS Processor

Prop. (Tests)	zChaff [2] (sec)	Inc. [8] (sec)	Improv. [2] vs [8]	Ours (sec)	Improv. [8] vs ours
ALU [1]	23.20	23.20	1	23.20	1
p_1	20.73	2.74	7.57	0.18	15.22
p_2	21.33	3.01	7.09	0.15	20.07
p_3	18.03	2.70	6.68	0.29	9.31
DIV [1]	18.78	18.78	1	18.78	1
p_4	23.55	2.72	8.66	0.13	20.92
p_5	18.31	3.60	5.09	0.14	25.71
p_6	18.11	3.72	4.87	0.18	20.67
$FADD$ [1]	22.90	22.90	1	22.90	1
p_7	16.95	4.46	3.80	0.23	19.39
p_8	18.89	2.71	6.97	0.16	16.94
p_9	19.80	4.70	4.21	0.39	12.05
MUL [1]	64.21	64.21	1	64.21	1
p_{10}	59.15	3.36	17.60	0.24	14.00
p_{11}	59.65	3.85	15.49	0.45	8.56
p_{12}	73.98	6.28	11.78	0.18	34.89

[1] Base property

During the SAT searching, conflict clause number and implication number strongly indicate the searching time. Figure 7 illustrates the conflict clause generation for each property during the search using different methods. Figure 8 shows the corresponding implication numbers. It can be seen that, by using our method, the number of conflict clauses and implications can be reduced drastically by several order-of-magnitude, which results in significant improvement in test generation time.

5.2 A Stock Exchange System

The formal NuSMV description of the on-line stock exchange system (OSES) is derived from its UML activity dia-

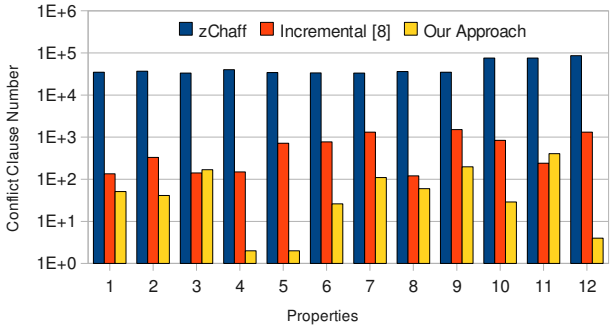

Figure 7. Conflict Statistics for MIPS Processor

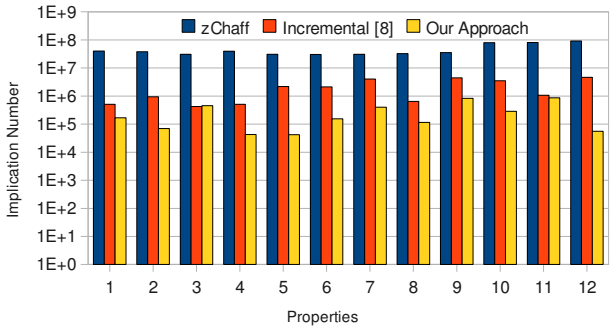

Figure 8. Implication Statistics for MIPS Processor

gram specification, which contains 27 activities, 29 transitions. It mainly deals with three scenarios: accept, check and execute the customers' orders (market orders and limit orders). A path in the UML activity diagram indicates a stock transaction flow. There are a total of 51 properties generated based on path coverage criteria. According to their similarity, we group them into nine clusters.

Table 2. Test Generation Result for Stock Exchange System

Cluster	Size	zChaff [2] (sec)	Inc. [8] (sec)	Improv. [2] vs [8]	Ours (sec)	Improv. [8] vs ours
C_1	3	1.18	2.18	0.54	0.70	3.11
C_2	4	14.53	9.53	1.52	0.78	12.22
C_3	8	375.91	170.06	2.21	36.19	4.70
C_4	4	12.98	8.33	1.56	1.24	6.72
C_5	4	7.13	16.88	0.42	1.02	16.55
C_6	8	720.13	474.68	1.52	28.60	16.60
C_7	4	10.80	24.55	0.44	1.95	12.59
C_8	8	656.95	321.14	2.05	77.65	4.14
C_9	8	248.17	82.42	3.01	37.93	2.17
Avg.	**-**	**227.53**	**123.31**	**1.85**	**20.67**	**5.97**

Table 2 shows the test generation results involving all the 9 clusters. The first column indicates the clusters. The second column indicates the size for each cluster. The third column presents the test generation time (including base property) using zChaff. The fourth and fifth columns indicate the time required to generate the counterexample (test) by using method proposed in [8] and corresponding improvement factor, respectively. The last two columns indicate the test generation time (including base property) and its improvement factor using our heuristic. The last row indicates the average value for each column. In this case study, our approach can produce an average of 5.97X improvement compared to the method proposed in [8].

6 Conclusions

Directed test based simulation is promising for function validation, since running time can be reduced with fewer tests while the coverage requirement can still be achieved. Most automatic directed test generation methods, especially for model checking based techniques, are impeded by the capacity restriction of corresponding tools. To address the complexity of test generation using SAT-based BMC, this paper presented a novel decision ordering heuristic. To the best of our knowledge, our work is the first attempt to share the learning across the decision ordering of multiple properties. By exploiting the commonalities during the search of satisfiable assignments, the test generation time of a set of similar properties can be reduced. The experimental results using both hardware and software designs demonstrated the effectiveness of our method.

References

[1] A. Biere, A. Cimatti, and E. M. Clarke. Bounded model checking. *Advances in Computers*, 58, 2003.

[2] http://www.princeton.edu/ chaff/zchaff.html. *zChaff*.

[3] E. Clarke, O. Grumberg and D. Peled. *Model Checking*. MIT Press, Cambridge, MA, 1999.

[4] N. Amla et al. An analysis of SAT-based model checking techniques in an industrial environment. *CHARME*, 254–268, 2005.

[5] M. Moskewicz et al. Chaff: Engineering an efficient SAT solver. *DAC*, pages 530–535, 2001.

[6] H. Jin and F. Somenzi. An incremental algorithm to check satisfiability for bounded model checking. *BMC*, 51–65, 2004.

[7] O. Strichman. Pruning techniques for the sat-based bounded model checking problem. *CHARME*, 58–70, 2001.

[8] P. Mishra and M. Chen. Efficient techniques for directed test generation using incremental satisfiability. In Proceedings of *International Conference of VLSI Design*, 65–70, 2009.

[9] J. P. Marques-Silva and K. A. Sakallah. The impact of branching heuristics in propositional satisfiability. In Proceedings of *the 9th Portuguese Conference on Artificial Intelligence*, 62–74, 1999.

[10] O. Shtrichman. Tuning SAT checkers for bounded model checking. CAV, 480–494, 2000.

[11] C. Wang et al. Refining the SAT decision ordering for bounded model checking. DAC, 535–538, 2004.

[12] M. Davis et al. A machine program for theorem proving. Communication of the ACM, 5:394–397, 1962.

[13] J. Hennessy and D. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, 2003.

[14] http://nusmv.irst.itc.it/. NuSMV.