A Technique for SAT-based Test Generation through History of Reusing Solutions

Kenji Ueda Fumiyuki Hafuri Toshiya Mukai
Tsuyoshi Iwagaki Hideyuki Ichihara Tomoo Inoue
Graduate School of Information Sciences, Hiroshima City University

Abstract— This paper presents a technique for test pattern generation (TPG) based on Boolean satisfiability (SAT) in a situation where a solution to a SAT instance is reused as the initial truth assignment for solving the successive instance. The efficiency of obtaining a solution to it depends on the order in reusing each variable of the previous solution one by one. The proposed technique utilizes the history of reusing solutions representing whether each variable of previous solutions is successfully reused. Experimental results show that the proposed technique using such a history is effective in test generation time.

I. Introduction

Test pattern generation (TPG) is one of the challenging issues for today's VLSI-CAD. Among many TPG methods previously proposed, TPG based on Boolean satisfiability (SAT-based TPG) is a promising approach. It has been studied for the past twenty years and is still being studied [1].

In efficient SAT-based TPG, many techniques (e.g., Boolean constraint propagation (BCP), non-chronological backtracking, conflict-based learning [1], etc.) are used to accelerate the decision process of solving a single SAT instance. On the other hand, for solving the sequence of SAT instances efficiently, the concept of incremental SAT has been introduced [2]. This concept has been widely applied to specific TPG problems [3, 4]. In our previous work [5], which is also based on incremental SAT, we presented a test generation technique where a solution to a SAT instance is reused to obtain that of another SAT instance, which is similar to the former instance in terms of clauses. Our preliminary experiment shows the effectiveness of reusing previous solutions to similar instances. However, there exists a degree of freedom when each variable of the previous solution is reused one by one for solving a current instance. That freedom can affect the efficiency of generating a solution to the current SAT instance although our previous work does not consider it.

In this work, we consider the history of reusing solutions representing whether each variable of previous solutions is successfully reused, and propose a test generation method considering such a history to utilize the freedom in reusing solutions adequately.

$$
\begin{aligned}
&(D+\overline{B})\cdot(D+\overline{C})\cdot(\overline{D}+B+C)\\
&\cdot(\overline{E}+\overline{A})\cdot(\overline{E}+\overline{D})\cdot(E+A+D)\\
&\cdot(\overline{G}+E)\cdot(\overline{G}+F)\cdot(G+\overline{E}+\overline{F})\\
&\cdot(B)\cdot(\overline{Bf})\\
&\cdot(Df+\overline{Bf})\cdot(Df+\overline{C})\cdot(\overline{Df}+Bf+C)\\
&\cdot(\overline{Ef}+\overline{A})\cdot(\overline{Ef}+\overline{Df})\cdot(Ef+A+Df)\\
&\cdot(\overline{Gf}+Ef)\cdot(\overline{Gf}+F)\cdot(Gf+\overline{Ef}+\overline{F})\\
&\cdot(XG+G+\overline{Gf})\cdot(XG+\overline{G}+Gf)\\
&\cdot(\overline{XG}+G+Gf)\cdot(\overline{XG}+\overline{G}+\overline{Gf})\\
&\cdot(XG)
\end{aligned}
$$

$$
\begin{aligned}
&(A,B,C,D,E,F,G,Bf,Df,Ef,Gf,XG)\\
&=(0,1,0,1,0,1,0,0,0,1,1,1)
\end{aligned}
$$

Fig. 1. Instance $I(f_1)$ for fault f_1 and a solution

$$
\begin{aligned}
&(D+\overline{B})\cdot(D+\overline{C})\cdot(\overline{D}+B+C)\\
&\cdot(\overline{E}+\overline{A})\cdot(\overline{E}+\overline{D})\cdot(E+A+D)\\
&\cdot(\overline{G}+E)\cdot(\overline{G}+F)\cdot(G+\overline{E}+\overline{F})\\
&\cdot(D)\cdot(\overline{Df})\\
&\cdot(\overline{Ef}+\overline{A})\cdot(\overline{Ef}+\overline{Df})\cdot(Ef+A+Df)\\
&\cdot(\overline{Gf}+Ef)\cdot(\overline{Gf}+F)\cdot(Gf+\overline{Ef}+\overline{F})\\
&\cdot(XG+G+\overline{Gf})\cdot(XG+\overline{G}+Gf)\\
&\cdot(\overline{XG}+G+Gf)\cdot(\overline{XG}+\overline{G}+\overline{Gf})\\
&\cdot(XG)
\end{aligned}
$$

$$
\begin{aligned}
&(A,B,C,D,E,F,G,Df,Ef,Gf,XG)\\
&=(0,0,1,1,0,1,0,0,1,1,1)
\end{aligned}
$$

Fig. 2. Instance $I(f_2)$ for fault f_2 and a solution

II. SAT-based TPG

Boolean satisfiability (SAT) is the problem of determining if the variables of a given Boolean formula can be assigned in such a way as to make the formula evaluate to true. Such a SAT instance is usually given in conjunctive normal form (CNF). In SAT-based TPG, the problem of finding a test pattern for a particular fault (e.g., stuck-at fault) is translated into a SAT instance. This translation can be done by using a basic simple gate-to-clause transformation [6]. Figures 1 and 2 show examples of instances for two faults f_1, f_2, which are denoted by $I(f_1), I(f_2)$, respectively, in a circuit. By applying a SAT solver to these instances, we can obtain the solutions, which correspond to test patterns, as shown in the bottoms of the figures.

III. Proposed ideas for efficiently solving SAT instances

A. *Solution reuse and instance similarity*

In general, it frequently occurs that a test vector for a fault is similar to that for another fault. In particular, this tendency may increase if the two faults are close to each other, i.e., SAT instances corresponding to these two faults have a number of common clauses. For example, $I(f_1)$ and $I(f_2)$, which have many common clauses, in Figs. 1 and 2 also resemble in their solutions as shown in the figures. From the above observation, in our previous work [5], we presented a test generation technique where a solution of a SAT instance is reused to obtain that of another instance. The technique first sorts given instances in such a way that adjacent instances have many common clauses. Then, a solution to the previous instance is reused for solving a current instance. The number of

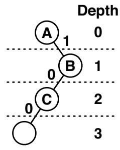

Fig. 3. Decision tree 1

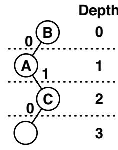

Fig. 4. Decision tree 2

backtracks can be reduced during search because the previous solution is likely to be similar to that of the current instance. Our preliminary experimental results show the effectiveness of reusing previous solutions.

B. History of reusing solutions

Our work discussed in this paper is intended to judiciously reuse the previous solution for solving a current SAT instance in order to improve the above-mentioned technique further. Here, let us consider two cases to solve the instance I_t of $(\overline{A}+\overline{D})\cdot(B+C+D)\cdot(B+\overline{C}+D)\cdot(\overline{B}+C+D)\cdot(\overline{B}+\overline{C}+D)$ by reusing a previous solution $(A, B, C) = (1, 0, 0)$. Notice that a solution to the above instance can be $(A, B, C, D) = (0, 0, 0, 1)$; the value of variable A in the previous solution conflicts with that of the current one. In the first case, each variable of the previous solution is reused one by one in the order of $A \rightarrow B \rightarrow C$ as shown in Fig. 3. The order of $B \rightarrow A \rightarrow C$ (Fig. 4) is considered in the second case. The first case may cause many backtracks in the decision process of deriving the solution of $(A, B, C, D) = (0, 0, 0, 1)$ compared to the second case. That is to say, if the cause of conflict such as variable A exists in a deeper depth of the decision tree during search, the number of backtracks can be reduced potentially. By identifying such variables for solution reuse in advance, the order of reusing variables can be determined so that as many backtracks as possible are suppressed.

We introduce the history of reusing solutions to identify reused variables that are likely to cause backtracks. Given a set of SAT instances $(I_1, I_2, \ldots, I_N)$, which are processed in this order, a solution to I_{i+1} is created by using the history H_j^i for each variable v_j in the instances defined as follows:

$$H_j^i = H_j^{i-1} + \begin{cases} Total(I_i) - Depth(v_j^i) & (\text{if } v_j^i = v_j^{i-1}) \\ Depth(v_j^i) - Total(I_i) & (\text{otherwise}) \end{cases},$$

where $Total(I_i)$ represents the total number of variables in I_i, v_j^i denotes a truth value to v_j reused in solving the i-th instance, and $Depth(v_j^i)$ is the depth at which v_j^i was reused in solving I_i. Note that H_j^0 and H_j^1 are zeros for any variable j. When the proposed method attempts to solve I_{i+1}, each variable will be reused in descending order of its history value. We give an example of calculating history values. Suppose that the solution of $(A, B, C) = (1, 0, 0)$ is reused to solve the instance I_t shown before in the order of $A \rightarrow B \rightarrow C$, and the solution of $(A, B, C, D) = (0, 0, 0, 1)$ is generated. In this situation, variable A fails to be reused while B and C are successfully done. As a result, $Total(I_t) = 4$, $Depth(A^t) = 0$, $Depth(B^t) = 1$ and $Depth(C^t) = 2$ can be obtained. Note that the history value of D cannot be changed for instance I_t in this case because the previous solution does not contain D. If the history values of $(H_A^{t-1}, H_B^{t-1}, H_C^{t-1}, H_D^{t-1})$ are $(2, 1, 0, 0)$ for the previous instance I_{t-1}, $(H_A^t, H_B^t, H_C^t, H_D^t) = (-2, 4, 2, 0)$ are derived.

TABLE I
TEST GENERATION RESULTS UNDER VARIOUS OPTIONS

Circuit	(History, Similarity)	Reused [%]	#Back	Time [s]
s208.1	(Off, Off)	66.32	40,822	6.618
	(On, Off)	66.81	31,897	5.378
	(Off, On)	84.95	26,914	4.741
	(On, On)	**84.81**	**24,228**	**4.281**
s298	(Off, Off)	58.44	23,114	11.665
	(On, Off)	59.98	15,822	8.003
	(Off, On)	78.80	4,323	1.685
	(On, On)	**81.20**	**1,538**	**0.575**
s386	(Off, Off)	60.37	5,899	2.983
	(On, Off)	60.93	5,109	2.491
	(Off,On)	77.77	4,323	1.918
	(On, On)	**80.19**	**3,259**	**1.469**

Thus, the order of $B \rightarrow C \rightarrow D \rightarrow A$ is accepted for reusing $(A, B, C, D) = (0, 0, 0, 1)$ to solve the next instance I_{t+1}.

History values are usually accumulated over all the instances. Here, we consider that history values of only the last α instances are used when a current instance is solved. If the value of α is changed, the overall processing time can vary according to α. It is generally difficult to determine an appropriate value of α. In this work, we empirically set α to 10.

IV. EXPERIMENTAL RESULTS AND FUTURE WORK

We implemented our SAT solver, which is reusing previous solutions and has the capability of Boolean constraint propagation and non-chronological backtracking, in C language, and applied it to all the stuck-at faults in some ISCAS '89 benchmark circuits [7] to generate test patterns. In Table I, "Reused [%]" represents the percentage of variables reused successfully and "#Back" denotes the number of backtracking during search. The item of "(History, Similarity)" shows whether the history and the similarity between instances in reusing solutions are considered. From the table, we can see that each of the two ideas that correspond to the options of "(On, Off)" and "(Off, On)" itself can suppress backtracks and reduce the processing time. If we combine these two ideas together, we can enhance the search efficiency further as listed in bold letters.

In the future, we should evaluate the proposed technique for large benchmark circuits and investigate the possibility of improving the technique further.

REFERENCES

[1] R. Drechsler *et al.*, *Test pattern generation using Boolean proof engines*, Springer, 2009.

[2] J. N. Hooker, "Solving the incremental satisfiability problem," *Journal of Logic Programming*, Vol. 15, pp. 177–186, 1993.

[3] J. Kim *et al.*, "On applying incremental satisfiability to delay fault testing," *Proc. DATE*, pp. 380–384, 2000.

[4] D. Tille *et al.*, "Incremental solving techniques for SAT-based ATPG," *IEEE Trans. on CAD*, Vol. 29, No. 7, pp. 1125–1130, 2010.

[5] T. Iwagaki *et al.*, "An approach to hardware SAT solvers for test generation based on instance similarity," *Proc. WRTLT*, pp. 108–112, 2011.

[6] T. Larrabee, "Test Pattern Generation Using Boolean Satisfiability," *IEEE Trans. on CAD*, Vol. 11, No. 1, pp. 4–15, 1992.

[7] F. Brglez *et al.*, "Combinational profiles of sequential benchmark circuits," *Proc. ISCAS*, pp. 1929–1934, 1989.