Random Stimulus Generation with Self-tuning

Yanni Zhao, Jinian Bian, Shujun Deng, Zhiqiu Kong
Department of Computer Science and Technology, Tsinghua University, Beijing, 100084
zhaoyn05@mails.tsinghua.edu.cn, bianjn@tsinghua.edu.cn,
{dengsj04, kzq07}@mails.tsinghua.edu.cn

Abstract

Constrained random simulation methodology still plays an important role in hardware verification due to the limited scalability of formal verification, especially for the large and complex design in industry. There are two aspects to measure the stimulus generator which are the quality of the stimulus generated and the efficiency of the generator. In this paper, we propose a self-tuning method to guide the generation for constrained random simulation by SAT solvers. We use a greedy search strategy in solving process to get the high-uniform distribution of the stimulus, and improve the efficiency of the generator by affinity grouping. Experimental results show that our methods can generate more uniform random stimulus with good performance.

Keywords: Constrained Random Simulation, Stimulus Generation, SAT, Self-tuning, Greedy Search.

1. Introduction

Constrained random stimulation is mainly adopted by Industrial verification because of the inherently unscalablility of the formal verification methodology [1]. A set of constraints is used to limit and control the input combinations that are sent to the design, and then the simulation enable partial validation of large designs even in the case that errors are hidden sequentially deep. It is a combination between formal methods and random simulation.

One of the previous works is to build a weighted BDD from the input constraints [2]. The stimuli with desired distribution are driven from the root to the terminal vertex. It requires the variables ordered and causes the memory blowups in the case of large designs. The idea in [3] with BDD structures avoids the blowups whereas leading highly skewed distribution.

The efficiency of the generator is as important as the quality of even distribution. This concern is partially addressed by the improvement of the DPLL-based SAT solver [4,5,6].The core of these approaches is to seek out a single solution with efficient constraints solving, while stimulus generation requires repeated generation of solutions with a good distribution over the solution space. A random pre-assignment of variables for the DPLL-style SAT solvers is used in [7] attempting to get a good distribution of the stimuli. However, the quality of the generation has much to do with the case of pre-assignment. Unsuitable decision may skew the solutions into a corner case and lead to highly non-uniform.

Another way to address the problem is drawn from AI community [8,11]. In [8], it uses random-walk to achieve uniformity of the distribution. But the performance is decreasing sharply for the large constraint sets. An alternative is presented in [11], which modifies the CNF-based instance with randomly generated XOR constraints guiding the SAT solver to sample more evenly among its solution space. However, the final generated solutions from both above methods are distributed with some randomization. In this paper, we use a greedy algorithm to dynamically guide the solution solving process. Targeted extra constraints are added into the constraint set according to the current sampling in order to direct the generator to the uniform distribution. Furthermore, performance of the generator is improved by affinity grouping for the input variables.

The rest of this paper is organized as follows: In Section 2, the distribution problem is defined in detail. To settle this problem, Section 3 proposes a methodology based on affinity grouping and greedy algorithm. In Section 4, the performance of the proposed method is verified based on the widely accepted cases. Finally, the conclusion is drawn in Section 5.

2. Problem Definition

In constrained random simulation(CRS), the stimulus generator mush obey the input constraints in order to avoid generating invalid stimuli that might lead to false negative verification results.

First of all, in order to meet the coverage goals as quickly as possible, it requires that the distribution of the input vectors derived by the generator should closely match the distribution of the coverage points. In the absence of the information about the distribution of coverage points, the best choice for the inputs is the ones with high-uniform distribution. It is easy to understand that maximizing the discrepancy of the input stimuli can augment the chance of entering unexplored

978-1-4244-3535-7/09/$25.00 ©2009 IEEE

regions of the state space. It is simple to use the Shannon's entropy for evaluating the distribution of the solution vectors.

$$E_S = -\frac{num\_diff(S)}{N}\log_2(\frac{num\_diff(S)}{N}) \quad (1)$$

where num_diff is the number of different stimuli in simulation patterns S examined from the input constraints set, and N is the number of all possible patterns. The value of the formula ranges from 0 to 1 where lower entropy indicates the evener distribution. In the extreme best condition, the simulation patterns are full of the whole solution space and all the state space is covered.

In this paper, we use SAT solvers to get the stimulus, therefore almost all the generated solutions are unique that it is not able to differentiate between the two groups for the patterns. So we utilize the piecewise-uniform evaluation to avoid the situation that points center on corners of the solution space, in order to increase the level of input "surprise", leading to the state space of design more fully.

Assume that the size of the solution space is separated into m pieces, which are $N_0, N_1, \cdots, N_m$. The entropy of each piece is defined as below according to Equation 1:

$$E_{S_i} = -\frac{num\_diff(S_i)}{N_i}\log_2(\frac{num\_diff(S_i)}{N_i}) \quad (2)$$

Instead of the complete evaluation value, we can evaluate the whole solution vectors with the average of piecewise-uniform values, which is adequate to dedicate the tendency of the patterns uniform, which can also determine the corner case of the distribution. Our goal is to find sample minimizing piecewise-uniform evaluation with input constraint limits.

3. Methodology

In this section, we first present the framework for stimulus generation with self-tuning. Then the core component of affinity grouping and the greedy algorithm in the search processing are applied for the stimulus generation.

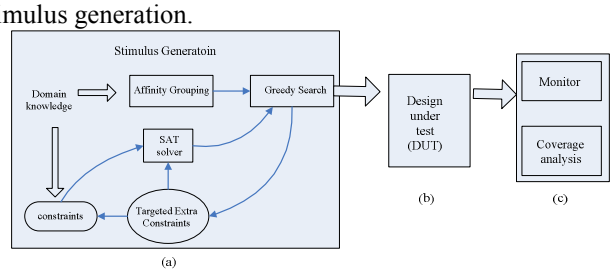

Fig. 1: the framework of the Constrained Random Simulation with self-tuning using affinity grouping and greedy search.

3.1. CRS framework

The framework for CRS is illustrated by Fig.1. It includes three parts: stimulus generator (Fig.1.a), design under test (Fig.1.b) and the monitor with coverage analyzer (Fig.1.c). As mentioned above, we focus on the generator with self-tuning to refine the input stimuli. Our strategy focuses on (Fig.1.a).

Firstly, we analyze the connection of input variables in the stimuli according to the constraints set and complete the affinity grouping. Our strategy is to collect the variables with less connection as a group and make sure the solution full of this part of space as much as possible. The domain of the solution space is decomposed by the groups so that we can evaluate the uniform to determine the corner case which would be handled by the greedy search. The partition is only processed only once in the flow, so the cost of the runtime is shared by the loop generation, making it negligible.

Input constraints are sent to a DPLL-style SAT solver in order to get a single solution as the stimulus sample and the uniform is evaluated as piecewise during the solving process. We use the evaluation result and the current sample to guide the greedy search, producing extra constraints targeting at jumping out of corner area.

Fig.2 displays the pseudo-code for the top level of the presented sampling algorithm. The $AffinitGrouping()$ (Line 1) is the function to implement the Affinity grouping, and $GreedySearch()$ (Line 10) is the procedure that produces the targeted extra constraints used to avoid the corner case. We use the $Recovery()$ (Line 16) process to handle the unsatisfiable result of the SAT solver. The variable $extra$ is a tag to check the backtracking of the extra constraints. If there are no extra constraints after recovery but the result of the constraints solving is still unsatisfiable, it means that there are not enough solutions for the constraints to generate the stimuli with the desired number. The next subsections provide more detail knowledge.

As mentioned in Section 2, SAT solvers do not guarantee the distribution of the generation. The value of new solution at each time of the generation may be different at only few variables. We use the piecewise-uniform evaluation to distinguish the corner case of the stimuli generated by SAT solvers.

Our strategy is to group the input variables with fewer constraint relationships. On the one hand, it can provide the piecewise for the distribution evaluation, which would avoid the corner skew in the stimulus generation. On the other hand, the variables directly connected by the constraints are to be separated into different groups. It means that tuning with one group unit will immediately has an affect on other groups in the solving process because of the constraints between them, which would improve the efficiency obviously. We complete the affinity grouping using relational graph partition.

3.2. Affinity grouping with graph partition

The goal of this procedure is to minimize the total connections between groups while ensuring that vertices among each group have fewer affinities. This objective leads to the proper piecewise evaluation and efficiency improvement for solving process. We perform recursive bisection, which is to make multiple cuts until the variable set is partitioned to a desired granularity. To identify the variables with fewer affinities, we use the relational graph, where variables are nodes; connections are edges and the more direct connections, the less weighted edges between the input variables.

```
Input:
n: bit-width
K: number of expected stimuli
Cons: specified constraints
 1.   AffinitGrouping(); /*divide the variables into groups */
 2.   cur_num ⇐ 0; /*current number of the stimuli*/
 3.   extra ⇐ 0;
      /*whether to add the extra constraints*/
 4.   limit ⇐ 0;
 5.   while(cur_num < K){
 6.     Solver(Cons);
 7.     if(sat){
 8.       stimuli ∪ solution;
        /*get a new stimulus
 9.       Cons ∪ ¬(solution);
      /*not to repeat the stimuli already in existence*/
10.       GreedySearch(solution);
11.       }
12.     else{
13.       if(not extra)
14.         exit();
      /*no enough solution for the stimuli expected*/
15.       else
16.         extra = Recovery()}
17.   }
```

Fig. 2. top level of the stimulus generation with self-tuning

Firstly, we construct totally connected graph with input variables as vertices and weight the edges of the vertices with the total number of the constraints. Then, real constraints between the variables cause the weight value to have decreased by one. After considering all the input constraints, we could build a complete relational graph as requested.

According to Equation 2, we just need to guarantee the difference of the input vectors in each group and make sure the solutions are full of the whole piece space as much as possible.

3.3. Greedy search during the generation process

We propose a mechanism of self-tuning, where the current generated solution can guide the following generation process. Our objective is to guarantee the difference of the input vectors in each group and make sure the solutions are full of the whole piece space as much as possible.

We use the Equation 2 to determine the impact of current generated stimulus on the distribution. We suppose that the ones where no entropy value change occurs are the potential corner cases. Pick out all the cases to form the targeted extra constraints. It is a greedy algorithm in order to prevent the next solution from falling into the parts already existed in these pieces. If all pieces are changed along with the new stimulus, pick up the one with largest entropy instead. The extra constraints in addition to constraint set are sent into the SAT solver for the next new stimulus. Executing the constraints adding-and-solving process in a loop until no satisfiable result is obtained.

The unsatisfiable result from the SAT solver indicates that the greedy search leads solution into illegal areas according to the input constraints. We propose a recovery to pull it back. Considering the efficiency of the generator, we cut the relative constraints in half when unsatisfiability occurs.

4. Experimental Results

We implemented our sampling algorithm for constrained stimulus generation with self-tuning called SGST (stimulus generator with self-tuning). We also implemented the DPLL-based random sampling and the near-uniform sampling algorithm using XOR constraints for comparison of the efficiency, robustness and the distribution quality with our methods.

We derived our benchmarks from the ISCAS89 Sequential Benchmark Circuits [9]. They were expanded for some time-frame like bounded model checking (BMC) [10] and translated into Conjunctive Normal Form (CNF) constraints which is completed by Perl script in linear time. We use the Minisat as solver engine.

For the DPLL-based random sampling, we used the Minisat for repeatedly generation of solutions. We chosen the pre-assignment at random and gain a stimulus at each generation. For the sampling with XOR constraints, we added the XOR constraints randomly between input variables of the circuit to reduce the solution space with high probability, consequently achieving more even distribution.

In order to evaluate the sampling distributions of the algorithms, we use a simple entropy statistics for each variable, which is an evaluation for the distribution of 0s and 1s for a single variable.

$$E(X) = \frac{\sum_i^n (-\frac{N1_{x_i}}{K}\log_2(\frac{N1_{x_i}}{K}) - \frac{N0_{x_i}}{K}\log_2(\frac{N0_{x_i}}{K}))}{n} \quad (3)$$

Where $N1$ is the number of times that $x_i = 1$ while $N0$ is the number of times that $x_i = 0$ among the stimuli patterns, and K is the number the stimuli achieved by the generator. The result of $E(X)$ is the average of the entropy for all the variables involved in the stimuli. The formula gives values that range from 0

to 1, where bigger value indicates higher quality of the distribution, i.e. an even distribution of ones and zeroes. Because of the input constraints, it might not achieve the biggest value in the most cases, but still can evaluate the quality of the distribution properly.

All the experiments were completed on the Intel Xeon 3GHz CPU with 6G RAM. Considering the randomization, we run each benchmark 10 times independently and get the average CPU time and evaluation values at last. The experimental results are listed in Table 1. The first column is the test cases in ISCAS89. ".1"in the name of the "s1238.1" means that bench "s1238" was expanded for 1 time-frame, and so on. Columns named "$\#var$" and "$\#cls$" are the numbers of the variables and clauses in the test bench. The columns left in the table in order from left are: the time and the average entropy respectively for DPLL-based random method, XOR constraints method and our SGST. We supposed number of the stimuli is 16, According to the result in the Table 1; the average entropy of our algorithm is quite more than other methods. Our time is a little more than XOR because of the recovery move in the strategy.

Table 1: Experimental Results for ISCAS89 Testcases ($K = 16$)

Test case	#var	#cls	DPLL		XOR		SGST	
			Time (sec)	Avg_E	Time (sec)	Avg_E	Time (sec)	Avg_E
s1238.1	540	1549	0.15	0.007	0.23	0.427	1.89	0.585
s1423.1	748	1821	0.17	0.016	0.27	0.061	1.11	0.819
s1196.2	1104	3076	0.22	0.005	0.37	0.530	2.29	0.596
s1238.2	1062	3098	0.20	0.005	0.36	0.426	2.95	0.564
s27.10	143	280	0.11	0.054	0.13	0.408	0.24	0.806
s344.10	1705	4290	0.28	0.002	0.49	0.408	2.68	0.781
s349.10	1715	4340	0.28	0.002	0.49	0.245	2.88	0.753
s382.10	1631	4640	0.28	0.008	0.50	0.486	5.23	0.701

5. Conclusion

The quality of the distribution and efficiency of the stimulus generation are dual challenges in the verification flow. We proposed a self-tuning generation framework with affinity grouping and the greedy search strategy for the constrained random simulation. The input variable is grouped according to the constraints affinity in order to be handled by the greedy search for the global optimization problem. The experimental results show that our methods described in this paper can provide a stimulus generation with good performance and high uniform distribution.

In the future, we will apply the algorithm to the more practical cases and focus on the research of adding feedback from the coverage analysis in order to dynamically gain a desired distribution.

Acknowledgement

This work was funded in part by the National Grand Fundamental Research 973 Program of China under Grant No.2005CB321605 and the National Natural Science Foundation of China under Grant No.60876030 and the project 60720106003.

References

[1] "Constrained-random test generation and functional coverage with Vera", Technical report, Synopsys, Inc, Feb, 2003.

[2] J. Yuan, K. Shultz, C. Pixley, H. Miller, and A. Aziz, "Modeling design constraints and biasing in simulation using BDDs," in Digest Tech. Papers IEEE/ACM Int'l Conf. Computer-Aided Design, pp. 584-589, Nov. 1999.

[3] J. H. Kukula and T. R. Shiple, "Building circuits from relations," in Proc. 12th Int'l Conf. Computer-Aided Verif. (CAV), pp. 113-123, Springer-Verlag, 2000.

[4] Marques Silva J P, Sakallah K A. GRASP:A search algorithm for propositional satisfiability. IEEE Transactions on Computers, 1999, 48 (5) : 506 521

[5] M.W.Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in Proc. 38th ACM/IEEE Design Automation Conf., pp. 530-535, June 2001.

[6] N.Een and N.Sorensson, "An extensible SAT-solver," in Proc. 6th Int'l Conf. Theory & Appl. Satisfiability Testing (SAT), pp.502-518, May 2003.

[7] N. Kitchen, A. Kuehlmann, "Stimulus Generation for Constrained Random Simulation," IEEEACM International Conference on Computer Aided Design (ICCAD), pp. 258-265, 2007.

[8] W. Jordan, "Towards efficient sampling: exploiting random walk strategies", AAAI, pp. 670-676, 2004.

[9] ISCAS89 Sequential Benchmark Circuits: http://www.ece.vt.edu/mhsiao/iscas89.html

[10] Edmund Clarke, Armin Biere, Richard Raimi and Yunshan Zhu, "Bounded Model Checking Using Satisfiability Solving," Formal Methods in System Design, vol. 19, No. 1, pp. 7-34, 2001.

[11] C. P. Gomes, W. Hoeve, A. Sabharwal, B. Selman, "Counting CSP Solutions Using Generalized XOR Constraints," the 22nd Conference on Artificial Intelligence, pp.204–209, July 2007.

.