Ninth International Workshop on Microprocessor Test and Verification

Mining Unreachable Cross-timeframe State-pairs for Bounded Sequential Equivalence Checking

Lynn C.-L. Chang and Charles H.-P. Wen
Dept. of Communication Engineering
National Chiao Tung University
Hsinchu, Taiwan 300
Email: tinger.cm95g@nctu.edu.tw and opwen@g2.nctu.edu.tw

Abstract

One common practice of checking equivalence for two sequential circuits often limits the timeframe expansion to a fixed number, and is known as bounded sequential equivalence checking (BSEC). Although the recent advances of Boolean satisfiability (SAT) solvers make combinational equivalence checking scalable for large designs, solving BSEC problems by SAT remains computationally inefficient. Therefore, this paper proposes a 3-stage method to exploit constraints to facilitate SAT solving for BSEC. The candidate set are first accumulated by checking each composition of functions derived by a data-mining algorithm for every two cross-timeframe flip-flop states. Each candidate can be further removed if it matches simulation data in history and its validity is finally confirmed through gate-level netlist. The verified set is feedbacked as constraints in SAT solving for the original BSEC problem. Experimental results show a 40X speedup in average on ISCAS 89 circuits.

1. Introduction

Functional verification in VLSI includes an important problem of determining equivalence for two circuit descriptions during the design process. Such a problem can be further divided into: *combinational equivalence checking* (BEC) and *sequential equivalence checking* (SEC). Due to the recent advances of Boolean satisfiability (SAT) solvers, such as Zchaff[1], MiniSAT[2], C-SAT[3], SAT-based method enables scalable and robust combinational equivalence checking for large VLSI designs. However, with increasing design complexity, the general problem of checking the equivalence for two sequential circuits remains intractable. Therefore, in sequential equivalence checking, one common practice constrains the timeframe expansion to a limited number and is as known as *bounded sequential equivalence checking* (BSEC).

Constraint extraction is a technique, which has been successfully applied in various electronic design automation (EDA) problems, such as logic optimization and automatic test pattern generation (ATPG). Authors in [4][5] propose an approach of finding internal don't-care states as constraints and merging them according to observability for Boolean network optimization. Static and dynamic learning techniques are applied in [6][7] to guide pair-wise implications to assist test pattern generation.

Many recent studies also incorporate constraint extraction techniques for BSEC problems. Authors in [8] use binary decision diagram (BDD) techniques to estimate the reachability of states and derive the don't care set. Association rule mining and logic implication are combined in [9][12] to derive 3-node relations among all internal nodes to accelerate SAT solving for BSEC problems.

Most of previous works focus on the relationship of internal signals at one single timeframe. However, in [11], the primary outputs and register inputs are shown to have greater impact than internal signals, and in [9][12], constraints across multiple timeframes are effective for speeding up SAT solving. Therefore, to avoid effort on internal signals over lengthy timeframes, we propose a method as shown in Figure 1 to only explore the relations of state-pairs across 2 timeframes and to derive a set of constraints for SAT solving of BSEC problems. Furthermore, although the total number of flip-flops is much smaller than that of internal signals, exhaustive analysis of checking all state-pairs is inefficient in time. Thus, multiple filtering strategies are incorporated in our proposed method to reduce the number of state-pairs. They are (1) *functionality filtering*, (2) *historical filtering* and (3) *structural filtering*.

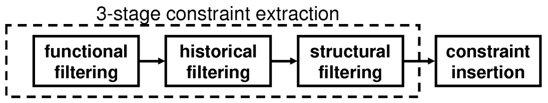

Figure 1. Proposed method with constraint extraction

In stage 1, a data-mining algorithm statistically analyzes the I/O (including flip-flops) data that can be given or from random simulation and derives the approximate functions for each flip-flop state at one specific timeframes. A pair of cross-timeframe flip-flops is said a unreachable state-pair candidate which can be computed by the conjunction of two Boolean functions for each flip-flop states. Once the conjunctive Boolean function is not empty, it can be removed from the candidate list. During the stage 2, *historical filtering* checks each candidate against the simulation data to prune false cases. Finally, the validity of unreachable state-pairs in the candidate set is confirmed by *structural filtering* where each candidate will be realized into one constraint gate and appended to the unrolled 2-timeframe miter to form an augmented circuit. If the augmented circuit is UNSAT, such a constraint is proven to be true.

After 3-stage filtering, the final set of unreachable state-pairs will be inserted as constraints for SAT solving of the original BSEC problem. Our experimental results show that the 3-stage filtering can reduce the total number of unreachable state-pair candidates sifinificantly and derive cross-timeframe constraints efficiently. The proposed method also reduces the runtime used by state-of-art SAT solvers remarkably on most ISCAS 89 circuits for BSEC problems.

The rest of the paper is organized as: Section 2 formulates the problem of bounded sequential equivalence checking. Section 3 proposes our method and discusses the techniques for extracting cross-timeframe constraints in detail. Section 4 presents the experimental results and demonstrates the usefulness of the extracted constraints in SAT solving for BSEC problems. Section 5 concludes the paper.

2. bounded sequential equivalence checking

Typically, the problem of sequential equivalence checking (SEC) can be formulated as checking the output of the miter circuit, which is composed of two finite-state machines (FSMs), over multiple timeframes. Bounded sequential equivalence checking, however, is a special case of SEC problems and limits the timeframes to be checked to a fixed number. Therefore, modeling of a BSEC problem as illustrated in Figure 2 consists of two steps: *miter construction* and *timeframe unrolling*.

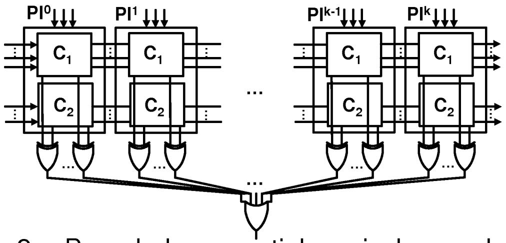

Figure 2. Bounded sequential equivalence checking (BSEC) model

The miter is constructed by connecting every pair of two corresponding outputs from FSMs by one XOR gate. The miter is then unrolled to a limited number of timeframes, say k, and forms the BSEC model. The combinational logic is duplicated into k copies and outputs of flip-flops in one timeframe are connected to corresponding inputs in next timeframe. After unrolling miter circuit, one big OR gate will connect the output of XOR gates from each timeframe. Finally, the output of this OR gate will be set as 1. If the BSEC model is UNSAT, then these two circuits are proven to be equivalent over a bounded number of timeframes. Otherwise, they are in-equivalent.

3. proposed method with constraint extraction

Since previous studies [8][9] that explore the constraints among internal nodes for SAT solving may suffer from a large number of constraint candidates, the proposed method instead considers cross-timeframe state-pairs as candidates and prunes the false cases on the basis of simulation data and the gate-level netlist of the circuit.

Since each state-pair can be validated by running SAT solving on the BSEC model, one intuitive method is to enumerate all combinations of state-pairs for checking. However, given n and k are the numbers of flip-flops and the number of timeframe unrolling, respectively, the combinations for state-pairs will go up to $4 \times C_2^{nk}$, where 4 represents different cases of state-pairs including $\{00\}, \{01\}, \{10\}$, and $\{11\}$. Running SAT solving for $4 \times C_2^{nk}$ times will be prohibitively time-consuming and even worse than solving the BSEC model directly. Therefore, a 3-stage constraint extraction shown in Figure 1 integrates multiple filtering strategies to help reduce the total number of state-pairs.

The first stage is *functional filtering*. A data-mining algorithm called the **support-confidence** framework is developed to construct the approximate Boolean functions for each flip-flop state at one specific timeframe by learning the simulation data. Then, the cross-timeframe state-pair could be a constraint candidate if the conjunction of Boolean functions for two such flip-flop states is empty. *Historical filtering* in the second stage scans through the simulation data to prune the rare cases escaped from approximate functional learned in the first stage. The final stage is *structural filtering* which validates the candidate through SAT solving of the augmented miter circuit. Note that *functional filtering* plays an important role in the proposed method and needs generating as few candidates as possible to make the *historical filtering* and *structural filtering* efficient in time.

The details of the proposed method, including 3-stage constraint extraction, will be elaborated as follows.

3.1. functional filtering

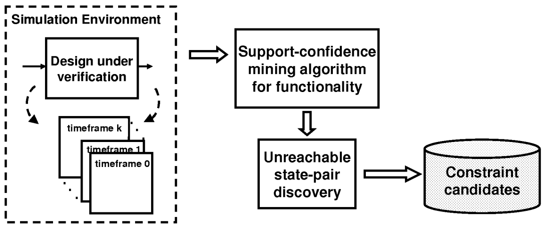
Simulation Environment
Design under verification
timeframe k
timeframe 1
timeframe 0
Support-confidence mining algorithm for functionality
Unreachable state-pair discovery
Constraint candidates

Figure 3. Flow of functional filtering

The flow of *functional filtering* is illustrated in Figure 3. First, two learning databases are constructed by collecting simulation data across timeframes as shown in Figure 4. For example, assume that the number of timeframe unrolling is 4. 1-timeframe database retrieves I/O data from each single timeframe. 2-timeframe database collects input data from every two consecutive timeframes but collects output data only from the latter timeframe. A support-confidence mining algorithm will be applied to extract functional information based on these two databases and for each filp-flop state, iteratively find the most frequent Boolean cubes, particularly termed *ruling cubes*.

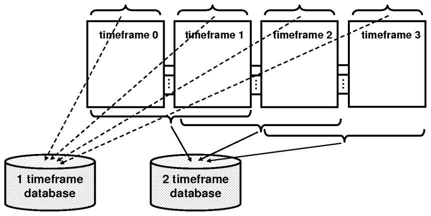

Figure 4. Data collection from simulation

3.1.1. ruling cube generation. Each Boolean function of one flip-flop state can be approximated by a set of Boolean cubes. For example, in Figure 5, $\{t_i\}$ denotes the test set in simulation database M and $\{c_i\}$ denotes the set of Boolean cubes to be evaluated. For each cube c_i, two corresponding metrics, support and confidence (denoted as sup_i and $conf_i$), are used to quantify the importance of such a cube. If both sup_i and $conf_i$ are larger than the threshold values, γ_{sup} and γ_{conf}, respectively, then the cube c_i is a *ruling cube* and can be used to construct the approximate Boolean function F^* later. In contrast, those Boolean cubes that fail to satisfy the support and confidence criteria will be dropped in F^*.

For the support-confidence framework [15], the support sup and the confidence $conf$ denote the frequency and the accuracy for one Boolean cube, respectively, on the basis of the database M, and their formal definitions are given as follows.

$$sup = \frac{|X|}{|M|} \text{ and } conf = \frac{|X \cap Y|}{|X|}$$

where M denotes the set of total tests in the database, X denotes the set of tests covered by one Boolean cube, Y denotes the set of tests with the target output response (either 0 or 1) in M, and $|.|$ denotes the size of one set.

	$x_1x_2x_3x_4x_5x_6$	y				
t_1	0 1 1 1 1 0	1	c_1	x x x x 1 0	***sup*** **= 3/5**	***conf*** **= 2/3**
t_2	1 0 1 0 0 1	1				
t_3	0 1 1 0 0 1	0	c_2	x 1 0 x x x	***sup*** **= 3/5**	***conf*** **= 2/3**
t_4	1 1 1 0 1 0	1				
t_5	0 0 0 0 1 0	0	c_3	x 1 x x 1 x	***sup*** **= 2/5**	***conf*** **= 2/2**

Figure 5. Example of support-confidence computation

Figure 5 shows an example for support-confidence computation. Given the database $M = \{t_1, ..., t_5\}$, the cube c_1 satisfies t_1, t_4 and t_5 and sup_1 is $\frac{3}{5}$. However, since only t_1 and t_4 have the target output response ($y = 1$), $conf_1$ is $\frac{2}{3}$. sup_i and $conf_i$ of any other cube i can be computed in the same manner. Moreover, suppose that γ_{sup} and γ_{conf} are 0.05 and 0.95, only c_3 satsifies the support and confidence criteia among three cubes and can be a ruling cube on the basis of M in this example.

Support-confidence mining algorithm is proposed to derive the set of ruling cubes for constructing the approximate Boolean function for each flip-flip state. In Figure 5, c_3 = x1xx1x represents one ruling cube x_2x_5 where x_2 and x_5 are *support literals* which represent the most important variable states in such a ruling cube. One ruling cube is geenrated by adding the support literal one by one until no futher support literal can be found.

According to [14], the impact of one variable state can be achieved by comparing the impurity difference between the original database M and the new M_v split with respect to one variable state v. Therefore, the gain value, $g(.)$, one popular impurity measure, is typically employed to gauge such an impact. For two literals (x and $\overline{x}$) of one input variable and the database M, $g(x)$ and $g(\overline{x})$ can be formulated as

$$g(x) \equiv \frac{n_{11}}{n_{10} + n_{11} + 1} \text{ and } g(\bar{x}) \equiv \frac{n_{01}}{n_{00} + n_{01} + 1}$$

where n_{11} is the number of tests with input variable $x = 1$ and output response $y = 1$, n_{10} is the number of tests with $x = 1$ and $y = 0$, n_{01} is the number of tests with $x = 0$ and $y = 1$, and n_{00} is the number of tests with $x = 0$ and $y = 0$.

Note that $g(x)$ represents the ratio of the number of tests with $x = 1$ and $y = 1$ to the number of all tests with $x = 1$ in M; $g(\overline{x})$ can be understood similarly. After the gain values of all variable states are computed, the variable state with maximum gain will be selected as the next support literal.

M

$x_1x_2x_3$	f
0 0 0	0
0 0 1	0
0 1 0	0
0 1 1	0
1 0 0	1
1 0 1	0
1 1 0	1
1 1 1	0

	x_1	x_2	x_3
n_{00}	4	3	4
n_{01}	0	1	0
n_{10}	2	3	2
n_{11}	2	1	2
$g(x)$	2/5	1/5	2/5
$g(\overline{x})$	0	1/5	0

(a)

$M_{x_1=1}$

x_2x_3	f
0 0	1
0 1	0
1 0	1
1 1	0

	x_2	x_3
n_{00}	1	0
n_{01}	1	2
n_{10}	1	2
n_{11}	1	0
$g(x)$	1/3	0
$g(\overline{x})$	1/3	2/3

(b)

Figure 6. Example for generating one ruling cube

Figure 6(a) illustrates the process for selecting the first support literal. The original database M has three inputs, x_1, x_2, and x_3. The values of $\{n_{00}, n_{01}, n_{10}, n_{11}\}$ for each variable state is first computed. For example, $\{n_{00}, n_{01}, n_{10}, n_{11}\}$ for x_1 is $\{4, 0, 2, 2\}$, and thus, $g(x_1) = \frac{2}{2+2+1}$ and $g(\overline{x_1}) = \frac{0}{4+0+1}$. $g(x_2)$, $g(\overline{x_2})$, $g(x_3)$ and $g(\overline{x_3})$ can be computed similarly. After all gain values are available, the variable state with the maximum gain is selected as the support literal. If two variable states have the maximum gain, the support literal can be selected arbitrarily. In our example, both x_1 and x_3 have the maximum gain $\frac{2}{5}$, and

x_1 is chosen arbitrarily to be the first support literal for the ruling cube c.

Given $\gamma_{sup} = 0.05$ and $\gamma_{conf} = 0.95$, for the current rule cube $c = x_1$, $sup_c = \frac{|M_{x_1=1}|}{|M|}$ is $\frac{4}{8}$ and $conf_c = \frac{|M_{x_1=1 \cap y_1=1}|}{|M_{x_1=1}|}$ is $\frac{2}{4}$. Since the current $conf_c$ is much lower than γ_{conf}, the ruling cube generation will continue to find the next support literal as shown in Figure 6(b). Note that the database M now becomes $M_{x_1=1}$ since the next support literal needs to be selected on the basis of all tests with $x_1 = 1$ in M.

Once the extracted Boolean cube c meets $sup_c \geq \gamma_{sup}$ and $conf_c \geq \gamma_{conf}$, it will be accumulated in the set of ruling cubes for constructing the approximate function of one flip-flop state later. However, if no other variable state can be selected and the current cube fails to meet the support and confidence criteria, the cube will be dropped. To avoid processing the same cubes, both the tests covered by ruling cubes and dropped cubes will be removed from the database.

Algorithm 1 shows the overall algorithm to construct the approximate function for one flip-flop state. Given database M, N is the maximum number of support literals in one ruling cube since the maximum number of literal to split database M is $log_2|M|$. F^* is the target function to be extracted and D is the set of current tests covered by F^*. The algorithm starts from constructing a Boolean cube representing a sub-function f by adding one variable state one at a time. $SupportVariableSelect()$ is applied to select the next suuport literal x under f. When both the frequency f_{sup} and the accuracy f_{conf} can meet the criteria, f is updated by conjuncting itself with x. The algorithm keeps finding the next support literal to update f until the current cube f has met the ruling cube criteia in line 9 or included more than N variable states in line 13. F^* continues accumulating ruling cube $f's$ for one flip-flop state until F^* covers a percentage γ_{cov} of the total tests in the database M.

Algorithm 1 MineFun(): Mining Function from Data

```
1:  $N = log_2|M|$;
2:  $F^* \leftarrow \emptyset$;
3:  while ($|D| \leq |M| \times \gamma_{cov}$)
4:    $f = 1$;
5:    do {
6:      $x$ = SupportVariableSelect($M, f$);
7:      $f \leftarrow f \cap x$;
8:      update($f_{sup}$,$f_{conf}$); // update $f_{sup}$ and $f_{conf}$
9:      if ($f_{sup} \geq \gamma_{sup}$ && $f_{conf} \geq \gamma_{conf}$)
10:       $F^* \leftarrow F^* \cup f$;
11:       update($D$); // update by ruling cube
12:       break;
13:   } while ($|f| < N$);
14:   if ($f_{sup} < \gamma_{sup}$ && $f_{conf} < \gamma_{conf}$)
15:     update($D$); // update by dropped cube
```

3.1.2. unreachable state-pairs discovery. unreachable state-pair discovery next conjuncts two functions for each combination of state-pairs followed by performing SAT solving on the conjunctive Boolean function. For example, given FF_p^i and FF_q^j as the functions for the states of flip-flop p at timeframe i and of flip-flop q at timeframe j, respectively, If $f = FF_p^i \cap FF_q^j$ is UNSAT, there exists no input test which can satisfy both flip-flop states concurrently. Therefore, (FF_p^i, FF_q^j) is a constraint candidate. Note that, for each flip-flop p at timeframe k, the support-confidence mining algorithm will run twice: one for ON state FF_p^k, and the other for OFF state $\overline{FF}_p^k$.

3.2. historical filtering

After generating the initial set of cross-timeframe state-pairs for constraint candidates, *historical filtering* prunes those pairs that have already been seen in simulation data. For example, given $(\overline{FF}_p^i, FF_q^{i+1})$ as the constraint candidate to be checked, if flip-flop p in some timeframe k has the state value of 0 and flip-flop q in some timeframe $k+1$ has the state value of 1, then $(\overline{FF}_p^i, FF_q^{i+1})$ will be removed from the candidate set.

3.3. structural filtering

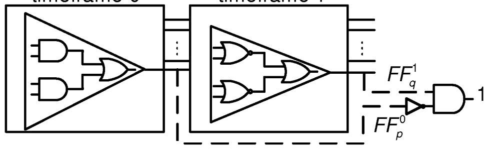

Figure 7. Illustration for structural filtering

At this stage, *structural filtering* is to ensure the validity of each candidate under the unrolled 2-timeframe miter. For example, if $(\overline{FF}_p^i, FF_q^{i+1})$ is one of remaining constraints, the *inverted* output of flip-flop p at timeframe 0 and the output of flip-flop q at timeframe 1 are connected by an additional AND gate. Such an example is illustrated in Figure 7. Next, SAT solving is performed on the unrolled 2-timeframe miter with enforcing 1 on the output of the additional AND gate. If the result is UNSAT, $(\overline{FF}_p^i, FF_q^{i+1})$ is a true constraint; otherwise, $(\overline{FF}_p^i, FF_q^{i+1})$ should be removed.

Since *structural filtering* requires SAT solving, it is most time-consuming stage in the constraint extraction. Therefore, an upper bound to the number of constraints that can be extracted is imposed considering the performance in time.

3.4. constraint insertion

Constraint insertion is the final step in the proposed method. Given k as the number of timeframe expansion in BSEC problems, each extracted constraint will be translated into multiple CNF clauses over k timeframes and appended to the CNF of the original BSEC model. For example, if (FF_3^i, FF_5^{i+1}) is one proven constraint, CNF clauses $(FF_3^0, FF_5^1)(FF_3^1, FF_5^2)...(FF_3^{k-1}, FF_5^k)$ will be appended to the original CNF for final SAT solving.

4. experimental results

The proposed method is implemented in C++. The experiments are run on Linux equipped with a 2.4GHz CPU and 2GB RAM. ISCAS 89 circuits are used as benchmarks for bounded sequential equivalence checking. Each circuit is re-synthesized by Design Complier from Synopsys. Zchaff [1] is applied for SAT solving. The default number of tests for simulation ranges from $1,000$ to $5,000$ depending on the number of primary inputs of the benchmark circuits. γ_{sup} and γ_{conf} are by default 0.05 and 0.95, respectively. The number of upper bound for constraint state-pairs to be inserted is $2,000$.

miter	# of PI	# of PO	# of FF in miter	# of k timeframes	# of FF $\times$ k-timframes
s298	3	6	28	40	1120
s349	9	11	30	40	1200
s713	35	23	36	30	1080
s832	18	19	10	30	300
s1196	14	14	36	30	1080
s1488	8	19	12	30	360
s4863	49	16	169	15	3035
s15850	77	150	1040	15	15600
s35932	35	320	3456	10	34560
s38584	38	304	2690	10	26900

Table 1. Characteristics of BSEC models for ISCAS 89 circuits

Table 1 shows the characteristics of BSEC models for ISCAS 89 circuits used in the experiments. # of PI and # of PO are the numbers of primary inputs and primary outputs for each circuit, respectively. # of FF is the number of the flip-flops in the original miter. k is # of timeframes to be unrolled in the BSEC model. # of FF $\times$ k-timeframe denotes the total numbers of the flip-flops in the BSEC model.

Table 2 demonstrates the effectiveness of 3-stage filtering by reporting the numbers of constraint candidates across 2 timeframes after each filtering. Column 1 lists the name of the benchmark circuits while column 2 represents the initial number of candidates across 2 timeframes. Column 3, 4 and 5 denote the numbers of candidates after *functional*, *historical* and *structural filterings*, respectively. Column 6 reports total runtime used for overall 3-stage constraint extraction.

Table 3 shows the improvement of SAT solving for BSEC problems. Column 1 lists the name of the benchmark circuits and column 2 is the # of unrolled timeframes. Column 3, 4 and 5 denote the runtime of SAT solving without constraint, runtime for SAT solving with constraints, the combined runtime of mining and SAT solving, respectively. Column 6 reports the speedups computed by the original runtime in Column 3 divided by the new runtime in Column 5.

miter	# of constraint across 2 timeframes				time (s)
	initial	functional filtering	historical filtering	structural filtering	
s298	6160	2139	1606	32	9.13
s349	7080	945	671	212	6.19
s713	10224	2929	2201	6	43.77
s832	760	84	68	32	5.69
s1196	10224	610	355	60	24.2
s1488	1141	409	298	74	16.02
s4863	227812	4081	1209	668	231.67
s15850	8648640	115429	4939	2000	1365.83
s35932	95537664	84169	2320	2000	1218.34
s38584	57878040	288373	3134	2000	1201.60

Table 2. Comparison of numbers of constraint candidates

miter	k time frames	origin(s)	new(s)	mining +new(s)	speedup
s298	40	30.39	0.12	9.25	3.29
s349	40	21.88	0.2	6.39	3.42
s713	30	346.48	56.78	100.55	3.45
s832	30	2028.65	0.72	6.41	316.48
s1196	30	96.19	55.78	79.98	1.20
s1488	30	754.03	5.68	21.70	34.75
s4863	15	7725.44	4.35	236.02	32.73
s15850	15	64860	227.63	1593.46	40.71
s35932	10	51744.3	173.78	1397.12	37.04
s38584	10	50464.3	19.50	1221.10	41.33

Table 3. Runtime for BSEC problems

Our experimental results show different speedups on benchmark circuits with an average as 40X. Significant improvement can be observed on the big circuits while minor improvement can be observed on the small circuits.

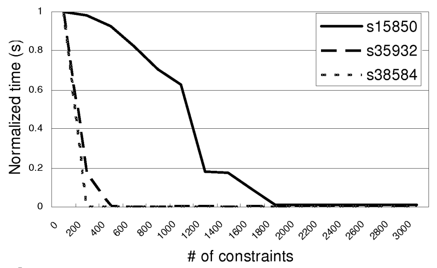

Figure 8. SAT solving time with different # of constraints

We further investigate the relations between the number of constraints and runtime for SAT solving on three big ISCAS 89 circuits. Figure 8 shows the result where Y-axis represents the rurntime for new SAT solving normalized to the original runtime used by SAT solving without any constraint. Obviously, s35932 and s38584 converge fast and only require 500 constraints while s15850 may require 1900 constraints to converge. However, since not each constraint has same contribution to SAT solving, the efficiency of solving BSEC may depend on the quality of constraints, not the number of constraints. Therefore, how to select enough good constraints to fast converge SAT solving is worth investigation and can be a topic for future research.

5. conclusions

The general problem of checking functional equivalence for two sequential circuit is still far from being solved. In this paper, we proposed a method which integrates data mining, simulation and structural analysis techniques to extract unreachable cross-timeframe state-pairs as constraints to facilitate SAT solving for bounded sequential equivalence checking (BSEC) problems. Experimental results shows that the 3-stage filtering can derive the set of unreachable cross-timeframe state-pairs efficiently. SAT solving with the extracted constraints can speed up 3X to 300X on most ISCAS 89 circuits. Future works include the quality analysis of the extracted constraints and a better strategy to exploit constraints efficiently.

References

[1] M.H. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang and S. Malik, ”ZChaff: Engineering an Efficient SAT Solver,” in Proc. Design Automation Conf. (DAC), pp. 530-535, 2001.

[2] N. Een and N. Sorensson, ”An Extensible SAT-Solver,” Theory and Applications of Satisfiability Testing, pp. 502-518, 2003.

[3] F. Lu, L. C. Wang and K. T. Cheng, ”A Circuit SAT Solver with Signal Correlation Guided Learning.” in Proc. Conf. Design, Automation and Test in Europe (DATE), pp. 892-897, 2003.

[4] H. Ichihara and K. Kinoshita, ”On Acceleration of Logic Circuit Optimization using Implication Relations” in Proc. Asian Test Symp. (ATS), pp.222-227, 1997.

[5] W. Kunz, D. Stoffel and P.R. Pradhan, ”Logic Optimization and Equivalence Checking by Implication Analysis”, in IEEE Trans. CAD (TCAD), vol. 15, No. 5, pp. 266-281, 1993.

[6] M.H. Schulz, E. Trischler and T.M. Sarfret, ”SOCRATES: A Highly Efficient Automatic Test Pattern Generation System,” in IEEE Trans. CAD (TCAD), vol. 7, No. 1, pp. 126-137, 1988.

[7] W. Kunz and P.R. Pradhan, ”Accelerated Dynamic Learning for Test Pattern Generation in Combinational Circuits,” in IEEE Trans. CAD (TCAD), vol. 12, No. 5, pp. 684-694, 1993.

[8] S. Safarpour, G. Fey, A. Veneris, and R. Drechsler, ”Utilizing Don’t Care States in SAT-based Bounded Sequential Problems,” in Proc. VLSI Great Lakes Symposium, pp. 264-269, 2005.

[9] W. Wu and M. S. Hsiao. ”Mining Global Constraints for Improving Bounded Sequential Equivalence Checking,” in Proc. Design Automation Conf. (DAC), pp. 743-748, 2006

[10] A. Mishchenko and R. K. Brayton, ”SAT-Based Complete Don’t-Care Computation for Network Optimization,” in Proc. Conf. Design, Automation and Test in Europe (DATE), pp. 412-417, 2005.

[11] M. L. Case, V. N. Kravets, A. Mishchenko and R. K. Brayton, ”Merging Nodes Under Sequential Observability,” in Proc. Design Automation Cconf. (DAC), pp. 540-545, 2008.

[12] W. Wu and M. S. Hsiao. ”Mining Global Constraints with Domain Knowledge for Improving Bounded Sequential Equivalence Checking,” in IEEE Trans. CAD (TCAD), vol. 27, No.1, pp. 197-201, Jan. 2006

[13] O. Guzey, L. C. Wang and J. Bhadra, ”Enhancing signal controllability in functional test-benches through automatic constraint extraction,” in Proc. Int’l Test Conf. (ITC), pp. 1-10, Oct, 2007

[14] H. P. Wen, L. C. Wang and J. Bhadra, ”An Incremental Learning Framework for Estimating Signal Controllability in Unit-Level Verification,” in Proc. Int’l Conf. Computer Aided Design (ICCAD), pp. 250-257, 2007.

[15] R. Agrawal, T. Imielinski and Swami AN, ”Mining Association Rules between Sets of Items in Large Databases,” In Proc. of the ACM SIGMOD Int. Conf. on Management of data, Jun. 1993.