

國立交通大學

電信工程學系

碩士論文

應用資料探勘的有界時序電路等價驗證之
加速方法設計

Speeding up Bounded Sequential Equivalence
Checking with Data Mining

研究生：張佳伶

指導教授：溫宏斌 教授

中華民國九十八年六月

應用資料探勘的有界時序電路等價驗證之加速方法設計
Speeding up Bounded Sequential Equivalence Checking
with Data Mining

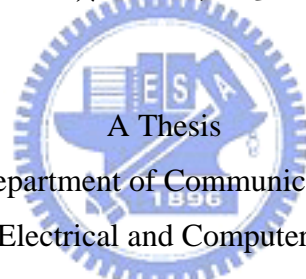
研究生：張佳伶

Student : Chia-Ling Chang

指導教授：溫宏斌

Advisor : Hung-Pin Wen

國立交通大學
電信工程系
碩士論文



Submitted to Department of Communication Engineering
College of Electrical and Computer Engineering
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in

Communication Engineering

June 2009

Hsinchu, Taiwan, Republic of China

中華民國九十八年六月

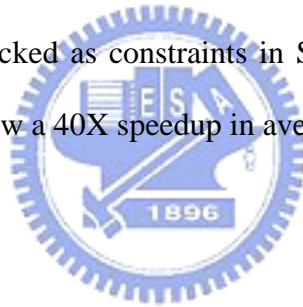
摘 要

為了確認二個不同版本的時序電路是否有相同的功能，最常見的方法是將電路展開到限定的數量進行驗證，此方法稱為有界時序等價驗證。雖然布林可滿足性解答器的大幅進展已經使得組合電路的等價驗證上，可以應用至大型的電路，但其解答器在解決時序電路或有界時序電路的問題時仍然非常沒有效率。因此，本論文提出一個利用三階段的開發方法尋找電路的限制點，如此可以加速布林可滿足性解答器在有界時序電路上的應用。這些限制點主要由時序電路上的正反器組合而成。首先，我們會利用資料探勘的方法得到每個正反器的近似函數，再由這些函數的組合找出不會同時發生的狀態組合，此則稱為限制點。被找出的限制點會再被確認是否與模擬的資料相符。最後，有界時序電路會針對這些限制點逐一驗證，通過驗證的限制點，才是有界時序電路上真實的限制點。完成三階段尋找限制點的流程之後，所有的限制點會再被加回有界時序電路中，如此可以加速布林可滿足性解答器的解答過程。實驗結果證明，對於 ISCAS89 電路的有界時序驗證可以達到平均 40 倍的加速。



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.



誌 謝

本論文得以順利完成，首先要感謝的是指導教授溫宏斌老師。感謝老師在我茫然無所措的時候，擔任我的指導老師。這二年來，老師在專業領域上悉心的指導、分享待人處事的道理，都使得我不論在研究、個性方面都有相當的成長。很高興可以成為老師的學生，在未來的研究路上定加倍努力，勉勵自己可以達到老師的要求。

另外，要感謝的是趙學永老師。雖然仍然未能在老師的實驗室中畢業，但是一年來老師的指導、分享學習的經驗，更是使我獲益良多。而更要感謝 HFEDA 實驗室的昱靜學姐、益廷學長、當榮學長，給予我支持與鼓勵，讓我在懵懂中逐漸成長。



接著要感謝 CIA 實驗室的成員振源、怡璋、雨欣、千慧、韋廷、彥后、南旭，謝謝你們一路的陪伴與分享，是我在研究的路途上很好的伙伴。同時也要感謝資科工所 BSP 實驗室的朋友們，令瑋、詠成、士暉，詠恬、姿樺、開印、煙玉，乙慈、淵耀、郁萱、亮維，永煌、筱苑、柏志，謝謝你們的勉勵與抵勵，讓我的碩士生涯多了許多豐富色彩，同時也是努力的目標。最後要感謝摯友慧伶，哲萱，思吟，給予我最多的鼓勵以及最大的包容，支撐我完成碩士學業。

最後僅以此文獻給我摯愛的父母及弟弟。

Contents

List of Figures	vi
List of Tables	vii
1 Introduction	1
1.1 Functional Verification	2
1.2 Thesis Scope	5
1.3 Thesis Organization	5
2 Background	6
2.1 Bounded Sequential Equivalence Checking	7
2.2 Boolean Satisfiability	9
2.3 Data Mining	11
2.3.1 Association Rule Mining	13
2.3.2 Support-confidence Framework	14
3 Learning Framework	17
3.1 Training Data Collection	18
3.2 Learning Relaxed Boolean Function	18
3.2.1 Support-confidence Method	18
3.2.2 Impurity Measure	20
4 Constraint Extraction Method	24
4.1 Constraints in BSEC Problem	25
4.2 3-stage Filtering Method	28
4.2.1 Functional Filtering	29
4.2.2 Historical Filtering	30
4.2.3 Structural Filtering	30
4.2.4 Constraint insertion	31
5 Experimental Results	32

6 Conclusion

41

Bibliography

43

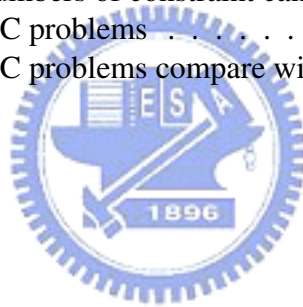


List of Figures

1.1	Typical design flow overview [1]	2
2.1	Bounded sequential equivalence checking (BSEC) model	7
2.2	A sample circuit	10
2.3	Example of ruling cubes	15
3.1	Flow of random simulation	18
3.2	Example of support-confidence learning	20
3.3	Example for generating one ruling cube	21
4.1	An example to illustrate constraint in SAT solving	25
4.2	A learning-and-filtering framework for BSEC	29
4.3	Illustration for structural filtering	30
5.1	Speedups of 4 large circuits with respect to 10 configurations	38
5.2	Bound for SAT solving of BSEC problems with and without constraints	39
5.3	SAT solving time with different # of constraints	40

List of Tables

2.1	Conjunctive normal form of some basic logic gates	9
3.1	Partitioning data into databases	19
4.1	Clauses for Figure 4.1	26
5.1	Characteristics of BSEC models for benchmark circuits	33
5.2	Comparison of numbers of constraint candidates	34
5.3	Runtime for BSEC problems	35
5.4	Runtime for BSEC problems compare with [20]	36



Chapter 1

Introduction



1.1 Functional Verification

In integrated circuit (IC) design flow, shown in Figure 1.1, functional verification involves to different levels of achievement. Since the increasing of circuit size and complexity affect the time-to-market of chip, functional verification has become one of important bottlenecks in the design process. In most of the industrial designs, more than 70% of the effort is spent on functional verification. Although it is difficult to detect all design problems in circuit, fixing the design errors earlier in design cycle or higher level of implementation is first target. If there is an undetected error that remains in circuit after manufacturing, the loss of cost is uncountable for a company. For example, Intel paid \$475 million to recall the floating point division math bugs in Pentium processor.

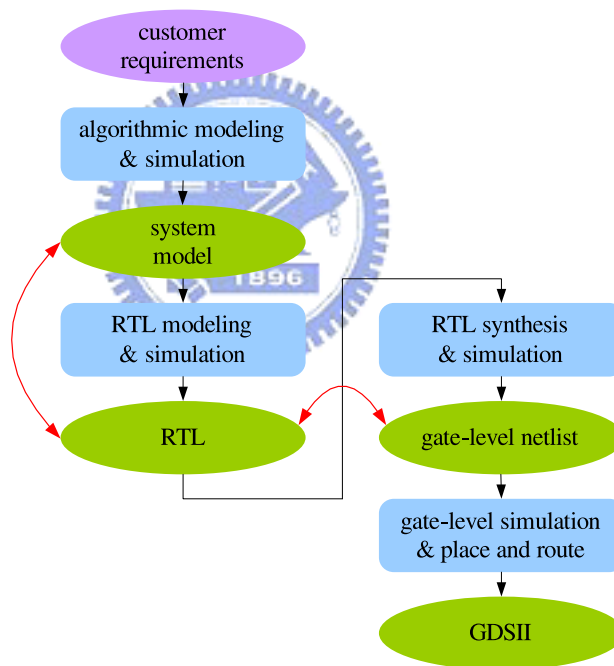


Figure 1.1: Typical design flow overview [1]

There are two major approaches in functional verification: formal approach and simulation approach. In formal verification, mathematical method is used to prove the correctness of circuit. Formal verification could discover all the design problems by analyzing the

circuit model. However, formal approach applicability in practice is limited due to the increase circuit size and complexity. On the other hand, simulation-based verification can be scalable to larger circuit and it is the most common way to verify the designs in industrial cases. In simulation approach, it simulates the inputs patterns and then collects the output responses to analyze the circuit behavior. If all possible input patterns are simulated, it is easy to discover the corner cases in the design. However, if there are n inputs in circuit, the number of possible test patterns is 2^n . While n is large, it is impossible to simulate such amount of test patterns in a reasonable time. With large and complex circuits, simulation-based verification has become ineffective to finding bugs.

In functional verification, formal-based verification can find all bugs but it is impractical in modern design, and simulation-based verification is scalable and practical but can not discover all bugs in circuit. Therefore, to complement strengths and weaknesses in formal and simulation techniques effectively, hybrid verification provides an immediate practical solution to overcome the verification problem. Hybrid verification is an approach which combines at least two techniques in formal and simulation verification method.

In this thesis, hybrid verification method is used to solve the bounded sequential equivalence checking (BSEC) [2] problem. In functional verification, equivalence checking involves to ensure that the current implementation of a design is functionally equivalent to an earlier version or the specification. Since solving general problem in equivalence checking is difficult, the target in this thesis is to solve the sequential circuit within a specific time-frame, which is called bounded sequential equivalence checking. Our hybrid verification method combines formal technique and constraint extraction from simulation. In general, the BSEC problem will be formulated as a Boolean satisfiable problem and solved by formal engines. However, the solving process for BSEC problem is still ineffective due to the increased size of circuits. To speedup the solving process, conflict constraints in BSEC model are explored to help the formal engines. The following will introduce the formal engine and constraint extraction method we used, respectively.

The most two popular of formal engines are developed by BDD (Binary decision diagram) and SAT (Boolean satisfiability) based techniques. BDD can work on reachability analysis by functional dependencies construction or test pattern generation by circuit behavior modeling. However, memory in BDD will grow up while circuit size is large or

bad variable ordering. With SAT-based verification method, the verification problems such as equivalence checking and ATPG (automatic test pattern generation) are converted as a Boolean satisfiability problem and solved by SAT engines. In recent years, SAT-based verification methods have been more popular than BDD-based verification methods due to greater improvement of the SAT engine, such as Zchaff [3], BerkMin [4], C-SAT [5] and MiniSAT [6]. SAT lies at the core of many practical application domains including EDA (e.g. automatic test generation and logic synthesis) and AI (e.g. automatic theorem proving). As a result, the subject of practical SAT solvers has received considerable research attention, and numerous solver algorithms have been proposed and implemented.

Most of SAT solvers are based on conflict-driven technique, which would learn conflict constraint implicitly. However, only local information is considered as the learnt constraints in SAT solving, but no cross-timeframe constraints are explored within SAT engines. Therefore, we propose a new data mining algorithm and constraint filtering method to exploit cross-timeframe conflict constraints through amount of simulation. Instead of modifying the kernel of SAT solver, we add the learnt constraints to the original problem explicitly to facilitate SAT solving.

To explore conflict constraints in circuit, analyzing the structure of circuit is straightforward. However, since the complexity of circuit grows in modern design, it is difficult to analyze the circuit structure directly. An alternative way to obtain circuit information is to observe the circuit behavior from simulation. Since simulation provides the underlying mechanism of circuit, it can be used to construct a model to mimic the circuit behavior and predict results with certain accuracy for a new instance by data mining techniques. The data mining method is used to explore the information statistically behind the data. In this work, we integrate the data mining techniques to obtain relaxed Boolean functions for particular signals. After Boolean function is constructed through simulation, the correlation of signals can be understood to form conflict constraints.

1.2 Thesis Scope

In this work, we integrate the data mining techniques to bounded sequential equivalence checking problem. To verify if the two circuits are functionally equivalence, a miter circuit could be constructed by connecting corresponding outputs and expanding to a specific timeframes. With this transformation, a sequential equivalence problem is converted to a combinational equivalence problem, which can be solved by start-of-the-art SAT solver. However, the size of miter circuit is still large. For example, a miter circuit with 40,000 gates per timeframe, which is expanded to 10 timeframe, requires more than one day to prove the correctness.

We propose a 3-stage filtering method to explore conflict constraint in miter circuit. The initial candidates of conflict constraint are constructed by comparing the functional space of any two signals in circuit. The functional space of each signal is obtained by extracting Boolean cubes with data mining technique. In this thesis, we also propose a new data mining algorithm, which combines support-confidence method and impurity measure. The filtering method starts from functional filtering, and historical filtering and structural filtering are used to verify the correctness of initial candidates of constraint. The conflict constraints can be feedbacked into original SAT problem and expected to reduce the runtime of SAT solving process.

1.3 Thesis Organization

In the first chapter, we introduce the functional verification and different approaches in verification techniques. The scope of the thesis is also introduced in this chapter. The rest of the thesis is organized as follows: we introduce the background knowledge of bounded sequential equivalence checking problem, Boolean satisfiable and data mining technique in chapter 2. In chapter 3 and 4, the learning framework and constraint extraction will be introduced, respectively. Then, the experimental results will be shown in chapter 5. Chapter 6 concludes this thesis.

Chapter 2

Background



2.1 Bounded Sequential Equivalence Checking

Typically, the problem of sequential equivalence checking (SEC) can be formulated as checking over time the output of the miter circuit which is composed of two finite state machines (FSMs). On the other hand, *Bounded sequential equivalence checking* (BSEC) [2] as shown in Figure 2.1, is a special case of SEC problems and simplifies the problem formulation by limiting the timeframes under verification to an affordable number.

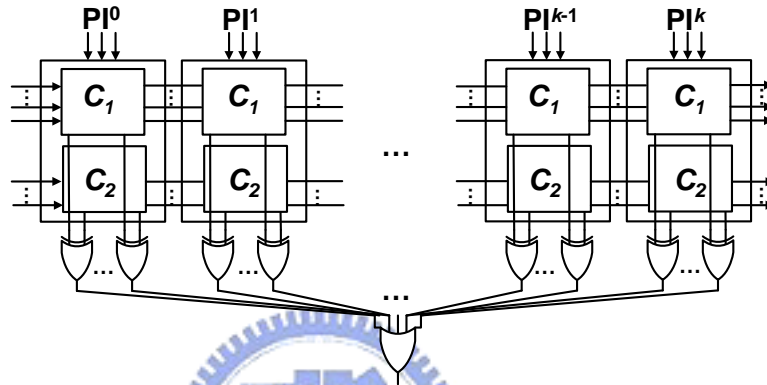


Figure 2.1: Bounded sequential equivalence checking (BSEC) model

Modeling a BSEC problem consists of two steps: *miter construction* and *timeframe unfolding*. The miter is constructed by linking every pair of two corresponding outputs from FSMs to one extra XOR gate. The miter is then unfolded to a user-specified number of timeframes, say k , to form the BSEC model. The combinational logic is duplicated into k copies. All flop-flops (FFs) are removed and inputs of FFs in one timeframe are connected to corresponding signals in the next timeframe. After unfolding the miter circuit, one big OR gate takes the disjunction of every output of added XOR gates from timeframe 1 to timeframe k .

Since the BSEC model is purely combinational, its satisfiability can be solved by any formal engine. If the BSEC model is unsatisfiable (UNSAT), then two circuits are proven to be equivalent under all possible input sequences over the given number of timeframes. On the other hand, if the result is satisfiable (SAT), at least one input sequence witnesses the discrepancy in k timeframes. Accordingly, two circuits can also be in-equivalent after

relaxing the bounds of timeframes to be checked.

However, the larger number of timeframe unfolding in BSEC, the better quality of verification. Once the size of the BSEC model goes larger, SAT solving also becomes less efficient. Many techniques in [3] [6] have been proposed for SAT solvers to extract constraints to early stop exploration during SAT solving. Constraint extraction is an important technique and has also been successfully applied in various electronic design automation (EDA) problems, such as logic optimization and automatic test pattern generation (ATPG). Authors in [10] [11] 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 [12] [17] to guide pair-wise implications to assist test pattern generation.

Many recent studies propose different constraint extraction techniques dedicated to equivalence checking. Binary decision diagram (BDD) techniques are used to estimate the reachability of states in [23] [24] and derive the don't care states in [19]. Equivalent state pairs can be also computed by BDD comparison to facilitate cutpoint finding [8] and to prove equivalence in a partitioning approach [9]. Association rule mining [28] and logic implication are combined in [20] [25] to derive 3-node relations among all internal nodes to facilitate SAT solving for BSEC problems.

Most of previous works [19] [20] [21] focus on the relationship of internal signals at one single timeframe. However, the primary outputs and register inputs are shown to have greater impact than internal signals in [22], and multi-timeframe constraints show to be effective for speeding up SAT solving in [20] [25]. Therefore, we are motivated to proposed a learning-and-filtering framework as shown in Figure 4.2 to uncover cross-timeframe state-pair constraints.

To avoid effort on internal signals over timeframes, our framework will learn the relaxed Boolean functions for flip-flop(FF) state at different timeframes according to a relatively small number of simulation data. FF states that can learn Boolean functions form the basis for the initial set of state-pair constraints. Furthermore, although the total number of FFs is much smaller than that of internal signals, exhaustive checking of all state-pairs is not necessary either. Multiple filtering strategies is required to reduce the total number of constraint candidates.

2.2 Boolean Satisfiability

Boolean Satisfiability (SAT) problem is a well-known constraint satisfaction problem. Given a propositional logic formula f , determining whether there exists a variable assignment ϕ that makes the formula evaluate to true. If such assignment ϕ exists, it indicates that f is satisfiable (SAT). Otherwise, f is unsatisfiable (UNSAT). For example, there is a satisfiable problem $f = (\neg y + x_1)(\neg y + x_2)(y + \neg x_1 + \neg x_2)$, there is a solution $\phi(f)$ satisfied the problem, which is $y = 1, x_1 = 1, x_2$. SAT is one of the central NP-complete problems. [7]

Most solvers operate on problems for which f is specified in conjunctive normal form (CNF). This form consists of the logical AND of one or more clauses, which consist of the logical OR of one or more literals. The literal comprises the fundamental logical unit in the problem, being merely an instance of a variable or its complement denoted as \neg . The advantage of CNF is that in this form, for f to be satisfied, each individual clause must be satisfied. If any clause is unsatisfied, f is unsatisfied. Table 2.1 lists the conjunctive normal form of some basic logic gates, where x_1 and x_2 denotes the inputs of a gate, and y denotes the output.

Table 2.1: Conjunctive normal form of some basic logic gates

Gate type	Conjunctive Normal Form
AND	$(\neg y + x_1)(\neg y + x_2)(y + \neg x_1 + \neg x_2)$
OR	$(y + \neg x_1)(y + \neg x_2)(\neg y + x_1 + x_2)$
NAND	$(y + x_1)(y + x_2)(\neg y + \neg x_1 + \neg x_2)$
NOR	$(\neg y + \neg x_1)(\neg y + \neg x_2)(y + x_1 + x_2)$
XOR	$(\neg y + x_1 + x_2)(\neg y + \neg x_1 + \neg x_2)(y + \neg x_1 + x_2)(y + x_1 + \neg x_2)$
XNOR	$(y + x_1 + x_2)(y + \neg x_1 + \neg x_2)(\neg y + \neg x_1 + x_2)(\neg y + x_1 + \neg x_2)$
NOT	$(\neg y + \neg x_1)(y + x_1)$
BUF	$(\neg y + x_1)(y + \neg x_1)$

Figure 2.2 shows a sample circuit, where I_1, I_2 and I_3 are inputs, O is output, and G_1, G_2 and G_3 are logic gates in this circuit. Table 2.1 is used to convert the circuit to CNF.

The formula of the circuit is, $f_O = (\neg G_1 + I_1)(\neg G_1 + I_2)(G_1 + \neg I_1 + \neg I_2)(G_2 + I_2)(G_2 + I_3)(\neg G_2 + \neg I_2 + \neg I_3)(G_3 + \neg G_1)(G_3 + \neg G_2)(\neg G_3 + G_1 + G_2)$, which is encoded by each logic gate.

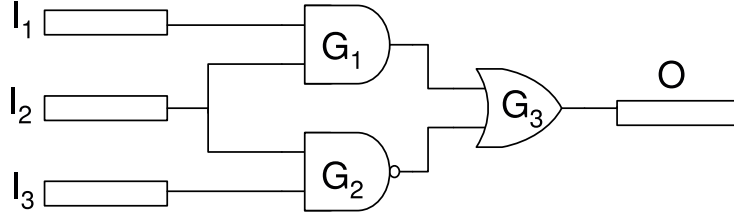


Figure 2.2: A sample circuit

The popular SAT-solvers based on the DPLL (Davis-Putnam-Logemann-Loveland) algorithm [13] [14], backtracking by conflict analysis and clause recording [18], and Boolean constraint propagation (BCP) using watched literals [3]. The search procedure in DPLL algorithm is to pick a variable heuristically and assign specific value to the selected variable, until the propagation detects a conflict. The backtracking process will continue until the conflict clause becomes unit since the assignment of the unit clause could propagate to other clauses. The search procedure stops if all variables are assigned or a conflict occurs in one variable, and the solver returns SAT and UNSAT, respectively.

Algorithm 1 is the basic procedure of DPLL-based SAT solver in MiniSAT [6]. Each run of outer loop starts from one variable assignment. If one variable has been assigned, the inner loop could use to examine the variable assignment is legal or not. The loop starts from unit clause propagation. In the conflict occurs, the algorithm will analyze the conflict and add a conflict clause to the original problem. If the conflict occurs in top-level, the solver will return UNSAT. Otherwise, if the conflict does not occur in top-level, the procedure will backtrack to undo assignments until the conflict clause is unit. If there is no conflict occurs in this loop, the algorithm will continue to decide the assignment of next variable or check if all variables are assigned.

MiniSAT [6] is a minimalistic implementation of a Chaff-like SAT solver [3]. Compared with Zchaff, MiniSAT provides the incremental SAT interface and support for user defined Boolean constraints. Although the performance of Zchaff and MiniSAT are similar,

MiniSAT is easy to integrate to larger system with verification techniques due to portable ability. In this work, we integrate MiniSAT to proposed verification method to solve the bounded sequential equivalence checking problem.

Algorithm 1 DPLL-based SAT solver in MiniSAT [6]

```
1: loop
2:   propagate() //propagate unit clauses
3:   if not conflict then
4:     if all variables assigned then
5:       return SATISFIABLE
6:     else
7:       decide() //pick a new variable and assign it
8:     else
9:       analyze() //analyze conflict and add a conflict clause
10:    if top-level conflict found then
11:      return UNSATISFIABLE
12:    else
13:      backtrack() //undo assignments until conflict clause is unit
```

2.3 Data Mining

Data mining technique is a technique to explore information from data. It utilizes statistical method to analyze the data and exploit the most frequent pattern to represent the data. Data mining method is widely used on many fields including marketing, science and engineering. Data mining techniques incorporated with verification problem will bring about test vector selection with classification [15], process variation modeling with regression [16], or constraints extraction with data correlation [20] [25].

The major procedure to apply data mining on specific problem involves the steps: problem definition, training data collection, data preprocessing and model construction. For example, in this work, learning relaxed Boolean function of particular signals in miter circuit

is the objective. Training data in this case is the input pattern and response on the particular signals, which could collect from amount of simulation. Data preprocessing is used to remove redundant information in training data, for example, the reset signal or clock signal in circuit. The last step is model construction, which built the simplified model for each signal through simulation.

We summarize a few examples to illustrate different types of data mining techniques on simulation data.

- Implication rule: The simplest form of mining may be the extraction of an implication rule such as $(X \implies Y)$. The problem can be harder if we consider finding all these rules such that both A and B are internal signals whose total amount is large. We propose to incorporate the concept of support-confidence framework used in association rule mining and develop a similar framework to mine implication rules.
- Signal correlation: We may relax the implication rule into signal correlation, denoted as $(X \sim Y)$. This means that with a probability of c , signal A and signal B change their values together. This is another type of association rule mining.
- Clustering: Given K signals, we would like to partition them into groups such that signals within the same group are highly correlated. Clustering can be an application from the result of signal correlation mining.
- Association rule: Given K signals, we would like to find a subset of signals S_1 and a subset of signals S_2 such that the changes on signals in S_2 is most likely due to the changes of signals in S_1 (with a confidence level c). We denote this as $(X \Rightarrow Y)$.
- Statistical modeling: Given two sets of signals: S_1 and S_2 , we would like to obtain a statistical model f such that $f(S_1)$ can be used to predict values on S_2 . This is a typical statistical learning problem. Techniques such as neural networks or computational learning techniques are suitable for this problem.

2.3.1 Association Rule Mining

Association rule mining (ARM) was first introduced by Agrawal et al. in [27], which aims to extract interesting correlations, frequent patterns, association or casual structures among sets of items in the transaction database or data repository. By definition, an association rule is an implication of the form $X \Rightarrow Y$, where X and Y are frequent itemsets in a transaction database and $X \cap Y \neq \emptyset$. Conventionally, X is called antecedent while Y is called consequent. An association rule can be further classified into four different forms, $X \Rightarrow Y$, $X \Rightarrow \neg Y$, $\neg X \Rightarrow Y$ and $\neg X \Rightarrow \neg Y$. The first form is called a positive rule while all of the others are called negative rules.

When studying association rules, there are two important basic measures, (1) support and (2) confidence. Support, denoted as $supp$, is a statistical measure and $supp(X)$ is defined as the fraction of records that contains the target itemset, X , to the total number of records in the database. However, confidence (denoted as $conf$) of an association rule $X \Rightarrow Y$ is a measure of strength and defined as the ratio $supp(X \cup Y)/supp(X)$ where $X \cup Y$ means both X and Y are present.

The support-confidence framework proposed in [27] seeks positive rules of the form $X \Rightarrow Y$ with support and confidence greater than, or equal to, user-specified minimum support (γ_{sup}) and minimum confidence (γ_{conf}) thresholds, respectively, where

- X and Y are disjoint itemsets; that is, $X \cap Y = \emptyset$
- $supp(X \Rightarrow Y) = supp(X \cup Y)$
- $conf(X \Rightarrow Y) = supp(X \cup Y)/supp(X)$

The negation of an itemset X is indicated by $\neg X$. The support of $\neg X$, $supp(\neg X)$, is $1 - supp(X)$. To take a particular itemset, $i_1 \neg i_2 i_3$, for example, $supp(i_1 \neg i_2 i_3) = supp(i_1 i_3) - supp(i_1 i_2 i_3)$. Like positive rule, a negative rule (e.g. $X \Rightarrow \neg Y$) also has a measure of its strength, $conf$, defined as $supp(X \cup \neg Y)/supp(X)$. By extending the definition, negative association rule discovery seeks rules of the form $X \Rightarrow \neg Y$ with support and confidence greater than, or equal to, user-specified minimum support (γ_{sup}) and minimum confidence (γ_{conf}) thresholds, respectively, where

- X and Y are disjoint itemsets; that is, $X \cap Y \neq 0$
- $sup(X) \geq ms$, $sup(Y) \geq ms$, and $sup(X \cup Y) < ms$
- $sup(X \Rightarrow \neg Y) = sup(X \cup \neg Y)$
- $conf(X \Rightarrow \neg Y) = sup(X \cup \neg Y)/sup(X)$

From the above discussion, we can conclude that association rule mining consists of two steps: finding frequent itemsets and generating rules. In general, finding frequent itemsets contains two sub-steps, candidate large itemset generation and frequent itemset generation. Minimum support (γ_{sup}) is specified from users to decide the itemsets of interest. Finding frequent itemsets is the focus of all association rule mining algorithms. Generating rules is relatively straightforward. Possible rules of interest from the frequent itemsets are enumerated. Then the specified minimum confidence (γ_{conf}) must be satisfied when a rule is finally derived.

2.3.2 Support-confidence Framework

In conventional decision diagram (DD) based mining algorithm, it used simulation data to construct a decision diagram to record the functional space, as shown in Figure 2.3(a). As can be seen in the figure, it is the decision diagram of $f = x_0x_1 + x_2x_3$ and the variable ordering is $x_1 > x_2 > x_3$. The solid line and the dotted line indicate the on space ($x_i = 1$) and the off space ($x_i = 0$), respectively. The decision diagram records the row data in the truth table of $f = x_0x_1 + x_2x_3$. Although the decision diagram could represent the functional space of f , there are two weaknesses in DD-based mining algorithm. (1) good variable ordering requirement and (2) statistical information lack for variable splitting. Although DD-based mining algorithm can handle verification problem such as reachability analysis, the variable ordering problem is still a bottleneck for such algorithm to drive the circuit with deeply sequence. During model generation by such mining algorithm, the learning model may be biased by noise information due to lack the statistical information for variable splitting. The two weaknesses of DD-based mining approach will limit its ability to apply on high complexity circuit.

To avoid weaknesses of DD-based algorithm, a new mining algorithm considering statistical information will be developed. This new approach is based on DD-based mining method and support-confidence method in section 2.3.1. It considers the statistical information while variable splitting and generate rule-based model instead of diagram-based model. The goal of this approach is to extract abstraction view of circuit through simulation, and the ruling results can be taken as bigger Boolean cubes in functional space of circuit. The ruling results are so called ruling cubes shown in Figure 2.3(b) and 2.3(c). In Figure 2.3(b), the ruling cubes x_0x_1 and x_2x_3 represents the ON space. That is mean both x_0x_1 and x_2x_3 will imply $f = 1$. For the same concept, Figure 2.3(c) shows the ruling cubes for OFF space.

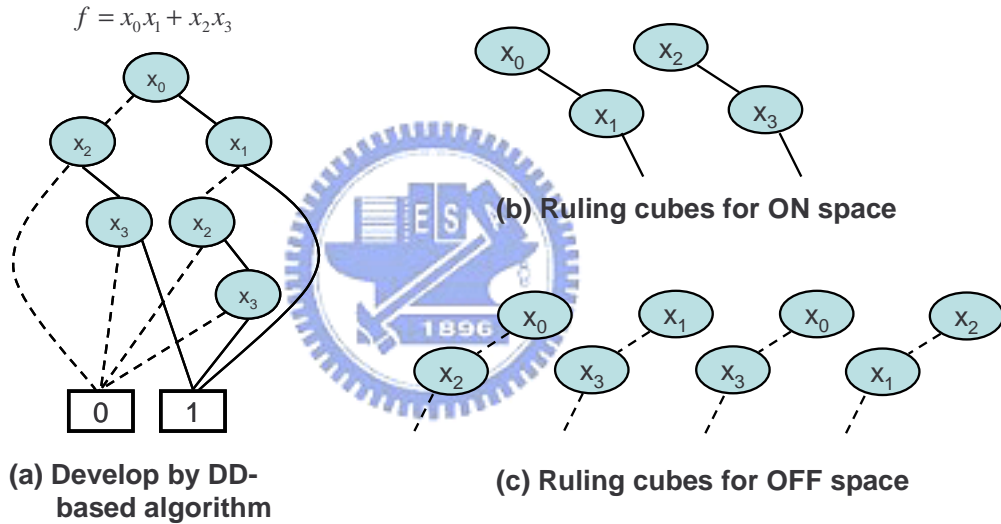


Figure 2.3: Example of ruling cubes

The ruling cubes are generated by taking the statistical information into account. First of all, a supporting variable method will explore the best spilting variable in each process. Next, the support-confidence measurement introduced in section 2.3.1 will calculate the sup and $conf$ rate to quantify the ruling cube. If the sup and $conf$ of ruling cube reach the minimal support (γ_{sup}) and minimal confidence (γ_{conf}) threshold, the ruling cube will be extracted. At the same time, the training data explained by the extracted ruling cube will be removed and the remaining data will continue to extract other ruling cubes. On the

other hand, if the *sup* and *conf* of ruling cube do not reach the γ_{sup} and γ_{conf} threshold, the supporting variable method will be applied to select another spilling variable until the ruling cube satisfying the *ms* and *mc* requirement. The detail algorithm will be introduced in section 3.2.

Return to the bounded sequential equivalence checking problem. The mined ruling cubes can be constructed the approximate function for each state (output of flip-flop) at some timeframe, the unreachable cross-timeframe can be examined by checking the intersection of functions for two arbitrary states. If the intersection is empty, then this state pair can be considered as a candidate of unreachable state pair. These unreachable state pairs will be set as constraints and apply to facilitate BSEC problem.



Chapter 3

Learning Framework



3.1 Training Data Collection

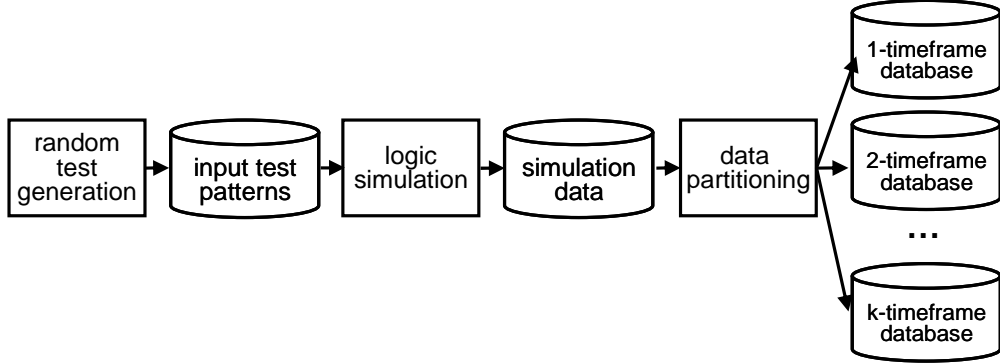


Figure 3.1: Flow of random simulation

The flow of *random simulation* is illustrated in Figure 3.1. First, a small number of test patterns are randomly generated and run through a logic simulator to collect the data from I/O and FFs. Then data partitioning prepares k (empirically, $k = 10$) databases of different timeframes for later learning. For example, for a finite state machine M with 4 PIs, 1 PO and 3 FFs, table 3.1(a) shows the simulation data after logic simulation. Since the FFs can be divided into PPOs and PPIs, each PPO value at one timeframe will be PPI value at the next timeframe. Given $k = 2$, 1-timeframe database collects PI and PPI data as input values and PO data as its output value from each single timeframe. 2-timeframe database collects PI and PPI data from every two consecutive timeframes but only collects PPO as its output only from the latter timeframe. Table 3.1(b) and Table 3.1(c) show the 1-timeframe database and 2-timeframe database, respectively.

3.2 Learning Relaxed Boolean Function

3.2.1 Support-confidence Method

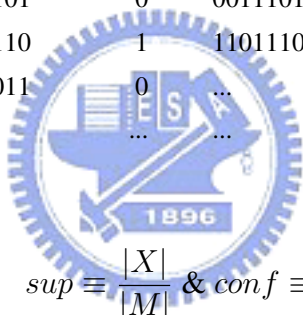
The original concept the support-confidence framework [27], is the measurement of frequency pattern by counting the simulation data. Applying the support-confidence framework to Boolean data learning, the support sup and the confidence $conf$ denote the frequency and the accuracy for one Boolean cube, respectively, on the basis of the database M . The formal definitions of support and confidence are given as follows.

Table 3.1: Partitioning data into databases

(a) raw data

time	PI	PPI	PO	PPO
0	1010	XXX	1	000
1	1111	000	0	101
2	0011	101	0	110
3	1101	110	1	011
4	1011	011	0	001
...

(b) 1-timeframe db.		(c) 2-timeframe db.	
input	output	input	output
1010XXX	1	1010XXX1111000	0
1111000	0	11110000011101	0
0011101	0	00111011101110	1
1101110	1	11011101011011	0
1011011	0
...



$$sup \equiv \frac{|X|}{|M|} \ \& \ conf \equiv \frac{|X \cap Y|}{|X|}$$

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 $|\cdot|$ denotes the size of one set.

Each Boolean function of one FF state can be approximated by a set of Boolean cubes. For example, in Figure 3.2, $\{t_i\}$ 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 relaxed Boolean function F^* later. In contrast, those Boolean cubes that fail to satisfy the support and confidence criteria will be excluded in F^* .

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 3.2: Example of support-confidence learning

Figure 3.2 shows an example of support-confidence learning. Given the database $M = \{t_1, \dots, t_5\}$, 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 satisfies the support and confidence criteria among three cubes and also is the only ruling cube on the basis of M in this example.

Support-confidence learning algorithm is proposed to derive the set of ruling cubes for constructing the relaxed Boolean function for each flip-flop state. In Figure 3.2, $c_3 = x1xx1x$ 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 generated by adding the support literal one by one until no further support literal can be found.

3.2.2 Impurity Measure

According to [26], 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 . In short, the support variable with maximum gain is the most important variable of the given database. For two literals (x and \bar{x}) of one input variable and the database M , $g(x)$ and $g(\bar{x})$ can be formulated as

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

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 ; $g(\bar{x})$ 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.

$x_1x_2x_3$	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
$g(\bar{x})$	0	1/5	0
$gain(x)$	2/5	1/5	2/5

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
$g(\bar{x})$	1/3	2/3
$gain(x)$	1/3	2/3

$v = x_1 \quad c = x_1$
 $c.sup = \frac{1}{2} \quad c.conf = \frac{1}{2}$

$v = \bar{x}_3 \quad c = x_1\bar{x}_3$
 $c.sup = \frac{1}{4} \quad c.conf = 1$

(a) select the first support variable (b) select the next support variable

Figure 3.3: Example for generating one ruling cube

Figure 3.3(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 $g(\bar{x}_1) = \frac{0}{4+0+1}$. $g(x_2)$, $g(\bar{x}_2)$, $g(x_3)$ and $g(\bar{x}_3)$ 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 3.3(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 2 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. $SupVarSelect()$ is applied to select the next support 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 criteria 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 .

Note that according to the learning theory, the quality of a data-mining algorithm depends upon the data complexity, not the structural complexity underlying. Therefore, given a small number of simulation data, the FF state at the smaller k -th timeframe seems relatively easy to learn its relaxed Boolean function. However, for those FF state at the large k -th timeframe, more simulation may be needed but not necessary. Since learning for relaxed Boolean functions is one-time cost, the user can allocate a sufficient amount of time for his BSEC problems.

Algorithm 2 SupportConfidenceLearning(): a support-confidence learning algorithm with the impurity measure

```
1:  $N = \log_2|M|$ ;  
2:  $F^* \leftarrow \emptyset$ ;  
3: while ( $|D| \leq |M| \times \gamma_{cov}$ )  
4:    $f = 1$ ;  
5:   do {  
6:      $x = \text{SupVarSelect}(M, f)$ ; // impurity measure  
7:      $f \leftarrow f \cap x$ ;  
8:      $\text{update}(f_{sup}, f_{conf})$ ; // 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:       $\text{update}(D)$ ; // update by ruling cube  
12:      break;  
13:   } while ( $|f| < N$ );  
14:   if ( $f_{sup} < \gamma_{sup} \ \&\& \ f_{conf} < \gamma_{conf}$ )  
15:      $\text{update}(D)$ ; // update according to the excluded cube
```

Chapter 4

Constraint Extraction Method



4.1 Constraints in BSEC Problem

Figure 4.1 is an example to illustrate how the conflict constraints work on the SAT solving process. Before the example, let's recall the basic idea in SAT solving. The DPLL-based algorithm, introduced in section 2.2, is the core of most start-of-art SAT engines. The procedure of DPLL-based algorithm is illustrated as follows:

1. Trace from the clause with minimum number of literals, especially in unit clause
Assign value to variable v , and push v into assignment queue
2. Use v to apply implication on other clauses
If variable u could be implied by v , then push u into assignment queue
3. While size of assignment queue is equal to the number of variables, return SAT
if implication $v' \neq v$ in assignment queue, return UNSAT

In DPLL-based algorithm, the assignment queue is used to record the assigned variables. Since all clauses in a SAT problem must be satisfiable and the problem can be satisfiable. DPLL-based algorithm starts from the clause with minimum number of literals. If one variable v could be assigned the specific value, the variable could be pushed into the assignment and propagated to other clauses. The procedure would continue to assigned other variables. If all variables have been assigned, the problem is satisfiable. If one implied variable conflicts the results in assignment queue, the problem is unsatisfiable.

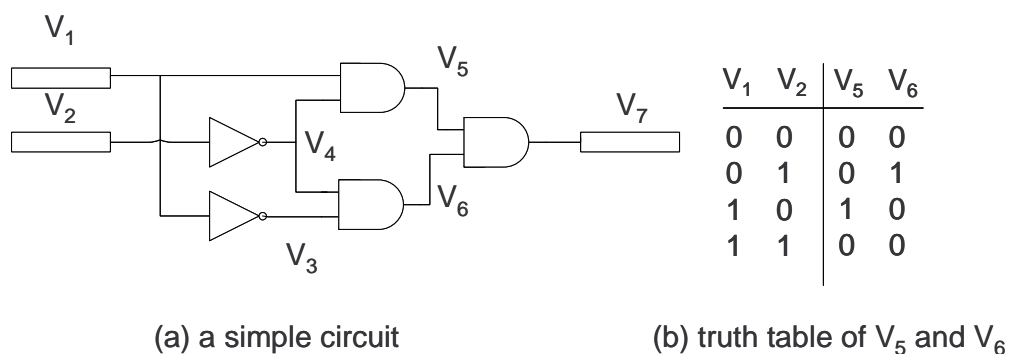


Figure 4.1: An example to illustrate constraint in SAT solving

Figure 4.1(a) is the sample example and Figure 4.1(b) is the truth table of V_5 and V_6 in sample circuit. From the truth table, it implies that the logic value of V_7 is stuck at 0. If the circuit is modeled as a SAT problem and target $V_7 = 1$, the SAT engine is expected to return an unsatisfiable answer. Table 4.1 lists the clauses of the sample circuit in Figure 4.1(a).

Table 4.1: Clauses for Figure 4.1

1	$\neg V_5 + V_1$	8	$V_4 + V_2$
2	$\neg V_5 + V_4$	9	$\neg V_3 + \neg V_1$
3	$V_5 + \neg V_1 + \neg V_4$	10	$V_3 + V_1$
4	$\neg V_6 + V_2$	11	$\neg V_7 + V_5$
5	$\neg V_6 + V_3$	12	$\neg V_7 + V_6$
6	$V_6 + V_2 + V_3$	13	$V_7 + \neg V_5 + \neg V_6$
7	$\neg V_4 + \neg V_2$	14	V_7

The procedure is used to solve the SAT problem, as follows:

- step1: start from unit clause 14, imply $V_7 = 1$, push V_7 into assignment queue
- step2: process V_7 , imply $V_5 = 1$ in clause 11 and imply $V_6 = 1$ in clause 12, push V_5 and V_6 into assignment queue
- step3: process V_5 , imply $V_1 = 1$ in clause 1 and imply $V_4 = 1$ in clause 2, push V_1 and V_4 into assignment queue
- step4: process V_6 , imply $V_2 = 1$ in clause 4 and imply $V_3 = 1$ in clause 5, push V_2 and V_3 into assignment queue
- step5: process V_1 , imply $V_3 = 0 \longleftrightarrow$ conflict with $V_3 = 1$, return UNSAT.

In the procedure, it requires five steps to prove the SAT problem in Table 4.1 is unsatisfiable. Although the five steps seems not heavy, it would be difficult as solving larger circuits. Conflict constraints are one way to speedup the proof. From the truth table in

Figure 4.1, it is clearly to observe that the $V_5 = 1$ and $V_6 = 1$ can not happen at the same time. we call that (V_5, V_6) is a conflict constraint in the sample circuit and add the conflict constraint as a conflict clause after original SAT problem. Since (V_5, V_6) means $V_5 = 1$ and $V_6 = 1$ can not happen at the same time, it indicates $V_5 = 0$ and $V_6 = 0$ may occur in the circuit. As a result, the clause of (V_5, V_6) can be written as $\neg V_5 + \neg V_6$ and added as 15th clause after original SAT problem. The solving procedure of new SAT problem is as follows,

- step1: start from unit clause 14, imply $V_7 = 1$, push V_7 into assignment queue
- step2: process V_7 , imply $V_5 = 1$ in clause 11 and imply $V_6 = 1$ in clause 12, push V_5 and V_6 into assignment queue
- step3: process V_5 , imply $V_1 = 1$ in clause 1 and imply $V_4 = 1$ in clause 2, imply $V_6 = 0$ in clause 15 \longleftrightarrow conflict with $V_6 = 1$, return UNSAT.

Compare with solving the original SAT problem, the SAT problem with a conflict constraint only requires 3 steps of proof. The example illustrates the power of conflict constraints. Since the BSEC problem is a combinational SAT problem and the solving process is similar to the BCP procedure, and the conflict constraints will also help SAT solving.

One way to find out conflict constraints in BSEC is to list the truth table for all internal signals and discover the unhappen situation in the truth table. However, it is impossible to enumerate all input combination to construct such truth table as in Figure 4.1(b). The alternative method is to construct the relaxed Boolean function of each signal, as mention in section 3.2. Since the number of combination of internal signals is quite large, in this work, we only extract the information on flip-flops. The detail explanation is introduced in section 2.1.

The previous work in [20] is the first study of applying a support-confidence algorithm named Apriori to explore the *implication constraints* among 3 internal nodes ($a \cdot b \rightarrow c$) for SAT solving. However, in this work, we do not mine constraints directly from data. Instead, we propose a new learning algorithm which modifies the notion of support-confidence with impurity measures [28] to infer the relaxed Boolean functions for each FF state at different timeframes. Later, *conflict constraints* will be derived by composing two relaxed Boolean

functions.

DEFINITION (Conflict Constraint)

A state-pair (p^i, q^j) , where i, j denote the timeframes and $j > i > 0$, in a finite state machine M is a *conflict constraint* if and only if $\forall k > i, q^{k+j-i}$ can never appear after p^k appears in M for all input sequences.

Such constraints can be used to early stop the random walk during the SAT solving process, and their effectiveness will be demonstrated through our experiments later. The proposed method to exploit conflict constraint is introduced in the following section.

4.2 3-stage Filtering Method

Since previous studies [19] [20] 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 4.2 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.

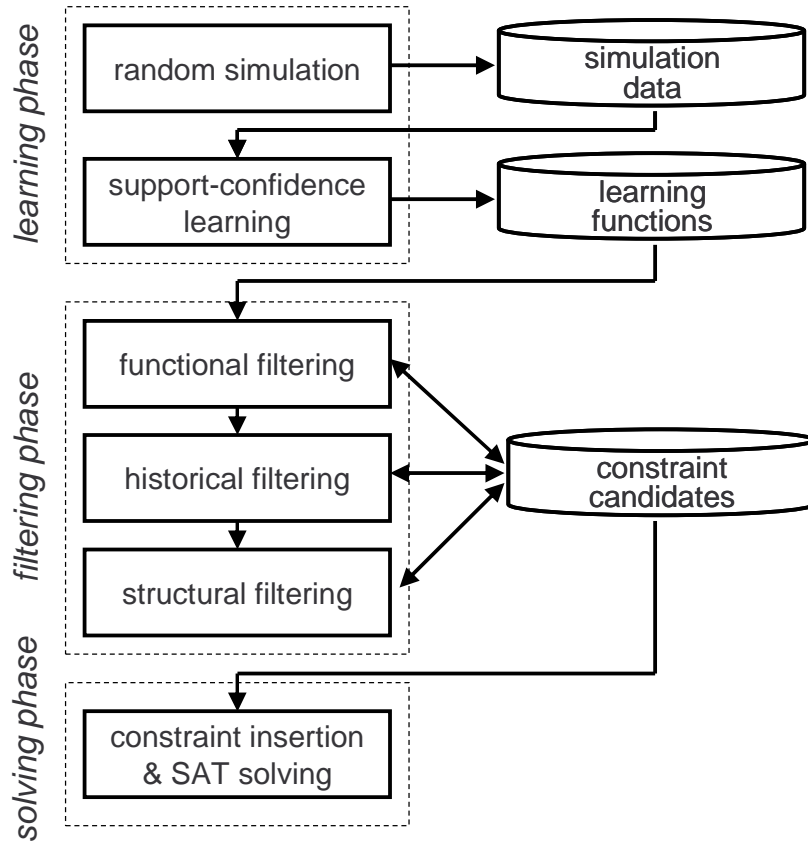


Figure 4.2: A learning-and-filtering framework for BSEC

4.2.1 Functional Filtering

As defined in the previous section 4.1, *conflict constraints* can be obtained by taking the conjunction of the relaxed Boolean functions for two states at different timeframes.

SAT solver will be then applied to the conjunctive function. For example, given $f(s_p^i)$ and $f(s_q^j)$ as the functions for the state s_p^i of FF p at timeframe i and state s_q^j of FF q at timeframe j , respectively. If $f = f(s_p^i) \cap f(s_q^j)$ is UNSAT, there exists no input test which can satisfy both FF states at individual timeframes concurrently. Therefore, $(f(s_p^i), f(s_q^j))$ is one constraint candidate. Note that, for each FF r at timeframe k , the support-confidence learning algorithm will run twice: one for ON state s_r^k , and the other for OFF state \bar{s}_r^k .

4.2.2 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 (\bar{s}_p^k, s_q^{k+2}) as the constraint candidate to be checked, if FF p in some timeframe k has the state value of 0 and FF q in two timeframes later has the state value of 1, then (\bar{s}_p^k, s_q^{k+2}) will be removed from the candidate set. This situation happens because the support-confidence learning is statistical and may overlook small Boolean cubes resulted from some patterns.

4.2.3 Structural Filtering

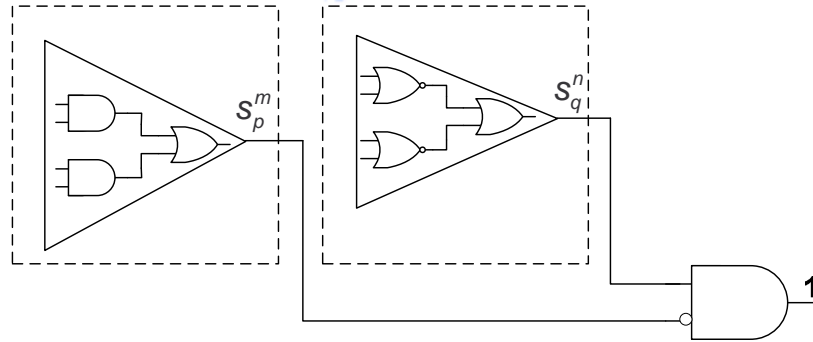


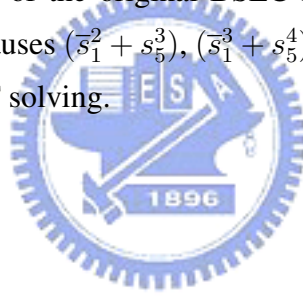
Figure 4.3: Illustration for structural filtering

At this stage, *structural filtering* is to ensure the validity of each candidate under the unfolded miter. The logic cones of individual FF states will be extracted first and combined by an extra AND gate. Such an augmented circuit is termed augmented constraint circuit (ACC). SAT solving is performed on the ACC by enforcing the output of the AND gate

as one. If the ACC is UNSAT, such a state-pair is a true conflict constraint. Otherwise, it should be removed from the candidate set. To give an example, if (\bar{s}_p^m, s_q^n) is one constraint candidate, the *inverted* output of flip-flop p at timeframe m and the output of flip-flop q at timeframe n are connected by an extra AND gate. Such an example is illustrated in Figure 4.3. Next, SAT solving is performed on the corresponding ACC with enforcing 1 on the output of the extra AND gate. If the result is UNSAT, (\bar{s}_p^m, s_q^n) is a true constraint; otherwise, (\bar{s}_p^m, s_q^n) should be removed.

4.2.4 Constraint insertion

Constraint insertion is the final step in the proposed framework. Given k as the number of timeframe unfolding in BSEC problems, each extracted constraint will be translated into multiple CNF constraint clauses of disjunction of inverting two FF states over k timeframes and appended to the CNF of the original BSEC model. For example, if (s_1^2, \bar{s}_5^3) is one proven constraint, CNF clauses $(\bar{s}_1^2 + s_5^3), (\bar{s}_1^3 + s_5^4), \dots, (\bar{s}_1^{k-1} + s_5^k)$ will be appended to the original CNF for final SAT solving.



Chapter 5

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 and ITC 99 circuits are used as benchmarks for bounded sequential equivalence checking. Each circuit is synthesized with 10 different configurations by Design Compiler from Synopsys. MiniSAT 2.0 [3] is the one of state-of-the-art SAT solvers and applied for SAT solving in our experiments. 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 in functional filtering is 2,000.

Table 5.1: Characteristics of BSEC models for benchmark circuits

miter	# of PI	# of PO	# of FF in miter	# of k timeframes	# of FF \times k -timeframes
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
b04	8	11	132	30	3960
b11	7	6	60	30	180
b13	10	10	102	30	3060
b15	36	70	834	15	12510

Table 5.1 shows the characteristics of BSEC models for ISCAS 89 and ITC 99 circuits in our 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 5.2 demonstrates the effectiveness of 3-stage filtering by reporting the numbers of constraint candidates across different timeframes after each filtering. Column 1 lists the name of the benchmark circuits while column 2 represents the initial number of constraint candidates. Column 3, 4 and 5 denote the numbers of candidates after *functional*, *historical* and *structural filterings*, respectively.

Table 5.2: Comparison of numbers of constraint candidates

miter	# of cross-timeframe constraints			
	initial	functional filtering	historical filtering	structural filtering
s298	6160	59.3	44.9	27.3
s349	7080	217.8	110.0	109.9
s713	10224	6.2	5.1	5.1
s832	760	88.3	63.3	30
s1196	10224	78.7	57.3	42.9
s1488	1104	88.0	52.9	52.8
s4863	227812	2000	312.9	310.2
s15850	8648640	4000	2631.3	2381.7
s35932	95537664	2000	1336.0	1105.0
s38584	57878040	2000	1214.4	825.6
b04	138864	2000	1783.2	1298.1
b11	28560	2000	1643.9	1138.2
b13	82824	2000	1122.2	939.0
b15	5561112	2000	1238.1	1732.8

Table 5.3 shows the improvement of SAT solving for BSEC problems. Column 1 lists the name of the benchmark circuits and column 2 is the # of unfolded timeframes. Column 3 represents the combined runtime for both learning and filtering. Column 4 and 5 denote the runtime of SAT solving without and with constraints, respectively. Column 6 reports the speedups computed by the original runtime in Column 4 divided by the new runtime in

Column 5.

Table 5.3: Runtime for BSEC problems

miter	k time-frames	learning time (s)	original (s) [A]	new (s) [B]	speedup (X) [A]/[B]
s298	40	1.0	13.3	0.2	66.5
s349	40	1.5	3.8	0.2	19.0
s713	30	7.7	6328.3	176.2	35.9
s832	30	3.9	8.8	0.4	22.0
s1196	30	8.6	14.4	13.3	1.1
s1488	30	3.9	13.0	2.7	4.8
s4863	15	180.4	7319.9	26.9	272.1
s15850	15	2684.3	7823.1	23.7	330.1
s35932	10	1135.3	7503.9	14.3	524.7
s38584	10	529.9	2077.5	11.8	176.1
b04	30	28.6	3485.2	0.5	6070.4
b11	30	41.2	35.9	0.9	39.9
b13	30	15.6	8.0	0.2	40.0
b15	15	335.1	4897.8	8.1	604.7

Our experimental results show different speedups on SAT solving of benchmark BSEC circuits with an average as 500X, excluding the time for learning and filtering. Significant improvement can be observed on the big circuits while minor improvement can be observed on the small circuits. We also shows the speedups on 4 large circuits, s35932, s38584, b13 and b15 with respect to 10 configurations in Figure 5.1. Although for each case, 10 configurations result in different speedups but the all speedups are of the same order.

We compare the results with [20], which used association rule to imply three nodes relation on internal signals. Column 1 lists the name of the benchamrk circuits, which is the same as in [20]. 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 time and SAT solving time, respectively. Column 6 reports the

speedups computed by Column 3/Column 5. Column 7 is the result excerpted from [20]. The results are only from one configuration in circuits.

miter	k time frames	origin(s)	new(s)	mining +new(s)	our speedup	speedup [20]
s298	40	30.39	0.12	9.25	3.29	3.11
s832	30	2028.65	0.72	6.41	316.48	50.30
s1196	30	96.19	55.78	79.98	1.20	2.59
s1488	30	754.03	5.68	21.70	34.75	33.93
s4863	15	7725.44	4.35	236.02	32.73	15.07
s15850	15	64860	227.63	1593.46	40.71	40.28
s35932	10	51744.3	173.78	1397.12	37.04	4.54
s38584	10	50464.3	19.50	1221.10	41.33	3.43

Table 5.4: Runtime for BSEC problems compare with [20]

Since we do not know the configuration of circuit used in [20], the comparison may be not faired. However, the results should be focused on the speedup in our approach and their result. Compared with results from [20], not all of cases with our approach would have such benefit because the effective of conflict constraints is depended on the property of circuit. Our approach only finds out constraints between states with different timeframes, but in [20], they consider about internal nodes. Different circuits may agree with different approaches. However, the results show that our approach is overall good in most cases. By the way, in [20], they explained that their approach has less improvement in easy miters, so they only reported hard miters. The same problem we meet. It is because the complexity is depended on the number of flip-flops, and the computation time of exploring constraints in easy miter is almost the same as in hard miters. So there is no obviously improvement in easy miter with our approach.

To demonstrate the effectiveness of the extracted constraints, we compare the numbers of timeframe unfolding on 4 large benchmark circuits without and with the extracted constraints in a fixed time, say 1200 seconds in our experiments. In Figure 5.2, the dotted lines denote the original benchmark circuits while the solid lines represent the benchmark circuits with extracted constraints. Results show that after adding constraints, the bound

k can increase from 8X to 20X. It also means that the quality of BSEC can be improved greatly by applying our framework. However, different time limits may result in different improvements.

We further investigate the relations between the number of constraints and runtime for SAT solving on three big ISCAS 89 circuits. Figure 5.3 shows the result where Y-axis represents the runtime 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.



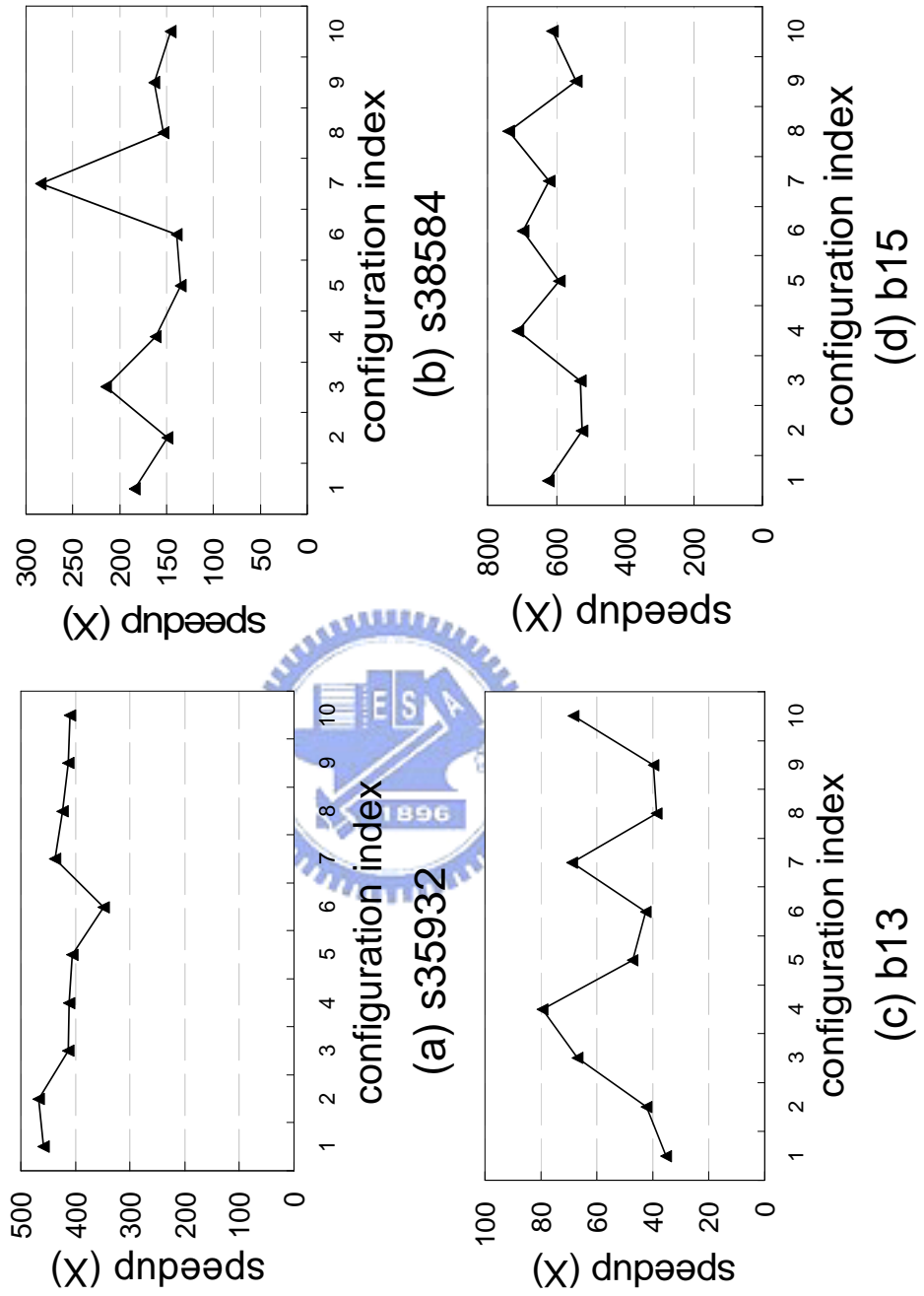
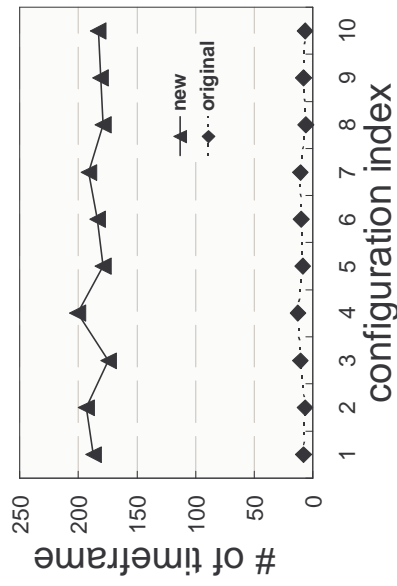
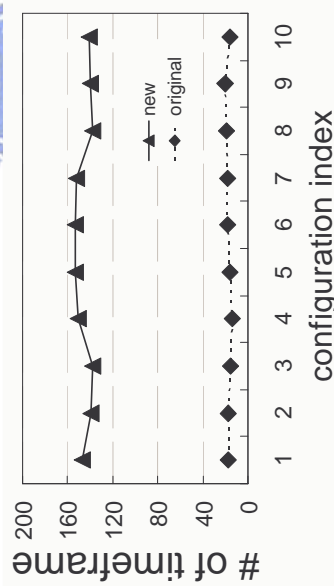


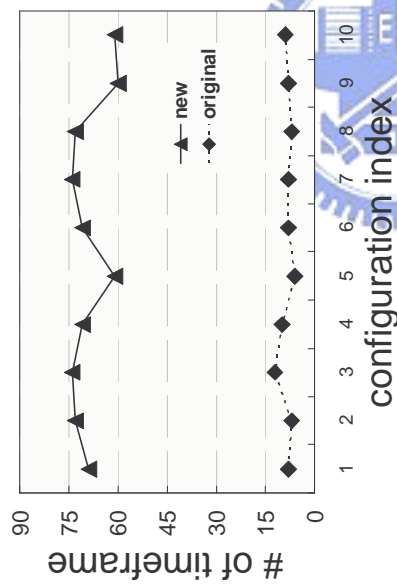
Figure 5.1: Speedups of 4 large circuits with respect to 10 configurations



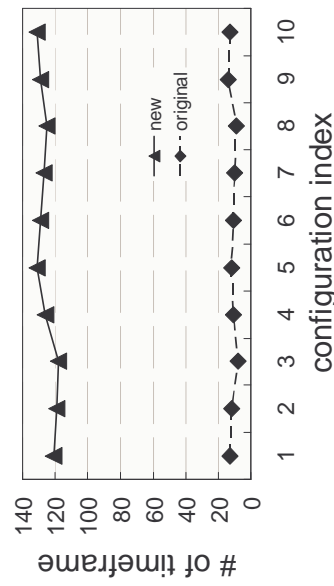
(a) s35932



(c) b13



(b) s38584



(d) b15

Figure 5.2: Bound for SAT solving of BSEC problems with and without constraints

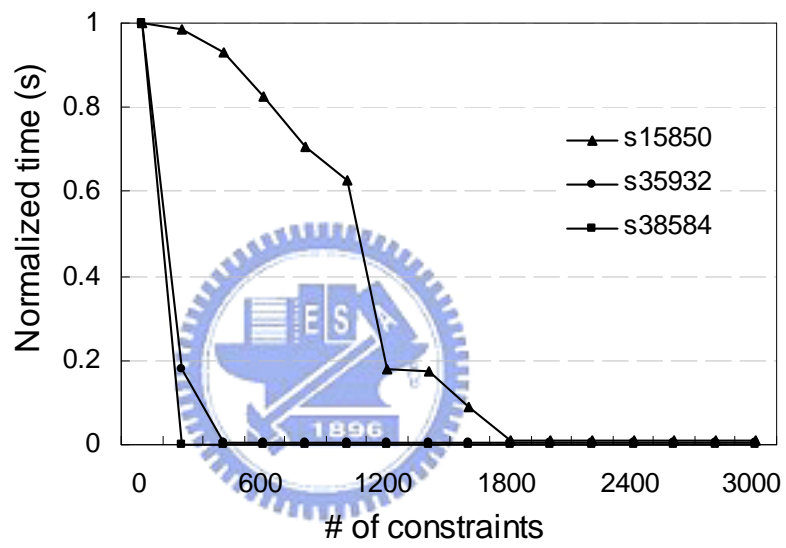


Figure 5.3: SAT solving time with different # of constraints

Chapter 6

Conclusion



The general problem of checking functional equivalence for two sequential circuit is still far from being solved. In this thesis, 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.

To exploit conflict constraints in BSEC problem, we propose a 3-stage filtering method. They are functional filtering, historical filtering and structural filtering. In functional filtering, we propose a new data mining method to construct relaxed Boolean function for particular signals. Instead of using decision diagram to construct Boolean function, our support-confidence method with impurity measure provides statistical perspective on Boolean functions. In historical filtering, we use simulation information to prune the false candidate in functional filtering since the goal of the learning model is simple instead of precise. The final stage is structural filtering. Since the remain constraints behind first two stages are from limited simulation data, the result may be not global true in BSEC model. To make sure the extracted constraints are global true, each of them should be injected into the original SAT problem to be verified.

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 and larger ITC 99 circuits. We also demonstrate that the rate of timeframe expansion could be grown on 8X to 20X.

Future works include the quality analysis of the extracted constraints and a better strategy to exploit constraints efficiently. Moreover, instead of conflict constraints, we will exploit other kinds of constraint. Since some circuits may be hard to exploit conflict constraints, such BSEC problems are difficult to improve. In the future, the procedure of constraint extraction can be integrated into sequential SAT solver to improve the performance the sequential equivalence checking problem.

Bibliography

- [1] Charles H.-P. Wen, Li-Chung Wang and Kwang-Ting Cheng, "Chapter 9: Functional Verification," in *Electronic Design Automation: Synthesis, Verification, and Testing*, Elsevier/Morgan Kaufmann, Oct. 2008
- [2] D. Stoffel, M. Wedler, P. Warkentin and W. Kunz, "Structural FSM Traversal" in *IEEE Trans. Computer Aided Design (TCAD)*, vol. 23, no. 5, pp. 598-619, 2004.
- [3] 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.
- [4] E. Goldberg and Y. Novikov, "BerkMin: a Fast and Robust SAT-Solver," in *Proc. Conf. Design, Automation and Test in Europe (DATE)*, pp. 142-149, 2002.
- [5] 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.
- [6] N. Een and N. Sorensson, "An Extensible SAT-Solver," *Theory and Applications of Satisfiability Testing*, pp. 502-518, 2003.
- [7] M. R. Garey and D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness," W.H. Freeman, 1979.
- [8] S.Y. Huang, K.T. Cheng, K.C. Chen, C.Y. Huang and F. Brewer, "AQUILA: An Equivalence Checking System for Large Sequential Designs," in *IEEE Trans. Computers (TC)*, vol. 49, no.5, 2000.

- [9] I.H. Moo, P. Bjesse and C. Pixley, "A Compositional Approach to the Combination of Combinational and Sequential Equivalence Checking of Circuits without Known Reset States," in Proc. Conference on Design, Automation and Test in Europe (DATE), pp. 1170-1175, 2007.
- [10] H. Ichihara and K. Kinoshita, "On Acceleration of Logic Circuit Optimization using Implication Relations" in Proc. Asian Test Symp. (ATS), pp.222-227, 1997.
- [11] 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.
- [12] 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.
- [13] M. Davis and H. Putnam, "A computing procedure for quantification theory," In Journal of the ACM, pp. 201-215, 1960.
- [14] M. Davis, G. Logeman, and D. Loveland, "A machine program for theorem proving," In Proceedings of the Communications of the ACM, pp. 394-397, 1962.
- [15] O. Guzey, L-C. Wang, J. Levitt and H. Foster, "Functional test selection based on unsupervised support vector analysis," in Proc. Design Automation Conf. (DAC), pp. 262-267, 2008.
- [16] H-K. Peng, H-P. Wen and J. Bhadra, "On Soft Error Rate Analysis Beyond Deep Sub-micron - A Statistical Perspective," submitted to Int'l Conf. Computer Aided Design (ICCAD'09), Nov. 2009.
- [17] 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.
- [18] J. P. Marques-Silva and K. A. Sakallah, "A Search Algorithm for Propositional Satisfiability," In IEEE Transactions on Computers, Vol. 48, No. 5, pp. 509-521, 1999

- [19] 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.
- [20] 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
- [21] 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.
- [22] 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.
- [23] G. Cabodi, S. Nocco, S. Quer, "Improving SAT-Based Bounded Model Checking by Means of BDD-Based Approximate Traversals," in Proc. Conference on Design, Automation and Test in Europe (DATE), pp. 898-203, 2003.
- [24] A. Gupta, M. Ganai, C. Wang, Z. Yang, P. Ashar, "Abstraction and BDDs Complement SAT-Based BMC in DiVer," in Proc. Computer Aided Verification (CAV), pp. 206-209, 2003.
- [25] 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
- [26] 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.
- [27] 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.

[28] P.N. Tan, M. Steinbach and V. Kumar. Introduction to Data Mining, Addison Wesley, May 2005

