Mining Multinode Constraints and Complex Boolean Expressions for Sequential Equivalence Checking

Neha Goel

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Michael S. Hsiao, Chair
A. Lynn Abbott
Narendran Ramakrishnan

July 30, 2010
Blacksburg, Virginia

Keywords: Formal verification, data mining, finite state machines, inductive invariants, implications

Mining Multinode Constraints and Complex Boolean Expressions for Sequential Equivalence Checking

Neha Goel

(ABSTRACT)

Integrated circuit design has progressed significantly over the last few decades. This increasing complexity of hardware systems poses several challenges to the digital hardware verification. Functional verification has become the most expensive and time-consuming task in the overall product development cycle. Almost 70% of the total verification time is being consumed by design verification and it is projected to worsen further. One of the reasons for this complexity is the synthesis and optimization (automated as well as manual) techniques used to improve performance, area, delay, and other measures have made the final implementation of the design very different from the golden (reference) model. Determining the functional correctness between the reference and implementation using exhaustive simulation can almost always be infeasible. An alternative approach is to prove that the optimized design is functionally equivalent to the reference model, which is known to be functionally correct. The most widely used formal method to perform this process is equivalence checking. The success of combinational equivalence checking (CEC) has contributed to aggressive combinational logic synthesis and optimizations for circuits with millions of logic gates. However, without powerful sequential equivalence checking (SEC) techniques, the potential and extent of sequential optimization is quite limited. In other words, the success of SEC can unleash a plethora of aggressive sequential optimizations that can take circuit design to the next level. Currently, SEC remains extremely difficult compared to CEC, due

to the huge search space of the problem.

Sequential Equivalence Checking remains a challenging problem, in this thesis we address the problem using efficient learning techniques. The first approach is to mine missing multi-node patterns from the mining database, verify them and add those proved as true during the unbounded SEC framework. The second approach is to mine powerful and generalized Boolean relationships among flip-flops and internal signals in a sequential circuit using a data mining algorithm. In contrast to traditional learning methods, our mining algorithms can extract illegal state cubes and inductive invariants. These invariants can be arbitrary Boolean expressions and can help in pruning a large don't-care space for equivalence checking.

The two approaches are complementary to each other in nature. One computes the subset of illegal states that cannot occur in the normal function mode and the other approach mines legal constraints that represent the characteristics of the miter circuit and can never be violated. These powerful relations, when added as new constraint clauses to the original formula, help to significantly increase the deductive power for the SAT engine, thereby pruning a larger portion of the search space. Likewise, the memory required and time taken to solve the SEC problem is alleviated.

To my family

Acknowledgments

I want to express my deepest gratitude to Professor Michael S. Hsiao for giving me the opportunity to work with him. His guidance and encouragement at every step of my graduate student life has been instrumental in the accomplishment of this work. Without his advices and persistent help, this work would not have been possible. I also want to thank Professor Naren Ramakrishnan and Professor Mohammed Zaki for their guidance and support. I would like to thank Professor Patrick Schaumont and Professor Lynn Abbott for serving in my MS committee.

I want to thank my husband Mr. Harsh Nandan for his continuous advices and support at every step of my life. I would like to thank all the Proactive members for being there with words of encouragement and suggestions.

Last but not the least, I owe my deepest appreciation to my whole family for their constant support and encouragement throughout my life and academic career.

Neha Goel
Virginia Polytechnic Institute and State University
June 2010

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Functional Verification

1.1.1 Necessity and complexity of functional verification

The advances in VLSI technology have led to increasing complexity and size in hardware systems. This increased complexity poses challenges in pre-silicon verification and post-silicon testing of the design. Design error or bugs in a complex system can be expensive to identify and fix. The problem is even worse if the bug is missed during the verification and testing stages, and it remains in the chip even in the field. One of the well-known examples is the floating point division bug in Intel's Pentium processor which cost $475 million to fix.

Functional verification is defined as the process of verifying if the design implementation conforms to its specifications in terms of functionality, timing, testability power dissipation, etc. Functional verification of sequential designs is rapidly becoming one of the most crucial and resource-intensive components of the product design cycle. In most of the industrial

designs, more than 70% of the effort is spent on design verification. The capacities of verification algorithms to handle large designs have been increased dramatically in recently years. However, these increases are often surpassed by the complexity increases of the design. In the worst case, the complexity of the verification task can be exponential in terms of the design size, which forms a major bottleneck for verification techniques. For example, a small design with 40 state elements can already have more than a billion reachable states, and checking if the property holds in all the design might require checking all those states. As today's designs have tens or thousands of state elements, methods to reduce the exponential cost of functional verification is a must. It has been projected that the cost of functional verification will continue to grow with ever-increasing sizes and complexity of designs.

1.1.2 Functional verification approaches

The functional verification approaches can be broadly classified as simulation, emulation, and formal verification. Simulation-based verification has traditionally been used as the primary approach for design verification at different levels of abstraction from behavioral (algorithmic) to RTL and gate level. It is the most natural and intuitive way to validate a design. In such approaches, a set of test patterns is applied at the inputs of the design implementation and the specification (could be a software model of the design). The obtained responses for the implementation are analyzed and compared against the expected response to validate the correctness of the design. If the outputs mismatch, it indicates a discrepancy in the implementation against the spec. However, in order to completely verify the design, all possible test patterns have to be generated. The number of test patterns grows exponentially with the number of primary inputs. For a design with n inputs, the total number of test patterns is 2^n and thus it might not be feasible to generate all the test patterns for large designs, making it impractical to completely verify them. Furthermore, in sequential circuits,

the ordering of the test patterns adds another dimension, exacerbating the cost. So, instead of generating all test patterns, a tractable set of input vectors can be intelligently generated to validate the design against those patterns. But it is possible that the test patterns might miss some corner-case bugs. Another drawback with this approach is that with increasing design complexity and increasing number of input vectors for simulating the design, the time to validate the design also grows. Especially with complex designs such as system on chip (SOC), traditional simulation methods might be time consuming and not very effective in finding subtle design bugs.

Emulation based methods, using Field Programmable Gate Array (FPGA) chips, offers a potential alternative to simulation based approach. They help to speed up software-based simulation by several orders of magnitude. However, the fundamental drawbacks of this approach are the expensive hardware emulators and need to map the design-under-verification to the emulator. Moreover, it might be hard to simultaneously monitor different sets of properties/assertions in the design.

An alternative method to simulation-based validation is formal verification. Formal verification provides a complete proof for verifying the design with respect to its specifications. It has gained increased attention in recent years. Some of the underlying engines used in formal methods include Binary decision diagrams (BDD) and Boolean Satisfiability (SAT). With advances in these techniques, formal verification has been increasingly applied to industrial designs. Since it can verify the design in its entirety, it offers promise in finding hard bugs that might be missed by simulation-based approaches. It employs mathematical proof procedures that can prove or disprove the correctness of the design regardless of the input vectors applied. The consideration of all the test cases is implicit in formal verification. In general, formal methods for verifying designs can be broadly classified as Model Checking and Equivalence Checking discussed in detail in Chapter 2.

The main advantage of simulation based approach is that it can scale to ever increasing design size and complexity, because it performs no analysis on the state space of the design. However, due to this reason it is difficult for it to find bugs in hard-to-reach corner areas of the state space. Formal verification on the other hand explores the entire state space of the design. It assures full coverage but is limited in terms of its processing capacity. Noting the complementary nature of simulation and formal techniques, semi-formal approach have been tried to combine the completeness of formal techniques with the speed, capacity and scalability of simulation. Early works using this approach involved augmenting simulation with small amounts of exhaustive, but bounded formal search on the design but the problem is that they perform formal verification on concrete design, which might be expensive. Abstraction guided simulation is a promising semi-formal approach which formally analyzes a simplified abstract version of the design and uses it to guide the simulation towards the target property to be verified in the concrete design.

1.2 Contribution of this thesis

It is clear that no single functional verification approach works best for all designs. We believe that the unified scheme that combines the merits of both simulation based approach and formal verification is a promising approach in assuring a satisfactory verification of complex large designs. The contribution of this thesis includes:

- *New approach for extracting subset of unreachable/don't-care space for sequential equivalence checking.* We propose the use of data mining to mine missing patterns from the mining database consisting of a subset of the entire reachable state space of the design. The missing patterns are potential candidates for the unreachable state space (illegal states). Our approach is a low-cost and effective strategy to compute these

illegal states without requiring any computationally expensive state space traversal. We also propose heuristics based on equivalent flip-flop pairs and grouping of flip-flops to further reduce the computational cost of these constraints. These constraints are very effective in pruning a large portion of the don't-care space and can be very critical in verifying the equivalence between the two designs.

- *New approach for extracting non-redundant complex Boolean expressions as constraints for sequential equivalence checking.* We propose the use of data mining algorithm to capture complex yet powerful Boolean relations that are potential candidates for legal constraints. Those potential candidates verified as true are a representative of the correct functionality of the design under verification. Adding them as constraints during the formal verification approach can even prove a property as a true invariant or reduce the time and memory needed to verify the property that might not be possible without adding these constraints. We also use the structural information of the design along with post-processing of the mined relations to discard some of the constraints that might be trivial or redundant.

1.3 Outline of the thesis

The rest of this thesis is organized as follows: The next chapter gives the preliminaries of various functional verification techniques such as: Boolean Satisfiability and logic implication. Data mining algorithms are also introduced. How mining techniques can be applied to discover multi-node missing patterns and complex non-redundant Boolean expressions as global invariants is also described. These global invariants are then applied to unbounded sequential equivalence checking are presented in Chapter 3 and Chapter 4. In these two chapters, we also present certain heuristics based on the structural information of the de-

sign that can help in reducing the computational cost of extracting these constraints and discarding some of them such that we obtain powerful yet as small a subset as possible of these functional constraints. Conclusions and future work are given in Chapter 5.

Chapter 2

Background

2.1 Boolean Satisfiability (SAT)

Boolean satisfiability (SAT) is the constraint satisfaction problem with a wide range of applications in the fields of computer-aided design and artificial intelligence. Given a propositional formula f that depends on set of Boolean variables V, the Boolean satisfiability problem is to determine whether there exists an assignment on these variables V that can make this Boolean formula true. If there exists no such assignment then the formula is *unsatisfiable*. Otherwise, the formula f is *satisfiable*. The SAT problem was known to be a NP-complete problem few decades ago (1).

Conjunctive Normal Form (CNF) is the most widely used format for a propositional formula f in most of the modern SAT solvers. In the CNF format, the Boolean expression is represented as a conjunction of one or more clauses. Each clause is a disjunction of one or more literals. A literal is a variable or its negation. A CNF formula is satisfiable if all the clauses

are true simultaneously. For instance, given a CNF formula

$$f = (a)(\bar{a} + b + c + \bar{d})(c + d)(d)$$

one of the *satisfiable* assignments on variables a, b, c, d is $a = 1, b = 0, c = 1, d = 1$. There exist polynomial algorithms (2) that can convert an arbitrary propositional formula into an equivalent CNF formula. The equivalent CNF formula is satisfiable only if the original propositional formula is satisfiable. Similarly, a Boolean circuit can be converted to an equivalent CNF formula (3). The CNF representations for some of the basic gates in a circuit are summarized in Table 2.1, where i_1 and i_2 are the inputs and o_1 is the output of the gates.

Table 2.1: Conjunctive Normal Form of basic gates

Gate Type	Conjunctive Normal Form
AND	$(\bar{o_1} + i_1)(\bar{o_1} + i_2)(o_1 + \bar{i_1} + \bar{i_2})$
OR	$(o_1 + \bar{i_1})(o_1 + \bar{i_2})(\bar{o_1} + i_1 + i_2)$
NAND	$(o_1 + i_1)(o_1 + i_2)(\bar{o_1} + \bar{i_1} + \bar{i_2})$
NOR	$(\bar{o_1} + \bar{i_1})(\bar{o_1} + \bar{i_2})(o_1 + i_1 + i_2)$
XOR	$(\bar{o_1} + i_1 + i_2)(\bar{o_1} + \bar{i_1} + \bar{i_2})(o_1 + \bar{i_1} + i_2)(o_1 + i_1 + \bar{i_2})$
NOT	$(o_1 + i_1)(\bar{o_1} + \bar{i_1})$

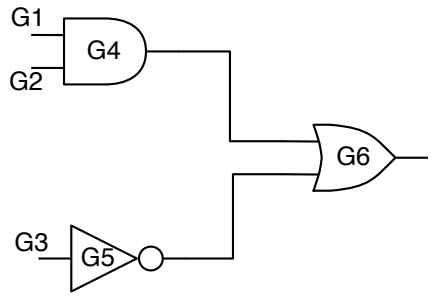

Figure 2.1: Simple Circuit

Figure 2.1 shows a simple logic circuit which can be represented as the following CNF formula

based on the rules listed in Table 2.1: $(\bar{G}4 + G1)(\bar{G}4 + G2)(G4 + \bar{G}1 + \bar{G}2)(G3 + G5)(\bar{G}3 + \bar{G}5)(G6 + \bar{G}4)(G6 + \bar{G}5)(\bar{G}6 + G4 + G5)$. The number of clauses in the CNF formula is linear to the number of gates in the circuit and the translation can be done very efficiently. Most of the modern SAT solvers are based on Davis-Putnam-Logemann-Loveland(DPLL) algorithm [(4),(5)] which performs decision tree branching along with backtracking on the variables to solve the SAT problem. It finds a satisfiable solution if and only if the formula is satisfiable else it concludes that no such assignment exists. The algorithm is sound and complete in itself. However, introduction of efficient conflict analysis, non-chronological backtracking and optimized Boolean constraint propagation (BCP) with two-literal watching [(6),(7)] have enhanced the performance of present day solvers significantly.

The basic algorithm for DPLL-based SAT solver is shown in Algorithm 1 in Figure 2.2. It is a search procedure that implicitly traverses the space of 2^n possible binary assignments to the problem where n is the number of variables. In the preprocessing step, if it finds that the formula is unsatisfiable then it is done else it picks an unassigned decision variable v_i and sets it to truth value (either 0 or 1) where v_i may be a necessary assignment. It asserts the implications of the decision (decide-next-branch) and if no such variable exists then it is satisfiable and a solution is found. Otherwise, the variable assignments deduced by the decision are made through Boolean constraint propagation (BCP) using deduce. It consists of the iterative application of the unit clause propagation rule where a clause becomes a unit clause when it only contains one unassigned literal and this clause can be satisfied by assigning the necessary value to make the literal true. Thus, no choice is necessary and saves a large part of the search space. A conflict occurs when a variable is implied both as true and false. If no conflict is discovered during BCP then the outer while loop is repeated by choosing the next decision variable. However, if a conflict occurs then backtracking is performed within an inner while loop to undo the decisions and their implications and backtrack

to the decision level that is the cause for the conflict. Backtracking also analyses the conflict and derives the conflict clause that will prune the search space by constraining the conflict space and avoiding reaching the same conflict again. However, if we backtrack to the root decision level due to the conflict, then the formula is concluded as unsatisfiable since the entire search space has been exhausted. Otherwise, we proceed until we find an assignment to all the variables in the formula that will make it satisfiable.

Algorithm 1: DPLL-based SAT solver

```
1  sat_solve(){
2      if (preprocess() == CONFLICT) then
         return UNSAT ;
       end
3      while (true) do
         /*Pick an unassigned variable as a decision                    */
4        if not decide_next_branch() then
5            return SAT ;
         end
         /*Find its implications and see if there is a conflict        */
6        while (deduce() == CONFLICT) do
           /*Add conflict clause & backtrack to that level             */
7          blevel ⇐ analyze_conflict() ;
8          if (blevel == 0) then
9              return UNSAT ;
           end
10         backtrack(blevel) ;
         end
       end
   }
```

Figure 2.2: DPLL Algorithm

The efficiency of a SAT solver depends on the variable ordering chosen and various variable

ordering heuristics have been investigated to pick a free decision variable depending on the application of SAT solver. We refer the reader to (8) for a survey on SAT solving and further information on SAT solvers. In recent years, SAT solvers have improved significantly and many tools such as zChaff (9), BerkMin (10), and MiniSat (5) have been developed which can efficiently handle large industrial designs. Also, a number of SAT solvers [(5), (11)] are available in the public domain for research purpose. We further refer the reader to a survey (12) on the recent advances on using SAT solvers for formal verification. In recent years, significant amount of research work has been conducted on using the aforementioned solvers for Model Checking and Equivalence Checking. We will discuss these works in detail in the subsequent sections.

2.2 Logic Implications

Logic implications capture the effect of asserting logic values (either 1 or 0) on signals throughout the circuit. Implications capture relationships among two or more signals in the circuit and can be viewed as constraints which can potentially help to constrain the search for several applications in electronic design automation (EDA) problems such as design verification (13), multi-level logic optimization (14), etc.

There has been a significant amount of work in computing static implications among signals. Static implication is a procedure of setting a gate to a logic value (either 0 or 1) and analyzing the effect of propagating these values throughout the circuit. Static implications consist of direct, indirect and extended backward (15) implications. Direct implications can be easily determined whereas indirect, extended backward are less trivial and finding them requires combination of logic simulation, application of the contrapositive and transitive laws. In general, the total number of implications can be exponential in the size of the circuit. Thus,

a memory-efficient technique must be used to store these implications associated with each gate. Zhao et al. (16) proposed a graphical representation to store the implications in a sequential circuit where implications between gates across different time frames can be easily modeled. For a circuit with k gates there are $2k$ nodes in the graph since each gate can have a logic value of 0 or 1. A directed edge between two nodes represents an implication between them. For instance, an edge from node a to node b means that a implies b ($a \rightarrow b$). The weight on the edge represents the relative time frame associated with the implication where the current time frame is 0. If an implication propagates across a D flip-flop, the time frame is incremented or decremented accordingly. Transitive closure of a node can be easily computed and contrapositive implication can be easily added to the implication graph. We use the following notation Impl[g,v,t] to represent the set of implications for gate g with value v in time frame t.

2.2.1 Direct Implications

Direct implications of gate g consist of implications associated with gates driving and driven by g and are computed by traversing through the immediate fanins and fanouts of gate. The direct implications are of two types: direct forward implications, direct backward implications. To compute direct forward implications, a controlling value (cv) at any of the gate $g's$ fanins implies a value of ($cv \oplus i$) at the gate output, where i is the inversion value of the gate. Similarly, to compute direct backward implications, a value of ($ncv \oplus i$) at the gate $g's$ output implies ncv at all the gates fanins. Non-controlling value is just the complement of the controlling value. Table 2.2 gives the controlling values, non controlling values and inversion values of basic gates. The direct implications can be learnt during an ATPG process.

Given the example circuit in Figure 2.3, a logic value of $D = 1$ implies $E = 0$, $F = 1$. It also

Table 2.2: Controlling, Non controlling and Inversion values of basic gates

Gate Type	CV	NCV	I
AND	0	1	0
OR	1	0	0
NAND	0	1	1
NOR	1	0	1

implies $C = 1$ and $B = 1$. The direct implications for $D = 1$ is the set $(E = 0, F = 1, C = 1, B = 1)$. Similarly, the direct implications of $E = 0$ are $G = 0$ and $D = 1$. These direct implications form the root $D = 1$ are stored in the implication graph as shown in Figure 2.4.

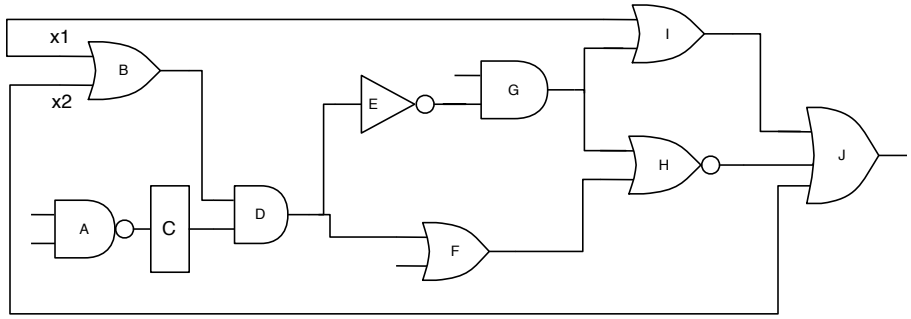

Figure 2.3: Example of sequential circuit

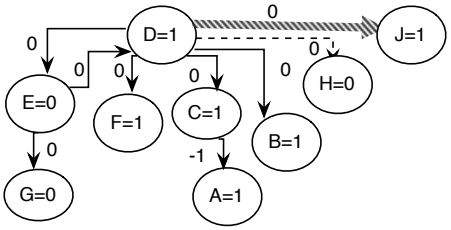

Figure 2.4: Implication graph

2.2.2 Indirect Implications

Indirect implications of a gate g with value v are computed by inserting all the current implications of the gate and then performing logic simulation on these inserted events. All the gates whose logic value changes from don't-care to either 0 or 1 are the indirect implications of gate $g = v$. In Figure 2.3 consider $D = 1$, by inserting the current implications of $D = 1$ into the circuit and performing logic simulation, we obtain $H = 0$ as an additional implication which cannot be obtained via either $G = 0$ or $F = 1$ alone. So, $D = 1$ implies $H = 0$ indirectly. This implication is added as an additional outgoing edge (dotted edge) from $D = 1$ in the implication graph shown in Figure 2.4.

2.2.3 Extended backward Implications

The extended backward implication extends the concept of indirect implications to unjustified gates in the circuit to obtain more signal relationships. The extended backward implications aim at relations of the implying gate with respect to the unjustified gates in its implication list. In Figure 2.3 gate B is an unjustified gate since none of inputs is currently implied to logic 1. Thus it is a candidate for extended backward implications. First the transitive closure for each of the unspecified inputs of gate $B = 1$ is done obtaining $Impl[x1, 1, t]$ and $Impl[x2, 1, t]$. Then the implications of $D = 1$ is simulated with $x1 = 1$ and similarly the implications of $D = 1$ is simulated with $x2 = 1$. For instance, the implications of $D = 1$ and $x1 = 1$ are $I = 1$ and $J = 1$. Similarly, the implications of $D = 1$ and $x2 = 1$ are $J = 1$. All the assignments such as $J = 1$ common to these sets are the extended backward implications of $D = 1$ and an edge (thick edge) is added in the implication graph in Figure 2.4 to represent the same.

2.2.4 Other kinds of implications

There have been some other interesting works on computing implications. A complete implication engine based on recursive learning (17) proposed by Kunz et al. can capture all pair-wise relationships in a circuit. However, the depth of recursion is kept low for large circuits to avoid excessive computational costs. Due to the NP-hard nature of finding all the implications for a given set of nodes, the practical use of these complete algorithms is limited. Syal et al. proposed extended forward implications (18) that uses implication frontiers to capture additional pair-wise implication relationships with low computation costs. To further the application of implications, static learning has been extended to dynamic learning (19).

2.3 Data Mining

The increasing computing power and storage has led to tremendous increase in collection of data. The increasing sizes of these datasets have made direct hands-on data analysis infeasible and call for more indirect, automated approaches for data processing. Methods such as neural networks, genetic algorithms, clustering, support vector machines, itemset mining, etc. discovered in computer science can assist in automating the process. Data mining is the process of applying these methods to extract useful information and hidden patterns from (usually large) observational information rich data sets. The extracted information is of great interest to many fields including science, engineering and business. The discovered knowledge can be in the form of model, pattern or derived values and relations from a given collection of data. Some of the examples could be rules, graphs, clusters, decision trees and frequent itemsets. The generic process of applying data mining to a problem involves the following steps:

- State the problem and formulate the hypothesis.
- Collect the data.
- Preprocess the data.
- Estimate the model.
- Verify the model.
- Interpret the model and draw conclusions.

Data mining techniques can be classified as below:

Structured data analysis is the statistical data analysis of structured data that consist of the typical techniques such as Bayesian inference (20), logistic regression (21), ANOVA analysis (22).

Cluster Analysis is the assignment of grouping observations into clusters such that observations in the same cluster are similar in some way. Commonly used techniques are agglomerative algorithms (23), partitional clustering (24), and incremental clustering (25).

Itemset Mining and Association Rules is the discovery of interesting relations among variables in large datasets and includes algorithms such as market based analysis, Apriori algorithm (26), BLOSOM (27), Charm (28).

Decision Tree Mining and Decision rules uses decision tree as a predictive model to map the observations about an item to predictions or conclusion about the value of the target variable. Some of the techniques are the CLS method (29), the ID3 algorithm (30), the C4.5 algorithm (31) and the corresponding pruning algorithms.

Artificial Neural Networks where the emphasis is on multilayer perceptrons with back-propagation learning and Kohonen networks (32).

Genetic Algorithms are useful as a methodology for solving optimization problems (33).

Fuzzy Inference Systems are based on the theory of fuzzy sets and fuzzy logic. Fuzzy modeling and decision making are steps very often included in data-mining process (34).

N-dimensional Visualization Methods are typical data mining visualization techniques including geometric, icon-based, pixel-oriented, and hierarchical techniques (35).

2.3.1 Itemset Mining and Association Rules

In data mining, association rule learning is the most common form of local pattern discovery; it finds interesting relations between variables that cannot be explicitly articulated in a database such as how often two or more objects co-occur or conditionally occur. In other words, it retrieves all the highly correlated relations in the database. It was introduced by Agrawal et al.(36) for discovering regularities between products in large scale transaction data recorded by point-of-scale (POS) systems in supermarkets. For example, the rule $[onions, potatoes] \rightarrow [tomatoes]$ indicates that if a customer buys onions and potatoes then he or she is likely to buy tomatoes too which may prompt grocery store to co-locate tomatoes close to onions and potatoes aisle. Apart from market based analysis association rules are employed in many application areas including web usage mining, intrusion detection and bioinformatics.

Several notations commonly used in itemset mining are:

1. *item:* The basic element or attribute in a database, e.g., an item sold in a supermarket.

2. *itemset:* The set of binary valued attributes or items in a database, e.g., collection of all products sold in the supermarket. Mathematically, $I = \{I_1, I_2, \cdots, I_m\}$.

3. *k-itemset:* An itemset of cardinality (size) k is called a k-itemset. In other words, $\{I_1, I_2, \cdots, I_k\} \subseteq I$ where $k \leq m$

4. *transaction:* Each row in the database D corresponds to a transaction with a unique id. It contains a subset of all the items. For e.g. in $D = \{t_1, t_2, \cdots, t_n\}$, where $t_j \subseteq I$, $t_1, t_2, \cdots, t_n$ are the unique transaction ids.

5. *basket:* A set of items, e.g., the things a customer buys.

6. *support for k-itemset:* The number of baskets containing all the items in a k-itemset.

7. *frequent itemsets:* Given a support threshold, itemset that appears in $\geq$ *threshold* baskets.

Table 2.3 shows an example of database D consisting of total six items A, B, C, D, E, F and four transactions. A '1' in the table indicates that the item in that column occurred in that transaction. For example, the 2-itemset $[B, E]$ occurs in transactions 2 and 3, then the support of $[B, E]$ is 2. If the support threshold is set to 1, then $[B, E]$ is a frequent 2-itemset since more than 1 baskets contains $[B, E]$.

Table 2.3: Database D

Transaction ID	A	B	C	D	E	F
1		1		1	1	
2	1	1	1	1		
3	1			1	1	
4		1	1	1	1	

For the purpose of association rule generation, it suffices to find all frequent itemsets. The search space of all itemsets can be represented by a subset-lattice. Figure 2.5 shows an example lattice over the set of items $I = \{x1, x2, x3, x4\}$. The bold line is an example of actual itemset support and separates the frequent itemsets in the upper part from the infrequent ones in the lower part. The task of discovering all frequent itemsets is quite challenging

because the search space is exponentially growing with $|I|$. Therefore it is impractical to calculate whether it is frequent for each subsets of I.

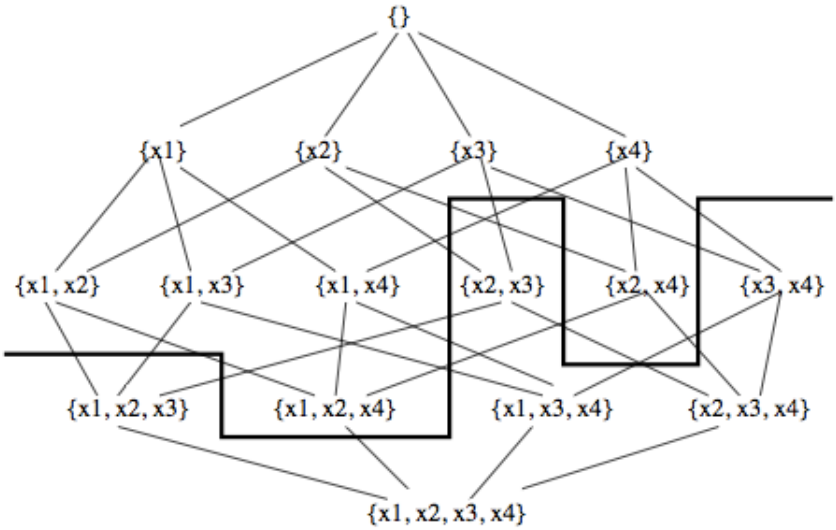

Figure 2.5: The lattice for the itemsets I

Instead, the downward closure property of itemset support is employed by most association rule mining algorithms to traverse the lattice as little as possible.

Theorem 1: (Downward Closure Property) Given a transaction database D over I, let $X, Y \subseteq I$ be two itemsets. Then $X \subseteq Y \Rightarrow support(Y) \leq support(X)$.
Hence, if an itemset is infrequent, all of its supersets must be infrequent and similarly if an itemset is frequent then all its subsets are frequent. With this pruning, the search space is drastically reduced.

A novel framework for Boolean pattern mining, called BLOSOM (27) can compute frequent Boolean expressions over binary valued datasets involving co-occurrences of certain events in the database. The details of BLOSOM are provided in the subsequent section.

2.3.2 BLOSOM

Let T denote the set of transaction identifiers and let I denote a set of items. Given a transaction $(t, t.X) \in D$, where $t \in T$ and $t.X \subseteq I$, then a transaction t satisfies an item/literal $i \in I$ if $i \in t.X$ and similarly t satisfies the negation of a item/literal $\bar{i}$ if the item $i \notin t.X$. For a literal l, the truth value of l in a transaction t is defined as $V_t(l)$ where

$$V_t(l) = \begin{cases} 1 & \textit{if } t \textit{ satisfies } l \\ 0 & \textit{if } t \textit{ does not satisfy } l \end{cases} \tag{2.1}$$

A Boolean expression E is the logical AND or OR of one or more clauses where each clause is either the logical AND or logical OR of one or more literals. A transaction t satisfies a Boolean expression E if the truth value of E denoted as $V_t(E)$ evaluates to true when each literal l in E is replaced with $V_t(l)$. The support s of a Boolean expression E in the dataset D is the number of transactions that satisfy E, i.e $|t(E)|$. A Boolean expression E is frequent if its support is more than or equal to a user specified minimum support (min_sup) value, i.e. $|t(E)| \geq min\_sup$. One can also define a maximum support threshold (max_sup) to disallow any expression with too high a support.

For example, in dataset D in table 2.4, there are a total of five items I = {A, B, C, D, E} and four transaction ids T = {1, 2, 3, 4}. The transaction t_1 contains the set of items {A, C, D} which is a subset of set of all the items in the dataset. The transactions t_1 and t_3 satisfy the Boolean expression $\{A \wedge C \wedge D\}$ since $V_t(E)$ evaluates to be true in both the transactions. Also, if the min_sup is 2 then this Boolean expression is a frequent Boolean expression.

BLOSOM (27) allows for mining frequent Boolean AND/OR/CNF/DNF expressions and

Table 2.4: Dataset D

t_{id}	set of items
1	ACD
2	BC
3	ABCD
4	ADE

their minimal generators. A set (x) is called a minimal generator of a closed set (y) if it satisfies the two properties:

1. The *closure* of set (x) is set (y).

2. No proper subset of (x) generates (y).

The closure operator on partially ordered set (P) is a function $C : P \rightarrow P$ such that C is monotone, idempotent, and extensive. A set (y) is considered closed with respect closure operator (C) if it satisfies the property $C(y) = y$.

The minimal generators can be viewed as succinct descriptors of data. When more than one minimal generator exists for the same subset of data, we refer to them as redescriptions (37). Redescriptions can be thought of as a generalization of association rule mining from finding implications to equivalences. These equivalences form the potential inductive invariants for the sequential equivalence checking problem. For example, $\{A, C, D\}$ is a closed AND clause in Table 2.4 and the minimal generators for this closed expression are $\{A, C\}$ and $\{C, D\}$. So, they can be represented in terms of redescription (equation 2.2):

$$A \wedge C \equiv C \wedge D \tag{2.2}$$

Similarly, $(A \vee D) \wedge (B \vee E)$ is one of the closed Boolean expressions for the example in Table 2.4. The minimal generators for this closed CNF clause can be represented in terms

of redescription (equation 2.3):

$$A \wedge (B \vee E) \equiv D \wedge (B \vee E) \tag{2.3}$$

2.4 SAT-based Model Checking

Model Checking refers to the following problem: Given a model of a system (or design), check whether this model meets the given specification or not. The specification is usually the set of properties that need to be verified for the design. For example, in a Traffic Light Controller, we need to verify the property that the lights on perpendicular roads are not green at the same time. In model checking [(38), (39)] the design to be verified is usually modeled as a Kripke structure or finite state machine and the properties are written in temporal logic. The reachable states of the design are traversed from an initial state to verify these properties formally.

Early model checking algorithms are explicit in the sense that they work directly by traversing the state transition graph (STG) of the design. However, with increasing design size, the constructing the STG representing all the reachable states in the design is impractical since the states are exponential to the number of state elements. Thus the practicality of explicit model checking is limited. Symbolic Model Checking (SMC) was introduced in (40), to efficiently automate the model checking process without needing explicit state space traversal. It uses Boolean encoding to represent the Kripke structure/finite state machines and the set of states. By replacing the transition relation and set of states with BDD, SMC can handle large designs. Boolean function manipulations can be performed in linear time with respect to the size of BDDs. However, an ordered BDD is sensitive to the variable ordering and explodes if bad variable ordering is chosen. Due to this symbolic techniques are limited

to small/medium sized designs. Thus, for very complex systems, formal verification often suffers from either state explosion problem or requires exponentially long time to finish. The reason is that these problems are the NP-complete problem. And the worst case complexity of NP-complete problem is exponential.

However, due to the availability of high performance SAT solvers, SAT based techniques for model checking have gained industry wide acceptance over symbolic techniques. All the properties that need to be verified are converted to propositional Boolean formula, which is checked for satisfiability/unsatisfiability using a SAT solver. Depending on the nature of model checking approaches they can be classified as (i) SAT-based Unbounded Model Checking (ii) SAT-based Model Checking (iii) SAT-based Induction. We will detail SAT-based Induction in the next section and discuss briefly about SAT-based Unbounded and Bounded Model Checking.

2.4.1 SAT-based Induction

In SAT-based Unbounded Model Checking (UMC), a SAT solver is employed to perform either forward or backward reachability analysis of the design to verify its correctness with respect to the property ϕ. Let ϕ be the property to be checked and let S_0 be the set of initial states of the model M. In property checking, we want to verify if $M \models \phi$. If there exists a path from any initial state to a state that satisfies $\neg\phi$, then we know that M does not satisfy the property ϕ. However, if the property is a true invariant then no counter-example will be obtained and the process of computing the set of states is continued until a fixed point is reached.

In SAT-based Bounded Model Checking (BMC), we search for a path of bounded length between S_0 and those states satisfying $\neg\phi$. To perform this, a Boolean formula is constructed

by unrolling the circuit into several time frames together with a monitor circuit. This constructed formula is satisfiable if and only if the underlying transition system can realize a bounded sequence of state transitions that exposes a path from S_0 to $\neg\phi$. However, if the property is a true invariant then there exists no such sequence of states that can violate the property and thus the bounded length keeps on increasing.

In contrast to UMC and BMC, Induction [(41), (42)] is a technique that aims at proving invariant properties without performing state-space traversals. For example, in (41), some internal signals are assumed as equivalent in initial clock cycles and it is verified if they are equivalent in remaining clock cycles. Those proved true are considered as inductive invariants and adding them as constraints can reduce the search space of the SAT solver when proving the two circuits equivalent. In SAT-based induction (43), the base case of induction, $S_0 \rightarrow \neg\phi_1$ (ϕ_1 means that the property ϕ is monitored in the 1^{st} time frame) is translated to a Boolean formula and checked for satisfiability using a SAT solver. If the SAT solver returns satisfiable then the property is not a true invariant. However, if is unsatisfiable, then the inductive step is performed where $\phi 1 \wedge T1 \rightarrow \phi 2$ is translated to Boolean formula $\phi 1 \wedge T1 \wedge \neg\phi 2$. Here $T1$ is the transition relation of the finite state machine for 1 time frame. If it is unsatisfiable then ϕ is a true invariant. Otherwise nothing can be concluded about ϕ. For most of the properties this simple induction method is not sufficient and thus a stronger induction scheme called induction with depth can be used. In this new scheme, the base case is modified to $S_0 \wedge T*(1,2,\cdots,k) \rightarrow \phi*(1,2,\cdots,k)$, and the induction step is modified to $\phi*(1,2,\cdots,k) \wedge T*(1,2,\cdots,k+1) \rightarrow \phi_{k+1}$, both of which are again converted into respective CNF formulas. If nothing can be concluded about ϕ at a depth k, it is increased to a higher depth k' and the SAT checks are repeated.

In general both SAT-based UMC and BMC based approaches are good for refutation of a property by finding short counter-examples. However, SAT-based induction is good for

proving property as a true invariant at smaller depths.

2.5 SAT based Sequential Equivalence Checking (SEC)

Equivalence Checking plays an important role in the design verification process. During modifications/optimizations to the original design for optimizing power, delay, area, etc. bugs can be introduced in the design. The objective is to verify if the original design and optimized design have the same functionality. Currently, available tools can verify the equivalence of RTL models and transformed net-list models after logic synthesis. Let $S1$ be

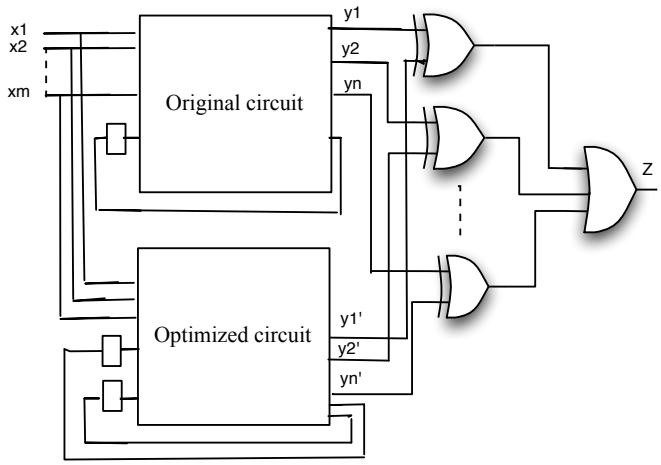

Figure 2.6: Miter Model

the original sequential design with m primary inputs $x1, x2, ...xm$ and n primary outputs $y1, y2, ...yn$. Let $S2$ be the optimized design with primary outputs $y1', y2', ...yn'$ obtained from $S1$ after sequential optimizations such as retiming, different state encodings, state minimizations, etc. To verify $S1 \equiv S2$ a miter circuit is constructed as shown in Figure 2.6. In the miter circuit, all the primary inputs of $S1$ are tied to $S2$ and the corresponding primary output pairs from $S1$ and $S2$ are tied together to an XOR gate. The output of all

the XOR gates are fed to an OR gate. The objective of the sequential equivalence checker is to verify if the output of OR gate (Z) is a tautology 0 (i.e. all the outputs of two circuits are the same) for all states reachable from the alignment state. In other words, $Z = 1$ is unachievable.

Sequential equivalence checking is considered EXP-space complete (44) whereas combinational equivalence checking is known to be a coNP-hard problem (45). This is because instead of simply checking the combinational logics for equality, verifying sequential equivalence involves verifying the equality of outputs in every reachable state of the product-machine of both the implementation and the golden models.

There exist several definitions of sequential equivalence checking which differ predominantly in the assumption(s) made with respect to the operation of sequential circuit and the surrounding environment in which the design might be embedded. For circuits with reset states, there exists a notion of reset equivalence where two circuits $S1$ and $S2$ are equivalent iff the two circuits are equivalent after reset. However, this definition is a strict and narrow form of equivalence and requires that a global reset sequence for both circuits exists.

For circuits without reset, different notions of sequential equivalence are possible such as sequential hardware equivalence, safe replaceability, and 3-valued safe replaceability. Pixley proposed the notion of Sequential hardware equivalence (SHE) (46) where two designs are said to be equivalent if there exists a universal alignment sequence that takes the two circuits to an equivalent state pair. The theory of SHE is motivated by the desire to prove two designs equivalent without reference to any intended environment, presence of reset states or reset sequences. Thus, the assumption is that there exists at least one alignment sequence under which the two circuits are being verified. We assume that the two circuits under verification can be brought to a known initial state by applying some synchronizing sequence that can take the circuits to a unique, specified state.

In the past few years, several methods have been proposed to solve the problem of SEC. Binary decision diagrams (BDDs) (47) have been used for various symbolic techniques [(48), (49)]. In these approaches a product machine of the two sequential circuits under verification is constructed and the output of the product machine is verified under all reachable states. A fixed point computation is conducted starting from the initial state to compute all the reachable states. These symbolic finite state machine (FSM) traversal techniques are possible only for small to medium sized circuits but become expensive and vulnerable to memory explosion for large designs with hundreds to thousands of registers.

To reduce the complexity of SEC, structural similarity used earlier for identifying internal equivalent signal pairs for incremental verification of combinational circuits have been explored for finite state machines [(50), (51)]. Sequential automatic test pattern generation (ATPG) is used to identify some equivalent flip-flop as well as internal signal pairs together with an induction based proof. These equivalent pairs are then used as constraints to reduce the search space. For portions of designs with different state encoding and not much similarity, symbolic FSM traversal techniques based on fixed point calculation are then applied, together with the constraints learned earlier. Thus, incremental verification and FSM traversal are integrated to verify designs that cannot be verified by each of these approaches separately. However, if the state space corresponding to different state encodings is large, such symbolic FSM traversal techniques may still blow up in both memory and time. In (51), a greatest fixed point iteration is used to identify functionally equivalent signals without explicit symbolic state space traversal. The method uses a two time frame assume-then-verify model. However, with increasing use of sequential optimization techniques, the probability of finding equivalent flip-flops, internal equivalent signals between two designs will reduce, leading to a large don't-care space and increasing problem complexity.

Lu & Cheng (52) proposed a framework for sequential equivalence checking based on k^{th} invariants. The framework has three complementary components: a bounded model checker

to identify and merge k^{th} invariants for circuit simplification. An inductive invariant component to find the true k^{th} invariants. If one of the true k^{th} invariant is the output of the miter circuit that is 1, then circuits are equivalent. If not, then these k^{th} invariants are added as constraints to prune the search space of the Seq-SAT.

Another approach is using logic implications to capture relationships among signals in the circuit. Besides computing static and dynamic implications, methods have been proposed to extract relationships among two or more variables in the form of multi-variable relationships. But computing relationships among multiple signals may require an exponential number of combinations and thus methods requiring low computational cost have been investigated. A method for finding static multi-node implications among node pairs based on circuit structure information (53) has also been proposed.

Recently, an approach based on data mining has been proposed (54) which mines implications among internal signals in the circuit. It uses Apriori (26), an efficient and popular methodology for association rule mining to compute potential three node invariants among signals in the database. To further prune the search space of mining process, domain knowledge of circuit is used to improve the quality of discovered rules (55). In (56) cross time frame constraints among flip-flops are learned using support-confidence notion from data mining for speeding up SAT-based BMC for equivalence checking. These static and dynamic multi node implications can significantly enhance bounded sequential equivalence checking.

We note that SEC is a special case of SAT-based induction where the property is to verify if the output of miter circuit is a constant '0' starting from any reachable state. To benefit from a SAT engine, we regard the problem setup for SAT-based equivalence checking as the following: Similar to SAT-based induction, the miter is unrolled for k time frames. The unrolled miter circuit is first translated into a Boolean formula, with its initial state unconstrained, and the objective is to see if the miter output can be set of '1'. We note that

inductive analysis can be applied in this setup so that only the miter output in the last time frame makes up the objective. If the formula is unsatisfiable, there exists no sequence of states (starting from any arbitrary state) that can make the miter output '1' and thus the two circuits are equivalent. If the formula is satisfiable, then one cannot conclude that the two circuits are not equivalent because it could be a false counterexample. In other words, due to the large don't-care space of the system, there may exist a counterexample that can violate the property from some arbitrary initial state and thus one cannot conclude whether the property holds or not. Thus, if nothing can be concluded at a depth k, it is increased to a higher depth k' and the SAT checks are repeated until an upper bound on k is reached.

Chapter 3

Mining Multinode Constraints for Sequential Equivalence Checking

In this chapter, we present a novel technique on mining relationships among multiple signals in a sequential circuit to discover a subset of global illegal constraints. We first utilize the true equivalent flip-flop pairs found to reduce the number of columns in the mining database and thus lower the computational cost of the mining process. For circuits with a large number of state elements, we divide them into smaller groups of highly correlated state elements based on their input support. We then perform mining on each of these smaller groups separately to keep the computation cost low. Next, an invariant checker based on assume and verify is used to validate the potential inductive invariants. Experimental results demonstrate that the application of these global illegal constraints can prune a large portion of the don't-care space and prove some of those circuits equivalent that cannot be verified using SAT based induction along with equivalent flip-flops and internal equivalent signals alone.

3.1 Motivation

The main challenge of SEC is the lack of knowledge about the illegal state space of the design under verification. This limitation manifests itself in two ways. Firstly, since the underlying SEC engine often uses a branch-and-bound procedure, the engine may make decisions on variables that are functionally not possible (correspond to the don't-care space) in the design. This problem becomes critical in the induction step since the initial state is not constrained to any reachable state and the solver might return a sequence of illegal states as a counterexample to invalidate the true invariant. Hence, the knowledge of some (if not all) illegal state space might be extremely helpful in pruning the search space. Second, if the formula for the main inductive step at depth k is satisfiable, then the satisfying solution forms a simple path $s_1, s_2, \cdots, s_k$ where the first $k-1$ states satisfy the property (miter output = 0) and the last state s_k does not. In such a situation there are two possibilities: (i) the sequence of states returned by the SAT solver are reachable states and thus it is a true counter-example. In other words, the two circuits are not equivalent; (ii) the sequence of states is illegal in which case the counter-example is a spurious counter-example and nothing can be concluded about the equivalence of two circuits. Generally, it is hard to determine whether or not a state is a reachable state. Moreover, for designs with complex optimizations such as different state encodings, equivalent flip-flop pairs and internal signals including pair-wise implications might not be sufficient in blocking the false-counterexample.

Due to aforementioned reasons, induction-based methods can be improved by learning as many unreachable states as possible with limited computational cost and adding them as constraints throughout the unrolled transition relations during the inductive runs to prevent false counterexamples. This in turn allows for more aggressive SEC optimizations. Since computing all the reachable states or unreachable states in a design might be expensive (or infeasible), we try to obtain a partial set of these illegal states with as many unspecified bits

as possible to cover a large portion of don't-care state space using mining approach.

3.2 Related Work

Identification of illegal states has been studied in the context of sequential ATPG [(57), (58)] via the identification of partial assignments to state variables that cannot be justified. In (57), illegal states are specified by the values of state variables, and it tries to merge illegal state space to larger cubes. In (58), three-value simulation is used to traverse the state space from an unknown initial state; all the valid states are recorded.

Illegal states can also be treated as functional constraints for pseudo-functional testing [(59), (60)]. Subsequently, these constraints are fed to, preferably a combinational ATPG engine which generates tests that avoid the illegal states specified. In (61) they show that the main reason for illegal states is the multi-fanout nets in the circuit and determine which state cubes can justify both logic '1' and '0' at the multi-fanout net independently. As per our understanding the expansion of illegal state cubes from n-bit to $(n+1)$ bit might be expensive for designs with large number of state elements. In [(62), (63)], implication-based learning for functional extraction was used. In (62), the illegal states are determined using implications learned during static learning in a preprocessing step. The illegal state cubes are saved in a list. The ATPG procedure derives a test cube, which does not contain any identified illegal state. In the work by Syal et al. (63), pair-wise and multi-node constraints are also identified via sequential implications. They demonstrate that sequential relationships are very useful for identifying untestable faults.

In (64), low cost ATPG-based illegal state identification technique is proposed for strengthening SAT-based induction runs during property checking. Also, work has been done to use the information obtained by the SAT solver in k^{th} inductive step for $(k+1)^{th}$ step. When-

ever, a counter-example (as explained above) is obtained, the authors in (65) assume s_1 to be an illegal state and obtain a strengthened property $(\phi \wedge \neg s_1)$. The idea behind it is to obtain the new property stronger then the original one and thus property might be proved at a smaller induction depth. However, if there are several false counter-examples then the new property might not be strong enough.

3.3 Proposed Approach

We propose the use of mining approach to compute a subset of those relationships among the state variables that are missing from the mining database. Since computing all (missing) patterns among all the variables in the mining database is computationally expensive and infeasible for circuits with large number of flip-flops, we aim to uncover a subset of these illegal constraints. Adding this subset of true illegal constraints that cannot occur during the normal functional mode of the miter circuit during SEC might be sufficient in proving the two circuits equivalent. These missing patterns are obtained only among state variables because the number of arbitrary nets is usually much larger than the number of state elements in the circuit. This also reduces the computation cost of mining these constraints. To the best of our knowledge, no work currently exists that employs data mining to capture global illegal constraints to solve a much harder problem of proving the circuits equivalent for the entire reachable state space (unbounded SEC). To further reduce the cost of mining we identify all the equivalent flip-flop pairs and use them to reduce the number of columns in the mining database without loss of useful illegal constraints. For circuits with large numbers of flip-flops, we group them using the MLP algorithm (66) and apply our mining approach on each of the highly correlated group of flip-flops independently.

Next, the mined relations are verified using an assume-verify based induction before they are

added as constraints in all the time frames of SAT-based equivalence checking framework. The reason for validating these mined relations is that they are obtained only from a subset of data being analyzed, which may not be true representatives of the complete reachable state space of the design. Thus, it might mine some constraints as illegal with respect to the current database. However, they might not be truly illegal since they may be part of the reachable state not present in the mining database. Our experimental results show that the proposed mining approach is useful for finding multi-node illegal constraints for up to 4 flip-flops. This is because the computational complexity of validating these relationships is exponential to the number of variables. However, the benefit of our approach is that it does not require any state-space traversal of the finite state machine for computing illegal states. Moreover, we are able to prove some of those benchmarks equivalent by adding these constraints, which cannot be proved by the ABC package (67) from Berkeley.

The rest of the chapter is organized as follows. Section 3.4 gives the details of mining approach and the heuristics using equivalent flip-flop pairs and grouping of flip-flops to reduce the mining cost. Section 3.5 presents the validity check of mined constraints using assume and verify. Section 3.6 gives its application to SEC. Section 3.6 discusses the experimental results, and Section 3.7 concludes the chapter.

3.4 Mining illegal constraints

First a miter is constructed from the sequential circuit and its optimized version. Then sequential logic simulation is performed on the miter circuit for M (vary from 10,000 to 100,000) random input vectors applied at the primary inputs to obtain the logic values on the flip-flops. These logic values are recorded in the mining database such that one dimension lists the flip-flops and the other dimension lists the input vector number (transaction id)

in the circuit. Each row in the database corresponds to a reachable state of the miter circuit. Unlike previous works on finding potential equivalent flip-flop pairs or true three node relationships from the database, we mine potential multi-node relationships among flip-flops *missing* from the database. These are potential candidates for illegal constraints in the miter circuit. We compute one-node, two-node, three-node and four-node constraints among flip- flops.

Table 3.1: Example of mining database DB

Vector#	f_1	f_2	f_3	$\cdots$	f_n
1	1	0	1	$\cdots$	1
2	0	0	0	$\cdots$	1
3	1	1	1	$\cdots$	0
4	1	1	0	$\cdots$	0
.	.	.	.	$\cdots$	.
m	1	0	1	$\cdots$	1

Consider the mining database DB in Table 3.1. To compute one-node illegal constraints, we compute the probability of getting a logic value of 0 ((P_i^0)) or 1 ((P_i^1)) on each flip-flop f_i in the database (equation 3.1 and 3.2)

$$P_i^0 = \sum_{i=0}^{M} (f_i = 0)/M \tag{3.1}$$

$$P_i^1 = \sum_{i=0}^{M} (f_i = 1)/M \tag{3.2}$$

where M is the total number of input vectors simulated. After computing (P_i^1) and (P_i^0), we check if the probability value is 0 for either logic '0' or '1' on the flip-flop. If 0, then it is considered as a potential illegal constraint to be verified later using assume and verify induction method. Similarly, for two node illegal constraints we compute the probability of getting each of the four combinations (2^2) for a pair of flip-flops. Thus, for n-node illegal

constraints, we compute the probability of getting each of the (2^n) combinations among flip-flops. For example, if we choose two flip-flops f_1, f_2 from the database then we compute the following four probabilities for these flip-flops (equation 3.3, 3.4, 3.5 and 3.6).

$$P_1{}^0 \& P_2{}^0 = \sum_{i=0}^{M} (f_1 = 0 \& f_2 = 0)/M \tag{3.3}$$

$$P_1{}^0 \& P_2{}^1 = \sum_{i=0}^{M} (f_1 = 0 \& f_2 = 1)/M \tag{3.4}$$

$$P_1{}^1 \& P_2{}^0 = \sum_{i=0}^{M} (f_1 = 1 \& f_2 = 0)/M \tag{3.5}$$

$$P_1{}^1 \& P_2{}^1 = \sum_{i=0}^{M} (f_1 = 1 \& f_2 = 1)/M \tag{3.6}$$

If the probability of getting $f_1 = 0$ and $f_2 = 1$ is 0, then "$f_1 f_2 = 01$" is a potential illegal constraint. Similarly, if the probability of getting a $f_1 = 0$, $f_2 = 1$ and $f_3 = 0$ is 0 then "$f_1 f_2 f_3 = 010$" is a potential three-node illegal constraint. It is observed that for some circuits, three-node illegal constraints are sufficient to prove them equivalent and for others four-node illegal constraints are needed to prove them equivalent. The complexity of computing (m) node illegal constraints for (n) flip-flops is $O(2^m)$ since we are looking for all possible missing combinations among m flip-flops in the database.

To further compute potential illegal constraints of size greater than four, a fast and effective approach based on automatic test pattern generation (ATPG) algorithm called PODEM (68) is used to find all the minimal necessary assignments at the pseudo primary inputs (PPI's) and the primary inputs (PI's) that can make the miter output 1 in one time frame. For example, consider the miter circuit in Figure 3.1 unrolled for one time frame (becomes a combinational circuit) and the output is constrained to 1. Then we backtrace starting from

the output until we reach an unassigned PI or PPI and assign it a logic value and repeat this process until miter output is equal to 1 or a conflict is obtained. In case, a conflict occurs we backtrack and flip the value of the current assignment at PPI or PI to find the next solution. If no solution exists for the both the logic values of the PPI or PI we backtrack to the previous assignments and do a backtrace to find the next solution. This process is repeated until no more solutions can be found. All the solutions obtained are considered potential illegal states. Next, all these potential states are validated using the invariant checker and added as constraints during equivalence checking. This method is exhaustive, fast and not restricted to three and four node constraints.

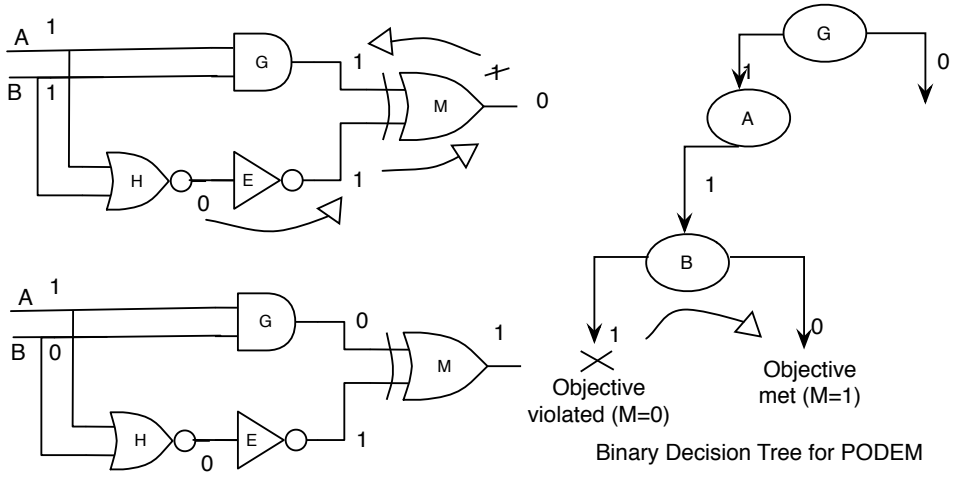

Figure 3.1: Example circuit for PODEM

We propose a heuristic in the next section that can minimize the count and subsequently the time to verify these constraints for both the mining and ATPG approach significantly as shown in Figure 3.2.

3.4.1 Heuristic for improving mining and ATPG approach

Based on the assumption that there exist some equivalent flip-flop pairs between the original and optimized designs, we utilize them to reduce the columns (flip-flops) in the database. For a miter circuit with n flip-flops $(f_i, f_j, f_k, f_l, f_m, \cdots f_n)$, we first find the potential constant flip-flops and equivalent flip-flop pairs and validate them using assume and verify based induction to find true constant and equivalent flip-flop pairs. The flip-flop is constant '0' or '1' if it always has either logic '0' or '1' respectively. Similarly, two flip-flops are equivalent if they either have the same or opposite values throughout the reachable state space of the circuits. Then we discard one of the flip-flop from the equivalent pair which has a higher observability value because there is a higher probability that its value will be blocked at the primary output as compared to the other flip-flop. Assume there are $m1$ true constant flops and $m2$ true equivalent flip-flop pairs, then there are $(m1 + m2/2)$ flip-flops that can be discarded. The remaining $(n - (m1 + m2/2))$ is considered for mining. The reason for removing one of the flip-flop from the equivalent pair is illustrated by the following example. Assume that flip-flops f_i and f_j are equivalent and there exists an illegal constraint shown in Table 3.2 that is necessary to prove two circuits equivalent. If both f_i and f_j are removed from the database, then this illegal constraint will not be captured. However, if only flip-flop f_j is removed, then the missing pattern (011) on (f_i, f_k, f_m) will be captured from the database. Adding this illegal constraint along with the constraints for equivalent flip-flop pairs $f_i = 0$ and $f_j = 1$ and $f_i = 1$ and $f_j = 0$, will prevent the illegal constraint (0011) during SEC framework. If on the other hand, f_j is not removed then we also capture illegal constraints $1011, 0111$ on (f_i, f_j, f_k, f_m) which are redundant and expensive to compute. In other words, it is more expensive to find 4 node missing patterns compared to 3 node missing patterns since the number of combinations are exponential to the nodes. Moreover, computing equivalent flip-flops is usually very fast and inexpensive.

Table 3.2: Example of illegal state

f_i	f_j	f_k	f_m	$\cdots$	f_n
0	0	1	1	$\cdots$	0

We utilize equivalent flip-flop pairs to reduce the number of backtraces during the all-solutions PODEM. If two flip-flops f_i and f_j are identified as equivalent, and if f_i has lower observability value than f_j, then all the fanouts of f_j are disconnected from f_j and tied to the f_i.

The effectiveness of the mining approach depends on the number of unique transactions present in the mining database. We keep the count of these unique transactions to be around $10,000$ in the mining database to capture as few missing patterns as possible. It is possible that the test vectors for random simulation of the sequential design might find very few (less than $10,000$) unique reachable states. So, to find more unique reachable states besides those found by random simulation we perform image computation for the subset of reachable states computed by random simulation using SAT solver. It provides the states reachable from these states in $k = 1, 2, \cdots, n$ steps. Once we obtain the required count of reachable states $(10,000)$ we stop and perform the mining approach as described above.

For circuits with large numbers of flip-flops, we use the MLP procedure (66) to obtain the initial ordering of the flip-flops. This procedure computes the input supports for the flip-flops and clusters the ones with closer supports. For our purpose, the size of each cluster is kept between $(24to70)$ and the mining database is constructed for each of these clusters separately. Since the flip-flops in each of these clusters are highly correlated it is very likely that we obtain most of the potential illegal constraints while reducing mining cost. The overall complexity of the above algorithm is $\mathrm{O}(n_{ff}.2MDL)$, which can be adjusted according to the user-specified value for MDL.

3.5 Novel Invariant checker for mined constraints

We need to verify each mined constraint to determine if it is indeed globally correct. We do so by leaving all initial PPIs states unconstrained. We also propose the verification of these illegal constraints in an incremental manner using 2 time frame assume-verify based induction to remove redundant constraints and reduce the time to verify them. The reason is an n node illegal constraint can be a superset of the $n-1$ node illegal constraint and conveys no useful information and can be discarded. For example, if $f_i f_j = 00$ has been proved as a true illegal constraint, then $f_i f_j f_k = 000$ (or 001) will also be a true illegal constraint and so we do not verify it. Similarly, if $f_i f_j f_k = 000$ has been proved as a false illegal constraint in (k) time frames by the SAT solver then $f_i f_k = 00$ will also be a false illegal constraint in the same (k) time frames since the sat solution that can satisfy the $f_i f_j f_k = 000$ will also satisfy $f_i f_j = 00$. Another observation is that if the miter output is not constrained to 0 in the first time frame (the assume window), then it is possible that some of these inductive invariants are proven false by the SAT solution that make the miter output 1 in the assume window. If the circuits are truly equivalent, then the SAT solution is most likely a false counter-example. So, we constraint the miter output as 0 in the assume window after doing the base case of induction.

Below is the summary of using assume-verify based induction for validating illegal constraints. First the base case of induction is performed to check if the miter output can be set to 1 in one time frame starting from the initial state S_0. If the SAT solver returns SAT then the circuits are not equivalent. However, if it is UNSAT then we unroll the miter for $k = 2$ time frame. In the first time frame (assume window) all one, two, three and four node illegal constraints as assumed to be true and the miter output is constrained to 0. For example, if the potential illegal constraint is $f_i = 0$ and $f_j = 1$ then $f_i + \neg f_j$ is added in the assume window and clauses $(\neg f_i)$ (f_j) are added in the verify window. In the verify window, first

one node constraints are verified and those proved true are used to discard constraints of size greater than one which are a superset of these true constraints. For constraints proved false, the sat solution from the SAT solver is used as proof to discard constraints, which are a subset of them. If the SAT solver return UNSAT, then it is a true constraint in that iteration else it is a false constraint. After discarding the true and false constraints of size greater than (m) we continue to verify the constraints of size greater than $(m+1)$ and use those proved true or false to discard constraints of size greater than $(m+1)$. This process is repeated until all the constraints are verified as either true of false in the first iteration. If some constraints are proved as false then perform the second iteration to see if all those proved true are still true even after removing the false constraints. This process is repeated until a fixed point is reached. This incremental approach reduces the time to verify these constraints significantly.

We use zChaff (2007.3.12 Version) (9) as the underlying SAT solver because it supports incremental SAT solving, where portions of clauses can be added or deleted from the database after each run based on their group ID. So we can easily add and delete the augment clauses from the original CNF formula. This could greatly reduce the overhead in checking the validity of the mined global constraints.

3.6 Application to SAT-based sequential equivalence checking

The illegal constraints can be applied to various verification problems. In our work, these constraints are applied to SAT-based unbounded SEC between an original and optimized version. The miter is unrolled for $k = 1$ time frame and miter output is constrained to

1 (meaning $\neg\phi$) and the PPI's are constrained to the initial state (s_0). If the SAT solver returns SAT then the circuits are not equivalent. However, if it returns UNSAT then the main inductive step is performed where the miter is unrolled for $k = 1$ time frame and the PPI's are unconstrained. The true illegal constraints are added to the $k = 1$ time frame. For example, if $f_i = 0$ and $f_j = 1$ is proven as a true illegal state then the clause $f_i + \neg f_j$ is added. If the SAT solver return UNSAT then the circuits are equivalent else nothing can concluded and the miter is unrolled for $k + 1$ time frame and the process is repeated until an upper bound on k ($= 20$) is reached or the circuits are proved equivalent. Since these are global constraints they can be added in all the time frames. We use zChaff as the underlying SAT solver.

To further strengthen the SAT-based induction steps we add additional constraints to the CNF formula such that any two consecutive states should be different in the unrolled miter circuit. These simple loop-free unique state constraints (43) can help in proving the properties at a smaller induction depth. For example, if the SAT solver returns a illegal sequence of states $(s_1, s_1, s_1, \cdots, s_k)$ such that miter output is 0 in the initial $k-1$ steps and it is 1 in the last k^{th} time frame, then because of the loop on state s_1 meaning there exists an input vector that can take s_1 to itself, the circuits will never be proved equivalent for any value of k. So, to prevent this, we add constraints as follows:

$s_1 \neq s_2 \wedge s_1 \neq s_3 \wedge \cdots \wedge s_1 \neq s_k \wedge$

$s_2 \neq s_3 \wedge \cdots \wedge s_2 \neq s_k \wedge$

$s_{k-1} \neq s_k$

The number of constraints to ensure unique states is quadratic to k. Adding these constraints guarantees that no state will be repeated.

3.7 Experimental Results

The proposed technique was implemented in C++. All tests were performed on a Intel Core i7 processor, 3.33GHz, with 6GB of RAM, running the Linux Ubuntu v9. The ISCAS'89 benchmarks along with their optimized circuits are used in our unbounded sequential equivalence checking. For some of the benchmarks adding equivalent flip-flops as constraints was sufficient to prove them equivalent and so we have excluded them from the experimental results. Those remaining retimed circuits could be proved equivalent by adding equivalent flip-flop pairs and equivalent internal signals as constraints. So, we generated a few hard to verify benchmarks using Synopsys design compiler for the ITC99 benchmarks. We generated optimized circuits with different state encodings (gray and one-hot) followed by optimizations from the design compiler such as state minimizations to increase the complexity of the benchmarks. Due to limited number of these hard benchmarks we show the results of our approach on both the retimed and different state encoding circuits.

Table 3.3 shows the mining and verification results for SEC. The first column shows the miter circuit used. The second column reports the number of random test vectors used for creating the mining database of size $10,000$. If these random vectors are not sufficient to get the mining database of that size then image computation is performed to obtain the remaining reachable states. The third column reports the flip-flops in the original circuit versus flip-flops in the optimized circuit. The fourth column reports the true equivalent flip-flop pairs found. The fifth column reports the subsets of flip-flops discarded from the mining database. This number may not always be half of the number of total equivalent flip-flop pairs (since one of the equivalent flip-flop pairs is discarded) because a flip-flop could be equivalent to more than one flip-flop. Thus this number is either half or less than the half of the fourth column. The sixth column reports the remaining flip-flops and is also the count of the number of columns in the mining database. The seventh column

($P1$) reports whether or not the miter circuit can be proved equivalent by equivalent flip-flop pairs, equivalent internal signals and pair-wise implications between the flip-flops and internal signals. The eighth column ($P2$) reports whether or not the benchmark can be proved by the ABC tool (67) from Berkley. The command "dprove" performs unbounded SEC checking. The ninth column reports the maximum size of multi-node illegal constraints computed using the mining approach. A size of 4 indicates that 4 node illegal constraints are necessary to prune the search space during SAT-based induction. The maximum limit of multi-node relationships is kept as 4 in our experimental results to keep the computational cost low. The tenth column reports the potential illegal candidates obtained by the mining approach and the eleventh column indicates those non-redundant and compressed illegal constraints proved true using assume and verify based induction. The twelfth column reports the upper bound for SAT-based induction and the thirteen column reports whether or not the benchmark is proved equivalent using our approach. The second last column reports the count of induction steps needed to prove the miter output = 0 is a true invariant after adding the true global constraints found by our approach and the last column reports the time to mine these constraints, verify them, add them to the SAT-based induction framework and perform unbounded SEC.

As can be observed from the results we are able to prove some benchmarks using these constraints, which cannot be proved by $P1$ and $P2$. For example, $b08$ has 44 flip-flops out of which 0 flip-flops have been removed from the mining database since they are no equivalent flip-flop pairs. The mining database is constructed for the 44 flip-flops and as can be observed this benchmark cannot be proved by either by $P1$ or $P2$ but can be proved by our mining approach. The 1364 true illegal constraints are added as constraints to the SAT-based induction SEC. Similarly, $b13$ has 63 flip-flops and their are 16 equivalent flip-flop pairs. So, 8 flip-flops are removed from the mining database and the subset of flip-flops

remaining in the database is 55. However, it is possible that the number of flip-flop discarded is not half of the total equivalent flip-flop pairs as for $b05$. The total number of potential one, two and three node illegal constraints is 109028 out of which 2373 were proved true in 6 time-steps.

For $b14$ since it has 733 flip-flops we use MLP to group the highly correlated flops together in clusters of size 24. The total number of equivalent flip-flop pairs identified is 270 out of which 135 are discarded. The MLP algorithm is run on the remaining 598 flip-flops and clusters of size 24 are obtained. The mining algorithm is applied to each of these clusters and total true illegal constraints found are 73364 proved in 6 time steps. Similarly, we are able to prove all the ICSAS'89 benchmarks using the mining approach.

However, some of the benchmarks could not be proved equivalent using this approach such as $b07$, $b10$ and $b11$. It appears that the count of potential illegal constraints is significant and very few are proved as true. One of the reasons could be that illegal constraints of size greater than 4 are needed to prove the benchmarks equivalent. It is also possible that some of those constraints cannot be proven true by our assume-verify based induction because of the false counter-examples. in our next approach, we add constraints of size bigger than 4 and are able to prove these benchmarks equivalent.

Figure 3.2 shows the number of potential illegal constraints obtained before and after removing the subset of flip-flops using the heuristic proposed versus the number of total number of flip-flops in the miter circuit. As can be observed the count of implications reduces significantly assuming there are some equivalent flip-flop pairs present between the original and the optimized circuit.

Figure 3.3 shows the number of flip-flops in the original circuit versus the flip-flops in the optimized circuits. For circuit b05 there is only flip-flop, which is removed from the mining

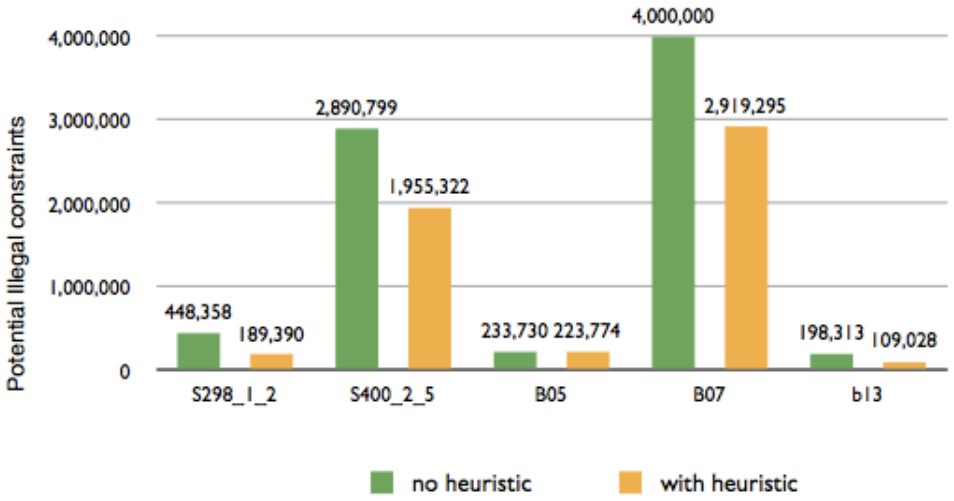

Figure 3.2: Potential illegal constraints before and after removing subset of constant and equivalent flip-flops

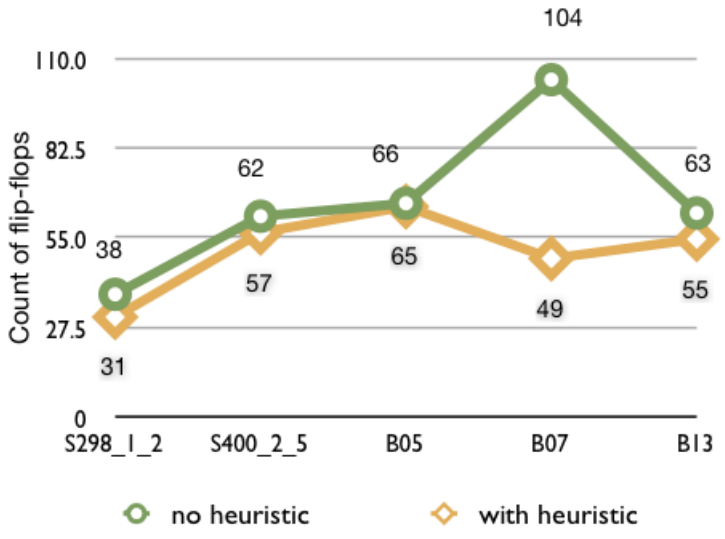

Figure 3.3: Flip-flops before and after using heuristics

database, thus the reduction in the implication count is small (9956) compared to others. However, there is no one-to-one mapping between the reduction in the number of flip-flops and the number of potential illegal states. This is because the number of unreachable states varies with circuit functionality.

3.8 Summary

In this chapter, we presented a novel mining technique to identify global illegal constraints in sequential circuits. The proposed mining method combined with heuristics reduces the computational cost of identifying the constraints in the huge number of possible missing relationships among flip-flops in the miter circuit. Experiments show that the proposed mining strategy is very efficient in identifying multi-nodes relationships up to size 4 that can significantly enhance the SAT solver in unbounded sequential equivalence checking framework. Some of the benchmarks that could not proved by the ABC tool and pair-wise relationships among flip-flops and internal signals can be proved by our approach.

Table 3.3: Mining and verification results using Illegal constraints (Subset of flip-flops.)

Benchmark	**# No. of test vec-tors**	**FFs (x/y)**	**# Equiv FF pairs**	**Subset of flops dis-carded**	**Flip flops re-main-ing**	**P1**	**P2**	**M1**	**Potential impl**	**True impl**	**Upper bound of in-duc-tion**	**Equiv**	**Proved in k steps**	**Time (sec)**
s298_1_2	100,000	17/21	14	7	31	Yes	Yes	4	189390	1539	20	Yes	3	289.84
s298_2_3	100,000	21/18	20	10	29	Yes	Yes	4	142595	1250	20	Yes	3	324.85
s298_2_4	100,000	21/20	22	11	30	Yes	Yes	4	157373	1720	20	Yes	3	425.74
s400_orig_1	100,000	21/23	10	5	39	Yes	Yes	3	47095	38875	20	Yes	3	77.3
s400_2_5	20,000	26/36	10	5	57	Yes	Yes	4	1955322	7973	20	Yes	3	14298.8
s444_2_3	223,360	25/30	2	1	37	Yes	Yes	3	9968	647	20	Yes	3	98.43
s444_3_4	223,360	30/36	18	9	57	Yes	Yes	3	41101	2377	20	Yes	3	334.74
s526_orig_3	223,360	21/49	8	4	66	Yes	Yes	4	3471912	14393	20	Yes	3	23286.4
s5378_orig_opt	11483	179/363	620	343	199	Yes	Yes	4	2488881	2058	20	Yes	3	35381.5
b01_gray_hot	110	5/10	0	0	15	**No**	**No**	4	16058	116	20	Yes	1	2.34
b02_gray_hot	2000	4/8	0	0	12	**No**	**No**	4	6325	84	20	Yes	1	0.5281
b03_gray_hot	2000	30/31	0	0	61	Yes	Yes	3	134934	646	20	Yes	1	190
b04_gray_hot	10000	69/70	10	5	134	Yes	Yes	3	143910	2144	10	Yes	1	497.08
b05_gray_hot	10000	32/34	3	1	65	**No**	**No**	3	223774	4377	10	Yes	1	41080.2
b06_gray_hot	10000	9/13	5	2	20	Yes	**No**	3	3803	352	10	Yes	1	0.71
b07_gray_hot	50000	51/53	110	55	49	**No**	**No**	4	2919295	33635	10	No	-	12445.1
b08_gray_hot	50000	21/23	0	0	44	**No**	**No**	3	39953	1364	10	Yes	1	250.2
b09_gray_hot	12000	28/30	0	0	58	Yes	**No**	3	131603	746	20	Yes	1	790
b10_gray_hot	50000	**17/24**	**0**	0	**41**	**No**	**No**	**4**	**730334**	**921**	**20**	**No**	-	**11313**
b11_gray_hot	50000	35/40	8	4	71	Yes	Yes	4	635953	200	20	No	-	10345
b13_gray_hot	50000	29/34	16	8	55	**No**	**No**	3	109028	2373	20	Yes	6	38023
b14_gray_hot	50000	366/367	270	135	598	Yes	Yes	3	>100000	73364	20	Yes	6	12089

P1 - Provable by equivalent flip-flop pairs, equivalent internal signals and pair-wise implications between flip-flops and internal signals.
P2 - Proved by ABC package from Berkley.
M1 - Maximum size of multi-node illegal constraint.

Chapter 4

Mining Minimal Complex Boolean Constraints

In this chapter we present a new low-cost technique on mining complex Boolean relationships among state variables and internal signals in the sequential miter circuit. In contrast to the method in the previous chapter that focused on illegal patterns, the algorithm in this chapter aims to find powerful legal relationships among several nodes efficiently. We also utilize the structural knowledge of the circuit to prune the search space of the mining process for internal signals and keep only those hard-to-find inductive invariants. We also propose a novel invariant checker to validate these potential inductive invariants. Experimental results demonstrate that the application of these global legal constraints can prune a large portion of the don't-care space and prove some of those circuits equivalent which cannot be verified using SAT based induction along with equivalent flip flops and internal equivalent signals alone.

4.1 Motivation

With aggressive sequential optimizations, there may exist few implications related to equivalent flip-flop pairs, equivalent internal signals, and pair-wise or three node relationships. Thus, conventional methods that target to learn such relations may not be sufficient to prune the search space during SEC. However, this makes the problem of SEC even harder, arising a need to learn constraints, which are more general in nature to effectively constrain the large don't-care space of the miter circuit. For example, consider the original design C_1 with n flip-flops (a_1 to a_n) and an optimized design C_2 with m flip-flops (b_1 to b_m) as shown in Figure 4.1. If there exists a relation among flip-flops (a_i, a_j, b_k, b_l) in the two designs such that when $a_i = 1$ and $a_j = 0$ in C_1 then $b_k = 0$ and $b_l = 1$ in C_2 and vice versa (i.e., $(a_i \wedge \overline{a_j}) \equiv (\overline{b_k} \wedge b_l)$). Then, adding this relation as a constraint will prune the search space by preventing those states that violate this constraint. The illegal states that could be blocked from this constraint are shown in Table 4.1. Note that conventional ways of finding internal signal equivalences will not uncover such general, arbitrary relationships among signals. We mine a subset of these general Boolean expressions among several nodes in the circuit.

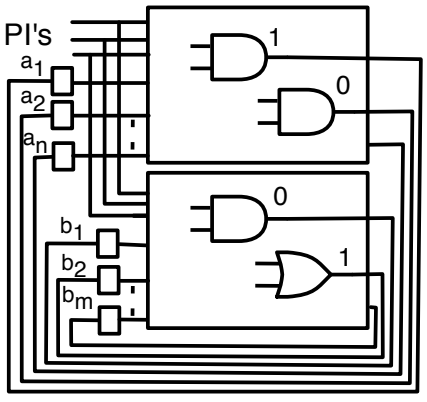

Figure 4.1: Example circuit

Table 4.1: Example of unreachable states (assume $n < m$)

a_i	b_k	a_j	b_l	$\cdots$	a_n	b_n	$\cdots$	b_m
1	0	0	0	$\cdots$	X	X	$\cdots$	X
1	1	0	0	$\cdots$	X	X	$\cdots$	X
1	1	0	1	$\cdots$	X	X	$\cdots$	X
0	0	0	1	$\cdots$	X	X	$\cdots$	X
0	0	1	1	$\cdots$	X	X	$\cdots$	X
1	0	1	1	$\cdots$	X	X	$\cdots$	X

4.2 Related Work

There has been a body of work that has dealt with computing non-trivial Boolean relations among signals via static implications. However, they are mainly focused on pair-wise signal relationships. Methods have been proposed to extend the learning to more than two variables. In (53) a method is proposed to find multi-node static implications, which select node pairs based on circuit structure information. Another method was proposed in (69) which uses local search to find multi-variable relationships over a subset of selected variables. Finding relations among multiple variables may involve exponential number of combinations and thus methods requiring low computational cost are needed.

A low cost data mining algorithm called Apriori (26), was proposed in (54) to compute two and three node relations of the form $(a \wedge b \rightarrow c)$ among nodes in the circuit. Apriori is an efficient and popular methodology for association-rule mining that can compute frequent association rules involving co-occurrences of certain events in the database. Adding these constraints can lead to one or two orders of magnitude speedup for bounded model checking. Data mining has also been used to compute cross-time frame constraints for speeding up SAT-based BMC for equivalence checking in (56). Bounded sequential equivalence checking can prove the equivalence of the two designs up to a certain depth k starting from any initial

state (say s_0). However, our target is unbounded SEC, which can prove the equivalence for the entire reachable state space of the design. But the problem is harder then bounded SEC due to the large don't-care state space. As such, we are trying to address a much harder problem.

4.3 Proposed Approach

We propose the use of data mining to obtain complex yet powerful Boolean relations among several nodes in the circuit. However, there is an exponential number of possible Boolean expressions relating the circuit signals, far too many to enumerate. To make the search space tractable, one approach is to mine all frequent Boolean expressions, but this may lose those constraints that manifest in infrequent sets. So, instead of mining only frequent Boolean expressions, we mine the minimal generators of the lossless subset called the closed Boolean expressions that retains the complete frequency information of the dataset. The reason for using the minimal generators for these closed expressions instead of the closed expressions themselves is that the latter may be complex and difficult to understand. On the other hand, the minimal generators are the simplest expressions that represent the same information as the closed expressions leading to no loss of useful information. These general inductive invariants can capture the functionality of the finite state machine and are not restricted to internal equivalences. We also propose the use of post processing techniques to remove redundant redescriptions obtained from the mining process. Next, we use our mining approach on selected internal signals in the circuit using structural knowledge to keep the computational cost low.

After data mining is completed, the potential inductive invariants are verified in an efficient way to further reduce the verification time using the assume-verify based induction. Our

experimental results show that the proposed mining approach is useful for finding powerful legal constraints of any size. However, computing relations among more then 8 variables is expensive and thus we limit the maximum size of these constraints as 8. As can be observed from the results we are able to prove some of those benchmarks equivalent by adding these constraints, which cannot be proved by the ABC package (67) from Berkeley. Thus, the relations obtained are effective to prove the circuits equivalent.

The rest of the chapter is organized as follows: Section 4.4 gives the details of mining approach on flip-flops and the post-processing techniques to discard redundant implications. Section 4.5 details the mining approach on internal signals, their selection and post-processing to discard trivial relations. It also talks about how to combine both the illegal (Chapter 3) and constraints (from BLOSOM). Section 4.6 presents the validity check of mined constraints using assume and verify. Section 4.7 gives its application to SEC. Section 4.8 discusses the experimental results, and Section 4.9 concludes the chapter.

4.4 Mining redescriptions using BLOSOM for flip-flops

Most potential constraints found today are restricted to either two or three node Boolean relationships in conjunctive form. One of the reasons is that it would become computationally expensive to exhaustively find all possible relations among signals in circuits. So, we propose to mine complex minimal Boolean expression with BLOSOM (27). The mined relations are those general constraints among signals that can constrain a bigger subset of the Boolean space than those simple conjunctive-type constraints. Thus we mine redescriptions, which represent equivalences between minimal generators/Boolean expressions represented in conjunctive normal form for the same closed set X in the dataset.

The miter circuit is simulated with M random vectors and the equivalent flip-flop pairs are

identified. Then the same heuristic as described in Section 3.4.1 is used to remove one of the flip-flop with low observability value from the equivalent pair. The mining database is constructed using the subset of flip-flops. Then BLOSOM is applied on the database with different minimum and maximum support values to obtain potential constraints in the form of redescriptions/equivalences among Boolean expressions. For example, the redescription (equation 4.1) represents equivalences between minimal generators ($f_i \wedge f_j \wedge f_k$) and ($f_l \wedge f_m \wedge f_n$) for the same closed set in the dataset. If it is proved as a true constraint then the following relations $1110XX$, $111X0X$, $111XX0$, $0XX111$, $X0X111$ and $XX0111$ on these six flip-flops represent the don't-care set and constrain a large subset of Boolean space.

$$f_i \wedge f_j \wedge f_k \equiv f_l \wedge f_m \wedge f_n \tag{4.1}$$

$$(f_i \vee f_j) \wedge (f_k \vee f_i) \equiv (f_l \vee f_m) \wedge (f_n \vee f_m) \tag{4.2}$$

The second form of redescription (equation 4.2) represents equivalences between minimal generators in the form of minimal Boolean CNF expressions. It says that if f_i is 1 or f_j is 1 and f_k is 1 or f_i is 1 then f_l is 1 or f_m is 1 and f_n is 1 or f_m is 1 and vice versa. These kinds of constraints cannot be obtained by the previous data mining techniques.

4.4.1 Post-processing for mined relations

Interestingly, we observed that some of these constraints may be redundant. We categorize them into four types of constraints that can be discarded from those obtained by BLOSOM. The reason is that we want to verify as small set of potential constraints as possible yet powerful enough to prune the search space effectively. It also reduces the time needed to verify these mined relations.

Type 1:

Given two redescriptions (equation 4.3) and (equation 4.4) for the two closed sets, any redescription that is a conjunction of these two redescriptions (equation 4.5) is discarded.

$$f_1 \vee f_2 \equiv f_3 \vee f_4 \tag{4.3}$$

$$f_5 \vee f_1 \equiv f_4 \vee f_6 \tag{4.4}$$

$$(f_1 \vee f_2 \equiv f_3 \vee f_4) \wedge (f_5 \vee f_1 \equiv f_4 \vee f_6) \tag{4.5}$$

Type 2:

If the redescription is of the following type (equation 4.6) then it can be discarded because the same information is captured by BLOSOM as a different redescription (equation 4.7)

$$(f_1 \wedge f_2 \equiv f_1 \wedge f_2 \wedge f_3) \tag{4.6}$$

$$(f_1 \wedge f_2 \equiv f_3) \tag{4.7}$$

Type 3:

Given a redescription (equation 4.8) consisting of n variables, if there exists another redescription (equation 4.9) which contains the same n variables as the first redescription (equation 4.8) but some or all the variables are swapped on either side of the redescription then we discard it. For example, in (equation 4.8) variable f_3 and f_5 are swapped as shown in (equation 4.9)

$$((f_1 \wedge f_2 \wedge f_3) \equiv ((f_1 \wedge f_2 \wedge f_5) \tag{4.8}$$

$$((f_1 \wedge f_2 \wedge f_5) \equiv ((f_1 \wedge f_2 \wedge f_3) \tag{4.9}$$

Type 4:

Given a redescription (equation 4.10), of size n where n is the total number of variables (same or different) on both sides of the equivalence relation and covers less than $(n-1)$ different variables then it is discarded. For example, in (equation 4.10) the size of redescription is six but it only covers four flip-flops, so it is discarded. In other words, we only verify redescriptions of size n covering at least $(n-1)$ variables.

$$(f_1 \wedge f_2 \wedge f_3 \equiv f_1 \wedge f_2 \wedge f_4) \tag{4.10}$$

Removing these four types of redescriptions reduces the mined constraints significantly and the remaining constraints are powerful enough to prove the two circuits equivalent as can be observed from the experimental results. We also provide the count for these four types of redescriptions in the experimental results. For bigger circuits, if the number of rows and columns in the mining database is large, then running BLOSOM with a lower minimum support might take a long time because the number of redescriptions can be significant. So, the database is divided into multiple small databases and BLOSOM is run on each of the small database, once with a lower minimum support and a lower maximum support and the second time with a higher minimum support and a higher maximum support. However, every time the constraints are computed from a small database, some of them may be violated in remaining small databases (called the filtering databases) and thus those constraints that violate the transactions in filtering databases are dropped.

4.5 Mining redescriptions for internal signals

For some circuits, constraints on flip-flops only might not be sufficient and so, we also find similar types of constraints on internal signals. Those constraints on internal signals close to the primary outputs might constrain a larger don't-care space by preventing certain combinations on a large subset of the primary inputs and pseudo primary inputs in the fan-in cone of these critical internal signals. For example, in the miter circuit, if there is a constraint that $g3 = 1 \wedge g4 = 0$ in original circuit is equivalent to $g5 = 0 \wedge g6 = 0$, then any combination on flip-flops and primary inputs that can violate this constraint is illegal.

However, the number of internal signals is much more than the number of flip-flops; thus, choosing all the internal signals for the BLOSOM database might not be scalable, leading to significant increase in the number of potential inductive invariants. So, we propose certain heuristics for selecting a subset of the internal signals by using structural information of the design. We also perform post-processing to discard some of the redescriptions obtained by BLOSOM, which might not be very effective in pruning the illegal state space. The mining database for internal signals is constructed such that each successive row consists of values on internal signals in every third time frame. This is done to filter combinational relations obtained by BLOSOM and is described in detail in the next section.

4.5.1 Selection criteria and post-processing techniques

We now describe the selection criteria for choosing only a subset of internal signals from the miter circuit. First, the miter is converted to And Inverter Graph (AIG) format where each gate in the design is either an AND gate or a NOT gate. The NOT gates are not considered in the mining database as the input of the NOT gate is sufficient to represent any relationship with other signals. For example, in Figure 4.2 if there exists a relationship

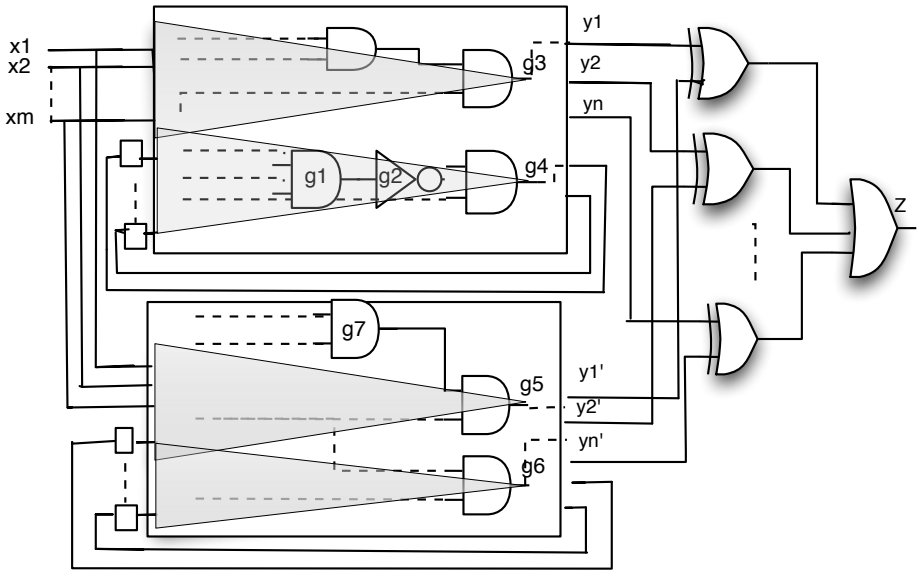

Figure 4.2: Example circuit

between gate $g1$ and NOT gate $g2$ such that when $g1 = 1$ is equivalent to $g2 = 0$, then since $g2$ drives another AND gate $g4$, its output will also be 0 and so there will exist another relation in the mining database $g1 = 1$ equivalent to $g4 = 0$. However, if $g2 = 1$ then the output of $g4$ depends on the other input and thus there may be no implication in the mining database. But this relation is not very useful and can be dropped. Thus, in either case the implication between $g1$ and $g2$ or between $g2$ and $g4$ are not useful and need not be included. So, currently the mining database has only signals whose gate type is an AND gate.

Our second approach is based on removing constant signals from the miter circuit. First all the potential constant nodes (having either '1' or '0' throughout the database) are identified. Then some of those proved, as true constant nodes might be redundant. Consider the example circuit in Figure 4.2 assuming that gate $g5$ is a true constant '1' node then the gate $g7$ will also be a true constant '1' node. So $g7$ can be removed from the mining database. Similarly, if the input gate of $g7$ is a true constant '1' signal then it is removed from the mining database. This process is repeated for all the true constant nodes starting from the

primary outputs towards the primary inputs. After removing the redundant true constant node, BLOSOM is run on the database for internal signals with reduced number of columns to find Boolean constraints among signals.

In the next step, we perform some post-processing on the redescriptions mined by BLOSOM to keep as small a subset of these constraints as possible yet powerful enough to prune the search space effectively. For the multi-node redescriptions we discard them based on the following criteria. The first criterion is to use the mining database $D1$ (database from time frame 1) to discard subset of possible combinational constraints and only retain as many sequential constraints as possible. In (63) pair-wise and multi-node sequential constraints are also identified via sequential implications. Sequential implications are those relations among signals in the same time frame that cannot be identified within a single time frame. In other words, a sequential implication $a \rightarrow b$ in the same time frame is only possible via two additional implications $a \rightarrow c$ and $c \rightarrow b$ with c in a different time frame. As a result, even though $a \rightarrow b$ seems like a combinational relation as both a and b reside in the same time frame, it is sequential in nature. We only target on sequential constraints since they are able to effectively prune the search space. The second criterion is to discard multi-node constraints, which involve signals only from one circuit. For example, if there is a redescription $(a \wedge b \equiv c \wedge d)$ then if signals (a, b, c, d) are from the same circuit then we discard this relation. However, if (a and b) are from circuit 1 and (c and d) are from circuit 2 then we keep it. Also, similar to post processing for flip-flops in Section 4.4.1, we remove the four different types of redundant constraints. Those remaining potential inductive invariants are verified using the novel invariant checker discussed in the next section.

4.5.2 Combining Illegal and Legal constraints

For circuits which cannot be proved either by illegal constraints alone in Chapter 3 or by BLOSOM on flip-flops/internal signals, we try to prove them by combining both of these approaches. The overall flow of mining all these constraints as follows:
First the equivalent flip-flop pairs are identified, followed by reduction in the columns of the mining database. The potential three or four node illegal constraints are mined and verified using assume-verify based approach. If the maximum size of the true illegal constraints is three, then all the potential legal constraints obtained by BLOSOM of size less than or equal to three are dropped since they will not provide any additional information that can further constrain the search space. All the remaining constraints are filtered as described in this chapter above. Next, adding the true illegal constraints in both the assume and verify window while verifying the remaining constraints using invariant checker might lead to more true inductive invariants. The true illegal constraints and true legal constraints from BLOSOM are then added to the SEC framework.

4.6 Novel Invariant Checker for mined constraints

To further reduce the time needed to verify the equivalences (or redescriptions), we use a similar approach as described in Section 3.6. We discard some of the constraints not yet verified based on those constraints that have been proved as either true or false in the current iteration. Here the approach is not as straightforward because the expressions are in CNF form. In assume and verify based induction, we assume all the potential implications in the assume window. In the verify window, we first verify simple two node implications between flip-flops of the following type where a and b are flip-flops.

$$(a) \equiv (b)$$
$$(c) \equiv (d)$$

Then we use those true implications to discard redundant implications of the following type:

$$(a \wedge c \wedge f) \equiv (b \wedge d \wedge f)$$

For implications that are proved false we use the sat solution provided by the SAT solver to mark those implications as false which satisfy the SAT solution in current iteration. For example, if the implication $(g \rightarrow h)$ is proven as false and the SAT solution with the following value assignments on the flip-flops is:

$$a = 1, b = 0, c = 1, d = 1, f = 0, g = 1, h = 0$$

Then the following implication $(a \vee b) \wedge (c) \wedge (d) \rightarrow (c \vee d) \wedge (f \vee g) \wedge (\bar{h})$ is also marked as false since it will be violated by the solution. In other words, there exists an assignment on the flip-flops that can make the above implication false since the implicant $(a \vee b) \wedge (c) \wedge (d)$ evaluates to true and the implied value $(c \vee d) \wedge (f \vee g) \wedge (h)$ evaluates to false. For those implications that are proven as true in the current iteration, a truth table is constructed for the CNF expression to determine those combinations on the variables for which the implication is true. Then all those implications that contain the same or subset of the variables as the current implication (I_1) and at least satisfy all those combinations which are satisfied by I_1 are marked as true in the current iteration. It reduces the time needed to verify the potential CNF equivalences obtained from BLOSOM. Figure 4.3 shows the overall flow of computing and verifying the mined Boolean relations.

The miter circuit is unrolled for two time frames and assume-then-verify is used to verify the mined relations in conjunctive normal form (CNF). These mined relations cannot be directly

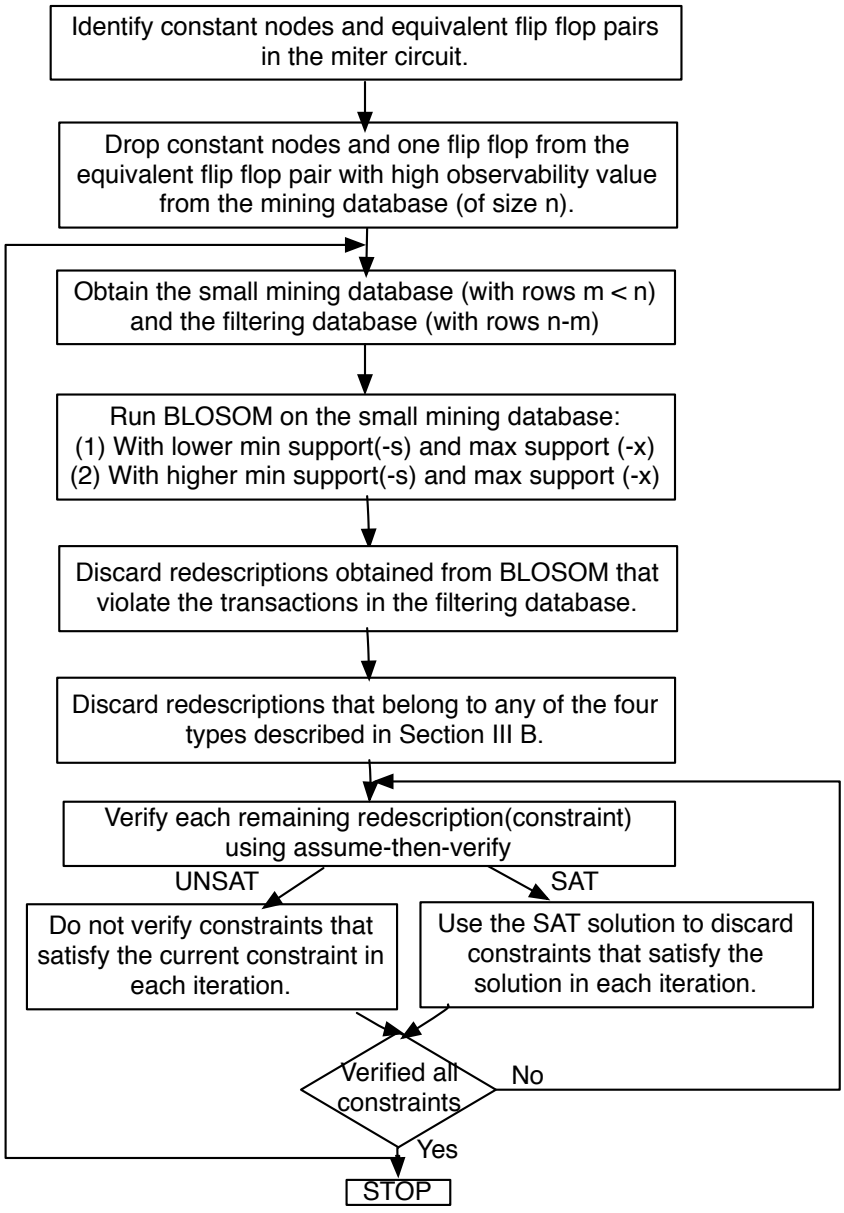

Figure 4.3: Flowchart for overall framework of mining general Boolean relations

converted to CNF clauses. So, extra AND and OR gates are added for each relation being verified. For example, if the potential constraint is $(\bar{f}_i \vee \bar{f}_j) \wedge (\bar{f_m}) \rightarrow (f_k) \wedge (\bar{f}_l \vee f_p)$ then OR gates $(o1)$ and $(o2)$ are added for $\bar{f}_i \vee \bar{f}_j$ and $\bar{f}_l \vee f_k$ and their corresponding OR clauses are added to the original CNF database. Next the OR gate output $(o1)$ is ANDed with flip-flop $\bar{f_m}$ and an AND gate (a1) is added. Similarly, the OR gate output $(o2)$ is ANDed with flip-flop (f_k) and an AND gate (a2) is added. The corresponding AND clauses for the AND gates (a1) and (a2) are added to the original CNF database. This is also represented by figure 4.4. The AND gate outputs (a1) and (a2) represent the Boolean expressions on left and right hand side of the implication in the first time frame. Thus the implication can be written as $(a1) \rightarrow (a2)$. We then add the clause $(\bar{a1} + a2)$ to the original CNF formula in the first time frame. Similarly, we add clauses for the OR and AND gates in the second time frame and add the two clauses $(a3)$, $(\bar{a4})$ for the implication. If the SAT solver returns UNSAT, then the implication is a true global constraint else it can be discarded.

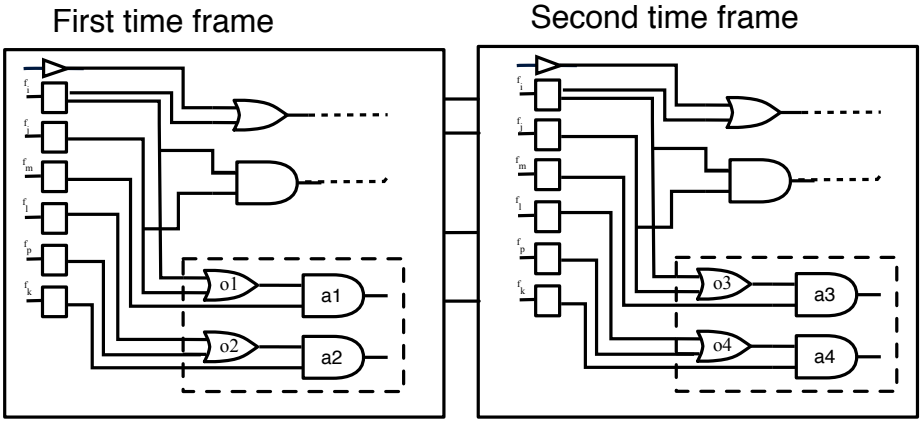

Figure 4.4: Two time frame unrolled miter with extra gates for implication

4.7 Application to SAT-based Sequential Equivalence Checking

The true constraints learned from above are now applied to (unbounded) sequential equivalence checking based on an inductive reasoning framework. Since these are global invariants they are added in the all the time frames of the unrolled miter during SAT-based induction runs. For example, if the true learned constraint is $(a \wedge b \rightarrow c \wedge d)$ then first, clauses for the AND gates (say x_1 and x_2) for $(a \wedge b$ and $c \wedge d)$ are added to the original CNF formula. Then the clause for the constraint $(x_1 \rightarrow x_2)$ is added as $(\neg x_1 + x_2)$.

The sequential equivalence checking model is shown in Figure 4.5, where each time frame consists of both the original and the optimized circuits. The miter circuit is unrolled for k time frames. The base case of induction as shown in Figure 4.5 is performed where the pseudo primary inputs are constrained to the initial state and it is checked if miter output can be satisfied to '1'. If the SAT solver returns SAT then the circuits are not equivalent. The assignment to the primary inputs serves as a witness to their inequality, and we are done. If it is UNSAT, then the induction step is performed, where the miter circuit is unrolled for k time frames with the initial state left unconstrained. The miter output is assumed to be '0' in the first $(k-1)$ time frames and the output is checked for 1 in the last time frame. The proven mined constraints by our approach are added to all the k time frames. If the SAT solver returns UNSAT in this step, then the two circuits are equivalent for all reachable states. However, if it returns SAT, then the counter example could be a false or true counterexample and nothing can be concluded. So, the miter is unrolled deeper for $(k+1)$ time frames and the process is repeated until SAT solver returns UNSAT (the two circuits are equivalent) or an upper bound is reached (assumed 20 in our experiments).

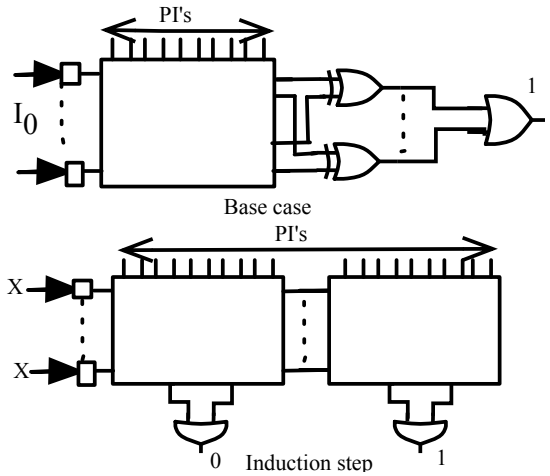

Figure 4.5: Model for sequential equivalence checking

4.8 Experimental Results

The proposed technique was implemented in C++. All tests were performed on a Intel Core i7 processor, 3.33GHz, with 6GB of RAM, running the Linux Ubuntu v9. The ISCAS'89 benchmarks along with their retimed and optimized version produced either using our script or ABC package (67) are used in our framework. For some of these benchmarks adding equivalent flip-flops as constraints was sufficient to prove them equivalent and so we have not included them in the experimental results and only report hard-to-verify instances. Please note that although these benchmarks can also be proved equivalent with internal equivalent signals, we avoid injection of these internal equivalences to show the power of our approach. We were also able to generate and use few ITC99 benchmarks in our framework where the original design was optimized using gray and one hot encoding followed by state minimization using Synopsys design compiler. These optimized designs have the same functionality as the original design, but they had very limited structural similarity. Currently, due to limited availability of these benchmarks that employ powerful sequential optimizations (beyond retiming and simple optimizations), we resort to such an experimental setup.

However, should there be more circuits with few or no internal equivalences, we believe our method will offer tremendous value to the current state of the art.

In Table 4.2 we only use BLOSOM on flip-flops to see if we can prove the circuits equivalent. The first column reports the benchmarks. The second column reports the flip-flops in the original versus optimized circuit. The third column reports the equivalent flip-flop pairs. The fourth column reports the internal equivalent pairs and pair-wise implications. The fifth column reports the subset of flip-flops after discarding one flip-flop from the equivalent pair. The sixth column reports whether or not the circuit can be proved equivalent by internal equivalences, pair wise implications and equivalent flip-flop pairs. The seventh column reports whether or not circuits can be proved by ABC tool (67) from Berkley. The eighth and ninth columns report the minimum and maximum support used to compute the potential equivalent expressions from BLOSOM. The tenth column reports the maximum number of flip-flops that could be included in a term in a general Boolean expression. The eleventh column reports whether the four types of redescriptions are removed after obtaining all the redescriptions from BLOSOM. The twelfth column reports the count of the redescriptions removed based on the four types described in Section 4.4.1. The thirteenth column reports the count of potential constraints obtained and the fourteenth column reports the true invariants. The fifteen column reports the upper bound for the induction runs that varies from 10 to 20 in our experiments. The sixteenth column reports whether or not circuits are proved equivalent using our approach. The seventeenth column reports the k steps in which the circuits are proved equivalent. The last column reports the total verification time for mining, verifying the potential constraints, adding them SEC framework and performing SAT-based SEC. We were able to prove most of the benchmarks equivalent using our approach.

The results show that some of the ITC99 benchmarks which we applied different state encodings could not be proved equivalent just by internal equivalences and equivalent flip-flop

pairs, but they could be proved equivalent by our approach. This reflects the effectiveness of our approach based on general Boolean expressions where a small number of constraints were effective in proving the two circuits equivalent. For example, in b08_gray_onehot, the b08 circuit with gray-code state encoding was checked against the b08 with one-hot state encoding. In this instance, one circuit had 21 FFs while the other had 23. There were no equivalent flip-flop pairs. Including the equivalent outputs, there were a total of 683 equivalent signal pairs (none of which involved flip-flops). Because there were no equivalent flip-flops, all 44 (21+23) flip-flops were used in our mining framework. The two circuits could be not proved equivalent using internal signal equivalences, equivalent flip-flop pairs and subset of pair-wise implications as reported under the **P1** column. They could be proved equivalent by the ABC tool (67) from Berkley (using "dprove" command) as reported under the **P2** column. Next, a min-support of 300 and max support of 10000 were used to mine potential relations among the flip-flops. We had used the 6 types of redescription filter as indicated under the column **R2**, with which 4109 implications were removed as they were found to be redundant. The remaining 756 potential implications were validated using an assume-then-verify flow, out of which 215 were found to be true implications. Then, these relations were used in a SEC setup. The total execution time, including mining all the way to SEC was just 3.2 seconds!

Note that for most of the ITC99 circuits (bxx circuits), there was no one-to-one mapping among the flip-flops (in terms of equivalences) between the two circuits. Even though there were 55 equivalent flip-flop pairs and 11880 internal signal equivalences for b07_gray_onehot, these were insufficient to prove the two circuits equivalent. On the other hand, our approach was able to prove them equivalent.

Next, for those ISCAS89 benchmarks (original vs. retimed versions), we were able to prove all the benchmarks equivalent. For example, consider s444_2_3, with 25 and 30 flip-flops, there

were 2 equivalent flip-flop pairs. Also, there were 1746 internal equivalent signals, including the corresponding output signals of the two circuits. We note that internal equivalences and flip-flop equivalences were sufficient to prove the two circuits equivalent, as shown under the **P1** column. The remaining columns could be interpreted in a similar manner as we had done earlier. Finally, the two circuits could be proved equivalent in only 16.8 seconds including the time to mine the relations, add then to the SEC framework and prove the two circuits equivalent. Note that in this case, we did not use any internal equivalences during SEC, so that we can test whether the general Boolean expressions were powerful enough. Moreover, in most of the benchmarks, setting a higher minimum support reduces the implication count and these implications are sufficient to prove the circuits equivalent in a significantly less time. Note: The benchmark s35932 is not included in the results since the original and the optimized version obtained by ABC package (67) can be proved equivalent just with just equivalent flip-flop pairs alone.

In Table 4.3 we show the results of using BLOSOM only on internal signals (as described in Section 4.5) for benchmarks, which cannot be proved by relations only on flip-flops or take more than 1000 seconds. The reason for doing this is that for some benchmarks constraints on internal signals might be much stronger than those only on flip-flops because of larger number of signals in the fanin cone of these internal signals. The first column reports the benchmarks. The second column reports the subset of flip-flops, which are the same as the fifth column in Table 4.2. The third column reports the total internal signals in the miter circuit after it is converted to AIG format. The forth column reports the total NOT gates in the AIG format which also represent the number of columns that are removed from the mining database. The fifth column reports the total true constant nodes found out of which some might be redundant. The sixth column reports the net count of the internal signals that are used in our mining database. The seventh and eighth column reports the minimum and maximum

support for BLOSOM. The ninth column reports the maximum number of internal signals that can be included in a term in a general Boolean expression. The tenth column reports whether the four types of redescriptions are removed after obtaining all the redescriptions from BLOSOM. The eleventh column reports the count of the redescriptions removed based on the four types described in Section 4.4.1. The twelfth column reports the count of potential constraints obtained and the thirteenth column reports the true invariants. The fourteenth column reports whether or not circuits are proved equivalent using our approach. The seventeenth column reports the k steps in which the circuits are proved equivalent. The last column reports the total verification time for mining, verifying the potential constraints, adding them SEC framework and performing SAT-based SEC.

The results show that constraints on internal signals can be more effective than flip-flops for some benchmarks. For example, for $b10$ the total count of true implications for internal signals is only 2907 compared to total count of true constraints for flip-flops 73924 and can be proved in 143 seconds which is much less than the time taken with constraints on flip-flops. Also, constraints on flip-flops are not sufficient to prove the miter circuit $b11$, however, constraints on internal signals including equivalent flip-flop pairs are sufficient to prove the circuit equivalent.

4.9 Summary

We presented a novel technique for mining complex multi-node Boolean relations among flip-flops and internal signals in sequential circuits for unbounded SEC. To the best of our knowledge, this is first of such mining of general Boolean expressions for SEC. These relationships can constrain the search space much more effectively than existing approaches. Experimental results show the potential and effectiveness of our mining approach.

Table 4.2: Mining and verification results using BLOSOM (Using subset of flip-flops.)

Benchmark	**FFs (x/y)**	**# Equiv FF pairs**	**# Internal Equiv pairs**	**Subset of flops**	**P1**	**P2**	**Min support**	**Max support**	**M**	**R2**	**Red impl**	**Potential impl**	**True impl**	**Upper bound of induction**	**Equiv**	**Proved in k steps**	**Time (sec)**
s298_1_2	17/21	14	240	31	Yes	Yes	50	1000	6	Yes	399	834	452	20	Yes	1	**4.41**
s298_2_3	21/18	20	244	29	Yes	Yes	75	1000	6	Yes	231	536	419	20	Yes	1	**8.2**
s298_2_4	21/20	22	252	30	Yes	Yes	75	1000	6	Yes	257	540	242	20	Yes	1	**8.7**
s400_orig_1	21/23	10	372	39	Yes	Yes	100	10000	6	Yes	7086	1648	423	20	Yes	1	**20.2**
s400_2_5	26/36	10	2942	57	Yes	Yes	195	1000	6	Yes	11509	19482	2859	20	Yes	3	192.9
s444_2_3	25/30	2	1746	37	Yes	Yes	1	1000	4	Yes	184	606	64	20	Yes	2	**8.8**
s444_3_4	30/36	18	1550	57	Yes	Yes	**950**	10000	4	No	150	126	64	20	Yes	2	**23.32**
s526_orig_3	21/49	8	529	66	Yes	Yes	350	1000	6	Yes	17481	32166	4450	20	Yes	3	432.7
s5378_orig_opt	179/203	620	-	321	Yes	Yes	1	1000	4	Yes	82250	164843	135453	20	Yes	1	**12036**
b01_gray_hot	**5/10**	**0**	195	**15**	**No**	**No**	**1**	**100**	**6**	**Yes**	**161**	**144**	**92**	20	Yes	1	**0.23**
b02_gray_hot	**4/8**	**0**	54	**12**	**No**	**No**	**1**	**100**	**6**	**Yes**	**88**	**112**	**81**	20	Yes	1	**0.19**
b03_gray_hot	30/31	0	47639	61	Yes	Yes	100	200	6	Yes	294	410	122	20	Yes	1	2.52
b04_gray_hot	69/70	10	917	134	Yes	Yes	710	10000	6	Yes	7604	1660	395	10	Yes	1	26.3
b05_gray_hot	**32/34**	**3**	-	**65**	**No**	**No**	**1**	**100**	**6**	**Yes**	**206781**	**3452**	**1105**	10	Yes	1	**139.4**
b06_gray_hot	**9/13**	**2**	**2205**	**20**	**Yes**	**No**	**1**	**100**	**6**	**Yes**	**16208**	**686**	**411**	10	Yes	1	**1.59**
b07_gray_hot	**51/53**	**110**	**118880**	**49**	**No**	**No**	**1**	**100**	**6**	**Yes**	**3746**	**18968**	**4344**	10	Yes	2	**814.1**
b08_gray_hot	**21/23**	**0**	683	**44**	**No**	**No**	**300**	**10000**	**6**	**Yes**	**4109**	**756**	**215**	10	Yes	1	**3.22**
b09_gray_hot	**28/30**	**0**	18161	**58**	**Yes**	**No**	**1**	**10000**	**6**	**Yes**	**31351**	**189702**	**4407**	20	Yes	2	**2298**
b10_gray_hot	**17/24**	**0**	1150	**41**	**No**	**No**	**100**	**10000**	**8**	**Yes**	**123013**	**801246**	**73924**	20	Yes	1	**31841**
b11_gray_hot	**35/40**	**8**	20623	**71**	**Yes**	**Yes**	**1**	**10000**	**6**	**Yes**	**5806**	**117186**	543	20	No	-	**18677**
b13_gray_hot	**29/34**	**16**	2084	**55**	**No**	**No**	**25**	**10000**	**6**	**Yes**	**44432**	**122640**	**19965**	20	Yes	1	**8202**

P1 - Provable by internal equivalent signals and equivalent flip-flop pairs. R2 - Count of four types of redescriptions obtained from BLOSOM.
P2 - Provable by ABC tool from Berkley, (67)
M - Maximum number of flip-flops in redescription.
'-' indicates time out in 12 hours.

Table 4.3: Mining and verification results using BLOSOM (Using subset of internal signals).

Benchmark	**Subset of flops**	**Total internal signals**	Total NOT gates	**True constant nodes**	**Subset of internal signals**	**Min support**	**Max support**	**M**	**R2**	**Red impl**	**Potential impl**	**True impl**	**Equiv**	**Proved in k steps**	**Time (sec)**
b09_gray_hot	**58**	**678**	**309**	**1**	**369**	**75**	**10000**	**2**	**Yes**	**0**	**18607**	**2035**	Yes	2	**350**
b10_gray_hot	**41**	**733**	**292**	**65**	**376**	**350**	**10000**	**4**	**Yes**	**2740**	**12145**	**2907**	Yes	2	**143.9**
b11_gray_hot	**71**	**1571**	**715**	**2**	**854**	**2000**	**10000**	**2**	**Yes**	0	**25577**	**3997**	**Yes**	**2**	2700
b13_gray_hot	**55**	**613**	**281**	**7**	**325**	**20**	**10000**	**4**	**Yes**	**9700**	**32867**	**13032**	**Yes**	**2**	**463**

R2 - Count of four types of redescriptions obtained from BLOSOM.
M - Maximum number of internal signals in redescription.
'-' - indicates time out in 12 hours.

Chapter 5

Conclusion and Future Work

In this thesis we have outlined the main limitation in SAT-based formal verification approaches specifically SAT-based induction for solving the problem of sequential equivalence checking. They can be summarized as follows: (i) Lack of the ability of the SAT solver to learn illegal and legal relations among flip-flops and internal variables. (ii) Lack of knowledge of non-trivial invariants of a design. (iii) Increase in the run-time at higher-depths that prohibits complete verification of the property (miter output = 1).

In Chapter 3, we propose a new technique of mining a subset of illegal constraints from the mining database. In addition to these mined constraints, we also use a low cost all solution ATPG algorithm called PODEM to find more of these illegal constraints. The goal is to aid the formal verification approach with powerful and effective learned constraints so that the search space can be pruned effectively during SAT-based induction. First, random simulation is used to generate a database and then missing patterns of size one, two, three and up to four nodes are mined from the database. For a circuit, with m flip-flops, a n node constraint (with n bits specified out of total m bits) covers a larger portion of the don't care space than a constraint of size $(n + 1)$. We further propose the use of some structural

similarity in the form of equivalent flip-flop pairs between the original and optimized design, to reduce the number of columns in the mining database and prune the search space during the mining process. Finally, for circuits with large number of flip-flops we use the MLP algorithm to find the group of highly correlated flip-flops and apply our mining approach on these smaller groups of flip-flops separately. Experimental results show that we are able to prove some of the benchmarks equivalent which have very less structural similarity between them and cannot be proved by constraints based on equivalences on signals and pair-wise implications. Future work could include improvement of mining technique to further reduce the overhead, mining of relationships that involve more than 4 nodes, elimination of some illegal constraints (some of them might not be as useful as others) and application to general model checking instances.

In Chapter 4, we propose the use of a novel mining technique called BLOSOM for finding complex general Boolean relations (also referred as redescriptions) that are a representative of the characteristics of the design under verification. This approach is complementary to the approach in Chapter 3. In Chapter 3 we are finding constraints that cannot occur in the normal functional mode and here we are mining constraints that have to be met for the correct functionality of the design. To the best of our knowledge, there exists no work that can mine such powerful relations among several nodes to solve a much harder problem of SEC. We do some post-processing on the mined relations to remove redundant redescriptions. We also use the structural information to select a subset of internal signals for mining and remove trivial constraints. To further reduce the verification time we propose a novel invariant checker. The learned constraints are then applied to the SEC framework. The experimental results show that we are able to prove those benchmarks equivalent in a reasonable time which cannot be proved by adding internal equivalences, pair-wise constraints and equivalent flip-flop pairs as constraints and also by ABC tool from Berkeley (67). Our future work involves

two directions, firstly to extract quality constraints to reduce the count of these implications which will further reduce the verification time for larger designs. Secondly, using structural information of the circuit to guide the data mining tool in extracting constraints that are non-trivial and hard to capture instead of the post-processing step on the mined constraints.

The major contribution of this thesis is that we address functional verification problem specifically sequential equivalence checking by novel multi-nodes mining techniques. The multi-nodes mining techniques proposed in this thesis can capture various important invariant relations in a design. We believe the research that has been presented in this thesis makes a substantial foundation to many ongoing and future researches.

Bibliography

[1] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.

[2] D. A. Plaisted and S. Greenbaum, "A structure-preserving clause form translation," *Journal of Symbolic Computation*, vol. 2, no. 3, pp. 293–304, 1986.

[3] G. Tseitin, "On the complexity of derivation in propositional calculus." *Studies in constructive mathematics and mathematical logic.*, vol. Part II, pp. 115–125, 1968.

[4] D. Martin, L. George, and L. Donald, "A machine program for theorem-proving," *Communications of the ACM*, vol. 5, no. 7, pp. 394–397, 1962.

[5] N. Een and N. Sorensson, "An extensible sat-solver [ver 1.2]," 2003.

[6] J. P. Marques-silva and K. A. Sakallah, "GRASP: A search algorithm for propositional satisfiability," *IEEE Transactions on Computers*, vol. 48, pp. 506–521, 1999.

[7] L. Zhang, C. F. Madigan, and M. H. Moskewicz, "Efficient conflict driven learning in a boolean satisfiability solver," in *Proceedings of International Conference on Computer-Aided Design (ICCAD)*, 2001, pp. 279–285.

[8] L. Zhang and S. Malik, "The quest for efficient boolean satisfiability solvers," in *Pro-*

ceedings of the 14th International Conference on Computer Aided Verification, London, UK, 2002, pp. 17–36.

[9] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: engineering an efficient sat solver," in *Proceedings of the 38th Annual Design Automation Conference*, 2001, pp. 530–535.

[10] E. Goldberg and Y. Novikov, "Berkmin: A fast and robust sat-solver," in *Proceedings of the conference on Design, Automation and Test in Europe*, 2002, pp. 142–149.

[11] S. Malik, "Boolean Satisfiability Research Group at Princeton, Release zChaff 2007.3.12." [Online]. Available: http://www.princeton.edu/ chaff/zchaff.html

[12] M. R. Prasad, A. Biere, and A. Gupta., "A survey of recent advances in sat-based formal verification," *International Journal on Software Tools for Technology and Transfer*, vol. 7, no. 2, pp. 156–173, April 2005.

[13] W. Kunz, D. Pradhan, and S. Reddy, "A novel framework for logic verification in a synthesis environment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 1, pp. 20–32, January 1996.

[14] W. Kunz, D. Stoffel, and P. Menon, "Logic optimization and equivalence checking by implication analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 3, pp. 266–281, March 1997.

[15] J. K. Zhao, E. Rudnick, and J. Patel, "Static logic implication with application to redundancy identification," in *Proceedings of 15th IEEE VLSI Test Symposium*, April 1997, pp. 288–293.

[16] J. K. Zhao, J. A. Newquist, and J. H. Patel, "A graph traversal based framework for

sequential logic implication with an application to C-Cycle redundancy identification." Los Alamitos, CA, USA: IEEE Computer Society, 2001, p. 163.

[17] W. Kunz and D. K. Pradhan, "Recursive learning: A new implication technique for efficient solutions to cad problems test, verification, and optimization." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 9, pp. 1149–1158, September 1994.

[18] M. Syal, R. Arora, and M. Hsiao, "Extended forward implications and dual recurrence relations to identify sequentially untestable faults," in *Proceedings of IEEE International Conference on Computer Design: VLSI in Computers and Processors*, October 2005, pp. 453–460.

[19] W. Kunz and D. K. Pradhan, "Accelerated dynamic learning for test pattern generation in combinational circuits," *IEEE Transactions on CAD of Integrated Circuits and Systems*, vol. 12, no. 5, pp. 684–694, May 1993.

[20] A. Agresti and D. B. Hitchcock, "Bayesian inference for categorical data analysis," *Statistical Methods and Applications*, vol. 14, pp. 297–330, 2005.

[21] A. Popescul, R. Popescul, and L. H. Ungar, "Structural logistic regression for link analysis," in *Proceedings of the Second International Workshop on Multi-Relational Data Mining*. Washington, DC: ACM Press, 2003 2003, pp. 92–106.

[22] A. Gelman, "Analysis of variance," *New Palgrave Dictionary of Economics*, 2006.

[23] T. Kurita, "An efficient agglomerative clustering algorithm using a heap," *Pattern Recognition*, vol. 24, no. 3, pp. 205–209, 1991.

[24] T. Krink and S. Paterlini, "Differential evolution and particle swarm optimisation in

partitional clustering," *Computational Statistics and Data Analysis*, vol. 50, pp. 1220–1247, 2006.

[25] M. Charikar and C. Chekuri, "Incremental clustering and dynamic information retrieval," in *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, 1997, pp. 626–635.

[26] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo, "Fast discovery of association rules," *Advances in Knowledge Discovery and Data Mining*, pp. 307–328, 1996.

[27] L. Zhao, M. J. Zaki, and N. Ramakrishnan, "BLOSOM: A framework for mining arbitrary boolean expressions over attribute sets," in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2006, pp. 827–832.

[28] M. J. Zaki and C. J. Hsiao, "CHARM: An efficient algorithm for closed association rule mining," Computer Science, Rensselaer Polytechnic Institute, Tech. Rep., 1999.

[29] E. Hunt, *Concept Learning: An Information Processing Problem*. New York: Wiley, 1962.

[30] J. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

[31] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[32] A. K. Jain, J. Mao, and K. Mohiuddin, "Artificial neural networks: A tutorial," *IEEE Computer*, vol. 29, pp. 31–44, 1996.

[33] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning.* Boston, MA, USA: Kluwer Academic, 1989.

[34] M. Z. Shafiq, S. A. Khayam, and M. Farooq, "Improving accuracy of immune-inspired malware detectors by using intelligent features," in *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation.* New York, NY, USA: ACM, 2008, pp. 119–126.

[35] S. Kromesch and J. S., "High dimensional data visualization," *6th International Symposium of Hungarian Researchers on Computational Intelligence*, pp. 1–12, 2005.

[36] R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large databases," in *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, 1993, pp. 207–216.

[37] M. J. Zaki and N. Ramakrishnan, "Reasoning about sets using redescription mining," in *Proceedings of the eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining.* New York, NY, USA: ACM, 2005, pp. 364–373.

[38] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications," *ACM Transactions on Programming Languages and Systems*, vol. 8, no. 2, pp. 244–263, 1986.

[39] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking.* Cambridge, MA, USA: The MIT Press, 2000.

[40] K. L. McMillan, *Symbolic Model Checking.* Norwell, MA, USA: Kluwer Academic Publishers, 1993.

[41] C. A. J. van Eijk, "Sequential equivalence checking without state space traversal," in

Proceedings of the conference on Design, Automation and Test in Europe. Washington, DC, USA: IEEE Computer Society, 1998, pp. 618–623.

[42] P. Bjesse and K. Claessen, "Sat-based verification without state space traversal," in *Proceedings of the Third International Conference on Formal Methods in Computer-Aided Design*. London, UK: Springer-Verlag, 2000, pp. 372–389.

[43] M. Sheeran, S. Singh, and G. Stålmarck, "Checking safety properties using induction and a sat-solver," in *Proceedings of the Third International Conference on Formal Methods in Computer-Aided Design*. London, UK: Springer-Verlag, 2000, pp. 108–125.

[44] M. Syal and M. S. Hsiao, "VERISEC: Verifying equivalence of sequential circuits using SAT," in *Proceedings of the High-Level Design Validation and Test Workshop, 2005. on Tenth IEEE International*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 52–59.

[45] I.-H. Moon and C. Pixley, "Non-miter-based combinational equivalence checking by comparing bdds with different variable orders," in *Proceedings of Formal Methods in Computer Aided Design*. Springer Verlag, November 2004, pp. 144–158.

[46] C. Pixley, "A theory and implementation of sequential hardware equivalence," *IEEE Transactions on CAD of Integrated Circuits and Systems*, vol. 11, no. 12, pp. 1469–1478, November 1992.

[47] R. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Transactions on Computers*, vol. C-35, no. 8, pp. 677–691, August 1986.

[48] O. Coudert and J. Madre, "A unified framework for the formal verification of sequential circuits," in *Proceedings of International Conference on Computer-Aided Design*, November 1990, pp. 126–129.

[49] J. H. R. Jiang and R. Brayton, "On the verification of sequential equivalence," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 6, pp. 686–697, June 2003.

[50] S. Y. Huang, K. T. Cheng, K. C. Chen, C. Y. Huang, and F. Brewer, "AQUILA: an equivalence checking system for large sequential designs," *IEEE Transactions on Computers*, vol. 49, no. 5, pp. 443–464, May 2000.

[51] C. van Eijk, "Sequential equivalence checking based on structural similarities," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 7, pp. 814–819, July 2000.

[52] F. Lu and K. T. Cheng, "SEChecker: A sequential equivalence checking framework based on kth invariants," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 6, pp. 733–746, June 2009.

[53] K. Gulrajani and M. Hsiao, "Multi-node static logic implications for redundancy identification," in *Proceedings of Design Automation and Test in Europe*, 2000, pp. 729–733.

[54] W. Wu and M. Hsiao, "Mining global constraints for improving bounded sequential equivalence checking," in *Proceedings of the 43rd Annual Conference on Design Automation*, San Francisco, CA, 2006, pp. 743–748.

[55] W. Wu and M. Hsiao, "Mining global constraints with domain knowledge for improving bounded sequential equivalence checking," in *Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 1, January 2008, pp. 197–201.

[56] C. L. Chang, C. H. P. Wen, and J. Bhadra, "Speeding up bounded sequential equivalence checking with cross-timeframe state-pair constraints from data learning," in *International Test Conference*, November 2009, pp. 1–8.

[57] M. H. Konijnenburg, J. van der Linden, and A. van de Goor, "Illegal state space identification for sequential circuit test generation," in *Proceedings IEEE/ACM Design, Automation and Test in Europe*, 1999, pp. 1184–1189.

[58] H. C. Liang, C. L. Lee, and J. E. Chen., "Identifying invalid states for sequential circuit test generation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 9, pp. 1025–1033, September 1997.

[59] L. Y. Chieh, L. Feng, and K. T. Cheng, "Pseudo- functional testing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 8, pp. 1535–1546, August 2006.

[60] Y. C. Lin and K. T. C. F. Lu, K. Yang, "Constraint extraction for pseudo-functional scan-based delay testing," *IEEE Asia and South Pacific Design Automation Conference*, pp. 166–171, January 2005.

[61] F. Yuan and Q. Xu, "On systematic illegal state identification for pseudo-functional testing," in *Proceedings ACM/IEEE Design Automation Conference (DAC)*, July 2009, pp. 702–707.

[62] Z. Zhang, S. M. Reddy, and I. Pomeranz, "On generating pseudo-functional delay fault tests for scan designs," *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 398–405, 2005.

[63] M. Syal, K. Chandrasekar, V. Vimjam, M. S. Hsiao, Y. S. Chang, and S. Chakravarty, "A study of implication based pseudo functional testing," *IEEE International Test Conference*, p. 24.3, 2006.

[64] V. C. Vimjam and M. S. Hsiao, "Fast illegal state identification for improving sat-based

induction," in *Proceedings of the 43rd annual Design Automation Conference*. New York, NY, USA: ACM, 2006, pp. 241–246.

[65] L. D. Moura, H. Rue, and M. Sorea, "Bounded model checking and induction: From refutation to verification (extended abstract)," in *Computer-Aided Verification*, A. Voronkov, Ed., vol. 2725. Springer-Verlag, 2003, pp. 14–26.

[66] I. H. Moon and F. Somenzi, "Border-block triangular form and conjunction schedule in image computation," in *Proceedings of Formal Methods in Computer-Aided Design*. Springer-Verlag, 2000, pp. 73–90.

[67] A. Mishchenko, "ABC: A system for sequential synthesis and verification, Release abc70930," 2006. [Online]. Available: http://www.eecs.berkeley.edu/ alanmi/abc/

[68] P. Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits." *IEEE Transactions on Computers*, vol. C-30, no. 3, pp. 215–222, March 1981.

[69] Y. Novikov, "Local search for boolean relations on the basis of unit propagation," in *Proceedings of Design, Automation and Test in Europe Conference*, 2003, pp. 810–815.