Mining constraints for Testing and Verification

Weixin Wu

Thesis submitted to the Faculty of
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
Chao Huang
Yilu Liu

January 20, 2009
Blacksburg, Virginia

Keywords: Mining, Learning, Multi-node Constraint, SAT, Simulation

Mining constraints for Testing and Verification

Weixin Wu

Abstract

With the advances in VLSI and System-On-Chip (SOC) technologies, the complexity of hardware systems has increased manifold. The increasing complexity poses serious challenges to the digital hardware design. Functional verification has become one of the most expensive and time-consuming components of the current product development cycle. Today, design verification alone often surpasses 70% of the total development cost and the situation has been projected to continue to worsen. The two most widely used formal methods for design verification are Equivalence Checking and Model Checking. During the design phase, hardware goes through several stages of optimizations for area, speed, power, etc. Determining the functional correctness of the design after each optimization step by means of exhaustive simulation can be prohibitively expensive. An alternative to prove functional correctness of the optimized design is to determine the design's functional equivalence with respect to some golden model which is known to be functionally correct. Efficient techniques to perform this process is known as Equivalence Checking. Equivalence Checking requires that the implementation circuit should be functionally equivalent to the specification circuit. Complexities in Equivalence Checking can be exponential to the circuit size in the worst case.

Since Equivalence Checking of sequential circuits still remains a challenging problem, in this thesis, we first address this problem using efficient learning techniques. In contrast to the traditional learning methods, our method employs a mining algorithm to discover global constraints among several nodes efficiently in a sequential circuit. In a Boolean satisfiability (SAT) based framework for the bounded sequential equivalence checking, by taking advantage of the repeated search space, our mining algorithm is only performed on a small window size of unrolled circuit, and the mined relations could be reused subsequently. 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 these problems are alleviated.

We also propose a pseudo-functional test generation method based on effective functional constraints extraction. We use mining techniques to extract a set of multi-node functional constraints which consists of illegal states and internal signal correlation. Then the functional constraints are imposed to a ATPG tool to generate pseudo functional delay tests.

To my Family

Acknowledgements

It is a pleasure to acknowledge all the people who made this work possible. I would like to thank Dr. Michael S. Hsiao, Dr. Chao Huang and Dr. Yilu Liu to serve on my committee and spend precious time on the thesis review and oral presentation. I would like also to thank all the Proactive members for their suggestions on my research and thesis.

Finally, I would like to express my deep gratitude to my family members and friends for their constant support and love.

Weixin Wu

January, 2009

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Functional Verification

1.1.1 The Need for Functional Verification

The advances in VLSI technology have led to an increased complexity in hardware system designs. The complexity of large designs pose serious challenges to pre-silicon verification and post-silicon testing. Design errors or bugs, especially the corner cases in a complex system, are expensive to discover and fix. However, it would be even more expensive if bugs slip the verification and testing stage and remain in the chip after they are manufactured. One well known example is the floating point division math bug in Intel's Pentium processor, which cost $475 million to fix.

Functional verification of sequential systems is rapidly becoming one of the most crucial and resource-intensive components of the product design cycle. Functional verification is the process of checking if a design implementation conforms to its specifications of functionality, timing, testability and power dissipation. 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. With the ever increasing sizes and complexity of designs, it has been projected that the reality of functional verification continues to worsen.

1.1.2 Different Functional Verification Approaches

Functional verification can be broadly classified into simulation, emulation and Formal approaches.

Simulation based verification has traditionally been used as the primary approach for design verification from system level to component level. In most of the industrial designs, simulation has been the most common way of verifying the designs. In simulation-based verification, a test pattern is applied at the inputs of the design implementation and the specification (could be a software model of the design) is simulated. The response obtained is analyzed against the expected response to check the correctness of the design. In order to completely verify a design, all the test patterns need to be generated, but the number of test patterns increases exponentially with the number of inputs. For instance, for an n-input circuit, there are 2^n possible input vectors and hence it becomes impractical to completely verify large circuits. Generally, instead of generating all the test patterns, a tractable set of input patterns are intelligently generated and the design is verified against those patterns. However, simulation based methods may miss some corner case errors as was seen in the case of infamous Pentium bug. Another drawback of simulation based methods is the huge amount of time taken to simulate the design for every test pattern. With complex designs, especially with system-on-chip (SOC), traditional simulation-based verification has become ineffective in finding subtle design bugs.

Emulation based methods, using Field Programmable Gate Array (FPGA) chips, offer a potential alternative to simulation based methods. They help to speed up simulation by several orders of magnitude. However, the fundamental drawbacks of this approach are the expensive hardware emulators and long time requirements to map the design under verification to the emulator. Furthermore, monitoring different sets of properties/assertions may be difficult to do.

A comparatively recent alternative to simulation has been formal verification. Formal verifi-

cation has gained increased attention in recent years. With advances of formal verification techniques, especially BDD and SAT, formal verification has been accepted and applied in many real projects. It has been shown that formal verification can be very cost-effective in finding hard bugs. Also it can mathematically prove or disprove the correctness of a design. Formal verification is analogous to a mathematical proof, where the correctness of a formally verified hardware design holds regardless of the input values that are applied. The consideration of all test cases is *implicit* in formal verification. In general, the formal methods for verifying hardware designs can be broadly classified into Equivalence checking and Model Checking.

In the case of equivalence checking, we attempt to see if the formal description of the implementation conforms to the "golden model" or specification. A common premise for equivalence checking is to see if the optimized version of a circuit conforms to its original version. The basic idea for equivalence checking is to construct a miter circuit as shown in Figure 1.1 and prove that the output of the miter circuit is a tautology zero. Basically, we tie the inputs of both the circuits to apply same input patterns. Then, we check if the outputs of both the circuits are same. If the output of the miter circuit is proved to be a tautology zero, then it implies that the outputs of the implementation circuit and the specification circuit are the same for all possible input patterns. Therefore, we can be 100% sure that the functionality of the implementation circuit is equivalent to the functionality of the golden model.

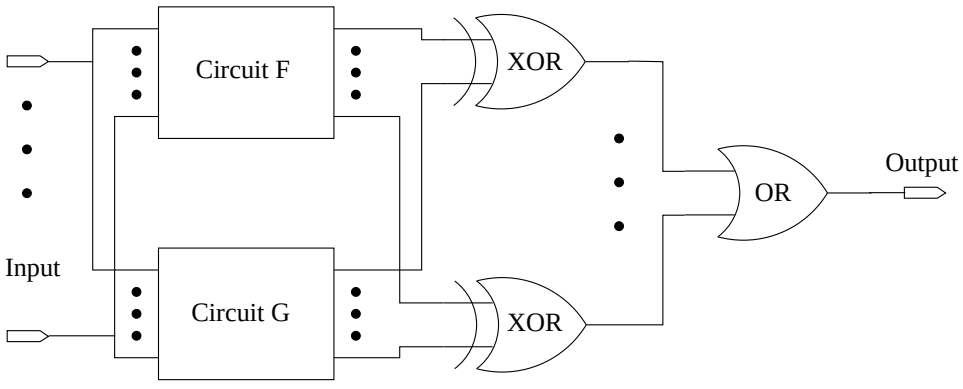

Figure 1.1: A miter circuit for equivalence checking

In the case of model checking, we need to verify if the design satisfies a property or not. The specification is usually a set of properties that need to be verified for the circuit. For example, in a Traffic Light Controller, we need to verify the property that two perpendicular roads should not get green at the same time. Model Checking is usually performed on Kripke structures or Finite State Machine models of the hardware design, where analytical techniques are used to verify the properties.

Model Checking largely relies on state-space traversal of the sequential circuit. Explicit methods for state space traversal need a complete representation of the State Transition Graph. Since the number of states is exponential to the number of state elements, explicit model checking methods are applicable only to small designs. With the increase in size of hardware designs, the state space increases exponentially for most of the them, thus making explicit methods infeasible. Therefore symbolic methods were introduced a decade ago, where the transition relation and set of states are represented as Boolean formulas and state space traversal is done formally using mathematical computations.

However, 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.

1.2 Contributions of This Thesis

It is clear that no single functional verification method will rule over all cases. We believe that the promising approach is a unified scheme, which combines the merits of different verification approaches and relies on both simulation-based approach and formal verification in assuring a satisfactory validation. The contributions of this thesis includes:

- Improve the performance of simulation assisted formal verification
- Search new constraint extraction techniques for pseudo functional testing

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 and testing techniques such as: Boolean Satisfiability and logic implication. Data mining algorithms are also introduced. How to applying mining technique to discover multi-nodes global invariants and how to apply global invariants to improve performance of bounded sequential equivalence checking are presented in chapter 3. In Chapter 4, we present several new mining techniques to extract powerful functional constraints to generate pseudo-functional transition and path delay tests. All works are summarized in Chapter 5.

Chapter 2

Background

2.1 Boolean Satisfiability (SAT)

Boolean Satisfiability (SAT) is a well-known constraint satisfaction problem, which has a wide variety of applications in the fields of computer-aided design and Artificial Intelligence. Given a propositional formula f that depends on a set of variables V, the Boolean Satisfiability problem is to determine whether there exists a satisfying assignment for V that evaluates f to true, or no such assignment exists. If such assignment exists, f is $satisfiable$. Otherwise, f is $unsatisfiable$. The SAT problem was known to be a NP-complete problem a few decades ago [13].

For most modern SAT solvers, Conjunctive Normal Form (CNF) is the widely used format for the propositional formula f. In a CNF formula, a boolean expression is represented as a conjunction of one or more disjunctive clauses. Each clause is a disjunction of one or more literals. A literal is a variable or its negation. For a CNF formula to be satisfiable, all of the disjunctive clauses must be true simultaneously. For instance, for a CNF formula $f = (a)(\overline{a}+b+c+\overline{d})(c+d)(d)$, one $satisfiable$ assignment is $a = 1, b = 0, c = 1, d = 1$. There exist polynomial algorithms [39] that can transform an arbitrary propositional formula into a equivalent CNF formula. The equivalent CNF formula is satisfiable if and only if the original formula is satisfiable. Similarly, a Boolean

circuit may be encoded as a satisfiability equivalent CNF formula [27]. Table 2.1 summarizes the conjunctive normal forms of some basic gates, where x_1 and x_2 are inputs, z is the output.

Table 2.1: Conjunctive normal forms of basic gates

Gate Type	Conjunctive Normal Form
AND	$(\overline{z}+x_1)(\overline{z}+x_2)(z+\overline{x_1}+\overline{x_2})$
OR	$(z+\overline{x_1})(z+\overline{x_2})(\overline{z}+x_1+x_2)$
NAND	$(z+x_1)(z+x_2)(\overline{z}+\overline{x_1}+\overline{x_2})$
NOR	$(\overline{z}+\overline{x_1})(\overline{z}+\overline{x_2})(z+x_1+x_2)$
XOR	$(\overline{z}+x_1+x_2)(\overline{z}+\overline{x_1}+\overline{x_2})(z+\overline{x_1}+x_2)(z+x_1+\overline{x_2})$
NOT	$(\overline{z}+\overline{x_1})(z+x_1)$

Figure 2.1 shows a simple circuit. Using Table 2.1, it is straightforward to translate it to a CNF formula: $(G_1+G_4)(\overline{G_1}+\overline{G_4})(\overline{G_2}+G_5)(\overline{G_3}+G_5)(G_2+G_3+\overline{G_5})(G_4+\overline{G_6})(G_5+\overline{G_6})(\overline{G_4}+\overline{G_5}+G_6)$. We can see that the number of the clauses in the translated CNF formula is linear to the number of the gates in the circuit, and the translation can be done very efficiently.

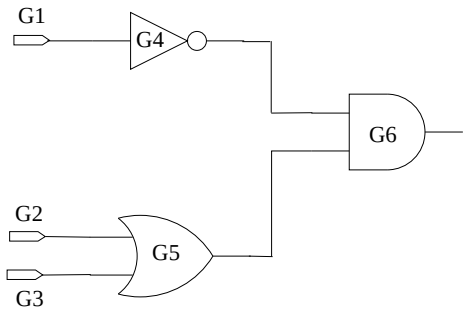

Figure 2.1: A Simple Circuit

Most modern SAT solvers are based on the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [10, 11], which performs a backtracking decision tree branching on the variables to solve the SAT problem. The DPLL algorithm is sound and complete. It finds a solution if and only if the formula is satisfiable. The introduction of conflict analysis [35, 53] and optimized Boolean Constraint

Propagation (BCP) with two-literal watching [36], has significantly accelerated the performance of present day SAT solvers.

The basic procedure of DPLL-based SAT solver is shown in Algorithm 1. The initial step consists of some preprocessing, during which it may be discovered that the formula is unsatisfiable. The outer loop starts by picking an unassigned decision variable and assert its implications (decide-next-branch). If no such variable exists, a solution has been found. Otherwise, the variable assignments deducible from this decision are made through BCP (using deduce). It typically consists of iterative application of the unit clause rule, which is invoked whenever a clause becomes a unit clause, i.e., all but one of its literals are false and the remaining literal is unassigned. According to the rule, the last unassigned literal is implied to be true C this avoids the search path where the last literal is also false, since such a path cannot lead to a solution. A conflict occurs when a variable is implied to be true as well as false. If no conflict is discovered during BCP, then the outer loop is repeated by choosing the next variable for making a decision. However, if a conflict does occur, backtracking is performed within an inner loop in order to undo some decisions and their implications and backtrack to the decision level that was responsible for the conflict. Backtracking also analyze the conflict and derive a clause that constraints the conflict space. If we backtrack to the root decision level, due to a conflict, the formula is declared unsatisfiable since the entire search space has been exhausted. Otherwise, we will find an assignment to all the variables in the formula and declare the formula to be satisfiable. We proceed in this manner until all the variables have been assigned a value.

Various variable ordering heuristics were investigated to pick a free decision variable, depending on the application of SAT solving. It was shown that the time taken to solve the problem depends significantly on the variable order chosen. We refer the reader to [52], for a survey on SAT solving and further information on SAT solvers. We also point out that a number of SAT solvers, such as [12, 34], are available in the public domain for research purpose. Further, we refer the reader to [41] for a recent survey on the recent advances on using SAT solvers for formal verification. In recent years, significant amount of research works have been done on using SAT solvers for Model Checking and Equivalence Checking. We will discuss these works in details in

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
   }
```

the subsequent sections.

2.2 Logic Implications

Logic implications capture the effect of asserting logic values throughout a logic circuit. As implications capture relationships among two or more signals in the circuit, they can be viewed as constraints which can potentially help to constrain the search for several electronic design automation (EDA) problems, such as design verification [25], multi-level logic optimization [17, 26], automatic test pattern generation (ATPG) [23, 47], logic and fault simulation [19], fault diagnosis [3], redundancy identification [33], etc.

There is a body of work that has dealt with computing static implications among signals. Static implication is obtained by a procedure which sets each gate in the Boolean circuit to logic value 1 and 0, and analyzing the result of propagating these values throughout the circuit. The static logic implications are made up of direct, indirect and extended backward implications. Direct implications can be easily determined whereas indirect and extended backward implications [46, 55, 56] are non-trivial, and their discoveries require combination of simulation, transitive law and contrapositive law [47].

In general, the total number of implications associated with the entire circuit can be exponential in the size of the circuit. Thus, a memory efficient technique must be used to store the implications associated with each gate. A graphical representation to store implications in a sequential circuit was proposed by Zhao et al. [56], which allows implications between nodes across different time-frames to be easily modeled. This representation has the advantage of being used for sequential circuits without suffering from the problem of memory explosion. For a given circuit with K gates, the total number of nodes in this graph is $2K$, since each gate can take on a logic value of 0 or 1. A directed edge between two nodes represents an implication between the two nodes. For example, An edge from node a to node b means that a implies b ($a \rightarrow b$). The weight associated with an edge represents the relative time frame associated with the implication where the current time-frame is

time-frame 0. When an implication propagates across a D flip-flop, the time frame is incremented or decremented accordingly. Also, by representing the sequential implications as a graph, transitive closure of a node can be easily obtained using the depth-first search technique. Another advantage is that whenever a new indirect or extended backward implication is computed, its contrapositive implication (from contrapositive law) can be immediately added to the implication graph.

In the following description of logic implications, we use (g,v,t) represent assign logic value v to gate g in time frame t. $imply(g=v)$ represents the set of implications from assign logic value v to gate g.

2.2.1 Direct Implications

Direct implications of a gate g consist of implications associated with the gates driving and driven by g. Such implications are easily computed by traversing through the immediate fanins and fanouts of the 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 gate's fanins. Table 2.2 gives the controlling value (cv), the non-controlling value (ncv) and the inversion value (i) for several basic gates. Note that the non-controlling value is just the complement of the controlling value. Direct implications can be easily learned during an ATPG process.

Table 2.2: Controlling, Non-Controlling and Inversion values

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

Consider the example circuit in Figure 2.2. A logic value of $A=1$ would imply $B=1, D=1$, it also imply $H=1, I=1$. The direct implication of $A=1$ is the set $(A,1,0),(B,1,0),(D,1,0),(H,1,0),(I,1,0)$.

Similarly, the direct implication of $B = 1$ is $(B,1,0),(C,1,0),(A,1,0)$. These implications are stored in the implication graph. Figure 2.3 (a) shows the complete set of direct implications from the root $A = 1$.

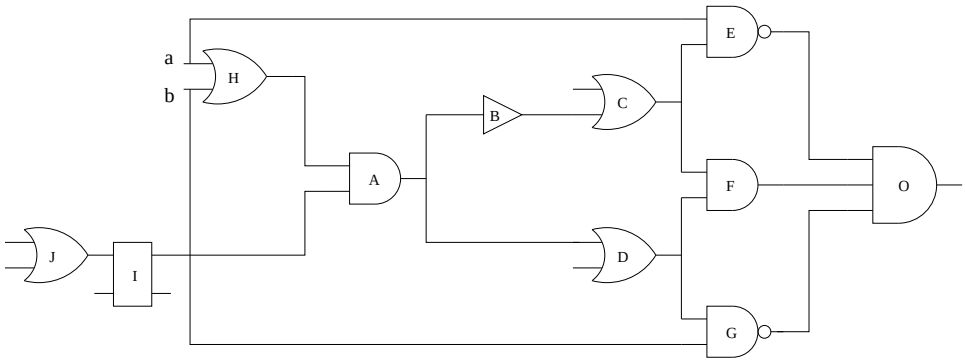

Figure 2.2: Example sequential circuit

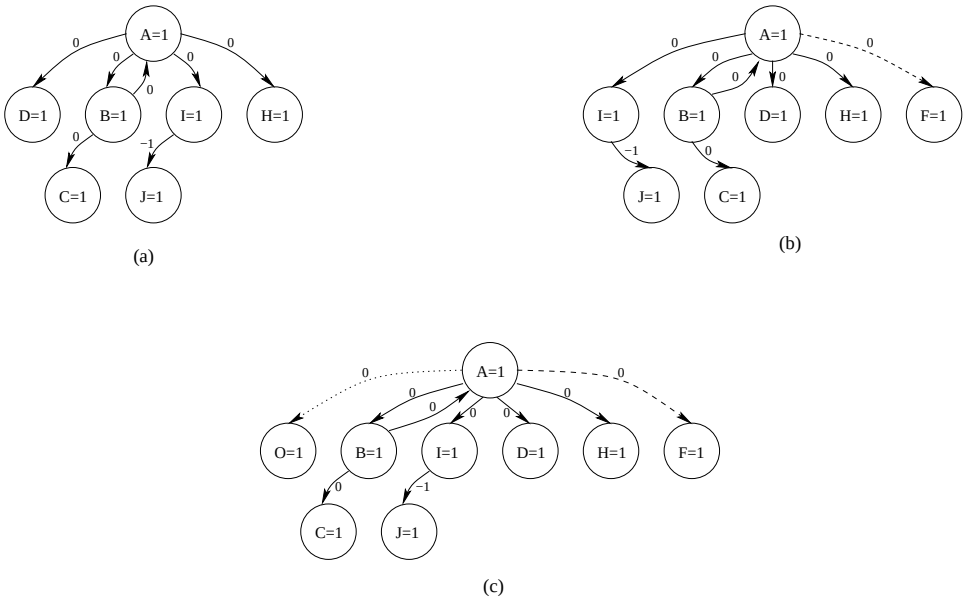

Figure 2.3: Implication Graph

2.2.2 Indirect Implications

Schulz et al. were the first to improve the quality of implications by computing indirect implications [46]. Computation of indirect implications involves extensive use of contra-positive law, transitive law, and logic simulation. The indirect implications of gate g set to value v are computed by inserting the gate values pertaining to all its direct implications onto the circuit, and performing logic simulation. All gates whose output value changes from a dont-care to logic 0 or 1, form the indirect implications of $(g = v)$. This could be represented as $imply(g = v) \equiv imply(g = v) \bigcup logicSim(imply(g = v))$.

In Figure 2.2, consider $A = 1$, by inserting the implications of $A = 1$ into the circuit and do logic simulation, we obtain $F = 1$. Here either $C = 1$ nor $D = 1$ implies a logic value of gate F, but together they imply $F = 1$. So indirectly $A = 1$ would imply $F = 1$. This new implication is added as an additional outgoing edge from $A = 1$ in the implication graph as shown in Figure 2.3 (b).

2.2.3 Extended backward implications

The extended backward implications [55] extend the concept of indirect implications to unjustified signals to obtain more signal relationships. These implications are computed by considering the target gate and the unjustified output specified gates in the implication list of the target gate. So the extended backward implication aim at the relations of the implying gate with respect to unjustified gates in its implication list.

In Figure 2.2, gate $H = 1$ is an unjustified gate in the implication list for $A = 1$, as none of its inputs is implied to a value of logic 1. Thus H is a candidate of extended backward implications. To perform extend backward implications on H, a transitive closure is first performed for each of its unspecified inputs, obtaining $imply(a = 1)$ and $imply(b = 1)$ respectively. The implications of $A = 1$ are simulated together with $imply(a = 1)$ and $imply(b = 1)$ in turn, create a set of newly found logic assignments for each input. For example, when the implications of $(a = 1)$ and $(A = 1)$

are simulated, the new assignments found are $(E,0,0)$ and $(O,0,0)$. For the implications of $(b=1)$ and $(A=1)$, the new assignments are $(G,0,0)$ and $(O,0,0)$. All logic assignments that are common among these sets are the extended backward implications of gate $A=1$. In this example, $(O,0,0)$ is the common assignment, it is the extended backward implication of $A=1$. This new implication is added as an additional outgoing edge from $A=1$ in the implication graph as shown in Figure 2.3 (c).

2.2.4 Other Works on Implication

There are some other works on logic implication that worth mention. Cox et al. introduced the use of a 16-valued algebra and reduction lists to determine node assignments [42]. A transitive closure procedure on implication graph was proposed by Chakradhar et al. [7]. A complete implication engine based on recursive learning [24] proposed by Kunz et al. can capture all pair-wise relationships in a circuit. However, for large circuits, the depth of recursion is kept low to avoid excessive computational costs. Due to the NP-hard nature of finding all the implications for a given set of nodes, the practicality of such complete algorithms is limited. Syal et al. proposed the concept of the extended forward implications [48], which use implication frontiers to capture additional pair-wise implication relationships. To further the application of implications, static learning was extended to dynamic learning [23, 46].

2.3 Data mining

Progress in digital data acquisition and storage technology has resulted in the growth of huge databases. We have grown accustomed to the fact that there are tremendous volumes of data filling our computers, networks, and lives. However, frequently only a small fraction of the data can be used because (1) the data volumes are simply too large to manage, or (2) the data structures themselves are too complicated to be analyzed effectively. Analysis of these complex, information-rich data sets and extraction of useful information is of great interest to many fields including business, science, and engineering. The discipline concerned with this task has become known as data

mining. Data mining is the process of applying a computer-based methodology for discovering knowledge from (often large) observational data sets. The knowledge could be various models, patterns and derived values from a given collection of data. Examples include linear equations, rules, clusters, graphs, tree structures, and recurrent patterns in time series. The general experimental procedure adapted to data mining problems involves the following steps:

- State the problem and formulate the hypothesis
- Collect the data
- Preprocessing the data
- Estimate the model
- Interpret the model and draw conclusions

Data mining techniques can be classified as follows:

Statistical Methods where the typical techniques are Bayesian inference, logistic regression, ANOVA analysis, and log-linear models.

Cluster Analysis are the common techniques of which are divisible algorithms, agglomerative algorithms, partitional clustering, and incremental clustering.

Decision Trees and Decision Rules are the set of methods of inductive learning developed mainly in artificial intelligence. Typical techniques include the CLS method, the ID3 algorithm, the C4.5 algorithm and the corresponding pruning algorithms.

Association Rules represent a set of relatively new methodologies that include algorithms such as market basket analysis, apriori algorithm, and WWW path-traversal patterns.

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

Genetic Algorithms are very useful as a methodology for solving hard optimization problems.

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

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

2.3.1 Association rule mining

An association rule is the most common form of local-pattern discovery; it finds interesting patterns in a database which probably cannot be explicitly articulated. Among all data mining techniques, mining association rules are one of the major techniques. It is a form of data mining that most closely resembles the process that most people think about when they try to understand the data mining process. Association rule mining retrieves all highly correlated relations in the database.

Several notations commonly used in association rule mining are:

1. *item*: The basic element in a database, e.g., an item sold in a supermarket.
2. *k-itemset*: A set of k different items.
3. *basket*: A set of items, e.g., the things a customer buys.
4. *support for k-itemset*: The number of baskets containing all items in a k-itemset.
5. *frequent itemsets*: Given a support threshold, sets of items that appear in $\geq threshold$ baskets.

Table 2.3 shows an example database DB. In Table 2.3 there is a total of six items, *A*, *B*, *C*, *D*, *E* and *F*, and four transactions. Each mark of '1' in the table indicates that the item in that column occurred in the corresponding transaction. For example, the 3-itemset {B,E,F} occurs in transactions 2 and 3, thus the support of {B,E,F} is 2. If the support threshold is set to be $s = 1$, then {B,E,F} is a frequent 3-itemset since more than 1 baskets contained {B,E,F}.

Table 2.3: Database DB

	Items					
Trans ID	A	B	C	D	E	F
1	1			1		1
2		1			1	1
3	1	1	1		1	1
4		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 2.4 shows an example lattice over the set of items $I = \{x_1, x_2, x_3, x_4\}$. 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.

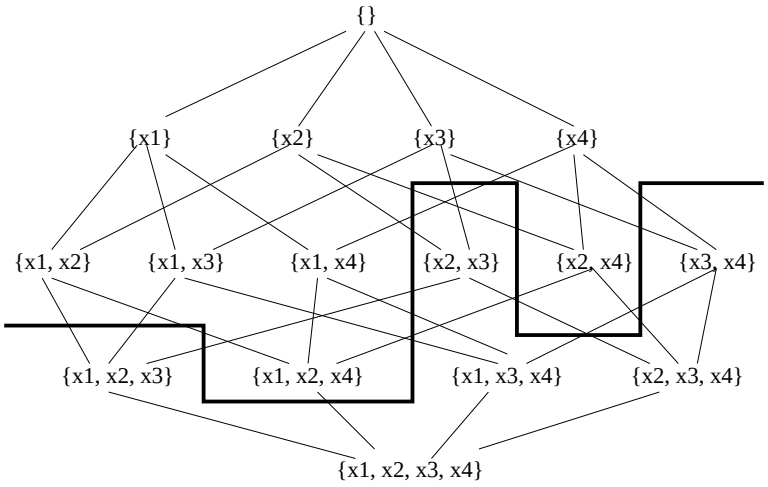

Figure 2.4: The lattice for the itemsets I

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. With this pruning, the search space is drastically reduced.

2.3.2 Apriori technique

Apriori [1, 2] is an efficient and popular methodology for association-rule mining. It combines breadth-first search with counting of occurrences of candidates. Using downward closure property of itemset support, Apriori prunes search space this way: Look at all the subsets of size $|k|$ of a candidate $(k+1)$-itemset, whenever there is at least one of those subsets infrequent, then prune this $(k+1)$-itemset without counting its support.

For example, applying Apriori algorithm on itemsets I in Figure 2.4, after 2 mining iterations, we obtain infrequent 2-itemsets: $\{x_2, x_3\}$ and $\{x_3, x_4\}$. For the 3-itemsets mining iteration, we only need to count support of 3-itemset $\{x_1, x_2, x_4\}$. The other 3-itemsets all have infrequent subsets, thus can be directly pruned.

Apriori computes the frequent itemsets in the database through several iterations. Each iteration has two steps: (1) candidate generation and (2) candidate counting and selection. In the first step of the first iteration, the generated set of candidate itemsets contains only 1-itemsets (i.e., all items in the database). In the second step, the algorithm counts the support s for each itemset by searching through the entire database. Finally, only 1-itemsets whose support is above a specified threshold will be selected. Thus, after the first iteration, all 1-itemsets whose supports are greater than the threshold will be selected.

In the second iteration, the 2-itemset candidates are selected only from the 1-itemsets obtained in the first iteration. Even though the 2-itemsets denote all possible pairs of items, the number of 2-itemsets will not include all possible items because we are considering only a filtered 1-itemset. The pruning is based on the observation that if an itemset is frequent, then all its subsets must be frequent as well. Therefore, before entering the candidate-counting step, the algorithm discards every candidate itemset that has an infrequent subset. This process iterates, and in any given iterations i, all the frequent i-itemsets will be obtained.

For example, consider the database illustrated in Table 2.3. Assume that the threshold is set to be $s = 50\%$. Then, an itemset is considered frequent if it is contained in at least 50% of the transactions. In each iteration, the Apriori algorithm constructs a candidate set of frequent itemsets, counts the number of occurrences of each candidate, and based on the predetermined minimum support $s = 50\%$ the frequent itemsets are determined. These two steps of Apriori are given in Table 2.4. Initially, six 1-itemsets are possible as illustrated in C_1 and, of these, only four are computed as frequent in L_1 because their support is greater than or equal to two, or $s \geq 50\%$.

Table 2.4: First iteration of Apriori

Initial 1-itemset C_1	count	s(%)	frequent itemset L_1	count	s(%)
{A}	2	50	{A}	2	50
{B}	3	75	{B}	3	75
{C}	1	25			
{D}	1	25			
{E}	3	75	{E}	3	75
{F}	3	75	{F}	3	75

Next, to discover the set of frequent 2-itemsets, the 1-itemsets are used. Because any subset of a frequent itemset must be frequent, the Apriori algorithm uses $L_1 \odot L_1$ to generate the candidates. The operation $\odot$ is defined as

$$L_k \odot L_k = \{X \cup Y \, where \, X, Y \in L_k, |X \cap Y| = k-1\}.$$

The operation $\odot$ guarantees that each candidate k-itemset is generated from a frequent (k-1)-itemset. For k = 1 the operation represents a simple concatenation. Therefore, C_2 consists of 2-itemsets generated by the operation $\frac{|L_1| \cdot (|L_1|-1|)}{2}$ as candidates in the second iteration. In our example, this number is $\frac{(4 \times 3)}{2} = 6$. Scanning the database DB with this list, the algorithm counts the support for every candidate, and in the end, a frequent 2-itemsets L_2 for which $s \geq 50\%$ is obtained. The results for the second iteration are given in Table 2.5.

By carrying these steps iteratively, the algorithm could mine all possible association rules. As we can see from the example, the key to reduce the computation complexity is by pruning the number of itemsets with support $s \geq threshold$ in each iteration.

Table 2.5: Second iteration of Apriori

Initial 2-itemset C_2	count	s(%)	frequent itemset L_2	count	s(%)
{A,B}	1	25			
{A,E}	1	25			
{A,F}	2	50	{A,F}	2	50
{B,E}	3	75	{B,E}	3	75
{B,F}	2	50	{B,F}	2	50
{E,F}	2	50	{E,F}	2	50

2.3.3 Min-Hashing and Locality-Sensitive Hashing

While the Apriori algorithm is efficient in mining the highly correlated relations with high support, the Min-hashing and Locality-Sensitive Hashing algorithms [6, 8, 18] target on the low support yet high-correlation relations in the database.

In low support, high-correlation mining, the similarity of two items i and j is the ratio of the sizes of the intersection and union of i and j.

$$Sim(i,j) = \frac{|\, i \bigwedge j \,|}{|\, i \bigvee j \,|}$$

For example, the similarity of item A and B in Table 2.3 is 0.2. The key idea of the hashing algorithm is to hash each column of item i to a small signature $Sig(i)$ such that two items i and j are highly similar if and only if $Sig(i)$ and $Sig(j)$ are highly similar.

The Min-hashing algorithm hashes each column of database using several different hash functions. In the signature matrix, each row maintains the lowest hash value of each column in database in which that column has a 1. Figure 2.5 shows a simple example of Min-hashing, where the database has 4 items and 7 baskets, three hash functions, H1, H2, and H3, are used to generate the signature matrix. $Sim(i,j)$ is the probability that $Sig(i) = Sig(j)$ and is computed as the fraction of the rows in the signature matrix that match. Table 2.6 shows the similarity of the column pairs for both the original database and signature matrix on the second and third rows, respectively. For example, columns 1 and 2 in the original database have no items in common, leading to a similarity of 0. In the signature matrix, between columns 1 and 2, no signatures match, thus the similarity is

also 0. As the number of hash functions increases, the similarity values will be closer between the original and signature matrices.

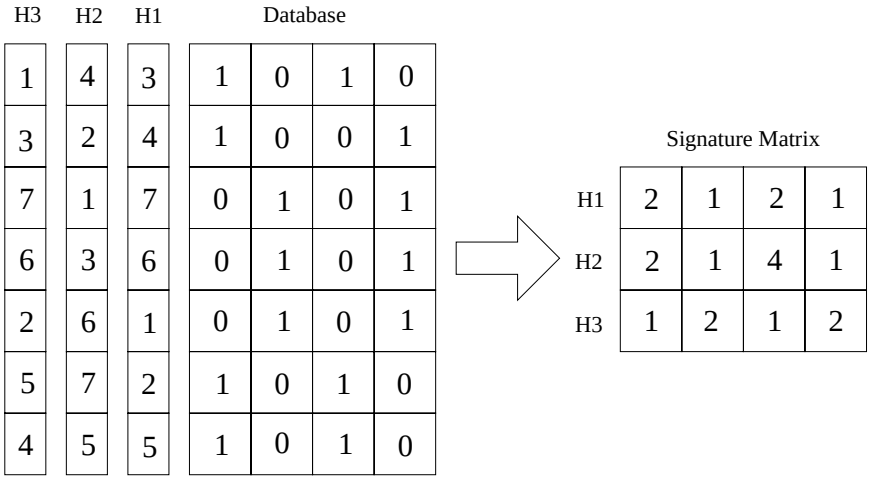

Figure 2.5: Min-hashing Example

Table 2.6: Similarity Comparison

Similarity	(1,2)	(1,3)	(1,4)	(2,3)	(2,4)	(3,4)
database	0	0.75	0.14	0	0.75	0
Sig matrix	0	0.67	0	0	1	0

After Min-hashing, the signatures for all N columns are stored in the signature matrix. Similar signatures will refer to similar columns. However, to identify similarity among k columns, with $2 < k < N$, it may require an exponential number of column combinations. Locality-Sensitive Hashing (LSH) is a technique to identify similar columns with sub-quadratic complexity. In LSH, the signature matrix is divided to b bands of r rows. Each band is hashed into k buckets. Candidate similar columns are those that hash to the same bucket for at least 1 band. Tuning b and r could catch most similar columns with least non-similar columns (false negative). Figure 2.6 shows the basic idea of LSH. In this figure, the portion of each column corresponding to a given band is hashed and placed into the respective buckets. All columns that appear in any one bucket are grouped together as a set of similar items.

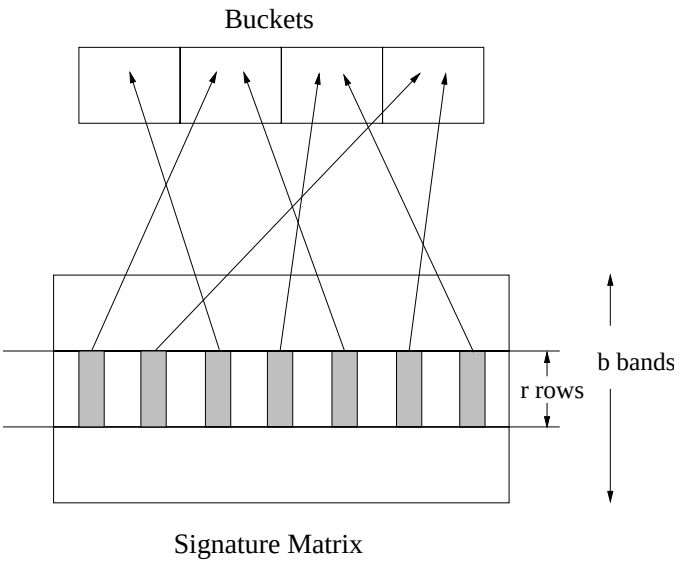

Figure 2.6: Locality-Sensitive Hashing

Chapter 3

Bounded Sequential Equivalence Checking

In this chapter, we present a novel technique on mining relationships in a sequential circuit to discover global constraints. In contrast to the traditional learning methods, our mining algorithm can find important relationships among several nodes efficiently. We utilize the domain knowledge to prune the search space for our mining process. We propose two types of domain-based constraints in this chapter: the distance based and the region based. Incorporating domain constraint in our mining algorithm, the nodes involved may often span several time-frames, thus improving the deductibility of the problem instance. Experimental results demonstrate that the application of these global constraints to SAT-based bounded sequential equivalence checking can achieve one to two orders of magnitude speedup. In addition, because it is orthogonal to the underlying SAT solver, it can help to enhance the efficiency of typical SAT-based verification flows.

3.1 Introduction

There is a body of work that has dealt with computing non-trivial Boolean relations among signals via static implications. However, they are mostly focused on pair-wise signal relationships. Methods to extend learning to more than two variables have been attempted. A method to find multi-node static implications has been proposed which selects node pairs based on circuit structure information [14]. In [38], a local search is performed over a subset of selected variables to

find multi-variable relationships. As finding relationships among multiple signals may involve an exponential number of signal combinations, methods that require low computational costs must be investigated.

A naive method to compute all relationships among three variables of the form $(a \cdot b \rightarrow c)$ can have a cubic cost in the number of nodes in the circuit. Instead of exhaustively searching for all possible pairs of signals *a* and *b*, we obtain a subset of hard-to-find implications via *data mining*. To the best of our knowledge, no work currently exists that employs data mining [9] algorithms to capture global constraints among signals in the circuit.

In our proposed mining approach, we view the set of all relationships among three signals as an ocean of information, and we develop a mining strategy to obtain the subset of global invariants from all the possible relationships. One advantage of using domain knowledge within the data mining process is that it can constrain the search space, thus make the mining process much more efficient [4, 37]. We utilize the domain knowledge in the circuit to constrain the search space for the mining approach, which we name *domain constraint*. We propose two domain constraint strategies: distance based and region based. These constraints are incorporated in our mining approach. We describe the parameters used in our mining approach which are targeted at increasing the usefulness of the learned constraints, while reducing the computational overheads. While numerous applications can benefit from these global constraints, we demonstrate the efficacy of mined global constraints by applying them to SAT-based bounded sequential equivalence checking, a special case of bounded model checking (BMC) [5, 22] on product machines. Our experimental results show that the proposed mining algorithm can drastically reduce the computation complexity of finding multi-node relationships. The global constraints obtained can be used to simplify the problem instance and prune the search space of the SAT solver. Consequently, one to two orders of magnitude speedup was achieved when compared with the platform without such learning for the SAT-based bounded sequential equivalence checking. Finally, because the technique is orthogonal to the underlying SAT solver, it can help to enhance typical SAT-based BMC flows.

The rest of the chapter is organized as follows. Section 3.2 gives the motivation, the concept of

domain knowledge and bounded sequential equivalence checking. Section 3.3 presents the basic flow of our mining framework. Section 3.4 discusses the experimental results, and Section 3.5 concludes the chapter.

3.2 Motivation

Among many sub-areas of data mining, mining association rules is one major sub-area. An association rule is a simple probabilistic statement about the co-occurrence of certain events in a database. An association rule takes the following form:

$$\text{If } A = 1 \text{ and } B = 1, \text{ then } C = 1 \text{ with probability } p$$

We note that the association rule is nothing but a probabilistic implication. Considering the successes of data mining in finding complicated association rules on huge database, we want to know if it is possible to use this technique help us identify powerful constraints efficiently.

3.3 Global constraint mining

In this section, we describe our mining approach which efficiently captures signal relationships that involve three circuit nodes. In our approach, the mining database is first built by running a number of logic simulations on the circuits and record the values for all circuit nodes. For example, consider the combinational portion of a sequential circuit shown in Figure 3.1, where G1 and G2 are primary inputs (PI), and G3 is a pseudo primary input (PPI). When performing logic simulation, random vectors are used. The values for the PPIs in the first time-frame are set to don't cares (X). Consider the four random patterns {0, 0, X}, {0, 1, X}, {1, 0, X}, {1, 1, X}. After logic simulation, the mining database is shown in Table 3.1.

By setting a threshold, we can use the mining algorithm to compute the global constraints from the table. However, the mining constraints obtained will be probabilistic, and we need to verify if they are true global constraints in the sequential circuit via an additional verification check, which

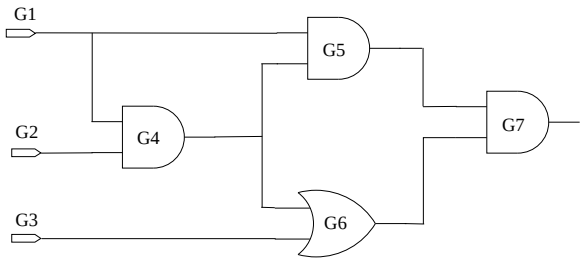

Figure 3.1: Combinational portion of a sequential circuit

Table 3.1: Mining database

Vector	*G*1	*G*2	*G*3	*G*4	*G*5	*G*6	*G*7
V1	0	0	X	0	0	X	0
V2	0	1	X	0	0	X	0
V3	1	0	X	0	0	X	0
V4	1	1	X	1	1	1	1

is described in Section 3.3.4.

3.3.1 Global constraint mining framework

Our framework of mining multi-node relationships is shown in Figure 3.2. First, a miter circuit is constructed from the sequential circuit and its optimized version. Then the circuit is unrolled to k time frames. M random input vectors are generated and applied to the circuit, and the logic value of each gate is recorded in the mining database. When performing logic simulation, the initial state of PPIs is always set to don't cares. This is because we are interested in finding potential 3-node relationships that do not depend on any state. Thus, the signal relationships we obtain will be more likely to be globally true. The mining database is thus obtained, where one dimension lists the circuit nodes, and the other dimension lists the vectors. Then we mine the 3-node constraints from the database.

Unlike general data mining where an item is either included in or excluded from a transaction, in our work, we need to learn relations among signals with both logic 0 or logic 1 values. Furthermore, as each item (gate) is 3-valued, a don't-care value indicates that the node is absent from

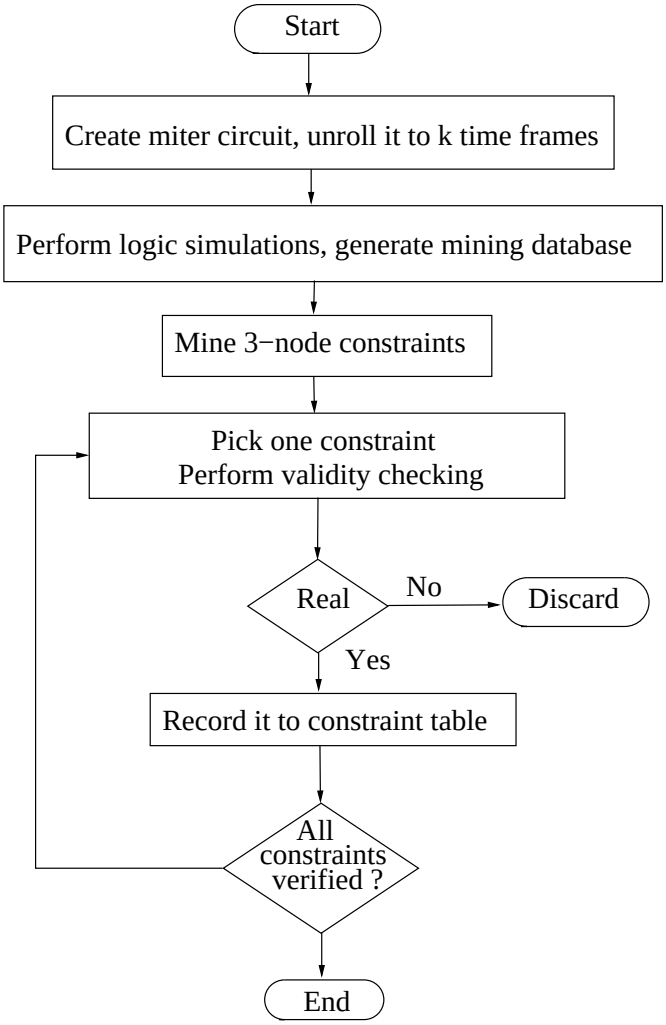

Figure 3.2: Mining framework

that entry in the database. To capture the effect of both logic 0 and 1 values, we need to compute association rules for both logic 0 and 1. We first calculate the signal probability of 0 and 1 for each gate with respect to the random test set. For each gate g_i, the probability of g_i having logic 0, (P_i^0), and the probability of g_i having logic 1, (P_i^1), are computed as follows:

$$P_i^0 = \sum(g_i = 0)/M$$
$$P_i^1 = \sum(g_i = 1)/M$$

where M is total number of vectors simulated. After P_i^0 and P_i^1 have been computed, we check if the probability is within the following range: $threshold_{low} \leq P_i^v \leq threshold_{high}$. If so, we mark it as a candidate gate. In our experiments, we set $threshold_{low}$ to 0.0 and $threshold_{high}$ to 0.1. The reason for choosing these threshold values is that the corresponding controllability values of these nodes are very low. Therefore, assigning a value to the gate might be regarded as difficult, and the constraints we learned are more likely to be hard-to-learn relationships. As we will see from the experiments, such constraints are very useful for the SAT solver.

After obtaining the initial candidate gate list, we will attempt to mine the potential candidate pairs from the database. Only those pairs whose combinations appear in the database within the same probability ranges are selected, as explained earlier on mining association rules. Furthermore, domain knowledge is used to reduce the number of candidates, which is described next.

3.3.2 Incorporating Domain Knowledge

Generally speaking, domain knowledge is the knowledge which is valid and directly used for a pre-selected domain of human or computer activity [15]. For example, most text editors encode specific knowledge about fonts and formatting of text. Domain knowledge is a useful way of constraining or pruning the search space for a mining process, enhancing the performance of the system. In data mining, if the user has some background information (domain knowledge), the mining system could be able to use it. Incorporating available domain knowledge into data mining

techniques could improve the quality of the discovered rules and could constrain the search space and improve the overall efficiency of the mining process.

We apply the domain knowledge in selecting the candidate pairs. The domain knowledge we propose is the structural information of the underlying sequential circuit. When we choose the candidate pairs, we favor those nodes that are not locally close. Because if two nodes are locally close, the correlation of their signals are generally higher than if the two nodes are far away. The mined constraints of locally closed pairs are more likely to be local relationships. As we are interested in mining global relationships, we employ the domain knowledge of structural information to help us avoid locally close candidate pairs.

In this chapter, we propose two types of domain constraints: the distance based and the region based. In the distance-based domain constraint, if the distance between 2 nodes is within a preset limit, they are considered locally close and will not be considered as a potential candidate pair. The distance between 2 nodes is defined as the number of nodes along the shortest path between two nodes. For example, in Figure 3.1, the distance between G1 and G7 is 2; the distance between G1 and G3 is 3. To select a candidate pair (a, b), after picking a gate a from the candidate gate list, we perform a depth-first search from node a, the search depth is the distance limit. All gates within the distance limit will not be considered to be candidate gates.

In the region-based domain constraint, each time-frame of the unrolled circuit is partitioned into 2 approximately equal-sized regions. By equal size, we mean that the time frame of circuit is divided at a circuit level, l, such that the number of gates in before and after the level l is approximately equal. For example, Figure 3.3 shows one time frame of the unrolled circuit, the region partition is at level 1, which makes 2 equal regions. To select a candidate pair (a, b), when a gate a is chosen from the candidate gate list, gate b must not in the same or adjacent region of gate a. This forces b to come from a different time-frame and is not locally close to a.

In the distance-based domain constraint, we favor mining of node relationships that are not locally close to each other. On the other hand, in the region-based domain constraint, the mining toward relationships of nodes is biased toward those that cross time frame boundaries and are not

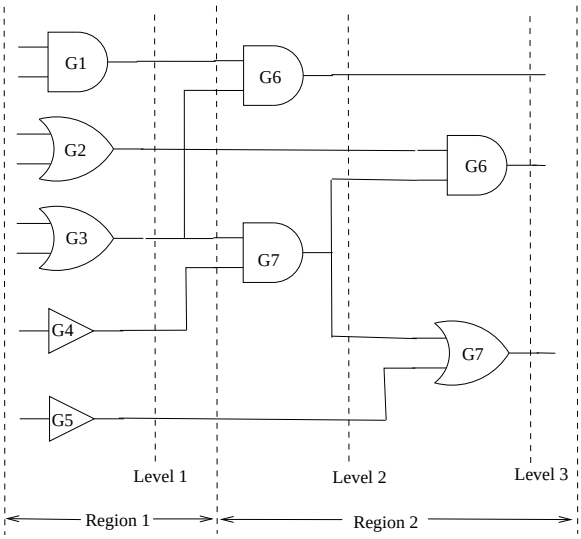

Figure 3.3: 1 time frame of unrolled circuit

locally close.

Besides the two mining approaches that have domain constraints, We also include a basic mining approach without any domain constraints. In this basic mining approach, when we choose the candidate pairs, we simply check the two gates' fanin gate lists. If 2 gates that have common fanin gates or if one gate is a successor of the other, we eliminate this pair. In the experiments, the efficiency of the tow proposed domain constrained mining approaches are compared with this basic mining approach.

3.3.3 Three-node Constraints

For each selected candidate pair (a, b), we will find the third node by computing the implied gates by the pair. We first build fanout cones for nodes a and b, then we calculate the overlapping area; only those gates in the overlapping area are considered for the potential implication gates. Note that the cones may cross several time-frame boundaries. After computing the implication gates for each pair of nodes, we check the consistency of implication gates. That is, we check the value of each implication gate in the database to see if it is consistent with every vector that was simulated.

In Algorithm 2 below, we describe the procedure of consistency checking.

For example, if the pair of gates is ($a = 1$ and $b = 1$), and gate c is a candidate implication gate, we check gate c's logic value for all $a = 1$ and $b = 1$ conditions in the database. If in all cases where $a = 1$ and $b = 1$, c appears in only one polarity, then we consider c as a *consistent* gate. Any inconsistent implication gate is discarded. For a consistent gate c, if $c = 0$, then the potential relationship is $a \cdot b \rightarrow \overline{c}$; if $c = 1$, then the relationship is $a \cdot b \rightarrow c$. We skip the $c = X$ instances. We record all these potential consistent 3-node constraints. The details of this 3-node constraint mining algorithm are shown in Figure 3.4.

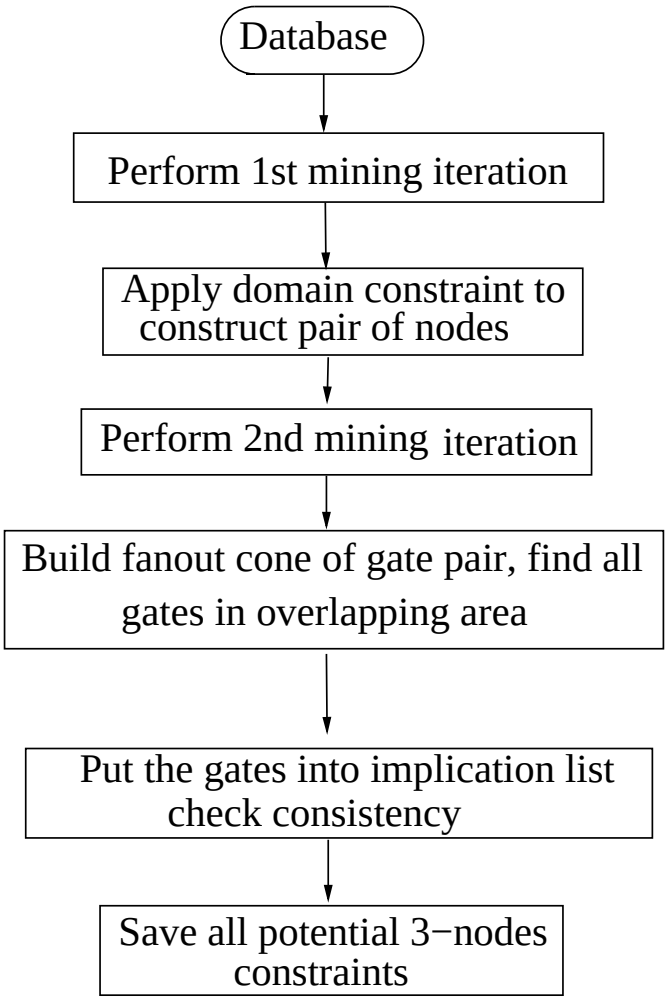

Figure 3.4: 3-node mining flow

Algorithm 2: Check consistency

```
1  check_consistency(g1, g2){
     /*Given a selected pair of gates (g1=val1, g2=val2), mining database is db, implication
       gate value is impVal, M is total number of simulation vectors.                    */
2    for (int i = 1; i ≤ total_number_of_gates; ++i) do
3      if (i in implication_gate_checklist) then
4        impVal = X ;
         /*X means don't care                                                            */
5        for (int j = 0; j < M; ++j) do
6          if ((db[j][g1]==val1) && (db[j][g2]==val2)) then
7            if (db[j][i] ≠ X && impVal == X) then
8              impVal = db[j][i] ;
               end
9            else if ((impVal == 0) && (db[j][i] == 1)) then
10             impVal = Invalid ;
11             break ;
               end
12           else if ((impVal == 1) && (db[j][i] == 0)) then
13             impVal = Invalid ;
14             break ;
               end
             end
           end
         end
         /*save consistent implication gate to implication gate list                     */
15       if (impVal == 0 || impVal == 1) then
16         implication_gate_list.push_back(i) ;
         end
       end
   }
```

3.3.4 Validity check of the mined constraints

We need to verify each mined constraint to determine if the constraint is indeed globally correct. We do so by leaving all initial PPIs' states unconstrained. For example, if the potential constraint $a \cdot b \rightarrow c$, we add three unit clauses (a) (b) $(\neg c)$ to the original CNF formula of the unrolled miter circuit. If the SAT solver returns UNSAT for the augmented CNF formula, then $a \cdot b \rightarrow c$ is indeed a globally true constraint, since the validity check puts no constraints on the initial state value. On the other hand, if the SAT solver returns SAT for the formula, no conclusion can be made for this 3-node relationship. Such relationships are removed.

We use zChaff [36](2004.11.15 Version) 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. Each time we complement the implication clause, only the complement clauses are added to CNF formula to be verified. After verification, we only need to remove these unit clauses. This could greatly reduce the overhead in checking the validity of the mined global constraints.

3.3.5 Application to SAT-based bounded sequential equivalence checking

The global constraints can be applied to various verification problems. In our work, these constraints are applied to SAT-based bounded sequential equivalence checking between an original sequential circuit with its optimized version. SAT-based bounded sequential equivalence checking is a special case of SAT-based bounded model checking, where the property is on trying to check if the miter circuit output is a constant 0 from an initial state. The miter circuit is built by connecting a sequential circuit to its optimized version in the following manner. The corresponding primary inputs are tied together, and the corresponding primary outputs are connected by XOR gates. Then all XOR gates are connected to an OR gate. Figure 3.5 shows the model of our miter circuit.

In our SAT-based bounded sequential equivalence, the miter is unrolled k time frames, and

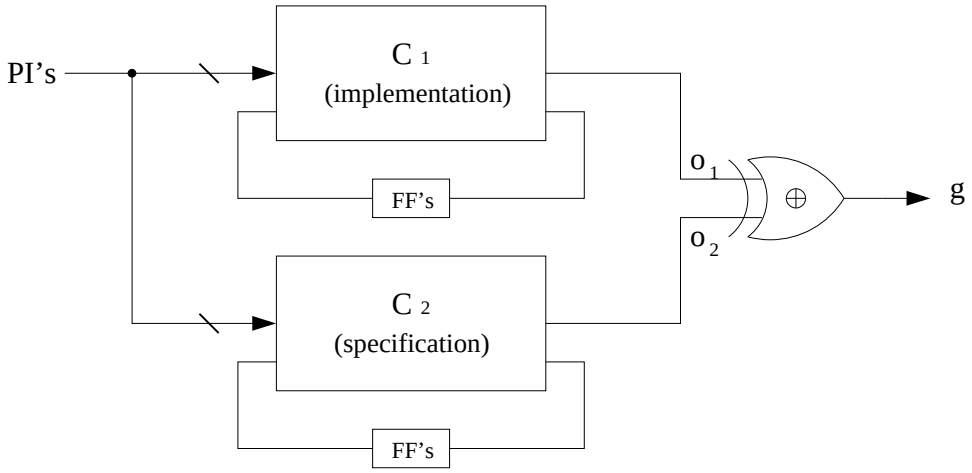

Figure 3.5: A sequential miter circuit

the equivalence property is checked on these k bounded time frames. The sequential equivalence checking model is shown in Figure 3.6, where each time frame consists of both the original and optimized circuits. For each time frame the circuit flip-flops are converted to pseudo primary inputs (PPIs) and pseudo primary outputs (PPOs). At time frame 0 the initial state of 0 is applied at each PPI. For each time frame $i, i \in \{0, k-2\}$, the PPOs are connected to the corresponding PPIs of time frame $i+1$. The verification is on checking if the miter output can be satisfied to 1. Zchaff is again used as underlying SAT solver.

In our lightweight and effective global constraint mining framework, the mined clauses can involve gates crossing not only time-frames, but also crossing the two circuits being checked. In addition, these mined constraints are global invariants, so the replicated constraints are also added to successive time frames. For example, consider that the mining is applied to a 4-time-frame unrolled miter. If a learned constraint involves node a in time frame #1 and node b in time frame #3 imply node c in time frame #4: $a_1 \cdot b_3 \rightarrow c_4$. Then, this constraint can be replicated to more deeply unrolled miters as well. For instance, the replicated constraint $a_3 \cdot b_5 \rightarrow c_6$ is also valid and can be added to the formula. In section 3.4 we will show that add learned constraints in this manner could obtain a significant performance gain.

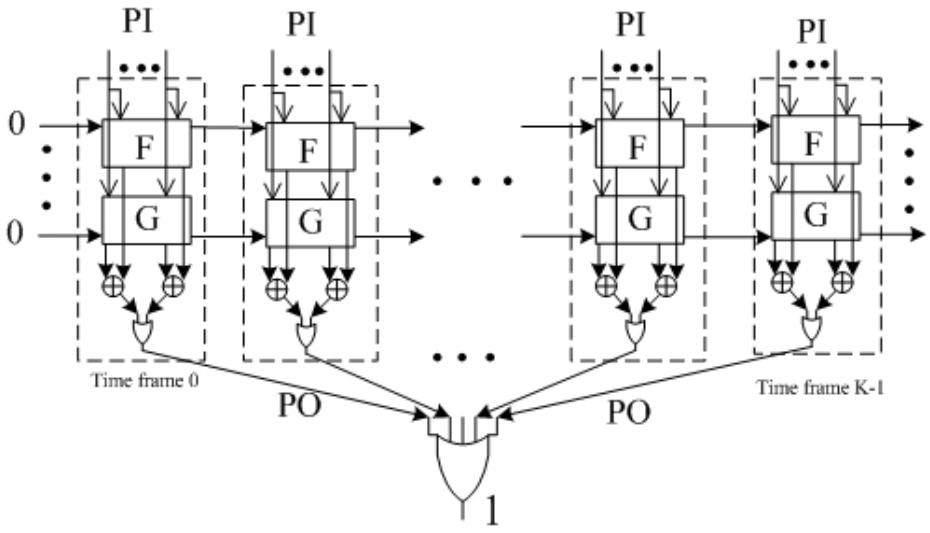

Figure 3.6: Bounded equivalence checking model

3.4 Experimental Results

The proposed technique was implemented in C++. All tests were performed on a Pentium IV, 2.8 GHz, with 1GB of RAM, running the Linux Redhat v9. Because equivalence checking on some ISCAS benchmarks was very easy for the SAT solvers, especially the ones with equivalent-node heuristic like BerkMin; For such easy benchmarks, adding constraints using our approach will not improve the results. Thus, we build the miter circuits ourselves in the experiments. First, the ISCAS'89 and ISCAS'93 Verilog circuits are deeply optimized by the Synopsys tools. Then the optimized version and the original circuit are combined to form the miter circuit. In doing this, we are expecting that the equivalent points of the optimized versions and the original circuits are reduced as much as possible. The miter circuits are checked using BerkMin561 with equivalence checking strategy turned on. We exclude those easy benchmarks in our experimental results and only report the hard instances. In our experiments, we implement 3 mining approaches: mining without any domain constraints, mining with the distance-based domain constraint, and mining with the region-based domain constraint.

Before we apply the learned 3-node constraints to bounded sequential equivalence checking,

we will first discuss the efficiency of our learning method. Table 3.2 reports the computational complexity of our method with a naive method. The naive method explores all possible 3-node combinations, whose complexity is $\frac{n(n-1)(n-2)}{6}$, where n is the number of nodes in the unrolled miter circuit. The column "3-node mining" shows the total number of 3-node relationships selected by our mining algorithm without domain constraint, where the probability threshold is 0.1. The last column reports the number of relationships using the naive 3-node combinations. From this table we can see that even for a very small circuit s298, the total number of 3-node combinations is already very large. It will easily cause both memory and temporal explosion, and completing such searches for global constraints will be infeasible. On the other hand, our mining algorithm is shown to be very efficient, with the computation complexity growing linearly with the circuit size.

Table 3.2: Comparison of computation complexity

Miter	# time frames	Total # nodes	Our 3-node mining	All 3-node combinations
s298	40	9948	65	1.64×10^{11}
s832	20	11670	42	2.64×10^{11}
s1196	20	20716	1437	5.06×10^{11}
s1488	20	24052	694	2.32×10^{12}
s3330	7	27963	300	3.64×10^{12}
s4863	7	27004	6115	3.28×10^{12}
s5378	7	42372	8607	1.27×10^{13}
s15850	7	147081	6280	5.30×10^{14}
s35932	5	183171	8217	1.02×10^{15}
s38584	5	211414	9817	1.57×10^{15}

Table 3.3 shows the mining results. The first column shows the miter circuits used. The second column reports how many time-frames the miter circuit was unrolled. While stronger constraints could be discovered in more deeply unrolled circuits, the mining time must also be considered to keep the learning cost low. We chose the number of time frames to keep the total number of nodes in the unrolled circuit between 10,000 to 500,000. Experimental results show that within this range the mining time is kept low, yet the global constraints learned are powerful enough. The third, fourth columns report mining method without domain constraint where the probability threshold is 0.1. The third column reports the number of true constraints identified for each benchmark.

The fourth column reports the total mining time in seconds. For instance, in miter-circuit s4863 with its optimized circuit, the learning was performed on a 7-time-frame unrolled miter, 4831 3-node true global constraints were learned. The total learning time was 147.8 seconds. Note that the number of true constraints depend on the threshold values used in the mining algorithm. For the comparison, the fifth, sixth columns report mining method without domain constraint where the probability threshold is 0.2. As the number of true constraints identified was increased, the mining time was also increased. In the experiments, we find that set the threshold to 0.1 could give us the best overall performance. The seventh to tenth columns report the mining method with distance based domain constraint. In the seventh and eighth columns, the distance limit is set to 4. In the ninth and tenth columns, the distance limit is set to 8. From the results we can see that the distance based mining method pruned some 3-node relationships learned by eliminating those locally close relationships, and the mining time are correspondingly reduced when compares to the method without any domain knowledge. When the distance limit was increased from 4 to 8, the number of true constraints learned were reduced as expected. However, the mining times were slightly increased in most instances. For example, in s5378, with the distance limit of 4, the number of constraints learned was 2948, with the mining time of 206.5 seconds; when the distance limit was increased to 8, the number of constraints is reduced to 2193, with the mining time of 218.7 seconds. The reason is that increasing the depth-first search limit on a large number of candidate nodes increases the overall overhead. Finally, the last two columns report the mining method with region-based domain constraint. From the results we can see that in all instances the region based method could further prune the 3-node relationships learned in mining process, the mining times were also reduced. From Table 3.3, we observed that applying domain constraints to mining approach could reduce the total mining times.

Table 3.4 shows the result of applying the learned constraints to bounded sequential equivalence checking. Because we are checking the equivalence of a sequential circuit with its optimized version, all results are UNSAT.[1] For each miter circuit, the number of time frames is first reported. Note that the number of time frames unrolled here can be different from the number of time frames

[1] Equivalence check of buggy optimized designs was very easy for the SAT solver, thus they are not included.

used to learn the constraints as reported in Table 3.3. The third column reports the original execution time of the SAT solver without any learned constraints. The next three columns report the results of mining approach without the domain constraint. In the forth column, the SAT solving time after adding global constraints is reported, followed by the total time (mining + equivalence checking) in fifth column. The speedup is reported in sixth column. All times are again reported in seconds. From the results we can see that in most cases the 3-node constraints could speed up the SAT solving time significantly. Noteworthy improvements were achieved for all miter circuits: in 6 of the 10 cases, more than one order of magnitude speedup was obtained; in the s3330 miter, more than 2 orders of magnitude speed was achieved. The seventh to ninth columns report results of the distance-based mining method. Finally, the last three columns report results of the region based mining method. For each miter circuit, the maximum speedup is shown in bold. We can see that in all cases, the domain constraint based mining methods are almost always better than the mining method without domain constraints. Comparing of the distance based method with the region based method, the region based method gave better results. Across the 10 cases, the distance-based method achieved better results in 2 instances while the region-based method achieved better results in the other 8 instances.

We notice in some cases, the equivalence checking time of the mining method with domain constraint is longer than the mining method without domain constraint. For instance, in miter circuit s3330, the equivalence checking time of the distance based method was 53.83 seconds, while the equivalence checking time of the basic method without domain constraint was only 28.62 seconds. In s15850, the equivalence checking time of the basic method without domain constraint was 10.28 seconds, while the equivalence checking time of the region-based method was 43.61 seconds. The reason is likely due to the fact that the domain constraint based methods prune some 3-node relationships which are useful for the equivalence checking. However, the total mining time of the domain constraint based methods was largely reduced, thus the overall performance was still improved.

Table 3.5 compares our technique to Berkmin, with the equivalence checking (EC) strategy turned on as well as turned of. The third column reports the original BerkMin (without EC on) run

time. The fourth column reports the BerkMin run time with EC strategy turned on. Finally, the last column reports our results, run with the basic Berkmin solver. Note that with the equivalence checking strategy, it is as if two-node constraints are added. From the results, we can see that the 3-node invariants are more powerful than equivalent checking strategy. For example, in the s15850-miter, the basic Berkmin solver took 19021.1 seconds; with EC turned on, Berkmin required 2575.0 seconds; finally, our approach only took 245.27 seconds. Results for other sequential miter circuits followed a similar trend. We note that for the miter circuits we used, the optimized versions and the original circuits did not have many structural equivalent points, thus the equivalent-node heuristic would not be powerful enough for these instances.

In Table 3.6, the learned constraints from the three mining approaches are analyzed. The CF columns report the percentage of true 3-node constraints that cross time frames. If not all 3 nodes are within the same time frame, the constraint is considered as placed in this category. The CC columns report the percentage of constraints that cross circuits. This means that the 3 nodes are located in both the original circuit and its optimized version. The PEP columns report the percentage of the pair of nodes that are potential equivalent points. When we calculated the PEP, we simply check the the logic values for the pair of nodes in the mining database to see if they are consistent. For example, if the pair of nodes is (a=1 and b=1), we check if there exist instance that (a=0 and b=1) or (a=1 and b=0). If no such instance exist, we consider the pair of nodes as PEP. From this table, we can see that the majority of the mined 3-node constraints cross time frames. This demonstrates that the constraints are not locally close. A good percentage of the constraints cross the original circuit and the optimized version. Finally, the percentage of PEP shows that the number of equivalent points in the 3-node relationship are not very high. With added domain knowledge, the percentage of CF clauses increased, as expected, since the emphasis was to weed out those locally close relationships.

3.5 Summary

In this chapter, we presented a novel mining technique to quickly identify global constraints in sequential circuits. The proposed mining method combined with domain knowledge allows us to efficiently identify the useful global constraints in the huge number of possible relationships in the circuit. Experiments show that the proposed mining strategy is very efficient in identifying important multi-nodes relationships that can significantly enhance the SAT solver in a bounded sequential equivalence checking framework. One to two orders of magnitude speedup was achieved for many cases. Future work includes improvement of mining technique to further reduce the overhead, mining of relationships that involve more than 3 nodes, and application to general model checking instances.

Table 3.3: Mining Results

		Without DK (0.1)		Without DK (0.2)		Distance based (d=4)		Distance based (d=8)		Region based	
Miter	# frames	# Clauses	Time(s)	# Clauses	Time(s)	# Clauses	Time(s)	# Clauses	Time(s)	# Clauses	Time(s)
s298	40	56	15.4	224	57.3	24	2.3	20	2.3	16	1.8
s832	20	40	31.5	209	108.7	28	19.7	22	20.3	18	5.2
s1196	20	1020	85.2	2792	175.4	704	59.6	612	63.2	423	42.8
s1488	20	229	122.0	578	226.2	176	87.9	161	92.4	115	66.5
s3330	7	225	103.6	1716	392.0	186	96.3	165	102.8	123	74.6
s4863	7	4831	147.8	11639	578.0	3237	113.2	3067	125.8	2305	91.4
s5378	7	3443	247.5	7458	712.3	2948	206.5	2193	218.7	1641	157.2
s15850	7	5087	562.2	16470	1013.0	3276	315.4	3035	342.8	2076	225.5
s35932	5	6245	1097.6	18374	2861.0	5354	897.1	4693	914.6	3743	587.6
s38584	5	8050	2567.0	24063	5432.0	6718	2036.4	6206	2247.8	3115	1641.6

Table 3.4: Bounded sequential equivalence Checking

			Without domain knowledge			Distance based			Region based		
Miter	# frames	original (s)	new (s)	total (s)	Speedup	new (s)	total (s)	Speedup	new (s)	total (s)	Speedup
s298	40	48.24	0.10	15.50	3.11	0.53	2.85	**16.93**	1.65	3.43	14.06
s832	30	1594.55	0.23	31.70	50.30	0.48	20.19	78.98	0.42	5.58	**285.76**
s1196	30	221.31	0.32	85.53	2.59	2.69	62.25	3.56	4.58	47.36	**4.67**
s1488	30	4187.11	1.35	123.39	33.93	5.10	93.00	45.02	8.77	75.29	**55.61**
s3330	15	>86400	28.62	132.22	>653.46	53.83	150.16	>575.38	21.36	95.96	>**900.37**
s4863	15	6594.76	289.68	437.52	15.07	172.26	285.48	23.10	89.30	180.66	**36.50**
s5378	15	8248.53	129.90	377.43	21.85	55.47	261.94	**31.49**	169.13	326.28	25.28
s15850	15	23057.20	10.28	572.45	40.28	39.72	355.10	64.93	43.61	269.09	**85.69**
s35932	10	4984.37	3.28	1097.62	4.54	3.51	900.63	5.53	1.77	589.38	**8.46**
s38584	10	8832.45	4.63	2571.58	3.43	7.13	2043.53	4.32	2.58	1644.16	**5.37**

Table 3.5: equivalent-node vs 3-node

Miter	# frames	BerkMin	BerkMin(EC)	Region based
s298	40	37.75	45.32	2.23
s832	30	1261.40	384.90	5.36
s1196	30	196.52	129.04	44.60
s1488	30	1760.29	1440.63	68.86
s3330	15	>86400	49053.10	84.16
s4863	15	3672.91	461.75	116.14
s5378	15	5386.46	2384.85	211.73
s15850	15	19021.10	2575.00	245.27
s35932	10	5602.94	1597.34	588.42
s38584	10	5156.34	3354.82	1642.78

EC: equivalence checking strategy

Table 3.6: Analysis of constraints

		Without domain knowledge mining			Distance based mining			Region based mining		
Miter	# frames	# clauses	CF(%)	CC(%)	PEP(%)	# clauses	CF(%)	CC(%)	PEP(%)	# clauses	CF(%)	CC(%)	PEP(%)
s298	40	56	75	62	10	24	100	64	8	16	100	58	6
s832	20	40	80	46	8	28	100	52	10	18	100	50	5
s1196	20	1020	72	52	9	704	94	58	12	423	100	63	12
s1488	20	229	81	48	6	176	100	56	11	115	100	51	8
s3330	7	225	85	46	10	186	96	54	7	123	100	45	5
s4863	7	4831	75	60	9	3237	92	67	14	2305	100	54	10
s5378	7	3443	85	65	11	2948	96	71	12	1641	100	62	17
s15850	7	5087	70	48	7	3276	85	56	5	2076	100	50	8
s35932	5	6245	85	74	16	5354	94	79	18	3743	100	66	14
s38584	5	8050	83	51	9	6718	90	63	8	3115	100	53	7

CF: Crossing Time Frame CC: Crossing Circuit PEP: Potential Equivalent Points

Chapter 4

Mining Sequential Constraints for Pseudo-Functional Testing

Using DFT methods such as scan can improve testability and increase fault coverage. However, scan tests may scan in illegal or unreachable states during test application, which may result in incidental detection of functional untestable delay faults during the scan test. Recent research has shown that these over-testing problems may cause yield loss. This chapter presents novel mining techniques for fast top-down functional constraint extraction. The extracted functional constraints capture illegal states through internal signal relations. Imposing these relations as functional constraints to a commercial ATPG tool allows for the generation of effective pseudo-functional tests. We analyze its impact on minimizing the over-testing problem of the scan-based circuits. The experimental results on transition faults and path delay faults reveal that the proposed method produces a small fraction, yet extremely powerful functional constraints effective for constraining the state space.

4.1 Introduction

DFT techniques such as scan are widely used to increase the testability of a design. However, some faults that are untestable under the functional operation mode may become testable under the scan mode. Furthermore, scanning vectors that have non-functional operations can cause higher

power/current consumption and results in over-testing. Recent research results have shown that such non-functional scan tests may incur yield loss, especially due to the exercising of functionally untestable path delay faults [44, 45].

Pseudo-functional testing was proposed to address the over-testing problem in recent years [29, 54]. In pseudo-functional testing, the test vectors are generated (or sometimes modified) such that they are as close to functional vectors as possible. These vectors are to achieve as high a fault coverage as possible without incidentally detecting the functionally untestable faults. Thus the scanned states should resemble the functionally reachable states as much as possible. This not only mitigates the detection of functionally untestable faults, but also restricts the peak power/current consumption during testing close to normal operation. Consequently, the over-testing problem is minimized, and the yield loss can be reduced. From a different angle, pseudo-functional testing restricts scan tests by avoiding illegal/unreachable states. This requires the computation of illegal states, which can be a computationally expensive process. Identification of illegal states has been studied in sequential ATPG via the identification of partial assignments to state variables that cannot be justified [20, 28]. In [28], three-value simulation is used to traverse the state space from an unknown initial state, all the valid states are recorded. In [20], illegal states are specified by the values of state variables, and it tries to merge illegal state space to larger cubes.

In pseudo-functional testing, a common method is to represent functional states or illegal states as functional constraints. By imposing such functional constraints, although the states generated do not necessarily guarantee to be within the functionally reachable states, the search space is reduced so that it more closely resembles the functional state space. Several methods to extract functional constraints have been proposed earlier. For instance, high-level design descriptions have been used to extract constraints in [43, 50]. At the gate level, identifying a subset of invalid states with low cost techniques have been investigated in [30, 31, 40, 49, 54]. In [30, 31], the constraint generation is fault specific. In [49, 54], implication-based learning for functional extraction was used. In [54], the illegal states are determined using implication learned during static learning in a preprocessing step. The illegal state cubes are saved in a list. The ATPG procedure derives test cube which does not contain any identified illegal state. In the work by Syal et al. [49], pair-wise

and multi-node functional 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. They demonstrate that such sequential relationships are very useful for identifying functionally untestable faults. In [40], a methodology to prevent overtesting in scan-based delay test was proposed in a test compression flow. They found that most functional constraints result in a decrease of the overall test data. Unlike all the previous techniques which use state variables to specify illegal states, the illegal states are captured with arbitrary nodes in the circuit [49]. The reason is that explicitly using state variable to specify illegal states may result in a very large number of constraints, while using arbitrary nodes could significantly reduce the storage overhead.

In our work, rather than starting from logic implications as the underlying platform to learn the functional constraints (which can be viewed as a bottom-up approach), we apply data mining techniques to extract the embedded functional constraints. Since we bypass the step of explicitly computing the logic implications, our approach can be viewed as a top-down strategy for learning these functional constraints. Stated differently, in [49], each sequential implication is computed by static learning techniques, and the functional constraints are thus computed. However, one problem of this bottom-up constraint extraction technique is the lack of good guiding heuristic on the sequential implication selection. Thus, a large number of constraints may result and the overall identified functional constraints might not be able to effectively prune the illegal state space. Our approach, on the other hand, employs intelligent search strategies to identify the important regions first. Then, it finds all the potential important implication candidates from these regions, then the actual functional constraints are proven from these candidates. We propose a novel methodology to identify these useful functional constraints, which can be easily imposed on any ATPG tools to generate pseudo-functional patterns. We target on sequential implications since the combinational constraints will be automatically derived during the ATPG process. We extract functional

constraints on arbitrary nodes instead of only state variables in the circuit.

Our approach is based on the observation that not all state variables are equally important on functional constraint extraction. For example, in [28], the state variables are grouped and their relationships are represented as dependence graph. Some state variable sets are more important than others on identifying invalid states. The dependency graph has also been used in [30] to search groups of state variables which are strongly correlated. In our approach, novel data mining techniques are employed to efficiently partition the state variables and capture non-trivial sequential relations over the circuit.

In chapter 3, we demonstrate that a mining strategy can capture important global invariants among several nodes in a sequential circuit for the purpose of bounded equivalence checking. In this chapter, several new mining techniques are employed to extract functional constraints for pseudo-functional testing. A mining algorithm Min-hashing and Locality-Sensitive Hashing (LSH)(refer to 2.3.3) is first applied to the state variable selection phase . After forming the state variable sets, the fanin cones of each set is computed. Then, we apply a second mining algorithm Apriori to identify any potentially important sequential implications in these fanin cones. These potential sequential implications are verified by the the assume-then-verify techniques [16] and form the functional constraints. In all, powerful functional constraints can be obtained in only a few minutes, even for large sequential circuits. These functional constraints are passed to a commercial ATPG tool to generate pseudo-functional transition and path delay tests. The experiments on transition faults and path delay faults reveal that the proposed method produces a small fraction, yet extremely powerful functional constraints effective for constraining the state space.

The rest of the chapter is organized as follows. Section 4.2 details the proposed approach. Section 4.3 analyzes the experimental results. Section 4.4 concludes the chapter.

4.2 The Proposed Approach

4.2.1 Overall framework

The basic flow of our approach is shown in Figure 4.1. First, a sequential circuit is unrolled to a k time-frames combinational circuit, with the state variables (pseudo primary inputs (PPIs)) unconstrained in the first time-frame. Next, n randomly generated input vectors are applied to the unrolled circuit. In our experiment, a maximum number of $20,000$ vectors are generated for each benchmark circuit. Each input vector assigns logic values to all the primary inputs (PIs) and PPIs. In our work, each node can take three values: logic 0, logic 1 or don't-care x. After logic simulation, the logic values for all nodes in time-frame 1 and time-frame k are recorded in the mining databases DB_1 and DB_k accordingly, where the columns list the sequential circuit nodes in the time-frame, the rows list the vectors, and the entries in the database denote gate values. An example partial database DB_k consisting of 3 PIs and 3 state variables is shown in Table 4.1.

Table 4.1: Mining database

Vector	PI1	PI2	PI3	FF1	FF2	FF3
v1	0	1	x	x	1	1
v2	1	x	0	1	1	0
v3	x	0	0	0	x	1
v4	1	1	1	0	1	x

From database DB_k (corresponding to the right-most time-frame), Min-Hashing and LSH algorithms are applied only to the state variables to perform state variable selection. For each state variable set, the nodes within these fanin cone of these state variables are candidate nodes for sequential relation mining, as these candidate nodes have common paths to a similar set of state variables in time-frame k. Apriori is applied to database DB_k on candidate nodes to mine pair-wise and 3-node relations. After consistency check, the potential relations are pruned using database DB_1 to remove any combinational relations. The rest of the potential *sequential* relations are verified by assume-then-verify validity check to determine if each constraint is indeed a true constraint. Then the constraints are passed to ATPG tool for pseudo-functional test generation. The details of

each step are described in the following subsections.

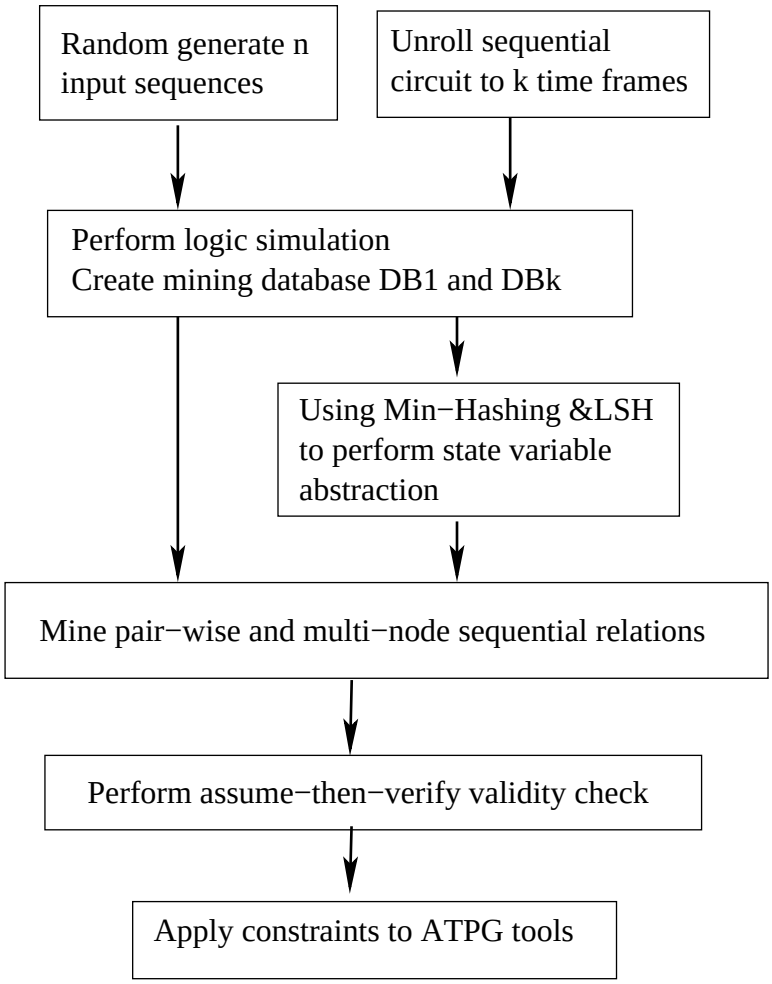

Figure 4.1: Overall framework

4.2.2 Selection of State Variables

The state variable selection is performed on the state variables from database DB_k. Min-Hashing and LSH are applied at this step. Unlike the general data mining approach, our mining algorithm needs to consider both logic 0 and logic 1 for each variable, as two variables can be positively or negatively correlated. So we first transform the mining database DB_k by splitting each state variable into positive and negative variables. For example, for mining database in Table 4.1, the

transformed database is shown in Table 4.2, where each FF_i is split into two columns. Since we only consider state variables, all the other nodes in DB_k can be excluded at this time. Then, the standard Min-hashing and LSH algorithm is applied to compute the frequent k-itemsets, where the state variables in a frequent k-itemset forms a partition state variable set.

Table 4.2: Transformed database

FF1	FF1'	FF2	FF2'	FF3	FF3'
x	x	1	x	1	x
1	x	1	x	x	1
x	1	x	x	1	x
x	1	1	x	x	x

The state variables in each set computed this way will be strongly correlated. Thus, the fanin nodes within the state variable set will likely have relations that can be discovered. By setting proper LSH algorithm parameters, the size of state variable set is generally not greater than 8. Limiting the size of state variables in each set could greatly reduced the computation effort needed in the next phase of sequential relation mining. This is based on the fact that generally in a test vector for a target delay fault, only a small subset of state variables are specified. Thus mining state variable sets with a small number of highly correlated flip-flops would be enough to identify the illegal space.

4.2.3 Mining sequential relations

We mine sequential relations in a way similar to the work in [51]; we briefly outline it here. For selected nodes from the state variable selection phase, we first calculate the signal probabilities for each node in the database DB_k. For each node g_i, the probability of g_i having logic 0 and logic 1, i.e., P_i^0 and P_i^1, are computed as: $P_i^0 = \sum(g_i = 0)/n$, $P_i^1 = \sum(g_i = 1)/n$, where n is total number of vectors in DB_k. After P_i^0 and P_i^1 have been computed, we mine frequent 1-itemset with $threshold = 0.3$. Next, we proceed to mine the potential candidate pairs from DB_k. For each candidate pair (a,b), we first check if there exists a correlation between them in DB_k. For example, we check the logic value of b for all rows that have $a = 1$ in DB_k. If in all cases where $a = 1$, $b = 1$ is always true, then we consider the node pair (a,b) as a *potential* pair-wise implications and add it

to the potential pair-wise implication list. Then next iteration of mining is carried to find potential 3-node relations.

Since we focus on the sequential relations within the same time-frame. The combinational relations need to be removed. Here the database DB_1 comes in handy. The selection is performed this way: for all potential 2-node and 3-node relations, we perform a consistency check in DB_1. But this time, only the relations that *fail* the consistency check will be considered as potential sequential relations, the rest will be removed from the list. For example, for a potential implication $(a \rightarrow b)$ obtained from DB_k, we check the logic value of b for all rows that have $a = 1$ in DB_1. If for all instances where $a = 1$, $b = 1$ is always true, then the relation $(a \rightarrow b)$ would very likely to be a combinational relation. We will delete all such relations. If there exists an instance where $a = 1$, $b \neq 1$, that means the implication $a \rightarrow b$ is definitely not true in the first time-frame. If it is proven that $(a \rightarrow b)$ is true in the k-th time-frame in the final validity check, then the relation $(a \rightarrow b)$ is a k-th time-frame invariant. The concept of k-th invariants is introduced in [32]. By choosing database at time-frame 1 and k, our methods could mine potential 2 to k-th time-frame sequential relations. These potential sequential relations will go to next phase for validity check.

4.2.4 Validity check of all mined relations

The validity check verifies each mined potential relation to determine if the relation is indeed globally correct. A method similar to the assume-then-verify [16] is used here. The difference of our method from [16] is that we do not simplify the circuit structure in the assumption step. We perform assume-then-verify in a window of $2 \times k$ time-frames. The first k time-frames make up the assume window, and the second k time-frames make up the verify window. We divide potential relations to 20 per group, and a group of relations is verified at a time. With all the initial PPIs unconstrained, we assume all 20 constraints are initially true in the assume window, and we iteratively verify each relation in the verify window.

For example, consider the potential constraints in a group to be $c_1 \ldots c_{20}$. We insert 20 clauses corresponding to these constraints to the assume window. If we are verifying c_1 at this time, we add

($\overline{c_1}$) to the verify window. If the solver returns SAT for this instance, no conclusion can be made for c_1, and this relation needs to be removed from both the assume and verify windows. If the solver returns UNSAT for this instance, then we remove ($\overline{c_1}$) from the verify window and continue to verify the next potential constraint c_2. If all constraints are verified UNSAT, we can conclude that this group of constraints is indeed globally true. On the other hand, if the solver returns SAT for any constraint c_i, after removing c_i from the assume window, all the previously verified constraints in this group need to be re-verified. To keep the overhead low, we only allow for 2 iterations for each group. In our experiments, we found that for most instances, two iterations were sufficient to identify most real constraints. All proven constraints are added to both assume and verify windows to expedite the verification of subsequent groups. We verify the easier potential relations first, and use the verified relations to help verifying the harder potential constraints later. In the experiments, we verify the relations in the following order: pair-wise relations, three-node relations. When we group the constraints, we place the constraints that from same cone in the same group.

We use zChaff [36](2004.11.15 Version) as underlying SAT solver because it has an excellent incremental SAT solving ability.

4.3 Experimental Results

We implemented the proposed method to extract functional constraints on a Pentium-4, 3.2GHz processor with 1GB RAM Linux machine. All constraints extracted are used with a commercial ATPG tool. Experimental results for generating pseudo-functional tests for transition faults and path-delay faults are presented on the large ISCAS-89 and ITC-99 sequential benchmark circuits.

Table 4.3 shows functional constraint extraction results. For each circuit, the number of flip-flops is first reported, followed by the number of state variable sets mined using Min-Hashing and LSH algorithm. The number of state variables in a set generally are from 3 to 8, and not every state variable may be included in any state variable set. The final two columns report the number of functional constraints obtained and the runtime in seconds to extract them. For example, in

s38417 which has 1452 flip-flops, 63 sets of flip-flops were mined, and a total of 6953 functional constraints were mined and proven in only 95.47 seconds.

Table 4.3: Constraints found by our approach

circuit	# FFs	# Sets	# Constr	time(s)
s5378	179	15	395	1.41
s9234	228	11	643	4.28
s13207	669	32	1208	8.9
s15850	597	28	1148	42.1
s38417	1452	63	6953	95.47
s38584	1426	51	2152	29.6
b14	245	7	406	1.3
b15	449	12	1403	76.38
b17	1415	35	6532	21.76

To demonstrate that the extracted constraints are useful for mitigating the over-testing problem, we run experiments for both transition faults and path delay faults with a commercial ATPG. By imposing all the extracted constraints, the ATPG is prohibited from generating any scan pattern that violates the constraints. As a result, some originally detectable faults without any functional constraints may become untestable by the constrained ATPG. The results are reported in Table 4.4. Column 2 gives the total number of transition faults. Column 3 and 4 compares the transition fault coverage of ATPG without constraints and with constraints. We note that no fault was aborted by the ATPG. It can be observed that in all circuits, the constrained ATPG avoids detection of many functionally untestable transitions. For example, in s5378, 14.8% of the originally detectable faults were actually functionally untestable. We performed a similar experiment on path-delay faults. Column 5 to 7 reports the results of path delay faults. Since the number of paths for large circuits are huge, we target a maximum of 5000 most critical paths (longest paths assuming unit delay). For each path, both rising and failing transitions on the input are targeted (making a total of 10,000 path delay faults). Column 6 shows the fault coverage obtained by ATPG without constraints imposed, and column 7 reports the coverage by the ATPG with functional constraints. We note that there is a larger percentage drop comparing to the transition fault model, indicating that a significant portion of the longest paths are actually functionally untestable, and they should be avoided during

scan-test to reduce over-testing.

Table 4.4: Results of Test Coverage

	Transition fault			Path delay fault		
circuit	#	w/o%	with%	#	w/o%	with%
s5378	6874	93.7	78.9	10k	78.45	19.7
s9234	11328	90.2	84.1	10k	2.9	0
s13207	15546	89.9	81.4	10k	11.3	5.8
s15850	19040	89.4	71.5	10k	2.1	0
s38417	49582	96.8	94.6	10k	23.6	1.8
s38584	60950	93.8	90.3	10k	28.4	17.2
b14	8888	96.7	88.04	10k	49.5	14.4
b15	16676	98.33	79.5	10k	65.1	32.9
b17	45290	96.85	89.46	10k	15.7	0

Note: No fault was aborted

Finally, Table 4.5 compares our mining based approach with static learning based approach in [49]. Columns 2 and 3 report the number of constraints obtained for each method. Columns 4 and 5 compare the percentage of faults declared as functionally untestable on transition faults. The right-most 2 columns compare the percentage of faults declared as functionally untestable on path-delay faults. It can be seen that in all cases our mining based approach generates less than 1% of the functional constraints as compared with [49], yet these constraints are powerful enough to identify more functional untestable transition faults! This demonstrates that our top-down learning strategy could identify extremely intelligent and powerful functional constraints that able to prune the illegal state space. We note that the constraint extraction time is similar between the two approaches. However, the ATPG performance may be significantly degraded if it needs to consider a large number of constraints.

4.4 Summary

We have presented a suite of novel mining techniques to quickly identify a small yet powerful set of functional constraints from a sea of relations in a sequential circuit. With less than 1% of functional constraints compared with a previous approach, our method is able to avoid much of

Table 4.5: Comparison of constraint extraction

	# of Constraints		% Func Unt TF		% Func Unt PDF	
circuit	Ours	[49]	Ours	[49]	Ours	[49]
s5378	395	135751	**14.8**	11.3	**58.75**	33.9
s9234	643	228736	**6.1**	3.5	**2.9**	1.4
s13207	1208	1500k	**8.5**	6.2	5.5	**12.2**
s15850	1148	1500k	**17.9**	2.5	2.1	**3.5**
s38417	6953	181106	**2.2**	1.1	**21.8**	1.4
s38584	2152	1500k	**3.5**	2.1	11.2	**46.2**

the illegal state space. The proposed mining methods allow us to efficiently extract a small yet powerful set of constraints from the huge number of possible relations in the circuit. Experimental results show that the constraints can be extracted in a matter of a few minutes even for large circuits. By adding the the small set of mined functional constraints, the ATPG avoids generating tests for functionally untestable delay faults and hence minimizing the over-testing problem.

Chapter 5

Summary of Thesis

This thesis is mainly focusing on how to mine constraints to solve testing and verification problems. In Chapter 1, we briefly discussed the development of functional verification and testing and why are they so important in EDA communities. Then the motivation of this thesis was discussed. Major contributions of this thesis were outlined.

To help readers to understand our proposed techniques, various background knowledge are introduced in Chapter 2. Boolean Satisfiability is described on section 2.1. Section 2.2 provided the necessary background in implication, such as direct implication, indirect implication and extended backward implication. Data mining techniques are introduced on section 2.3. In this thesis, we mainly focusing on association rule mining techniques such as Apriori algorithm 2.3.2, Min-Hashing and Locality-Sensitive Hashing algorithm 2.3.3. .

In Chapter 3, we presented a learning aided formal verification approach. In our approach, we first ran logic simulation to generate a database, then we applied a mining technique on this database to discover global invariants in a sequential circuit. In contrast to the traditional learning methods, our mining algorithm can efficiently discover important relations among several nodes. We also utilized domain knowledge to prune the search space during mining process. We propose two types of domain-based constraints in this chapter: the distance based and the region based. Incorporating domain constraint in our mining algorithm, the nodes involved may often span several time-frames, thus improving the deductibility of the problem instance. Finally, we applied

these invariants to help the formal verification process. In our case, it is bounded sequential equivalence checking. Experimental results demonstrated that the application of these global invariants to SAT-based bounded sequential equivalence checking can achieve one to two orders of magnitude speedup. In addition, because it is orthogonal to the underlying SAT solver, it can help to enhance the efficiency of typical SAT-based verification flows.

Research results have shown that the traditional structural testing for delay and crosstalk faults may result in over-testing [21, 44]. The reason is due to the non-trivial number of such faults that are testable in the scan-test mode while untestable in the functional mode. The reason why a structurally testable fault might become untestable in the functional mode is that various types of functional constraints exist due to the functional operations of the logic. The pseudo-functional test generation is a methodology of identifying useful functional constraints, which can be imposed on the ATPG process so that the generated patterns are more likely to be functional patterns [29]. The goal of the pseudo-functional test generation approach is to find effective functional constraints so that the space restricted by these constraints (called pseudo-functional space) is as close to the functional space as possible. In a pseudo-functional test method, the first pattern of a two-pattern test is still scanned in the test mode but the pattern is generated in such a way that it tries not to violate the functional constraints extracted from the functional logic.

In Chapter 4, we explored effective functional constraints extraction method. Our approach employs several mining techniques to extract a set of multi-node functional constraints which consists of illegal states and internal signal correlation. Then the functional constraints are imposed to a ATPG tool to generate pseudo functional delay tests.

The major contribution of this thesis is that we address functional verification and testing problems 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] R. Agrawal, T. Imielinski, and A. Swami. Database mining: A performance perspective. *IEEE Trans. on Knowledge and Data Engineering*, 5(6):914–925, December 1993.

[2] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In *Advances in knowledge discovery and data mining*, pages 307–328, 1996.

[3] M. E. Amyeen, W. K. Fuchs, I. Pomeranz, and V. Boppana. Implication and evaluation techniques for proving fault equivalence. In *Proceedings of VLSI Test Symposium*, pages 201–207, 1999.

[4] S. S. Anand, D. A. Bell, and J. G. Hughes. The role of domain knowledge in data mining. In *Proceedings of International Conference on Information and knowledge management*, pages 37–43, 1995.

[5] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using SAT procedures instead of bdds. In *Proceedings of Design Automation Conference*, pages 317–320, 1999.

[6] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. In *Proc. of ACM Symp. on Theory of Computing*, pages 327–336, 1998.

[7] S. Chakradhar, V. D. Agrawal, and S. G. Rothweiler. A transitive closure algorithm for test generation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(7):1015–1028, July 1993.

[8] E. Cohen and et al. Finding interesting associations without support pruning. In *Proc. of Int'l Conference on Data Engineering*, pages 489–500, 2000.

[9] H. M. D. J. Hand and P. Smyth. *Principles of Data Mining*. MIT Press, 2001.

[10] M. Davis and H.Putnam. A computing procedure for quantification theory. In *Journal of the ACM*, pages 201–215, 1960.

[11] M. Davis, G. Logeman, and D. Loveland. A machine program for theorem proving. In *Proceedings of the Communications of the ACM*, pages 394–397, 1962.

[12] N. Eén and N. Sörensson. An extensible sat-solver. In *SAT '03: Proceedings of SAT*, 2003.

[13] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of Np-Completeness*. W.H. Freeman, 1979.

[14] K. Gulrajani and M. S. Hsiao. Multi-node static logic implications for redundancy identification. In *Proceedings of Design, Automation and Test in Europe Conference*, pages 729–733, 2000.

[15] B. Hjorland and H. Albrechtsen. Toward a new horizon in information science: domain analysis. *Journal of the American Society for Information Science*, 46(6):400–425, July 1995.

[16] S. Y. Huang, K. C. Chen, K. T. Cheng, C. Y. Huang, and F. Brewer. Aquila: An equivalence verifier for large sequential designs. *IEEE Trans. on Computers*, 49(5):443–464, May 2000.

[17] H. Ichihara and K. Kinoshita. On acceleration of logic circuits optimization using implication relations. In *Proceedings of Asian Test Symposium*, pages 222–227, 1997.

[18] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proc. of ACM Symp. on Theory of Computing*, pages 604–613, 1998.

[19] S. Kajihara, K. K. Saluja, and S. M. Reddy. Enhanced 3-valued logic/fault simulation for full scan circuits using implicit logic values. In *Proceedings of European Test Symposium*, pages 108–113, 2004.

[20] M. H. Konijnenburg, J. T. V. D. Linden, and A. J. V. de Goor. Illegal state space identification for sequential test generation. In *IEEE Design and Test in Europe*, pages 741–746, 1999.

[21] A. Krstic, J. J. Liou, K. T. Cheng, and L. C. Wang. On structural vs. functional testing for delay faults. In *Proceedings of International Symposium on Quality Electronic Design*, pages 438–441, 2003.

[22] A. Kuehlmann. Dynamic transition relation simplification for bounded property checking. In *Proc. of Int'l Conference on Computer Aided Design*, pages 50–57, 2004.

[23] W. Kunz and D. K. Pradhan. Accelerated dynamic learning for test pattern generation in combinational circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(5):684–694, May 1993.

[24] 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*, 13(9):1149–1158, September 1994.

[25] W. Kunz, D. Pradhan, and S. M. Reddy. A Novel Framework for Logic Verification in a Synthesis Environment. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(1):20–32, Janurary 1996.

[26] W. Kunz, D. Stoffel, and P. R. Menon. Logic optimization and equivalence checking by implication analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(3):266–281, March 1997.

[27] T. Larrabee. Test pattern generation using boolean satisfiability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(1):4–15, Janurary 1992.

[28] H.-C. Liang, C. L. Lee, and J. E. Chen. Identifying invalid states for sequential circuit test generation. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 16 (9):1025–1033, Sept. 1997.

[29] Y. C. Lin, F. Lu, and K.-T. Cheng. Pseudo-functional scan-based bist for delay fault. In *Proceedings of VLSI Test Symposium*, pages 229–234, 2005.

[30] Y.-C. Lin, F. Lu, K. Yang, and K.-T. Cheng. Constraint extraction for pseudo-functional scan-based delay testing. In *IEEE Proc. of ASP-DAC*, pages 166–171, 2005.

[31] X. Liu and M. S. Hsiao. A novel transition fault atpg to reduce yield loss. *IEEE Design & Test of Computers*, 22(6):576–584, November-December. 2005.

[32] F. Lu and K.-T. Cheng. Sequential equivalence checking based on k-th invariants and circuit SAT solving. In *High-Level Design Validation and Test Workshop*, pages 45–51, 2005.

[33] M. lyer and M. Abramovici. FIRE: A Fault-independent Combinational Redundancy Identification Algorithm. *IEEE Transactions on VLSI Systems*, 4(2):295–301, June 1996.

[34] S. Malik. Boolean Satisfiability Research Group at Princeton. In *http://www.princeton.edu/ chaff/zchaff.html*.

[35] J. P. Marques-Silva and K. A. Sakallah. A Search Algorithm for Propositional Satisfiability. In *IEEE Transactions on Computers, Vol. 48, No. 5*, pages 506–521, 1999.

[36] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proc. of Design Automation Conference*, pages 530–535, 2001.

[37] Z. Nazeri and E. Bloedorn. Exploiting available domain knowledge to improve mining aviation safety and network security data. In *Proceedings of European Conference on Machine Learning*, 2004.

[38] Y. Novikov. Local search for Boolean relations on the basis of unit propagation. In *Proceedings of Design, Automation and Test in Europe Conference*, pages 810–815, 2003.

[39] D. A. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2(3):293–304, September 1986.

[40] I. Polian and H. Fujiwara. Functional constraints vs. test compression in scan-based delay testing. In *Proc. of Design, Automation and Test in Europe Conference*, pages 1039–1044, 2006.

[41] 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*, 7(2):156–173, April 2005.

[42] J. Rajski and H. Kox. A Method to Calculate Necessary Assignments in ATPG. In *Proceedings of the International Test Conference*, pages 25–34, 1990.

[43] R. S. Ram and D. E. Thomas. Behavioral test generation using mixed integer non-linear programming. In *IEEE Int'l Test Conference*, pages 958–967, 1994.

[44] J. Rearick. Too much delay fault coverage is a bad thing. In *Proceedings of the International Test Conference*, pages 624–633, 2001.

[45] J. Saxena, K. M. Butler, V. B. Jayaram, and et.al. A case study of IR-drop in structured at-speed testing. In *Proc. of Int'l Test Conference*, pages 1098–1104, 2003.

[46] M. H. Schulz and E. Auth. Improved deterministic test pattern generation with applications to redundancy identification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(7):811–816, July 1989.

[47] M. H. Schulz, E. Trischler, and T. M. Sarfert. SOCRATES: A highly efficient automatic test pattern generation system. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 7(1):126–137, Janurary 1988.

[48] M. Syal, R. Arora, and M. S. Hsiao. Extended forward implications and dual recurrence relations to identify sequentially untestable faults. In *Proceedings of IEEE International Conference on Computer Design*, pages 453–460, 2005.

[49] M. Syal, K. Chandrasekar, V. Vimjam, M. S. Hsiao, Y.-S. Chang, and S. Chakravarty. A study of implication based pseudo functional testing. In *IEEE Int'l Test Conference*, pages 1–10, 2006.

[50] V. M. Vedula and J. A. Abraham. Actor: A hierarchical methodology for functional analysis,. In *IEEE Design and Test in Europe*, pages 730–734, 2002.

[51] W. Wu and M. S. Hsiao. Mining global constraints for improving bounded sequential equivalence checking. In *Proc. Design Automation Conf.*, pages 743–748, 2006.

[52] L. Zhang and S. Malik. The quest for efficient boolean satisfiability solvers. In *Proceedings of International Conference on Computer Aided Verification*, pages 17–36, 2002.

[53] L. Zhang, C. F. Madigan, M. H. Moskewicz, and S. Malik. Efficient Conflict Driven Learning in a Boolean Satisfiability. In *Proceedings of International Conference on Computer Aided Design*, pages 279–285, 2001.

[54] Z. Zhang, S. M. Reddy, and I. Pomeranz. On generating pseudo-functional delay fault tests for scan designs. In *IEEE Int'l Symp. Defect and Fault Tolerance in VLSI Systems*, pages 398–405, 2005.

[55] J. K. Zhao, E. M. Rudnick, and J. H. Patel. Static Logic Implication with Application to Fast Redundancy Identification. In *Proceedings of VLSI Test Symposium*, pages 288–293, 1997.

[56] 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. In *Proceedings of the VLSI Design Conference*, pages 163–169, 2001.