Identification and Analysis of Illegal States in the Apoptotic Discrete Transition System Model using ATPG and SAT-based Techniques

Anupam Shrivastava

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

Master of Science
In
Bradley Department of Electrical and Computer Engineering

September 25, 2008
Blacksburg, Virginia

Keywords: Apoptosis, SAT, Bounded Model Checking, ATPG, Image Computation

Identification and Analysis of Illegal States in the Apoptotic Discrete Transition System Model using ATPG and SAT-based Techniques

Anupam Shrivastava

ABSTRACT

Programmed Cell Death, or Apoptosis, plays a critical role in human embryonic development and in adult tissue homeostasis. Recent research efforts in Bioinformatics and Computational Biology focus on gaining deep insight into the Apoptosis process. This allows researchers to clearly study the relation between the dysregulation of apoptosis and the development of cancer. Research in this highly interdisciplinary field of bioinformatics has become much more quantitative, using tools from computational sciences to understand the behavior of Biological systems.

Previously, an abstracted model has been developed to study the Apoptosis process as a Finite State Discrete Transition Model. This model facilitates the reutilization of the digital design verification and testing techniques developed in the Electronic Design Automation domain. These verification and testing techniques for hardware have become robust over the past few decades. Usually simulation is the cornerstone of the Design Verification industry and bulk of states are covered by simulation. Formal verification techniques are then used to analyze the remaining corner case states. Techniques like Genetic Algorithm guided Logic Simulation (GALS) and SAT-based Induction have already been applied to the Apoptosis Discrete Transition Model. However, the Apoptosis model presents some unique problems. The simulation techniques have shown to be unable to *cover most of the states of the Apoptosis model. When SAT-based Induction is applied to the Apoptosis model, in particular to find illegal states, very few illegal states are identified. It particularly suffers from the fact that the Apoptosis Model is rather complex and the formulation for testing and verification is hard to tackle at larger bounds greater than 20 or so. Consequently, the state space of the Apoptosis model largely lies in the unknown region, meaning that we are unable to either reach those states or prove that they are illegal. Unless we know whether these states are reachable or illegal, it is not feasible to infer information about the model such as what protein concentrations can be reached under what kind of input stimuli. Questions such as whether certain protein concentrations can be reached or not in this model can only be answered if we have a clear picture of the reachability of state space.*

In this thesis, we propose techniques based on ATPG and SAT based image computation of the Apoptosis finite transition model. Our method leverages the results obtained in previous research work. It uses the reachable states obtained from the simulation traces of the previous work as initial states for our technique. This enables us to identify more illegal states in less number of iterations; in other words, we are able to reach the fixed point in image computation faster. Our experimental analysis illustrates that the proposed techniques could prove most of the former unknown states as illegal states. We are able to extend our analysis to obtain clearer picture of the interaction of any two proteins in the system considered together.

To my family

Acknowledgements

I want to express my deepest gratitude to Professor Michael Hsiao for giving me the opportunity to work with him. His guidance and encouragement at every step of my graduate student life has been instrumental in the accomplishment of this work. Without his advices and persistent help, this work would not have been possible. I also want to thanks Professor Chao Huang and Professor Leyla Nazhandali for serving in my MS committee.

I want to thank my fellow graduate students and friends Mainak for providing some useful preliminary data which was used in this work and Ankur for discussing some important aspects of good coding practices. I want to thank all the Proactive members for being there with words of encouragement and suggestions especially Karthik, Shrirang and Mahesh. I am also thankful to all my room-mates and friends for there support and help during my stay in Blacksburg.

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

Anupam Shrivastava

Virginia Polytechnic Institute and State University

September 2008

CONTENTS

List of Figures

List of Tables

Chapter 1

Introduction

Recent years have seen significant advances in interdisciplinary research, such as those involving Computational Biology/Bioinformatics and Computer Science & Engineering. Algorithms along with their analytical strengths developed in Computer Science & Engineering broaden the scope of investigation for biomedical problems. The two seemingly unrelated and disparate disciplines might yield fresh and possibly unexpected insights when they are collaborated. Lately, software modeling of cell biology and cellular geometry, biochemical reaction pathways, prediction of cell behavior by simulation and interpreting the experimental data of subsystems has been broadly used in order to study and gain more knowledge about cell biology.

One such phenomenon that has exploded as an area of research is Apoptosis. Usually, when a cell begins to multiply in a dangerously abnormal way, a series of biochemical signals trigger it to die. Called Apoptosis, this process plays a vital role in keeping the body healthy. However, this process sometimes fails and leads to diseases including cancer. A complete insight into the process would help scientists to develop drugs to cure cancer by effectively triggering apoptosis in cancer cells. Apoptosis does more than ward off disease. Millions of cells in our bodies kill themselves everyday to make room for healthier cells in order to keep the body functioning properly. As well as Apoptosis's implications in many diseases, it is an integral part of biological development. We refer the readers to [1] [2] [3] for comprehensive details on Apoptosis.

1.1 Finite State Transition System modeling of Apoptosis

The Apoptosis process involves a complex system of pathways of different proteins and signals. By constructing the pathways as a finite state transition Model (FSM), it provides a unique level of abstraction. Such type of modeling helps us to study Apoptosis as a discrete hardware model and analytical algorithms and techniques can be applied to study this model in detail.

One such recent work is the modeling of the mitochondrially mediated Apoptosis signaling pathways as a discrete transition system model [1]. The choice of Finite State Transition model presents a unique method of analysis of the Apoptosis process. The framework can model both, healthy as well as slightly altered, or faulty, signaling pathways. A healthy model is useful in studying the reachability information of various protein concentrations. The corresponding input stimuli and the state trace can be easily determined if the target concentration state is found to be reachable. We draw a simple analogy by comparing the input stimuli to all the external factors that can affect the Apoptosis model. The state values denote the protein concentrations in the model. There might be protein concentrations which cannot be reached. Whether the protein concentrations are hard to reach or cannot be reached is interesting information that can be obtained from this model.

Faulty models can be constructed and analyzed using similar techniques. These faulty models provide information such as answers to the questions like what kinds of changes occur in a healthy model when an external pathogen alters the capacity of signaling pathways. Whether unreachable protein concentrations in the healthy model have now become reachable? Or whether reachable protein concentrations in the healthy model have now become unreachable concentrations? Such type of information is essential to study how external factors affect the Apoptosis process by altering its normal functionality.

The conversion of the protein signaling pathways into an FSM is described in detail in [1]. In the rest of this chapter, we mention some of the salient features of [1]. We also discuss our contributions in brief.

1.1.1 Salient features of the Apoptosis Finite State Machine

The individual elements and the reactions of the mitochondria mediated signaling pathway have been taken from the REACTOME database [4]. As an example, one such protein “BAD” forms following state network, as depicted in Figure 1-1. There are many other such proteins. And for each protein state network, there are interactions with other protein state networks. This whole system forms a complex model and makes it interesting to study it as a Finite State Transition system.

Without going into more details regarding the Protein pathway Network, which has been discussed in detail in [1], what we have as an outcome of [1] is a hardware model which represents certain section of the signaling pathways. Different Protein pathways have been implemented as interacting FSM modules representing the corresponding finite state transition diagram. The protein diagrams are then combined as one interconnected FSM module to represent the whole Apoptosis system. The inputs to this FSM consist of all external substrates that assist the reaction. These include: Granzyme B, Caspase 8, NMT1, BcL_XL, etc. Mitochondria and Cytoplasm, acting as assembly sites for many of the pro- and anti-apoptotic factors have also been take into consideration in the build up of this full system FSM. In the rest of this thesis, we will refer to the full system FSM as Apoptosis FSM.

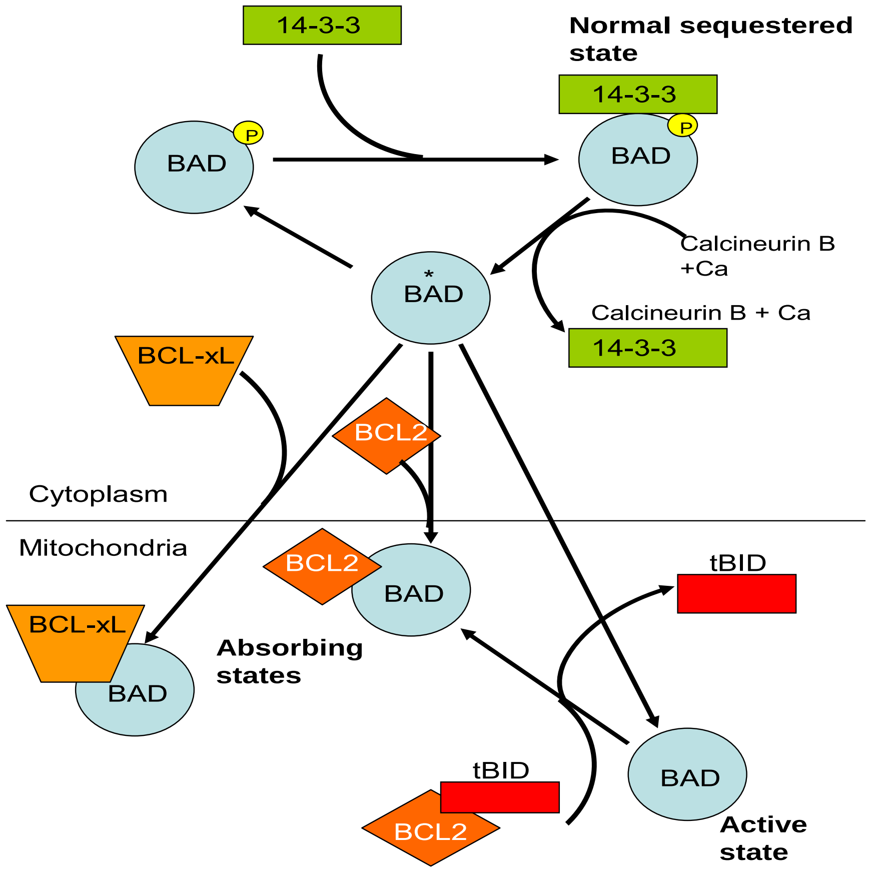


Figure 1-1: Sample BAD Protein state network. It's interactions with other protein state networks (for example BCL2, tBID, etc) forms a complex overall system.

The state of the Apoptosis FSM is defined as the combination of all protein concentrations in this network. The Apoptosis FSM has 8 protein concentrations in it – BCL2 (1), BAD_MITO (2), BAD_p14 (3), BAD_cyto (4), BAD_BCL2 (5), tBID_MITO (6), tBID_BCL2 (7), tBID_cyto (8). The numbers in brackets serves as an identification number for the proteins in the full Apoptosis FSM. Henceforth, for example, we can also refer to the protein tBID_MITO as protein 6 for simplicity.

The concentration of a protein is basically the value stored in registers used to denote that particular protein. Registers, simply, are D flip-flops in the hardware circuit. The number of bits used to represent the concentration determines its resolution and range. More bits allow for higher resolution, but at an increased cost during analysis. Eight bits have been used for each protein in the Apoptosis FSM. Hence, there are 256 (0 to 255) possible values that can represent the concentration of a protein. The unit of concentration is fixed to be 0.1 nano-molar (nM). The resolution is thus 0.1 nM and the range is 0 – 25.5 nM which is analogous to the state range from 0 - 255. A total of 8 proteins in the Apoptosis FSM mean that we have 64 flip-flops in the FSM. Hence, the circuit's state space potentially consists of 2^{64} states. We refer the reader to [1] for more information on Apoptosis FSM model for example details on the rate constants values and the initial concentrations of the proteins have been described in [1].

1.2 Previous Results of Apoptosis model analysis

After modeling the Apoptosis model as a finite state transition system, the next step is to analyze the FSM in order to gain a clear insight into the whole system. Genetic Algorithm based guided logic simulation (GALS) and Bounded Model Checking (BMC) have been used to find out a subset of reachable states in the Apoptosis Model. SAT-based Induction has also been used to identify a subset of illegal states in the FSM.

A phased approach is chosen in which initially a standalone, simplified model of a single protein is developed. Other proteins affect this standalone model in form of inputs which are left unconstrained, meaning that they can assume any value. All the data obtained after the analysis of the standalone models is combined together to build an FSM denoting the complete protein state network of the section under study. Table 1-1 presents a brief analysis of the results obtained on state reachability with the techniques mentioned above.

Table 1-1: Previous Results of Apoptosis FSM Analysis

Protein type	# Reachable States	# Illegal States	# Unknown states
BCL2	125	1	**130**
BAD_mito	11	0	**245**
BAD_p14	251	1	**4**
BAD_cyto	239	1	**16**
BAD_BCL2	233	0	**23**
tBID_mito	103	9	**144**
tBID_BCL2	114	0	**142**
tBID_cyto	12	224	**20**

From the results in Table 1-1, for certain proteins like BCL2, BAD_mito, tBID_mito and tBID_BCL2, a large portion of state space lies in the unknown category. This means that the techniques could not conclusively determine the reachability of these states. Here we explain the importance of classifying the unknown states as reachable or illegal. If we consider each protein state flip-flop group, there are in total 256 states for a protein because of 8 flip-flops used for specifying the concentration. Even if a single illegal state is identified among the 256 states of a single protein, we are able to say that other seven protein's total 2^{56} combinations will not occur with this particular state. So we effectively are able to assert a large portion of state space as illegal by working on individual protein state space. GALS has been able to reach many states in case of some of the proteins. However, in cases where very few states were reachable, the number of unknown states is large and number of illegal states proven is few. For example, BAD_mito has only 11 reachable states. It has 245 unknown states and none of it is proven as illegal.

1.3 Our contributions

As we wish to have a clearer picture of the protein state space in the Apoptosis model, the problem we are addressing and the specific contribution are on the classification of the unknown states in the Apoptosis model to either reachable or illegal states category. In chip verification, usually a high percentage, say 80%, of the states are covered by simulation methods and then formal verification methods are used to hit the remaining corner case states. By looking at the different protein states reached as shown in Table 1-1, it is true for proteins like BAD_p14, BAD_cyto and BAD_BCL2. Around 90% of the states are accounted for. But for other proteins such as BAD_mito, tBID_mito, etc, less than 50% of states have been accounted for. Other states are neither reachable nor proven to be illegal. In our work, we attempt to gain further insight into the Apoptosis model by finding out the nature of these unknown states.

The apoptosis model is complex and it is clear from the fact that SAT-based induction technique applied previously either aborts or is inconclusive [1]. We propose an image computation based methodology in which we make use of the previous results shown in Table 1-1. This allows us to reduce the unknown state space size. Our contributions can be summarized as follows:

1. We are able to categorize close to 93% of unknown states of individual proteins as illegal states with a new formulation to illegal state identification.
2. We extend our analysis to study the interaction of any two proteins in the Apoptosis system to investigate correlations between pairs of proteins.

3. We show by another experiment that simulation techniques do not yield desirable results on this model. Consequently, further analysis of these types of models should be broadly based on formal verification techniques.

The more information we are able to gain about this Apoptosis discrete transition model, the better it will aid us in the understanding of the Apoptosis process through the EDA domain.

1.4 Organization

This thesis is organized in the following sections. Chapter 2 covers the basics of testing and verification techniques. It also covers some details about the approaches used in the previous work of analyzing the Apoptosis model and techniques that form the basis of our developed methodology. Chapter 3 describes our approach of customizing the ATPG techniques to find illegal states in the Apoptosis Model. We also mention the results obtained with this approach. Chapter 4 describes our methodology which involves modified SAT based image computation technique that also utilizes the data obtained from previous research work. This particular technique provides good results and we are able to reduce the unknown state space in the Apoptosis model considerably. With the improved knowledge of the state space of Apoptosis model, we further enhance our learning of the Apoptosis model by studying the relation of various protein combinations. In Chapter 5, we conclude by providing a recap of the work done in this thesis and also future research directions are mentioned.

Chapter 2

Background

Over the past few decades, considerable research efforts have been seen in the Electronic Design Automation (EDA) industry. This can be partly attributed to the Moore’s law which states that the relevant parameters of digital chips double every 18 months. Consequently, there is the design productivity gap which means that the technology matures faster than the design tools. And then there is the verification gap which denotes the fact that test and verification takes an ever-increasing portion (70-80%) of total design expenses. It is clear from the following picture taken from Verisity website [5]. As a result of this phenomenon, considerable improvements have been made in the digital design testing and verification domain in order to keep up with the technology.

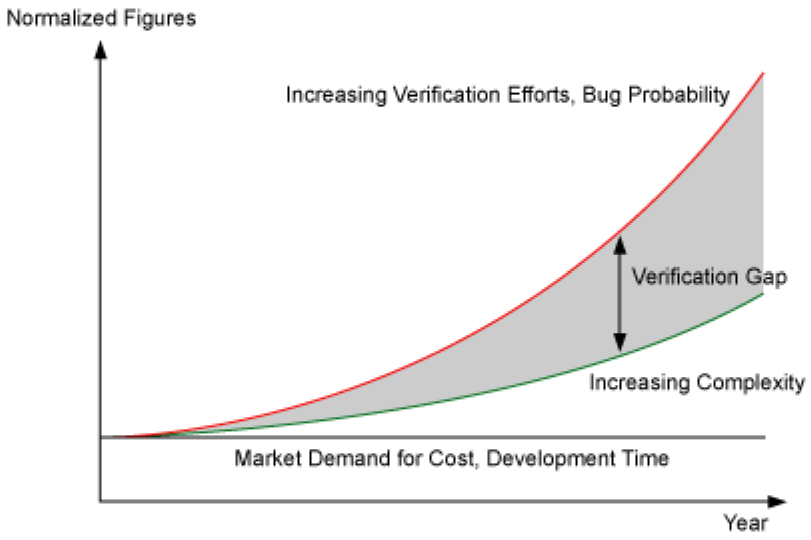


Figure 2-1: The Verification Gap.

The bulk of verification in the EDA industry today is carried out by advanced simulation techniques [7] [8] [9]. Formal Verification tools, such as model checkers and theorem proving tools, use mathematical reasoning to check if a given design adheres to the design properties specified in the functional specifications of the design. Unfortunately, often the abilities of the current formal verification tools fall short of handling the high-complexity designs owing to their computational cost. But lately, hybrid approaches that combine simulation with formal verification techniques have gained popularity [10] [11] [12]. Simulation tools, which are the de facto standard in the verification of industrial designs today, are non-exhaustive in nature and can only guarantee the correct behavior of scenarios that they investigate. On the other hand, Formal verification conducts exhaustive exploration of all possible behaviors. If some behavior is incorrect, a counter-example is provided by formal verification methods. Large sequential circuits present tremendous challenge for the verification community. In this chapter, we discuss some of the concepts regarding verification of sequential circuits. All these techniques can be applied to the Apoptosis model in order to analyze it.

2.1 FSM

Finite State machine is typically denoted by $M(X, S, \partial, \lambda, O)$. There can be a start state S_0 specified as well for the Machine M. Inputs X, Outputs O, States S, Next State Function $\partial(S, X) : S \times X \rightarrow S$, and Output Function $\lambda(S, X) : S \times X \rightarrow O$ completely specify the FSM. A FSM, when implemented in hardware, is shown in Figure 2-2.

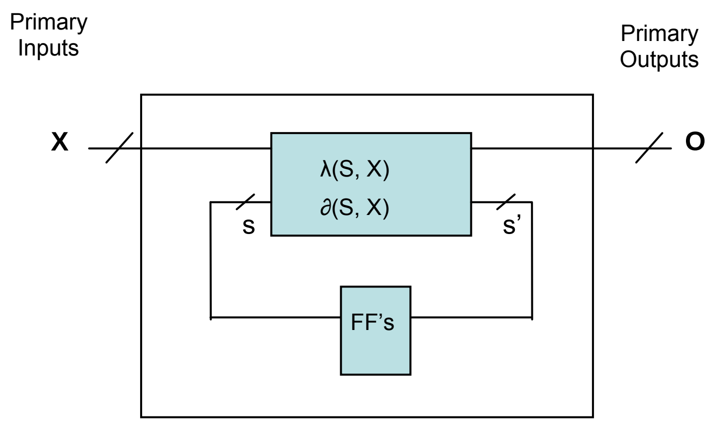


Figure 2-2: Synchronous Sequential Circuit

2.1.1 ILA

Usually many verification and testing techniques work on a FSM by viewing it as Iterative Logic Array (ILA) model. ILA of bound K is shown in Figure 2-3.

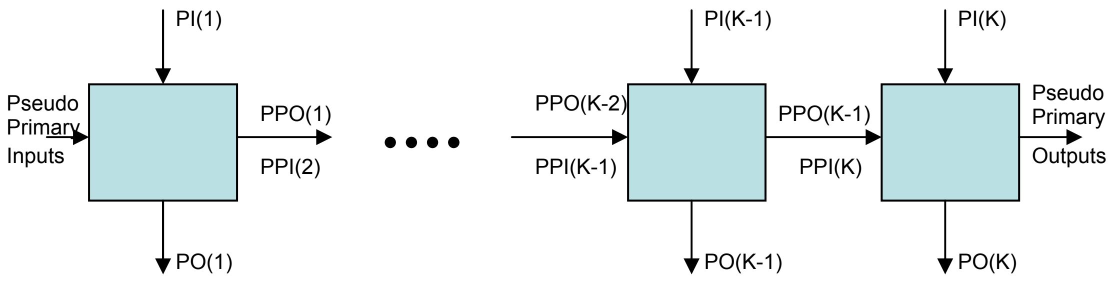


Figure 2-3: Iterative Logic Array

As clear from the figures above, an ILA is constructed by combining together the combinations logic block of the sequential circuit. The consecutive combinational logic blocks are connected together by the state variables. The next state variables of the current block are same as the present state variables of the next block. The first timeframe present state variables are fully controllable like primary inputs and form the Pseudo Primary Inputs (PPI). The last timeframe next state variables are fully observable like primary outputs and form the Pseudo Primary Outputs (PPO) [13] [14]

[15]. For a sequential circuit of n PIs, m POs and f flip-flops, a k time frame ILA will have ((n*k) + f) inputs and ((m*k) + f) outputs.

2.2 SAT (Satisfiability)

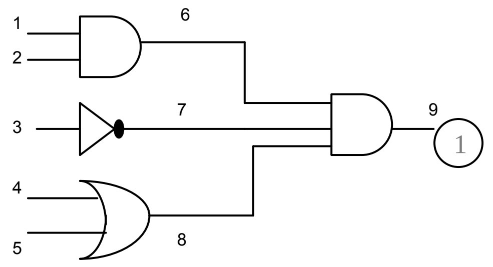
Boolean Satisfiability problem (SAT) is a decision problem, and has the honor of being the first known NP-complete problem, proved by Stephen Cook [16]. Advanced SAT solvers, example Zchaff [17], Berkmin [18] and Minisat [19] have been developed which use efficient heuristics and algorithms to tackle the SAT problem efficiently. Most of the modern SAT solving algorithms are based on variants of DPLL algorithm [20] such as chaff [17], GRASP [21]. A detailed survey of SAT Solving capabilities existing today is given in [22].

The Boolean formula for the Satisfiability problem is usually presented in the Conjunctive Normal Form (CNF), other form being the Disjunctive Normal Form (DNF). A CNF formula Φ on *n* binary variables $x_1, \ldots, x_n$ is the conjunction (AND) of *m* clauses $\omega_1, \ldots, \omega_m$ each of which is the disjunction (OR) of one or more literals, where a literal is the variable or its complement. A function can be represented by many equivalent CNF formulas.

Boolean Satisfiability is the problem of determining if there exists a variable assignment such that the formula evaluates to TRUE. The Boolean formula is termed *satisfiable* if such an assignment exists. If the whole Boolean search space of the input variables is searched and no such variable assignment can be found out, the formula is always FALSE and termed as *unsatisfiable*. SAT Solving today is all about finding that variable assignment efficiently, in less time, using all the advanced heuristics and computer science techniques and providing detailed debugging information on user's fingertips. We provide here an example to understand the basics of SAT as it will be used extensively in later parts of this work.

2.2.1 CNF Example

Consider the following simple sub-section of any arbitrary circuit and the corresponding CNF formula as shown in Figure 2-4.



(6’ + 1).(6’ + 2).(6 + 1’ + 2’).
(7 + 3).(7’ + 3’).
(8’ + 4 + 5).(8 + 4’).(8+5’).
(9’ + 6).(9’ + 7).(9’ + 8).(9 + 6’ + 7’ + 8’).
(9)

Figure 2-4: CNF Formula with output constrained to be logic 1.

The figure above shows the CNF formula of a subsection of a circuit fragment. In the clauses, we see that the variables are present in their complemented form as well. The formula above has 13 clauses. Notice the last clause which is simply (9). This is a *unit clause* and in other words we have constrained this particular gate to be at an output of 1 always. The other clauses simply represent the gate logic functionalities. So, the SAT Solver, when receives these 13 clauses as input, will try to find an input assignment that satisfies the formula or in other words will find an input assignment such that all the clauses are 1 simultaneously. Note that there are 5 inputs and hence there are 5 decision taking variables in this formula. The input space is thus 32.

One such formula-satisfying input assignment is {(1:1), (2:1), (3:0), (4:1), (5:0)}. Note that such a satisfying input assignment is not unique and in this case another satisfying input assignment is {(1:1), (2:1), (3:0), (4:0), (5:1)}.

The majority of the SAT-based techniques used in verification of sequential circuits are done by first unrolling the circuit as shown in ILA and then representing the ILA as a CNF formula. The necessary constraints in form of constraint clauses are then placed on the formula and formula is given to the SAT Solver which returns SAT or UNSAT or some error condition like ABORT/TIMEOUT/MEMOUT. The error conditions are present because there might be certain limits to the extent of time/memory we are going to spend for such a task. In case the formula is SAT, we also receive the Variable Assignment from the SAT Solver. Currently, for academic research, many advanced SAT Solvers like ZChaff and Berkmin are present. We have used ZChaff extensively in our research.

2.3 ATPG

Automatic Test Pattern Generation (ATPG), as the name suggests, generate test vectors for every fault in the circuit according to some fault model. As we saw in the previous section, the basic aim is the same which is to generate an input assignment that satisfies a certain objective in the circuit. Some of the well known ATPG algorithms are PODEM [23], SOCRATES [24], FAN [25], etc.

We utilized a modified PODEM approach in one of our methods to identify illegal states in the Apoptosis FSM. The main point in PODEM is to explore the 2^n possible input combinations to generate an input test vector, if it exists. Hence, a decision tree is generated as the algorithm progresses to find the input combination that satisfies the given objective. We backtrace from the objective to find a value assignment to an input that the heuristics suggest might meet the objective or take us closer to the objective. Then, the implications of the new assignment are asserted in the circuit and a check is performed for the objective. If the objective is met, we obtain a test vector. If the objective is not met, we backtrace in the decision tree and continue in similar manner. A number of heuristics are present which accelerate the basic algorithm by improved decision making and solution space exploration.

The problem formulation of SAT and ATPG are closely related as mentioned in [26]. We might be forced to think that there are important differences between SAT and ATPG in spite of all the similarities, simply because SAT Solvers operate on a CNF while conventional ATPG Algorithms operate on a multi-level Boolean network. However, both these techniques are complementary since both approach the decision problem by a backtrack search in the finite Boolean space that is spanned by the variables of the CNF or the Boolean network, respectively. Hence, in [27], efforts are made to merge the SAT and ATPG domain to exploit the best of both worlds.

2.4 Model Checking

Modeling, Specification and Verification are the fundamental task for model checking. However, before anything else, let's explain what an invariant means. A property Φ is said to be an invariant if that property is true in all the reachable state space of the sequential design. In the Apoptosis model, we are primarily interested in identifying whether a property is reachable in the design. These kinds of properties are termed as *safety properties*. So, an illegal state in the given circuit has the property that it will never occur in the lifetime of the model. Liveness and Fairness are other properties which may be considered.

In model Checking, Temporal Logic is generally used to specify the properties [28] [29] and the system is modeled as an FSM. Binary Decision Diagram (BDD) [30] , a canonical form for Boolean expressions, has traditionally been used as the underlying representation for symbolic model checkers. But as shown in [28] [29], SAT based techniques have gained popularity in Model Checking domain. Without explaining further about BDDs, we focus on the SAT based technique of Model Checking.

2.4.1 Bounded Model Checking

Bounded Model Checking (BMC) coined in terms of Satisfiability problem is one of the current main techniques that has gained popularity in Model checking domain. Broadly speaking, there are two main steps in Bounded Model Checking. The first step is to encode the sequential behavior of a transition system over a finite interval into a propositional formula. The higher the bound, the more it can inform us about reachability. In the second step, the formula is given to a Satisfiability solver, to either obtain a satisfying assignment or to prove there is none. Each satisfying assignment then can be plugged back into a state sequence which reaches the target state from the any of the initial states. In BMC, only finite length sequences are explored. The technique can be used to find counter examples quickly, or it can also be used to verify safety properties for the entirety of the design by looking at only a bounded length sequence.

In the BMC framework shown in Figure 2-5, let the initial state be S_0. T_1, T_2, …, T_k denote each unrolled transition for up to k timeframes. A monitor circuit is present that monitors for the occurrence of the property in any timeframe. The inputs to this monitor circuit are the property assertions of every timeframe. The total CNF instance can be formed by concatenating the CNF formulas for S_0, T_0, T_1, …, T_k, P_1, P_2, …, P_k and the assertion on the monitor circuit. As an example, if we are checking for the reachability of a property in k bounds, then the monitor circuit checks for the reachability of the property in any of the k steps.

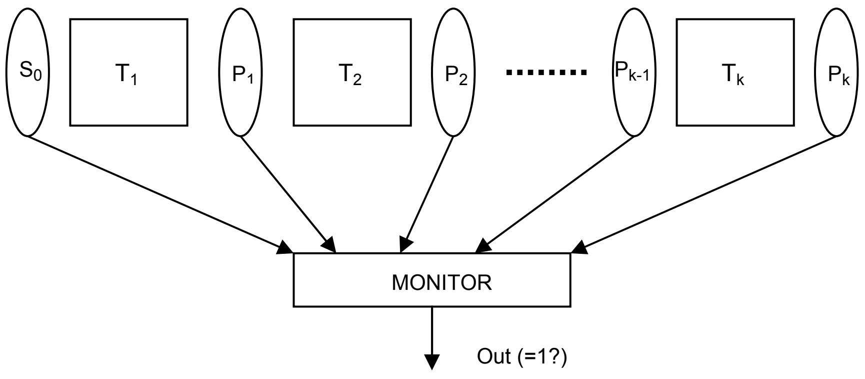


Figure 2-5: BMC Skeleton for Property Checking

2.4.2 SAT Based Induction

Induction is a complete proof technique [31]. Traditionally, Principle of Mathematical Induction can be used to prove that a property P(n) holds for all nonnegative integers n. An induction proof consists of proving the following two sub-goals:

a) Prove that P(0) is true.

b) Prove that for all k, P(k) implies P(k+1).

Based on similar lines, Induction is used in verification to prove an invariant P is true for a transition system by showing that P holds in the initial state set of the system and also that P is maintained throughout the transition relation of the system. The power of induction is that one need not unroll the sequential circuit indefinitely to prove a property.

Windowed Induction is a modified induction technique which has been discussed extensively in [32] for induction proofs in hardware models. To prove that P is an invariant of system M, we do the following:

Find an N for which the following two proofs are achievable:

a) Base: P holds in all paths of length N starting from an initial state.

$S_0(X_0) \wedge T(X_0, X_1) \wedge \ldots \wedge T(X_{N-1}, X_N)$ implies $P(X_0) \wedge \ldots \wedge P(X_N)$.

b) Step: For an arbitrary path of length N+1, if P holds in the first N+1 states, then it holds in state N+2 too.

$T(X_0, X_1) \wedge \ldots \wedge T(X_N, X_{N+1}) \wedge P(X_0) \wedge \ldots . \wedge P(X_N)$ implies $P(X_{N+1})$.

This is clearer from Figure 2-6.

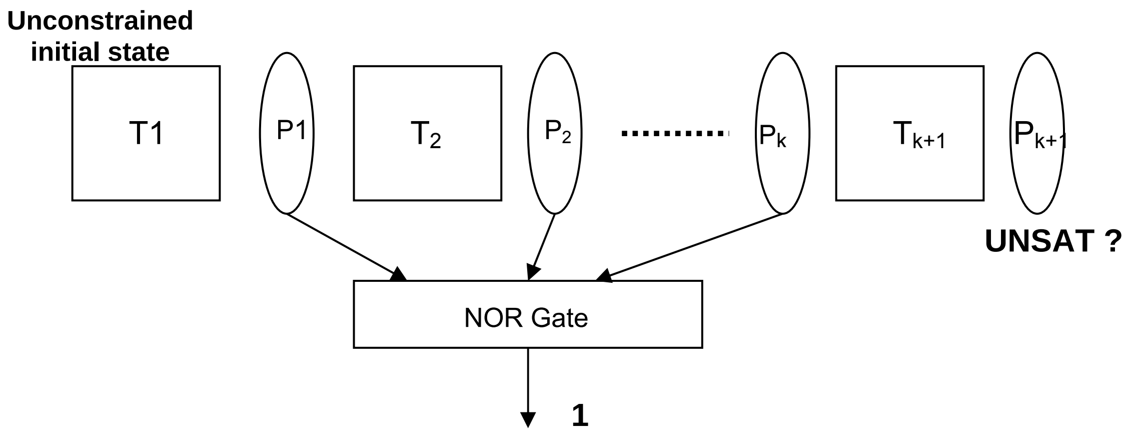


Figure 2-6: Induction step in Bounded Model Checking

For the base case, the MONITOR is an OR gate. We proceed to the induction step only if the base case yields UNSAT. If the base case is satisfiable, it simply means that the property is reachable in that bound of k. For the induction step, the initial state is left unconstrained and for all the timeframes leaving the Pseudo Primary Outputs, the property is constrained to be holding true. Hence a NOR gate is used. The property is then checked to hold at the (k+1)th time-frame. If it is UNSAT again, the property is proved to be an invariant. If the induction step yields SAT (which many times it does), k has to be increased because SAT in induction step indicates nothing is conclusive at that bound k.

2.5 Image and Preimage Computation

We now discuss image computation, which forms an essential part of formal verification. In simple terms, the image of a set of states A is the set of states B that can be reached from A by applying any input vector sequence. Formally, the one-step image of a state s, with the sequential machine having next state function as ∂ and Input Set as I can be defined as:

$$\mathit{Image}(\partial, s) = \{ s' \mid \exists\, \boldsymbol{i} \in \boldsymbol{I},\ s' = \partial(s, i) \}$$

If this operation is performed again and again on the resultant states in the Image set, we reach a point where no new states can be learned. This indicates a complete Image set has been reached. In practice, if the circuit is having a start or a reset state, the image is computed until the Image set is able to reach the target state of interest as clear from Figure 2-7.

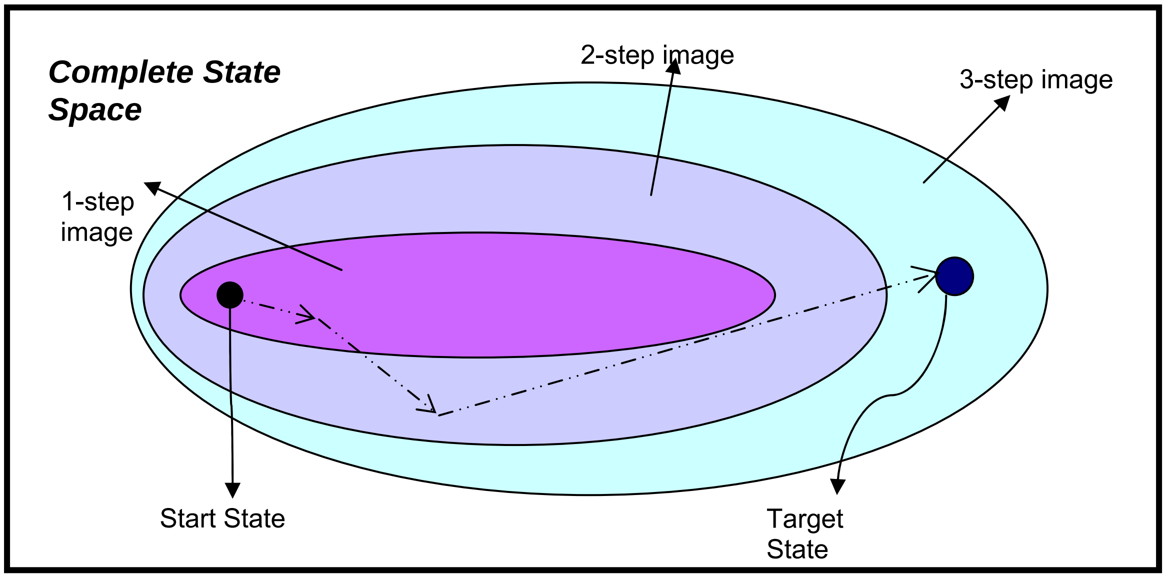


Figure 2-7: Image computation to reach a target state

Similarly, the Preimage of a state s' Є S is defined as the set of states from which the sequential circuit can transition to state s' by application of any input sequence. The one-step preimage of a state s' is thus all those states s of the circuit from which, by the application of an input vector, the circuit can transition to state s'. Formally, the one-step preimage of a state s' can be defined as:

$$\text{Preimage}(\partial, s') = \{s \mid \exists\, i \in I, s' = \partial(s, i)\}$$

Again, a fixed point, when reached as in case of Image computation, indicates a complete Preimage set of state s'.

2.5.1 Image Computation using SAT

As noted earlier [30], the sets of states and sets of transitions are traditionally represented by BDDs. It is well known that while BDDs represent many state sets compactly, they unfortunately suffer from size explosion for many circuits. SAT procedure based image computation and fixed point detection methods have been thus proposed [33] which displays gradual degradation in performance with increase in size and is robust. The run time of these methods depends on the size of the input and circuit diameter as opposed to the added factor of variable ordering in BDDS.

BMC formulates the reachability test as a series of SAT checks for paths of bounded length. The transition relation is unrolled k times to see if a path to a target set of states of length [less than / equal to] exists. For finite systems, the process eventually must terminate as the length of the shortest path between two states cannot exceed the number of states. Hence, if no path is found with length up to the number of states, the target state set is proved to be unreachable. This method, however, does not help in practice because of the large circuits. Diameter of the circuit can provide a good upper bound for k, but to find the diameter of a circuit is again a hard problem.

Image computation can be used to identify illegal states. The design is simply viewed in a single frame as shown in Figure 2-8. Let T be the transition relation of the design, I be the set of Pseudo Primary Input variables, O be the set of Pseudo Primary Output variables, S' be the current image set.

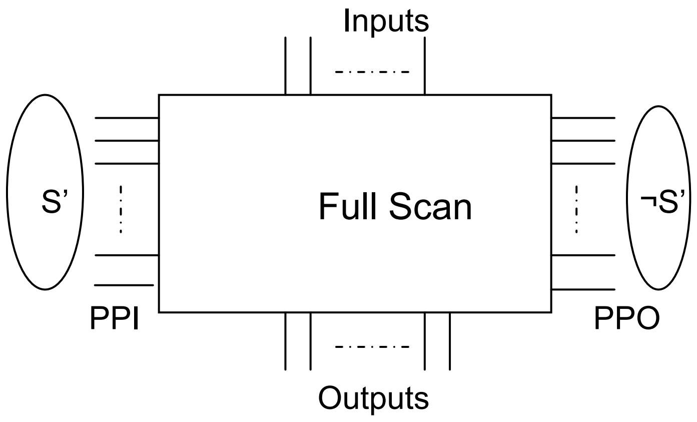


Figure 2-8: One time frame ILA to compute Image.

A satisfying solution to the following CNF formula is then sought from the SAT Solver:

$$(I \equiv S') \land (T1) \land (O \equiv \neg S')$$

If the result is SAT, then a new state is yielded which is added to the Image set of the target state. The term ¬S' is also called *Blocking Clause.*

Chapter 3

ATPG Based Technique to Identify Illegal States

3.1 Motivation

Guided Simulation experiments in [1] conducted on the Apoptosis model have shown to reach very few states for some proteins as clear from the data of individual proteins in Table 1-1. For example, as few as 11 states have been hit and remaining 245 states are still unknown for BAD_mito. For other proteins like BCL2, tBID_mito and tBID_BCL2, more than 50% of the states are still in the unknown category. So it is an indicative of the fact that the unknown states might be illegal, or if they are reachable, they are very hard states to reach.

In an effort to prove that these states are illegal, we investigate an ATPG technique modified to find illegal states. ATPG has been used earlier to identify illegal states quickly as shown in [34]. The idea is to employ a low-cost combinational ATPG to identify unreachable partial-states among groups of related flip-flops.

3.2 Overall Framework

Combinational ATPG algorithms can always be applied on an unrolled sequential circuit when they are considered as Iterative Logic Arrays. We first look at some of the fundamentals of identifying illegal states by looking at the combinations of certain group of flip-flops and then identifying the combinations not appearing in this set. Illegal state can also be explained in terms of n-Cycle-unreachable states. An n-Cycle-unreachable state is defined as a state that cannot be reached from any state in n cycles [34]. And if a state is n-cycle-unreachable then it is also (n+1)-cycle-unreachable. Consider a framework in which the circuit has been unrolled for k timeframes as shown in Figure 2-3. Keeping this in mind, we explain our framework below.

3.2.1 Specifics of the Combinational ATPG application on ILA

Figure 3-1 is a picture of what our circuit will look like.

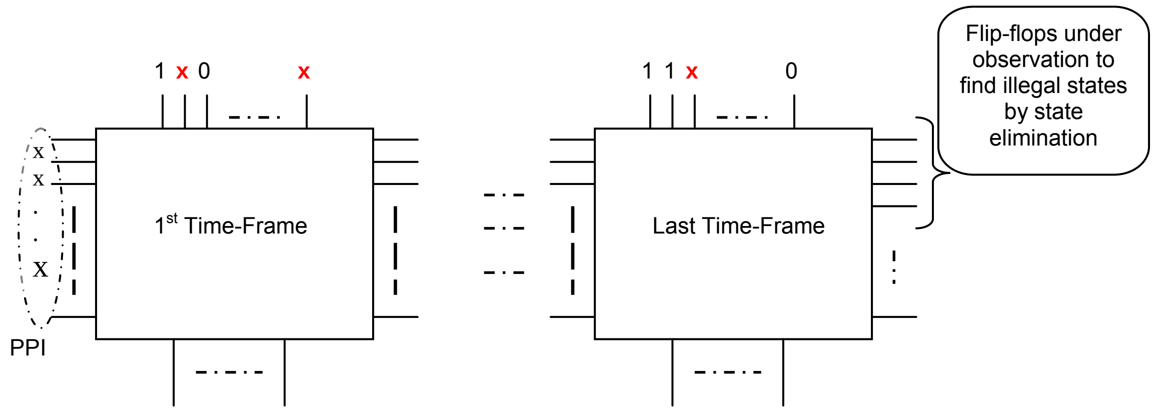
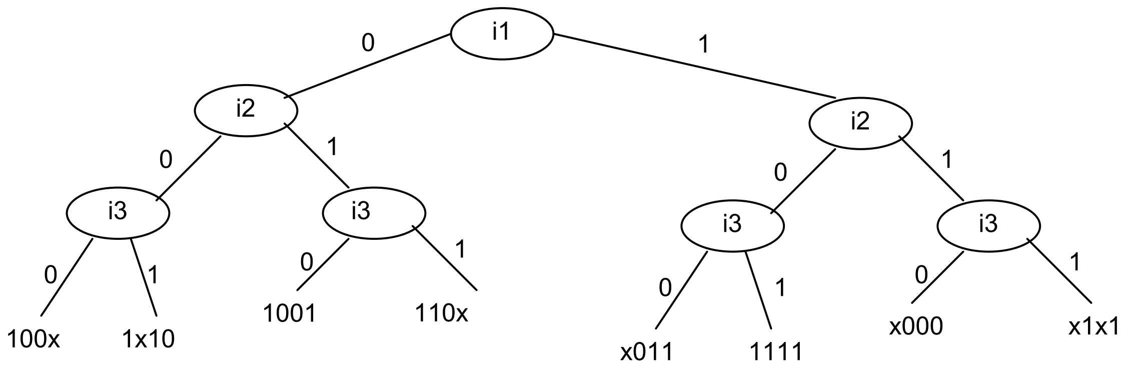

1 x 0 x
1 1 x 0
Flip-flops under observation to find illegal states by state elimination
x
x
.
.
X
PPI
1st Time-Frame
Last Time-Frame

Figure 3-1: An overview of the ILA under modified combinational ATPG approach.

The first thing to note is that the Pseudo Primary Inputs (PPIs) are unconstrained. This means that any state combination is possible at the PPIs. In case the sequential circuit has **n** primary inputs and **m** flip-flops, a **k**-frame ILA will have (k*n + m) primary inputs. Our aim is to keep the PPIs as unspecified throughout our simulation so that any illegal states at the PPO we learn will be true illegal states and not the illegal state with respect to the image of certain starting state. Another main attribute of this approach is that we need to explore the complete input space. If we do not explore the input space completely, and then learn the illegal states by elimination (a technique which we explain next), there might be some input combinations ignored which would have been able to reach some states now falsely considered as illegal.

3.2.2 Learning Illegal States by Elimination

Consider the Figure 3-2.



States visited:

0000 – yes	0100 –	1000 – yes	1100 – yes
0001 –	0101 – yes	1001 – yes	1101 – yes
0010 –	0110 –	1010 – yes	1110 – yes
0011 – yes	0111 – yes	1011 – yes	1111 – yes

Figure 3-2: Identifying illegal states by enumerating all input space.

In the Figure 3-2, we have given an example of complete input space exploration of depth 3 and then analyzing the state values at the leaves of the free BDD built during input enumeration. Keep in mind that the PPIs are left unconstrained. This means that all the states are possible at that initial boundary and hence states that are not visited at the PPOs via this complete input space enumeration are definitely illegal. Another important point to note is that since we are enumerating the complete input space of a certain depth, it becomes infeasible to enumerate the input space of all the inputs. Consider the Apoptosis circuit as an example which has 8 primary inputs. If we perform this experiment on an ILA of 6 timeframes of Apoptosis model, the number of primary inputs to enumerate is 48 which provide an input space of 2^{48} which makes it impractical. So we have to select a proper set of inputs, not too large a number, and enumerate those inputs. This is the reason we also see some don't-care values ('x') in the states at the leaves of the Free BDD.

As an example, consider the following scenario where the state we see is x1x1 at one of the leaves of the BDD. We expand it to mark all the states contained in this cube. Hence, x1x1 expands to 4 states: 0101, 0111, 1101 and 1111. This approach ensures that we conservatively mark all states marked as potentially reachable/legal. But indeed, there can be cases when a state that is illegal might be marked as visited because of the expansion of 'x's in the states.

3.2.3 Details on Maximum Decision Level

Considering the Apoptosis circuit for this kind of experiment, there are 64 PPI in the ILA of k time frames. And if we fix 24 as the Maximum Decision Level (MDL), then all the remaining primary inputs of the ILA (k*n – 24) will be having a logic value of 'x'. We cannot really increase MDL beyond this point because it causes an exponential increase in the input space. A MDL value of 24 means there are 16 million input vectors (and 16 million leaves) for logic simulation and that is somewhat manageable.

Now in order to obtain maximum specified state bits at the Free BDD leaves, we need to heuristically select 24 primary inputs. SCOAP values [35] are a good metric to pick that set. We perform this experiment for a particular set of flip-flops as the target state set to consider. Usually, in a circuit where there is no high level information or where there is no clear partition of the state set, MLP procedure [36] is used to have a fairly good partition of flip-flops where the state variables in each partition have high correlation. This procedure computes the input support for flip-flops and clusters the ones with closer supports. In the case of Apoptosis model, we have a clear picture of the state partition in the form of proteins. Hence, we target these state partitions in the form of proteins to analyze.

3.2.4 Selection of a set of primary inputs in ILA for a particular protein

In order to select 24 (MDL) primary inputs for our target proteins, we use the SCOAP values. To calculate the SCOAP value for a particular protein flip-flops, we assign very high observability values to all other flip-flops and zero observability is assigned to the target flip-flops. Note that observability value of zero means that the signal is completely observable. Since observability values are calculated from outputs towards the inputs, we proceed in a levelized manner to calculate the observability values of the inputs.

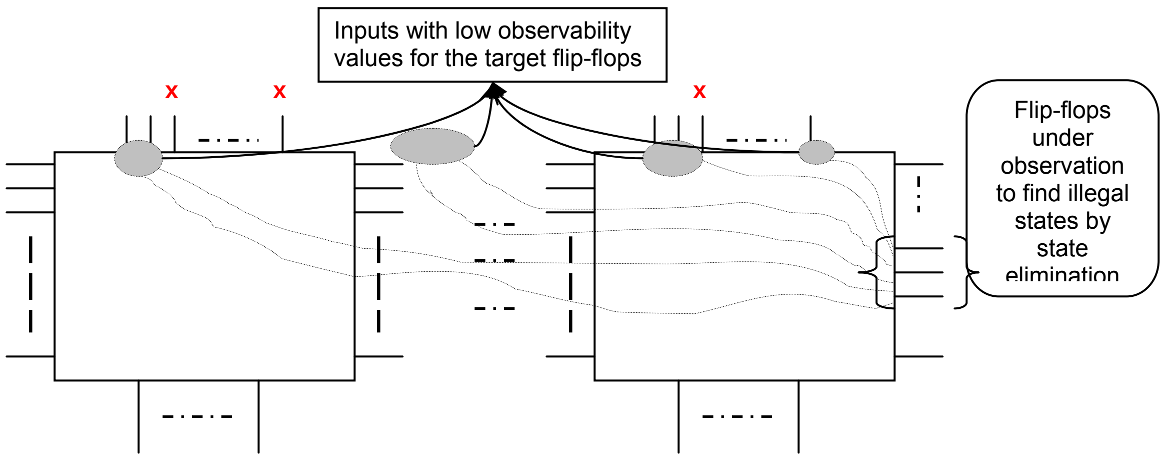

Inputs with low observability values for the target flip-flops
x
x
x
Flip-flops under observation to find illegal states by state elimination

Figure 3-3: After assigning high observability values to all the other PPO, we calculate the observability values of the primary inputs to choose the best set.

It is desirable to obtain the observability values of inputs in such a way that there is a group of inputs having low observability values preferably that number of inputs being near to the MDL and rest of the inputs have high observability values, as shown in Figure

3-4. This simply means that the input set for which we will explore complete input space will consist of the inputs having low observability values (low observability value means the signal is very observable) and we need to pick up MDL number of such inputs.

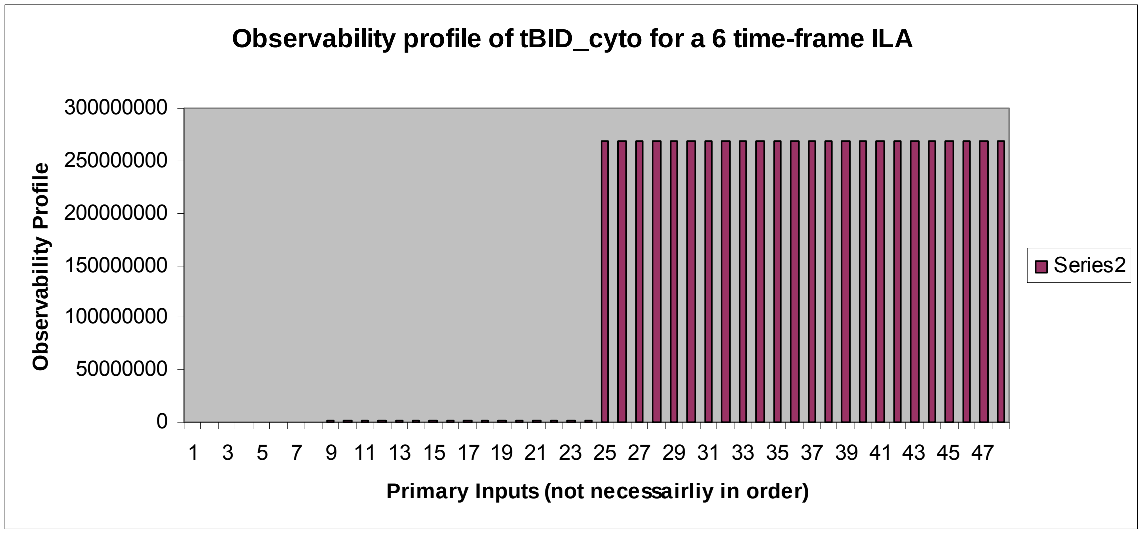


Figure 3-4: Observability profile of the inputs for tBID-cyto in a 6 timeframe ILA. 24 inputs (not necessarily in order) have very low observability values as compared to the other inputs.

3.2.5 Algorithm to identify illegal states

Following from the above discussion, we obtain such input sets for different proteins and try to perform complete input space enumeration of depth 24. Here is the pseudo code for the algorithm:

Algorithm to enumerate the input space and check for illegal states

```
1. Calculate Observability values for the target protein.
2. Select the input set to enumerate for the target protein.
3. Initialize DecisionTree.
4. Decision_level = 0;
5. Get_Illegal_States( decision_level);
6. States not marked are the illegal states.

Get_Illegal_States(decision_level ) {
    this_decision_level = decision_level.
    Take Decicion 0 on the input at the current decision level.
    if (decision_level = = MDL)
            Logic_Simulate( );
            Mark the target state value observed at the PPO;
    else
            Get_Illegal_States(decision_level++).

    decision_level = this_decision_level.
    Take Decision 1 on the input at the current decision level.
    if (decision_level = = MDL)
            Logic_Simulate( );
            Mark the target state value observed at the PPO;
    else
            Get_Illegal_States(decision_level++).
}
```

The main hurdle from this approach was the elimination of as many ‘x’s in the target observed states as possible. All the states were marked as observed due to the expansion of ‘x’s and consequently no illegal states were obtained even though there are many illegal states present in the circuit.

3.2.6 Distinguishing X’s Simulation

The Logic Simulation used in the previous section could not distinguish X arriving at the inputs of a gate. This becomes clearer from Figure 3-5.

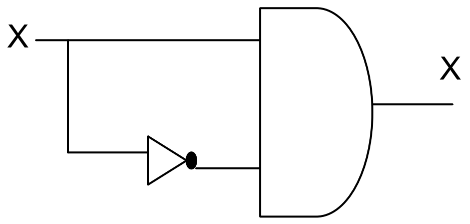


Figure 3-5: Inability to distinguish X

The figure shows that even though the output of the AND gate should be 0 but the logic simulation used previously is not able to judge that and assigns the output as X. To tackle this, we incorporate the Distinguishing X’s feature in logic simulation which takes more time on an average as compared to the normal Logic simulation, but removes some Xs from the simulation trace. It does so by keeping track of which X and ¬X are merging. Each new X has a unique id. But this also proved to be insufficient to find any new illegal states.

3.3 Complete input space enumeration at cut-set boundary

This approach is similar to the approach in Section 3.2, but this time we don’t take decisions on the primary inputs. Instead, decisions are made on the inputs at a cut-set boundary of the fanin cone of the target flip-flops. It becomes clearer from the Figure 3-6. The rest of the approach is same in the sense that we explore the complete input space of depth MDL according to their observability values. We also applied Distinguishing X's simulation with this setup.

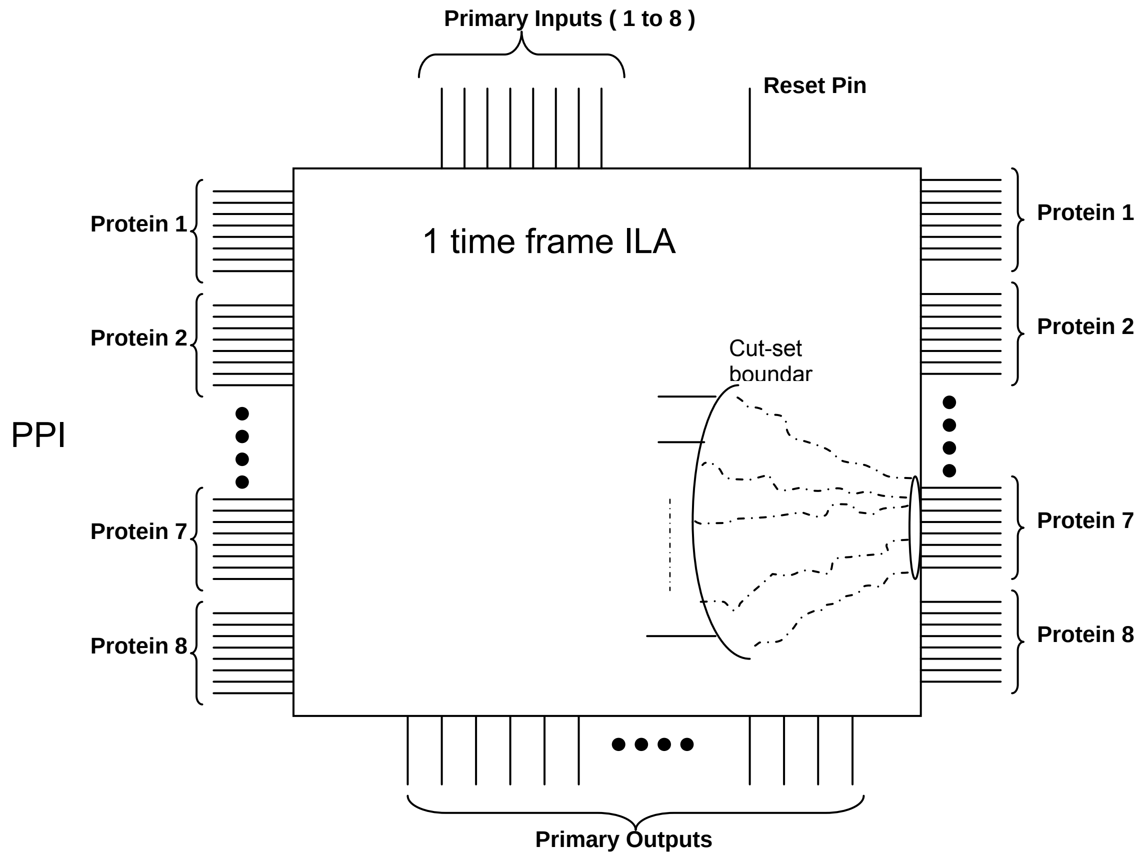


Figure 3-6: Enumeration of Inputs at cut-set Boundary.

3.4 Results

Unfortunately, the results obtained with the approaches discussed in Sections 3.2 and 3.3 were not very satisfactory. In fact, conventional BMC would have provided better results as compared to these approaches. There were cases where lots of 'x' occurred at the leaves of the enumerated decision tree and consequently all the states were marked as visited. In the case of cut-set enumeration, there were cases when some 'x's were eliminated. However, since we are exploring all the input combinations at the cut-set boundary, there are many input combinations which are not possible if the circuit is considered as a whole. Lots of illegal states are thus falsely visited and consequently no good results are obtained as compared to the previous results.

This chapter thus confirms the difficulty of this research problem. Determining the legality of states is extremely difficult. Next, we describe an image computation based methodology which achieves very good results and enables us to dig deeper into the Apoptosis circuit.

Chapter 4

SAT Based Technique to Identify Illegal States

4.1 Motivation

As clear from the approaches discussed thus far, logic simulation (random as well as guided) is only able to reach a subset of reachable states and has proven to be ineffective in identifying any illegal states. Also BMC has not proven to be useful for this Apoptosis circuit, as larger bounds cause the SAT solver to time out [1]. In addition, SAT-based induction, when applied to this circuit to find out illegal states, faces similar kind of problems as lower bounds give inconclusive results and large bounds lead to timing out of the SAT Solver [1]. In Chapter 3, we discussed ATPG techniques for the Apoptosis model. However, it only showed us the tremendous difficulty of this problem.

So far, a large portion of the individual state space of the proteins (concentration values from 0 to 255 in terms of state variable values) has thus been left unknown. For example, referring back to Table 1-1, protein 2 (BAD_mito) has as many as 245 states out of total 256 states left as unknown, with only 11 states found to be reachable. We tackle the problem of proving those unknown states as illegal by proposing a new methodology based on SAT based image computation of abstract circuit. This methodology benefits from the results that have been obtained till now through previous experiments and avoids logic simulation during its runtime. We start here by explaining some basics of circuit abstractions.

4.1.1 Circuit Abstraction

Circuit Abstraction has shown to be a promising approach for Model Checking [38]. Circuit Abstraction attempts to reduce the space complexity of the circuit in order to enable us to draw certain conclusions about the circuit in less time as compared to doing the same analysis in the concrete circuit. As discussed in [38], a commonly used abstraction technique is the localization reduction as shown in Figure 4-1.

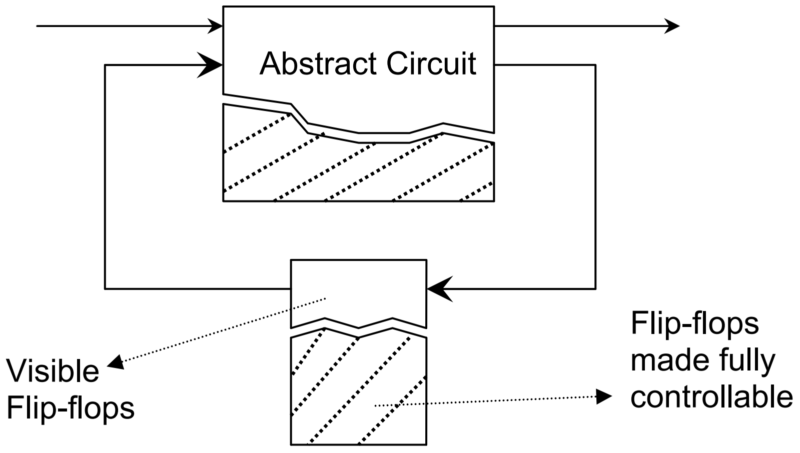


Figure 4-1: Localization Reduction in Circuit Abstraction

The abstract model is computed from the concrete circuit by making some flip-flops fully controllable in the circuit. The flip-flops made fully controllable are called *invisible* and the remaining flip-flops are termed *visible*. Localization Reduction leads to an abstract circuit which is an over-approximation of the original circuit in terms of reachable states and hence an under-approximation in terms of illegal states. This is because the constraints present due to flip-flops in the concrete circuit are now absent in the abstract model. Hence, the reachable state space is increased and some states that are illegal in concrete model may now be reachable in the abstract model due to the invisible flip-flops. However, the illegal states of abstract circuit are definitely illegal in the concrete model as well. The state space comparison of the abstract and concrete models is illustrated in Figure 4-2.

A counterexample on an abstract system hence may not correspond to any realizable path in the concrete system, in which case this counterexample would be called spurious. To get rid of spurious counter-examples, the abstraction needs to be made more precise via refinement.

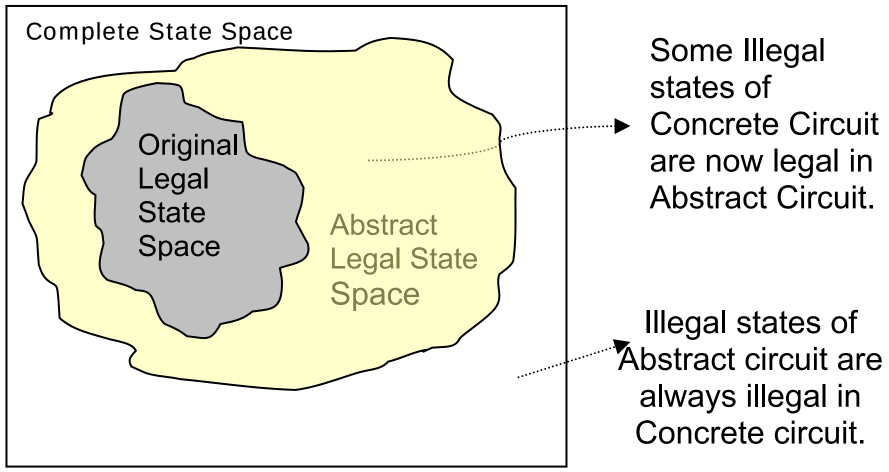

Complete State Space
Original Legal State Space
Abstract Legal State Space
Some Illegal states of Concrete Circuit are now legal in Abstract Circuit.
Illegal states of Abstract circuit are always illegal in Concrete circuit.

Figure 4-2: State Space Comparison of Concrete and Abstract Model

Abstraction-refinement is a general strategy for automatic abstraction. Abstraction-refinement usually involves the following process:

1) Generation of an initial abstraction
2) Model check the abstract system
3) If a counter-example is produced, check whether the counter example holds on the concrete system. If no counter-example exists in the abstract model, the property holds also in the concrete models.

If a counterexample is obtained in the abstract system, we try to simulate it on the concrete system symbolically using a SAT solver. The refinement is performed by restoring a small set of invisible variables in the abstract model. Research has already been done [38] to identify such variables, called the refinement variables, through the analysis of Boolean Constraint Propagation and Conflicts during the SAT checking run of counterexample simulation.

4.1.2 Use of previously generated results on the Apoptosis Model

In our work, we are assisted by the fact that we have a very clear picture of the state space partition by the distinct proteins. Hence, we are able to focus our efforts on a very specific problem: that of identifying as many illegal states as possible of individual proteins. Once illegal states of individual proteins are identified, we can continue to study the correlation of combination of proteins. We try to build up a good starting point for our methodology by extracting relevant information from the simulation traces we already have from the application of GALS. This helps us in reducing the time taken to find out the illegal states. We describe in detail our methodology in the next few sections.

4.2 General Framework Overview

In order to expand our learning of the Apoptosis model, we propose a new framework, for which the pseudo-code is as shown in Algorithm 4.1. The target of this framework is to extract illegal states of a particular target protein. We refer to the core procedure of our framework as **extract_illegal( )**. As clear from Algorithm 4.1, we methodically constrain the one time-frame ILA as follows:

1. Constrain the target protein at the Pseudo Primary Inputs (PPI) to be in the current set of reachable states as obtained from the simulation trace of the concrete model. The simulation trace should be obtained from the concrete model because the simulation trace of the abstract model will contain many

illegal states of the concrete model and this will inhibit our purpose. Details are shown in Figure 4-3.

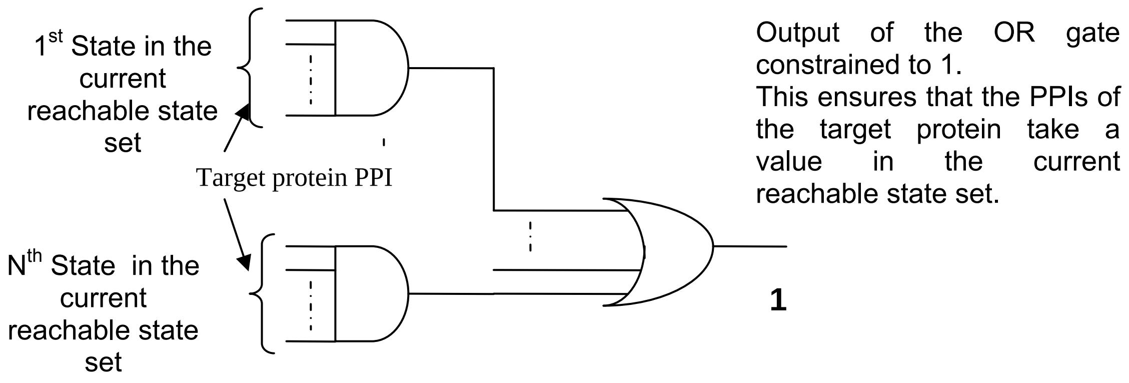

1st State in the current reachable state set
Target protein PPI
Nth State in the current reachable state set
Output of the OR gate constrained to 1.
This ensures that the PPIs of the target protein take a value in the current reachable state set.
1

Figure 4-3: PPI Constraint Example

2. Constrain the target protein at the Pseudo Primary Output (PPO) to be not in the current reachable state set. This ensures that if the SAT Solver returns a satisfying solution, we will obtain a new reachable state in the abstract circuit and consequently add it to the current reachable state set and constrain it properly again in the next iteration. This is formulated by a NOR gate as shown in Figure 4-4.

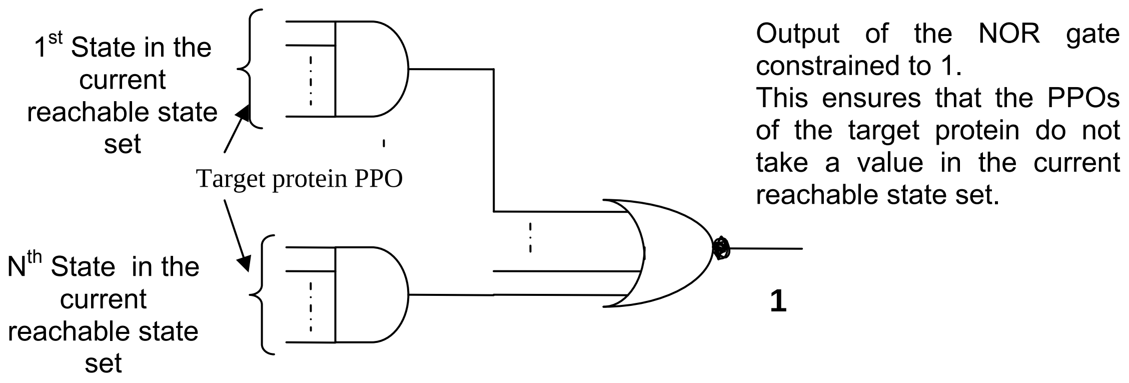

1st State in the current reachable state set
Target protein PPO
Nth State in the current reachable state set
Output of the NOR gate constrained to 1.
This ensures that the PPOs of the target protein do not take a value in the current reachable state set.
1

Figure 4-4: PPO Constraint Example

Algorithm 4.1: Framework to extract illegal states

```
  extract_illegal (target_protein) {
0        -Unroll the circuit for only one timeframe.
1        -Extract the reachable states of the target protein from the simulation trace
2        -Constrain the target protein at PPI to be in the current reachable state space
3        -Constrain the target protein at PPO to be not in the current reachable state space.
4        -Constrain the illegal states learned till now (by any method) on the PPIs and PPOs.
5        -Leave the other PPIs (other than the target protein) unconstrained resulting in an abstracted
          circuit.
6        while (TRUE) {
              switch (ila.sat_solve( )) {
                case UNSAT:
7                   -Extract the illegal states by taking a difference of the reachable set obtained
                     till now from the complete state space of target protein.
                    Exit;
                case SAT:
8                   -Extract the protein flip-flop assignment at PPO.
9                   -Add this newly learned state to the constraints of protein flip-flops at PPI
                    Continue;
              } end switch
         } end while
  }
```

3. Use the illegal states learned till now to constrain both the PPI and PPO. This is a very important step as this can significantly constrain the abstract circuit state space and aid the SAT Solver to reach conclusions quickly. It is done in a manner similar to step 2. The state bits to be constrained can belong to any protein group in this case. The learned illegal states are mostly in the form of state cubes; for example, an illegal partial state of the form 1X0X engulfs 4 illegal states in total, namely 1000, 1001, 1100 and 1101. This is one of the advantages of working on abstract circuits because any illegal states that we learn are state cubes which are compact in representation.

4.2.1 Extraction of Illegal States from Image Computation

As noted from the above Algorithm, the non-target protein PPIs are left unconstrained. When the SAT solver is called to solve this constrained model, it gives either SAT or UNSAT as the result. If the result is SAT, the target PPO variable assignment is extracted from the SAT Solver and this represents an additional potentially reachable state. This state is added to the constraints at the PPI (through the OR gate) and PPO (through the NOR gate). This assignment is actually part of the image of the target protein in the abstract circuit. The process continues until the SAT solver returns ‘UNSAT’. At this point, we have gathered all the states that are potentially reachable in this abstract circuit.

As an example, let us take a particular set of states in which 8 flip-flops are involved. We extract the state combinations corresponding to these flip-flops from the concrete model simulation traces. Let that set consisting of different state combination values be S. There are in total 256 different state combinations possible ranging from 00000000 to 11111111. As the algorithm progresses and it keeps on adding newly learned reached states of the abstract model to this set, the set grows and finally comes to a halt when the SAT Solver returns ‘UNSAT’. At this point, let’s say that the set S consists of *n* states. Hence, out of total 256 states, the remaining (256-n) states are illegal in the abstract model and are thus also definitely illegal in the concrete model. We cannot say anything about the reachability of all of the *n* states of set S. The reason is that many of these states (those states not in the initial starting set S) have been reached in the abstract model and hence they might be illegal in the concrete model. But we gather the difference, 256-n states in this case, as the illegal states.

4.2.2 Need to iterate with continuously increasing constraints

After a call to ***extract_illegal( )****,* we obtain some illegal states of a particular target protein. This function is repeated for each different target protein. However, there is a possibility that we still have not obtained all of the illegal states of a given target protein. This is possible due to the following two reasons:

a) The incompleteness of the illegal state set may render some illegal states to be reachable in our abstraction. In other words, if a state s2 (which is illegal in the concrete model) is found to be reachable in the abstract circuit, s2 might further make another illegal concrete state s1 reachable in the abstract circuit.

b) An illegal state s1 of target protein p1 might be reachable from another illegal state s2 of some other protein p2. Initially, protein p2 is not constrained in the abstract model of the target protein p1. So, effectively all the 256 states are possible at the PPI corresponding to p2. Once we constrain illegal states of p2, we reduce the possible state combinations of p2 at the PPI from 256 to some lower value. However, we haven’t yet learnt enough illegal states of protein p2 to add to the constraints at PPI. It then becomes possible to reach state s1 of target protein p1 in p1’s abstract model when illegal states of protein p2 are not yet constrained. An example of this is shown Figure 4-5.

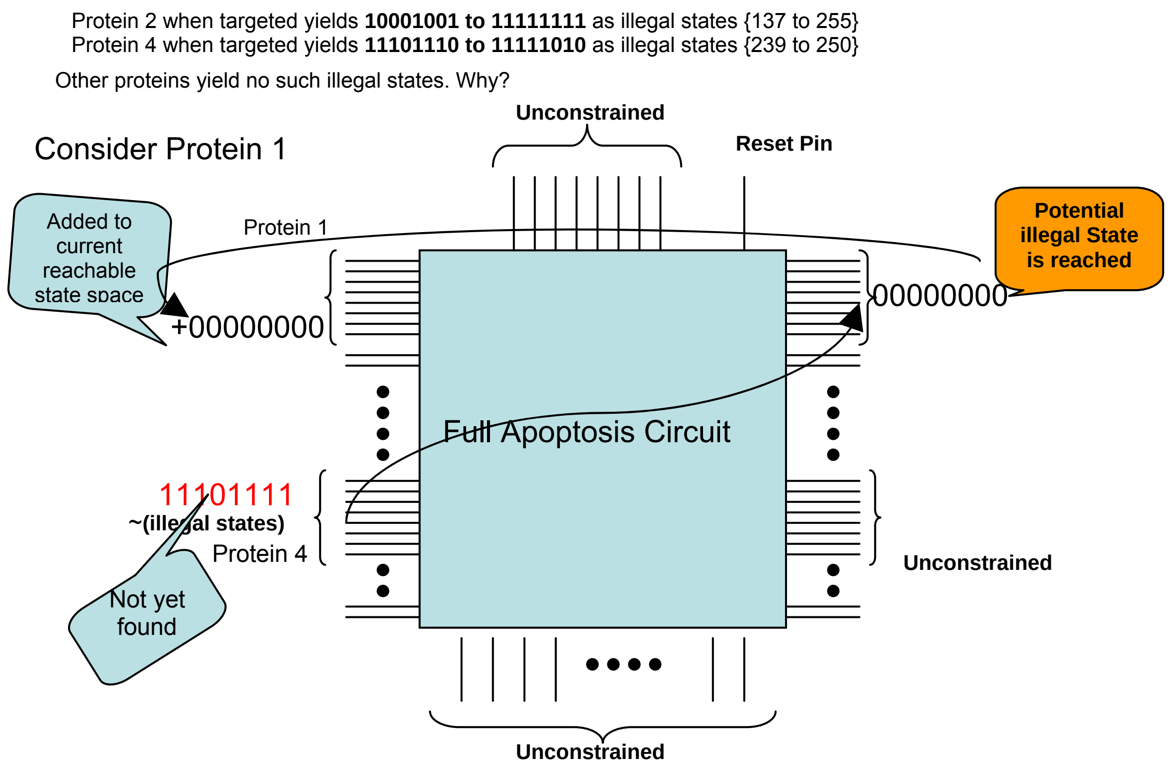

Protein 2 when targeted yields 10001001 to 11111111 as illegal states {137 to 255}
Protein 4 when targeted yields 11101110 to 11111010 as illegal states {239 to 250}
Other proteins yield no such illegal states. Why?
Unconstrained
Reset Pin
Consider Protein 1
Added to current reachable state space
Protein 1
Potential illegal State is reached
+00000000
00000000
Full Apoptosis Circuit
11101111
~(illegal states)
Protein 4
Not yet found
Unconstrained
Unconstrained

Figure 4-5: A reason why all illegal states are not detected.

Figure 4-5 shows the setup of the algorithm ***extract_illegal( )*** for protein 1 (BCL2). In this setup, we did not find any illegal states of protein 1 in the first pass, even though we were able to learn new illegal states for other proteins such as protein 2 (BAD_mito) and protein 4 (BAD_cyto) when they are targeted. A possible reason for not being able to identify any illegal state for protein 1 is that some illegal states of protein 4 are necessary to constrain the search when targeting protein 1. However, during the first pass, illegal states for protein 4 have not yet been learned. One such illegal state is 11101111 which is not yet constrained at the PPIs for protein 4. Without the knowledge of this illegal state, we might have computed state 00000000 of protein 1 as a potentially reachable state. But we have no way to avoid this until and unless we constrain illegal states of protein 4 at PPI. Furthermore, the lack of knowledge of illegal states in other protein may result in marking many illegal states as potentially reachable.

Hence in order to avoid such incomplete illegal state sets, we have to keep on learning and adding any proved illegal states to the constraints at PPI. We need to thus create a feedback loop in order to keep on learning new illegal states as depicted in the pseudo-code, *iterate_proteins( )*:

Iteration of extract_illegal( )

```
function iterate_proteins( ) {
   list global_illegal_states;
   for_each (protein p)
         -extract_illegal(p, global_illegal_states);
         -store the newly learned illegal states in the global_illegal_states;
   end for_each
}
```

It has to be noted that this function has to be called until we are unable to find any more new illegal states. Hence, there are several iterations of the function *iterate_proteins( )* as well.

4.2.3 Results and Discussion of individual protein analysis

Figure 4-6 shows the progress of learning illegal states of different proteins with iterative calls to *iterate_proteins( )*. We follow an order from protein1 to protein8 regarding the calls to this function. As evidenced from Figure 4-6, for protein2

(BAD_mito), the illegal states learning increases with the increase in learning illegal states of protein 4 (BAD_cyto) and protein 7 (tBID_BCL2) in the first few iterations. The fixed point, that is when we are able to learn no new illegal states, is reached after the 11th iteration.

The graph also shows the interdependency of the proteins in terms of defining their illegal states. We were not able to learn any new illegal states of protein8 (tBID_cyto) during the first 5 iterations. During the 4th iteration, new illegal states of protein6 (tBID_mito) are learned (even though the number of illegal states of other proteins remain constant in the 4th iteration). Due to all the learned illegal states in the first 5 iterations for other proteins, we finally learn a few new illegal states of protein8 in the very next iteration. Just before the last iteration, all proteins except protein8 (tBID_cyto) have achieved maximum possible illegal states. We learn more illegal states of tBID_cyto in the last iteration due to all the learned illegal states of other proteins.

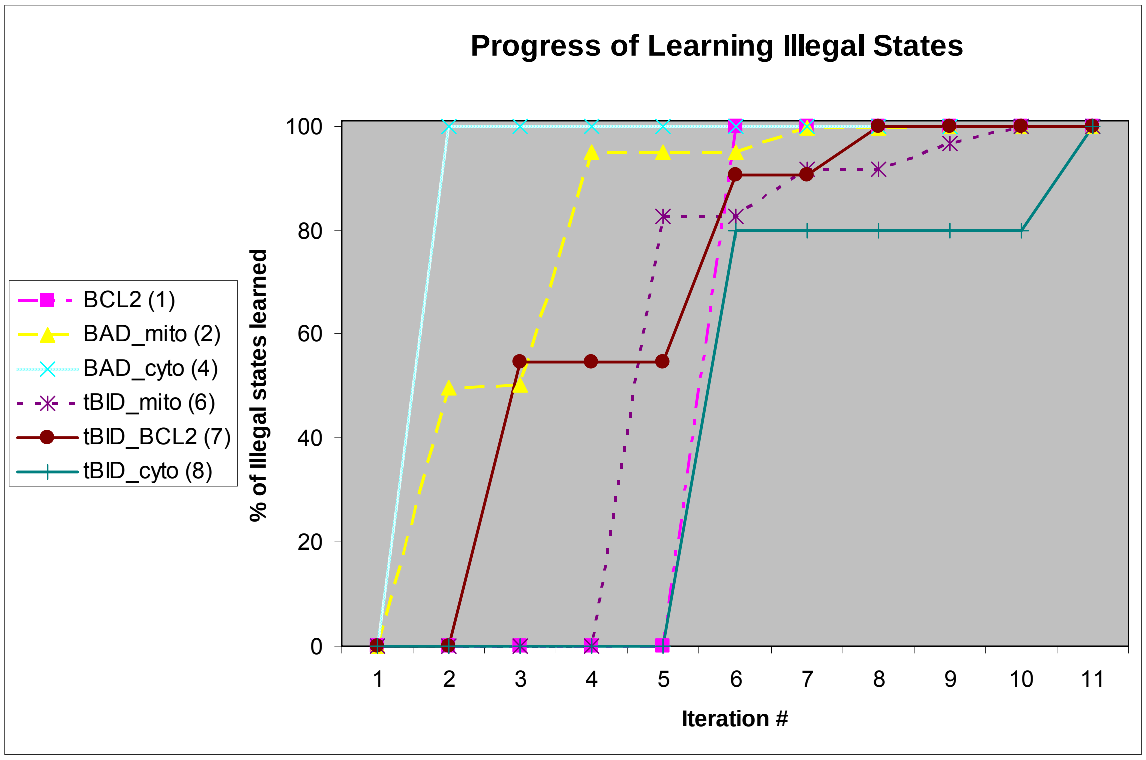


Figure 4-6: Graph showing the progress of learning of illegal states as we iterate.

Protein 5 (BAD_BCL2) did not give any new illegal states out of the remaining 20 unknown states. This might be due to the fact that it is quite independent of other flip-flops in the circuit, or perhaps it also depends a lot on the other unknown states of other proteins (20 states of protein 6 / 11 states of protein 7). Further analysis will show us that some state variables are quite independent of each other in this circuit. From the iteration graph, we see that protein6 and protein8 are greatly interdependent as we are able to find more illegal states of protein8 whenever we find more illegal states of protein6. The combined protein analysis that we conduct later will show that protein6 and protein8 are definitely interrelated and generate more illegal states when analyzed together.

Table 4-1 is a summary of results in the learning of new illegal states in Apoptosis model. And Table 4-2 below shows the different ranges in the protein state space that we were able to identify as illegal.

Table 4-1: Comparison of current and previous results

Proteins	**Illegal states**		**Unknown States**	
	Old Approach	Abstraction based new approach	Old Approach	Abstraction based new approach
BCL2 (1)	1	**131**	130	**0**
BAD_mito (2)	0	**239**	245	6
BAD_p14 (3)	1	**5**	4	**0**
BAD_cyto (4)	1	**17**	16	**0**
tBID_mito (6)	9	**133**	144	20
tBID_BCL2 (7)	0	**131**	142	11
tBID_cyto (8)	224	**244**	20	**0**

Table 4-2: Details of the states in the range 0-255

Proteins	Reachable States Range	Illegal States Range	Unknown States Range
BCL2 (1)	1 to 125	0, 126 to 255	NA
BAD_mito (2)	0 to 10	17 to 255	11 to 16
BAD_p14 (3)	0 to 250	251 to 255	NA
BAD_cyto (4)	0 to 238	239 to 255	NA
BAD_BCL2 (5)	0 to 235	?	236 to 255
tBID_mito (6)	1 to 103	0, 124 to 255	104 to 123
tBID_BCL2 (7)	139 to 252	0 to 127, 253 to 255	128 to 138
tBID_cyto (8)	0 to 11	12 to 255	NA

As evidenced from the tables, our approach based on image computation of the abstract circuit was able to reduce the number of unknown states significantly. This further helps us to analyze protein combinations as discussed in the next section.

4.3 Study of protein combinations

Till this point we have been studying the protein states individually and we had no idea about how any two or more proteins may interact with each other. An attempt was made in [1] using simulation as well as SAT-based Induction to study state space partition where two proteins are involved but the results were unsuccessful for many state combinations due to the following reasons:

a) With a lot of state space of individual proteins lying in the "unknown" region, there was little opportunity to obtain good results when two protein interactions are studied, even with symbolic simulation techniques.

b) A study of protein interactions through reachable state space analysis via simulation was conducted, but it did not give good results. As we know from all the experiments done till now, random and guided logic simulation based techniques have shown to work poorly on Apoptosis model.

c) The many unknown states simply present a very large unknown state space. Consequently, we cannot sufficiently constrain the state space and the target state space blows up which hampers the analysis.

We try to analyze two-protein interactions now that we have a more complete knowledge of the illegal state space. From the results, there were several proteins for which there is no unknown state remaining (e.g., tBID_cyto, BAD_p14, etc…). This provides a good starting point for analyzing protein pairs.

4.3.1 Initial Approach

We need to analyze only the cross product of legal state space of any 2 proteins. Table 4-3 depicts the magnitude of cross-products of legal state spaces that we will have to deal with.

Table 4-3: Reachability spaces of individual proteins

Protein type	*Reachable*	*Unknown*	*Illegal*
BCL2 (1)	**125**	0	131
BAD_mito (2)	**11**	6	239
BAD_p14 (3)	**251**	0	5

Protein type	*Reachable*	*Unknown*	*Illegal*
BAD_cyto (4)	**238**	0	18
BAD_BCL2	**236**	20	0
tBID_mito (6)	**103**	20	133
tBID_BCL2 (7)	**114**	11	131
tBID_cyto (8)	**12**	0	244

For example, proteins 1 and 2 have potentially 125*11 = 1375 legal state combinations. Proteins 3, 4 and 5 have many more individual legal states and in particular there cross product leads to many states, close to 65000 state combinations. The cross-product of proteins 1 and 6 involves 125*103=12875 states. We start our initial approach and then try to improve our method of illegal state computation. Our initial method is as follows:

Method to calculate illegal states in protein combinations

```
-constrain all the illegal states of individual proteins learned till now at PPI and PPO
-while (no new illegal states can be learned) {
        -for_each (legal combination of the two protein state space) {
                -constrain the PPOs to the target state combination
                switch (ila.sat_solve( )) {
                   case (SAT):
                       // state is reachable in the abstract circuit, hence no conclusion
                   case (UNSAT):
                      - add the state combo to the illegal constraints at the PPI
                 }
        }
}
```

This method has to be run till it yields no new illegal state combinations for the target protein pair. Although it gives good results for cases where the number of legal states to consider is less than 2000, for cases where number of states to be considered is greater than that, it simply takes too long to go through the possible value combinations. In other words, this algorithm becomes less practical for cases where the number of states is large, especially for those involving 65000 state combinations. Nevertheless, the initial results are encouraging for some of the combinations as shown in Table 4-4.

Table 4-4: Illegal state combinations

Protein Combination	*Illegal states found out of total legal states possible*
1_2	3 out of 1375
1_8	**96 out of 1500**
2_4	25 out of 2618
2_5	1 out of 2596
2_6	1 out of 1133
6_8	**305 out of 1236**

This basic method was tried for protein pair 3 & 4 which have 59738 potential legal states and it yielded only 1 illegal state out of 59738 states, and that took a long time because there were so many SAT instances to be solved. So an alternate method had to be developed to tackle this problem. In addition, as discussed in Section 4.2.2, the same reasons apply here also. It is possible that many illegal states were in fact never learned and falsely being reached due to abstraction. Surely, protein combinations 1_8, 2_4, 6_8 have shown some good results with this naïve initial approach which encourages us to improve it even further.

4.3.2 Improving the naïve approach

There are several opportunities for improvements to the previous approach. One such approach is the application of Algorithm 4.1 to the protein combinations instead of individual proteins. We have a database of around 5300 different states (all 64 flip-flops considered) from the simulation trace. The size of the initial reachable state set for a single protein is typically less; for example, for tBID_cyto, only 11 states out of 256 states are extracted from the simulation trace. As we started with the reachable state set from the simulation trace of the concrete circuit in order to learn illegal states of individual proteins, we can also have the same kind of approach for protein combinations as well. The size of the initial reachable state set for two protein combinations is found to be around 150 to 200 states as obtained from the simulation traces. The pseudo-code is as shown in Algorithm 4.2.

Algorithm 4.2: Framework to extract illegal state combinations

```
  extract_illegal (target_combination) {
0     -Unroll the circuit for only one timeframe.
1     -Extract the unique reachable states of the target 2-proteins from the simulation trace
2     -Constrain the target 2 proteins at PPI to be in the current reachable state space
3     -Constrain the target 2 proteins at PPO to be not in the current reachable state space.
4     -Constrain the illegal states learned till now (by any method) on the PPIs and PPOs.
5     -Leave the other PPIs (other than the target protein) unconstrained resulting in an abstracted
       circuit.
6     while (TRUE) {
          switch (ila.sat_solve( )) {
            case UNSAT:
7             -Extract the illegal states by taking a difference of the reachable set obtained
               till now from the complete legal state space (cross product) of target 2
               proteins.
              Exit;
            case SAT:
8             -Extract the 2 protein flip-flop assignment at PPO.
9             -Add this newly learned state to the constraints of 2 protein flip-flops at PPI
              Continue;
          } end switch
      } end while
  }
```

This approach leads to better results for those protein combinations which also gave some illegal states with the approach in Section 4.3.1. The results are displayed in Table 4-5.

Table 4-5: Comparison of different approaches.

Protein Combination	**Illegal states from Section 4.3.1**	**Improved approach of Section 4.3.2**
1_8	96 out of 1500	606 out of 1500
6_8	305 out of 1236	627 out of 1236

As shown in Table 4-5, the revised method has given good results in the cases mentioned. We will present the detailed graphs, depicting the illegal state space more clearly later. Results for other protein combination were not that encouraging and were similar to the results as obtained in Section 4.3.1. Also for cases where the number of states of two proteins combination is greater than 3000, it takes a long time to finish this procedure. We need to thus tackle this problem of large number of states and reduce the runtime in order to make it more scalable.

4.4 An incremental approach to calculate illegal states

We now propose an incremental technique to learn the illegal states for those protein combinations which have very large potentially legal state spaces. When the cross product of legal state space of two proteins becomes more than 10000, it is impractical to compute the illegal states as we suggested in Section 4.3.2. It will simply take too long because of large number of SAT solver calls involved. Our aim is to avoid the large number of SAT solver calls and still be able to learn maximum illegal states.

4.4.1 The Role of Most Significant Bits in Identification of Illegal States

We now discuss a technique which helps us tackle the problem discussed in previous section. We explain the technique with the help of following example. Consider the state 1101XXXX. This state specifies the states ranging from 11010000 to 11011111 i.e., from 208 to 223 in terms of integer values. Now consider the state bits of 2 proteins taken together. Let us consider a state 1101XXXX1100XXXX. The first 8 bits belong to protein A and last 8 bits belong to protein B. Only the 4 Most Significant Bits (MSB) are specified for both the proteins. When we enumerate the MSB only, there would be a total of 256 different combinations possible ranging from 0000XXXX0000XXXX to 1111XXXX1111XXXX. Hence, we only explore the MSBs under observation and leave the other bits as unknown. Suppose we obtain the state 1101XXXX1100XXXX as an illegal state. This specifies that any state between 208 to 223 of protein A together with any state between 192 to 207 of protein B would form an illegal state combination.

Hence, in order to study two protein combinations, if we consider the MSBs of the two proteins together and then we find illegal states comprising of those MSBs (other bits being don't cares), we discover a large number of illegal state combinations more quickly. We learn illegal state combinations quickly because of the following simple reason: if all the 8 bits of both the proteins are specified, there are 16 bits to expand and therefore 2^{16} different state combinations are possible which will take a long time to analyze. However, when only 4 MSBs of each protein are specified, there are only 2^{8} different state combinations possible and we are able to reach a fixed point in the algorithm quickly. It is easy to extract the initial reachable state set of the MSB bits from the simulation traces. The rest of the process is then quite similar to Algorithm 4.2. We extract illegal states by taking a difference of the states reached so far from all possible combinations (256 total state combinations in case of 4 MSB learning).

We are able to learn more illegal states with 5 MSB learning as compared to 4 MSB learning, due to the reason explained here. Consider state cube 1100XXXX. This state cube denotes states with value ranging from 192 to 192+15. If the states 192 to 192+7 constitute legal state combinations with the other protein and states 192+8 to 192+15 form the illegal state combination with other protein, then 4 MSB learning will not yield conclusive results. However, when we consider 5 MSB learning, 11000XXX (192 to 192+7) will denote legal state space with other protein and 11001XXX (200 to 200+7) will denote illegal state space.

Hence, apparently, 5 MSB yields better results due to its higher resolution but the penalty lies in the number of SAT iterations because of large number of different states possible. The number of states to consider are obviously more in case of 5 MSB learning (1024 states) as compared to 4 MSB learning (256 states). We adopt the procedure of first learning the illegal states with 4 MSB learning. This presents us with only 256 different combinations to search and illegal states are learned quickly. However, illegal states learned are not complete because of less resolution of 4 MSB learning. The illegal states learned from 4 MSB learning are then used as constraints for 5 MSB learning and so on. 5 MSB learning will consist of 1024 different state combinations to target for (in total, 10 bits of two different proteins are specified). Similarly, 6 MSB learning will consist of 4096 different state combinations to target for (in total, 12 bits of two different proteins are specified). Since the state space is constrained at each step due the illegal states learned in previous steps, new illegal states are learned more quickly.

4.5 Results

Individual protein analysis results have already been discussed in Table 4-1 and Table 4-2. Now we discuss some of the results obtained for the two protein analysis. Although we obtain lot of illegal states during our analysis, here we show some of the significant results and discuss them in detail.

The first graph in Figure 4-7 shows the illegal state space in the cross product of legal states of protein 6 (tBID_mito) and protein 8 (tBID_cyto). After several iterations of learning illegal states and constraining them repeatedly at the PPIs, we were able to prove 728 illegal states (out of 1236 total states) in the cross product of legal state space of protein tBID_cyto and tBID_mito.

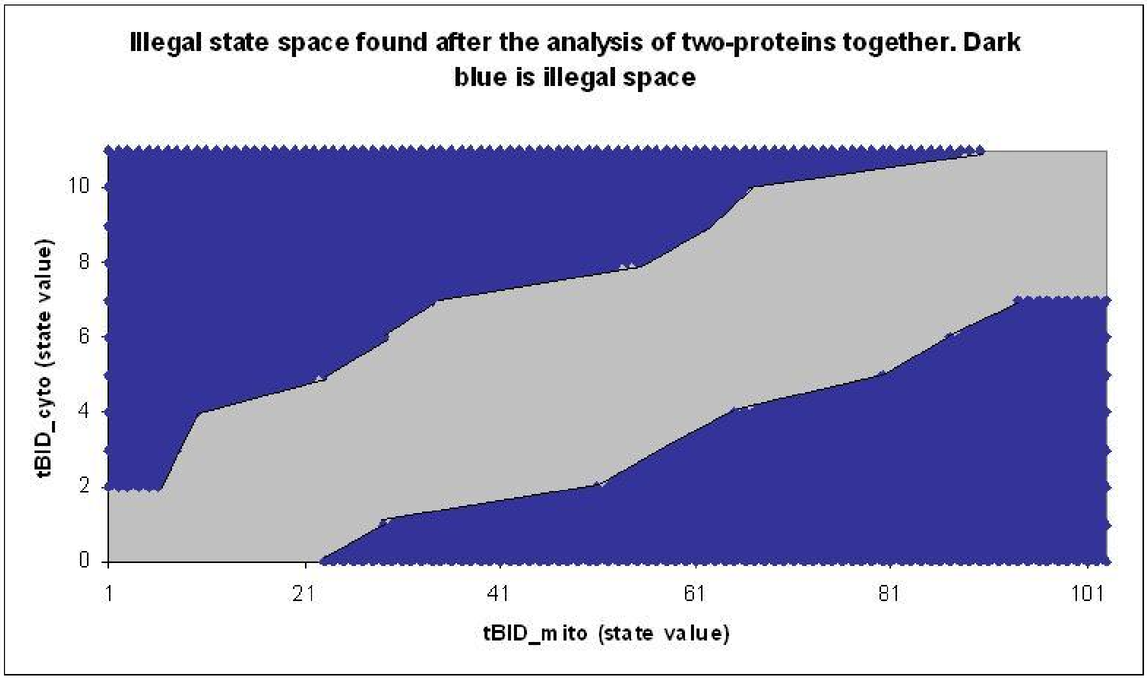


Figure 4-7: Illegal states in the cross product of the legal states of tBID_mito & tBID_cyto

Similarly, in the cross product of legal state space of BCL2 and tBID_cyto, we proved 658 states (out of 1500) states as illegal. Figure 4-8 shows a clear graph for this analysis.

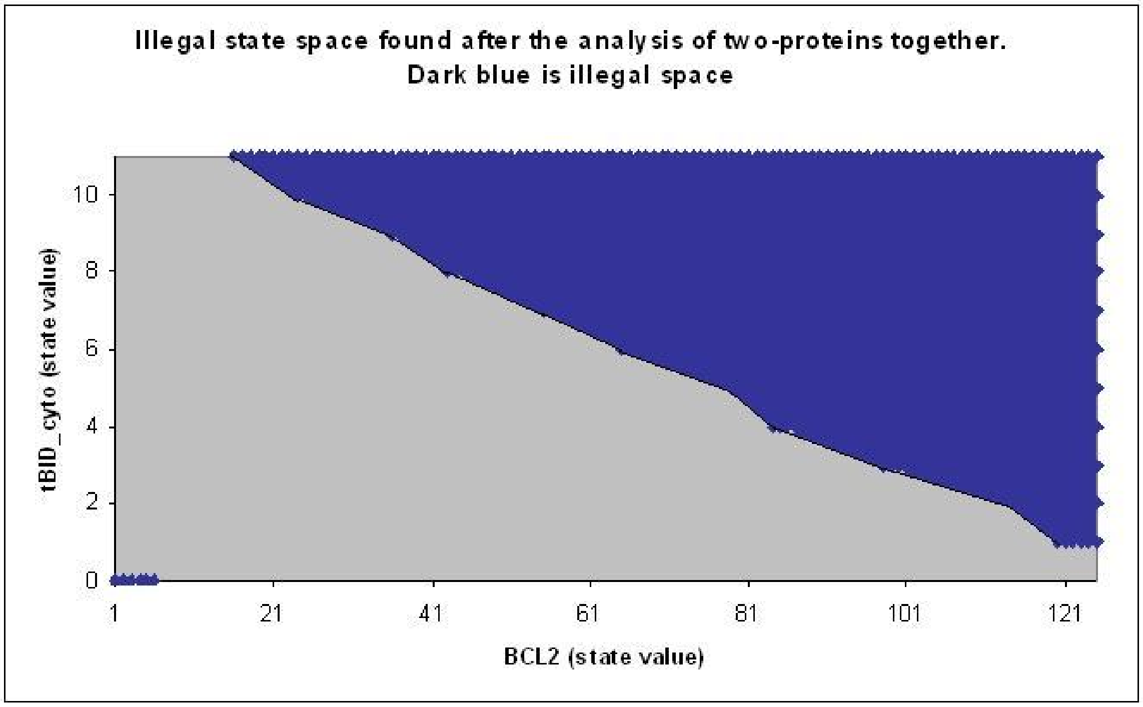


Figure 4-8: Illegal states in the cross-product of the legal states of BCL2 & tBID_cyto

As clear from the previous two figures, one can picture how the two proteins behave in conjunction with each other.

In Figure 4-9, we specify the relationship between the proteins BAD_cyto and BAD_mito, although the large space denoted as not illegal is unknown (could be either illegal or reachable). All we say is that the space marked in dark color is definitely illegal.

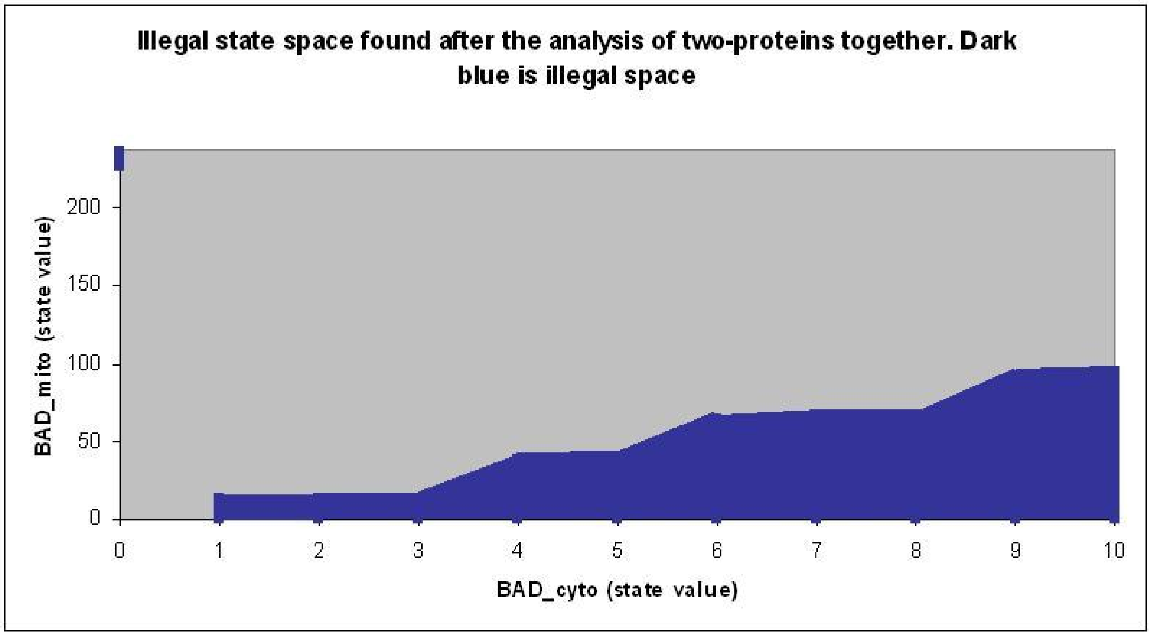


Figure 4-9: Illegal states in the cross-product of the legal states of BAD_mito & BAD_cyto

Next, we give an example which specifies the importance of the MSB learning approach (Section 4.4.1) that we adopted with regards to the study of illegal states in those cases where legal state cross products is very large. BCL2 and tBID_mito have 12876 states as possibly legal. When we first perform the 4 MSB learning, we get the following graph as shown in Figure 4-10.

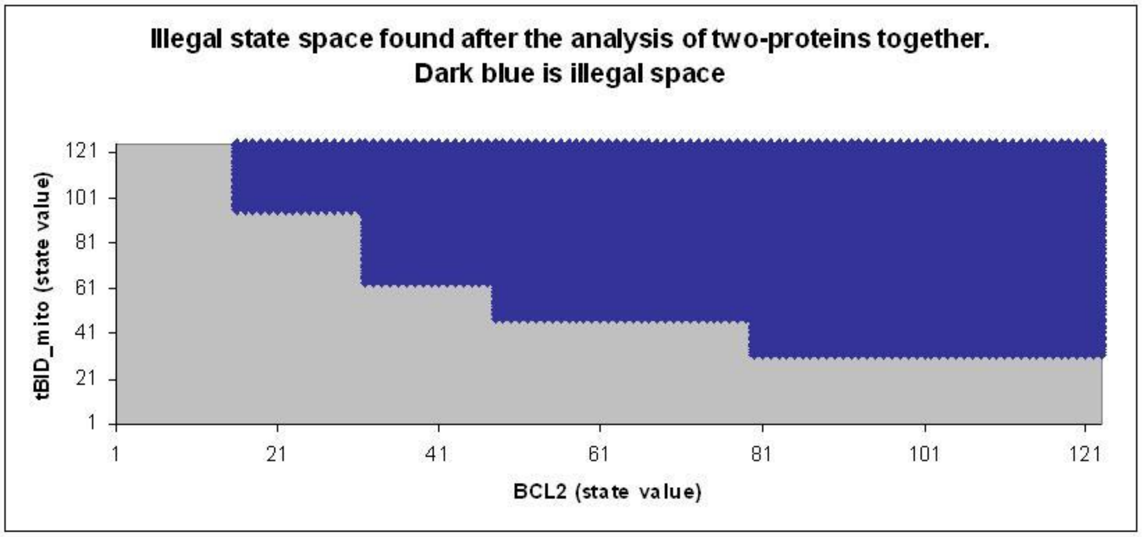


Figure 4-10: 4 MSB illegal state learning for BCL2 and tBID_mito showing the less resolution of the illegal states obtained as indicated by the rectangular boundaries. But the advantage is that we learn many illegal states in less time.

One notable point is that tBID_mito still has 20 unknown states. These 20 states (ranging from 104 to 123) could not be proven either legal or illegal in the individual protein analysis. But as apparent from Figure 4-10, we were able to see that states 104 to 123 of tBID_mito also present some illegal states in conjunction with BCL2.

As clear from the argument that we gave in Section 4.4.1, since the resolution is less in case of 4 MSB learning, we get crude blocks of illegal states as clear from the rectangular boundaries in the above graph. More illegal states are learned with 5 MSB learning as shown in Figure 4-11.

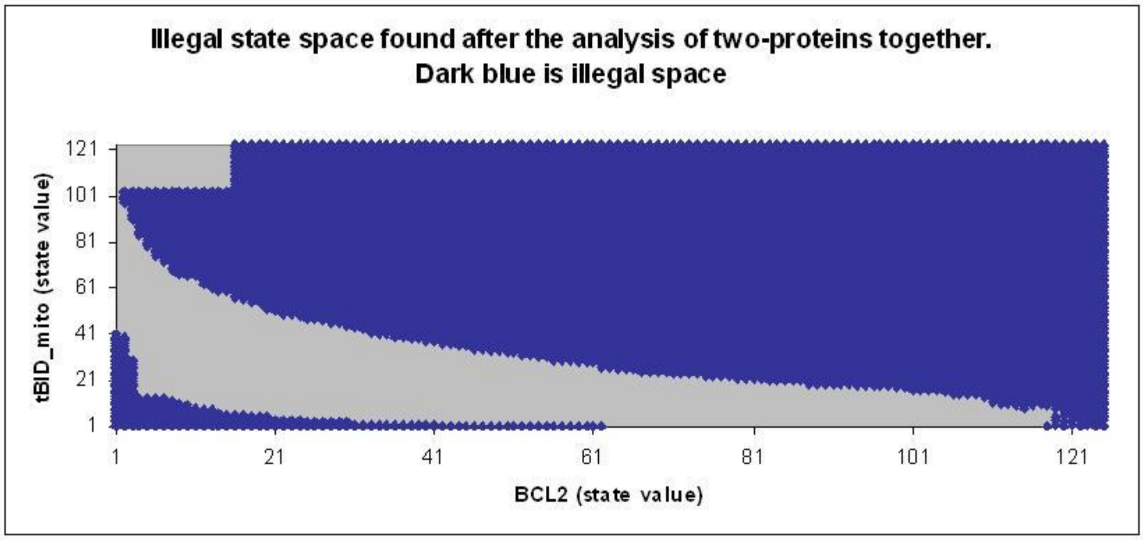


Figure 4-11: 5 MSB learning refines and adds to the illegal states obtained from 4 MSB learning for proteins BCL2 and tBID_mito. No longer there are any rectangular boundaries which are indicative of possibility of more illegal state learning.

Here is a summary of learned states for two protein combinations.

Table 4-6: Illegal state combinations

Protein Combination	**Illegal states found out of total legal states (cross product) possible**
1_2	3 out of 1375
1_8	**658 out of 1500**
1_6	**9233 out of 12876**
2_4	**524 out of 2618**
2_5	2 out of 2596
2_6	4 out of 1133
6_8	**728 out of 1236**
3_4	1920 out of 59738

To have an overall view of the amount of illegal states learned when two protein interactions are considered, take a look at Figure 4-12 and Figure 4-13.

Figure 4-12 shows the interaction of BCL2 with tBID_mito. Since Biological systems are continuous transition systems, we can safely assume that the non-illegal space surrounded by a rectangle (1 to 125 (x-axis) with 104 to 123 (y-axis)) is also illegal in Figure 4-12.

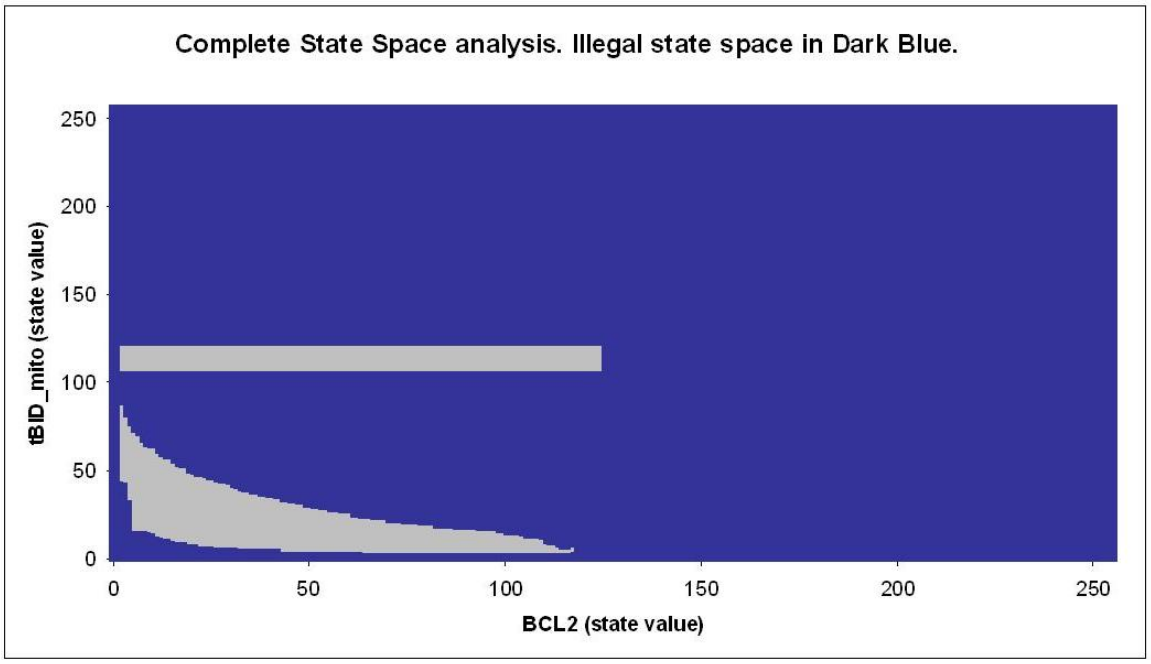


Figure 4-12: Complete state space overview of BCL2 and tBID_mito

Figure 4-13 shows how tBID_cyto and tBID_mito interact with each other. It is quite clear that the two proteins can only be in the narrow zone shown in the graph. Apart from that narrow zone, the rest of the state space is illegal.

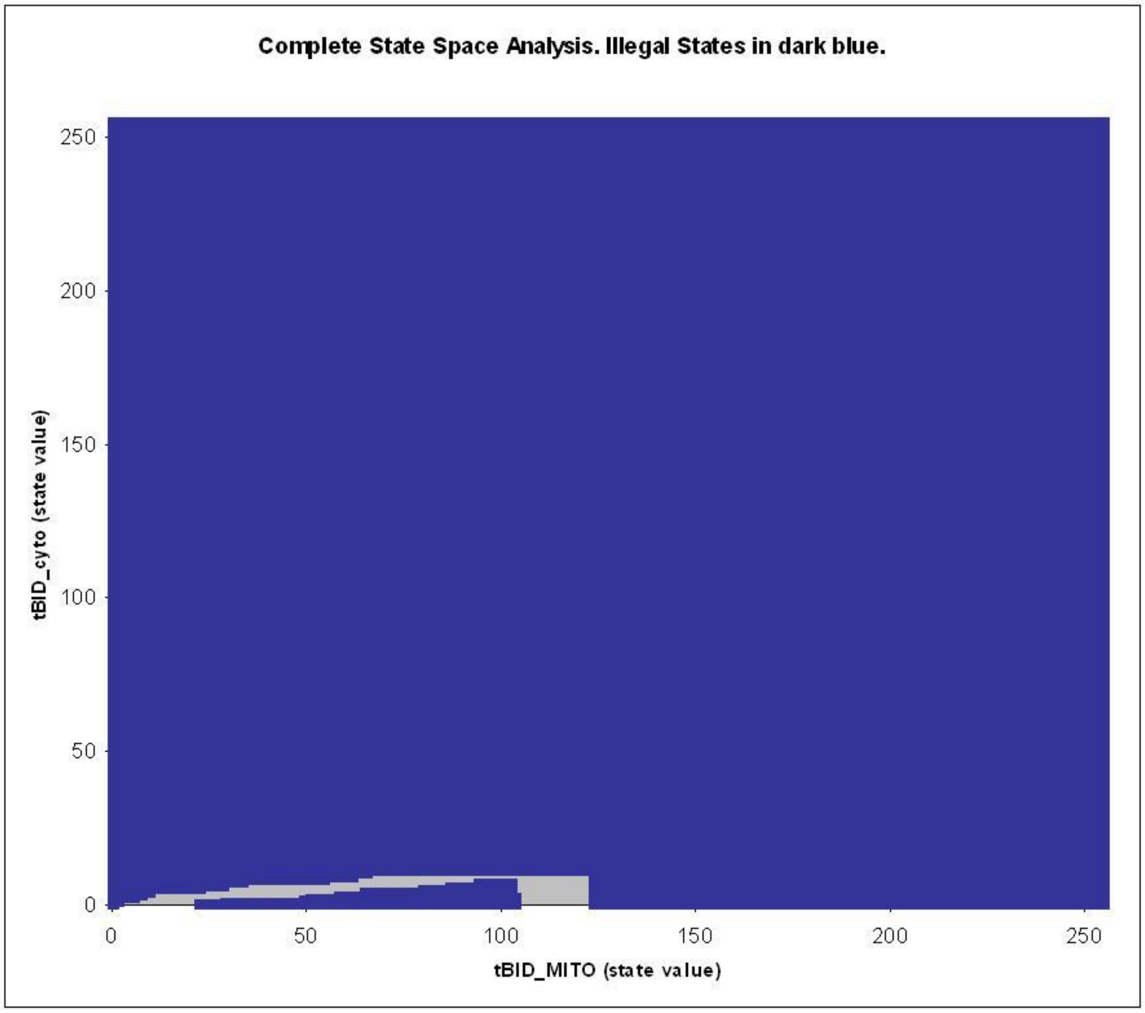


Figure 4-13: Complete state space overview of tBID_mito and tBID_cyto

Just a fact to be noted again is that the non-illegal state space is unknown (either reachable or illegal). We are unable to assert anything about that. More information can be obtained about those regions through simulation of the system. From the discussed results, it is evident that the amount of learning we have been able to achieve, using techniques discussed in this thesis, is quite significant for the Apoptosis Model.

Chapter 5
Conclusion

We have been able to obtain a clear picture of the Apoptosis model state space. With the application of GALS, BMC and SAT-based Induction in earlier research, we were able to analyze the Apoptosis model to a large extent. On the other hand, the limitations of the previous method render us an incomplete picture of the reachability space of the Apoptosis model. From the results presented in the earlier work, the majority of the state space of different proteins was lying in the unknown region.

In this thesis, we tried to further analyze the states in the Apoptosis model and reduce that unknown region. Since the state space partition information is available to us in the form of different proteins, we are able to focus our efforts for the in-depth analysis of different proteins. We first tried to use ATPG and simulation based methods to identify illegal states and even tried Distinguishing X's logic simulation to get better results. But no improvements were obtained, reassuring us of the fact that in the Apoptosis model, a drastically different approach is needed.

Next, we presented a methodology to identify illegal states using image computation of the abstract Apoptosis circuit. Using various kinds of abstractions for different proteins, we were able to extract many illegal states in the Apoptosis model. These newly learned illegal states further constrained the Apoptosis model and we were able to converge to a fixed point where no more illegal states could be identified using this method. The unknown state space was thus reduced by a large margin.

Once we had a clearer picture of the individual proteins, we were able to study the state space of the protein combinations using a similar method and once again much information was obtained on how proteins might behave in combination with each other. We presented a MSB based approach to quickly identify illegal states for those protein combinations where simple iterative procedure might have taken a long time to compute illegal states.

As a result of our work, the number of states that were previously in the unknown region is greatly reduced. We have developed an improved methodology to identify illegal states in the Apoptosis model. Fault analysis can now be efficiently conducted because we have a more complete knowledge of the state space. The illegal states can be added as constraints in all kinds of analysis such as SAT-based Induction, analysis involving implication graphs. More pathways can be added to the model and we will be able to study more complex models as well because of the scalability of our abstraction based methodology.

Bibliography

[1] "Discrete Transition System Model and Verification for Mitochondrially Mediated Apoptosis Signaling Pathways". *Huy Lam*, M.S. Thesis, Bradley Department of Electrical and Computer Engineering, Virginia Tech, June 2007.

[2] Apoptosis. Linda J. Miller and Jean Marx. Science, 281(5381): 1301-, 1998.

[3] Cell death in us and others. Pierre Golstein. Science, 281(5381): 1283-, 1998.

[4] Reactome: a knowledge base of biological pathways, *G. Joshi-Tope, M. Gillespie, I. Vastrik, P. D'Eustachio, E. Schmidt, B. de Bono, B. Jassal, G.R. Gopinath, G. R. Wu, L. Matthews, S. Lewis, E. Birney, and L. Stein*. Nucleic Acids Research, 33:D428-D432, 2005.

[5] http://www.verisity.com/resources/whitepaper/soc_nec.htm

[6] Enhancing signal controllability in functional test-benches through automatic constraint extraction. *Guzey, O., Wang, Li.-C., Bhadra, J.* Test Conference, 2007, ITC 2007, IEEE International.

[7] Leveraging Design Insight for Intelligent Verification Methodologies, EDA DesignLine, *Chris Wilson*, Nusym Technologies, Inc.

[8] Verification Flow Optimization using an Automatic Coverage Driven Testing Policy, *Y. Lahbib, O. Missaoui, Hechkel, D Lahbib, Badreddine M, R Tourki*, 2006 International conference on DTIS in Nanoscale technology.

[9] A Verification Synergy: Constraint-Based Verification. *Carl Pixley, John Havlicek*, Electronic Design Process 2003 symposium.

[10] Abstraction Guided Semi-formal Verification. Ankur Parikh. M.S. Thesis, Bradley Department of Electrical and Computer Engineering. Virginia Tech. June 2007.

[11] Distance Guided hybrid verification with GUIDO. *Smitha Shyam, Valeria Bertacco.* Design, Automation and Test in Europe. 2006.

[12] Validation with Guided Search of the State Space. *C. Han Yang and David Dill.* 35th Design Automation Conference, June 1998.

[13] Combinational ATPG theorems for identifying untestable in sequential circuits, *V.D. Agrawal, S.T. Chakradhar,* ETS, 1993

[14] On identifying undetectable and redundant faults in synchronous sequential circuits, *I. Pomeranz, Sudhakar Reddy,* VLSI Test Symposium, 1994

[15] Improving the performance of automatic sequential test generation by targeting Hard-to-Test Faults. *L Lingappan, Niraj Jha*, Proceedings of 19th International Conference on VLSI Design (VLSID' 06), 2006.

[16] The Complexity of Theorem Proving Procedures. *S. A. Cook*. Proc. Of 3rd Annual ACM symposium on theory of computing, 1971.

[17] Chaff: Engineering an efficient SAT Solver. *M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, S. Malik,* 39th Design Automation Conference (DAC 2001), Las Vegas, June 2001.

[18] BerkMin: a fast and robust SAT-solver (2002). *E. Goldberg, Yakov Novikov*. In Design, Automation and Test in Europe (DATE' 02).

[19] Minisat: A SAT Solver with conflict-clause minimization. *N. Een, N. Sorensson*. International Conference on Theory and Applications of Satisfiability testing, Poster, 2005.

[20] A Machine program for theorem-proving. *M. Davis, G. Logemann, D. Loveland*. Communications of the ACM, vol. 5, no. 7, pp. 394-397, 1962.

[21] GRASP: A search algorithm for Propositional Satisfiability. *J. M. Silva, K. Sakallah.* ICCAD 1996 IEEE.

[22] The quest for efficient Boolean Satisfiability Solvers. *L.Zhang, S. Malik*. Proc. of Computer Aided Verification, 2002.

[23] An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuit. *P. Goel*. IEEE Transactions on Computers, vol. C-30 pp. 215-222, March 1981.

[24] SOCRATES: A Highly Efficient Automatic Test Pattern Generation. *M. Schulz, E. Trischler, T. Sarfert.* IEEE Transactions on Computer Aided Design. Vol. 7, No. 1, pp. 126-137, 1988.

[25] On the Acceleration of Test Generation Algorithms. *H. Fujiwara, T. Shimono.* IEEE Transactions on computers Vol. C-32, pp. 1137-1144, December 1983.

[26] SAT and ATPG: Boolean engines for formal hardware verification. *Wolfgang Kunz, Armin Biere.* 2002 International Conference on Computer-Aided Design. ICCAD' 02.

[27] Combining strengths of circuit-based and CNF-based algorithms for high performance SAT-solver. *M. K. Ganai, L. Zhang, P. Ashar, A. Gupta and S. Malik.* In Proc. of Design Automation Conference (DAC) 2002.

[28] Bounded Model Checking using Satisfiability Solving. *E. Clarke, Armin Biere, R. Raimi, Y. Zhu.*

[29] Symbolic Model Checking without BDDs. *Armin Biere, A Cimatti, E. Clarke, Yunshan Zhu.* TACAS'99.

[30] Graph-Based Algorithms for Boolean Function Manipulation. *Randal Bryant*. IEEE transactions on computers C-35-8, pp. 677-691, August, 1986.

[31] SAT-based Induction for Temporal Safety Properties. *R. Armoni, L. Fix, R. Fraer, S. Huddleston, N. Piterman, M. Vardi*. Electronic notes in Theoretical Computer Science.

[32] Check Safety Properties using Induction and a SAT Solver. *M. Sheeran, S. Singh and G. Stalmarck.* In Proc. 3rd conference on Formal methods in Computer Aided Design. Lecture notes in Computer Science 2000 pp. 108-125.

[33] Using SAT based image computation for reachability analysis. *P. Chauhan, E. Clarke, D. Kroening.* CMU-CS-03-151 2003.

[34] Fast illegal state identification for improving SAT-based induction. *Vishnu Vimjam & Michael Hsiao.* Annual ACM IEEE Design Automation Conference. Proc. of the 43rd annual conference on Design Automation.

[35] SCOAP: Sandia Controllability/Observability analysis program. *H. Goldstein and E.L. Thigpen.* Proceedings of Design Automation Conference, 1980.

[36] Border-block Triangular Form and Conjunction Schedule in Image Computation. *In-Ho Moon, G. Hatchel and F. Somenzi.* Proc. of Conference on FMCAD, Nov 2000.

[37] Explicit Safety Property Strengthening in SAT-based Induction. *V. Vimjam, M. Hsiao.* 20th International Conference on VLSI Design, VLSID' 07.

[38] Automated abstraction refinement for model checking large state spaces using sat based conflict analysis. *Pankaj Chauhan, Edmund Clarke, James Kukula, Samir Sapra, Helmut Veith, Dong Wang*. In FMCAD' 02: Proceedings of the 4th international conference on Formal Methods in Computer-Aided Design. Pages 33-51, London UK 2002 Springer-Verlag.