Reachability Analysis of Large Sequential Circuits

Second Progress Seminar Report

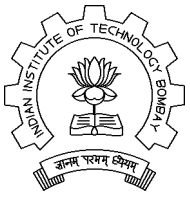
by

Seetha Jayasankar

Roll No. 04405003

under the guidance of

Prof. Supratik Chakraborty



Department of Computer Science and Engineering
Indian Institute of Technology, Bombay
October, 2006.

Acknowledgement

I would like to express my deep gratitude towards my guide, *Prof. Supartik Chakraborty*, for his valuable guidance and encouragement. I am also thankful to *Dina Thomas* for her prompt replies of the numerous email exchanges we shared. Special thanks to *Saurabh Joshi* for his cheerful company in the lab while conducting experiments. Discussions with him were motivating and really helped lift my spirits when things weren't quite working out. I would also like to thank *Gaurishankar Govindaraju* for providing us the partitions of ISCAS-89 benchmark suite.

Last, but not the least, I would also wish to thank all my friends for their company and help whenever I was in need for especially during this summer.

Seetha Jayasankar
October, 2006.

Contents

List of Figures

Abstract

Reachability analysis is a powerful technique to extract information from finite state machine models of large sequential circuits. Approaches based on explicit representation of the transition graph of the machine are limited to a great extent by the enormous size of the state space. Symbolic representation using Binary Decision Diagrams (BDDs) are typically used to overcome this state space explosion. But still, computing the real set of reached states is computationally expensive. Approximations, commonly in the form of abstraction are then introduced to model the circuits, such that either an over-approximated or an under-approximated reached set is computed. Different techniques for abstraction have been extensively studied over the years, wherein the aim is to keep this approximated set as close as possible to the real set of reached states.

Representation of such large circuits using partitioned transition relations (or functions) rather than as a monolithic transition relation helps immensely in alleviating the space explosion. The theory of reachability expressions provides a nice theoretical framework for analysing the performance of different schedules in an unpartitioned asynchronous system having partitioned transition relations.

In this work, we have built a BDD-based *generic framework* which extends the theory of reachability expressions to a partitioned state space by adding labels to expressions. In our framework, the state space is represented as a vector of BDDs, where an appropriate sequence of intersections gives the over-approximated reachable set. The framework allows us to encode different schedules and analyse their performance. In addition, we have also tried adding constraints on the Primary Inputs to tighten the over-approximation.

Exploiting structural and functional information about large circuits could also be helpful in arriving at better approximation methods. We note some observations regarding the we found in some of the large circuits taken from ISCAS-89 benchmark suite.

Chapter 1

Introduction

Our interest is to formulate a framework which allows us to perform reachability analysis of large sequential circuits having atleast 500 state elements. Such large circuits are common in the area of digital VLSI design, Determining reachability of large sequential circuits has been an area of active research. In general, the existing literature [1, 2, 3, 4] is mainly confined to relatively small sequential circuits having around 400 state elements and doesn't scale up well for the large circuits. Effectively then, attempting yet another technique for reachbility of circuits of smaller size is an *artificial problem*. This chapter presents a brief overview of sequential circuits, the problem of addressing reachability of large sequential circuits, related work and outline of our problem definition.

1.1 Modeling of Sequential Circuits for Reachability Analysis

1.1.1 Sequential Circuits

Sequential circuits are composed of a combinational logic block and a set of storage elements, most commonly flip-flops (henceforth called FFs), where the current output(s) depend on both the current input(s) and previous state values. Figure 1.1 depicts a generic model of a sequential circuit. **PI** refers to the *Primary Inputs* of the circuit, while **PO** refers to the Primary Outputs of the circuit. A hardware realization (starting from an all-zero initial state of the FFs) computes the output values from the current input values and the system state at every clock tick. Each FF can store one bit per clock tick. The current clock FF output is fed back into the circuit and is available for usage only at the ensuing clock tick. The state variables at a given clock tick are called *current state variables* (denoted as X), and those in the next clock tick are called *next state variables* (denoted as X').

1.1.2 Reachability Analysis

Given a gate level description of a sequential circuit, one can construct a state transition graph of the circuit. Formally, a state transition graph is a Mealy Machine $M = (Q, Q_0, R, \lambda, \mu)$ where
Q is the set of all states,
Q_0 is the set of intial system state,

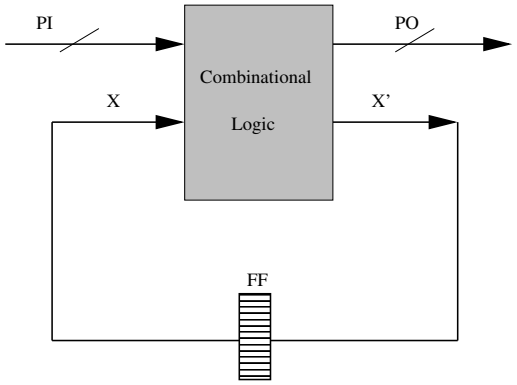


Figure 1.1: Generic Model of a Sequential Circuit

$R \subseteq Q \times Q$ is the set of edges,
$\lambda : Q \rightarrow \{0,1\}^n$ is the label for states,
$\mu : R \rightarrow 2^{\{0,1\}^{m+k}}$ is the label for the edges,
n is the number of state variables in the circuit, and
m and k are the number of PIs and POs respectively of the circuit. In the state transition graph representation, each edge is hence annotated with the format: $i_1 i_2 \ldots i_m / o_1 o_2, \ldots o_k$, and each vertex contains the current system state as $x_1 x_2 \ldots x_n$. The number of vertices in the transition graph is 2^n. More details about this can be found in first progress seminar report [5].

Given a state transition graph and an initial system state, the set of *reachable states* of the system, denoted rch_{Q_0} can be determined by traversing the state transition graph from the initial set of states Q_0, as shown
$\forall q \in Q_0, q \in rch_{Q_0} \wedge$
$\forall q \in rch_{Q_0}, q' \in Q \wedge (q,q') \in R \Rightarrow q' \in rch_{Q_0}$.
Reachability analysis is then posed as a decision problem to determine if a designated set of target states is reachable.

1.2 Background and Related Work

Explicit representation of the system as a transition graph may not be feasible for practical circuits. An alternate representation often used is a *Boolean Function*. Boolean function representations of the initial and target states are also termed as *characteristic functions*. Consider a circuit, where I is a vector of primary inputs, X is a vector of the current state variables and X' is a vector of the next state variables. A Boolean function on I, X and X' can represent the transitions in the transition graph and is called the *transition relation*.

A convenient and powerful data structure to represent Boolean functions is a *Binary Decision Diagram* (BDD) [6]. A Boolean formula on a set of state variables evaluates to either True or False for valuations of the state variables. Let the set of states be $S(X)$ and the transition relation be $T(I,X,X')$, then the next state computation is then made as

$$S'(X') = \exists_{x \in X, i \in I}[S(X) \wedge T(I,X,X')]$$

which is actually a series of nested quantifications, one for each variable in X. This is also termed as relational product for image computation. Computing this efficiently is crucial to

reachability analysis. Unfortunately it is often the case that the BDD for $T(I,X,X')$ is very huge. With repeated image computations, the entire set of states which can be encountered from a set of initial states can be computed. Symbolic Model Checking (SMC) can be done by using such an implicit representation using Boolean functions. Extensive research in SMC using BDDs has been done and several heuristics have been developed that have improved the size of systems that can be formally verified. Bryant [6] gives a number of algorithms to perform logical operations using BDDs. The complexity of these operations have been discussed in the paper [6]. Among the operations, image computation has been found to be expensive with complexity exponential in the number of variables being quantified. The bottleneck of existential quantification has been studied by Trivedi *et al.* [7] and Joshi *et al.* in [8].

1.2.1 Approximation by Partitioning state space

Approximate traversal algorithms take a partitioned state space, traverse each sub-space separately to compute the set of reachable states in its corresponding state sub-space, and the state variables belonging to other partitions, but needed to represent the next state function for the partition being traversed, can be treated as primary inputs. Hence, the set of reachable states computed by these algorithms is an over-estimation of the set of real reachable states in the unpartitioned system, guaranteeing that all the real reachable states are included in the computed super-set of reachable states [2]. The accuracy of the approximate traversals also depends the partitioning strategy.

Cho *et al.* in [2] present an algorithm to automatically determine a good partition of the state space a metric called *latch affinity*. It is argued that both latch connectivity and latch correlation (determined by circuit analysis) are crucial and hence a convex combination of both these parameters is used to compute the latch affinity. Based on this measure, a latch affinity graph is constructed. Then, an initial partition is computed by determining the connected components from this graph. Each connected component forms one sub-FSM and any sub-FSM exceeding a predefined bound is further partitioned into smaller sub-FSMs. Several seeding/clustering techniques are experimented with.

Govindaraju *et al.* in [9] have shown that instead of having disjoint partitions of the state space as was proposed by Cho *et al.* in [1, 2], it greatly helps if overlapping partitions are used as it leads to tighter approximations. The reason is that if there is any interaction among the constituent sub-machines of the state space, with disjoint partitions either the two sub-machines would have to be included in a single projection or the interaction will have to be ignored in which there are greater degrees of freedom which reduce the quality of the approximation. In contrast by using overlapping projections, each such interaction can be modeled as a sub-machine and achieve tighter approximations. A more general constrain operator, termed *multiple constrain* is also introduced in [9] which allows for overlaping support-sets.

Govindaraju *et al.* in [10] propose addition of a set of auxillary state variables to the state space which is chosen by studying the circuit structure such that this variables captures important properties of many state variables into this one single bit. To choose the set of auxillary variables, internal wires with high fainin and high fanout such that their fanin cone only includes state variables become ideal candidates. The results show at least an order of magnitude improvement over their earlier results published in Govindaraju *et al.* in [9].

1.2.1.1 Image Computation for overlapping projections

Instead of using transition relations for image computation, Coudert and Madre [4] have proposed a way of computing the image of a set by manipulating the vector of next state functions **T**. Henceforth, our notation of $Img(S,T)$ denotes the BDD for image of a vector of Boolean functions T: $[t_1,t_2,\ldots,t_k]$, when the domain is restricted to the set represented by S. Each individual function t_i is represented by a BDD $t_i(X)$. Coudert and Madre proposed two algorithms to compute the BDD for $Img(S,T)$ without computing the transition relation. Both algorithms are based on the *constrain* operator, denoted as $\downarrow$ defined by them in [4].

The notations used here are borrowed from Govindaraju *et al.* in [9]. Let $X = \{x_1,\ldots,x_n\}$ be the set of current state variables, $X' = \{x'_1,\ldots,x'_n\}$ be the set of next state variables, and I be the set of PIs, $Init(X)$ be the predicate for the initial system state, and $S(X)$ be a predicate with support in X, the image of $S(X)$ under the next state transition function $T(I,X)$ can be computed as

$$Img(S(X),T(I,X)) = \lambda X'.\exists I,X.(X' \leftrightarrow T(I,X)) \wedge S(X)$$

Img computes a predicate with support X', which evaluates to 1 *iff* X' is in image of S under T. Also, the set of reachable states of the circuit would be computed by a least fixpoint iteration as

$$\text{Reachable} = \text{lfp } S.\ \lambda X.\ (Init(X) \vee Img(S(X),\ T(I,X)))$$

Suppose the state space is decomposed into a collection of k (not necessarily disjoint) subsets, $Y = \{y_1,\ y_2,\ \ldots,\ y_k\}$ which may be overlapping. We can define two operations: *approximation* and *concretization* [9], denoted α and γ respectively [1].

$$\alpha_j(S(X)) = \lambda X.\ \exists_{z \in X - y_j} S(X)$$

projects $S(X)$ onto the variables present in y_j.
The approximation operator α operates as

$$\alpha(S(X)) = (\alpha_1(S(X)),\alpha_2(S(X)),\ldots,\alpha_k(S(X)))$$

and would return a k-tuple of the state space.
The concretization operator γ conjoins the collection of projections as shown

$$\gamma(S_1,S_2,\ldots,S_k) = \bigwedge_{i=1}^{i=k} S_i$$

The approximation operator α, trades size for accuracy, while the concretization operator γ can result in a bigger BDD, so we would like to avoid computing $\gamma(S_1,S_2,\ldots,S_k)$ explicitly. We need to compute the image of an implicit conjunction as an implicit conjunction of the elements of a k-tuple.
Let $Img_{ap}(S,T) = (R_1,R_2,\ldots,R_k) = \alpha(Img(\gamma(S),T(I,X)))$ compute the approximate projected version of the image. Now, an overapproximation, $Reachable_{ap}(Init)$ of the reachable states would be

[1] α and γ are the same as *abstraction* and *concretization* functions respectively used in the theory of abstract interpretation for describing a Galois connection.

$$Reachable_{ap} = \text{lfp S. } (\alpha(Init) \sqcup Img_{ap}(S,T))$$

where $\sqcup$ performs pairwise union of the k-tuple where *lfp* is the least fixed point iteration which starts with **S** = (0, 0, ..., 0), and in each iteration *joins* the current approximate set with the approximate successor set. Finally upon reaching convergence, it returns a tuple **S** to $Reachable_{ap}$. The overapproximate reachable states set is the *implicit* conjunction of $\gamma(Reachable_{ap}(Init))$. This set is an overapproximation because image at every iteration of the least fixed point routine is an overapproximation. We would get the exact results in the special case when there is just one subset, y_1 = X, in the collection Y.

Coudert and Madre in [4] have shown how to compute the image of a Boolean function vector using the *generalized cofactor* or *constrain* operator, denoted as $\downarrow$. The *constrain* operator allows one to cofactor a function f, with respect to another function c, and reduces to ordinary cofactor when c is a single variable. Intutively, given two Boolean functions f and c over n bit variables, $(f \downarrow c)(x)$ evaluates to the valuation of $f(x)$ if $c(x)$ is true. Else, it performs minimum number of flipping of bit values (in a fixed and specified variable order) to map x to a new x', such that $c(x')$ is true and returns the valuation of $f(x')$.

For example, consider a Boolean space over three bit variables p, q, and r and a fixed variable ordering $p < q < r$.

Example 1: Let f=$p \vee r$ and c=q, then $f \downarrow c = f = p \vee r$.

Example 2: Suppose f=$q \wedge r$ and c=$\overline{p} \vee (p \wedge q \wedge r)$, then $f \downarrow c = p \vee (\overline{p} \wedge q \wedge r)$.

Let $R_j = Img(\gamma(S, \alpha_j(T(I,X))))$ be the set of predicates determining the next state for bits in the projections y_j. As computing the explicit conjunction of $\gamma(S)$ could result in a huge BDD, it is tempting to do a serial constrain which involves computing $\alpha_j(T) \downarrow S_1 \downarrow S_2 \ldots \downarrow S_p$. This works well if the supports of S_i's are disjoint. McMillan has shown [11] that if g and h have independent support, then $f \downarrow (g \wedge h) = (f \downarrow g) \downarrow h$. This method won't work for overlapping subsets. Govindaraju *et al.* [9] proposed the notion of *multiple constrain* which computes $f \downarrow (g \wedge h) = (f \downarrow h) \downarrow (g \downarrow h)$. It is also shown how this can be extended to constraining a Boolean function with a vector of Boolean functions.

1.2.2 Other BDD based approximation schemes

Hong *et al.* in [12] propose the usage of *sibling substitution* and node sharing which allows for selectively minimizing sub-BDDs while the traditional BDD minimization techniques would look to blindly minimize all sub BDDs. Cabodi *et al.* in [13] introduce a new operator called *existential cofactor* which allows to distribute quantification over conjunction and hence permits selective cofactoring. This helps in reduction of peak memory usage and lowers the overall memory consumption also. Ranjan *et al.* in [14] propose several BDD variable reordering heuristics, use of partitioning the state space, and some heuristics on choosing ordering of clusters which minimizes peak memory usage.

Ravi *et al.* in [15] use a metric based on circuit analysis to decide whether to use implicit conjunction or to split based on domain/codomain variable. They propose an algorithm based on this dynamic collection of statistics of the behavior of BDD nodes. Cabodi *et al.* in [16] propose another method for gathering usage statistics for each BDD node and record the involvement of each node during recursive calls, cache hits etc.

Ravi *et al.* in [17] apply hints by constraining the transition relation of the system to be verified. They specify possible values for (subsets of) the primary inputs and state variables.

Once all states reachable following the hints have been visited, the system is unconstrained and reachability analysis proceeds on the original system. Given enough resources, the algorithm will therefore search all reachable states, unless a state that violates the invariant is found. Deriving the hints requires some knowledge of the circuit organization and behavior. There is no guarantee, however, that a hint will be beneficial.

High-density reachability analysis by Ravi *et al.* in [18] is based on the observation that BDDs are most effective at representing sets of states when their *density* is high. The *density* of a BDD is defined as the ratio of the number of minterms of the function to the number of nodes of the BDD. Their approach tries to direct the exploration of the state space so that the state sets have dense BDDs. Dense sets are obtained by applying subsetting algorithms in [18] to the frontier states when the BDDs grow large. High density reachability is a fully automated approach, which can be applied regardless of the knowledge of the system, but occasionally exhibits erratic behavior. Also, it addresses directly the density of sets of states, but only indirectly the difficulties of image computation that may be connected to the representation of transitions. (When the intermediate products of image computation grow too large, they are subsetted. This leads to the computation of partial images in [18].)

Cabodi *et al.* in [19] introduce the notion of λ-latches. These are latches on which no other state variables depend; they are constrained at their initial value until no new states are achieved. Guided traversal subsumes this technique.

1.2.3 SAT based approximation techniques

SAT based SAT solvers enjoy several properties which make them attractive as a complement to BDDs in SMC. Their performance is less sensitive to the size of the formulas. Typical SAT solvers do not suffer from state explosion problem, and do not require an external variable ordering to be supplied. Abdulla *et al* in [20] have shown how the strength of SAT solving procedures can be exploited to increase the class of systems that can be verified using traditional SMC methods. Representation of circuits is done using Reduced Boolean Circuits and SATO is used as the back-end SAT solver. They have a tool FixIT which implements these ideas.

In Cabodi *et al.* [21], the central idea is to complement the initial over-approximate BDD information with a final SAT-solver search, using BDDs to prune and focus the search. In the first phase, they compute (in the forward and/or backward directions) an over-approximate estimate of the traces connecting the initial state set to the target one. Then the estimate is combined, as an additional constraint, with the Bounded Model Check problem, to be solved by a SAT tool. Their target is to obtain an efficient pruning of the SAT solver search space, which somehow mimics the contribution of *conflict clauses*, generated by means of *conflict analysis* in state of the art SAT tools. Each new conflict clause specifies a subset of the state space in which there exist no solution. Also, the over-approximate knowledge of reachable states restricts the SAT solver state space. This information is presented as an initial pre-processing or learning phase. They also introduce a set of strategies to store a BDD (in a monolithic or conjoined form) as a CNF formula.

1.3 Problem Statement

Our interest lies in addressing the reachability problem for large sequential circuits, typically having more than 500 FFs. We have already noted [5] that exact methods do give some computational efficiency, but they are not very significant for handling large circuits. Our current focus is to develop a *generic BDD-based framework* which help us to analyse different approximate traversal methods. Our framework is built on top of NuSMVDP [22].

1.4 Outline of report

In Chapter 2, we present labeled reachability expressions as a convenient way to manipulate a partitioned state space.
Chapter 3 discusses our generic framework, encoding of a few state of the art traversal methods in our framework and some results on large circuits and compare it with already reported results.
In Chapter 4, we try out another approximation technique called *Input Constraining* which helps in tightening the approximations using our framework. We report orders of magnitude improvement using this approximation method.
Chapter 5 presents our conclusions and future research plans.

Chapter 2

Labeled Reachability Expressions

The theory of Reachability Expressions [23] provides a framework for analyzing the performance of different schedules in an asynchronous system having partitioned transition relations and unpartitioned state space. Traversing large state spaces benefits from partitioning the state space of the system [1, 2, 9]. We extend the theoretical framework of reachability expressions to allow manipulation with a vector of BDDs (henceforth called a *State Vector*), one representing each partition of the state space, and add annotations in the form of labels to the expressions to tag expressions to the relevant component of the state vector. Two new operators are also added to help formulate the different traversal algorithms as an encoding of these labeled expressions. It is also noted that this can be easily extended to use with the theory of Reachability Matrices [24] as well.

2.1 Background

A partitioned state transition system can be represented as a conjunctive decomposition of (partitioned)transition relations. Following the notation introduced in Chapter 1 in Section 1.2.1.1, given k projections of the transition system, every step of image computation in such a partitioned state system can be written as
$S'(X') = \exists_{I,X}\{T_1(I,X_1,X'_1) \wedge \ldots \wedge T_k(I,X_k,X'_k) \wedge S(X)\}$ where X_k and X'_k denote the set of current state variables and next state variables belonging to partitioned transition relation T_k respectively. Since this is a conjunctively decomposed representation, if a subformula doean't depend on any of the variables being quantified, it can be moved out of the scope of the quantification using the idea of *early quantification* in [3], we can compute this expression as a conjunction of image computations of $S(X)$ under each partitioned transition relation. In other words, we can conjunct each $T_i(I,X_i,X'_i)$ with $S(X)$ one at a time and quantify out a variable x_l when none of the remaining $T_j(I,X_j,X'_j)$ depend on x_l. The order in which the T_i's are conjuncted is very significant and affects the sizes of the BDDs generated to a great extent. We refer to this formulation of image computation as the *vanilla* method of image computation and refer to this order of conjunction as the *cluster order* in this work.

If in addition, the state space is also projected into k components, the image computation can be written as
$S'_j(X'_j) = \exists_{I,\, X-X_j}\{T_1(I,X_1,X'_1) \wedge \ldots T_j(I,X_j,X'_j) \wedge \ldots \wedge T_k(I,X_k,X'_k) \wedge S_1(X_1) \wedge S_2(X_2) \wedge \ldots \wedge S_k(X_k)\}$.
And

$S'(X) = \gamma(S'_1(X'_1),\ \ldots,\ S'_k(X'_k))$ So, we need a mechanism where we can apply any arbitrary transition relation to compute the image of any arbitrary set of states.

2.1.1 Reachability Expressions

The theory of Reachability Expressions [23] was developed as a framework for asynchronous systems where it provided the ease of specifying different schedules and evaluate their performance. We present the grammar of Reachability Expressions here.

2.1.1.1 Syntax

Given a set of states and a partitioned transition relation representation, syntactically a Reachability Expression over such a partitioned transition system can be obtained from the following grammar:
$\mathsf{E} \rightarrow \mathsf{E} + \mathsf{E} \mid \mathsf{E}; \mathsf{E} \mid \mathsf{E} \circ \mathsf{E} \mid (\mathsf{E}) \mid *\mathsf{E} \mid T_1 \mid \ldots \mid T_k$
where T_i indicates the operation of image computation of the set of states (S) under the transition relation T_i and is denoted as $Img(S, T_i)$.

2.1.1.2 Semantics

The notion of evaluating reachability expressions can be formalized by defining their semantics. The semantics of a reachability expression is defined with respect to an underlying state transition system, and is naturally described as a mapping from sets of states to sets of states. Let σ be a reachability expression over a partitioned transition system where Q is the set of states. The semantics of σ is a mapping from 2^Q to 2^Q and is denoted by $[\![\sigma]\!]$. Define δ to be the *identity* transition relation, i.e. $[\![\delta]\!](\mathbf{S}) = \mathbf{S}$ and Θ to be the *null* transition relation, i.e. $[\![\Theta]\!](\mathbf{S}) = \phi$. We give below an inductive definition of $[\![\sigma]\!](S)$ for $S \subseteq Q$.

- $[\![\mathbf{T_i}]\!](S) = Img(S, T_i)$.
- $[\![\sigma_1 + \sigma_2]\!](S) = [\![\sigma_1]\!](S) \cup [\![\sigma_2]\!](S)$
- $[\![\sigma_1 \circ \sigma_2]\!](S) = [\![\sigma_2]\!]([\![\sigma_1]\!](S))$
- $[\![\sigma_1 ; \sigma_2]\!](S) = [\![(\sigma_1 + \delta) \circ (\sigma_2 + \delta)]\!](S)$.
- $[\![(\sigma)]\!](S) = [\![\sigma]\!](S)$
- $(\sigma)^0 = \delta$ for any expression σ.
- $(\sigma)^i = \sigma \circ (\sigma)^{i-1}$, for all $i \geq 1$.
- $[\![*\sigma]\!](S) = \bigcup_{i=0}^{\infty} [\![(\sigma)^i]\!](S)$.

2.1.1.3 Examples

Consider a set of states **S**. The following examples show the evaluation of a Reachability Expression σ on **S**.

- **Example 1**: $\sigma = T_1 \ + \ T_2$
  By the definition of the semantics, $[\![T_1 \ + \ T_2]\!](\mathbf{S}) = Img(\mathbf{S}, T_1) \cup Img(\mathbf{S}, T_2)$
- **Example 2**: $\sigma = (T_1 \ + \ T_2) \circ T_3$
  By the definition of the semantics, $[\![(T_1 \ + \ T_2) \circ T_3]\!](\mathbf{S}) = [\![T_3]\!]([\![(T_1 \ + \ T_2)]\!](\mathbf{S})) = Img(Img(\mathbf{S}, T_1) \cup Img(\mathbf{S}, T_2), T_3)$
- **Example 3**: $\sigma = *T_1 \circ T_2$
  By the definition of the semantics, $[\![*T_1 \circ T_2]\!](\mathbf{S}) = [\![T_2]\!]([\![*T_1]\!](\mathbf{S})) = Img(\bigcup_{i=0}^{\infty}((Img(S, T_1))^i), T_2)$.

Applying arbitrary transition relations on arbitrary sets od states is not possible in the framework of Reachability Expressions. But this is one feature which is central to the idea of approximations by partitioning the state space. Hence we need an extension of this framework which would allow us to express such things. Specifically, we would need mechanisms which would help in computing applications of arbitrary expressions over an arbitrary set of states. In the next section, we discuss the use of *labels* which would allow us to specify the set of states on which we want to apply an arbitrary transition relation.

2.2 Labels and Reachability Expressions

Given a partitioned transition system with projected state space, it is necessary for any framework to allow specification of how to apply arbitrary transition relations on arbitrary sets of states. We formulate this using the use of *labels* which would be used to tag expressions to signify which set of states they apply to. Also, we would also need to store the result of computation of some arbitrary expression and possibly use it in some other expression evaluation.

Syntactically, a Labeled Reachability Expression over a partitioned state transition system with partitioned state space is a terminal string obtained from the following grammar:
$\mathsf{LE} \ \rightarrow \ \mathsf{LE} + \mathsf{LE} \ | \mathsf{LE}; \mathsf{LE} \ | \mathsf{LE} \circ \mathsf{LE} \ | \mathsf{LE} \wedge \mathsf{LE} \ | \mathsf{LE} \downarrow \mathsf{LE} \ | \mathsf{L} : \mathsf{LE} \ | (\mathsf{LE})_{\mathsf{L}} \ | (\mathsf{LE}) \ | *\mathsf{LE} \ | T_1 \ | \ \ldots | \ T_k \ | \delta \ | \Theta$
$\mathsf{L} \ \rightarrow \ L_1 \ | \ L_2 \ | \ \ldots, \ | \ L_n$
where T_i's denote the application of the *Img* operator, i.e. evaluate the image of the state component, $S_k(X_k)$ under the transition relation $T_i(I, X_i, X_i')$, and is denoted as $Img(S_k(X_k), T_i(I, X_i, X_i'))$. Define Θ to be the *null* transition relation, i.e., $T(I, X, X') = 0$ and δ to be the *identity* transition relation, i.e. $T(I, X, X') = (x_1 \leftrightarrow x_1') \wedge (x_2 \leftrightarrow x_2') \wedge \ \ldots \ \wedge (x_n \leftrightarrow x_n'))$.
Note that in our current work, we do not use expressions derived from $\mathsf{LE} \ ; \ \mathsf{LE}$. But, it is retained in the syntax for the purpose of ensuring that Labeled Reachability Expressions subsumes Reachability Expressions.

A Labeled Reachability Expression, λ is defined over LE and L. Each Labeled Reachability Expression can be viewed as a *predicate transformer* of a state vector to another state vector.

We now present the denotational semantics of λ which is a mapping from $\underbrace{2^Q \times 2^Q \times \ \ldots \times 2^Q}_{n}$ to $\underbrace{2^Q \times 2^Q \times \ \ldots \times 2^Q}_{n}$ where n is the number of state components (or labels) and is denoted by

$[\![\lambda]\!]$. Note that when n=1, this reduces to being the same as grammar for Reachability Expressions without the use of the operators $\wedge$ and $\downarrow$. We also list how to operationally compute the semantic value defined by the denotational semantics.

We denote the *State Vector* as $\vec{\mathbf{S}}$ with n components, such that S_i represents the set of states for component labeled L_i. We present the definition of $[\![\lambda]\!](\vec{\mathbf{S}})$.
Define $\sqcup$ to be the operator which performs component-wise union. In other words, given two state vectors $\vec{\mathbf{S_1}}$ and $\vec{\mathbf{S_2}}$ of length n, define a new state vector (also of length n) $\vec{\mathbf{S_3}} = \vec{\mathbf{S_2}} \sqcup \vec{\mathbf{S_2}}$ as $\forall i, S_{3i} = S_{1i} \cup S_{2i}$.

Define $\sqcap$ to be the operator which performs component-wise intersection. In other words, given two state vectors $\vec{\mathbf{S_1}}$ and $\vec{\mathbf{S_2}}$ of length n, define a new state vector (also of length n) $\vec{\mathbf{S_3}} = \vec{\mathbf{S_2}} \sqcap \vec{\mathbf{S_2}}$ as $\forall i, S_{3i} = S_{1i} \cap S_{2i}$.

Define $\downarrow$ to be the operator which performs component-wise constraining. In other words, given two state vectors $\vec{\mathbf{S_1}}$ and $\vec{\mathbf{S_2}}$ of length n, define a new state vector (also of length n) $\vec{\mathbf{S_3}} = \vec{\mathbf{S_2}} \downarrow \vec{\mathbf{S_2}}$ as $\forall i, S_{3i} = S_{1i} \downarrow S_{2i}$.

- $[\![\mathbf{T_1}]\!](\vec{\mathbf{S}}) = (Im(S_1, T_1), \ldots, S_n)$.
  This computes the image of the first component S_0 under the transition relation T_1 and stores the result in S_0, leaving the rest of the state vector unchanged. The operational procedure for this is

  1. Extract the state component S_1
  2. Perform image computation of S_1 under T_1
  3. Return a new state vector with the first component being the result of image computation of S_1 under T_1 and the rest of the components remain unchanged

- $[\![\lambda_1 + \lambda_2]\!](\vec{\mathbf{S}}) = [\![\lambda_1]\!](\vec{\mathbf{S}}) \sqcup [\![\lambda_2]\!](\vec{\mathbf{S}})$
  This computes the component wise union of the two state vectors. The operational procedure for this is

  1. Perform $[\![\lambda_1]\!](\vec{\mathbf{S}})$ and $[\![\lambda_2]\!](\vec{\mathbf{S}})$, and store the result in new state vectors $\vec{\mathbf{S'_1}}$ and $\vec{\mathbf{S'_2}}$ respectively.
  2. Perform component wise union of the components of $\vec{\mathbf{S'_1}}$ and $\vec{\mathbf{S'_2}}$ and store in a new state vector $\vec{\mathbf{S'_3}}$, where $S'_{3i} = S'_{1i} \cup S'_{2i}$
  3. Return the state vector $\vec{\mathbf{S'_3}}$

- $[\![\lambda_1 \wedge \lambda_2]\!](\vec{\mathbf{S}}) = [\![\lambda_1]\!](\vec{\mathbf{S}}) \sqcap [\![\lambda_2]\!](\vec{\mathbf{S}})$
  This computes the component wise intersection of the two state vectors. The operational procedure for this is

  1. Perform $[\![\lambda_1]\!](\vec{\mathbf{S}})$ and $[\![\lambda_2]\!](\vec{\mathbf{S}})$, and store the result in new state vectors $\vec{\mathbf{S'_1}}$ and $\vec{\mathbf{S'_2}}$ respectively.
  2. Perform component wise intersection of the components of $\vec{\mathbf{S'_1}}$ and $\vec{\mathbf{S'_2}}$ and store in a new state vector $\vec{\mathbf{S'_3}}$, where $S'_{3i} = S'_{1i} \cap S'_{2i}$
  3. Return the state vector $\vec{\mathbf{S'_3}}$

- $[\![\lambda_1 \circ \lambda_2]\!](\vec{\mathbf{S}}) = [\![\lambda_2]\!]([\![\lambda_1]\!](\vec{\mathbf{S}}))$
  This evaluates λ_2 on the state vector that is computed by the application of λ_1 on $\vec{\mathbf{S}}$. The operational procedure for this is

  1. Perform $[\![\lambda_1]\!](\vec{\mathbf{S}})$ and store the result in a new state vector $\vec{\mathbf{S}'_1}$.
  2. Perform $[\![\lambda_2]\!](\vec{\mathbf{S}'_1})$ and store the result in a new state vector $\vec{\mathbf{S}'_2}$
  3. Return the state vector $\vec{\mathbf{S}'_2}$

- $[\![\lambda_1 ; \lambda_2]\!](\vec{\mathbf{S}}) = [\![(\lambda_1 + \delta) \circ (\lambda_2 + \delta)]\!](\vec{\mathbf{S}})$.
  This is a derived operator which evaluates $\lambda_2 + \delta$ on $\lambda_1 + \delta$ over the state vector $\vec{\mathbf{S}}$. The operational procedure for this is

  1. Perform $[\![\lambda_2 + \delta]\!](\vec{\mathbf{S}})$ and store the result in a new state vector $\vec{\mathbf{S}'_1}$.
  2. Perform $[\![\lambda_1 + \delta]\!](\vec{\mathbf{S}'_1})$ and store the result in a new state vector $\vec{\mathbf{S}'_2}$
  3. Return the state vector $\vec{\mathbf{S}'_2}$

- $[\![(\lambda)]\!](\vec{\mathbf{S}}) = [\![\lambda]\!](\vec{\mathbf{S}})$
  The operational procedure for this is the same as that of $[\![\lambda]\!](\vec{\mathbf{S}})$.

- $[\![\lambda_1 \downarrow \lambda_2]\!](\vec{\mathbf{S}}) = [\![\lambda_1]\!](\vec{\mathbf{S}}) \downarrow [\![\lambda_2]\!](\vec{\mathbf{S}})$
  This computes the component wise constraining of the state vectors. The operational procedure for this is

  1. Perform $[\![\lambda_1]\!](\vec{\mathbf{S}})$ and store the result in a new state vector $\vec{\mathbf{S}'_1}$.
  2. Perform $[\![\lambda_2]\!](\vec{\mathbf{S}})$ and store the result in a new state vector $\vec{\mathbf{S}'_2}$
  3. Compute and return the state vector $\vec{\mathbf{S}'_3}$, where $S'_{3_i} = S'_{1i} \downarrow S'_{2i}$

- $[\![*\lambda]\!](\vec{\mathbf{S}}) = \bigsqcup_{i=0}^{\infty} [\![(\lambda)^i]\!](\vec{\mathbf{S}})$.
  We define $\lambda^0 = \delta$ and $\forall\, i \geq 0, \lambda^i = \lambda \circ \lambda^{i-1}$.

- $[\![L_i : \lambda]\!](\vec{\mathbf{S}}) = \vec{\mathbf{S}'}$ *where* $\vec{S'}_i = ([\![\lambda]\!](\vec{\mathbf{S}}))_1$, *and* $\vec{S'}_j = ([\![\lambda]\!](\vec{S}))_j,\ j \in \{1,2,\ldots,n\},\ j \neq i$
  This allows storing the result of the evaluation of $[\![\lambda]\!](\vec{S})_1$ in the state component labeled using L_i. The operational procedure for this is

  1. Perform $[\![\lambda]\!](\vec{\mathbf{S}})$ and store the result in a new state vector $\vec{\mathbf{S}'}$.
  2. Copy the contents of $\vec{\mathbf{S}'}_1$ to the state component $\vec{\mathbf{S}'}_i$
  3. Return the state vector $\vec{\mathbf{S}'}$

- $[\![(\lambda)_{L_i}]\!](\vec{\mathbf{S}}) = [\![\lambda]\!](\vec{\mathbf{S}'})$, *where* $S'_1 = S_i$, *and* $S'_j = S_j\ \forall j \in \{2,\ldots,n\}$
  This permits applying arbitrary transition relations on arbitrary sets of states by allowing a way to specify which state vectoer component is to be considered for evaluating the expression. The operational procedure for this is

1. Copy the contents of the state vector $\vec{\mathbf{S}}$ to a new state vector $\vec{\mathbf{S}}'$
2. Copy the contents of S'_i into S'_1, i.e. changing the sets of states on which the expression λ is to be evaluated
3. Evaluate $[\![\lambda]\!](\vec{\mathbf{S}}')$ to get a new state vector $\vec{\mathbf{S}}''$. Return the state vector $\vec{\mathbf{S}}''$.

2.2.1 Examples of Labeled Reachability Expressions

Given a Labeled Reachability Expression λ defined over LE and L, we present a few examples here. Assume for sake of simplicity that the state vector $\vec{\mathbf{S}}$ has three components $(S_1,\ S_2,\ S_3)$.

- **Example 1**: If $\lambda = L_3 : \delta_{L_2}$
  By definition of the semantics and the operational procedure, this is evaluated as follows:

  1. Evaluate $[\![\delta_{L_2}]\!](S_1,\ S_2,\ S_3)$. This would return a new state vector $\vec{\mathbf{R}} = (S_2,\ S_2,\ S_3)$
  2. Since the label for the expression is L_3, next copy the contents of R_1 into R_3. So, the state vector now becomes $\vec{\mathbf{R}} = (S_2,\ S_2,\ S_2)$
  3. Return the state vector $\vec{\mathbf{R}}$.

  Hence, $[\![L_3 : \delta_{L_2}]\!](S_1,\ S_2,\ S_3) = (S_2,\ S_2,\ S_2)$. This shows how the labels can be used to update or change different state components.

- **Example 2**: If $\lambda = L_1 :\ (T_1)_{L_2}\ +\ L_2 :\ (T_2)_{L_1}$
  By definition of the semantics and the operational procedure, this is evaluated as follows:

  1. Evaluate $[\![L_1 :\ (T_1)_{L_2}]\!](S_1,\ S_2,\ S_3)$. This would return a state vector $\vec{\mathbf{R}} = (Img(S_2, T_1), S_2, S_3)$
  2. Evaluate $[\![L_2 :\ (T_2)_{L_1}]\!](S_1,\ S_2,\ S_3)$. This would return a state vector $\vec{\mathbf{Q}} = (Img(S_1, T_2), Img(S_1, T_2), S_3)$
  3. Compute and return the component-wise union of $\vec{\mathbf{R}}$ and $\vec{\mathbf{S}}$, i.e. $\vec{\mathbf{R}} \sqcup \vec{\mathbf{Q}} = (Img(S_2, T_1) \cup Img(S_1, T_2), S_2 \cup Img(S_1, T_2), S_3)$

  This example demonstrates the capability that Labeled Reachability Expressions provide as it allows us to apply the transition relation T_1 on S_2 and T_2 on S_1.

- **Example 3**: If $\lambda = L_1 :\ ((T_1)_{L_2}\ \downarrow\ \delta_{L_3})$
  By definition of the semantics and the operational procedure, this is evaluated as follows:

  1. Evaluate $[\![(T_1)_{L_2}\ \downarrow\ \delta_{L_3}]\!](S_1,\ S_2,\ S_3)$. This would return a state vector $\vec{\mathbf{R}} = (Img(S_2, T_1), S_2, S_3)\ \downarrow\ (S_3, S_2, S_3) =$ $(Img(S_2, T_1)\ \downarrow\ S_3, S_2\ \downarrow\ S_2, S_3\ \downarrow\ S_3) = (Img(S_2, T_1)\ \downarrow\ S_3, 1, 1)$
  2. As the label for the expression is L_1, copy contents of R_1 into R_1. This does not change the vector $\vec{\mathbf{R}}$
  3. Return the state vector $(Img(S_2, T_1)\ \downarrow\ S_3, 1, 1)$

  This example again demostrates a possible application of constraining and using the transition relation to find the image computation of an arbitrary set of states.

- **Example 4**: If $\lambda = (T_1 \wedge T_2)_{L_2}$
  By definition of the semantics and the operational procedure, this is evaluated as follows:

  1. Evaluate $[\![(T_1 \wedge T_2)_{L_2}]\!](S_1,S_2,S_3) = [\![(T_1 \wedge T_2)]\!](S_2,S_2,S_3)$.
  2. Evaluate $[\![T_1]\!](S_2,S_2,S_3) \sqcap [\![T_2]\!](S_2,S_2,S_3) = (Img(S_2,T_1),S_2,S_3) \sqcap (Img(S_2,T_2),S_2,S_3)$ $= (Img(S_2,T_1) \cap (Img(S_2,T_2)),S_2 \cap S_2,S_3 \cap S_3) = (Img(S_2,T_1) \cap (Img(S_2,T_2)),S_2,S_3)$.
  3. Return the state vector $(Img(S_2,T_1) \cap (Img(S_2,T_2)),S_2,S_3)$.

2.3 Properties

We list a few properties of Labeled Reachability Expressions with their proofs.

Let λ, λ_1, λ_2, and λ_3 be labeled reachability expressions defined over LE and L. Let $\vec{\mathbf{S}}$, $\vec{\mathbf{R}}$ and $\vec{\mathbf{Q}}$ be state vectors, and $[\![\lambda_1]\!](\vec{\mathbf{S}}) = \vec{\mathbf{R}}$, and $[\![\lambda_2]\!](\vec{\mathbf{S}}) = \vec{\mathbf{Q}}$. We say that $\lambda_1 = \lambda_2$ *iff* $\forall i \in \{1,\ldots,n\}$, $R_i \subseteq Q_i$ and $Q_i \subseteq R_i$.

We have the following properties

1. $\lambda + \lambda = \lambda$
   Let $\vec{\mathbf{S}}$ be any state vector. By definition $[\![\lambda + \lambda]\!](\vec{\mathbf{S}}) = [\![\lambda]\!](\vec{\mathbf{S}}) \sqcup [\![\lambda]\!](\vec{\mathbf{S}}) = [\![\lambda]\!](\vec{\mathbf{S}})$. Hence, $\lambda + \lambda = \lambda$. This is the *idempotent* property.

2. $\lambda_1 + \lambda_2 = \lambda_2 + \lambda_1$
   Let $\vec{\mathbf{S}}$ be any state vector. By definition $[\![\lambda_1 + \lambda_2]\!](\vec{\mathbf{S}}) = [\![\lambda_1]\!](\vec{\mathbf{S}}) \sqcup [\![\lambda_2]\!](\vec{\mathbf{S}})$. Due to commutativity of set union, $[\![\lambda_1]\!](\vec{\mathbf{S}}) \sqcup [\![\lambda_2]\!](\vec{\mathbf{S}}) = [\![\lambda_2]\!](\vec{\mathbf{S}}) \sqcup [\![\lambda_1]\!](\vec{\mathbf{S}})$. Hence, $\lambda_1 + \lambda_2 = \lambda_2 + \lambda_1$. This is the *commutativity* property.

3. $\lambda_1 \wedge \lambda_2 = \lambda_2 \wedge \lambda_1$
   Let $\vec{\mathbf{S}}$ be any state vector. By definition $[\![\lambda_1 \wedge \lambda_2]\!](\vec{\mathbf{S}}) = [\![\lambda_1]\!](\vec{\mathbf{S}}) \sqcap [\![\lambda_2]\!](\vec{\mathbf{S}})$. Due to commutativity of set intersection, $[\![\lambda_1]\!](\vec{\mathbf{S}}) \sqcap [\![\lambda_2]\!](\vec{\mathbf{S}}) = [\![\lambda_2]\!](\vec{\mathbf{S}}) \sqcap [\![\lambda_1]\!](\vec{\mathbf{S}})$. Hence, $\lambda_1 \wedge \lambda_2 = \lambda_2 \wedge \lambda_1$. This is again the *commutativity* property.

4. $\lambda_1 + (\lambda_2 + \lambda_3) = (\lambda_1 + \lambda_2) + \lambda_3$
   Let $\vec{\mathbf{S}}$ be any state vector. By definition, $[\![\lambda_1 + (\lambda_2 + \lambda_3)]\!](\vec{\mathbf{S}}) = [\![\lambda_1]\!](\vec{\mathbf{S}}) \sqcup \{[\![\lambda_2]\!](\vec{\mathbf{S}}) \sqcup [\![\lambda_3]\!](\vec{\mathbf{S}})\}$. From the definition of *associativity* of set union, the property follows.

5. $\lambda_1 \wedge (\lambda_2 \wedge \lambda_3) = (\lambda_1 \wedge \lambda_2) \wedge \lambda_3$
   Let $\vec{\mathbf{S}}$ be any state vector. By definition, $[\![\lambda_1 \wedge (\lambda_2 + \lambda_3)]\!](\vec{\mathbf{S}}) = [\![\lambda_1]\!](\vec{\mathbf{S}}) \sqcap \{[\![\lambda_2]\!](\vec{\mathbf{S}}) \sqcap [\![\lambda_3]\!](\vec{\mathbf{S}})\}$. From the definition of *associativity* of set intersection, the property follows.

6. $\lambda \circ \Theta = \Theta \circ \lambda = \Theta$
   By the definition of semantics for Θ, for any state vector $\vec{\mathbf{S}}$, $[\![\Theta]\!](\vec{\mathbf{S}}) = \Phi$, which is the *null* state vector, i.e., all components of it are ϕ. Hence, $[\![\lambda \circ \Theta]\!](\vec{\mathbf{S}}) = [\![\Theta]\!]([\![\lambda]\!](\vec{\mathbf{S}})) = \Phi$. Also, $[\![\Theta \circ \lambda]\!](\vec{\mathbf{S}}) = [\![\lambda]\!]([\![\Theta]\!](\vec{\mathbf{S}})) = \Phi$. This is the *zero* element.

7. $\lambda \circ \delta = \delta \circ \lambda = \lambda$
   By the definition of semantics for δ, for any state vector $\vec{\mathbf{S}}$, $[\![\delta]\!](\vec{\mathbf{S}}) = \vec{\mathbf{S}}$. Hence, $[\![\lambda \circ$

$\delta]](\vec{\mathbf{S}}) = [[\delta]]([[\lambda]](\vec{\mathbf{S}})) = [[\lambda]](\vec{\mathbf{S}})$. Also, $[[\delta \circ \lambda]](\vec{\mathbf{S}}) = [[\lambda]]([[\delta]](\vec{\mathbf{S}})) = [[\lambda]](\vec{\mathbf{S}})$. This is the *identity* element.

8. $\lambda_1 \circ (\lambda_2 + \lambda_3) = (\lambda_1 \circ \lambda_2) + (\lambda_1 \circ \lambda_3)$
Let $\vec{\mathbf{S}}$ be any state vector. By definition, $[[\lambda_1 \circ (\lambda_2 + \lambda_3)]](\vec{\mathbf{S}}) = [[\lambda_2 + \lambda_3]]([[\lambda_1]](\vec{\mathbf{S}}))$ $= [[\lambda_2]]([[\lambda_1]](\vec{\mathbf{S}})) \sqcup [[\lambda_3]]([[\lambda_1]](\vec{\mathbf{S}})) = [[\lambda_1 \circ \lambda_2]](\vec{\mathbf{S}}) \sqcup [[\lambda_1 \circ \lambda_3]](\vec{\mathbf{S}})$. This proves the property and indicates that $\circ$ distributes over $+$ on the left.

9. $(\lambda_1 + \lambda_2) \circ \lambda_3 = (\lambda_1 \circ \lambda_3) + (\lambda_2 \circ \lambda_3)$
Let $\vec{\mathbf{S}}$ be any state vector. By definition, $[[(\lambda_1 + \lambda_2) \circ \lambda_3]](\vec{\mathbf{S}}) = [[\lambda_3]]([[(\lambda_1 + \lambda_2)]](\vec{\mathbf{S}}))$ $= [[\lambda_3]]([[\lambda_1]](\vec{\mathbf{S}})) \sqcup [[\lambda_2]](\vec{\mathbf{S}})) = [[\lambda_1 \circ \lambda_3]](\vec{\mathbf{S}}) \sqcup [[\lambda_2 \circ \lambda_3]](\vec{\mathbf{S}})$. This proves the property and indicates that $\circ$ distributes over $+$ on the right.

10. $\lambda + \Theta = \lambda$
Let $\vec{\mathbf{S}}$ be any state vector. By definition of the semantics, $[[\lambda + \Theta]](\vec{\mathbf{S}}) = [[\lambda]](\vec{\mathbf{S}}) \sqcup [[\Theta]](\vec{\mathbf{S}}) = [[\lambda]](\vec{\mathbf{S}})$. Hence, $\lambda + \Theta = \lambda$.

11. $\lambda_1 \circ (\lambda_2 \circ \lambda_3) = (\lambda_1 \circ \lambda_2) \circ \lambda_3$
Let $\vec{\mathbf{S}}$ be any state vector. By definition of the semantics, $[[\lambda_1 \circ (\lambda_2 \circ \lambda_3)]](\vec{\mathbf{S}}) = [[(\lambda_2 \circ \lambda_3)]]([[\lambda_1]](\vec{\mathbf{S}})) = [[\lambda_3]]([[\lambda_2]]([[\lambda_2]](\vec{\mathbf{S}})))$. Similarly, $[[(\lambda_1 \circ \lambda_2) \circ \lambda_3]](\vec{\mathbf{S}}) = [[\lambda_3]]([[\lambda_1 \circ \lambda_2]](\vec{\mathbf{S}})) = [[\lambda_3]]([[\lambda_2]]([[\lambda_2]](\vec{\mathbf{S}})))$. Hence, the operator $\circ$ is associative.

12. $\lambda_1 + (\lambda_2 \wedge \lambda_3) = (\lambda_1 + \lambda_2) \wedge (\lambda_1 + \lambda_3)$
Let $\vec{\mathbf{S}}$ be any state vector. By definition of the semantics, $[[\lambda_1 + (\lambda_2 \wedge \lambda_3)]](\vec{\mathbf{S}}) = [[\lambda_1]](\vec{\mathbf{S}}) \sqcup [[(\lambda_2 \wedge \lambda_3)]](\vec{\mathbf{S}}) = [[(\lambda_2 \wedge \lambda_3)]](\vec{\mathbf{S}}) \sqcup [[\lambda_1]](\vec{\mathbf{S}})$ due to commutativity of set union. Hence, $\lambda_1 + (\lambda_2 \wedge \lambda_3) = (\lambda_1 + \lambda_2) \wedge (\lambda_1 + \lambda_3)$. The operator $+$ distributes over $\wedge$ on the left.

13. $\lambda_1 \wedge (\lambda_2 + \lambda_3) = (\lambda_1 \wedge \lambda_2) + (\lambda_1 \wedge \lambda_3)$
Let $\vec{\mathbf{S}}$ be any state vector. By definition of the semantics, $[[\lambda_1 \wedge (\lambda_2 + \lambda_3)]](\vec{\mathbf{S}}) = [[\lambda_1]](\vec{\mathbf{S}}) \sqcap [[(\lambda_2 + \lambda_3)]](\vec{\mathbf{S}}) = [[(\lambda_2 + \lambda_3)]](\vec{\mathbf{S}}) \sqcap [[\lambda_1]](\vec{\mathbf{S}})$ due to commutativity of set union. Hence, $\lambda_1 \wedge (\lambda_2 + \lambda_3) = (\lambda_1 \wedge \lambda_2) + (\lambda_1 \wedge \lambda_3)$. The operator $+$ distributes over $\wedge$ on the right.

14. $\lambda_1 \circ (\lambda_2 \wedge \lambda_3) = (\lambda_1 \circ \lambda_2) \wedge (\lambda_1 \circ \lambda_3)$
Let $\vec{\mathbf{S}}$ be any state vector. By definition of the semantics, $[[\lambda_1 \circ (\lambda_2 \wedge \lambda_3)]](\vec{\mathbf{S}}) = [[(\lambda_2 \wedge \lambda_3)]]([[\lambda_1]](\vec{\mathbf{S}})) = [[\lambda_2]]([[\lambda_1]](\vec{\mathbf{S}})) \sqcap [[\lambda_3]]([[\lambda_1]](\vec{\mathbf{S}})) = [[(\lambda_1 \circ \lambda_2)]](\vec{\mathbf{S}}) \sqcap [[(\lambda_1 \circ \lambda_3)]](\vec{\mathbf{S}})$. Hence the property holds true. This shows that the operator $\circ$ distributes over $\wedge$ on the left.

15. $\lambda \downarrow \lambda = \mathbf{1}$
This follows from the definition of the *constrain* ($\downarrow$) operator [4] and the definition of semantics.

16. $(\lambda_1 \wedge \lambda_2) \downarrow \lambda_3 = (\lambda_1 \downarrow \lambda_3) \wedge (\lambda_2 \downarrow \lambda_3)$
McMillan in [11] has proved that *constrain* operator distributes over $\wedge$ on the left. Using this result and the definition of semantics, the property follows.
Follows from a similar result proved by McMillan in [11] and the definition of semantics.

17. $(\lambda_1 + \lambda_2) \downarrow \lambda_3 = (\lambda_1 \downarrow \lambda_3) + (\lambda_2 \downarrow \lambda_3)$
McMillan in [11] has proved that *constrain* operator distributes over $+$ on the left. Using this result and the definition of semantics, the property follows.

2.4 Labeled Reachability Expressions and Reachability Matrices

A Reachability Matrix [24] is another framework which permits operations over state vectors. A Reachability Matrix $\mathbf{R}$ of order $k \times k$ is a matrix with k^2 terms, each of which is a Reachability Expression. The Reachability Expressions are applied column-wise to the state vector and are aggregated to get an element of the transformed state vector. Formally, let $\mathbf{R}$ be a Reachability Matrix which takes $\vec{\mathbf{S}}$ as an input state vector and computes a new state vector $\vec{\mathbf{S}}'$. The application of $\mathbf{R}$ on $\vec{\mathbf{S}}$ is denoted as $[\![\mathbf{R}]\!](\vec{\mathbf{S}})$ and is evaluated as shown

$$\mathbf{R} = \begin{pmatrix} r_{11} & \dots & r_{1k} \\ r_{21} & \dots & r_{2k} \\ \vdots & \vdots & \ddots \\ r_{k1} & \dots & r_{kk} \end{pmatrix}$$

$[\![\mathbf{R}]\!](S_1, S_2, \ldots, S_k) = (S'_1, S'_2, \ldots, S'_k)$ where $\forall i\ S'_i = [\![r_{1i}]\!](S_1) \cup [\![r_{2i}]\!](S_2) \cup \ldots \cup [\![r_{ki}]\!](S_k)$

2.4.1 Operations on Reachability Matrices

We briefly describe some operators and (operational) semantics of Reachability Matrices taken from [24].

- Φ is the additive identity matrix where each element $\Phi_{ij} = \Theta$.
- Δ is the multiplicative identity matrix where each element $\Delta_{ij} = \delta$ if $i = j$ and $\Delta_{ij} = \Theta$ otherwise.
- Given two Reachability Matrices $\mathbf{R}$ and $\mathbf{Q}$, define a Reachability Matrix $\mathbf{P} = \mathbf{R} + \mathbf{Q}$, where $\mathbf{P_{ij}} = \mathbf{R_{ij}} + \mathbf{Q_{ij}}$.
- Given two Reachability Matrices $\mathbf{R}$ and $\mathbf{Q}$, define a Reachability Matrix $\mathbf{P} = \mathbf{R} \circ \mathbf{Q}$, where $\mathbf{P_{ij}} = +_{l=1}^{k} \mathbf{R_{il}} \circ \mathbf{Q_{lj}}$.
- Given a Reachability Matrix $\mathbf{R}$, define $\mathbf{R}^0$ as Δ. Also, $\mathbf{R}^i = \mathbf{R}^{i-1} \circ \mathbf{R}$, $i \geq 0$. Define $*\mathbf{R} = +_{i=0}^{\infty} \mathbf{R}^i$
- Given two Reachability Matrices $\mathbf{R}$ and $\mathbf{Q}$, define a Reachability Matrix $\mathbf{P} = \mathbf{R}\ ;\ \mathbf{Q}$ to be $(\mathbf{R} + \Delta) \circ (\mathbf{Q} + \Delta)$.

Semantically, the following hold true [24]. We omit presenting proofs of the same here.

- $\forall \vec{\mathbf{S}}\ [\![\mathbf{R} + \mathbf{Q}]\!](\vec{\mathbf{S}}) = [\![\mathbf{R}]\!](\vec{\mathbf{S}}) \cup [\![\mathbf{Q}]\!](\vec{\mathbf{S}})$
- $\forall \vec{\mathbf{S}}\ [\![\mathbf{R} \circ \mathbf{Q}]\!](\vec{\mathbf{S}}) = [\![\mathbf{Q}]\!]([\![\mathbf{R}]\!](\vec{\mathbf{S}}))$

2.4.2 Properties of Reachability Matrices

We mention some properties of Reachability Matrices here. Given Reachability Matrices $\mathbf{R}_1$, $\mathbf{R}_2$ and $\mathbf{R}_3$, the following properties hold [24]:

- $\mathbf{R}_1 + (\mathbf{R}_2 + \mathbf{R}_3) = (\mathbf{R}_1 + \mathbf{R}_2) + \mathbf{R}_3$
- $(\mathbf{R}_1 + \mathbf{R}_2) = (\mathbf{R}_2 + \mathbf{R}_1)$
- $(\mathbf{R}_1 + \Phi) = (\Phi + \mathbf{R}_1) = \mathbf{R}_1$
- $\mathbf{R}_1 + \mathbf{R}_1 = \mathbf{R}_1$
- $\mathbf{R}_1 \circ (\mathbf{R}_2 \circ \mathbf{R}_3) = (\mathbf{R}_1 \circ \mathbf{R}_2) \circ \mathbf{R}_3$
- $\mathbf{R}_1 \circ \Delta{=}\Delta \circ \mathbf{R}_1 = \mathbf{R}_1$
- $\mathbf{R}_1 \circ (\mathbf{R}_2 + \mathbf{R}_3) = (\mathbf{R}_1 \circ \mathbf{R}_2) + (\mathbf{R}_1 \circ \mathbf{R}_3)$
- $(\mathbf{R}_1 + \mathbf{R}_2) \circ \mathbf{R}_3 = (\mathbf{R}_1 \circ \mathbf{R}_3) + (\mathbf{R}_2 \circ \mathbf{R}_3)$

It is clear from the operators and properties of Reachability Matrices that like Labeled Reachability Expressions, these also allow for applying any arbitrary transition relation on any arbitrary sets of states by writing the suitable Reachability Matrix and applying appropriate matrix operations to get the resultant state vector.

We claim that Labeled Reachability Expressions without the use of two operators: $\wedge$, and $\downarrow$ can be completely cast as Reachability Matrices using the following translation. Assume for ease of explanation that the length of the state vector is 3. It is possible to give a general translation scheme. We are working on the proofs which are not put in this report for lack of time. Given two Labeled Reachability Expressions λ_1 and λ_2 over LE and L, we can formulate equivalent Reachability Matrices $\mathbf{M}_1$ and $\mathbf{M}_2$ of order 3 as shown

- If $\lambda_1 = T_1$, the corresponding matrix representation for $\mathbf{M}_1$ would be

$$\mathbf{M}_1 = \begin{pmatrix} T_1 & \Theta & \Theta \\ \Theta & \delta & \Theta \\ \Theta & \Theta & \delta \end{pmatrix}$$

Evaluating $[\![T_1]\!](S_1, S_2, S_3) = (Img(S_1, T_1), S_2, S_3)$. The same effect would be captured by evaluating $[\![\mathbf{M}_1]\!](S_1, S_2, S_3)$.

- Given two Labeled Reachability Expressions λ_1 and λ_2 with equivalent Reachability Matrix formulations $\mathbf{M}_1$ and $\mathbf{M}_2$, evaluating $[\![\lambda_1 + \lambda_2]\!](S_1, S_2, S_3)$ would be the same as evaluating $[\![\mathbf{M}_1 \cup \mathbf{M}_2]\!](S_1, S_2, S_3) = [\![\mathbf{M}_1]\!](S_1, S_2, S_3) \sqcup [\![\mathbf{M}_2]\!](S_1, S_2, S_3)$.
- Given two Labeled Reachability Expressions λ_1 and λ_2 with equivalent Reachability Matrix formulations $\mathbf{M}_1$ and $\mathbf{M}_2$, evaluating $[\![\lambda_1 \circ \lambda_2]\!](S_1, S_2, S_3)$ would be the same as evaluating $[\![\mathbf{M}_1 \circ \mathbf{M}_2]\!](S_1, S_2, S_3)$.

- Given two Labeled Reachability Expressions λ_1 and λ_2 with equivalent Reachability Matrix formulations $\mathbf{M}_1$ and $\mathbf{M}_2$, evaluating $[\![\lambda_1 \ ; \ \lambda_2]\!](S_1, S_2, S_3)$ would be the same as evaluating $[\![\mathbf{M}_1 \ ; \ \mathbf{M}_2]\!](S_1, S_2, S_3)$.

- Given a Labeled Reachability Expression λ_1 with equivalent Reachability Matrix formulation $\mathbf{M}_1$, evaluating $[\![*\lambda_1]\!](S_1, S_2, S_3)$ would be the same as evaluating $[\![*\mathbf{M}_1]\!](S_1, S_2, S_3)$.

- Given a Labeled Reachability Expression λ_1 with equivalent Reachability Matrix formulation $\mathbf{M}_1$, evaluating $[\![(\lambda_1)_{L_2}]\!](S_1, S_2, S_3)$ would be the same as evaluating $[\![\mathbf{M}_1]\!]([\![\mathbf{M}_2]\!](S_1, S_2, S_3))$, where $\mathbf{M}_2$ is the following Reachability Matrix Formulation

$$\mathbf{M}_2 = \begin{pmatrix} \Theta & \delta & \Theta \\ \Theta & \delta & \Theta \\ \Theta & \Theta & \delta \end{pmatrix}$$

- Given a Labeled Reachability Expression λ_1 with equivalent Reachability Matrix formulation $\mathbf{M}_1$, evaluating $[\![L_2 : \lambda_1]\!](S_1, S_2, S_3)$ would be the same as evaluating $[\![\mathbf{M}_1]\!]([\![\mathbf{M}_2]\!](S_1, S_2, S_3))$, where $\mathbf{M}_2$ is the following Reachability Matrix Formulation

$$\mathbf{M}_2 = \begin{pmatrix} \delta & \Theta & \Theta \\ \delta & \Theta & \Theta \\ \Theta & \Theta & \delta \end{pmatrix}$$

This shows that without the use of the two operators $\wedge$, and $\downarrow$, Labeled Reachability Expressions can also be represented using Reachability Matrices. For our work, we these two operators are of prime importance in dealing with large circuits. Additionally, formulating a Reachability Matrix is a cumbersome experience. Writing traversal techniques using Labeled Reachability Expressions is more intuitive. We are yet to implement the translator of a Labeled Reachability Expression to its corresponding Reachability Matrix.

2.5 Implementation of Generic Framework

Figure 2.1 shows the Layered Architecture of our generic framework. The framework is built on top of a BDD-based reachability engine, NuSMV which performs the very basic operation of image computation only. Though we have currently used NuSMV as the back-end, it is possible to extend the framework to be integrated with any SAT solver. The middle layer interprets the search techniques encoded as a Labeled Reachability Expression. Additionally it also provides some Boolean and set-theoretic functionality. The topmost layer is the one that allows us to write *meta-programs* to encode different search techniques. Each Labeled Reachability Expression is a meta program. We conjecture that any search method based on least fixpoint formulation can be used with our framework by only translating the technique into its equivalent Labeled Reachability Expression. This layer provides tremendous flexibility to the framework by allowing us to write different search techniques and analyse and compare their performances.

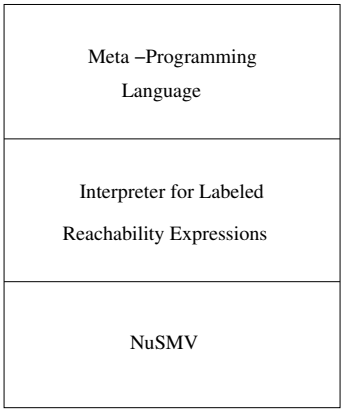


Figure 2.1: Layered Architecture of Framework

Additionally, we conjecture that using this framework we can completely capture a restricted version of modal μ-calculus (without greatest fixpoint negation operators). We are yet to investigate if everything that can be captured using modal mu-calculus would also be allowed by our framework. This would then help us to reason about a whole class of least fixpoint based search techniques. The use of labels allows the flexibility of applying any arbitrary transitions on any arbitrary sets of states. The main contribution of having such a framework is that specification of any new search technique will be done at the meta-programming layer, without having to do any modifications in the reachability engine (back-end). This is in stark contrast to the earlier work reported by Cho *et al.* in [1] and Govindaraju *et al.* in [9] who had to build separate model checkers for each traversal technique that has been analysed.

Next, in Chapter 3, we discuss encoding of a few state of the art traversal algorithms using our framework.

Chapter 3

Encoding Traversal Algorithms in Our Framework

In Chapter 2, we discussed our framework based on Labeled Reachability Expressions. In this chapter, we discuss how a few state of the art approximate traversal algorithms [1], which are taken from a class of least fixpoint approximation methods can be encoded in our generic framework. We also note that our framework would allow analysing the performance of any search technique which can be encoded using Labeled Reachability Expressions.

3.1 Approximate Traversal Algorithms

We discuss the algorithms presented in Cho *et al.* in [1]. Assume the partitioning of the circuit as was explained in Section 1.2.1.1 in Chapter 1. We provide intuitive descriptions of each strategy and least fixpoint approximation characterization as given by Moon *et al.* in [25] for each of these algorithms.

3.1.1 Use of *Constrain* operator in Image Computation

Given a set of states S and a transition relation T, Coudert and Madre in [4] have shown that $Img(S,T) = Img(1, T \downarrow S)$, i.e., the problem of finding the image of S under T now reduces to the problem of finding the range of $T \downarrow S$. For a large transition system with say two overlapping projections S_1 and S_2, we can write the image computation as follows:
$Img(S_1 \wedge S_2, T) = Img(S_1, T \downarrow S_2)$.

3.1.2 Use of labels in encoding search methods

Labeled Reachability Expressions help in choosing any arbitrary state component by means of referring to its corresponding label. This feature is very useful in encoding the search techniques discussed in this chapter. Some state components can actually be used to store the results of intermediate computations and they need not represent the state space of the circuit. We highlight such state components while explaining the encoding of the different search methods.

3.1.3 Machine by Machine Traversal

This strategy traverses the partitions serially and iteratively, until a fixpoint in the computation of the reached set is obtained. MBM initially sets the reachable states of all submachines to tautology and computes the reachable states of each submachine in turn. In each iteration, the transition relation of the submachine being traversed is constrained by the reachable states of the other submachines obtained from the previous iteration. The process is repeated until the sets of reachable states stops shrinking. It is hence computing a greatest fixpoint computation with least fixpoint computations (reachability analyses of the submachines) performed at each iteration.

This can be characterised as a least fixpoint formulation as shown
$M_j^{-1} = I_j$

$$\mathrm{M}_j^i = \begin{cases} \mu S.I_j \vee Img(S, \alpha_j(T(I,X,X')) \wedge \bigwedge_{l \neq j} M_l^{i-1}) & \text{if } j = \rho(i) \\ \mathrm{M}_j^{i-1} & \text{otherwise} \end{cases}$$

where $\rho(i)$ is defined as
$\rho(0) = 0$

$$\rho(i+1) = \begin{cases} \rho(i) & \text{if } M^i_{\rho(i)} \neq M^{i-1}_{\rho(i)} \\ (\rho(i)+1) mod\ k & \text{otherwise} \end{cases}$$

This selection function ensures that MBM takes each submachine to convergence before switching to another submachine.

3.1.3.1 Encoding as a labeled reachability expression

Assume for ease of explanation, that we have only two projections of the system. Let the state vector $\vec{\mathbf{S}} = (S_1,\ S_2,\ S_3,\ S_4)$, where the first two components are the state components and the last two components are used during intermediate computations but don't represent any component of the state space of the circuit. In the i^{th} iteration, components S_3 and S_4 store the reachable states obtained from the traversal of the projections 1 and 2 computed in the i^{th} iteration respectively, while the components S_1 and S_2 store the reachable sets computed in the $(i-1)^{th}$ iteration which are used to constrain the image computations in the i^{th} iteration. The encoding is done as shown.
$*(\ (*L_3 :\ (((T_1)_{L_1}\ \downarrow\ \delta_{L_2})\ +\delta_{L_1})\ +\ *L_4 :\ (((T_2)_{L_2}\ \downarrow\ \delta_{L_1})\ +\delta_{L_2}))\ \circ\ L_1 : \delta_{L_3}\ \circ\ L_2 : \delta_{L_4})$ Assume that the state vector $\vec{\mathbf{S}}$ is initialised to tautology. This encoding computes the reachable states of partition 1, denoted by $*L_3 :\ (((T_1)_{L_1}\ \downarrow\ \delta_{L_2})\ +\delta_{L_1})$ and stores the set of reachable states in the state component S_3. The state vector component S_2 is not affected by this computation. The resultant state vector is then transformed to also contain the reachable states of the second partition by applying $*L_4 :\ (((T_2)_{L_2}\ \downarrow\ \delta_{L_1})\ +\delta_{L_2})$. This computes and stores the set of reachable states in this partition in the state component S_4. Next, the labels are updated before the iteration completes. The contents of the state component S_4 is copied in

the component S_2, while the contents of S_3 is copied in S_1. This completes one iteration of the strategy. The procedure terminates when the state vector $\vec{\mathbf{S}}$ reaches a fixpoint, i.e. each state vector component reaches its fixpoint. The over-approximated set of reachable states is then returned as $\gamma(S_1, S_2)$.

This encoding incorporates the same selection function as is used in MBM. We provide an inductive argument to show that this encoding produces the same set of reachable sets as its corresponding least fixpoint approximation formalism.
Let predicates r_j^i and q_j^i denote the reachable sets from the least fixpoint characterization of MBM and our encoding respectively for the j^{th} partition at the end of the i^{th} iteration. To start off, r_1^0 and r_2^0 are both initialised to tautology. In our encoding too, q_1^0 and q_2^0 are initialised to Tautology (**T**). Assume that both the representations compute the same set of states at the end of the i^{th} iteration. At the end of the $i+1^{th}$ iteration, MBM computes the reachable sets as
$r_1^{i+1} = \mu S.(r_1^i \vee img(S, \alpha_1(T) \wedge r_2^{i-1}))$, and
$r_2^{i+1} = \mu S.(r_2^i \vee img(S, \alpha_2(T) \wedge r_1^{i-1}))$

The current approximated reachable set for the system as $\gamma(r_1^{i+1}, r_2^{i+1})$ which is returned if a fixpoint of each of the reachable states has been attained. The same computation is done with our encoding too. At the end of the $i+1^{th}$ iteration, it computes the reachable sets as
$q_1^{i+1} = \mu S.(q_1^i \vee img(S, \alpha_1(T) \downarrow q_2^{i-1}))$, and
$q_2^{i+1} = \mu S.(q_2^i \vee img(S, \alpha_2(T) \downarrow q_1^{i-1}))$, and
and the current approximated reachable set for the system as $\gamma(q_1^{i+1}, q_2^{i+1})$. This would be true for any number of ensuing iterations till a fixpoint has been generated. Hence, our encoding is a correct representation of the MBM strategy.

3.1.4 Frame by Frame Traversal

In this strategy, instead of traversing partition M_i before partition M_{i+1}, one step of image computation is done for every sub-space, then all sub-spaces move one time frame ahead, and then another image computation is performed (one per sub-space). The algorithm terminates when a fixpoint in the computation of the reached set is obtained. Depending on how the constraints are updates, there are two variants: Reached FBF and To FBF.

3.1.4.1 Reached FBF

RFBF initially sets the reachable states of each submachine to its respective initial states. It then interleaves the reachability analyses of the submachines by computing one image computation for each submachine in turn. It also constrains the transition relation of each submachine. When computing the i^{th} step of reachability analysis for a submachine, the accumulated reachable states of the other submachines from tghe previous iteration are used. The computation continues till the sets of reachable states stop growing.

This can be characterised as a least fixpoint formulation as shown

$$R_j^{-1} = I_j$$

$$R_j^i = \begin{cases} \mu S.I_j \vee Img(S, \alpha_j(T(I,X,X')) \wedge \bigwedge_{l \neq j} R_l^{i-1}) & \text{if } j = i \bmod k \\ R_j^{i-1} & \text{otherwise} \end{cases}$$

The selection function used here picks one submachine at a time and performs one image computation in each to complete an iteration.

3.1.4.2 Encoding as a labeled reachability expression

Assume for ease of explanation, that we have only two projections of the system.

RFBF uses as constraints the reachable sets computed in the previous iteration. So we would need two more state components to keep track of the constraints to be imposed in every iteration, but are not representing the reachable states of the circuit. Let the state vector $\vec{\mathbf{S}} = (S_1,\ S_2,\ S_3,\ S_4)$, where the last two components are used to store the reachable sets computed in the current iteration to be used as constraints in the next iteration. The encoding is done as shown.

$*((((L_3 :\ (((T_1)_{L_1} \downarrow \delta_{L_2}) + \delta_{L_1}) \circ L_4 : (((T_2)_{L_2} \downarrow \delta_{L_1}) + \delta_{L_2})) \circ (L_1 : \delta_{L_3}) \circ (L_2 : \delta_{L_4}))$

Assume that the first two components of the state vector $\vec{\mathbf{S}}$ are intialised to their projected versions of the initial state. Components S_3 and S_4 are intialised to the projected versions of projection 1 and 2 respectively. This encoding first performs image computation of partition 1, denoted by $(L_3 :\ (((T_1)_{L_1} \downarrow \delta_{L_2}) \circ \delta_{L_1}))$ and stores the set of reachable states after one image computation in the state component S_3. The state vector component S_2 is not affected by this computation. The resultant state vector is then transformed to also contain the reachable states after one image computation of the second partition by applying $(L_4 :\ (((T_2)_{L_2} \downarrow \delta_{L_1}) + \delta_{L_2}))$. This computes and stores the set of reachable states after one image computation in this partition in the state component S_4. Next, the labels are updated before the iteration completes. The contents of the state component S_4 is copied in the component S_2, while the contents of S_3 is copied in S_1. This completes one iteration of the strategy. The procedure terminates when the state vector $\vec{\mathbf{S}}$ reaches a fixpoint, i.e. each state vector component reaches its fixpoint. The over-approximated set of reachable states is then returned as $\gamma(S_1, S_2)$. i This encoding incorporates the same selection function as is used in RFBF. We provide an inductive argument to show that this encoding produces the same set of reachable sets as its corresponding least fixpoint approximation formalism.

Let predicates r_j^i and q_j^i denote the reachable sets from the least fixpoint characterization of RFBF and our encoding respectively for the j^{th} partition at the end of the i^{th} iteration. To start off, r_1^0 and r_2^0 are initialised to $\alpha_1(Init)$ and $\alpha_2(Init)$, where $Init$ is the intial set of states in the system. In our encoding too, q_1^0 and q_2^0 are their respective projected values of $Init$. Assume that both the representations compute the same set of states at the end of the i^{th} iteration. At the end of the $i+1^{th}$ iteration, RFBF computes the reachable sets as

$r_1^{i+1} = \mu S.(r_1^i \ \vee\ img(S, \alpha_1(T) \ \wedge\ r_2^{i-1}))$, and

$r_2^{i+1} = \mu S.(r_2^i \ \vee\ img(S, \alpha_2(T) \ \wedge\ r_1^{i-1}))$

Due to the selection strategy, one image computation per partition is performed in each iteration and the set of reachable states is stored to be used as constraints in the next iteration.

The current approximated reachable set for the system as $\gamma(r_1^{i+1}, r_2^{i+1})$ which is returned if a fixpoint of each of the reachable states has been attained. The same computation is done with our encoding too. At the end of the $i+1^{th}$ iteration, it computes the reachable sets as

$q_1^{i+1} = (q_1^i \ \vee \ img(S, \alpha_1(T) \ \downarrow \ q_2^{i-1}))$, and
$q_2^{i+1} = (q_2^i \ \vee \ img(S, \alpha_2(T) \ \downarrow \ q_1^{i-1}))$, and
and the current approximated reachable set for the system as $\gamma(q_1^{i+1}, \ q_2^{i+1})$. This would be true for any number of ensuing iterations till a fixpoint has been generated. Hence, our encoding is a correct representation of the RFBF strategy.

3.1.4.3 To FBF

Unlike RFBF which uses the *reached* sets as constraints, this method uses the *to* sets (i.e. the images themselves) from the previous image computation to constrain the transition relation. Since this uses the smallest constraint set, it gives the smallest reached set upon convergence.

This can be characterised as a least fixpoint formulation as shown
$L_j^{-1} = I_j$

$$\mathrm{L}_{to\_j}^{i} = \begin{cases} \mu S.I_j \vee \mathrm{L}_{to\_j}^{i} & \text{if } j = i \bmod k \\ \mathrm{L}_j^{i-1} & \text{otherwise} \end{cases}$$

where

$\mathrm{L}_{to\_j}^{i} = Img(S, \alpha_j(T(I,X,X')) \wedge \bigwedge_{l \neq j} L_{to\_l}^{i-1})$ depicts the *to* sets.

The selection function used here again picks one submachine at a time and performs one image computation in each to complete an iteration.

3.1.4.4 Encoding as a labeled reachability expression

Assume for ease of explanation, that we have only two projections of the system. TFBF uses as constraints the to sets computed in the previous iteration. We would need two state components which keep track of the constraints to be imposed in every iteration. In addition, we would also two more state components to update the set of reachable states of every component after the current iteration completes. So, the state vector $\vec{\mathbf{S}} = (S_1, \ S_2, \ S_3, \ S_4, \ S_5, \ S_6)$, where the set of reachable states is the implicit conjunction of the first two components. Components S_3 and S_4 are used to store the *to* sets of projections 1 and 2 respectively, while components S_5 and S_6 are usec to store the reachable sets of projections 1 and 2 respectively. The encoding is done as shown.
$*(((((L_1 : (L_3 : ((T_1)_{L_1} \downarrow \delta_{L_6}) \ + \ \delta_{L_1}) \circ L_6 : (L_4 : ((T_2)_{L_2} \downarrow \delta_{L_5}) \ + \ \delta_{L_2}))) \circ (L_5 : \delta_{L_3}) \circ (L_6 : \delta_{L_4}))$
Assume that the first two components and the last two components of the state vector $\vec{\mathbf{S}}$ are intialised to their projected versions of the initial state of projections 1 and 2 respectively. Also, S_3 and S_5 are intialised to the projected versions of the initial states of projection 1, while S_4 and S_6 are intialised to the projected versions of the initial states of projection 2. This encoding first performs image computation of projection 1, denoted by $L_1 : (L_3 : ((T_1)_{L_1} \downarrow \delta_{L_6}) \ + \ \delta_{L_1})$ and stores the set of reachable states after one image computation in the state component S_1. Also, the *to* set is stored in the state component S_3. The other state vector components are not affected by this computation. The resultant state vector is then transformed to also contain the reachable states after one image computation of the second partition by applying $L_2 : (L_4 : ((T_2)_{L_2} \ \downarrow \ \delta_{L_5}) \ + \ \delta_{L_2})$. This computes and stores the set of reachable states after one image

computation in this partition in the state component S_2 and the *to* set is stored in S_4. Next, the labels are updated before the iteration completes. The contents of S_3 is copied into S_5 and those of S_4 is copied into S_6. This completes one iteration of the strategy. The procedure terminates when the state vector $\vec{\mathbf{S}}$ reaches a fixpoint, i.e. each state vector component reaches its fixpoint. The over-approximated set of reachable states is then returned as $\gamma(S_1, S_2)$.

We provide an inductive argument to show that this encoding produces the same set of reachable sets as its corresponding least fixpoint approximation formalism. Let predicates r_i^j and q_i^j denote the reachable sets from the procedure TFBF and our encoding respectively for subset i at the end of iteration j. Also, let predicates $to\_r_i^j$ and $to\_q_i^j$ denote the *to* sets from the procedure TFBF and our encoding respectively for subset i at the end of iteration j.

To start off, r_1^0 and r_2^0 are initialised to $\alpha_1(Init)$ and $\alpha_2(Init)$, where *Init* is the intial set of states in the system. In our encoding too, q_1^0 and q_2^0 are their respective projected values of *Init*. Assume that both the representations compute the same set of states at the end of the i^{th} iteration. At the end of the $i+1^{th}$ iteration, TFBF computes the reachable sets as
$r_1^{i+1} = \mu S.(r_1^i \vee img(S, \alpha_1(T) \wedge to\_r_2^{i-1}))$, and
$r_2^{i+1} = \mu S.(r_2^i \vee img(S, \alpha_2(T) \wedge to\_r_1^{i-1}))$
where the *to* sets are as shown
$to\_r_1^{i+1} = img((r_1^i, \alpha_1(T) \wedge to\_r_2^{i-1}))$, and
$to\_r_2^{i+1} = img((r_2^i, \alpha_2(T) \wedge to\_r_1^{i-1}))$. Due to the selection strategy, one image computation per partition is performed in each iteration and the set of *to* states is stored to be used as constraints in the next iteration.

The current approximated reachable set for the system as $\gamma(r_1^{i+1}, r_2^{i+1})$ which is returned if a fixpoint of each of the reachable states has been attained. The same computation is done with our encoding too. At the end of the $i+1^{th}$ iteration, it computes the reachable sets as
$q_1^{i+1} = (q_1^i \vee img(S, \alpha_1(T) \downarrow to\_q_2^{i-1}))$, and
$q_2^{i+1} = (q_2^i \vee img(S, \alpha_2(T) \downarrow to\_q_1^{i-1}))$, and
where the *to* sets are as shown
$to\_q_1^{i+1} = img((q_1^i, \alpha_1(T) \wedge to\_q_2^{i-1}))$, and
$to\_q_2^{i+1} = img((q_2^i, \alpha_2(T) \wedge to\_q_1^{i-1}))$. and the current approximated reachable set for the system as $\gamma(q_1^{i+1}, q_2^{i+1})$. This would be true for any number of ensuing iterations till a fixpoint has been generated. Hence, our encoding is a correct representation of the TFBF strategy.

3.1.5 TMBM

This is a hybrid traversal method, which starts off as TFBF and after a specified number of iterations switches to MBM. Use of TFBF at the start of traversal gives the benefit of accuracy in image computation, while MBM helps in faster convergence.

First, TFBF runs for t time frames. The reached set is saved, and MBM uses the *to* set (frontier) at time t as the initial state set. When MBM reaches a fixpoint, the reached set computed by MBM is added to the saved reached set computed by TFBF to give the final reached set.

3.1.5.1 Encoding as a labeled reachability expression

Assume for ease of explanation, that we have only two projections of the system. Let the state vector $\vec{\mathbf{S}} = (S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8, S_9, S_{10})$, where the set of reachable states is returned

as $\gamma((S_1 \cup S_9), (S_2 \cup S_{10}))$. The remaining components are used to update the constraints and the reachable sets accordingly. The encoding is done as shown.
$((((((L_5 : (L_3 : ((T_1)_{L_1} \downarrow \delta_{L_8}) + \delta_{L_1}) \circ L_6 : (L_4 : ((T_2)_{L_2} \downarrow \delta_{L_7}) + \delta_{L_2})) \circ (L_1 : \delta_{L_5})) \circ ((L_2 : \delta_{L_6})) \circ ((L_7 : \delta_{L_3})) \circ ((L_8 : \delta_{L_4})))^t \circ *(*(L_9 : (((T_1)_{L_3} \downarrow \delta_{L_{10}}) + \delta_{L_9})) + *(L_{10} : (((T_2)_{L_4} \downarrow \delta_{L_9}) + \delta_{L_{10}})))$
Here t is the number of iterations of TFBF before switching over to MBM strategy. This encoding first performs t iterations of TFBF and stores its reachable states in the state components S_1 and S_2. The components S_3 and S_4 store the to sets. Next, MBM starts off by using the to sets as initial states and iterates till a fixpoint is obtained. The reachable states computed by MBM are stored in S_9 and S_{10}. Finally the set of reachable states is returned as $\gamma((S_1 \cup S_9), (S_2 \cup S_{10}))$.

Proving that this encoding truely computes reachable states according the TMBM strategy is trivial and follows from the fact that our encodings of MBM and TFBF are correct.

3.2 Salient Features of our Generic Framework

Shown in Figure 2.1 is the layered architecture of our generic BDD-based framework. The back-end engine used is NuSMV [22]. The search techniques are encoded as a meta program involving Labeled Reachability Expressions which is interpreted by the layer for Interpreter of Labeled Reachability Expressions. Each Labeled Reachability Expression allows us to apply arbitrary transition functions on arbitrary sets of states. If as we conjecture, we are able to encode every Labeled Reachability Expression as a modal μ-calculus expression, then we can develop a *prover* component within our framework which would assist in checking certain kinds of relations between the semantics of two different Labeled Reachability Expressions by appropriately encoding them as two modal μ-calculus formulas, and checking for implication between these formulas. This would help immensely to also check if the semantics of any encoded search technique is correct by checking its relation in this *prover* component with the *vanilla* search technique which is the most inefficient traversal method but is nevertheless a correct formulation of computing the reachable states.

Also, our interpreter uses natural operational semantics of manipulating a state vector. Evaluating a Labeled Reachability Expression on a state vector involves the three following steps:

1. Select the state component on which the expression is to be applied, and
2. Perform the application of the expression on the selected component. The other components remain unaffected, and
3. Generate a transformed state vector which reflects the application of the expression.

Reachability Expressions do not provide the facility of applying arbitrary transitions on arbitrary sets of states.

3.2.1 Comparison with NuSMV

Our framework is much more powerful than NuSMV. Coding up such a facility of embedding different traversal methods in NuSMV would require lot of familiarity with the functions available and the parameter passing required. As compared to that, learning to use our meta-programming language is much easier, more intutive and less cumbersome.

Table 3.1: **Circuit Details**: This presents details of the circuits we ran our experiments on. They are picked from ISCAS-89 benchmark of sequential circuits. The last column contains the number of clusters in each of the circuits which we got from Gaurishankar Govindaraju and are the same ones used by him [9], except for the circuit s35932.v. We explain our method of determining the clusters for this circuit.

Circuit Details			
Ckt	PI	FF	#clusters
s13207.v	31	669	47
s15850.v	14	597	64
s38584.v	12	1452	205
s35932.v	35	1728	54

It would be extremely difficult to implement a prover for equivalence or implication between different search strategies in NuSMV because these search strategies are specified as pieces of code, and so one would need a full-blown sequential program verifier. On the other hand, we conjecture that Labeled Reachability Expressions allow us to get away with a decision procedure for modal μ-calculus, and the decision problem here is known to be in EXPTIME.

Also, our framework allows flexibility in choosing different cluster orders and choice of a different back-engine. It hence is an extensible framework which can also be integrated with any SAT solver instead of a BDD package at the bottom-most layer of the framework (i.e. the reachability engine).

3.3 Results

We have implemented the traversal algorithms using labeled reachability expressions on top of NuSMV [22]. We have run all experiments on a machine having Intel Pentium-IV 3.0 Ghz processor, 1GB RAM, 160GB HDD with Fedora Core Linux 3.4.3-6.fc3 installed. Our results match up with those reported by Govindaraju *et al.* [9] in most cases. We note our observations about the other discrepancies.

Table 3.1 presents details of the circuits we ran our experiments on. They are picked from ISCAS-89 benchmark of sequential circuits. The last column contains the number of clusters in each of the circuits which we got from Gaurishankar Govindaraju and are the same ones used by him [9]. Table 3.2 presents the results obtained from our framework. We have chosen these circuits to run our experiments as they have allowed us to compare our results with the already published ones by Cho *et al.* in [1] and Govindaraju *et al.* [9].

The columns A and C list sat_fr (satisfying fraction) of the reachable state results from Govindaraju *et al.* [9] for disjoint and overlapping projections respectively. It is computed as the ratio of the reachable states and the total state space of the circuit. Columns B and D list results from our implementation for disjoint and overlapping projections respectively. We experimented with different ordering of the partitions (i.e. the order of choice in the *selection* function) and report our best results. The *iter* columns contain the number of iterations it has taken for the method to converge. For TMBM based traversals, the *iter* column lists the

Table 3.2: **Traversal Results**: Columns A and C list sat_fr (satisfying fraction) of the reachable state from Govindaraju *et al.* [9] for disjoint and overlapping projections respectively. It is computed as the ratio of the reachable states and the total state space of the circuit. Columns B and D list results from our implementation for disjoint and overlapping projections respectively. We experimented with different ordering of the partitions (i.e. the order of choice in the *selection* function) and report our best results. The *iter* columns contain the number of iterations it has taken for the method to converge. For TMBM based traversals, the *iter* column lists the number of TFBF iterations + the number of iterations it took MBM to converge.

Traversal Results									
Ckt	Method	A	iter	B	iter	C	iter	D	iter
s13207.v	TMBM	3.42e-106	10+6	3.02e-106	10+6	1.13e-115	10+5	1.03e-116	10+8
s15850.v	TMBM	5.84e-102	10+5	5.98e-100	10+8	3.94e-102	10+4	2.81e-101	10+6
s38584.v	TMBM	6.49e-41	10+2	5.96e-41	10+5	5.76e-57	10+5	4.93e-57	10+5
s35932.v	TMBM	–	–	4.3e-71	10+5	–	–	3.99e-71	10+5

Table 3.3: **Additional results**: This table shows the comparison of the Peak BDD node count and traversal times published by Govindaraju *et al.* in [9] and those obtained from our framework. Govindaraju *et al.* have not revealed their results of traversal times. The Time column lists the total time it takes for the traversal in seconds.

Peak Memory Usage (BDD nodes) & Traversal times(in secs.)						
Ckt	A	B	Time (secs.)	C	D	Time (secs.)
s13207.v	161447	173215	1265.0	198779	219769	1427.4
s15850.v	271093	400125	2193.5	336048	496117	3095.3
s38584.v	646258	831054	6352.7	1853461	235198	7996.9
s35932.v	–	96543	20811.3	–	1015639	25811.4

number of TFBF iterations + the number of iterations it took MBM to converge. For circuit s13207.v, we experimented with different cluster orderings and have reported our best results. In the case of circuit s15850.v, we are unable to match results with Govindaraju *et al.* [9]. As we have shown that our implementation is a correct translation of the traversal algorithms, we attribute this difference to possibly the difference in the BDD packages used for implementation. Govindaraju *et al.* [9] use David Long's BDD package [26]. For the largest circuit s35932.v, Govindaraju *et al.* [9] doesn't report any results as they were unable to get the same partitions as were used by Cho *et al.* in [1]. We have some knowledge of the structure of this particular circuit by looking at the dependency plot of this circuit.

The dependency plot shown in Figure 3.1 between the next and the current state variables in the circuit s35932.v, reveals a lot about the structure of this circuit. We observed that the circuit is built up of nine *boxes* which are functionally equivalent. In each of the nine boxes, we were able to identify 6 groups of state variables which have have similar next state equations as their corresponding groups in other boxes. Only the state variables belonging to the first box depend on the PIs.

We have used each group of variables as a partition of the circuit and have hence got 54 clusters in this circuit, which incidentally is also the same number of clusters mentioned by Cho *et al.* in [1] but like Govindaraju *et al.* in [9], we were also unable to generate 54 clusters based on the partitioning strategy proposed by Cho *et al.* in [1]. To generate projections for our experiment with this circuit, we chosen some random state variables as the overlapping variables across the disjoint partitions. We are yet to fully investigate the benefits of such a regular structure which can be exploited to get better traversal results.

Shown in Table 3.3 are details of the Peak BDD nodes in the traversal results and the total traversal time it takes in seconds. Govindaraju *et al.* in [9] report their Peak BDD node counts but no data regarding their traversal times was made available. Columns A and C list their Peak BDD node counts for disjoint partitions and overlapping projections respectively. Columns B and D list the Peak BDD node counts from our framework for disjoint partitions and overlapping projections respectively. The columns of Time denote our traversal times in seconds. We observe that our Peak Node counts are higher than those reported in [9]. This is expected as the framework incurs the overhead of storing constraints which are to be used in the next iteration of traversal as temporary state components. The largest circuit s35932.v takes more than 5 hours for a complete approximate traversal.

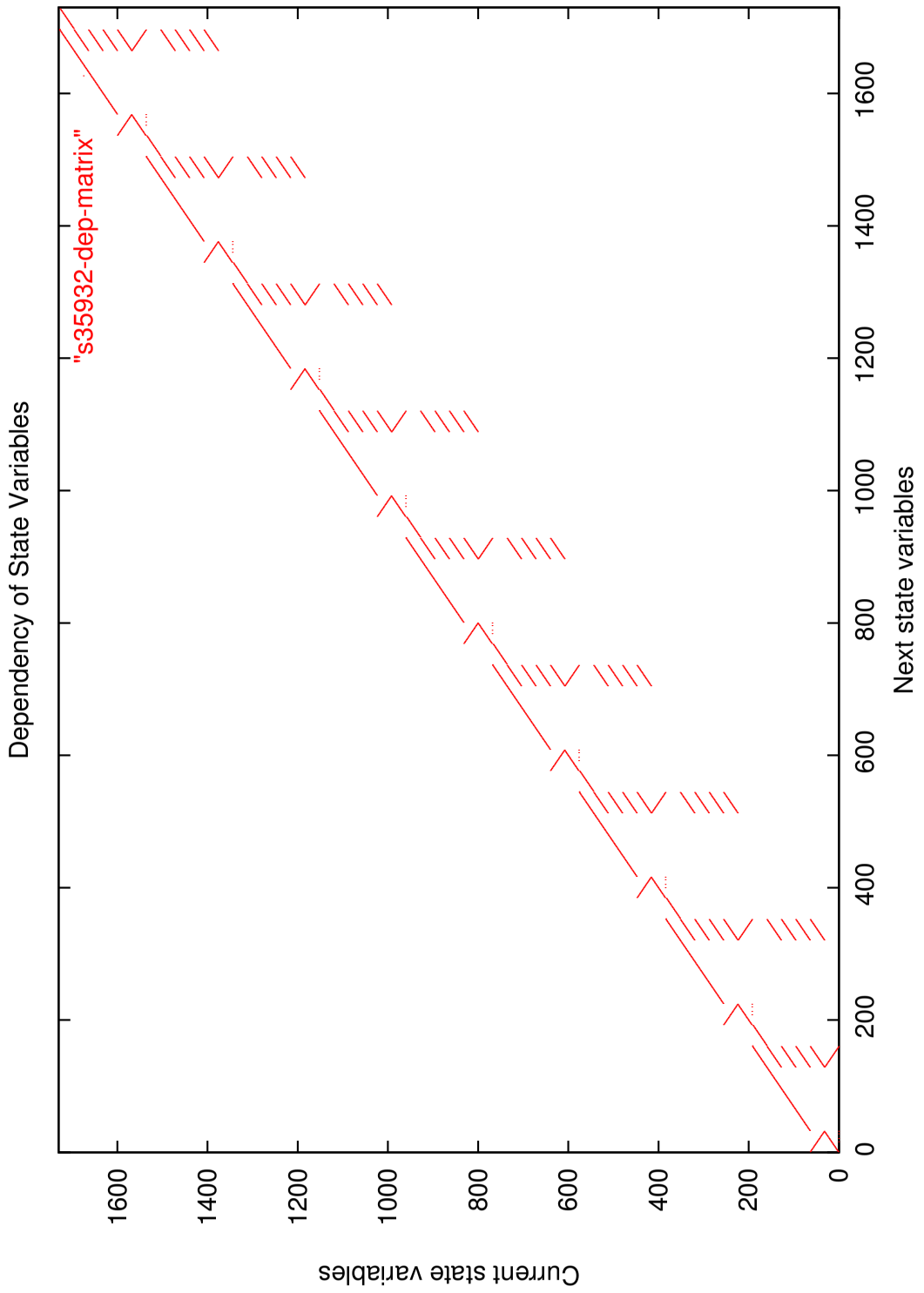


Figure 3.1: Dependency Plot of s35932.v

Chapter 4

Use of Input Constraining

In this chapter, we investigate the benefits of applying some constraints on the primary inputs in an attempt to tighten the over-approximation of the reachable set.

4.1 Basic Idea

Computing the reachable state space in a partitioned system involves the following steps:

- Perform an exact image computation of each projection, update the constraints on the state space suitably
- Iterate till a fixpoint has been reached (individually) for all projections. If yes, then combine the reachable states to get the final reachable set of states and terminate.

Whenever the states from different projections are combined or the constraints are updated, it is quite likely that some *undesired* state combinations get added which would otherwise never be a part of the reachable state space. In other words, states which can be obtained or reached due to a particular combination of PIs only should not be combined with states that get generated from other input sequences. In section 4.2, we illustrate this with an example.

4.2 A Motivating Example

Consider a small sequential circuit with three state variables $\{x_1,\ x_2,\ x_3\}$ and two Primary inputs $\{i_1,\ i_2\}$, partitioned into 2 overlapping projections: P1 and P2, with $\{x_1,\ x_2\}$ and $\{x_2,\ x_3\}$ respectively. Suppose the image computation starting from the initial state (0, 0, 0) is as shown in Table 4.1.

Cases 1 and 4 result in generating an unnecessary over-approximation by including states (000) and (001) respectively in their individual state spaces. Also, case 2 incorrectly generates the state (011) as being reachable. This particular generation can be avoided if we club cases 1 and 2 under one input constrain, namely $\overline{i_1}$ and cases 3 and 4 under another input constrain, namely i_1. Alternatively, if we club cases 1 and 3 in one partition (with the constraint as $\overline{i_2}$) and cases 2 and 4 in another (with the constraint as i_2), then the approximate state space generated after one application of image operation would exactly be the same as the one listed in the *Actual* column. So, it is seen even from this small example that choice of input constraints can

Table 4.1: Example of Input Constraining

One step of Image computation				
PIs (i1, i2)	P1 (x1, x2)	P2 (x2, x3)	Approx. space	Actual possible space
Case 1: 0 0	0 0	0 1	000, 001	001
Case 2: 0 1	0 1	0 0	011	–
Case 3: 1 0	1 0	1 1	111	111
Case 4: 1 1	1 1	0 0	000, 001	000

greatly affect the *quality* of the over-approximation. Having some insight of the structure of the circuit could be very useful in determining the input constraints. Choosing good constraints could be extremely helpful in tightening the over-approximation. Our experiments show that this approach indeed gives better traversal results i.e. a better over-approximation.

In the next section we present a formal explanation of this approach and the *modified* least fixpoint formulation of the approach.

4.3 Approach

Let S^k_{ij} be the set of states obtained from the i^{th} projection in the k^{th} using the j^{th} input constraint. Assume there are three projections and two input constraints. Assume all the sets are initialized to the same initial state of the circuit (in the zeroth step) . In the first step, the states obtained would be S^1_{11}, S^1_{21} and S^1_{31} using the first input constrain and S^1_{12}, S^1_{22} and S^1_{32} using the second input constrain. Next, we have to update the constraints carefully such that states that get generated from the two different input constraints do not get merged. The central idea is that in the k^{th} step of image computation,

$$(S^k_{11} \times S^k_{21} \times S^k_{31}) \cup (S^k_{12} \times S^k_{22} \times S^k_{32}) \subseteq (S^k_{11} \cup S^k_{12}) \times (S^k_{21} \cup S^k_{22}) \times (S^k_{31} \cup S^k_{32})$$

Further the constraints on the inputs should be chosen such that all possible input combinations are covered. For instance, if there are p constraints on the set of primary inputs (I):

$$C_1(\mathrm{I}), C_2(\mathrm{I}), \ldots C_p(\mathrm{I}),$$

then the following property should be satisfied by these constraints

$$\neg(C_1 \vee C_2 \vee \ldots \vee C_{p-1}) \rightarrow C_p$$

There will not be any improvement in the efficiency of MBM algorithm by using constraints on primary inputs. This is due to the fact that in MBM, before starting the traversal of the i^{th} sub-space, the $i-1^{th}$ sub-space has already been completely traversed and a fixpoint of the same has been obtained. So, there is no real way to keep track of input sequences to be merged in a way to disallow mixing of states which cannot get generated from the same input sequences.

We present the least fixpoint characterization [25] for this approach using RFBF startegy here
$R_{jl}^{-1} = I_j$

$$\mathrm{R}_{jl}^{i} = \begin{cases} \mu S.I_j \vee Img(S, \alpha_j(T(I,X,X')) \wedge C_l \wedge \bigvee_{l=1}^{l=p} R_{jl}^{i-1} \cdot \overset{r=k}{\underset{r\neq j, r=1}{}} R_{rl}^{i-1}) & \text{if } j = i \bmod k \\ \mathrm{R}_{jl}^{i-1} & \text{otherwise} \end{cases}$$

where

C_l is the l^{th} input constraint. Assume a total of p input constraints in the system.
The other strategies: TFBF and TMBM can be similarly characterized to accomodate the idea of input constraining.

4.4 Encoding using Labeled Reachability Expressions

Assume for ease of explanation, that we have only two partitions of the system. The encoding is done as shown.

$*(L_9 : ((\delta_{L_1} \wedge \delta_{L_3}) + (\delta_{L_2} \wedge \delta_{L_4})) \circ (L_5 : (((T_1)_{L_9} \downarrow \delta_{L_2} \downarrow C_1) + \delta_{L_1}) \circ (L_6 : ((T_1)_{L_9} \downarrow \delta_{L_1} \downarrow C_2) + \delta_{L_1}) \circ (L_7 : ((T_2)_{L_9} \downarrow \delta_{L_2} \downarrow C_1) + \delta_{L_2}) \circ (L_8 : ((T_2)_{L_9} \downarrow \delta_{L_1} \downarrow C_2) + \delta_{L_2}) \circ L_1 : \delta_{L_5} \circ L_2 : \delta_{L_6} \circ L_3 : \delta_{L_7} \circ L_4 : \delta_{L_8})$

RFBF uses as constraints the reachable sets computed in the previous iteration. So we would need four state components to keep track of the constraints to be imposed in every iteration. In addition in the presence of input constraints, we would also four more state components to keep track of the predicates R_{jl}^{i}'s to avoid unnecessary mixing of states. Let the state vector $\vec{\mathbf{S}} = (S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8, S_9)$, where the set of reachable states returned is $\gamma(S_1, S_3) \cup \gamma(S_2, S_4)$ upon convergence.

Encoding the use of input constraints in our framework is very easily done by adding the input constraint C_l as constraining the next state function of the cluster being traversed. The benefits of this strategy is due to the intelligent mixing of states to avoid some obvious over-approximations.

4.5 Identifying constraints on PIs

In order to identify constraints to be imposed on the PIs, we observed the dependency plot between the next state variables in the circuit and the PIs. Shown in Figure 4.1 is the dependency plot of the state variables in circuit s13207.v on its PIs. It is easy to see that there are certain groups of PIs which affect only a particular subset of state variables. Such plots can be indicative of the input constraints that can be imposed to tighten the overapproximation. For instance, from the figure 4.1, we see that a subset 5 PIs numbered from 23 to 27, affect only a very small part of the state space (State variables numbered 212 to 250). We used 4 PIs from this subset to generate the constraints and tried out all the 16 possible combinations of the PIs. Other subsets of PIs that can be used to generate constraints could also be PIs numbered 21 and 22 which also affect only a small subset of next state variables.

For all the circuits we ran our experiments on, we restricted ourselves to the use of constraining atmost 4 PIs from any subset of PIs chosen to impose constraints.

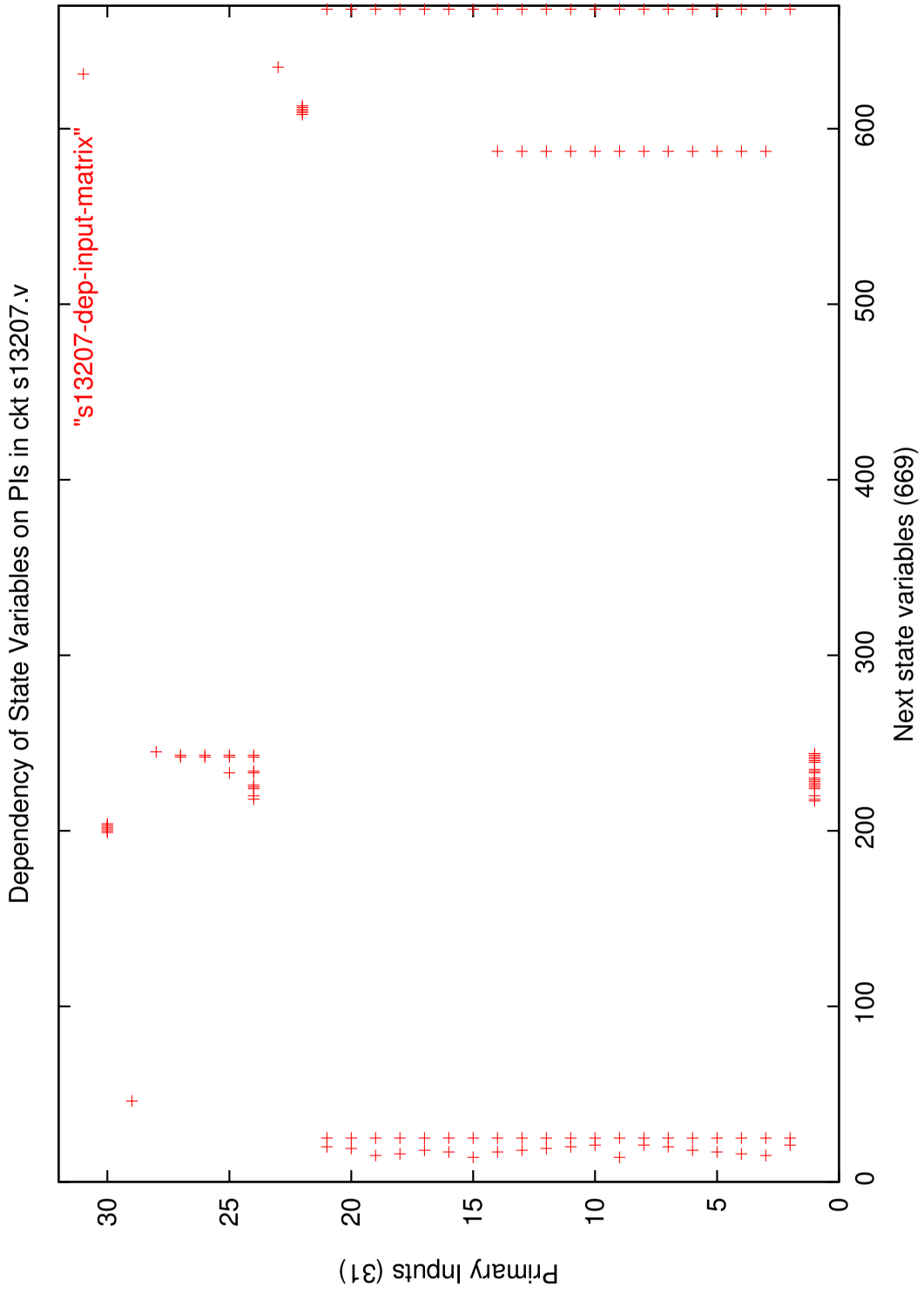


Figure 4.1: Dependency on PI in s13207.v

Table 4.2: **Traversal Results with Input Constraining**: Column Method denotes the traversal method being used. Columns E and F denote the sat-fr of the reachable space computed by the method (after using input constraining) with disjoint partitions and overlapping projections respectively. The columns iter-E and iter-F denote the number of iterations (number of TFBF iterations + number of MBM iterations) it has taken for the reachable set to converge using disjoint partitions and overlapping projections respectively.

Traversal Results with Input Constraining					
Ckt	Method	E	iter_E	F	iter_F
s13207.v	TMBM	2.56e-120	10+6	1.39e-125	10+7
s15850.v	TMBM	4.97e-108	10+6	3.32e-110	10+8
s38584.v	TMBM	4.26e-59	10+6	5.25e-61	10+6
s35932.v	TMBM	3.27e-75	10+5	2.93e-78	10+6

4.6 Results and Discussion

Columns E and F denote the sat-fr of the reachable space computed by the method (after using input constraining) with disjoint partitions and overlapping projections respectively. The columns iter-E and iter-F denote the number of iterations (number of TFBF iterations + number of MBM iterations) it has taken for the reachable set to converge using disjoint partitions and overlapping projections respectively.

Note that it takes longer to converge with the use of input constraints, but we also get orders of magnitude improvement in the *quality* of the approximated reachable set for all the circuits.

Chapter 5

Conclusion and Future Research Plans

5.1 Conclusion

Large digital systems are designed by a team of designers. Each designer has a very detailed understanding of the working of their own respective units and the way it interfaces with the other units of the system. The understanding of the other units is generally at a higher level of abstraction. So, even though the designer may not clear knowledge of the complete state of the system, they are confident about the local properties relevant to their own unit. This idea is central to the notion of using approximate methods for verifying large state spaces using partitioning. The key idea is to retain complete information about one partition and *just sufficient* information about other partitions.

In this work, we have built a generic BDD-based framework using Labeled Reachability Expressions which manipulate a state vector. This helps to unify the search techniques on large sequential circuits. We conjecture that Labeled Reachability Expressions can be embedded within a restricted version of modal μ-calculus (which does not have the negation and greatest fixpoint operators). It would help to test not only our traversal methods but also any other search technique which can be cast in the form of a Labeled Reachability Expression.

We have shown how this framework allows us to experiment with different approximate traversal algorithms and try out different traversal schemes. We have compared the results obtained from our framework with already published ones in [1, 9]. We have tried out using Input constraining as a refinement to tighten the overapproximation and we report orders of magnitude improvement in traversal results. We have evaluated all the techniques on a public domain benchmark suite of sequential circuits: ISCAS-89.

5.2 Future Research Plans

Our goal is to automate the process of reachability analysis using approximations with minimal manual intervention.

- **Abstraction using Encoding Decoding blocks**: We had discussed this framework in our first progress report [5]. This idea is based on the fact that there is enough statistical correlation between the state variables in large circuits. This is not surprising as it is not possible to build large circuits in a monolithic manner. Our initial observations about the

circuit structure by using dependency plots does show that some large circuits are indeed built by compositions of constituent smaller circuits in a well-defined manner. We are yet to fully investigate this approach.

- **Use SAT solvers to explore new ways to perform existential quantification**: A lot of research work is ongoing and is also available in literature. We want to use these techniques and improve upon them to be used as the backend Reachability Engine in our generic framework and also for the abstraction based on Encoding-Decoding strategy.

- **Using the generic framework**:

  1. We would like to incorporate the *prover* component in this framework which would allow us to determine the *quality* of the traversal technique we want to analyse.
  2. Also, leveraging other back-end support packages apart from NuSMV (which is currently being used) like some SAT solver could prove to be very beneficial in trying to extend this framework.
  3. It would be beneficial to add the abstractions based on Encoding-Decoding of the state variables also in the framework and analyse the performance of the traversal schedules.
  4. Build a completely automated integrated tool environment which would have enough capability to decide what would be good partitioning strategies for the circuits, decides between diferent traversal schedules, scales up well for large circuits, and improving the flexibility of the framework by also providing ease of use. Any user of the framework should only be required to do the appropriate translation of her/his search technique as a Labeled Reachability Expression and all other details could be kept hidden from her/him.

Bibliography

[1] H. Cho, Hatchel G. D., E. Macii, B. Plessier, and F. Somenzi. Algorithms for Approximate FSM Traversal. In *Proc. of Design Automation Conference*, pages 25 – 30, June 1993.

[2] H. Cho, Hatchel G. D., E. Macii, B. Plessier, and F. Somenzi. Algorithms for Approximate FSM Traversal Based on State Space Decomposition. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 15, pages 1465 – 1478, December 1996.

[3] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proc. of Conference in Logic in Computer Science*, pages 428–439, 1990.

[4] O. Coudert and J. Madre. A Unified Framework for the Formal Verification of Sequential Circuits. In *IEEE International Conference on Computer-Aided Design*, pages 126–129, November 1990.

[5] Seetha Jayasankar. Reachability Analysis of Large Sequential Circuits. Technical report, IIT Bombay, 2005. First Progress Seminar Report, Also available at http://www.cfdvs.iitb.ac.in/∼seetha/aps1.pdf.

[6] R. E. Bryant. Graph-based algorithms for boolean function manipulation. In *IEEE Transactions on Computers*, volume 35, pages 677–691, 1986.

[7] Ashutosh Trivedi. Techniques in Symbolic Model Checking. Master's thesis, IIT Bombay, 2003.

[8] Saurabh Joshi. Symbolic Model Checking of Large Sequential Circuits. Master's thesis, Dept. of C. S. E., IIT Bombay, 2006.

[9] S. G. Govindaraju, D. L. Dill, A. J. Hu, and M. A. Horowitz. Approximate Reachability with BDDs using Overlapping Projections. In *Proc. of Design Automation Conference*, pages 451–456, 1998.

[10] S. G. Govindaraju, D. L. Dill, and J. P. Bergmann. Improved Approximate Reachability using Auxillary State Variables. In *Proc. of Design Automation Conference*, pages 251–255, 1999.

[11] K. L. McMillan. A Conjunctively Decomposed Boolean Representation for Symbolic Model Checking. In *Proc. of Computer-Aided Verification*, pages 13–25, 1990.

[12] Y. Hong, P. A. Beerel, J. A. Burch, and K. L. McMillan. Sibling-Substitution-Based BDD Minimization Using Don't Cares. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 19, pages 44–55, January 2000.

[13] G. Cabodi and P. Camurati. Symbolic FSM Traversals Based on the Transition Relation. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 16, pages 448–457, May 1997.

[14] R. Ranjan, A. Aziz, C. Plessier, C. Pixley, and R Brayton. Efficient BDD Algorithms for FSM Synthesis and Verification. In *Proc. of the International Workshop on Logic Synthesis*, pages 190–197, 1995.

[15] In-Ho Moon, James H. Kukula, Kavitha Ravi, and Fabio Somenzi. To Split or to Conjoin: The Question in Image Computation. In *Proc. of Design Automation Conference*, pages 271–275, 2000.

[16] G. Cabodi, P. Camurati, and S. Quer. Improving Symbolic Traversals by means of Activity Profiles. In *Proc. of Design Automation Conference*, pages 220–225, 1999.

[17] Kavitha Ravi and Fabio Somenzi. Hints to Accelerate Symbolic Traversal. In *CHARME'99*, pages 250–266, 1999.

[18] Kavitha Ravi and Fabio Somenzi. High Density Reachability Analysis. In *Proc. of International Conference on Computer-Aided Design*, pages 154–158, 1995.

[19] G. Cabodi, P. Camurati, and S. Quer. Improved Reachability Analysis of Large Finite State Machines. In *Proc. of Internal Conference on Computer-Aided Design*, pages 354–360, November 1996.

[20] P. A. Abdulla, P. Bjesse, and N. Een. Symbolic Reachability Analysis Based on SAT-Solvers. In *Proc. of the 6th International Conference on Tools and Algorithms for Construction and Analysis of Systems: Held as Part of the European Joint Conferences on the Theory and Practice of Software (ETAPS)*, pages 411–425, 2000.

[21] G. Cabodi, S. Nocco, and S. Quer. Improving SAT-based Bounded Model Checking by Means of BDD-based Approximate Traversals. In *DATE '03: Proceedings of the conference on Design, Automation and Test in Europe*, pages 898–903, 2003.

[22] Dina Thomas, Supratik Chakraborty, and Paritosh Pandya. NuSMV-DP: User's Manual. Technical report, CFDVS, IIT Bombay, 2005. http://www.cfdvs.iitb.ac.in/∼dina/NuSMVDP.

[23] Dina Thomas, Supratik Chakraborty, and Paritosh Pandya. Efficient Guided Symbolic Reachability using Reachability Expressions. Technical Report TR-06-19, CFDVS, IIT Bombay, 2006.

[24] Varun Kanade. Guided Symbolic Reachability using Partitioning. Master's thesis, Dept. of C. S. E., IIT Bombay, 2006.

[25] In-Ho Moon, James Kukula, Tom Shiple, and Fabio Somenzi. Least fixpoint approximations for reachability analysis. In *ICCAD '99: Proceedings of the 1999 IEEE/ACM international conference on Computer-aided design*, pages 41–44, 1999.

[26] David Long. The Design of a Cache-Friendly BDD Library. In *Proc. of Internal Conference on Computer-Aided Design*, pages 639–645, 1998.