

LREs: A Framework for Guiding Symbolic Reachability Analysis

Seetha Jayasankar and Supratik Chakraborty

Indian Institute of Technology, Bombay, India
seetha@cse.iitb.ac.in, supratik@cse.iitb.ac.in

Abstract. Symbolic reachability analysis of large sequential circuits is a computationally hard problem. Approximate techniques tradeoff precision for scalability by devising new ways of computing approximate images efficiently. Each new technique, however, requires non-trivial work to be implemented in frameworks like NuSMV or VIS. In addition, the soundness and completeness of a new technique is often left unverified. In this report, we propose Labeled Reachability Expressions (LRE) as a generic framework for expressing, reasoning about and implementing a large family of symbolic reachability techniques, including exact and approximate ones. We show how a Boolean decomposition of the transition relation can be used to discover and optimize LREs that capture the spirit of the decomposition. We discuss properties of LREs that allow us to reason about their correctness, and also permit comparison of alternative techniques expressed as LREs. We have built a BDD-based tool on top of the public-domain symbolic model checker NuSMV, that can interpret LREs to give custom symbolic reachability analyzers. We illustrate the effectiveness of our approach by implementing state-of-the-art approximate reachability algorithms and a few new ones simply by feeding appropriate LREs to our tool.

1 Introduction

Finding the set of reachable states of a circuit, or *reachability analysis*, finds diverse applications in VLSI CAD, including synthesis, optimization, formal verification and testing. Existing symbolic reachability analyzers either use BDD [6] or SAT based techniques [1, 11], or a combination of both [19, 10]. BDD-based techniques [12, 21, 33, 27, 18] are eventually limited by memory in practice. SAT-based algorithms [2, 5, 34] generally scale much better for shallow bug-hunting in large industrial circuits, although optimized BDD-based techniques have also been reported to be effective for shallow bug-hunting [7]. Recent research has also shown how SAT solvers can be used to compute interpolants for approximate reachability analysis of large circuits [25]. As the number of state variables increases, the computational difficulty of searching the exact reachable state space increases roughly exponentially. This makes exact reachability analysis impractical for large circuits. Consequently, several approximate techniques have been proposed for searching a circuit's state space. Successful search techniques often incorporate a designer's intuition about how a circuit's state transition behaviour can be thought of as being composed of state transition behaviours of smaller components. In this report, we present a framework for discovering, implementing and reasoning about a large class of such search techniques, focussing on BDD based techniques.

A sequential circuit is a 6-tuple $B = (I, X, O, S_0, \mathbf{T}, \Delta)$ where I is the set of primary inputs, X is the set of present state variables, O is the set of primary outputs, S_0 is the initial set of states, \mathbf{T} is a $|X|$ -dimensional vector of next-state functions, and Δ is a $|O|$ -dimensional vector of output functions. The i^{th} component of \mathbf{T} gives the next state function $T_i(I, X)$ for X_i . For purposes of our discussion, the set of primary outputs are of no concern, and henceforth we will ignore the set of primary outputs and Δ .

Given a sequential circuit, its transition relation is given by $Y(I, X, X') = \bigwedge_{j=1}^{|X|} x'_j \leftrightarrow T_j(I, X)$, where x'_j denotes the value of state variable $x \in X$ in the next state. Let $S(X)$ be the characteristic predicate

of a set of states. The image of $S(X)$ under $\Upsilon(I, X, X')$, denoted $Img(S(X), \Upsilon(I, X, X'))$, is computed as $\lambda X'. \exists I, X (\Upsilon(I, X, X') \wedge S(X))$. Starting from an initial set of states $S_0(X)$, the exact set of reachable states of B can be computed as the least fixpoint $\mu S. (S_0(X) \vee Img(S(X), \Upsilon(I, X, X')))$. Unfortunately, in most practical circuits, the state variable count runs into thousands or more, rendering the computation of $Img(S(X), \Upsilon(I, X, X'))$ difficult in practice. In addition, the total number of steps required to compute the above fixpoint can also be large, leading to computational bottlenecks in computing the exact reachable state set.

A variety of reachability analysis techniques for sequential circuits work by decomposing the transition relation $\Upsilon(I, X, X')$ either disjunctively or conjunctively. Such decompositions have been used primarily for two purposes: (i) to optimize the image computation step of a single sequential machine [21, 33, 8, 27, 9, 20, 16, 28–30, 26], and (ii) to decompose a single machine into multiple submachines, which are then analyzed independently or in an interleaved manner according to a suitable schedule [26, 30, 29, 12, 18, 17]. In this report, we primarily focus on submachine based traversal techniques using BDDs.

Submachine-based techniques that project states on a subset of state and auxiliary variables [12, 18, 17] effectively exploit the conjunctive form of $\Upsilon(I, X, X')$. Typically, each projection is represented separately, and their conjunction gives the best overapproximation of the reachable state set. Cho *et al.* presented an automated state space decomposition method in [13] that produces submachines with disjoint state variables. These are then traversed in carefully interleaved ways to compute over-approximations of the reachable state space [12, 13]. Two classes of approximate methods were introduced in these works: *Machine By Machine* (MBM) and *Frame By Frame* (FBF). Govindaraju *et al.* [18, 17] subsequently extended these techniques by allowing overlapping projections in which a state variable can belong to multiple submachines. This led to significant improvements in accuracy. We will use these traversal techniques as examples later in our discussion to demonstrate the effectiveness of our framework.

Our work extends and builds on Thomas *et al.*'s work [32] in which reachability expressions were introduced as a framework for expressing and reasoning about search methods for asynchronous systems. A significant limitation of reachability expressions is their inability to simultaneously express multiple reachability computations over different but possibly correlated state sets. Yet another limitation is their inability to describe complements of state sets. This makes it difficult to express several interesting reachability techniques that involve traversal of multiple interacting submachines. In this report, we propose Labeled Reachability Expressions (LRE) to overcome these limitations, and significantly expand the applicability of the basic reachability expressions formalism.

Our primary contribution is in presenting a richer framework than that proposed in [32] for expressing a diverse range of search strategies, including those that are not amenable to any natural description using reachability expressions. This allows us to discover, express, reason about and easily implement a large family of symbolic reachability techniques, both approximate and exact, using either a single machine or multiple submachines. As an application, we show how a boolean decomposition of a circuit's transition relation can guide the discovery of approximate and exact reachability analysis techniques using LREs. This not only leads to well-known techniques like MBM but also throws open the possibility of new approximate state space traversal techniques, one of which is discussed later. We present some properties of LREs (not all of which coincide with properties of reachability expressions as in [32]) that allow us to reason about the correctness of reachability analysis techniques, and also permit comparison of alternative techniques. We have built a BDD-based tool on top of the public-domain symbolic model checker NuSMV, to give custom symbolic reachability analyzers. We present our initial experimental findings to demonstrate the effectiveness of our approach.

2 Definitions

We consider synchronous sequential circuits composed of combinational gates and flip-flops. All flip-flops are controlled by the same clock. We assume that all flip-flops can be reset to a given state.

Definition 1. A Finite State machine (FSM) representation of a sequential circuit M , is a Mealy machine given by

$$B = (I, O, X, Q, S_0, \mathbf{T}, \Delta) \quad (1)$$

where

- $I = \{i_1, i_2, \dots, i_q\}$ is a set of boolean valued Primary Inputs
- $O = \{o_1, o_2, \dots, o_m\}$ is a set of boolean valued Primary Outputs
- $X = \{x_1, x_2, \dots, x_l\}$ is a set of boolean valued (current) State Variables.
- A state of machine M is defined by the boolean values of the state variables x_1, \dots, x_l . Let Q be the finite set of states of the machine B
- S_0 is the set of initial states of the machine B
- \mathbf{T} is a vector of l next state functions such that $\forall i \in \{1, \dots, l\}$, the i^{th} component of \mathbf{T} denoted $T_i : \{0, 1\}^q \times \{0, 1\}^l \rightarrow \{0, 1\}$ is the next state function for the state variable x_i , denoted $x'_i \leftrightarrow T_i(I, X)$ where x'_i is the next state variable corresponding to x_i .
- Δ is a vector of m output functions such that $\forall i \in \{1, \dots, m\}$, the i^{th} component of Δ denoted $\Delta_i : \{0, 1\}^q \times \{0, 1\}^l \rightarrow \{0, 1\}$ is the output function for the Primary Output o_i .

For our application, we ignore the vector O and the output function vector Δ . Henceforth, we will use a 5-tuple FSM representation

$$B = (I, X, Q, S_0, \mathbf{T}) \quad (2)$$

Example 1. Consider $B = ([i_1, i_2], [x_1, x_2, x_3, x_4], Q, \{-x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \neg x_4\}, [i_1 \wedge x_1 + x_2, \neg x_2 + x_3 \wedge \neg x_4, i_1 \wedge i_2 + \neg x_1, \neg x_4])$. B represents a sequential circuit with two Primary Inputs: i_1 , and i_2 and has four State Variables: x_1, x_2, x_3 and x_4 . It has a single initial state $\neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \neg x_4$ which is the all-zero state. The last tuple is a vector of next state functions: \mathbf{T} having one entry for each next state function of B .

$$\begin{aligned} x'_1 &= T_1(I, X) = i_1 \wedge x_1 + x_2 \\ x'_2 &= T_2(I, X) = \neg x_2 + x_3 \wedge \neg x_4 \\ x'_3 &= T_3(I, X) = i_1 \wedge i_2 + \neg x_1 \\ x'_4 &= T_4(I, X) = \neg x_4 \end{aligned}$$

Definition 2. *Boolean Function Vector (BFV):* Any multiple output boolean function can be represented as a vector of single output boolean functions which is called a Boolean Function Vector (BFV).

Example 2. T in Definition 2 is a BFV of the next state functions of a sequential circuit.

Definition 3. *Sets and Characteristic Functions:* Let S be a set of states such that $S \subseteq Q$. The characteristic function of S is the function

$$S(X) : Q \rightarrow \{0, 1\} \quad \text{defined by } S(X)(x) = 1 \text{ if } x \in S, S(X)(x) = 0 \text{ otherwise}$$

A characteristic function is simply a functional representation of a subset of a set.

Note: Given a sequential circuit with l state variables, characteristic function of the set of all states i.e. the universe $\{0, 1\}^l$ is 1 and characteristic function of \emptyset is 0.

Definition 4. *Support of a boolean function:* Let $f : \{0,1\}^b \rightarrow \{0,1\}$ denote a non-constant Boolean function of b variables t_1, \dots, t_b . We say that f depends on t_i iff $f|_{t_i} \neq f|_{\neg t_i}$. We define the support of f , denoted $Sup(f)$, as the set of Boolean variables f depends on.

Example 3. Consider a boolean function $f = x_1 \neg x_2 + \neg x_1 x_3$. Then, $Sup(f) = \{x_1, x_2, x_3\}$

Definition 5. *State Transition Diagram (STD):* Given an FSM $B = (I, X, Q, S_0, \mathbf{T})$, its State Transition Diagram is a graph $G = (V, V_0, E)$, where

- $V = Q$ is the set of vertices of the graph
- $V_0 \subseteq V$ is the set of start vertices of the graph, $V_0 = S_0$
- $E = V \times V$ is the set of edges, $\forall s_1, s_2 \in Q, (s_1, s_2) \in E$ if $\mathbf{T}(i, s_1) = s_2$ where $i \in I$

Example 4. For the sequential circuit in Example 1, the corresponding STD is shown in Figure 4.

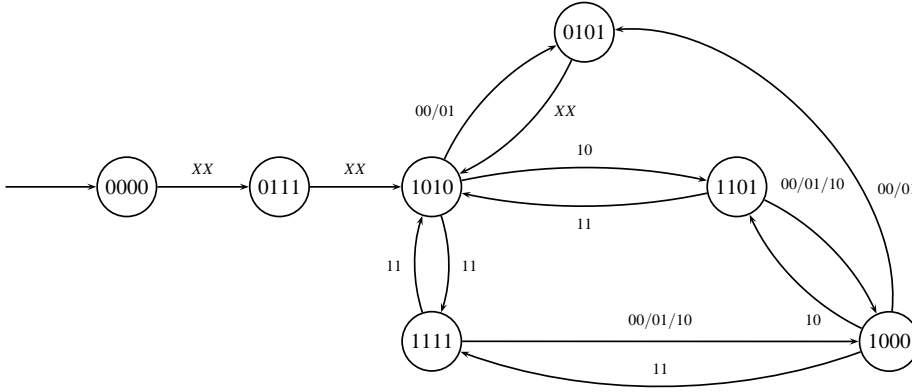


Fig. 1: STD for Example 1

Definition 6. *Reachable States:* Given an FSM representation $B = (I, X, Q, S_0, \mathbf{T})$ of a sequential circuit, the set of states that can be reached by application of all input sequences from the set of initial states (S_0) are called *reachable*. *Reachability* R can be defined as a relation $R \subseteq Q \times Q$ such that

$$\begin{aligned} \forall s \in Q, (s, s) \in R \wedge \\ \forall s_1, s_2, s_3 \in Q, ((s_1, s_2) \in R) \wedge (s_2, s_3) \in R \rightarrow (s_1, s_3) \in R \end{aligned}$$

Example 5. For the sequential circuit in Example 1, the set of reachable states is $\{0000, 0101, 0111, 1000, 1010, 1101, 1111\}$.

Reachability Analysis is a decision problem which checks if a specified set of states is reachable or not. Using the STD representation, this reduces to being the problem of traversal of the STD from its initial states to determine which states are reachable. But, such an explicit graph representation may not be practical for large sequential circuits with a large number of flip-flops.

Definition 7. *BDD (Binary Decision Diagram):* A BDD (proposed by Lee [23] and later Akers in [3]) is a data structure used to represent a Boolean function. A Boolean function can be represented as a rooted, directed, acyclic graph, which consists of decision nodes and two terminal nodes called 0-terminal and 1-terminal. Each decision node is labeled by a Boolean variable and has two child nodes called low child and high child. The edge from a node to a low (high) child represents an assignment of the variable to 0 (1). Such a BDD is called *ordered* if different variables appear in the same order on all paths from the root. In this report we use the term BDD to refer to OBDD.

Example 6. Shown in Figure 6 is a BDD for the next state equation $x'_1 = i_1x_1 + x_2$. We use the variable order $i_1 < i_2 < x_1 < x_2 < x_3 < x_4$.

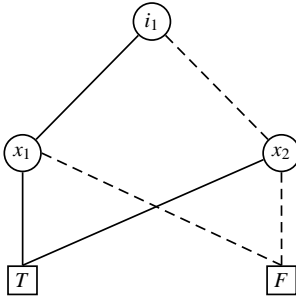


Fig. 2: BDD for Example 1

Definition 8. *Monotonic Function:* Given a domain H , a function $f : H \rightarrow H$ is said to be monotone iff for all $x, y \in H$, $x \leq y$ iff $f(x) \leq f(y)$.

Definition 9. *Transition Relation representation:* Given a BFV $\mathbf{T}(I, X)$ of length l , where each $\forall i, T_i : \{0, 1\}^q \times \{0, 1\}^l \rightarrow \{0, 1\}$, the corresponding transition relation of the sequential circuit is represented as $\Delta : \{0, 1\}^q \times \{0, 1\}^l \times \{0, 1\}^l \rightarrow \{0, 1\}$, i.e.

$$\Delta(I, X, X') = \bigwedge_{1 \leq j \leq l} (x'_j \leftrightarrow T_j(I, X))$$

Definition 10. *Given a set of state variables X , characteristic function $S(X)$ of a set of states S and a set of k projections of the state variables $\{\pi_1, \pi_2, \dots, \pi_k\}$, the **projection operator** α_i projects $S(X)$ onto π_i .*

$$\begin{aligned} \forall i \in \{1, 2, \dots, k\} \quad \alpha_i(S(X)) &= \exists X \setminus \pi_i S(X) \\ &= \exists \overline{\pi_i} S(X) \end{aligned}$$

Example 7. Let $X = \{x_1, x_2, x_3\}$, $S(X) = \neg x_1 \vee x_2 \wedge \neg x_3$ and $P = \{\pi_1, \pi_2\}$ where $\pi_1 = \{x_1, x_2\}$ and $\pi_2 = \{x_2, x_3\}$. Then

$$\begin{aligned} \alpha_1(S(X)) &= \alpha_1(\neg x_1 \vee x_2 \wedge \neg x_3) \\ &= \exists x_3 (\neg x_1 \vee x_2 \wedge \neg x_3) \\ &= \neg x_1 \vee x_2 \end{aligned}$$

and,

$$\begin{aligned}\alpha_2(S(X)) &= \alpha_2(\neg x_1 \vee x_2 \wedge \neg x_3) \\ &= \exists x_1 (\neg x_1 \vee x_2 \wedge \neg x_3) \\ &= \mathbf{1}\end{aligned}$$

Definition 11. Given a set of state variables X , characteristic function $S(X)$ of a set of states S and a set of k projections of the state variables $\{\pi_1, \pi_2, \dots, \pi_k\}$, the **abstraction** operator α projects $S(X)$ onto the various π_j 's as

$$\alpha(S(X)) = \left(\alpha_1(S(X)), \dots, \alpha_k(S(X)) \right)$$

Definition 12. Given a characteristic function $S(X)$ of a set of states S and a set of k projections of the state variables, the **concretization** operator γ conjoins all the k projections

$$\gamma\left(\alpha_1(S(X)), \alpha_2(S(X)), \dots, \alpha_k(S(X))\right) = \bigwedge_{i=1}^{i=k} \alpha_i(S(X))$$

Example 8. From Example 7 we know that $\alpha_1(S(X)) = \neg x_1 \vee x_2$ and $\alpha_2(S(X)) = \mathbf{1}$. So,

$$\begin{aligned}\gamma(\alpha_1(S(X)), \alpha_2(S(X))) &= \alpha_1(S(X)) \wedge \alpha_2(S(X)) \\ &= (\neg x_1 \vee x_2) \wedge \mathbf{1} \\ &= \neg x_1 \vee x_2\end{aligned}$$

3 Notations

We list the notations used in this report.

1. I, X and X' denote sets of boolean valued Primary Inputs of cardinality q , boolean valued Current State Variables of cardinality l and boolean valued Next State Variables of cardinality l respectively.
2. Y denotes a set of free variables.
3. S, R, \dots denote sets of states such that $S, R \subseteq Q$.
4. $S(X)$ denotes the characteristic function of the set of states S .
5. $\mathbf{T}(I, X)$ denotes a BFV of length l
6. $I' = I \cup Y$ is a set obtained by the union of the sets I and Y .
7. $\mathbf{1}$ denotes *tautology*.
8. k denotes the number of projections of the set of state variables X .

4 Approximations using projections of state variables

Lemma 1. Let $S_1(X), S_2(X)$ be characteristic functions of sets of states S_1 and S_2 respectively. Then $\forall i, S_1(X) \rightarrow S_2(X) \Rightarrow \alpha_i(S_1(X)) \rightarrow \alpha_i(S_2(X))$.

Proof: Follows from the monotonicity of the projection operator α_i which is based on existential quantification. Hence $\forall i, \alpha_i$ is a monotone function.

Example 9. Let $S_1(X) = \neg x_1$ and $S_2(X) = \neg x_1 \vee x_2 \wedge \neg x_3$. Then,

$$\alpha_1(S_1(X)) = \exists x_3(\neg x_1) = \neg x_1$$

$$\alpha_1(S_2(X)) = \exists x_3(\neg x_1 \vee x_2 \wedge \neg x_3) = \neg x_1 \vee x_2$$

$$\text{Hence } \alpha_1(S_1(X)) \rightarrow \alpha_1(S_2(X))$$

$$\alpha_2(S_1(X)) = \exists x_1(\neg x_1) = \mathbf{1}$$

$$\alpha_2(S_2(X)) = \exists x_1(\neg x_1 \vee x_2 \wedge \neg x_3) = \mathbf{1}$$

$$\text{Hence } \alpha_2(S_1(X)) \rightarrow \alpha_2(S_2(X))$$

Lemma 2. Let $S_1(X), S_2(X)$ be characteristic functions of sets of states S_1 and S_2 respectively. Then $\forall i$, $\alpha_i(S_1(X) \vee S_2(X)) = \alpha_i(S_1(X)) \vee \alpha_i(S_2(X))$.

Proof: Projection is based on existential quantification which distributes over \vee . Hence, the projection operator also distributes over \vee .

Example 10. From Example 9, we know that

$$\begin{aligned} \alpha_1(S_1(X) \vee S_2(X)) &= \alpha_1(\neg x_1 \vee \neg x_1 \vee x_2 \wedge \neg x_3) \\ &= \alpha_1(\neg x_1 \vee x_2 \wedge \neg x_3) \\ &= \exists x_3(\neg x_1 \vee x_2 \wedge \neg x_3) \\ &= \neg x_1 \vee x_2 \end{aligned}$$

$$\begin{aligned} \alpha_1(S_1(X)) \vee \alpha_1(S_2(X)) &= \alpha_1(\neg x_1) \vee \alpha_1(\neg x_1 \vee x_2 \wedge \neg x_3) \\ &= \exists x_3(\neg x_1) \vee \exists x_3(\neg x_1 \vee x_2 \wedge \neg x_3) \\ &= \neg x_1 \vee \neg x_1 \vee x_2 \\ &= \neg x_1 \vee x_2 \end{aligned}$$

$$\text{Hence } \alpha_1(S_1(X) \vee S_2(X)) = \alpha_1(S_1(X)) \vee \alpha_1(S_2(X))$$

Lemma 3. Let $S(X)$ be the characteristic function of a set of states S , and $P = \{\pi_1, \pi_2, \dots, \pi_k\}$ be a set of projections of the state variables. Then, $\forall i \in \{1, 2, \dots, k\}$,

$$S(X) \rightarrow \alpha_i(S(X))$$

Proof: Follows from the fact of predicate logic that for any function $\phi(A)$, $\phi \rightarrow \exists x \phi$ where $x \subseteq A$ and also that the projection operator α_i is based on existential quantification.

Lemma 4. Let $S(X)$ be the characteristic function of a set of states S , and $P = \{\pi_1, \pi_2, \dots, \pi_k\}$ be a set of projections of the state variables. Then, $\forall i \in \{1, 2, \dots, k\}$,

$$\begin{aligned} S(X) &\rightarrow \gamma(\alpha(S(X))) \\ &= \gamma(\alpha_1(S(X)), \dots, \alpha_k(S(X))) \end{aligned}$$

Proof: From Lemma 3, we know that $\forall i \in \{1, 2, \dots, k\}, S(X) \rightarrow \alpha_i(S(X))$. Using the fact from propositional logic that $\wedge_i(r \rightarrow q_j)$ implies that $r \rightarrow \wedge_i q_i$. Hence, $S(X) \rightarrow \gamma(\alpha_1(S(X)), \dots, \alpha_k(S(X)))$.

Note: From Lemma 4, we can conclude that projecting a set of states $S(X)$ onto a collection of projections using the α_i operator and then concretizing the projections using the γ operator results in an over-approximation.

Lemma 5. *Let S be any set of states. Then, $P = (S, \subseteq)$ forms a complete lattice under the join operation of set union and under the meet operation of set intersection.*

Definition 13. *Let $S(X)$ be the characteristic function of a set of states S and $\mathbf{T}(I, X)$ be a BFV of length l , then the image of $S(X)$ under $\mathbf{T}(I, X)$ is computed as*

$$\text{Img}(S(X), \mathbf{T}(I, X)) = \lambda X' \exists I, \exists X \left(S(X) \wedge \bigwedge_{j=1}^{j=l} (x'_j \leftrightarrow T_j(I, X)) \right)$$

This produces a predicate with support in X' . The resulting predicate is $\mathbf{1}$ iff $\forall j \in \{1, \dots, l\}$ x'_j is in the image of $S(X)$ under $T_j(I, X)$.

Definition 14. *Let $S(X)$ be the characteristic function of a set of states S and $\Delta(I, X, X')$ be the transition relation of the sequential circuit, then the image of $S(X)$ under $\Delta(I, X, X')$ is computed as*

$$\text{Img}_{TR}(S(X), \Delta(I, X, X')) = \exists I, \exists X \left(S(X) \wedge \bigwedge_{j=1}^{j=l} (x'_j \leftrightarrow T_j(I, X)) \right)$$

This produces a predicate with support in X' which is true iff $\forall j \in \{1, \dots, l\}$ x'_j is in the image of $S(X)$ under $\Delta(I, X, X')$.

Lemma 6. *Let $S(X)$ be the characteristic function of a set of states S , $\mathbf{T}(I, X)$ be a BFV of length l , and $\Delta(I, X, X')$ be the transition relation equivalent of $\mathbf{T}(I, X)$, then*

$$\text{Img}(S(X), \mathbf{T}(I, X)) = \text{Img}_{TR}(S(X), \Delta(I, X, X'))$$

This follows from the definitions of the Img and Img_{TR} operations.

Lemma 7. *Let $\mathbf{T}(I, X)$ be the BFV for a sequential circuit and $S_1(X)$ and $S_2(X)$ be characteristic functions of sets of states S_1 and S_2 respectively such that $S_1(X) \rightarrow S_2(X)$. Then, $\text{Img}(S_1(X), \mathbf{T}(I, X)) \rightarrow \text{Img}(S_2(X), \mathbf{T}(I, X))$. In other words, Image computation is a monotone function.*

Proof: The image of $S_1(X)$ under $\mathbf{T}(I, X)$ is defined as

$$\begin{aligned} \text{Img}(S_1(X), \mathbf{T}(I, X)) &= \lambda X' \exists I, \exists X S_1(X) \wedge (X' = \mathbf{T}(I, X)) \\ &\rightarrow \lambda X' \exists I, \exists X S_2(X) \wedge (X' = \mathbf{T}(I, X)) \quad (\text{Since } S_1(X) \rightarrow S_2(X)) \\ &= \text{Img}(S_2(X), \mathbf{T}(I, X)) \quad \text{By definition of image of } S_2(X) \text{ under } \mathbf{T}(I, X). \end{aligned}$$

Hence, Image computation is a monotone function.

Theorem 1. *The set of reachable states of a sequential circuit can be computed by BFS(S_0, \mathbf{T}) described in Algorithm 1 [33].*

Proof: It is a well known result and a proof is in [33].

Algorithm 1 Symbolic BFS Traversal $BFS(S_0, \mathbf{T})$

```

1:  $reached = S_0$ 
2: loop
3:    $old\_reached = reached$ 
4:    $reached = reached \vee \text{Img}(reached, \mathbf{T})$ 
5:   if  $old\_reached = reached$  then
6:      $\text{return } reached$ 
7:   end if
8: end loop

```

Definition 15. Given a projection π_p of the state variables, characteristic function $S(X)$ of a set of states S , and a BFV $\mathbf{T}(I, X)$ of length l , we define $[\mathbf{T}(I, X)]_{\pi_p}$ to be another BFV of length l such that its j^{th} element is

$$T_j(I, X) \quad \text{if } x_j \in \pi_p$$

$$y_j \quad \text{If } x_j \notin \pi_p, \text{ where } y_j \in Y \text{ is a free variable}$$

Definition 16. Given a projection π_i of the state variables, characteristic function $S(X)$ of a set of states S , and a BFV $[\mathbf{T}(I, X)]_{\pi_i}$, we define

$$\text{Img}(S(X), [\mathbf{T}(I, X)]_{\pi_i}) = \lambda X' \exists I \exists Y \exists X \left(S(X) \wedge \bigwedge_{\{j|j \in \pi_i\}} (x'_j \leftrightarrow T_j(I, X)) \wedge \bigwedge_{\{j|j \notin \pi_i\}} (x'_j \leftrightarrow y_j) \right)$$

Lemma 8. Given a projection π_i of the state variables, characteristic function $S(X)$ of a set of states S , and a BFV $\mathbf{T}(I, X)$ of length l ,

$$\text{Img}(S(X), \mathbf{T}(I, X)) \rightarrow \text{Img}(S(X), [\mathbf{T}(I, X)]_{\pi_i})$$

Proof:

$$\begin{aligned}
\text{Img}(S(X), \mathbf{T}(I, X)) &= \lambda X' \exists \{i \in I, x \in X\} \left(S(X) \wedge \bigwedge_{j=1}^{j=l} (x'_j \leftrightarrow T_j(I, X)) \right) \\
&\text{From definition} \\
&= \lambda X' \exists \{x \in X\} \left(S(X) \wedge \exists i \in I \left[\bigwedge_{j=1}^{j=l} (x'_j \leftrightarrow T_j(I, X)) \right] \right) \\
&\text{Since } X \cap I = \emptyset \\
&= \lambda X' \exists \{x \in X\} \left(S(X) \wedge \exists i \in I \left[\bigwedge_{\{j|x_j \in \pi_i\}} (x'_j \leftrightarrow T_j(I, X)) \wedge \bigwedge_{\{j|x_j \notin \pi_i\}} (x'_j \leftrightarrow T_j(I, X)) \right] \right) \\
&\text{From definition} \\
&\rightarrow \lambda X' \exists \{x \in X\} \left(S(X) \wedge \exists i \in I \left(\bigwedge_{\{j|x_j \in \pi_i\}} (x'_j \leftrightarrow T_j(I, X)) \right) \wedge \right. \\
&\quad \left. \exists i \in I \left(\bigwedge_{\{j|x_j \notin \pi_i\}} (x'_j \leftrightarrow T_j(I, X)) \right) \right) \\
&\text{From predicate logic, } \exists x (P(x) \wedge Q(x)) \rightarrow \exists x P(x) \wedge \exists x Q(x) \\
&= \lambda X' \exists \{x \in X\} \left(S(X) \wedge \exists i \in I \cup Y \left[\bigwedge_{\{j|x_j \in \pi_i\}} (x'_j \leftrightarrow T_j(I, X)) \wedge 1 \right] \right) \\
&\text{Since } \exists \{i \in I\} (\bigwedge_{j|x_j \notin \pi_i} (x'_j \leftrightarrow T_j(I, X))) = 1, \\
&\quad \text{as } x'_j \text{ can be assigned the value of } T_j(I, X) \text{ to make it true} \\
&= \lambda X' \exists \{x \in X\} \left(S(X) \wedge \exists i \in I \left(\bigwedge_{\{j|x_j \in \pi_i\}} (x'_j \leftrightarrow T_j(I, X)) \right) \wedge \right. \\
&\quad \left. \exists y_j \in Y \bigwedge_{\{j|x_j \notin \pi_i\}} (x'_j \leftrightarrow y_j) \right) \\
&\text{Since } \exists \{y_j \in Y\} (\bigwedge_{\{j|x_j \notin \pi_i\}} (x'_j \leftrightarrow y_j)) = 1, \text{ as again by the same reasoning as in} \\
&\quad \text{previous step one can always find an assignment for } x'_j \text{ which makes it true} \\
&= \lambda X' \exists \{x \in X\} \left(S(X) \wedge \exists i \in I \cup Y \left[\bigwedge_{\{j|x_j \in \pi_i\}} (x'_j \leftrightarrow T_j(I, X)) \wedge \bigwedge_{\{j|x_j \notin \pi_i\}} (x'_j \leftrightarrow y_j) \right] \right) \\
&= \lambda X' \exists \{i \in I \cup Y, x \in X\} \left(S(X) \wedge \left[\bigwedge_{\{j|x_j \in \pi_i\}} (x'_j \leftrightarrow T_j(I, X)) \wedge \bigwedge_{\{j|x_j \notin \pi_i\}} (x'_j \leftrightarrow y_j) \right] \right) \\
&\quad I \cap Y = \emptyset, \text{ so } \exists \{i \in I\} \phi \wedge \exists \{y \in Y\} \psi = \exists \{i' \in (I \cup Y)\} (\phi \wedge \psi) \\
&= \text{Img}(S(X), [\mathbf{T}(I, X)]_{\pi_i}) \quad \text{From definition}
\end{aligned}$$

Hence proved.

Lemma 9. Given a projection π_i of the state variables, characteristic function $S(X)$ of a set of states S , and a BFV $\mathbf{T}(I, X)$ of length l ,

$$\alpha_i(\text{Img}(S(X), \mathbf{T}(I, X))) = \alpha_i(\text{Img}(S(X), [\mathbf{T}(I, X)]_{\pi_i}))$$

Proof: Let $\overline{\pi}_i = \{x_j | x_j \notin \pi_i\}$ and $\overline{\pi}'_i = \{x'_j | x_j \notin \pi_i\}$. Then we know from definition that

$$\begin{aligned}
\alpha_i(\text{Img}(S(X), \mathbf{T}(I, X))) &= \exists \overline{\pi}'_i \lambda X' \exists I \exists X \left(S(X) \wedge \bigwedge_{\{j|x_j \in \pi_i\}} (x'_j \leftrightarrow T_j(I, X)) \wedge \bigwedge_{\{j|x_j \notin \pi_i\}} (x'_j \leftrightarrow T_j(I, X)) \right) \\
&= \lambda \pi'_i \exists I \exists X \left(S(X) \wedge \bigwedge_{\{j|x_j \in \pi_i\}} (x'_j \leftrightarrow T_j(I, X)) \wedge \exists \overline{\pi}'_i \bigwedge_{\{j|x_j \notin \pi_i\}} (x'_j \leftrightarrow T_j(I, X)) \right) \\
&\quad \text{Note that } \exists \overline{\pi}'_i \bigwedge_{\{j|x_j \notin \pi_i\}} (x'_j \leftrightarrow T_j(I, X)) = 1 \\
&\quad \text{Since } x_j \notin \pi_i \text{ implies that } x'_j \in \overline{\pi}'_i \\
&= \lambda \pi'_i \exists I \exists X \left(S(X) \wedge \bigwedge_{\{j|x_j \in \pi_i\}} (x'_j \leftrightarrow T_j(I, X)) \wedge 1 \right) \\
&\quad \text{Also, } \exists y_j (x'_j \leftrightarrow y_j) = 1 \\
&= \lambda \pi'_i \exists I \exists X \left(S(X) \wedge \bigwedge_{\{j|x_j \in \pi_i\}} (x'_j \leftrightarrow T_j(I, X)) \wedge \exists Y \bigwedge_{\{j|x_j \notin \pi_i\}} (x'_j \leftrightarrow y_j) \right) \\
&= \exists \overline{\pi}'_i \lambda X' \exists I \exists X \exists Y \left(S(X) \wedge \bigwedge_{\{j|x_j \in \pi_i\}} (x'_j \leftrightarrow T_j(I, X)) \wedge \bigwedge_{\{j|x_j \notin \pi_i\}} (x'_j \leftrightarrow y_j) \right) \\
&= \alpha_i(\text{Img}(S(X), [\mathbf{T}(I, X)]_{\pi_i})) \quad \text{From definition}
\end{aligned}$$

Theorem 2. Given a projection π_i of the state variables, characteristic function $S(X)$ of a set of states S , and a BFV $\mathbf{T}(I, X)$ of length l ,

$$\text{Img}(S(X), \mathbf{T}(I, X)) \rightarrow \gamma \left(\alpha_1(\text{Img}(S(X), [\mathbf{T}(I, X)]_{\pi_1})), \dots, \alpha_k(\text{Img}(S(X), [\mathbf{T}(I, X)]_{\pi_k})) \right)$$

Proof: We know that

$$\begin{aligned}
\text{Img}(S(X), \mathbf{T}(I, X)) &\rightarrow \gamma \left(\alpha(\text{Img}(S(X), \mathbf{T}(I, X))) \right) \\
&= \gamma \left(\alpha_1(\text{Img}(S(X), \mathbf{T}(I, X))), \dots, \alpha_k(\text{Img}(S(X), \mathbf{T}(I, X))) \right) \\
&= \gamma \left(\alpha_1(\text{Img}(S(X), [\mathbf{T}(I, X)]_{\pi_1})), \dots, \alpha_k(\text{Img}(S(X), [\mathbf{T}(I, X)]_{\pi_k})) \right) \quad \text{From Lemma 9}
\end{aligned}$$

Definition 17. *Generalised Cofactor Operation (Constrain):*

The generalised cofactor operation or the constrain operator, denoted by \downarrow allows one to cofactor a function f , with respect to another function c , and reduces to ordinary cofactor when c is a literal. Intuitively, 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 $f(x')$.

The result of the constrain operator depends on the variable ordering used in the BDD representation. (If c is a cube, then constrain reduces to being ordinary cofactor operation and is also independent of the variable ordering). Algorithm 2 describes the constrain operation [15].

Algorithm 2 constrain $f \downarrow c$

```

1: assert ( $c \neq 0$ )
2: if ( $(c = 1) \vee (f = 0) \vee (f = 1)$ ) then
3:   return  $f$ 
4: end if
5: if ( $f = c$ ) then
6:   return 1
7: end if
8: if ( $f = \neg c$ ) then
9:   return 0
10: end if
11: Let  $x_1$  be the top variable in  $c$ 
12: if ( $c|_{\neg x_1} = 0$ ) then
13:   return  $f|_{x_1} \downarrow c|_{x_1}$ 
14: end if
15: if ( $c|_{x_1} = 0$ ) then
16:   return  $f|_{\neg x_1} \downarrow c|_{\neg x_1}$ 
17: end if
18: return  $\text{ite}(x_1, f|_{x_1} \downarrow c|_{x_1}, f|_{\neg x_1} \downarrow c|_{\neg x_1})$ 

```

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

Example 11. Let $f = p \vee r$ and $c = q$, then $f \downarrow c$ reduces to ordinary cofactor operation. Hence $f \downarrow c = (p \vee r)|_q = p \vee r$.

Example 12. Let $f = q \wedge r$ and $c = \neg p \vee (p \wedge q \wedge r)$, then using the algorithm 2 we get

$$\begin{aligned}
f \downarrow c &= \left[p \wedge \left((q \wedge r)|_p \downarrow (\neg p \vee (p \wedge q \wedge r))|_p \right) \right] \vee \left[\neg p \wedge \left((q \wedge r)|_{\neg p} \downarrow (\neg p \vee (p \wedge q \wedge r))|_{\neg p} \right) \right] \\
&= \left[p \wedge \left((q \wedge r) \downarrow (q \wedge r) \right) \right] \vee \left[\neg p \wedge \left((q \wedge r) \downarrow 1 \right) \right] \\
&= [p \wedge 1] \vee [\neg p \wedge (q \wedge r)] \\
&= p \vee (\neg p \wedge q \wedge r)
\end{aligned}$$

Example 13. Consider the same boolean functions f and c as in Example 12, but a different variable ordering $r < q < p$. Using the algorithm 2 we get

$$\begin{aligned}
f \downarrow c &= [r \wedge (f|_r \downarrow c|_r)] \vee [\neg r \wedge (f|_{\neg r} \downarrow c|_{\neg r})] \\
&= \left[r \wedge [q \downarrow (\neg p \vee (p \wedge q))] \right] \vee \left[\neg r \wedge (0 \downarrow \neg p) \right] \\
&= \left(r \wedge [q \downarrow (\neg p \vee (p \wedge q))] \right) \vee 0 \\
&= r \wedge [q \downarrow (\neg p \vee (p \wedge q))] \\
&= r \wedge \left[\left(q \wedge (q|_q \downarrow (\neg p \vee (p \wedge q))|_q) \right) \vee \left(\neg q \wedge (q|_{\neg q} \downarrow (\neg p \vee (p \wedge q))|_{\neg q}) \right) \right] \\
&= r \wedge \left[(q \wedge (1 \downarrow (\neg p \vee p))) \vee (\neg q \wedge 0) \right] \\
&= r \wedge \left[[q \wedge (1 \downarrow (\neg p \vee p))] \vee 0 \right] \\
&= r \wedge q
\end{aligned}$$

Hence we see that the variable order is very crucial when using the constrain operator. Infact this forces that any variable ordering chosen at the start of any computation must be maintained throughout the BDD implementation. No dynamic variable reorderings should be allowed.

Definition 18. Given boolean functions f , g and h , define a general constrain operator called multiple constrain as proposed by Govindaraju in [18] as

$$\begin{aligned}
f \downarrow (g \wedge h) &= (f \downarrow h) \downarrow (g \downarrow h) \\
&= (f \downarrow g) \downarrow h \text{ Only if } g \text{ and } h \text{ have disjoint supports, shown by McMillan [24].}
\end{aligned}$$

Touati *et al.* [33] have shown that constrain is a specific instance of a more general operation called the generalized cofactor. Since the use of the constrain operator is known to result in performance improvements in practical reachability analyzers, we wish to extend LREs to allow applications of this operator. In fact, we introduce a variant of the constrain operator, called *modified constrain*, denoted \downarrow_M , defined as follows.

Let $Y = \{y_1, \dots, y_n\}$ be a set of boolean variables and $f : \{0, 1\}^n \rightarrow \{0, 1\}$ and $g : \{0, 1\}^n \rightarrow \{0, 1\}$ be functions of variables from the set Y .

Definition 19. The support of f is $\{y_i \in Y \mid f|_{y_i} \not\equiv f|_{\neg y_i}\}$, where $f|_{y_i}$ and $f|_{\neg y_i}$ denote the positive and negative cofactors respectively of f with respect to y_i .

Definition 20. Let $y_1 < \dots < y_n$ be a variable order and $M \subseteq Y$. Given two points a and b in $\{0, 1\}^n$, let $\eta_M(a, b) = \sum_{y_i \in M} 2^{n-i} \cdot (a_i + b_i) \pmod 2$. For a subset N of Y and a point $c \in \{0, 1\}^n$, we define $(f \downarrow_N g)(c) = f(\rho_{g, \text{sup}(f) \setminus N}(c))$, where $\rho_{g, Z} : \{0, 1\}^n \rightarrow \{0, 1\}^n$ maps each point c in $\{0, 1\}^n$ to its nearest neighbour d (according to η_Z) such that $g(d) = 1$.

Coudert and Madre's algorithm [15] for applying the constrain operator can be easily adapted to apply the modified constrain operator by incorporating an additional check that tests whether a variable in $\text{sup}(g)$ is also in $\text{sup}(f) \setminus N$. If $M = \emptyset$ and $\text{sup}(g) \subseteq \text{sup}(f)$, we get back the original constrain operator.

Theorem 3. (a) $\text{sup}(f \downarrow_N g) \subseteq \text{sup}(f)$.

(b) $\text{Img}(S(X), \Upsilon_1(I, X, X') \wedge \Upsilon_2(I, X, X')) = \text{Img}(S(X), \Upsilon_1(I, X, X') \downarrow_{X'} \Upsilon_2(I, X, X'))$.

Theorem 3(a) ensures that when computing the image of a state set as in Theorem 3(b), the support set of the constrained transition relation doesn't increase.

Govindaraju et al [18] extended the idea of constraining by proposing a *multiple constrain* operator to efficiently compute the image of a set of states represented as the intersection of multiple (abstract) state sets. Their algorithm is based on the observation that $\Upsilon(I, X, X') \downarrow (S(X) \wedge R(X)) = (\Upsilon(I, X, X') \downarrow R(X)) \downarrow (S(X) \downarrow R(X))$. Henceforth we will denote this operation as $\Upsilon(I, X, X') \Downarrow [S(X), R(X)]$, where the vector to the right of \Downarrow can be extended to have more than 2 elements in general. Use of the *modified constrain* operator, \downarrow_M instead of the \downarrow operator in the computation of \Downarrow , gives the *modified multiple constrain* operator (denoted \Downarrow_M). The following result now follows from Theorem 3(b).

Theorem 4. $\text{Img}(S, \Upsilon_1 \wedge \Upsilon_2) = \text{Img}(1, \Upsilon_1 \Downarrow_{X'} [\Upsilon_2, S])$, where Υ_i is a predicate over $I \cup X \cup X'$ and S is a predicate of X .

In our subsequent discussion, the \downarrow_M operator is always used with $M = X'$, where X' denotes the set of next state variables of a sequential circuit.

We now discuss the algorithms presented in Cho *et al.* in [12]. We consider the use of overlapping projections of the state variables as proposed by Govindaraju *et al.* in [18]. Let F_i to refer to the transition relation for the state variables in the projection π_i .

5 Machine by Machine (MBM) Traversal

This strategy traverses the subspaces serially and iteratively, until a *fixpoint* in the computation of the reached set is obtained. MBM initially sets the reachable states of all subspaces to *tautology* and computes the reachable states of each subspace in turn. In each iteration, the BFV of the subspace being traversed is constrained by the *reachable states* of the other subspaces 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 subspaces) performed at each iteration.

Theorem 5. *The procedure MBM converges.*

Proof: Follows from Theorem 4.1 in Cho *et al.* [12].

Theorem 6. *Let R be the set of exact reachable states and reached_i be the computed set of reachable states for the subspace M_i corresponding to the projection π_i . Then $\forall i, \alpha_i(R) \subseteq \text{reached}_i$. In other words, the procedure MBM computes an over-approximation of the reachable state space.*

Proof: Follows from Theorem 4.2 in Cho *et al.* [12].

6 Frame by Frame (FBF) Traversal

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

Algorithm 3 Standard MBM Traversal MBM

```

1: for  $i = 1$  to  $k$  do
2:    $reached_i = \mathbf{1}$ 
3:    $from_i = \alpha_i(S_0)$ 
4:    $traverse_i = \mathbf{true}$ 
5: end for
6: repeat
7:    $converged = \mathbf{true}$ 
8:   for  $i = 1$  to  $k$  do
9:     if  $traverse_i$  then
10:       $old\_reached_i = reached_i$ 
11:       $reached_i = \text{BFS}(from_i \wedge \bigwedge_{l=1, l \neq i}^{l=k} (reached_l), F_i)$ 
12:       $traverse_i = \mathbf{false}$ 
13:      if  $old\_reached_i \neq reached_i$  then
14:        for  $\forall j \in \text{FANOUT}(M_i)$  do
15:           $traverse_j = \mathbf{true}$ 
16:        end for
17:      end if
18:    end if
19:   end for
20:   for  $i = 1$  to  $k$  do
21:     if  $traverse_i$  then
22:        $converged = \mathbf{false}$ 
23:     end if
24:   end for
25: until  $converged = \mathbf{true}$ 
26: return  $(\gamma(reached_1, \dots, reached_k))$ 

```

6.1 Reached FBF

RFBF initially sets the reachable states of each subspace to its respective initial states. It then interleaves the reachability analyses of the subspaces by computing one image computation for each subspace in turn. It also constrains the BFV of each submachine by the reachable sets of the other subspaces. When computing the i^{th} step of reachability analysis for a subspace, the accumulated reachable states of the other subspaces from the previous iteration are used. The computation continues till the sets of reachable states stop growing, i.e. a *fixpoint* has been attained.

Algorithm 4 Standard RFBF Traversal RFBF

```

1: for  $i = 1$  to  $k$  do
2:    $reached_i = \alpha_i(S_0)$ 
3: end for
4:  $j = 0$ 
5: repeat
6:    $converged = \mathbf{true}$ 
7:   for  $i = 1$  to  $k$  do
8:      $old\_reached_i = reached_i$ 
9:      $to_i^{j+1} = \text{Img}(\gamma(reached_1^j, \dots, reached_k^j), F_i)$ 
10:     $reached_i = reached_i \vee to_i^{j+1}$ 
11:    if  $old\_reached_i \neq reached_i$  then
12:       $converged = \mathbf{false}$ 
13:    end if
14:  end for
15:   $j = j + 1$ 
16: until  $converged = \mathbf{true}$ 
17: return  $(\gamma(reached_1, \dots, reached_k))$ 

```

Theorem 7. *The procedure RFBF converges.*

Proof: Follows from Theorem 4.4 in Cho *et al.* [12].

Theorem 8. *The procedure RFBF computes an over-approximation of the reachable state space.*

Proof: Follows from Theorem 4.3 and Lemma 4.2 in Cho *et al.* [12].

6.2 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 BFV of the subspace being traversed currently. Since this uses the smallest constraint set, it gives the smallest reached set upon convergence.

Theorem 9. *The procedure TFBF converges.*

Proof: Follows from Theorem 4.6 in Cho *et al.* [12].

Theorem 10. *The procedure TFBF computes an over-approximation of the reachable state space.*

Proof: Follows from Theorem 4.3 and Lemma 4.3 in Cho *et al.* [12].

Algorithm 5 Standard TFBB Traversal TFBB

```

1: for  $i = 1$  to  $k$  do
2:    $reached_i = \alpha_i(S_0)$ 
3: end for
4:  $j = 0$ 
5: repeat
6:    $converged = \mathbf{true}$ 
7:   for  $i = 1$  to  $k$  do
8:      $to_i^{j+1} = \text{Img}(\gamma(to_1^j, \dots, to_k^j), F_i)$ 
9:      $reached_i = reached_i \vee to_i^{j+1}$ 
10:    if  $\exists t, t < j, \gamma(to_1^t, \dots, to_k^t) \neq \gamma(to_1^j, \dots, to_k^j)$  then
11:       $converged = \mathbf{false}$ 
12:    end if
13:  end for
14:   $j = j + 1$ 
15: until  $converged = \mathbf{true}$ 
16: return  $(\gamma(reached_1, \dots, reached_k))$ 

```

7 TMBM

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

First, TFBB 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 TFBB to give the final reached set.

Algorithm 6 TMBM Traversal TMBM

```

1:  $(\{tfbf\_reached_i\}, \{to_i^t\}) = \text{TFBB}(\text{run-for-}t\text{-iterations})$ 
2: for  $i = 1$  to  $k$  do
3:    $\alpha_i(S_0) = to_i^t$ 
4: end for
5:  $\{reached_i\} = \text{MBM}()$ 
6: for  $i = 1$  to  $k$  do
7:    $tmbm\_reached_i = tfbf\_reached_i \vee reached_i$ 
8: end for
9: return  $(\gamma(tmbm\_reached_1, \dots, tmbm\_reached_k))$ 

```

Theorem 11. *The procedure TMBM converges.*

Proof: Follows from convergence proof of MBM by Theorem 5.

Theorem 12. *The procedure TMBM computes an over-approximation of the reachable state space.*

Proof: Follows from overapproximation proofs of MBM by Theorem 6 and TFBB by Theorem 10.

We have seen in detail the various state of the art search methods. We want to investigate if it would be possible to express every search method based on least fixpoint computation in a unified framework. In this regard, we first briefly discuss the theory of Reachability Expressions [31] which have been very useful in writing traversal schedules for large asynchronous systems. Another framework for writing schedules has been introduced by Varun Kanade in [22] which uses Reachability Matrices (henceforth called RM). In this chapter, we propose a variant of Reachability Expressions (henceforth called RE) called *Labeled Reachability Expressions* (henceforth called LRE) and detail its applicability. We also discuss the expressive capabilities of these frameworks.

8 Reachability Expressions

Reachability Expressions have been very useful in writing traversal schedules for applications like breadth first search and priority round-robin [31]. It has been demonstrated in [31] that it is very useful to compare the performances of different search schedules based on a few metrics like time taken and memory consumption. We present the grammar of Reachability Expressions [31] here. We have added a new operator \neg , which permits negation with REs. This operator now provides extra functionality with REs than earlier [31]. We will explain these extra features in the context of LREs later in this report.

8.1 Syntax

Given a transition system expressed as a disjunction of k transition relations, let $C_\Gamma = \{\Upsilon_1, \dots, \Upsilon_k, \delta, \theta\}$. Let $C' = C \cup \{+, \circ, ;, \neg, *, (\,)\}$ be the set of terminal symbols. An RE over the set C_Γ is a terminal string obtained from the following grammar:

$$\begin{aligned}
 E &\rightarrow E + E \\
 &| E \circ E \\
 &| E ; E \\
 &| (E) \\
 &| *E \\
 &| \neg E \\
 &| \Upsilon_1 | \dots | \Upsilon_k \\
 &| \delta \\
 &| \theta
 \end{aligned}$$

We have added the negation operator (\neg) has been added to the grammar of REs.

8.2 Semantics

REs provide a framework which formalizes the sequence of image computation operations of the underlying transition system. The semantics of an RE σ over the set C_Γ for the transition system B , denoted $\llbracket \sigma \rrbracket_B$ is

defined with respect to the underlying state transition system B , and is naturally described as a mapping from characteristic function representation of a set of states to characteristic function representation of a set of states. We shall henceforth omit the subscript B when it is clear from the context. We denote $Img_{TR}(S(X), \Upsilon_i)$ to be the image operation computed as the characteristic function given by $Img_{TR}(S(X), F_i(I, X, X'))$.

We give below an inductive definition of $\llbracket \sigma \rrbracket (S(X))$ for the characteristic function $S(X)$ of a set of states $S \subseteq Q$ following the definition by Dina Thomas *et al.* in [31].

- $\llbracket \delta \rrbracket (S(X)) = S(X)$
- $\llbracket \theta \rrbracket (S(X)) = 0$
- $\llbracket \Upsilon_i \rrbracket (S(X)) = Img_{TR}(S(X), \Upsilon_i)$
- $\llbracket \sigma_1 + \sigma_2 \rrbracket (S(X)) = \llbracket \sigma_1 \rrbracket (S(X)) \vee \llbracket \sigma_2 \rrbracket (S(X))$
- $\llbracket \sigma_1 \circ \sigma_2 \rrbracket (S(X)) = \llbracket \sigma_2 \rrbracket (\llbracket \sigma_1 \rrbracket (S(X)))$
- $\llbracket \sigma_1 ; \sigma_2 \rrbracket (S(X)) = \llbracket (\sigma_1 + \delta) \circ (\sigma_2 + \delta) \rrbracket (S(X))$
- $\llbracket (\sigma) \rrbracket (S(X)) = \llbracket \sigma \rrbracket (S(X))$
- $\llbracket \neg \sigma \rrbracket (S(X)) = \neg \llbracket \sigma \rrbracket (S(X))$
- Let $(\sigma)^0 = \delta$ and $(\sigma)^i = (\sigma)^{i-1} \circ \sigma, \forall i \geq 1$, then $\llbracket * \sigma \rrbracket (S(X)) = \bigvee_{i=0}^{\infty} \llbracket (\sigma)^i \rrbracket (S(X))$

8.3 Examples

Consider characteristic function $S(X)$ of a set of states S . The following examples show the evaluation of an RE σ over the set C_T on $S(X)$.

Example 14. Let σ be $\Upsilon_1 + \Upsilon_2$.

From RE semantics,

$$\begin{aligned} \llbracket \Upsilon_1 + \Upsilon_2 \rrbracket (S(X)) &= \llbracket \Upsilon_1 \rrbracket (S(X)) \vee \llbracket \Upsilon_2 \rrbracket (S(X)) \\ &= Img_{TR}(S(X), \Upsilon_1) \vee Img_{TR}(S(X), \Upsilon_2) \end{aligned}$$

Example 15. Let σ be $(\Upsilon_1 + \Upsilon_2) \circ \Upsilon_3$.

From RE semantics,

$$\begin{aligned} \llbracket (\Upsilon_1 + \Upsilon_2) \circ \Upsilon_3 \rrbracket (S(X)) &= \llbracket \Upsilon_3 \rrbracket \left(\llbracket (\Upsilon_1 + \Upsilon_2) \rrbracket (S(X)) \right) \\ &= \llbracket \Upsilon_3 \rrbracket \left(\llbracket \Upsilon_1 \rrbracket (S(X)) \vee \llbracket \Upsilon_2 \rrbracket (S(X)) \right) \\ &= \llbracket \Upsilon_3 \rrbracket \left(Img_{TR}(S(X), \Upsilon_1) \vee Img_{TR}(S(X), \Upsilon_2) \right) \\ &= Img_{TR} \left(Img_{TR}(S(X), \Upsilon_1) \vee Img_{TR}(S(X), \Upsilon_2), \Upsilon_3 \right) \end{aligned}$$

All properties of REs in [31] still hold good for any RE without any occurrence of the negation operator. Reachability expressions (REs) [32] provide a framework for expressing reachability strategies for asynchronous systems, but lack the (i) set negation and intersection operators (needed for expressing certain strategies), and (ii) ability to store and refer to subresults, as in the nodes of *curr*, *next* and *acc*. We extend the concept of REs by introducing a *negation* operator to address the former and *labels* for the latter. A label tags a set of states used either as the argument of an image operation or to store the result of an image operation. This allows us to express and reason about significantly more complex and optimized reachability strategies than is possible using REs. The negation operator, although crucial, however makes reasoning about LREs more difficult than REs.

9 A motivating example

For brevity, we will henceforth refer to the transition relation $\Upsilon(I, X, X')$ and set of states $S(X)$ as Υ and S respectively. Each step in a reachability analysis algorithm involves computing $S' = \text{Img}(S, \Upsilon)$ for an appropriately chosen set of present states S , and updating various state sets. Designers and verification engineers often use an intuitive boolean decomposition of Υ to decompose a sequential machine into smaller submachines, each of which can be efficiently traversed. A co-ordinated submachine traversal, henceforth called a *strategy*, yields the set of reachable states.

We now illustrate how knowledge of the boolean structure of Υ can be used to discover strategies, which we'd like to capture naturally in our framework. Let $\Upsilon = \Upsilon_1 \vee (\Upsilon_2 \wedge \Upsilon_3)$. Throughout this paper, we will use negation-free Boolean decompositions of transition relations. The structure of the decomposition of Υ can be conveniently represented as a tree, henceforth called a *transition tree (TT)*, as shown in Figure 3(a). The exact image of S under Υ , or $\text{Img}(S, \Upsilon)$ is given by $\lambda X'. \exists I, X (S \wedge \Upsilon)$. Using the decomposition of Υ , and noting that existential quantification distributes over disjunction, but leads to overapproximations when distributed over conjunction, $\text{Img}(S, \Upsilon)$ is approximated from above by $S'_1 \vee (S'_2 \wedge S'_3)$, where $S'_i = \text{Img}(S, \Upsilon_i)$. The similarity with the structure of the decomposition of Υ is not surprising, since we used the decomposition of Υ to compute the overapproximate image.

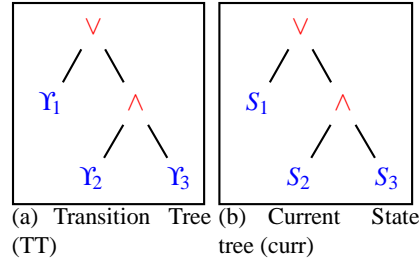


Fig. 3: Motivating example

The decomposition of Υ therefore naturally suggests the following simple approximate reachability analysis technique. We maintain a *current state tree (curr)* (shown in Figure 3(b)), a *next state tree (next)* and an *accumulated state tree (acc)*, each of which has the same structure as the transition tree TT , with every internal node being associated with the same Boolean operator as the corresponding internal node in TT . Every node in $curr$, $next$ and acc is also thought of as representing a *view* of the current state set, next state set and accumulated state set, respectively. The root of each tree gives the best view of the corresponding state set. The view at each internal node is obtained by applying the Boolean operator associated with the node on the views associated with its children. To begin with, we initialize the leaves of $curr$, $next$ and acc in the following way. All leaves of $next$ are initialized to False, representing the empty view of states. If Υ_i potentially updates the values of state variables in $X_i \subseteq X$, we initialize the corresponding leaf of $curr$ and acc with the projection of the initial set of states on X_i . Every step of reachability analysis then picks an Υ_i , computes the image of the view at the root of $curr$ under Υ_i , and updates the corresponding leaf of $next$. Once all leaves of $next$ have been updated, each leaf of acc is updated by accumulating (or disjuncting) the view already represented at this leaf with the view generated at the corresponding leaf of $next$. The views at all nodes of acc are then copied to the corresponding nodes of $curr$, and the process repeated until the view at the root of acc stops changing.

Notice that in the above algorithm, we did not use the view at any internal node of *next*, or at any non-root node of *curr*. One may therefore ask why these nodes were needed. To understand this, we note that the above algorithm is just one of many ways for doing reachability analysis using the given decomposition of Υ . As an alternative, we could have computed $Img(S, \Upsilon)$ as $S'_1 \vee S'_{2,3}$, where $S'_{2,3} = Img(S, \Upsilon_2 \wedge \Upsilon_3)$. Of course, such a computation is possible only if $\Upsilon_2 \wedge \Upsilon_3$ can be represented and operated on efficiently. Indeed, it is often the case that for the initial few steps of reachability analysis when the reachable state sets are not large, such a computation may be practically feasible, although it becomes infeasible in later stages of reachability analysis. Maintaining the complete *next* tree allows us to update the view at non-leaf nodes in this tree (and also in *acc*) by new sets of states, if non-leaf nodes in the transition tree are used to compute an image in a step of reachability analysis. Note that once the view at a non-leaf node in *acc* is updated by accumulating the corresponding view from *new*, we need not update the view at this node of *acc* from the views of its children. In fact, it can be shown that updating a view directly at node n in the *next* or *acc* tree results in no less accurate a view than that obtained by updating views at children of n and then combining these views using the Boolean operator associated with n . Furthermore, it may not always be computationally efficient to use the view S at the root of *curr* when computing an image, if the state variable count is large. This is because computation of $\exists I, X(S \wedge \Upsilon_i)$ may involve BDDs with very large number of variables. In such situations, an overapproximation of $S'_i = Img(S, \Upsilon_i)$ can be computed by using the view at an internal node of *curr* if all ancestors of this node are associated with the conjunction operator. If, however, all ancestors of the node are associated with the disjunction operator, using the view at the this node of the *curr* tree gives an underapproximation of S'_i . Thus, maintaining all nodes in *curr* allows the flexibility of using approximations of the view at the root of this tree, if computational constraints forbid using the view at the root.

In addition to the above mentioned ways of using the decomposition of Υ for reachability analysis, we could also have different ways of iterating through the different Υ_i 's. For example, one may consider repeatedly computing and accumulating the image of the view at the root of *curr* under Υ_1 , before proceeding to compute and accumulate the image under Υ_2 or Υ_3 . One may also consider applying the distributivity laws of Boolean algebra to obtain alternative decomposition trees from a given decomposition of Υ . Each such decomposition gives rise to a family of reachability strategies as described above. As we will show later, it is indeed beneficial at times to use the distributivity laws to obtain new decomposition trees. Certain reachability analysis strategies using the new decomposition tree gives provably better approximations than that obtained using the original decomposition tree. Thus, having a decomposition of the transition relation opens the doors to a large number of possible reachability analysis strategies. In the subsequent section, we present a generic framework which helps in capturing, reasoning about and implementing a large family of such strategies.

10 Labels and Reachability Expressions

We extend the concept of REs by introducing a *negation* operator to address the former and *labels* for the latter. A label tags a set of states used either as the argument of an image operation or to store the result of an image operation. This allows us to express and reason about significantly more complex and optimized reachability strategies than is possible using REs. The negation operator, although crucial, however makes reasoning about LREs more difficult than REs. We first present the syntax and semantics of LREs which are inspired from [32].

10.1 Syntax

Definition 21. Let Q be the set of all states of a sequential circuit. A State Set Vector \bar{S} , of length n , is $\bar{S} = (S_1, \dots, S_n)$, where $\forall i, S_i \subseteq Q$.

Henceforth, we will assume that all State Set Vectors are of length n . Given a negation-free Boolean decomposition of Υ in terms of $\{\Upsilon_1, \dots, \Upsilon_k\}$, let $\mathcal{T}_{\Upsilon, n} = \{\Upsilon_1, \dots, \Upsilon_k, \delta, \theta, 1, \dots, n\}$, j is a label used to refer to the j^{th} component of \bar{S} , and δ and θ are transition relations such that $\text{Img}(S, \delta) = S$ and $\text{Img}(S, \theta) = \emptyset$. An LRE λ over $\mathcal{T}_{\Upsilon, n}$ is a terminal string obtained from the following grammar, where LE is the start symbol.

$$\begin{aligned}
\text{LE} &\rightarrow L : (\text{E})_L \\
&| \text{LE} + \text{LE} \\
&| \text{LE} \circ \text{LE} \\
&| \text{LE} ; \text{LE} \\
&| (\text{LE}) \\
&| * \text{LE} \\
&| \neg(\text{LE}) \\
\text{E} &\rightarrow \Upsilon_i \downarrow_{X'} [C] \mid \Upsilon_i \mid \delta \mid \theta \\
\text{C} &\rightarrow \Upsilon_i, C \mid L, C \mid L \\
\text{L} &\rightarrow 1 \mid 2 \mid \dots \mid n
\end{aligned}$$

where i ranges from 1 through k . It is worth noting the key additions made to the syntax to overcome the limitations of REs. We have allowed for

1. Use of labels to specify the starting set of states of each image operation using the subexpression $L : (\text{E})_L$. The second occurrence of L specifies the the starting state set to be used while evaluating the corresponding LRE.
2. Storing the result of an image computation sequence is permitted using the subexpression $L : (\text{E})_L$, where the first occurrence of L specifies the storage location of the operation, i.e. the state set which stores the result of the evaluation of the corresponding LRE.
3. The operator $\downarrow_{X'}$ has been added to allow the *modified multiple constrain* operation which provides computational efficiency.
4. The subexpression $[C]$ allows to write a finite vector of labels to indicate the state sets/ Υ_i 's which are used as constraints in an image computation operation.

10.2 Semantics

Let \bar{S} , \bar{P} and \bar{R} be State Set Vectors.

- Definition 22.** (i) \bar{S} is defined to be $\bar{R} \sqcup \bar{P}$ if $\forall i, S_i = R_i \cup P_i$.
(ii) \bar{S} is defined to be $\neg \bar{R}$, if $\forall i, S_i = \bar{R}_i$.
(iii) \bar{S} is said to be covered by \bar{R} , denoted $\bar{S} \preceq \bar{R}$, if $\forall i, S_i \subseteq R_i$.

The semantics of λ , denoted $\llbracket \lambda \rrbracket_B$, is naturally described as a transformer of a State Set Vector into another State Set Vector of the same length for a sequential machine B . We present an inductive definition of $\llbracket \lambda \rrbracket_B(\bar{S})$ and omit B when it is clear from the context.

1. $\llbracket [i : (\delta)_k] \rrbracket(\bar{S}(X)) = \bar{R}(X)$ where $\forall l \neq i, R_l(X) = S_l(X)$ and $R_i(X) = S_k(X)$.
compute($i : (\delta)_k, \bar{S}(X)$) is clear from the definition.

2. $\llbracket i : (\theta)_k \rrbracket (\bar{S}(X)) = \bar{R}(X)$ where $\forall l \neq i, R_l(X) = S_l(X)$ and $R_i(X) = 0$.
 $compute(i : (\theta)_k, \bar{S}(X))$ is clear from the definition.

3. $\llbracket i : (\Upsilon_j)_k \rrbracket (\bar{S}(X)) = \bar{R}(X)$ where $\forall l \neq i, R_l(X) = S_l(X)$ and $R_i(X) = \text{Img}_{TR}(S_k(X), \Upsilon_j)$.
 $compute(i : (\Upsilon_j)_k, \bar{S}(X))$ is clear from the definition.

4. $\llbracket \lambda_1 \circ \lambda_2 \rrbracket (\bar{S}(X)) = \llbracket \lambda_2 \rrbracket (\llbracket \lambda_1 \rrbracket (\bar{S}(X)))$.
 $compute(\lambda_1 \circ \lambda_2, \bar{S}(X))$ is clear from the definition.

5. $\llbracket (\lambda_1) \rrbracket (\bar{S}(X)) = \llbracket \lambda_1 \rrbracket (\bar{S}(X))$.
 $compute((\lambda_1), \bar{S}(X))$ is clear from the definition.

6. $\llbracket -\lambda_1 \rrbracket (\bar{S}(X)) = \neg \llbracket \lambda_1 \rrbracket (\bar{S}(X))$.

$compute(-\lambda_1, \bar{S}(X))$ is evaluated as

(a) Let $\bar{R}(X) = compute(\lambda_1, \bar{S}(X))$.

(b) Let $\bar{P}(X) = \neg \bar{R}(X)$ such that $\forall i, P_i(X) = \neg R_i(X)$.

(c) Return $\bar{P}(X)$.

7. Let $[C] = [\Upsilon_p, \dots, \Upsilon_q, u, \dots, v]$. Then $\llbracket i : (\Upsilon_j \downarrow_M [C])_k \rrbracket (\bar{S}) = \bar{R}$, where $\forall l \neq i, R_l = S_l$ and $R_i = \text{Img}_{TR}(1, \Upsilon_j \downarrow_{X'} [\Upsilon_p, \dots, \Upsilon_q, S_u, \dots, S_v, S_k])$.

8. $\llbracket \lambda_1 + \lambda_2 \rrbracket (\bar{S}(X)) = \llbracket \lambda_1 \rrbracket (\bar{S}(X)) \sqcup \llbracket \lambda_2 \rrbracket (\bar{S}(X))$.

$compute(\lambda_1 + \lambda_2, \bar{S}(X))$ is evaluated as

(a) Let $\bar{R}(X) = compute(\lambda_1, \bar{S}(X))$.

(b) Let $\bar{P}(X) = compute(\lambda_2, \bar{S}(X))$.

(c) Return $\bar{Q}(X) = \bar{R}(X) \sqcup \bar{P}(X)$ such that $\forall i, Q_i(X) = R_i(X) \vee P_i(X)$.

9. $\llbracket \hat{\delta} \rrbracket (\bar{S}(X)) = \llbracket 1 : (\delta)_1 \rrbracket (\bar{S}(X)) = \bar{S}(X)$.

$compute(\hat{\delta}, \bar{S}(X))$ is clear from the definition. Note that $\hat{\delta} = i : (\delta)_i$ for any $i \in \{1, \dots, n\}$.

10. $\llbracket \hat{\theta} \rrbracket (\bar{S}(X)) = \llbracket \circ_{i=1}^n (i : (\theta)_1) \rrbracket (\bar{S}(X)) = \bar{R}(X)$ where $\forall i, R_i(X) = 0$.
 $compute(\hat{\theta}, \bar{S}(X))$ is clear from the definition.

11. Let $(\lambda_1)^0 = \hat{\delta}$ and $\forall i \geq 1, \lambda_1^i = (\lambda_1)^{i-1} \circ \lambda_1$. Then $\llbracket * \lambda_1 \rrbracket_B(\bar{S}) = \lim_{i \rightarrow \infty} \llbracket (\hat{\delta} + \lambda_1)^i \rrbracket_B(\bar{S}) = \llbracket (\hat{\delta} + \lambda_1)^{n \cdot |Q|} \rrbracket_B(\bar{S})$, where \bar{S} has n components and Q denotes the set of all states of B .

12. $\llbracket \lambda_1 ; \lambda_2 \rrbracket (\bar{S}(X)) = \llbracket (\lambda_1 + \hat{\delta}) \circ (\lambda_2 + \hat{\delta}) \rrbracket (\bar{S}(X))$.

$compute(\lambda_1 ; \lambda_2, \bar{S}(X))$ is evaluated as

(a) Let $\bar{R}(X) = compute(\lambda_1 + \hat{\delta}, \bar{S}(X))$.

(b) Let $\bar{P}(X) = compute(\lambda_2 + \hat{\delta}, \bar{R}(X))$.

(c) Return $\bar{P}(X)$.

We prefer the use of negation over intersection as it allows expressing strategies that intersection alone cannot capture. For instance, use of frontier sets, or the set of *new* states that arise from an image computation and that have been reached earlier are used in several search methods like BFS for improving the efficiency of reachability analysis. Computing a frontier set requires negating the current reachable state set and intersecting this with the image of the current state set. For notational convenience, we will use the notation $\lambda_i \wedge \lambda_j$ to denote $\neg(\neg\lambda_i + \neg\lambda_j)$.

10.3 Examples

Given an LRE λ defined over $\mathcal{T}_{Y,n}$, we present a few examples here. Assume for sake of exposition that the State Set Vectors \bar{S} , \bar{P} and \bar{R} are of length $n=3$.

Example 16. Let λ be $3 : (\delta)_2$.

From the operational procedure $\bar{R}(X) = \text{compute}(3 : (\delta)_2, \bar{S}(X))$ described in step 1 of LRE semantics, we get

1. $R_1(X) = S_1(X)$, $R_2(X) = S_2(X)$, and $R_3(X) = S_2(X)$ (From definition).
2. Return $\bar{R}(X) = (S_1(X), S_2(X), S_2(X))$ (From definition).

Hence, $\llbracket 3 : (\delta)_2 \rrbracket (S_1(X), S_2(X), S_3(X)) = (S_1(X), S_2(X), S_2(X))$. This shows how the labels can be used to update or change the contents of different state set components.

Example 17. Let λ be $1 : (\theta)_3 \circ 2 : (\Upsilon_2)_3$.

From the operational procedure $\bar{P}(X) = \text{compute}(1 : (\theta)_3 \circ 2 : (\Upsilon_2)_3, \bar{S}(X))$ described in step 4 of LRE semantics, we get

1. $\bar{R}(X) = (0, S_2(X), S_3(X))$ (From definition).
2. $\bar{P}(X) = (R_1(X), \text{Img}_{TR}(R_3(X), \Upsilon_2), R_3(X))$ (From definition).
3. Return $\bar{P}(X) = (0, \text{Img}_{TR}(S_3(X), \Upsilon_2), S_3(X))$ (From definition).

Example 18. Let λ be $\neg(1 : (\Upsilon_1)_2)$.

From the operational procedure $\bar{P}(X) = \text{compute}(\neg(1 : (\Upsilon_1)_2), \bar{S}(X))$ described in step 6 of LRE semantics, we get

1. $\bar{R}(X) = \text{compute}(1 : (\Upsilon_1)_2, \bar{S}(X))$ (From step 6a).
 - (a) $R_2(X) = S_2(X)$, and $R_3(X) = S_3(X)$ (From definition).
 - (b) $R_1(X) = \text{Img}_{TR}(S_2(X), \Upsilon_1)$ (From definition).
 - (c) Return $\bar{R}(X) = (\text{Img}_{TR}(S_2(X), \Upsilon_1), S_2(X), S_3(X))$ (From definition).
2. $\bar{P}(X) = \neg\bar{R}(X) = (\neg\text{Img}_{TR}(S_2(X), \Upsilon_1), \neg S_2(X), \neg S_3(X))$ (From step 6b).
3. Return $\bar{P}(X) = (\neg\text{Img}_{TR}(S_2(X), \Upsilon_1), \neg S_2(X), \neg S_3(X))$ (From step 6c).

Example 19. Let λ be $*(1 : (\Upsilon_1)_2)$

From the operational procedure $\bar{P}_0(X) = \bar{S}(X)$ described in step 11 of LRE semantics, we get

1. $\bar{P}_1(X) = \text{compute}(1 : (\Upsilon_1)_2, \bar{S}(X)) = (\text{Img}_{TR}(S_2(X), \Upsilon_1), S_2(X), S_3(X))$ (From definition).
2. $\bar{P}_2(X) = \text{compute}((1 : (\Upsilon_1)_2)^2, \bar{S}(X)) = \bar{P}_1(X)$ (From definition).
3. Return $\bar{P}_1(X) \sqcup \bar{P}_0(X) = (S_1(X) \vee \text{Img}_{TR}(S_2(X), \Upsilon_1), S_2(X), S_3(X))$

11 Classification of LREs

We want to establish certain useful properties of LREs so that we do not have to refer to their semantics every time for evaluating an LRE over a State Set Vector. With the introduction of \neg and $\Downarrow_{X'}$ operator, it has been seen that several properties which hold with the negation free fragment of REs are no longer true. But \neg and $\Downarrow_{X'}$ are essential operators for writing realistic search schedules for large systems. Hence, we want to investigate if there is a certain fragment of LREs which preserves some useful properties. Monotonicity is a crucial property to possess. In this regard, we would like to be able to classify LREs based on their monotonicity.

Definition 23. Given two State Set Vectors $\overline{S}(X)$ and $\overline{R}(X)$ of length n , we say that $\overline{S}(X) \succ \overline{R}(X)$ if $\exists i \in \{1, \dots, n\} S_i \supset R_i$ and $\forall l \neq i, S_l \supseteq R_l$.

Definition 24. Given a transition system $B = (I, X, Q, S_0, \mathbf{T})$, and an LRE λ over $\mathcal{T}_{\Upsilon, n}$, $[[\lambda]]_B : (2^Q)^n \rightarrow (2^Q)^n$ is said to be monotonically increasing if for all transition systems B and for all $\overline{S}(X)$ and $\overline{R}(X)$ of length n where $(S_1, \dots, S_n) \preceq (Q, \dots, Q)$ and $(R_1, \dots, R_n) \preceq (Q, \dots, Q)$, if $\overline{S}(X) \preceq \overline{R}(X)$, then $[[\lambda]]_B(\overline{S}(X)) \preceq [[\lambda]]_B(\overline{R}(X))$.

Definition 25. Given a transition system $B = (I, X, Q, S_0, \mathbf{T})$, and an LRE λ over $\mathcal{T}_{\Upsilon, n}$, $[[\lambda]]_B : (2^Q)^n \rightarrow (2^Q)^n$ is said to be monotonically decreasing if for all transition systems B and for all $\overline{S}(X)$ and $\overline{R}(X)$ of length n where $(S_1, \dots, S_n) \preceq (Q, \dots, Q)$ and $(R_1, \dots, R_n) \preceq (Q, \dots, Q)$, if $\overline{S}(X) \preceq \overline{R}(X)$, then $[[\lambda]]_B(\overline{S}(X)) \succeq [[\lambda]]_B(\overline{R}(X))$.

Definition 26. Monotonicity Metric (m): Given a transition system $B = (I, X, Q, S_0, \mathbf{T})$, we intend to classify an LRE λ using a monotonicity metric, denoted $m(\lambda)$, where $m(\lambda) \in \{0, 1, 2\}$ such that

$$m(\lambda) = \begin{cases} 0 & \text{if } [[\lambda]]_B \text{ is monotonically increasing} \\ 1 & \text{if } [[\lambda]]_B \text{ is monotonically decreasing} \\ 2 & \text{otherwise} \end{cases}$$

We desire to capture monotonically increasing LREs with an m -value of 0. Let $\overline{S}(X)$ and $\overline{R}(X)$ be characteristic functions of State Set Vectors such that $\overline{S}(X) \preceq \overline{R}(X)$. Given an LRE λ over $\mathcal{T}_{\Upsilon, n}$, we inductively define $m(\lambda)$ as follows (and give justifications for use of $m(\lambda) = 2$).

1. If $\lambda = i : (\delta)_j$, then $m(\lambda) = 0$.
2. If $\lambda = i : (\theta)_j$, then $m(\lambda) = 0$.
3. If $\lambda = i : (\Upsilon_k)_j$, then $m(\lambda) = 0$.
4. If $\lambda = i : (\Upsilon_k \Downarrow_{X'} [\Upsilon_p, \dots, \Upsilon_q, u, \dots, v])_j$, then $m(\lambda) = 0$.
5. If $\lambda = \lambda_1 + \lambda_2$, then

$$m(\lambda) = \begin{cases} 0 & \text{if } m(\lambda_1) = m(\lambda_2) = 0 \\ 1 & \text{if } m(\lambda_1) = m(\lambda_2) = 1 \\ 2 & \text{otherwise} \end{cases}$$

Let $\lambda = 1 : (\delta)_1 + \neg(2 : (\Upsilon_2)_2)$. We know from basis that $m(1 : (\delta)_1) = 0$ and from basis and definition of m that $m(\neg(2 : (\Upsilon_2)_2)) = 1$. Consider for ease of exposition that the lengths of $\overline{S}(X)$ and $\overline{R}(X)$ is $n = 3$.

Hence by definition of semantics,

$$\begin{aligned}
[[\lambda]](\bar{S}) &= [[1 : (\delta)_1]](\bar{S}(X)) \sqcup [[\neg(2 : (\Upsilon_2)_2)]](\bar{S}(X)) \\
&= (S_1(X), S_2(X), S_3(X)) \sqcup (\neg S_1(X), \neg \text{Img}_{TR}(S_2(X), \Upsilon_2), \neg S_3(X)) \\
&= (1, S_1(X) \vee \neg \text{Img}_{TR}(S_2(X), \Upsilon_2), 1) \\
\text{Also, } [[\lambda]](\bar{R}) &= [[1 : (\delta)_1]](\bar{R}(X)) \sqcup [[\neg(2 : (\Upsilon_2)_2)]](\bar{R}(X)) \\
&= (R_1(X), R_2(X), R_3(X)) \sqcup (\neg R_1(X), \neg \text{Img}_{TR}(R_2(X), \Upsilon_2), \neg R_3(X)) \\
&= (1, R_1(X) \vee \neg \text{Img}_{TR}(R_2(X), \Upsilon_2), 1)
\end{aligned}$$

We see that the vectors $(1, S_1(X) \vee \neg \text{Img}_{TR}(S_2(X), \Upsilon_2), 1)$ and $(1, R_1(X) \vee \neg \text{Img}_{TR}(R_2(X), \Upsilon_2), 1)$ are incomparable even though $\bar{S}(X) \preceq \bar{R}(X)$.

6. If $\lambda = \neg\lambda_1$, then

$$m(\lambda) = \begin{cases} 0 & \text{if } m(\lambda_1) = 1 \\ 1 & \text{if } m(\lambda_1) = 0 \\ 2 & \text{if } m(\lambda_1) = 2 \end{cases}$$

Let $\lambda = \neg(\neg(1 : (\Upsilon_2)_2) + 3 : (\delta)_2)$.

We know from definition that $m(\neg(1 : (\Upsilon_2)_2) + 3 : (\delta)_2) = 2$. Assume for ease of exposition that $\bar{S}(X)$ and $\bar{R}(X)$ are of length $n = 3$. From definition of LRE semantics, we know that

$$\begin{aligned}
[[\lambda]](\bar{S}(X)) &= [[\neg(\neg(1 : (\Upsilon_2)_2) + 3 : (\delta)_2)]](S_1(X), S_2(X), S_3(X)) \\
&= (\neg S_1(X) \wedge \text{Img}_{TR}(S_2(X), \Upsilon_2), 1, \neg S_2(X) \wedge S_3(X))
\end{aligned}$$

Similarly,

$$[[\lambda]](\bar{R}(X)) = (\neg R_1(X) \wedge \text{Img}_{TR}(R_2(X), \Upsilon_2), 1, \neg R_2(X) \wedge R_3(X))$$

We see that $[[\lambda]](\bar{S}(X))$ and $[[\lambda]](\bar{R}(X))$ are incomparable even though $\bar{S}(X) \preceq \bar{R}(X)$.

7. If $\lambda = \lambda_1 \circ \lambda_2$, then

$$m(\lambda) = \begin{cases} 0 & \text{if } m(\lambda_1) = m(\lambda_2) \text{ and } m(\lambda_1) \neq 2 \\ 1 & \text{if } m(\lambda_1) = 1 \text{ and } m(\lambda_2) = 0, \text{ or } m(\lambda_1) = 0 \text{ and } m(\lambda_2) = 1 \\ 2 & \text{otherwise} \end{cases}$$

We consider three examples:

(a) If $m(\lambda_1) = 2$ and $m(\lambda_2) = 0$.

Let $\lambda_1 = \neg 2 : (\Upsilon_2)_2 + 3 : (\delta)_2$ and $\lambda_2 = 2 : (\Upsilon_1)_3$. Then we know from definition of LRE semantics that

$$\begin{aligned}
[[\lambda_1 \circ \lambda_2]](\bar{S}(X)) &= (S_1(X) \vee \neg \text{Img}_{TR}(S_2(X), \Upsilon_2), \text{Img}_{TR}(S_2(X) \vee \neg S_3(X), \Upsilon_1), S_2(X) \vee \neg S_3(X)) \\
[[\lambda_1 \circ \lambda_2]](\bar{R}(X)) &= (R_1(X) \vee \neg \text{Img}_{TR}(R_2(X), \Upsilon_2), \text{Img}_{TR}(R_2(X) \vee \neg R_3(X), \Upsilon_1), R_2(X) \vee \neg R_3(X))
\end{aligned}$$

We see that $[[\lambda]](\bar{S}(X))$ and $[[\lambda]](\bar{R}(X))$ are incomparable.

(b) If $m(\lambda_1) = 2$ and $m(\lambda_2) = 1$.

Let $\lambda_1 = \neg 2 : (\Upsilon_2)_2 + 3 : (\delta)_2$ and $\lambda_2 = \neg 2 : (\Upsilon_1)_3$. Then we know from definition of LRE semantics

that

$$\begin{aligned} \llbracket \lambda_1 \circ \lambda_2 \rrbracket (\overline{S}(X)) &= (\neg(S_1(X) \vee \neg \text{Img}_{TR}(S_2(X), Y_2)), \neg(\text{Img}(S_1(X) \vee \neg \text{Img}_{TR}(S_2(X), Y_2), Y_1)), \\ &\quad \neg(S_2(X) \vee \neg S_3(X))) \\ \llbracket \lambda_1 \circ \lambda_2 \rrbracket (\overline{R}(X)) &= (\neg(R_1(X) \vee \neg \text{Img}_{TR}(R_2(X), Y_2)), \neg(\text{Img}(R_1(X) \vee \neg \text{Img}_{TR}(R_2(X), Y_2), Y_1)), \\ &\quad \neg(R_2(X) \vee \neg R_3(X))) \end{aligned}$$

We see that $\llbracket \lambda \rrbracket (\overline{S}(X))$ and $\llbracket \lambda \rrbracket (\overline{R}(X))$ are incomparable.

(c) If $m(\lambda_1) = 2$ and $m(\lambda_2) = 2$. Consider the same λ_1 as earlier, and let $\lambda_2 = 2 : (Y_1)_3 + \neg 2 : (\delta)_1$. It can be shown that $\llbracket \lambda \rrbracket (\overline{S}(X))$ and $\llbracket \lambda \rrbracket (\overline{R}(X))$ are incomparable.

8. If $\lambda = (\lambda_1)$, then $m(\lambda) = m(\lambda_1)$ is obvious from the definition of LRE semantics.

9. If $\lambda = \lambda_1 ; \lambda_2$, then $m(\lambda) = m\left((\lambda_1 + 1 : (\delta)_1) \circ (\lambda_2 + 1 : (\delta)_1)\right)$.

10. If $\lambda = * \lambda_1$, then

$$m(\lambda) = \begin{cases} 0 & \text{if } m(\lambda_1) = 0 \\ 2 & \text{otherwise} \end{cases}$$

If $m(\lambda_1) \neq 0$, then by definition $m(* \lambda_1) = m(1 : (\delta)_1) + m(\lambda_1) + m(\lambda_1^2) + \dots$. It follows from the definition of m that $m(* \lambda_1) = 2$.

Lemma 10. Let λ be an LRE over $\mathcal{T}_{T,n}$. Then,

1. If $m(\lambda) = 0$, then $\llbracket \lambda \rrbracket$ is monotonically increasing.
2. If $m(\lambda) = 1$, then $\llbracket \lambda \rrbracket$ is monotonically decreasing.

Proof. 1. Let $\overline{S}(X)$ and $\overline{R}(X)$ be State Set Vectors of length n such that $\overline{S}(X) \preceq \overline{R}(X)$. We show by induction on the structure of λ that $\llbracket \lambda \rrbracket (\overline{S}(X)) \preceq \llbracket \lambda \rrbracket (\overline{R}(X))$ when $m(\lambda) = 0$.

Basis:

(a) If $\lambda = i : (\delta)_j$.

$$\begin{aligned} \llbracket i : (\delta)_j \rrbracket (\overline{S}(X)) &= (S_1(X), \dots, S_{i-1}(X), S_j(X), \dots, S_n(X)) \quad \text{From LRE semantics} \\ &\preceq \llbracket i : (\delta)_j \rrbracket (\overline{R}(X)) \quad \text{From LRE semantics and } \overline{S}(X) \preceq \overline{R}(X) \end{aligned}$$

(b) If $\lambda = \hat{\delta}$.

We know from definition that $\hat{\delta} = 1 : (\delta)_1$. Hence from case 1a, $\llbracket \hat{\delta} \rrbracket$ is also monotonically increasing.

(c) If $\lambda = i : (\theta)_j$.

$$\begin{aligned} \llbracket i : (\theta)_j \rrbracket (\overline{S}(X)) &= (S_1(X), \dots, S_{i-1}(X), 0, \dots, S_n(X)) \quad \text{From LRE semantics} \\ &\preceq \llbracket i : (\theta)_j \rrbracket (\overline{R}(X)) \quad \text{From LRE semantics and } \overline{S}(X) \preceq \overline{R}(X) \end{aligned}$$

(d) If $\lambda = i : (\Upsilon_j)_k$.

We know from definition that

$$\begin{aligned} \llbracket i : (\Upsilon_j)_k \rrbracket (\overline{S}(X)) &= \overline{P}(X) \\ \text{such that } \forall l \neq i, P_l(X) &= S_l(X) \\ \text{and } P_i(X) &= \text{Img}_{TR}(S_k(X), \Upsilon_j) \end{aligned}$$

$$\begin{aligned} \text{Similarly, } \llbracket i : (\Upsilon_j)_k \rrbracket (\overline{R}(X)) &= \overline{H}(X) \\ \text{such that } \forall l \neq i, H_l(X) &= R_l(X) \\ \text{and } H_i(X) &= \text{Img}_{TR}(R_k(X), \Upsilon_j) \end{aligned}$$

Also,

$$\begin{aligned} \forall l \neq i, P_l(X) = S_l(X) &\rightarrow R_l(X) = H_l(X) \quad \text{Since } \overline{S}(X) \preceq \overline{R}(X) \\ \text{and, } P_i(X) = \text{Img}_{TR}(S_k(X), \Upsilon_j) &\rightarrow \text{Img}_{TR}(R_k(X), \Upsilon_j) = H_i(X) \\ &\text{Since } \text{Img}_{TR} \text{ is monotonically increasing and } S_k(X) \rightarrow R_k(X) \end{aligned}$$

Hence proved.

(e) If $\lambda = i : (\Upsilon_j \downarrow_{X'} [C])_k$.

Let $[C] = [\Upsilon_p, \dots, \Upsilon_q, u, \dots, v]$ be a vector. We know from definition that

$$\begin{aligned} \llbracket i : (\Upsilon_j \downarrow_{X'} [C])_k \rrbracket (\overline{S}(X)) &= \overline{P}(X) \\ \text{such that } \forall l \neq i, P_l(X) &= S_l(X) \\ \text{and } P_i(X) &= \text{Img}_{TR}(S_k(X) \wedge S_u(X) \wedge \dots \wedge S_v(X), \Upsilon_j \wedge \Upsilon_p \wedge \dots \wedge \Upsilon_q) \end{aligned}$$

$$\begin{aligned} \text{Similarly, } \llbracket i : (\Upsilon_j \downarrow_{X'} [C])_k \rrbracket (\overline{R}(X)) &= \overline{H}(X) \\ \text{such that } \forall l \neq i, H_l(X) &= R_l(X) \\ \text{and, } H_i(X) &= \text{Img}_{TR}(R_k(X) \wedge R_u(X) \wedge \dots \wedge R_v(X), \Upsilon_j \wedge \Upsilon_p \wedge \dots \wedge \Upsilon_q) \end{aligned}$$

Also,

$$\begin{aligned} \forall l \neq i, P_l(X) = S_l(X) &\rightarrow R_l(X) = H_l(X) \quad \text{since } \overline{S}(X) \preceq \overline{R}(X) \\ \text{and, } (S_k(X) \wedge S_u(X) \wedge \dots \wedge S_v(X)) &\rightarrow (R_k(X) \wedge R_u(X) \wedge \dots \wedge R_v(X)) \end{aligned}$$

Since $\overline{S}(X) \preceq \overline{R}(X)$ &

\wedge is a monotonicity preserving operator

$$\text{Hence } P_i(X) = \text{Img}_{TR}(S_k(X) \wedge \dots \wedge S_{rp}(X), \Upsilon_j \wedge \Upsilon_p \wedge \dots \wedge \Upsilon_q) \rightarrow \text{Img}_{TR}(R_k(X) \wedge \dots \wedge R_{rp}(X), \Upsilon_j \wedge \Upsilon_p \wedge \dots \wedge \Upsilon_q) = B_i(X)$$

Since Img_{TR} is monotonically increasing

Hence proved.

Hypothesis: Let λ_1 and λ_2 be LREs over $\mathcal{T}_{\Upsilon, n}$ such that if $m(\lambda_1) = 0$, then $\llbracket \lambda_1 \rrbracket$ is monotonically increasing and if $m(\lambda_1) = 0$, then $\llbracket \lambda_2 \rrbracket$ is monotonically increasing.

Induction Step: We consider the set representation of the State Set Vectors \overline{S} and \overline{R} both of length n for the remaining part of this proof. We consider the following cases:

(a) Let λ be $\lambda_1 + \lambda_2$.

$\llbracket \lambda \rrbracket$ is monotonically increasing follows from definition of LRE semantics, monotonicity of set union and induction hypothesis.

(b) Let λ be $\lambda_1 \circ \lambda_2$.

We know by induction hypothesis that $\llbracket \lambda_1 \rrbracket (\bar{S}) \preceq \llbracket \lambda_1 \rrbracket (\bar{R})$. Also from induction hypothesis we know that

$$\begin{aligned} \llbracket \lambda_2 \rrbracket (\llbracket \lambda_1 \rrbracket (\bar{S})) &\preceq \llbracket \lambda_2 \rrbracket (\llbracket \lambda_1 \rrbracket (\bar{R})) \\ \text{Or } \llbracket \lambda_1 \circ \lambda_2 \rrbracket (\bar{S}) &\preceq \llbracket \lambda_1 \circ \lambda_2 \rrbracket (\bar{R}) \text{ from LRE semantics} \end{aligned}$$

Hence $\llbracket \lambda \rrbracket$ is monotonically increasing.

(c) Let λ be $\hat{\theta}$.

We know from definition that $\hat{\theta} = \circ_{i=1}^{i=n} (i : (\theta)_i)$. Proof follows from above and case 1c of basis.

(d) Let λ be $\lambda_1 ; \lambda_2$.

We know from LRE semantics that $\llbracket \lambda_1 ; \lambda_2 \rrbracket (\bar{S}) = \llbracket (\lambda_1 + \hat{\delta}) \circ (\lambda_2 + \hat{\delta}) \rrbracket (\bar{S})$. $\llbracket \lambda \rrbracket$ is monotonically increasing follows from the induction hypothesis and induction steps 1a and 1b proved above.

(e) Let λ be (λ_1) .

$\llbracket (\lambda_1) \rrbracket$ is monotonically increasing follows from the definition of $\llbracket (\lambda_1) \rrbracket (\bar{S})$ and induction hypothesis.

(f) Let λ be $*\lambda_1$.

We show by induction on i that then $\llbracket (\lambda_1)^i \rrbracket$ is also monotonically increasing for all $i \geq 0$.

- *Basis:* If $i = 0$, $\llbracket \lambda_1^0 \rrbracket (\bar{S}) = \llbracket \hat{\delta} \rrbracket (\bar{S})$ by definition. Hence, $\llbracket \lambda_1^0 \rrbracket$ is monotonically increasing.
- *Hypothesis:* Assume that $\llbracket \lambda_1^k \rrbracket$ is monotonically increasing for all i such that $0 \leq k \leq i$.
- *Induction Step:* We know by definition that $\lambda_1^{i+1} = \lambda_1^i \circ \lambda_1$. Also both $\llbracket \lambda_1 \rrbracket$ and $\llbracket \lambda_1^i \rrbracket$ are monotonically increasing from induction hypothesis and that $\lambda_1^i \circ \lambda_1$ is monotonically increasing from case 1b. This proves that $\llbracket \lambda_1^{i+1} \rrbracket$ is monotonically increasing for all $i \geq 0$.

From definition of LRE semantics we know that

$$\begin{aligned} \llbracket *\lambda_1 \rrbracket (\bar{S}) &= \bigsqcup_{i=0}^{n-|\mathcal{Q}|} \llbracket (\hat{\delta} + \lambda_1)^i \rrbracket (\bar{S}) \text{ and,} \\ \llbracket *\lambda_1 \rrbracket (\bar{R}) &= \bigsqcup_{i=0}^{n-|\mathcal{Q}|} \llbracket (\hat{\delta} + \lambda_1)^i \rrbracket (\bar{R}) \end{aligned}$$

So, if for any $j, s \in (\llbracket *\lambda_1 \rrbracket (\bar{S}))_j$, then there exists an integer $k (\geq 0)$ such that $s \in (\llbracket (\hat{\delta} + \lambda_1)^k \rrbracket (\bar{S}))_j$. Since $(\lambda_1)^i$ is monotonically increasing $\forall i \geq 0$, we have $\llbracket (\hat{\delta} + \lambda_1)^k \rrbracket (\bar{S}) \preceq \llbracket (\hat{\delta} + \lambda_1)^k \rrbracket (\bar{R})$. Hence, $s \in (\llbracket (\hat{\delta} + \lambda_1)^k \rrbracket (\bar{R}))_j$ as well. Hence, $\llbracket *\lambda_1 \rrbracket (\bar{S}) \preceq \llbracket *\lambda_1 \rrbracket (\bar{R})$.

Hence proved.

2. Let $\bar{S}(X)$ and $\bar{R}(X)$ be State Set Vectors of length n such that $\bar{S}(X) \preceq \bar{R}(X)$. We prove by induction on the structure of λ that if $m(\lambda) = 1$, then $\llbracket \lambda \rrbracket$ is monotonically decreasing.

Basis: If $\lambda = -\lambda_1$, then since $m(\lambda) = 1$, it follows from definition of m that $m(\lambda_1) = 0$. Hence it follows from case 1 proved above that $\llbracket \lambda_1 \rrbracket$ is monotonically increasing.

$$\begin{aligned} \llbracket \lambda_1 \rrbracket(\bar{S}(X)) &\preceq \llbracket \lambda_1 \rrbracket(\bar{R}(X)) && \text{Since } \bar{S}(X) \preceq \bar{R}(X) \\ \text{Also, } -\llbracket \lambda_1 \rrbracket(\bar{S}(X)) &\succeq -\llbracket \lambda_1 \rrbracket(\bar{R}(X)) \\ \text{Hence, } \llbracket -\lambda_1 \rrbracket(\bar{S}(X)) &\succeq \llbracket -\lambda_1 \rrbracket(\bar{R}(X)) && \text{From LRE semantics} \end{aligned}$$

Hence proved.

Hypothesis: Assume that if $m(\lambda_1) = 1$, then λ_1 is monotonically decreasing and if $m(\lambda_2) = 1$, then λ_2 is monotonically decreasing.

Induction step: There are 2 cases to be considered

- (a) If $\lambda = (\lambda_1)$. We know from induction hypothesis that

$$\begin{aligned} \llbracket \lambda_1 \rrbracket(\bar{S}(X)) &\succeq \llbracket \lambda_1 \rrbracket(\bar{R}(X)) \\ \text{Hence, } \llbracket \lambda \rrbracket(\bar{S}(X)) &\succeq \llbracket \lambda \rrbracket(\bar{R}(X)) && \text{Follows from LRE semantics of } \lambda \end{aligned}$$

Hence proved.

- (b) If $\lambda = \lambda_1 + \lambda_2$. Then,

$$\begin{aligned} \llbracket \lambda_1 + \lambda_2 \rrbracket(\bar{S}(X)) &= \llbracket \lambda_1 \rrbracket(\bar{S}(X)) \sqcup \llbracket \lambda_2 \rrbracket(\bar{S}(X)) && \text{from definition} \\ &\succeq \llbracket \lambda_1 \rrbracket(\bar{R}(X)) \sqcup \llbracket \lambda_2 \rrbracket(\bar{R}(X)) && \text{From induction hypothesis and monotonicity of } \sqcup \\ &= \llbracket \lambda_1 + \lambda_2 \rrbracket(\bar{R}(X)) \end{aligned}$$

Hence proved.

LREs can be classified depending on the monotonicity metric m as follows.

1. Subclass 0 which includes only those LREs (λ) over $\mathcal{T}_{Y,n}$ such that $m(\lambda) = 0$. This is referred to as Restricted LREs (RLREs) in the paper.
2. Subclass 1 which includes only those LREs (λ) over $\mathcal{T}_{Y,n}$ such that $m(\lambda) = 1$.
3. Subclass 2 which includes only those LREs (λ) over $\mathcal{T}_{Y,n}$ such that $m(\lambda) = 2$.

Given a transition system which can be expressed as a disjunction of k transition relations, and a set $\mathcal{T}_{Y,n}$ we use LE_i , for $i \in \{0, 1, 2\}$ to represent LREs with monotonicity metric i . Then an LRE over $\mathcal{T}_{Y,n}$ is a terminal

string obtained from the following grammar (\mathcal{G}') using LE as the start symbol.

$$LE' \rightarrow LE_0 \mid LE_1 \mid LE_2$$

$$\begin{aligned} LE_0 &\rightarrow L : (E)_L \\ &\mid LE_0 + LE_0 \\ &\mid LE_0 \circ LE_0 \mid LE_1 \circ LE_1 \\ &\mid LE_0 ; LE_0 \\ &\mid *LE_0 \\ &\mid (LE_0) \\ &\mid \neg(LE_1) \end{aligned}$$

$$\begin{aligned} LE_1 &\rightarrow \neg LE_0 \\ &\mid LE_1 + LE_1 \\ &\mid LE_1 \circ LE_0 \mid LE_0 \circ LE_1 \\ &\mid (LE_1) \end{aligned}$$

$$\begin{aligned} LE_2 &\rightarrow \neg LE_2 \\ &\mid LE_0 + LE_2 \mid LE_1 + LE_2 \mid LE_2 + LE_2 \\ &\mid LE_2 + LE_0 \mid LE_2 + LE_1 \mid LE_1 + LE_0 \mid LE_0 + LE_1 \\ &\mid LE_0 \circ LE_2 \mid LE_2 \circ LE_0 \mid LE_2 \circ LE_1 \\ &\mid LE_1 \circ LE_2 \mid LE_2 \circ LE_2 \\ &\mid LE_0 ; LE_1 \mid LE_1 ; LE_0 \mid LE_0 ; LE_2 \mid LE_2 ; LE_0 \\ &\mid LE_1 ; LE_2 \mid LE_2 ; LE_1 \mid LE_1 ; LE_1 \mid LE_2 ; LE_2 \\ &\mid *LE_1 \mid *LE_2 \\ &\mid (LE_2) \end{aligned}$$

$$E \rightarrow \Upsilon_i \Downarrow_{X'} [C] \mid \Upsilon_i \mid \delta \mid \theta$$

$$C \rightarrow \Upsilon_i, C \mid L, C \mid L$$

$$L \rightarrow 1 \mid 2 \mid \dots \mid n$$

Lemma 11. For any LRE λ over $\mathcal{T}_{\Upsilon, n}$ generated from the grammar \mathcal{G} , $m(\lambda) = i \in \{0, 1, 2\}$ iff λ is generated from the grammar \mathcal{G}' with LE_i as the start symbol.

Proof. We prove by induction on the structure of λ .

Basis: We consider three cases, one for each value of $m(\lambda)$:

1. We prove $m(\lambda) = 0$ iff λ is generated from the grammar \mathcal{G}' with LE_0 as the start symbol.

- (a) If $\lambda = i : (\delta)_j$, then from the grammar \mathcal{G}' we know that λ can only be generated using the production rule $\text{LE}_0 \rightarrow i : (\delta)_j$ with LE_0 as the start symbol. Also, using the production rule $\text{LE}_0 \rightarrow i : (\delta)_j$ gives $m(i : (\delta)_j) = 0$ from definition of m .
 - (b) If $\lambda = i : (\theta)_j$, then from the grammar \mathcal{G}' we know that λ can only be generated using the production rule $\text{LE}_0 \rightarrow i : (\theta)_j$ with LE_0 as the start symbol. Also, using the production rule $\text{LE}_0 \rightarrow i : (\theta)_j$ gives $m(i : (\theta)_j) = 0$ from definition of m .
 - (c) If $\lambda = i : (\Upsilon_k)_j$, then from the grammar \mathcal{G}' we know that λ can only be generated using the production rule $\text{LE}_0 \rightarrow i : (\Upsilon_k)_j$ with LE_0 as the start symbol. Also, using the production rule $\text{LE}_0 \rightarrow i : (\Upsilon_k)_j$ gives $m(i : (\Upsilon_k)_j) = 0$ from definition of m .
 - (d) If $\lambda = i : (\Upsilon_k \Downarrow_{X'} [C])_j$, then from the grammar \mathcal{G}' we know that λ can only be generated using the production rule $\text{LE}_0 \rightarrow i : (\Upsilon_k \Downarrow_{X'} [C])_j$ with LE_0 as the start symbol. Also, using the production rule $\text{LE}_0 \rightarrow i : (\Upsilon_k \Downarrow_{X'} [C])_j$ gives $m(i : (\Upsilon_k \Downarrow_{X'} [C])_j) = 0$ from definition of m .
2. We prove $m(\lambda) = 1$ iff λ is generated from the grammar \mathcal{G}' with LE_1 as the start symbol. No base cases (expressions derived using E as the start symbol) have an m -value = 1. Also, from the grammar \mathcal{G}' no base cases can be generated using LE_1 as the start symbol. Hence this holds trivially.
 3. We prove $m(\lambda) = 2$ iff λ is generated from the grammar \mathcal{G}' with LE_2 as the start symbol. No base cases (expressions derived using E as the start symbol) have an m -value = 2. Also, from the grammar \mathcal{G}' no base cases can be generated using LE_2 as the start symbol. Hence this holds trivially.

Induction Hypothesis: Let λ_1 and λ_2 be two LREs over $\mathcal{T}_{\Upsilon,n}$ generated from the grammar \mathcal{G} such that $m(\lambda_1) = i$ iff λ_1 is generated from the grammar \mathcal{G}' with LE_i as the start symbol and $m(\lambda_2) = j$ iff λ_2 is generated from the grammar \mathcal{G}' with LE_j as the start symbol.

Induction Step: We consider the following cases:

1. If $\lambda = \lambda_1 + \lambda_2$. We again consider three cases:
 - (a) If $m(\lambda) = 0$, then from definition of m , we know that $m(\lambda_1) = m(\lambda_2) = 0$. We also know from induction hypothesis that both λ_1 and λ_2 are derived using the start symbol LE_0 . Also the production rule $\text{LE}_0 \rightarrow \text{LE}_0 + \text{LE}_0$ generates λ .
If λ is generated from the grammar \mathcal{G}' using the start symbol LE_0 , the only way to derive λ is to use the production rule $\text{LE}_0 \rightarrow \text{LE}_0 + \text{LE}_0$. From induction hypothesis and definition of m , it follows that $m(\lambda) = 0$.
 - (b) If $m(\lambda) = 1$, then from definition of m , we know that $m(\lambda_1) = m(\lambda_2) = 1$. We also know from induction hypothesis that both λ_1 and λ_2 are derived using the start symbol LE_1 . Also the production rule $\text{LE}_1 \rightarrow \text{LE}_1 + \text{LE}_1$ generates λ .
If λ is generated from the grammar \mathcal{G}' using the start symbol LE_1 , the only way to derive λ is to use the production rule $\text{LE}_1 \rightarrow \text{LE}_1 + \text{LE}_1$. From induction hypothesis and definition of m , it follows that $m(\lambda) = 1$.
 - (c) If $m(\lambda) = 2$, then from definition of m , we know that either $m(\lambda_1) \neq m(\lambda_2)$ or atleast one of λ_1 or λ_2 has an m value of 2. We also know from induction hypothesis that for any $j \in \{0, 1, 2\}$, $m(\lambda_j) = i$ iff λ_j is derived from the grammar \mathcal{G}' with LE_j as the start symbol.
 - i. If $m(\lambda_1) = 1$ and $m(\lambda_2) = 0$. Then, from induction hypothesis, we know that λ_1 is derived from the grammar \mathcal{G}' using LE_1 as the start symbol and λ_2 is derived from the grammar \mathcal{G}' using LE_0 as the start symbol. Then, the only way to derive λ from \mathcal{G}' is by using the start symbol LE_2 and then the production rule $\text{LE}_2 \rightarrow \text{LE}_1 + \text{LE}_0$.
 - ii. If $m(\lambda_1) = 0$ and $m(\lambda_2) = 1$. Then, from induction hypothesis, we know that λ_1 is derived from the grammar \mathcal{G}' using LE_0 as the start symbol and λ_2 is derived from the grammar \mathcal{G}' using LE_1

as the start symbol. Then, the only way to derive λ from \mathcal{G}' is by using the start symbol LE_2 and then the production rule $LE_2 \rightarrow LE_0 + LE_1$.

iii. We consider three cases:

- A. If $m(\lambda_1) = 2$ and $m(\lambda_2) \neq 2$, then we know from induction hypothesis, that λ_1 is derived from the grammar \mathcal{G}' using LE_2 as the start symbol and λ_2 is derived from the grammar \mathcal{G}' using LE_0 as the start symbol if $m(\lambda_2) = 0$, else it was derived using LE_1 as the start symbol. Then, the only way to derive λ from \mathcal{G}' is by using the start symbol LE_2 and then the production rule $LE_2 \rightarrow LE_2 + LE_0$ or the rule $LE_2 \rightarrow LE_2 + LE_1$ as the case maybe.
- B. If $m(\lambda_1) \neq 2$ and $m(\lambda_2) = 2$, then we know from induction hypothesis, that λ_2 is derived from the grammar \mathcal{G}' using LE_2 as the start symbol and λ_1 is derived from the grammar \mathcal{G}' using LE_0 as the start symbol if $m(\lambda_1) = 0$, else it was derived using LE_1 as the start symbol. Then, the only way to derive λ from \mathcal{G}' is by using the start symbol LE_2 and then the production rule $LE_2 \rightarrow LE_0 + LE_2$ or the rule $LE_2 \rightarrow LE_1 + LE_2$ as the case maybe.
- C. If $m(\lambda_1) = m(\lambda_2) = 2$, then we know from induction hypothesis, that both λ_1 and λ_2 are derived from the grammar \mathcal{G}' using LE_2 as the start symbol. The only way to derive λ from \mathcal{G}' is by using the start symbol LE_2 and then the production rule $LE_2 \rightarrow LE_2 + LE_2$.

If λ is generated from \mathcal{G}' using LE_2 as the start symbol, then it can be derived using one of the seven following production rules:

- i. $LE_2 \rightarrow LE_0 + LE_2$
- ii. $LE_2 \rightarrow LE_2 + LE_0$
- iii. $LE_2 \rightarrow LE_2 + LE_2$
- iv. $LE_2 \rightarrow LE_1 + LE_2$
- v. $LE_2 \rightarrow LE_2 + LE_1$
- vi. $LE_2 \rightarrow LE_0 + LE_1$
- vii. $LE_2 \rightarrow LE_1 + LE_0$

In each of the cases mentioned, it follows from the induction hypothesis and definition of m that $m(\lambda) = 2$.

2. If $\lambda = \neg(\lambda_1)$. We again consider three cases:

- (a) If $m(\lambda) = 0$, then from definition of m , we know that $m(\lambda_1) = 1$. We also know from induction hypothesis that λ_1 is derived from the start symbol LE_1 . Hence the production rule $LE_0 \rightarrow \neg LE_1$ generates λ . If λ is generated from the grammar \mathcal{G}' using the start symbol LE_0 , then from the grammar the only way to derive λ is by the production rule $LE_0 \rightarrow \neg LE_1$. From induction hypothesis and definition of m , it follows that $m(\lambda) = 0$.
- (b) If $m(\lambda) = 1$, then from definition of m , we know that $m(\lambda_1) = 0$. We also know from induction hypothesis that λ_1 is derived from the start symbol LE_0 , and the production rule $LE_1 \rightarrow \neg LE_0$ generates λ . If λ is generated from the grammar \mathcal{G}' using the start symbol LE_1 , then from the grammar the only way to derive λ is by the production rule $LE_1 \rightarrow \neg LE_0$. From induction hypothesis and definition of m , it follows that $m(\lambda) = 1$.
- (c) If $m(\lambda) = 2$, then from definition of m , we know that $m(\lambda_1) = 2$. We also know from induction hypothesis that λ_1 is derived from the start symbol LE_2 , and the production rule $LE_2 \rightarrow \neg LE_2$ generates λ . If λ is generated from the grammar \mathcal{G}' using the start symbol LE_2 , then from the grammar the only way to derive λ is by the production rule $LE_2 \rightarrow \neg LE_2$. From induction hypothesis and definition of m , it follows that $m(\lambda) = 2$.

3. If $\lambda = *\lambda_1$. We consider two cases:

- (a) If $m(\lambda) = 0$, then from definition of m , we know that $m(\lambda_1) = 0$. We also know from induction hypothesis that λ_1 is derived from the start symbol LE_0 , and the production rule $LE_0 \rightarrow *LE_0$ generates λ . If λ is generated from the grammar \mathcal{G}' using the start symbol LE_0 , then from the grammar

the only way to derive λ is by the production rule $LE_0 \rightarrow *LE_0$. From induction hypothesis and definition of m , it follows that $m(\lambda) = 0$.

- (b) If $m(\lambda) = 2$, then from definition of m , we know that $m(\lambda_1) \neq 0$. Suppose $m(\lambda_1) = 1$, then we also know from induction hypothesis λ_1 is derived from the start symbol LE_1 , and the production rule $LE_2 \rightarrow *LE_1$ generates λ . If λ is generated from the grammar \mathcal{G}' using the start symbol LE_2 , then from the grammar the only way to derive λ is by the production rule $LE_2 \rightarrow *LE_1$. From induction hypothesis and definition of m , it follows that $m(\lambda) = 2$.

Suppose $m(\lambda_1) = 2$, then we also know from induction hypothesis λ_1 is derived from the start symbol LE_2 , and the production rule $LE_2 \rightarrow *LE_2$ generates λ . If λ is generated from the grammar \mathcal{G}' using the start symbol LE_2 , then from the grammar the only way to derive λ is by the production rule $LE_2 \rightarrow *LE_2$. From induction hypothesis and definition of m it follows that $m(\lambda) = 2$.

4. If $\lambda = (\lambda_1)$. We consider three cases:

- (a) If $m(\lambda) = 1$, then from definition of m , we know that $m(\lambda_1) = 1$. We also know from induction hypothesis that λ_1 is derived from the start symbol LE_1 , and the production rule $LE_1 \rightarrow (LE_1)$ generates λ . If λ is generated from the grammar \mathcal{G}' using the start symbol LE_1 , then from the grammar the only way to derive λ is by the production rule $LE_1 \rightarrow (LE_1)$. From induction hypothesis and definition of m , it follows that $m(\lambda) = 1$.

- (b) If $m(\lambda) = 0$, then from definition of m , we know that $m(\lambda_1) = 0$. We also know from induction hypothesis that λ_1 is derived from the start symbol LE_0 , and the production rule $LE_0 \rightarrow (LE_0)$ generates λ . If λ is generated from the grammar \mathcal{G}' using the start symbol LE_0 , then from the grammar the only way to derive λ is by the production rule $LE_0 \rightarrow (LE_0)$. From induction hypothesis and definition of m , it follows that $m(\lambda) = 0$.

- (c) If $m(\lambda) = 2$, then from definition of m , we know that $m(\lambda_1) = 2$. We also know from induction hypothesis that λ_1 is derived from the start symbol LE_2 , and the production rule $LE_2 \rightarrow (LE_2)$ generates λ . If λ is generated from the grammar \mathcal{G}' using the start symbol LE_2 , then from the grammar the only way to derive λ is by the production rule $LE_2 \rightarrow (LE_2)$. From induction hypothesis and definition of m , it follows that $m(\lambda) = 2$.

5. If $\lambda = \lambda_1 \circ \lambda_2$. We consider three cases:

- (a) If $m(\lambda) = 0$, then by definition of m , we know that $m(\lambda_1) = m(\lambda_2)$ and $m(\lambda_1) \neq 2$. If $m(\lambda_1) = m(\lambda_2) = 0$, then from induction hypothesis, both λ_1 and λ_2 are derived using LE_0 as the start symbol from grammar \mathcal{G}' . The only way to derive λ from \mathcal{G}' is by the production rule $LE_0 \rightarrow LE_0 \circ LE_0$.

If $m(\lambda_1) = m(\lambda_2) = 1$, then from induction hypothesis, both λ_1 and λ_2 are derived using LE_1 as the start symbol from grammar \mathcal{G}' . The only way to derive λ from \mathcal{G}' is by the production rule $LE_0 \rightarrow LE_1 \circ LE_1$.

If λ is generated from grammar \mathcal{G}' using LE_0 as the start symbol, then there are two possible production rules to be used: either $LE_0 \rightarrow LE_0 \circ LE_0$ or $LE_0 \rightarrow LE_1 \circ LE_1$. From induction hypothesis and definition of m , it follows that $m(\lambda) = 0$.

- (b) If $m(\lambda) = 1$, then by definition of m , we know that either of the following cases hold true:

- i. $m(\lambda_1) = 0$ and $m(\lambda_2) = 1$. Then it follows from induction hypothesis that λ_1 was derived using LE_0 as the start symbol and λ_2 was derived using LE_1 as the start symbol. The only way to derive λ from \mathcal{G}' is by the production rule $LE_1 \rightarrow LE_0 \circ LE_1$.

- ii. $m(\lambda_1) = 1$ and $m(\lambda_2) = 0$. Then it follows from induction hypothesis that λ_1 was derived using LE_1 as the start symbol and λ_2 was derived using LE_0 as the start symbol. The only way to derive λ from \mathcal{G}' is by the production rule $LE_1 \rightarrow LE_1 \circ LE_0$.

If λ is generated from grammar \mathcal{G}' using LE_1 as the start symbol, then there are two possible production rules to be used: either $LE_1 \rightarrow LE_0 \circ LE_1$ or $LE_1 \rightarrow LE_1 \circ LE_0$. From induction hypothesis and definition of m , it follows that $m(\lambda) = 1$.

- (c) If $m(\lambda) = 2$, then by definition of m , atleast one of λ_1 and λ_2 must have an m - value of 2. We consider the following cases:
- i. If $m(\lambda_1) = m(\lambda_2) = 2$. It follows from induction hypothesis that both λ_1 and λ_2 have been derived using LE_2 as the start symbol. Then, the only way to derive λ is by the production rule $LE_2 \rightarrow LE_2 \circ LE_2$.
 - ii. If $m(\lambda_1) = 2$ and $m(\lambda_2) = 0$. It follows from induction hypothesis that λ_1 has been derived using LE_2 as the start symbol and λ_2 has been derived using LE_0 as the start symbol. Then, the only way to derive λ is by the production rule $LE_2 \rightarrow LE_2 \circ LE_0$.
 - iii. If $m(\lambda_1) = 0$ and $m(\lambda_2) = 2$. It follows from induction hypothesis that λ_1 has been derived using LE_0 as the start symbol and λ_2 has been derived using LE_2 as the start symbol. Then, the only way to derive λ is by the production rule $LE_2 \rightarrow LE_0 \circ LE_2$.
 - iv. If $m(\lambda_1) = 2$ and $m(\lambda_2) = 1$. It follows from induction hypothesis that λ_1 has been derived using LE_2 as the start symbol and λ_2 has been derived using LE_1 as the start symbol. Then, the only way to derive λ is by the production rule $LE_2 \rightarrow LE_2 \circ LE_1$.
 - v. If $m(\lambda_1) = 1$ and $m(\lambda_2) = 2$. It follows from induction hypothesis that λ_1 has been derived using LE_1 as the start symbol and λ_2 has been derived using LE_2 as the start symbol. Then, the only way to derive λ is by the production rule $LE_2 \rightarrow LE_1 \circ LE_2$.

If λ is generated from grammar \mathcal{G}' using LE_2 as the start symbol, then it is derived using one of the five following production rules:

- i. $LE_2 \rightarrow LE_2 \circ LE_2$.
- ii. $LE_2 \rightarrow LE_2 \circ LE_0$.
- iii. $LE_2 \rightarrow LE_0 \circ LE_2$.
- iv. $LE_2 \rightarrow LE_2 \circ LE_1$.
- v. $LE_2 \rightarrow LE_1 \circ LE_2$.

In each of the cases mentioned, it follows from the induction hypothesis and definition of m that $m(\lambda) = 2$.

6. If $\lambda = \lambda_1 ; \lambda_2$. The proof follows from the proofs of \circ and $+$ operators.

Hence proved.

Corollary 1. *Every LRE λ over $\mathcal{T}_{\Upsilon,n}$ which can be generated from the grammar \mathcal{G} can also be generated from the grammar \mathcal{G}' and every LRE λ' over $\mathcal{T}_{\Upsilon,n}$ which can be generated from the grammar \mathcal{G}' can also be generated from the grammar \mathcal{G} .*

- Proof.*
1. If λ can be generated from the grammar \mathcal{G} , then by definition $m(\lambda) \in \{0, 1, 2\}$ and always gets a unique value. Hence it can also be generated from the grammar \mathcal{G}' using $LE_{m(\lambda)}$ as the start symbol as shown in Lemma 11.
 2. If λ' is generated from the grammar \mathcal{G}' , then using the production rule $LE' \rightarrow LE_0 \mid LE_1 \mid LE_2$, λ' can be generated from either of these as the start symbol. Again using Lemma 11, λ' can also be generated from the grammar \mathcal{G} .

12 Examples

Example 20. Consider the STD of a transition system $B = (I, X, Q, S3, \mathbf{T})$ shown in Figure 20. Let λ be an LRE over $\mathcal{T}_{Y,n}$ such that

$$\begin{aligned}\lambda &= \lambda_1 + \lambda_2 \\ \lambda_1 &= \neg \left(\neg(1 : (\Upsilon_1)_1) + \neg(1 : (\Upsilon_1)_2) \right) \circ (2 : (\theta)_2) \\ \lambda_2 &= \neg \left(\neg(2 : (\Upsilon_2)_2) + \neg(2 : (\Upsilon_2)_1) \right) \circ (1 : (\theta)_1)\end{aligned}$$

Assume for ease of exposition that $n = 2$. We illustrate through this example that $\llbracket *\lambda \rrbracket (S_1, S_2) \succeq \llbracket *\lambda \rrbracket (S_1, S_2)$. With the use of $*\lambda$, we get a smaller state space as reachable.

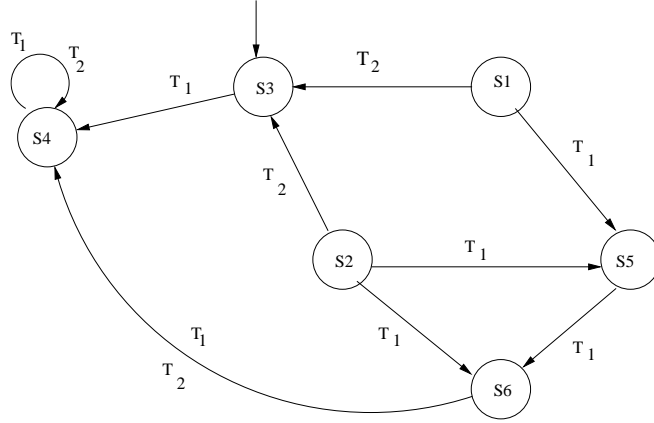


Fig. 4: STD for Example 20

Example 21. Let $\lambda \in SC_2$ be an LRE over $\mathcal{T}_{Y,n}$. Suppose we want to encode that only the frontier set of states (i.e. the new set of states seen upto now) is being used in some computation. Let To indicate the image of the current set of states and $Reached$ be the current set of *seen* states. To encode $To \wedge \overline{Reached}$ using the grammar \mathcal{G}' , we would use $\lambda = \neg(\lambda_1 + \lambda_2)$, where

$$\begin{aligned}\lambda_1 &= \neg(1 : (\Upsilon_1)_1) \\ \lambda_2 &= 1 : (\delta)_1\end{aligned}$$

Assumptions that S_1 stores the set $Reached$ before the evaluation of λ on \overline{S} . Also need to mention that the length of the State Set Vectors is kept small for ease of exposition. Explain also why this cannot be written using any other production rule. Also mention that we have not yet tried out such encodings in our experiments, but this would be a crucial step in encoding reachability algorithms.

Example 22. Let $\lambda \in SC_0$ be an LRE over $\mathcal{T}_{Y,n}$. An useful LRE using the production rule $LE_0 \rightarrow \neg LE_1$ would be $\lambda = \neg(\neg(1 : (\delta)_1) + \neg(1 : (\delta)_2))$. $\llbracket \lambda \rrbracket (S_1(X), S_2(X)) = (S_1(X) \wedge S_2(X), S_2(X))$.

Example 23. Let $\lambda \in SC_0$ be an LRE over $\mathcal{T}_{r,n}$. An useful LRE using the production rule $LE_0 \rightarrow LE_1 \circ LE_1$ would be

$$\begin{aligned}\lambda &= \lambda_1 \circ \lambda_2 \quad \text{where} \\ \lambda_1 &= \neg 1 : (\delta)_1 + \neg 1 : (\delta)_2 + \neg 3 : (\delta)_3 + \neg 3 : (\delta)_4 \\ \lambda_2 &= \neg 1 : (\delta)_1 + \neg 1 : (\delta)_3\end{aligned}$$

$$\llbracket \lambda_1 \circ \lambda_2 \rrbracket (S_1(X), S_2(X), S_3(X), S_4(X)) = ((S_1(X) \wedge S_2(X)) \vee (S_3(X) \wedge S_4(X)), S_2(X), S_3(X) \wedge S_4(X), S_4(X)).$$

13 Properties of LREs

We also prove some other properties. Let $\bar{S}(X)$ be a State Set Vector of length n .

Property 1. $\lambda + \lambda = \lambda$

Proof. We know that

$$\begin{aligned}\llbracket \lambda + \lambda \rrbracket (\bar{S}(X)) &= \llbracket \lambda \rrbracket (\bar{S}(X)) \sqcup \llbracket \lambda \rrbracket (\bar{S}(X)) \quad (\text{From LRE semantics}) \\ &= \llbracket \lambda \rrbracket (\bar{S}(X)) \quad (\text{From definition of } \sqcup)\end{aligned}$$

Hence $\lambda + \lambda = \lambda$. In other words, $+$ is an idempotent operator.

Property 2. $\lambda_1 + \lambda_2 = \lambda_2 + \lambda_1$

Proof. We know that

$$\begin{aligned}\llbracket \lambda_1 + \lambda_2 \rrbracket (\bar{S}(X)) &= \llbracket \lambda_1 \rrbracket (\bar{S}(X)) \sqcup \llbracket \lambda_2 \rrbracket (\bar{S}(X)) \quad (\text{From LRE semantics}) \\ &= \llbracket \lambda_2 \rrbracket (\bar{S}(X)) \sqcup \llbracket \lambda_1 \rrbracket (\bar{S}(X)) \quad (\text{From commutativity of } \sqcup)\end{aligned}$$

Hence $\lambda_1 + \lambda_2 = \lambda_2 + \lambda_1$. In other words the operator $+$ is commutative.

Property 3. $\lambda + \hat{\theta} = \lambda$

Proof. We know that

$$\begin{aligned}\llbracket \lambda + \hat{\theta} \rrbracket (\bar{S}(X)) &= \llbracket \lambda \rrbracket (\bar{S}(X)) \sqcup \llbracket \hat{\theta} \rrbracket (\bar{S}(X)) \quad (\text{From LRE semantics}) \\ &= \llbracket \lambda \rrbracket (\bar{S}(X)) \sqcup (0, \dots, 0) \quad (\text{From LRE semantics}) \\ &= \llbracket \lambda \rrbracket (\bar{S}(X))\end{aligned}$$

Hence $\lambda + \hat{\theta} = \lambda$. In other words, $\hat{\theta}$ is the *additive identity* LRE.

Property 4. $\lambda_1 + (\lambda_2 + \lambda_3) = (\lambda_1 + \lambda_2) + \lambda_3$

Proof. We know that

$$\begin{aligned}\llbracket \lambda_1 + (\lambda_2 + \lambda_3) \rrbracket (\bar{S}(X)) &= \llbracket \lambda_1 \rrbracket (\bar{S}(X)) \sqcup \llbracket (\lambda_2 + \lambda_3) \rrbracket (\bar{S}(X)) \quad (\text{From LRE semantics}) \\ &= \llbracket \lambda_1 \rrbracket (\bar{S}(X)) \sqcup \llbracket \lambda_2 \rrbracket (\bar{S}(X)) \sqcup \llbracket \lambda_3 \rrbracket (\bar{S}(X)) \quad (\text{From LRE semantics}) \\ &= \llbracket \lambda_1 + \lambda_2 \rrbracket (\bar{S}(X)) \sqcup \llbracket \lambda_3 \rrbracket (\bar{S}(X)) \quad (\text{From associativity of } \sqcup) \\ &= \llbracket (\lambda_1 + \lambda_2) + \lambda_3 \rrbracket (\bar{S}(X))\end{aligned}$$

Hence $\lambda_1 + (\lambda_2 + \lambda_3) = (\lambda_1 + \lambda_2) + \lambda_3$. In other words, the operator $+$ is associative.

Property 5. $\lambda \circ \hat{\delta} = \hat{\delta} \circ \lambda = \lambda$

Proof. We know that

$$\begin{aligned} \llbracket \lambda + \hat{\delta} \rrbracket (\overline{S}(X)) &= \llbracket \hat{\delta} \rrbracket (\llbracket \lambda \rrbracket (\overline{S}(X))) \quad (\text{From LRE semantics}) \\ &= \llbracket \lambda \rrbracket (\overline{S}(X)) \quad (\text{From LRE semantics}) \end{aligned}$$

$$\begin{aligned} \text{Also, } \llbracket \hat{\delta} \circ \lambda \rrbracket (\overline{S}(X)) &= \llbracket \lambda \rrbracket (\llbracket \hat{\delta} \rrbracket (\overline{S}(X))) \quad (\text{From LRE semantics}) \\ &= \llbracket \lambda \rrbracket (\overline{S}(X)) \quad (\text{From LRE semantics}) \end{aligned}$$

Hence $\lambda \circ \hat{\delta} = \hat{\delta} \circ \lambda = \lambda$. In other words, $\hat{\delta}$ is the *multiplicative identity* LRE.

Property 6. $\lambda \circ \hat{\theta} = \hat{\theta} \circ \lambda = \hat{\theta}$

Proof. We know that

$$\begin{aligned} \llbracket \lambda \circ \hat{\theta} \rrbracket (\overline{S}(X)) &= \llbracket \hat{\theta} \rrbracket (\llbracket \lambda \rrbracket (\overline{S}(X))) \quad (\text{From LRE semantics}) \\ &= (0, \dots, 0) \quad (\text{From LRE semantics}) \\ &= \llbracket \hat{\theta} \rrbracket (\overline{S}(X)) \quad (\text{From LRE semantics}) \end{aligned}$$

$$\begin{aligned} \text{Also, } \llbracket \hat{\theta} \circ \lambda \rrbracket (\overline{S}(X)) &= \llbracket \lambda \rrbracket (\llbracket \hat{\theta} \rrbracket (\overline{S}(X))) \quad (\text{From LRE semantics}) \\ &= \llbracket \lambda \rrbracket (0, \dots, 0) \quad (\text{From LRE semantics}) \\ &= (0, \dots, 0) \\ &= \llbracket \hat{\theta} \rrbracket (\overline{S}(X)) \quad (\text{From LRE semantics}) \end{aligned}$$

Hence $\lambda \circ \hat{\theta} = \hat{\theta} \circ \lambda = \hat{\theta}$.

Property 7. $\lambda_1 \circ (\lambda_2 \circ \lambda_3) = (\lambda_1 \circ \lambda_2) \circ \lambda_3$

Proof. We know that

$$\begin{aligned} \llbracket \lambda_1 \circ (\lambda_2 \circ \lambda_3) \rrbracket (\overline{S}(X)) &= \llbracket (\lambda_2 \circ \lambda_3) \rrbracket (\llbracket \lambda_1 \rrbracket (\overline{S}(X))) \quad (\text{From LRE semantics}) \\ &= \llbracket \lambda_3 \rrbracket (\llbracket \lambda_2 \rrbracket (\llbracket \lambda_1 \rrbracket (\overline{S}(X)))) \quad (\text{From LRE semantics}) \end{aligned}$$

$$\begin{aligned} \text{Also, } \llbracket (\lambda_1 \circ \lambda_2) \circ \lambda_3 \rrbracket (\overline{S}(X)) &= \llbracket \lambda_3 \rrbracket (\llbracket (\lambda_1 \circ \lambda_2) \rrbracket (\overline{S}(X))) \quad (\text{From LRE semantics}) \\ &= \llbracket \lambda_3 \rrbracket (\llbracket \lambda_2 \rrbracket (\llbracket \lambda_1 \rrbracket (\overline{S}(X)))) \quad (\text{From LRE semantics}) \end{aligned}$$

Hence $\lambda_1 \circ (\lambda_2 \circ \lambda_3) = (\lambda_1 \circ \lambda_2) \circ \lambda_3$. In other words, the operator \circ is associative.

Property 8. $\lambda_1 \circ (\lambda_2 + \lambda_3) = (\lambda_1 \circ \lambda_2) + (\lambda_1 \circ \lambda_3)$

Proof. We know that

$$\begin{aligned} \llbracket \lambda_1 \circ (\lambda_2 + \lambda_3) \rrbracket (\overline{S}(X)) &= \llbracket \lambda_2 + \lambda_3 \rrbracket (\llbracket \lambda_1 \rrbracket (\overline{S}(X))) \quad (\text{From LRE semantics}) \\ &= \llbracket \lambda_2 \rrbracket (\llbracket \lambda_1 \rrbracket (\overline{S}(X))) \sqcup \llbracket \lambda_3 \rrbracket (\llbracket \lambda_1 \rrbracket (\overline{S}(X))) \quad (\text{From LRE semantics}) \\ &= \llbracket \lambda_1 \circ \lambda_2 \rrbracket (\overline{S}(X)) \sqcup \llbracket \lambda_1 \circ \lambda_3 \rrbracket (\overline{S}(X)) \quad \text{From LRE semantics} \\ &= \llbracket (\lambda_1 \circ \lambda_2) + (\lambda_1 \circ \lambda_3) \rrbracket (\overline{S}(X)) \quad (\text{From LRE semantics}) \end{aligned}$$

Hence $\lambda_1 \circ (\lambda_2 + \lambda_3) = (\lambda_1 \circ \lambda_2) + (\lambda_1 \circ \lambda_3)$. In other words, the operator \circ distributes over $+$ on the left.

Property 9. $\hat{\delta} + \lambda_1; \lambda_2 = \lambda_1; \lambda_2$.

Proof. We know from definition of LRE semantics that

$$\begin{aligned} \lambda_1; \lambda_2 &= (\lambda_1 + \hat{\delta}) \circ (\lambda_2 + \hat{\delta}) \\ \text{Hence, } \hat{\delta} + \lambda_1; \lambda_2 &= \hat{\delta} + ((\lambda_1 + \hat{\delta}) \circ (\lambda_2 + \hat{\delta})) \\ &= \hat{\delta} + (\lambda_1 + \hat{\delta}) \circ \lambda_2 + \lambda_1 + \hat{\delta} \quad \text{From Property 8} \\ &= (\lambda_1 + \hat{\delta}) \circ \lambda_2 + \lambda_1 + \hat{\delta} + \hat{\delta} \quad \text{From Property 2} \\ &= (\lambda_1 + \hat{\delta}) \circ \lambda_2 + \lambda_1 + \hat{\delta} \quad \text{From Property 1} \\ &= \lambda_1; \lambda_2 \quad \text{From LRE semantics} \end{aligned}$$

Property 10. $\lambda_1; (\lambda_2; \lambda_3) = (\lambda_1; \lambda_2); \lambda_3$

Proof.

$$\begin{aligned} \lambda_1; (\lambda_2; \lambda_3) &= (\lambda_1 + \hat{\delta}) \circ ((\lambda_2; \lambda_3) + \hat{\delta}) \quad \text{From LRE semantics} \\ &= (\lambda_1 + \hat{\delta}) \circ (\lambda_2; \lambda_2) \quad \text{From Property 9} \\ &= (\lambda_1 + \hat{\delta}) \circ ((\lambda_2 + \hat{\delta}) \circ (\lambda_3 + \hat{\delta})) \quad \text{From LRE semantics} \\ &= ((\lambda_1 + \hat{\delta}) \circ (\lambda_2 + \hat{\delta})) \circ (\lambda_3 + \hat{\delta}) \quad \text{From Property 7} \\ &= (\lambda_1; \lambda_2); \lambda_3 \end{aligned}$$

Hence proved. In other words, the operator $;$ is associative.

Property 11. $\hat{\theta}; \lambda = \lambda; \hat{\theta} = \lambda + \hat{\delta}$

Proof.

$$\begin{aligned} \llbracket \hat{\theta}; \lambda \rrbracket (\overline{S}(X)) &= \llbracket (\hat{\theta} + \hat{\delta}) \circ (\lambda + \hat{\delta}) \rrbracket (\overline{S}(X)) \quad \text{From LRE semantics} \\ &= \llbracket \hat{\delta} \circ (\lambda + \hat{\delta}) \rrbracket (\overline{S}(X)) \quad \text{From Property 3} \\ &= \llbracket \lambda + \hat{\delta} \rrbracket (\overline{S}(X)) \quad \text{From Property 5} \end{aligned}$$

$$\begin{aligned} \text{Also, } \llbracket \lambda; \hat{\theta} \rrbracket (\overline{S}(X)) &= \llbracket (\lambda + \hat{\delta}) \circ (\hat{\theta} + \hat{\delta}) \rrbracket (\overline{S}(X)) \quad \text{From LRE semantics} \\ &= \llbracket (\lambda + \hat{\delta}) \circ \hat{\delta} \rrbracket (\overline{S}(X)) \quad \text{From Property 3} \\ &= \llbracket \lambda + \hat{\delta} \rrbracket (\overline{S}(X)) \quad \text{From Property 5} \end{aligned}$$

Hence proved.

Definition 27. *Write set of an LRE:* The Write set of an LRE λ over $\mathcal{T}_{\Gamma,n}$, denoted $W(\lambda)$ gives the set of labels corresponding to the components of the State Set Vector \overline{S} of length n which are potentially updated while evaluating λ on \overline{S} .

Given an LRE λ over $\mathcal{T}_{r,n}$, we provide an inductive definition of $W(\lambda)$ as

1. If $\lambda = j : (\delta)_i$ and $j \neq i$, then $W(\lambda) = \{j\}$.
2. If $\lambda = j : (\theta)_i$ then $W(\lambda) = \{j\}$.
3. If $\lambda = j : (\Upsilon_k)_i$ then $W(\lambda) = \{j\}$.
4. If $\lambda = j : (\Upsilon_k \Downarrow_{X'} [\Upsilon_p, \dots, \Upsilon_q, u, \dots, v])_i$ then $W(\lambda) = \{j\}$.
5. If $\lambda = \lambda_1 + \lambda_2$ then $W(\lambda) = W(\lambda_1) \cup W(\lambda_2)$.
6. If $\lambda = \lambda_1 \circ \lambda_2$ then $W(\lambda) = W(\lambda_1) \cup W(\lambda_2)$.
7. If $\lambda = \lambda_1 ; \lambda_2$ then $W(\lambda) = W\left((\lambda_1 + 1 : (\delta)_1) \circ (\lambda_2 + 1 : (\delta)_1)\right) = W(\lambda_1 + 1 : (\delta)_1) \cup W(\lambda_2 + 1 : (\delta)_1) = W(\lambda_1) \cup W(\lambda_2)$.
8. If $\lambda = (\lambda_1)$ then $W(\lambda) = W(\lambda_1)$.
9. If $\lambda = *\lambda_1$ then $W(\lambda) = W(\lambda_1)$.

Justification: We know from definition that $*\lambda_1 = (1 : (\delta)_1 + \lambda_1 + (\lambda_1)^2 + \dots)$. Also, $W(1 : (\delta)_1) = \emptyset$ from step 1, and for any $k \geq 1$, $W((\lambda_1)^k) = W(\lambda_1)$ from step 6. Hence $W(*\lambda_1) = W(1 : (\delta)_1) \cup W(\lambda_1) = W(\lambda_1)$.

10. If $\lambda = \neg\lambda_1$ then $W(\lambda) = \{1, \dots, n\}$.

Justification: We know from definition of LRE semantics that the \neg operator negates (complements) the values of every component of the State Set Vector on which the expression λ is evaluated. Hence, as it changes the values of every state component, its write set includes all the labels.

Definition 28. *Read set of an LRE:* The Read set of an LRE λ over $\mathcal{T}_{r,n}$, denoted $R(\lambda)$ gives a set of labels corresponding to the components of a State Set Vector \bar{S} of length n whose values are used to potentially change the values of some component in the State Set Vector resulting from the evaluation of λ on \bar{S} .

Given an LRE λ over $\mathcal{T}_{r,n}$, we inductively define $R(\lambda)$ as

1. If $\lambda = j : (\delta)_i$ and $j \neq i$, then $R(\lambda) = \{i\}$.
 2. If $\lambda = j : (\theta)_i$ then $R(\lambda) = \emptyset$.
 3. If $\lambda = j : (\Upsilon_k)_i$ then $R(\lambda) = \{i\}$.
 4. If $\lambda = j : (\Upsilon_k \Downarrow_{X'} [\Upsilon_p, \dots, \Upsilon_q, u, \dots, v])_i$ then $R(\lambda) = \{i, u, \dots, v\}$.
 5. If $\lambda = \lambda_1 + \lambda_2$ then $R(\lambda) = R(\lambda_1) \cup R(\lambda_2)$.
 6. If $\lambda = \lambda_1 \circ \lambda_2$ then $R(\lambda) = R(\lambda_1) \cup (R(\lambda_2) \setminus W(\lambda_1))$.
- Justification:** Consider a State Set Vector \bar{S} . We know from LRE semantics that $\llbracket \lambda_1 \circ \lambda_2 \rrbracket (\bar{S}(X)) = \llbracket \lambda_2 \rrbracket (\llbracket \lambda_1 \rrbracket (\bar{S}))$. So $R(\lambda_1) \subseteq R(\lambda)$. Also, it is enough to consider only those labels from $R(\lambda_2)$ which have not been potentially changed by the application of λ_1 on \bar{S} . Hence, $R(\lambda) = R(\lambda_1) \cup (R(\lambda_2) \setminus W(\lambda_1))$.
7. If $\lambda = \lambda_1 ; \lambda_2$ then $R(\lambda) = R\left((\lambda_1 + (1 : (\delta)_1)) \circ (\lambda_2 + (1 : (\delta)_1))\right) = R(\lambda_1) \cup (R(\lambda_2) \setminus W(\lambda_1))$ from cases 6, 5 and 1.
 8. If $\lambda = (\lambda_1)$ then $R(\lambda) = R(\lambda_1)$.
 9. If $\lambda = *\lambda_1$ then $R(\lambda) = R(\lambda_1)$.
- Justification:** We know from definition that $*\lambda_1 = (1 : (\delta)_1 + \lambda_1 + (\lambda_1)^2 + \dots)$. Also, $R(1 : (\delta)_1) = \emptyset$ from step 1, and for any $k \geq 1$, $R((\lambda_1)^k) = R(\lambda_1)$ from step 6. Hence $R(*\lambda_1) = R(1 : (\delta)_1) \cup R(\lambda_1) = R(\lambda_1)$.
10. If $\lambda = \neg\lambda_1$ then $R(\lambda) = \{1, \dots, n\}$.

Justification: We know from definition that $\neg\lambda_1$ changes the values of all components of the State Set Vector $\bar{S}(X)$ on which it is evaluated. Hence, all labels belong to the read set.

Lemma 12. Let λ be an LRE over $\mathcal{T}_{\Upsilon,n}$ and a $\bar{S}(X)$ be a State Set Vector of length n . If $i \notin W(\lambda)$, then $(\llbracket \lambda \rrbracket (\bar{S}(X)))_i = S_i(X)$.

Proof. Follows from definition of LRE semantics and Write set.

Lemma 13. Let λ be an LRE over $\mathcal{T}_{\Upsilon,n}$. Given State Set Vectors $\bar{S}(X)$ and $\bar{P}(X)$, if $\forall k \in R(\lambda), S_k(X) = P_k(X)$, then $\forall j \in (W(\lambda) \cup R(\lambda)), (\llbracket \lambda \rrbracket (\bar{S}(X)))_j = (\llbracket \lambda \rrbracket (\bar{P}(X)))_j$.

Proof. We consider the following cases:

1. If $j \in R(\lambda)$ and $j \notin W(\lambda)$, then $(\llbracket \lambda \rrbracket (\bar{S}(X)))_j = (\llbracket \lambda \rrbracket (\bar{P}(X)))_j$ follows from Lemma 12 and given condition.
2. If $j \in W(\lambda)$, then $(\llbracket \lambda \rrbracket (\bar{S}(X)))_j = (\llbracket \lambda \rrbracket (\bar{P}(X)))_j$ follows from LRE semantics and given condition.

Hence proved.

Corollary 2. Let λ be an LRE over $\mathcal{T}_{\Upsilon,n}$ and $\bar{S}(X)$ and $\bar{P}(X)$ be State Set Vectors of length n each. If $i \notin R(\lambda)$, and $\forall j \neq i, S_j(X) = P_j(X)$, then $\forall k \neq i, (\llbracket \lambda \rrbracket (\bar{S}(X)))_k = (\llbracket \lambda \rrbracket (\bar{P}(X)))_k$.

Proof. Follows from Lemma 12 and Lemma 13.

Example 24. We illustrate some special cases of the LREs $\hat{\delta}$ and $\hat{\theta}$.

LRE λ	$R(\lambda)$	$W(\lambda)$
$\hat{\delta}$	\emptyset	\emptyset
$\hat{\theta}$	\emptyset	$\{1, 2, \dots, n\}$

Corollary 3. Let λ be an LRE over $\mathcal{T}_{\Upsilon,n}$. Then,

1. $W(\lambda + \hat{\delta}) = W(\lambda)$
2. $R(\lambda + \hat{\delta}) = R(\lambda)$
3. $W(\lambda + \hat{\theta}) = \{1, \dots, n\}$
4. $R(\lambda + \hat{\theta}) = R(\lambda)$

Proof. Follows from definition of read and write sets.

Property 12. Let λ_1, λ_2 be LREs over $\mathcal{T}_{\Upsilon,n}$. If $W(\lambda_1) \subseteq W(\lambda_2)$ and $(R(\lambda_2) \cap W(\lambda_1)) = \emptyset$, then

1. $\lambda_1 \circ \lambda_2 = \lambda_2$.
2. $\lambda_1 ; \lambda_2 = \lambda_2 + \hat{\delta}$.

Proof. 1. We consider two cases:

- (a) If $i \notin W(\lambda_2)$, then $i \notin W(\lambda_1)$ also. Hence, from Lemma 12, we have

$$(\llbracket \lambda_2 \rrbracket (\bar{S}(X)))_i = S_i(X)$$

$$(\llbracket \lambda_1 \rrbracket (\bar{S}(X)))_i = S_i(X)$$

$$\text{And, } (\llbracket \lambda_2 \rrbracket (\llbracket \lambda_1 \rrbracket (\bar{S}(X))))_i = (\llbracket \lambda_1 \rrbracket (\bar{S}(X)))_i$$

Hence, $\lambda_1 \circ \lambda_2 = \lambda_2$.

(b) If $i \in W(\lambda_2)$, then if $i \notin W(\lambda_1)$, then from Lemma 12, we have $(\llbracket \lambda_2 \rrbracket (\llbracket \lambda_1 \rrbracket (\overline{S}(X))))_i = (\llbracket \lambda_1 \rrbracket (\overline{S}(X)))_i = S_i(X)$. Else if $i \in W(\lambda_1)$ also, then $i \notin R(\lambda_2)$, then from Corollary 2, we have $(\llbracket \lambda_2 \rrbracket (\llbracket \lambda_1 \rrbracket (\overline{S}(X))))_i = (\llbracket \lambda_2 \rrbracket (\overline{S}(X)))_i$. Hence, $\lambda_1 \circ \lambda_2 = \lambda_2$.

Hence proved.

2. We know from definition that $\lambda_1; \lambda_2 = (\lambda_1 + \hat{\delta}) \circ (\lambda_2 + \hat{\delta})$. Also from Corollary 3, we know $W(\lambda_1 + \hat{\delta}) = W(\lambda_1)$, $W(\lambda_2 + \hat{\delta}) = W(\lambda_2)$, $R(\lambda_1 + \hat{\delta}) = R(\lambda_1)$, and $R(\lambda_2 + \hat{\delta}) = R(\lambda_2)$. The result follows from these and case 1.

Property 13. Let λ be an LRE over $\mathcal{T}_{\Gamma, n}$. If $R(\lambda) \cap W(\lambda) = \emptyset$, then

1. $\lambda \circ \lambda = \lambda$
2. $(\lambda + \hat{\delta}) \circ \lambda = \lambda$
3. $\forall m \geq 1, \lambda^m = \lambda$
4. $*\lambda = \lambda; \lambda = \hat{\delta} + \lambda$

Proof. Let $\overline{S}(X)$ be a State Set Vector \overline{S} of length n .

1. (a) We know from Lemma 12 that $\forall i \notin W(\lambda)$,

$$(\llbracket \lambda \rrbracket (\overline{S}(X)))_i = S_i(X)$$

$$\text{Also, } (\llbracket \lambda \rrbracket (\llbracket \lambda \rrbracket (\overline{S}(X))))_i = (\llbracket \lambda \rrbracket (\overline{S}(X)))_i = S_i(X)$$

$$\text{Hence, } \lambda \circ \lambda = \lambda \quad \text{From LRE semantics}$$

- (b) $\forall i \in W(\lambda)$, we know that $i \notin R(\lambda)$. Then, from Lemma 13

$(\llbracket \lambda \rrbracket (\overline{S}(X)))_i = (\llbracket \lambda \rrbracket (\llbracket \lambda \rrbracket (\overline{S}(X))))_i$. Also from the previous case we know that $\forall k \notin W(\lambda)$, we have $\lambda \circ \lambda = \lambda$.

Hence proved.

2. From Corollary 3, we know $R(\lambda + \hat{\delta}) = R(\lambda)$, and $W(\lambda + \hat{\delta}) = W(\lambda)$. Hence the property follows from Property 12.
3. We prove by induction on m .

Basis: When $m = 1$, it holds trivially. When $m = 2$, it follows from case 1b.

Induction Hypothesis: Assume $\forall 1 \leq i \leq m, \lambda^i = \lambda$.

Induction Step: We know that $\lambda^{m+1} = \lambda^m \circ \lambda = \lambda \circ \lambda = \lambda$ from hypothesis.

Hence proved.

- 4.

$$\begin{aligned} *\lambda &= \hat{\delta} + \lambda + (\lambda)^2 + \dots \quad \text{From definition} \\ &= \hat{\delta} + \lambda + \lambda + \dots \quad \text{From cases 1a and 3} \\ &= \hat{\delta} + \lambda \quad \text{From Property 1} \end{aligned}$$

Next, we prove that $\lambda; \lambda = \hat{\delta} + \lambda$.

$$\begin{aligned} \lambda; \lambda &= (\lambda + \hat{\delta}) \circ (\lambda + \hat{\delta}) \quad \text{From definition} \\ &= \hat{\delta} + \lambda \quad \text{From Corollary 3, case 1a and Property 1} \end{aligned}$$

Property 14. Let λ_1, λ_2 be LREs over $\mathcal{T}_{\Gamma, n}$. If $\lambda_1 \preceq \lambda_2$, then $\neg\lambda_1 \succeq \neg\lambda_2$.

Proof. Let $\bar{S}(X)$ be a State Set Vector of length n . We have

$$\begin{aligned} \llbracket \lambda_1 \rrbracket (\bar{S}(X)) &\preceq \llbracket \lambda_2 \rrbracket (\bar{S}(X)) && \text{Since } \lambda_1 \preceq \lambda_2 \\ \text{So, } \neg \llbracket \lambda_1 \rrbracket (\bar{S}(X)) &\succeq \neg \llbracket \lambda_2 \rrbracket (\bar{S}(X)) && \text{From the definition of } \neg \\ \text{Hence, } \llbracket \neg \lambda_1 \rrbracket (\bar{S}(X)) &\succeq \llbracket \neg \lambda_2 \rrbracket (\bar{S}(X)) && \text{From LRE semantics} \end{aligned}$$

Hence proved.

Property 15. Let λ be an LRE over $\mathcal{T}_{\Gamma,n}$. Then, $\neg(\neg(\lambda)) = \lambda$.

Proof. Let $\bar{S}(X)$ be a State Set Vector of length n . Then,

$$\begin{aligned} \llbracket \neg(\neg(\lambda)) \rrbracket (\bar{S}(X)) &= \neg(\neg \llbracket \lambda \rrbracket (\bar{S}(X))) && \text{From LRE semantics} \\ &= \llbracket \lambda \rrbracket (\bar{S}(X)) && \text{From definition of } \neg \end{aligned}$$

Hence proved.

Property 16. Let λ_1, λ_2 be LREs over $\mathcal{T}_{\Gamma,n}$. Then, $\neg(\lambda_1 \circ \lambda_2) = \lambda_1 \circ \neg \lambda_2$.

Proof. Let $\bar{S}(X)$ be a State Set Vector of length n . Then,

$$\begin{aligned} \llbracket \lambda_1 \circ \neg \lambda_2 \rrbracket (\bar{S}(X)) &= \llbracket \neg \lambda_2 \rrbracket (\llbracket \lambda_1 \rrbracket (\bar{S}(X))) && \text{From LRE semantics} \\ &= \neg \llbracket \lambda_2 \rrbracket (\llbracket \lambda_1 \rrbracket (\bar{S}(X))) && \text{From LRE semantics} \\ &= \neg \llbracket \lambda_1 \circ \lambda_2 \rrbracket (\bar{S}(X)) && \text{From LRE semantics} \\ &= \llbracket \neg(\lambda_1 \circ \lambda_2) \rrbracket (\bar{S}(X)) && \text{From LRE semantics} \end{aligned}$$

Hence proved.

13.1 Properties of LREs belonging to either subclass 0 or subclass 1

Property 17. Let λ_1, λ_2 be LREs over $\mathcal{T}_{\Gamma,n}$. If $\lambda_1, \lambda_2 \in SC_0$, then for State Set Vectors $\bar{S}(X)$ and $\bar{P}(X)$ of length n each,

$$\llbracket \lambda_1 \rrbracket (\bar{S}(X)) \sqcup \llbracket \lambda_2 \rrbracket (\bar{P}(X)) \preceq \llbracket \lambda_1 + \lambda_2 \rrbracket (\bar{S}(X) \sqcup \bar{P}(X))$$

Proof.

$$\begin{aligned} \bar{S}(X) &\preceq \bar{S}(X) \sqcup \bar{P}(X) \\ \bar{P}(X) &\preceq \bar{S}(X) \sqcup \bar{P}(X) \\ \text{Since } \lambda_1, \lambda_2 &\in SC_0, \llbracket \lambda_1 \rrbracket (\bar{S}(X)) \preceq \llbracket \lambda_1 \rrbracket (\bar{S}(X) \sqcup \bar{P}(X)) \\ \text{and, } \llbracket \lambda_2 \rrbracket (\bar{P}(X)) &\preceq \llbracket \lambda_2 \rrbracket (\bar{S}(X) \sqcup \bar{P}(X)) \\ \text{Hence, } \llbracket \lambda_1 \rrbracket (\bar{S}(X)) \sqcup \llbracket \lambda_2 \rrbracket (\bar{P}(X)) &\preceq \llbracket \lambda_1 + \lambda_2 \rrbracket (\bar{S}(X) \sqcup \bar{P}(X)) \\ &\text{From monotonicity of } \sqcup \end{aligned}$$

Property 18. Let λ_1, λ_2 be LREs over $\mathcal{T}_{\Gamma,n}$. If $\lambda_1, \lambda_2 \in SC_1$, then for State Set Vectors $\bar{S}(X)$ and $\bar{P}(X)$ of length n each,

$$\llbracket \lambda_1 \rrbracket (\bar{S}(X)) \sqcup \llbracket \lambda_2 \rrbracket (\bar{P}(X)) \succeq \llbracket \lambda_1 + \lambda_2 \rrbracket (\bar{S}(X) \sqcup \bar{P}(X))$$

Proof.

$$\begin{aligned}
\bar{S}(X) &\preceq \bar{S}(X) \sqcup \bar{P}(X) \\
\bar{P}(X) &\preceq \bar{S}(X) \sqcup \bar{P}(X) \\
\text{Since } \lambda_1, \lambda_2 &\in SC_1, \llbracket \lambda_1 \rrbracket (\bar{S}(X)) \succeq \llbracket \lambda_1 \rrbracket (\bar{S}(X) \sqcup \bar{P}(X)) \\
&\text{and, } \llbracket \lambda_2 \rrbracket (\bar{S}(X)) \succeq \llbracket \lambda_2 \rrbracket (\bar{S}(X) \sqcup \bar{P}(X)) \\
\text{Hence, } \llbracket \lambda_1 \rrbracket (\bar{S}(X)) \sqcup \llbracket \lambda_2 \rrbracket (\bar{P}(X)) &\succeq \llbracket \lambda_1 + \lambda_2 \rrbracket (\bar{S}(X) \sqcup \bar{P}(X)) \\
&\text{From monotonicity of } \sqcup
\end{aligned}$$

Definition 29. Given an LRE λ over $\mathcal{T}_{Y,n}$ and a State Set Vector $\bar{S}(X)$ of length n for a transition system B , $\llbracket \lambda \rrbracket_B$ is both monotonically increasing and monotonically decreasing if $\llbracket \lambda \rrbracket_B(\bar{S}(X))$ evaluates to a constant State Set Vector i.e. which only depends on the underlying transition system B .

Example 25. $\lambda = \hat{\theta}$ is an example of an LRE which is both monotonically increasing and monotonically decreasing.

Property 19. Let $\lambda_1, \lambda_2, \lambda_3$ be LREs over $\mathcal{T}_{Y,n}$.

1. If $\lambda_3 \in SC_0$, then $(\lambda_1 + \lambda_2) \circ \lambda_3 \succeq (\lambda_1 \circ \lambda_3) + (\lambda_2 \circ \lambda_3)$
2. If $\lambda_3 \in SC_1$, then $(\lambda_1 + \lambda_2) \circ \lambda_3 \preceq (\lambda_1 \circ \lambda_3) + (\lambda_2 \circ \lambda_3)$
3. If $\llbracket \lambda_3 \rrbracket$ is both monotonically increasing and monotonically decreasing, then $(\lambda_1 + \lambda_2) \circ \lambda_3 = (\lambda_1 \circ \lambda_3) + (\lambda_2 \circ \lambda_3)$

Proof. 1. Let $\bar{S}(X)$ be a State Set Vector of length n . From LRE semantics, we get

$$\begin{aligned}
\llbracket (\lambda_1 + \lambda_2) \circ \lambda_3 \rrbracket (\bar{S}(X)) &= \llbracket \lambda_3 \rrbracket (\llbracket \lambda_1 + \lambda_2 \rrbracket (\bar{S})) \\
&= \llbracket \lambda_3 \rrbracket (\llbracket \lambda_1 \rrbracket (\bar{S}(X)) \sqcup \llbracket \lambda_2 \rrbracket (\bar{S}(X))) \\
&\succeq \llbracket \lambda_3 \rrbracket (\llbracket \lambda_1 \rrbracket (\bar{S}(X))) \sqcup \llbracket \lambda_3 \rrbracket (\llbracket \lambda_2 \rrbracket (\bar{S}(X))) \quad \text{From Property 17} \\
&= \llbracket \lambda_1 \circ \lambda_3 \rrbracket (\bar{S}(X)) \sqcup \llbracket \lambda_2 \circ \lambda_3 \rrbracket (\bar{S}(X))
\end{aligned}$$

Hence proved.

2. Let $\bar{S}(X)$ be a State Set Vector of length n . From LRE semantics, we get

$$\begin{aligned}
\llbracket (\lambda_1 + \lambda_2) \circ \lambda_3 \rrbracket (\bar{S}(X)) &= \llbracket \lambda_3 \rrbracket (\llbracket \lambda_1 + \lambda_2 \rrbracket (\bar{S})) \\
&= \llbracket \lambda_3 \rrbracket (\llbracket \lambda_1 \rrbracket (\bar{S}(X)) \sqcup \llbracket \lambda_2 \rrbracket (\bar{S}(X))) \\
&\preceq \llbracket \lambda_3 \rrbracket (\llbracket \lambda_1 \rrbracket (\bar{S}(X))) \sqcup \llbracket \lambda_3 \rrbracket (\llbracket \lambda_2 \rrbracket (\bar{S}(X))) \quad \text{From Property 18} \\
&= \llbracket \lambda_1 \circ \lambda_3 \rrbracket (\bar{S}(X)) \sqcup \llbracket \lambda_2 \circ \lambda_3 \rrbracket (\bar{S}(X))
\end{aligned}$$

Hence proved.

3. Follows from previous two cases.

Property 20. Let $\lambda_1, \lambda_2, \lambda_3$ and λ_4 be LREs over $\mathcal{T}_{Y,n}$. Then

1. If $\lambda_1 \preceq \lambda_2$ and $\lambda_3 \preceq \lambda_4$, then $(\lambda_1 + \lambda_3) \preceq (\lambda_2 + \lambda_4)$
2. If $\lambda_1 \preceq \lambda_2$ and $\lambda_3 \preceq \lambda_4$, then $(\lambda_1 \text{ op } \lambda_3) \preceq (\lambda_2 \text{ op } \lambda_4)$, where $\text{op} \in \{\circ, ;\}$ if either $\lambda_3 \in SC_0$ or $\lambda_4 \in SC_0$
3. $(\lambda_1 ; \lambda_2)^i \preceq (\lambda_1 ; \lambda_2)^{i+1}$, for all $i \geq 0$ if either $\lambda_1 \in SC_0$ or $\lambda_2 \in SC_0$
4. If $\lambda_1 \preceq \lambda_2$, then $(\lambda_1)^i \preceq (\lambda_2)^i$ for all $i \geq 0$, and $(*\lambda_1) \preceq (*\lambda_2)$ if either $\lambda_1 \in SC_0$ or $\lambda_2 \in SC_0$

Proof. 1. Let $\bar{S}(X)$ be a State Set Vector of length n . We know that

$$\begin{aligned} \llbracket \lambda_1 + \lambda_3 \rrbracket (\bar{S}(X)) &= \llbracket \lambda_1 \rrbracket (\bar{S}(X)) \sqcup \llbracket \lambda_3 \rrbracket (\bar{S}(X)) \quad \text{from LRE semantics} \\ &\preceq \llbracket \lambda_2 \rrbracket (\bar{S}(X)) \sqcup \llbracket \lambda_4 \rrbracket (\bar{S}(X)) \quad \text{Since } \lambda_1 \preceq \lambda_2 \text{ and } \lambda_3 \preceq \lambda_4 \\ &= \llbracket \lambda_2 + \lambda_4 \rrbracket (\bar{S}(X)) \end{aligned}$$

2. Let $\bar{S}(X)$ be a State Set Vector of length n . We know from LRE semantics that

$$\begin{aligned} \llbracket \lambda_1 \circ \lambda_3 \rrbracket (\bar{S}(X)) &= \llbracket \lambda_3 \rrbracket (\llbracket \lambda_1 \rrbracket (\bar{S}(X))) \\ &\preceq \llbracket \lambda_4 \rrbracket (\llbracket \lambda_1 \rrbracket (\bar{S}(X))) \quad \text{Since } \lambda_3 \preceq \lambda_4 \\ &\preceq \llbracket \lambda_4 \rrbracket (\llbracket \lambda_2 \rrbracket (\bar{S}(X))) \quad \text{Since } \lambda_4 \in SC_0 \text{ and } \lambda_1 \preceq \lambda_2 \\ &= \llbracket \lambda_2 \circ \lambda_4 \rrbracket (\bar{S}(X)) \end{aligned}$$

Similarly, it can also be shown that

$$\begin{aligned} \llbracket \lambda_3 \rrbracket (\llbracket \lambda_1 \rrbracket (\bar{S}(X))) &\preceq \llbracket \lambda_3 \rrbracket (\llbracket \lambda_2 \rrbracket (\bar{S}(X))) \quad \text{Since } \lambda_1 \preceq \lambda_2 \text{ and } \lambda_3 \in SC_0 \\ &\preceq \llbracket \lambda_4 \rrbracket (\llbracket \lambda_2 \rrbracket (\bar{S}(X))) \quad \text{Since } \lambda_3 \preceq \lambda_4 \\ &= \llbracket \lambda_2 \circ \lambda_4 \rrbracket (\bar{S}(X)) \end{aligned}$$

Also, $(\lambda_1 ; \lambda_3) \preceq (\lambda_2 ; \lambda_4)$ follows using a similar argument and case 1 if either $\lambda_3 \in SC_0$ or $\lambda_4 \in SC_0$.

3. We proceed by induction on i .

Basis: When $i = 0$, we have $\hat{\delta} \preceq (\lambda_1 ; \lambda_2)$.

Hypothesis: Assume for $0 \leq k \leq i$, $(\lambda_1 ; \lambda_2)^k \preceq (\lambda_1 ; \lambda_2)^{k+1}$.

Induction Step: We know by definition of semantics that $(\lambda_1 ; \lambda_2)^{i+1} = (\lambda_1 ; \lambda_2)^i \circ (\lambda_1 ; \lambda_2)$. Also, $(\lambda_1 ; \lambda_2)^i \preceq (\lambda_1 ; \lambda_2)^{i+1}$ from induction hypothesis and $(\lambda_1 ; \lambda_2) \preceq (\lambda_1 ; \lambda_2)$ from LRE semantics. Hence from Case 2, we have $(\lambda_1 ; \lambda_2)^{i+1} \preceq (\lambda_1 ; \lambda_2)^{i+2}$. Hence proved.

4. We proceed by induction on i .

Basis: When $i = 0$, we have $\lambda_1 \preceq \lambda_2$.

Hypothesis: Assume for $0 \leq k \leq i$, $(\lambda_1)^k \preceq (\lambda_2)^k$.

Induction Step: We know from LRE semantics that $(\lambda_1)^{i+1} = (\lambda_1)^i \circ \lambda_1$. Also, from induction hypothesis, we know that $(\lambda_1)^i \preceq (\lambda_2)^i$ and $\lambda_1 \preceq \lambda_2$. Hence the property follows from Case 2. Also $\bigsqcup_{i=0}^{\infty} (\lambda_1)^i \preceq \bigsqcup_{i=0}^{\infty} (\lambda_2)^i$. Hence proved.

Definition 30. Let λ be an LRE over $\mathcal{T}_{\Upsilon, n}$ and $\bar{S}(X)$ be a State Set Vector of length n . Then, $\llbracket *\lambda \rrbracket (\bar{S}(X)) = \lim_{i \rightarrow \infty} P_i(\bar{S}(X))$, where

$$P_j(\bar{S}(X)) = \begin{cases} \bar{S}(X) & \text{if } j = 0 \\ P_{j-1}(\bar{S}(X)) \sqcup \llbracket \lambda \rrbracket (P_{j-1}(\bar{S}(X))) & \text{otherwise} \end{cases}$$

Lemma 14. For a transition system $B = (I, X, Q, S_0, \mathbf{T})$, given a State Set Vector $\bar{S}(X)$ of length n and an LRE $\lambda \in SC_0$ over set $\mathcal{T}_{\Upsilon, n}$,

1. $P_i(\bar{S}(X)) \preceq P_{i+1}(\bar{S}(X))$
2. If $P_{i+1}(\bar{S}(X)) = P_i(\bar{S}(X))$, then $P_j(\bar{S}(X)) = P_i(\bar{S}(X))$, $\forall j \geq i$

$$3. \lim_{i \rightarrow \infty} P_i(\bar{S}(X)) = P_{n|Q|}(\bar{S}(X))$$

Proof. 1. We proceed by induction on i .

Basis: When $i = 0$, we have $P_0(\bar{S}(X)) = \bar{S}(X)$ by definition. Also $P_1(\bar{S}(X)) = P_0(\bar{S}(X)) \sqcup \llbracket \lambda \rrbracket (P_0(\bar{S}(X))) \succeq P_0(\bar{S}(X))$.

Induction Hypothesis: Assume that $P_r(\bar{S}(X)) \preceq P_{r+1}(\bar{S}(X))$, for $0 \leq r \leq i$.

Induction Step: We know from definition that

$$\begin{aligned} P_{i+2}(\bar{S}(X)) &= P_{i+1}(\bar{S}(X)) \sqcup \llbracket \lambda \rrbracket (P_{i+1}(\bar{S}(X))) \\ &\succeq P_i(\bar{S}(X)) \sqcup \llbracket \lambda \rrbracket (P_i(\bar{S}(X))) \quad \text{since } \lambda \in SC_0 \text{ and from induction hypothesis} \\ &= P_{i+1}(\bar{S}(X)) \quad \text{From definition} \end{aligned}$$

2. We know that

$$\begin{aligned} P_{i+1}(\bar{S}(X)) &= P_i(\bar{S}(X)) \\ \text{Also, } P_{i+2}(\bar{S}(X)) &= P_{i+1}(\bar{S}(X)) \sqcup \llbracket \lambda \rrbracket (P_{i+1}(\bar{S}(X))) \quad \text{From definition} \\ \text{Hence, } P_{i+1}(\bar{S}(X)) &= P_i(\bar{S}(X)) \sqcup \llbracket \lambda \rrbracket (P_i(\bar{S}(X))) \quad \text{Since } P_{i+1}(\bar{S}(X)) = P_i(\bar{S}(X)) \\ &= P_{i+1}(\bar{S}(X)) \quad \text{from definition} \\ &= P_i(\bar{S}(X)) \end{aligned}$$

3. It is important to note that each P_i is evaluated on a State Set Vector $\bar{S}(X)$ of n components. Each $S_i(X)$ belongs to a lattice of height $|Q|$ ordered by subset inclusion. So the State Set Vector $\bar{S}(X)$ can be viewed as a composite lattice of height $n|Q|$. Hence the computation would terminate in $n|Q|$ steps as from Case 1, we know that every step is only going to possibly add more sets of states, and from Case 2, we know that if at any step i , the current State Set Vector does not change, then fixpoint has been reached.

Lemma 15. For a transition system $B = (I, X, Q, S_0, \mathbf{T})$ given a State Set Vector $\bar{S}(X)$ of length n and an LRE $\lambda \in SC_0$ over $\mathcal{T}_{T,n}$, $\llbracket * \lambda \rrbracket (\bar{S}(X)) = P_{n|Q|}(\bar{S}(X))$.

Proof. Follows from Lemma 14 and definition of LRE semantics.

Property 21. Given a State Set Vector $\bar{S}(X)$ of length n and using the definition from Definition 30,

1. $P_{n|Q|}(\bar{S}(X)) \succeq P_i(\bar{S}(X))$, $\forall i$
2. $P_i(P_j(\bar{S}(X))) = P_{i+j}(\bar{S}(X))$, $\forall i, \forall j$
3. $\forall i, P_{n|Q|}(P_i(\bar{S}(X))) = P_{n|Q|}(\bar{S}(X))$

Proof. 1. From definition, we know that $\forall i \leq n|Q|$, we have $P_{n|Q|}(\bar{S}(X)) \succeq P_i(\bar{S}(X))$. Also, from Lemma 2, we have $\forall i > n|Q|, P_{n|Q|}(\bar{S}(X)) = P_i(\bar{S}(X))$. Hence proved.

2. We proceed by induction on i .

Basis: When $i = 0$, we have $P_0(P_j(\bar{S}(X))) = P_j(\bar{S}(X))$ from definition.

Induction hypothesis: Assume for $1 \leq k \leq i, P_k(P_j(\bar{S}(X))) = P_{k+j}(\bar{S}(X))$.

Induction step: We have by definition

$$\begin{aligned}
P_{i+1}(P_j(\overline{S}(X))) &= P_i(P_j(\overline{S}(X))) \sqcup [[\lambda]](P_i(P_j(\overline{S}(X)))) \\
&= P_{i+j}(\overline{S}(X)) \sqcup [[\lambda]](P_{i+j}(\overline{S}(X))) \quad \text{From induction hypothesis} \\
&= P_{i+j+1}(\overline{S}(X)) \quad \text{From definition}
\end{aligned}$$

3. We know that

$$\begin{aligned}
P_{n|Q}(P_i(\overline{S}(X))) &= P_{n|Q+i}(\overline{S}(X)) \quad \text{From case 2} \\
&= P_{n|Q}(\overline{S}(X)) \quad \text{From case 1}
\end{aligned}$$

Property 22. For a transition system $B = (I, X, Q, S_0, \mathbf{T})$, let $\lambda, \lambda_1, \lambda_2 \in SC_0$ be LREs over $\mathcal{T}_{T,n}$ and $\overline{S}(X)$ be a State Set Vector of length n . Then

1. $*(\ast\lambda) = \ast\lambda$
2. $(\ast\lambda)^i = \ast\lambda = (\ast\lambda)^{i-1}, \forall i > 1$
3. If $\lambda_1 \preceq \lambda_2$, then $\ast\lambda_1 \preceq \ast\lambda_2$
4. $\ast(\lambda_1; \lambda_2) = \ast(\ast\lambda_1; \lambda_2) = \ast(\ast\lambda_1; \ast\lambda_2) = \ast(\lambda_1; \ast\lambda_2)$

Proof. 1. We know from Lemma 15 and Definition 30 that $[[\ast\lambda]](\overline{S}(X)) = P_{n|Q}(\overline{S}(X))$, where

$$P_j(\overline{S}(X)) = \begin{cases} \overline{S}(X) & \text{if } j = 0 \\ P_{j-1}(\overline{S}(X)) \sqcup [[\lambda]](P_{j-1}(\overline{S}(X))) & \text{otherwise} \end{cases}$$

Let $\ast\lambda = \lambda'$, and $[[\ast\lambda']](\overline{S}(X)) = P'_{n|Q}(\overline{S}(X))$, where

$$P'_j(\overline{S}(X)) = \begin{cases} \overline{S}(X) & \text{if } j = 0 \\ P'_{j-1}(\overline{S}(X)) \sqcup [[\lambda']](P'_{j-1}(\overline{S}(X))) & \text{otherwise} \end{cases}$$

Hence $\forall i > 1$, we have

$$\begin{aligned}
P'_i(\overline{S}(X)) &= P'_{i-1}(\overline{S}(X)) \sqcup [[\ast\lambda]](P'_{i-1}(\overline{S}(X))) \\
&= P'_{i-1}(\overline{S}(X)) \sqcup P_{n|Q}(P'_{i-1}(\overline{S}(X))) \\
&= P'_{i-1}(\overline{S}(X)) \sqcup P_{n|Q}(\overline{S}(X)) \quad \text{From Property 21 case 3}
\end{aligned}$$

We have $P'_0(\overline{S}(X)) = \overline{S}(X) = P_0(\overline{S}(X))$. When $i = 1$, we have

$$\begin{aligned}
P'_1(\overline{S}(X)) &= P'_0(\overline{S}(X)) \sqcup P_{n|Q}(\overline{S}(X)) \\
&= P_0(\overline{S}(X)) \sqcup P_{n|Q}(\overline{S}(X)) \\
&= P_{n|Q}(\overline{S}(X)) \quad \text{From definition} \\
&= [[\ast\lambda]](\overline{S}(X))
\end{aligned}$$

Assume $0 \leq i \leq j$, $P'_j(\overline{S}(X)) = P_{n|Q|}(\overline{S}(X))$.

So,

$$\begin{aligned} P'_{j+1}(\overline{S}(X)) &= P'_j(P'_1(\overline{S}(X))) \quad \text{From Property 21 case 2} \\ &= P'_j(P_{n|Q|}(\overline{S}(X))) \\ &= P_{n|Q|}(\overline{S}(X)) \quad \text{From Property 21 case 3} \end{aligned}$$

Hence proved.

2. We proceed by induction on i .

Basis: When $i = 2$, we have

$$\begin{aligned} \llbracket * \lambda \circ * \lambda \rrbracket (\overline{S}(X)) &= \llbracket * \lambda \rrbracket (\llbracket * \lambda \rrbracket (\overline{S}(X))) \\ &= P_{n|Q|}(P_{n|Q|}(\overline{S}(X))) \\ &= P_{n|Q|}(\overline{S}(X)) \\ &= * \lambda \end{aligned}$$

Induction Hypothesis: Assume for $2 \leq k \leq i$, $* \lambda = (* \lambda)^k$.

Induction Step: We know that

$$\begin{aligned} (* \lambda)^{k+1} &= (* \lambda)^k \circ * \lambda \quad \text{From definition} \\ &= * \lambda \circ * \lambda \quad \text{From induction hypothesis} \\ &= * \lambda \quad \text{From basis} \end{aligned}$$

Hence proved.

3. Let $\llbracket * \lambda_1 \rrbracket (\overline{S}(X)) = P_{n|Q|}(\overline{S}(X))$, and $\llbracket * \lambda_2 \rrbracket (\overline{S}(X)) = P'_{n|Q|}(\overline{S}(X))$. We proceed by induction on i .

Basis: When $i = 0$, we have $P_0(\overline{S}(X)) \preceq P'_0(\overline{S}(X))$.

Induction Hypothesis: Assume $0 \leq j < i$, $P_j(\overline{S}(X)) \preceq P'_j(\overline{S}(X))$.

Induction Step: We know that

$$P_{i+1}(\overline{S}(X)) = P_i(\overline{S}(X)) \sqcup \llbracket \lambda_1 \rrbracket (P_i(\overline{S}(X))) \quad \text{From definition}$$

$$P'_{i+1}(\overline{S}(X)) = P'_i(\overline{S}(X)) \sqcup \llbracket \lambda_1 \rrbracket (P'_i(\overline{S}(X))) \quad \text{From definition}$$

$$\text{Also, } P_i(\overline{S}(X)) \preceq P'_i(\overline{S}(X)) \quad \text{From induction hypothesis}$$

$$\text{Hence, } P_{i+1}(\overline{S}(X)) \preceq P'_{i+1}(\overline{S}(X)) \quad \text{as } \lambda_1 \preceq \lambda_2$$

$$\text{It follows that } P_{n|Q|}(\overline{S}(X)) \preceq P'_{n|Q|}(\overline{S}(X))$$

4. By definition of semantics, we know that $\lambda_2 \preceq * \lambda_1$. Hence from Property 20 case 2, we have $\lambda_1 ; \lambda_2 \preceq \lambda_1 ; * \lambda_2$. Also, from cases 3 and 1, we have $* (\lambda_1 ; \lambda_2) \preceq * (\lambda_1 ; * \lambda_2)$.

We now show that $* (\lambda_1 ; \lambda_2) \succeq (\lambda_1 ; * \lambda_2)$. $\llbracket \lambda_1 ; * \lambda_2 \rrbracket (\overline{S}(X)) = \llbracket (\lambda_1 + \hat{\delta}) \circ P_{n|Q|} \rrbracket (\overline{S}(X)) = P_{n|Q|}(\overline{S}(X)) \sqcup \llbracket \lambda_2 \rrbracket (\overline{S}(X))$, and let $\llbracket * (\lambda_1 ; \lambda_2) \rrbracket (\overline{S}(X)) = P'_{n|Q|}(\overline{S}(X))$. We have $P_0(\overline{S}(X)) = \overline{S}(X) = P'_0(\overline{S}(X))$. $\forall i \geq 1$, $P_i(\overline{S}(X)) = P'_{i-1}(\overline{S}(X)) \sqcup \llbracket \lambda_1 ; \lambda_2 \rrbracket (P'_{i-1}(\overline{S}(X)))$. We proceed by induction on i .

Basis: When $i = 0$, we have $P_0(\overline{S}(X)) \preceq P'_0(\overline{S}(X))$. When $i = 1$, we have $P_1(\overline{S}(X)) = P_0(\overline{S}(X)) \sqcup \llbracket \lambda_1 \rrbracket (\overline{S}(X)) = \overline{S}(X) \sqcup \llbracket \lambda_1 \rrbracket (\overline{S}(X))$. Also, $P'_1(\overline{S}(X)) = \overline{S}(X) \sqcup \llbracket \lambda_1 ; \lambda_2 \rrbracket (\overline{S}(X)) \succeq \overline{S}(X) \sqcup \llbracket \lambda_2 \rrbracket (\overline{S}(X))$ since $\lambda_1 ; \lambda_2 \succeq$

$\hat{\delta} + \lambda_1 + (\hat{\delta} + \lambda_1) \circ \lambda_2$. Hence $P_1(\overline{S}(X)) \preceq P'_1(\overline{S}(X))$.

Hypothesis: Assume that $P_j(\overline{S}(X) \sqcup \llbracket \lambda_1 \rrbracket (\overline{S}(X))) \preceq P'_{j+1}(\overline{S}(X))$ for $1 \leq j \leq i$.

Induction Step: $P_{i+1}(\overline{S}(X) \sqcup \llbracket \lambda_1 \rrbracket (\overline{S}(X))) = P_i(\overline{S}(X) \sqcup \llbracket \lambda_1 \rrbracket (\overline{S}(X))) \sqcup \llbracket \lambda_2 \rrbracket (\overline{S}(X) \sqcup \llbracket \lambda_1 \rrbracket (\overline{S}(X))) \preceq P'_{i+1}(\overline{S}(X) \sqcup \llbracket \lambda_1 \rrbracket (\overline{S}(X))) \sqcup \llbracket \lambda_2 \rrbracket (P'_{i+1}(\overline{S}(X) \sqcup \llbracket \lambda_1 \rrbracket (\overline{S}(X))))$ from induction hypothesis. It follows that $P_{i+1}(\overline{S}(X) \sqcup \llbracket \lambda_1 \rrbracket (\overline{S}(X))) \preceq P'_{i+1}(\overline{S}(X))$ using induction hypothesis.

Hence we have shown that $(\lambda_1 ; * \lambda_2) \preceq *(\lambda_1 ; \lambda_2)$. It follows from cases 3 and 1 that $*(\lambda_1 ; * \lambda_2) \preceq *(\lambda_1 ; \lambda_2)$. Hence, we now get $*(\lambda_1 ; * \lambda_2) = *(\lambda_1 ; \lambda_2)$. It can be shown similarly that $*(* \lambda_1 ; \lambda_2) = *(\lambda_1 ; \lambda_2)$. To prove $*(* \lambda_1 ; \lambda_2) = *(* \lambda_1 ; * \lambda_2)$, we can set λ_1 as $* \lambda_1$ in the previous result.

Property 23. Let $\lambda_1, \lambda_2 \in SC_0$ be LREs over $\mathcal{T}_{\Upsilon, n}$, then $*(\lambda_1 ; \lambda_2) = *(\lambda_1 + \lambda_2)$.

Proof. We prove this in two parts:

1. We first show that $*(\lambda_1 + \lambda_2) \preceq *(\lambda_1 ; \lambda_2)$.

$$(\lambda_1 + \lambda_2) \preceq (\lambda_1 ; \lambda_2) \quad \text{From LRE semantics}$$

$$\text{Hence, } *(\lambda_1 + \lambda_2) \preceq *(\lambda_1 ; \lambda_2) \quad \text{From Property 22 case 3}$$

2. Now we show that $*(\lambda_1 ; \lambda_2) \preceq *(\lambda_1 + \lambda_2)$.

$$\begin{aligned} \lambda_1 ; \lambda_2 &= (\lambda_1 + \hat{\delta}) \circ (\lambda_2 + \hat{\delta}) \quad \text{From definition} \\ &= \hat{\delta} + \lambda_1 + (\lambda_1 + \hat{\delta}) \circ \lambda_2 \end{aligned}$$

Also, let $\lambda = \lambda_1 + \lambda_2$. Hence by definition, $* \lambda = P_{n|Q|}(\overline{S}(X))$, where

$$P_j(\overline{S}(X)) = \begin{cases} \overline{S}(X) & \text{if } j = 0 \\ P_{j-1}(\overline{S}(X)) \sqcup \llbracket \lambda \rrbracket (P_{j-1}(\overline{S}(X))) & \text{otherwise} \end{cases}$$

Also from Property 21 case 1, we know that for any $i \leq n|Q|$, $P_i(\overline{S}(X)) \preceq P_{n|Q|}(\overline{S}(X))$. We will now prove that $P_2(\overline{S}(X)) \succeq \lambda_1 ; \lambda_2$. It follows from definition that

$$\begin{aligned} P_2(\overline{S}(X)) &= \overline{S}(X) \sqcup \llbracket \lambda_1 \rrbracket (\overline{S}(X)) \sqcup \llbracket \lambda_2 \rrbracket (\overline{S}(X)) \sqcup \llbracket \lambda_1 + \lambda_2 \rrbracket (\overline{S}(X) \sqcup \llbracket \lambda_1 \rrbracket (\overline{S}(X))) \sqcup \llbracket \lambda_2 \rrbracket (\overline{S}(X)) \\ &\succeq \overline{S}(X) \sqcup \llbracket \lambda_1 \rrbracket (\overline{S}(X)) \sqcup \llbracket \lambda_2 \rrbracket (\overline{S}(X)) \sqcup \llbracket (\lambda_1 + \lambda_2)^2 \rrbracket (\overline{S}(X)) \quad \text{From Property 19 case 1} \end{aligned}$$

$$\text{Also, } (\lambda_1 + \lambda_2)^2 \succeq (\lambda_1)^2 + (\lambda_1 \circ \lambda_2) + (\lambda_2 \circ \lambda_1) + (\lambda_2)^2 \quad \text{From definition and Property 19 case 1}$$

$$\text{But, } (\lambda_2)^2 \succeq \lambda_2 \quad \text{From Property AA}$$

$$\text{Hence, } (\lambda_1 + \lambda_2)^2 \succeq (\lambda_1 + \hat{\delta}) \circ \lambda_2$$

$$\text{Hence, } \lambda_1 ; \lambda_2 \preceq *(\lambda_1 + \lambda_2)$$

$$\text{Also, } *(\lambda_1 ; \lambda_2) \preceq *(*(\lambda_1 + \lambda_2)) \quad \text{From Property XX case YY}$$

$$\text{Hence proved } *(\lambda_1 ; \lambda_2) \preceq *(\lambda_1 + \lambda_2)$$

Hence proved $*(\lambda_1 + \lambda_2) = *(\lambda_1 ; \lambda_2)$.

Theorem 13. Let $\{\lambda_1, \dots, \lambda_p\}$, $p \geq 1$ be a finite set of LREs over $\mathcal{T}_{\Upsilon, n}$ such that $\forall i, \lambda_i \in SC_0$. Then

$$*(\lambda_1 ; \dots ; \lambda_p) = *(\lambda_1 + \dots + \lambda_p)$$

Proof. We proceed by induction on p .

Basis: When $p = 1$, the result follows trivially.

Induction Hypothesis: Assume that the result holds true for all p in 1 through m .

Induction Step: Consider a set of $m + 1$ LREs over $\mathcal{T}_{\Gamma,n}$. Let $\lambda' = \lambda_2 + \dots + \lambda_{m+1}$. We know from Property 23 that $\ast(\lambda_1 + \lambda') = \ast(\lambda_1; \lambda')$. Also from Property 22 case 4, we have

$$\begin{aligned} \ast(\lambda_1; \lambda') &= \ast(\lambda_1; \ast\lambda') \\ &= \ast(\lambda_1; \ast\lambda'') \quad \text{From induction hypothesis where } \lambda'' = \lambda_2; \dots; \lambda_{m+1} \end{aligned}$$

$$\text{Hence, } \ast(\lambda_1 + \lambda') = \ast(\lambda_1; \ast\lambda'')$$

$$\text{Also, } \ast(\lambda_1 + \lambda') = \ast(\lambda_1; \lambda'') \quad \text{From Property 22 case 4}$$

Hence, $\ast(\lambda_1; \dots; \lambda_{m+1}) = \ast(\lambda_1 + \dots + \lambda_{m+1})$ Since $+$ and $;$ are associative (Property 4 & Property 10 resp.)

Hence proved.

Definition 31. *Union Norm of a State Set Vector \bar{S} :* Given a State Set Vector \bar{S} of length n , its union norm denoted $\|\bar{S}\|_{\cup}$ is defined as $\|\bar{S}\|_{\cup} = \bigcup_{i=1}^n S_i$.

Definition 32. *Intersection Norm of a State Set Vector \bar{S} :* Given a State Set Vector \bar{S} of length n , its intersection norm denoted $\|\bar{S}\|_{\cap}$ is defined as $\|\bar{S}\|_{\cap} = \bigcap_{i=1}^n S_i$.

Lemma 16. *Let \bar{S} and \bar{R} be State Set Vectors of length n such that $\bar{S} \preceq \bar{R}$. Then,*

1. $\|\bar{S}\|_{\cap} \subseteq \|\bar{R}\|_{\cap}$
2. $\|\bar{S}\|_{\cup} \subseteq \|\bar{R}\|_{\cup}$

Proof. 1. We know that $\bar{S} \preceq \bar{R}$. Hence by definition of \preceq we have $\forall i, S_i \subseteq R_i$. As \cup preserves order, we have $\bigcup_{i=1}^n S_i \subseteq \bigcup_{i=1}^n R_i$. Hence, $\|\bar{S}\|_{\cup} \subseteq \|\bar{R}\|_{\cup}$ from definition of Union Norm.

2. We know that $\bar{S} \preceq \bar{R}$. Hence by definition of \preceq we have $\forall i, S_i \subseteq R_i$. As \cap preserves order, we have $\bigcap_{i=1}^n S_i \subseteq \bigcap_{i=1}^n R_i$. Hence, $\|\bar{S}\|_{\cap} \subseteq \|\bar{R}\|_{\cap}$ from definition of Intersection Norm.

Definition 33. *Upper bound vector of a State Set Vector \bar{S} :* Given a State Set Vector \bar{S} of length n , its Upper bound vector denoted \bar{S}^U is defined as $\forall i, S_i^U = \|\bar{S}\|_{\cup}$.

Definition 34. *Lower bound vector of a State Set Vector \bar{S} :* Given a State Set Vector \bar{S} of length n , its Lower bound vector denoted \bar{S}^L is defined as $\forall i, S_i^L = \|\bar{S}\|_{\cap}$.

Lemma 17. *For any State Set Vector \bar{S} of length n , $\bar{S}^U \succeq \bar{S} \succeq \bar{S}^L$.*

Proof. Follows from the definition of \bar{S}^U and \bar{S}^L .

Lemma 18. *Let $\lambda \in SC_0$ be an LRE over $\mathcal{T}_{\Gamma,n}$ and \bar{S} be a State Set Vector of length n . Then*

$$\llbracket \lambda \rrbracket (\bar{S}^U) \succeq \llbracket \lambda \rrbracket (\bar{S}) \succeq \llbracket \lambda \rrbracket (\bar{S}^L)$$

Proof. Follows from the fact that $\llbracket \lambda \rrbracket$ is monotonically increasing (since $\lambda \in SC_0$) and Lemma 17.

Lemma 19. *Let $\lambda \in SC_1$ be an LRE over $\mathcal{T}_{\Gamma,n}$ and \bar{S} be a State Set Vector of length n . Then*

$$\llbracket \lambda \rrbracket (\bar{S}^U) \preceq \llbracket \lambda \rrbracket (\bar{S}) \preceq \llbracket \lambda \rrbracket (\bar{S}^L)$$

Proof. Follows from the fact that $[[\lambda]]$ is monotonically decreasing (since $\lambda \in SC_1$) and Lemma 17.

Theorem 14. Let $\lambda \in SC_0$ be an LRE over $\mathcal{T}_{r,n}$ and \bar{S} be a State Set Vector of length n . Then

$$\| [[\lambda]](\bar{S}^U) \|_{op} \supseteq \| [[\lambda]](\bar{S}) \|_{op} \supseteq \| [[\lambda]](\bar{S}^L) \|_{op}$$

where $op \in \{\cup, \cap\}$.

Proof.

$$\bar{S}^U \succeq \bar{S} \succeq \bar{S}^L \quad \text{From Lemma 17}$$

Hence, $[[\lambda]](\bar{S}^U) \succeq [[\lambda]](\bar{S}) \succeq [[\lambda]](\bar{S}^L)$ Since $\lambda \in SC_0$ is monotonically increasing

Hence, $\| [[\lambda]](\bar{S}^U) \|_{op} \supseteq \| [[\lambda]](\bar{S}) \|_{op} \supseteq \| [[\lambda]](\bar{S}^L) \|_{op}$ From Lemma 16

Hence proved.

Theorem 15. Let $\lambda \in SC_1$ be an LRE over $\mathcal{T}_{r,n}$ and \bar{S} be a State Set Vector of length n . Then

$$\| [[\lambda]](\bar{S}^U) \|_{op} \subseteq \| [[\lambda]](\bar{S}) \|_{op} \subseteq \| [[\lambda]](\bar{S}^L) \|_{op}$$

where $op \in \{\cup, \cap\}$.

Proof.

$$\bar{S}^U \succeq \bar{S} \succeq \bar{S}^L \quad \text{From Lemma 17}$$

Hence, $[[\lambda]](\bar{S}^U) \preceq [[\lambda]](\bar{S}) \preceq [[\lambda]](\bar{S}^L)$ Since $\lambda \in SC_1$ is monotonically increasing

Hence, $\| [[\lambda]](\bar{S}^U) \|_{op} \subseteq \| [[\lambda]](\bar{S}) \|_{op} \subseteq \| [[\lambda]](\bar{S}^L) \|_{op}$ From Lemma 16

Lemma 20. Let \bar{S} and \bar{R} be State Set Vectors of length n . Then,

$$\| \bar{S} \sqcup \bar{R} \|_{\cup} = \| \bar{S} \|_{\cup} \cup \| \bar{R} \|_{\cup}$$

Proof. Let $\bar{P} = \bar{S} \sqcup \bar{R}$, then from definition $\forall i, P_i = S_i \cup R_i$.

$$\begin{aligned} \| \bar{P} \|_{\cup} &= \bigcup_{i=1}^n (S_i \cup R_i) \quad \text{From definition} \\ &= \bigcup_{i=1}^n S_i \cup \bigcup_{i=1}^n R_i \\ &= \| \bar{S} \|_{\cup} \cup \| \bar{R} \|_{\cup} \quad \text{From definition} \end{aligned}$$

Hence proved.

Lemma 21. Let \bar{S} and \bar{R} be State Set Vectors of length n . Then,

$$\begin{aligned} \| \bar{S} \sqcup \bar{R} \|_{\cap} &\supseteq \| \bar{S} \|_{\cap} \cup \| \bar{R} \|_{\cap} \quad \text{and,} \\ \| \bar{S} \sqcup \bar{R} \|_{\cap} &\subseteq \| \bar{S} \|_{\cup} \cup \| \bar{R} \|_{\cup} \end{aligned}$$

Proof. Let $\bar{P} = \bar{S} \sqcup \bar{R}$, then $\forall i, P_i = S_i \cup R_i$.

$$\begin{aligned} \|\bar{P}\|_{\cap} &= \bigcap_{i=1}^n (S_i \cup R_i) \quad \text{From definition} \\ &\supseteq \bigcap_{i=1}^n S_i \cup \bigcap_{i=1}^n R_i \\ &= \|\bar{S}\|_{\cap} \cup \|\bar{R}\|_{\cap} \end{aligned}$$

Similarly,

$$\begin{aligned} \|\bar{P}\|_{\cup} &= \bigcap_{i=1}^n (S_i \cup R_i) \quad \text{From definition} \\ &\subseteq \bigcup_{i=1}^n (S_i \cup R_i) \\ &= \bigcup_{i=1}^n S_i \cup \bigcup_{i=1}^n R_i \\ &= \|\bar{S}\|_{\cup} \cup \|\bar{R}\|_{\cup} \end{aligned}$$

Hence proved.

Lemma 22. Let \bar{R} be a State Set Vector of length n . Then

$$\neg(\|\bar{R}\|_{\cap}) = \|\neg\bar{R}\|_{\cup}.$$

Proof.

$$\begin{aligned} \neg(\|\bar{R}\|_{\cap}) &= \neg\left(\bigcap_{i=1}^n R_i\right) \quad \text{From definition} \\ &= \bigcup_{i=1}^n \neg R_i \quad \text{From definition of } \neg \\ &= \|\neg\bar{R}\|_{\cup} \quad \text{From definition of Union norm} \end{aligned}$$

Hence proved.

Corollary 4. Let \bar{R} be a State Set Vector of length n . Then

$$\neg(\|\bar{R}\|_{\cup}) = \|\neg\bar{R}\|_{\cap}.$$

Proof. Follows from definition and Lemma 22.

14 Interpreting an LRE from an RE

Given an RE σ over \mathcal{C}_Y and a State Set Vector $\bar{S}(X)$ of length n , if we want to evaluate $\llbracket \sigma \rrbracket(S_j(X))$ and store the result in $S_i(X)$, i.e. perform $S_i(X) = \llbracket \sigma \rrbracket(S_j(X))$, it can be performed by the translation $\beta(i, \sigma, j)$ inductively defined as

$$\beta(i, \sigma, j) = \begin{cases} i : (\sigma)_j & \text{if } \sigma = \Upsilon_1 \mid \Upsilon_2 \mid \dots \mid \Upsilon_k \mid \delta \mid \theta \\ \beta(i, \sigma_1, j) + \beta(i, \sigma_2, j) & \text{if } \sigma = \sigma_1 + \sigma_2 \\ \beta(i, \sigma_1, j) \circ \beta(i, \sigma_2, i) & \text{if } \sigma = \sigma_1 \circ \sigma_2 \\ (\beta(i, \sigma_1, j) + i : (\delta)_j) \circ (\beta(i, \sigma_2, i) + i : (\delta)_i) & \text{if } \sigma = \sigma_1 ; \sigma_2 \\ \beta(i, \sigma_1, j) & \text{if } \sigma = (\sigma_1) \\ -\beta(i, \sigma_1, j) & \text{if } \sigma = \neg\sigma_1 \\ i : (\delta)_j \circ *(\beta(i, \sigma_1, i)) & \text{if } \sigma = *\sigma_1 \end{cases}$$

Lemma 23. Let $\bar{S}(X)$ be a State Set Vector of length n . Given an RE σ over C_{Υ} , there exists an LRE $\lambda = \beta(i, \sigma, j)$ over $\mathcal{T}_{\Upsilon, n}$, such that for any i, j , $\llbracket \sigma \rrbracket (S_j(X)) = (\llbracket \lambda \rrbracket (\bar{S}(X)))_i$.

Proof. We prove by induction on the structure of σ .

Basis:

1. Let σ be δ .

We know from the construction that

$$\begin{aligned} \lambda &= \beta(i, \delta, j) \\ &= i : (\delta)_j \end{aligned}$$

$$\begin{aligned} \text{Also, } (\llbracket \lambda \rrbracket (\bar{S}(X)))_i &= (\llbracket i : (\delta)_j \rrbracket (\bar{S}(X)))_i \\ &= S_j(X) \quad \text{From the definition of LRE semantics} \\ &= \llbracket \delta \rrbracket (S_j(X)) \quad \text{From the definition of RE semantics} \end{aligned}$$

Hence proved.

2. Let σ be θ .

We know from the construction that

$$\begin{aligned} \lambda &= \beta(i, \theta, j) \\ &= i : (\theta)_i \end{aligned}$$

$$\begin{aligned} \text{Also, } (\llbracket \lambda \rrbracket (\bar{S}(X)))_i &= (\llbracket i : (\theta)_i \rrbracket (\bar{S}(X)))_i \\ &= 0 \quad \text{From the definition of LRE semantics} \\ &= \llbracket \theta \rrbracket (S_j(X)) \quad \text{From the definition of RE semantics} \end{aligned}$$

Hence proved.

3. Let σ be Υ_p .

We know from the construction that

$$\begin{aligned}\lambda &= \beta(i, \Upsilon_p, j) \\ &= i : (\Upsilon_p)_j\end{aligned}$$

$$\begin{aligned}(\llbracket \lambda \rrbracket (\bar{S}(X)))_i &= (\llbracket i : (\Upsilon_p)_j \rrbracket (\bar{S}(X)))_i \\ &= (S_1(X), \dots, S_{i-1}(X), \text{Img}_{TR}(S_j(X), \Upsilon_p), S_{i+1}(X), \dots, S_n(X))_i \\ &\quad \text{From definition of LRE semantics} \\ &= \text{Img}_{TR}(S_j(X), \Upsilon_p) \\ &= \llbracket \Upsilon_p \rrbracket (S_j(X)) \quad \text{From the definition of RE semantics}\end{aligned}$$

Hence proved.

Hypothesis: Let σ_1 and σ_2 be two REs over \mathcal{C}_Υ and a vector of sets of states \bar{S} of length n , such that there exist LREs $\lambda_1 = \beta(i, \sigma_1, j)$ and $\lambda_2 = \beta(i, \sigma_2, j)$ over $\mathcal{T}_{\Upsilon, n}$ such that

$$\begin{aligned}\text{for any } i, j \llbracket \sigma_1 \rrbracket (S_j(X)) &= (\llbracket \lambda_1 \rrbracket (\bar{S}(X)))_i \\ \text{And, } \llbracket \sigma_2 \rrbracket (S_j(X)) &= (\llbracket \lambda_2 \rrbracket (\bar{S}(X)))_i\end{aligned}$$

Induction Step: We consider the following cases:

1. Let σ be $\sigma_1 + \sigma_2$.

We know from the construction that

$$\begin{aligned}\lambda &= \beta(i, \sigma_1 + \sigma_2, j) \\ &= \beta(i, \sigma_1, j) + \beta(i, \sigma_2, j)\end{aligned}$$

$$\begin{aligned}(\llbracket \lambda \rrbracket (\bar{S}(X)))_i &= (\llbracket \beta(i, \sigma_1, j) + \beta(i, \sigma_2, j) \rrbracket (\bar{S}(X)))_i \\ &= (\llbracket \beta(i, \sigma_1, j) \rrbracket (\bar{S}(X)))_i \vee (\llbracket \beta(i, \sigma_2, j) \rrbracket (\bar{S}(X)))_i \\ &\quad \text{From the definition of LRE semantics} \\ &= \llbracket \sigma_1 \rrbracket (S_j(X)) \vee \llbracket \sigma_2 \rrbracket (S_j(X)) \quad \text{From induction hypothesis} \\ &= \llbracket \sigma_1 + \sigma_2 \rrbracket (S_j(X)) \quad \text{From the definition of RE semantics}\end{aligned}$$

Hence proved.

2. Let σ be $\sigma_1 \circ \sigma_2$.

We know from the construction that

$$\begin{aligned}\lambda &= \beta(i, \sigma_1 \circ \sigma_2, j) \\ &= \beta(i, \sigma_1, j) \circ \beta(i, \sigma_2, i)\end{aligned}$$

$$\begin{aligned}(\llbracket \lambda \rrbracket (\overline{S}(X)))_i &= (\llbracket \beta(i, \sigma_1, j) \circ \beta(i, \sigma_2, i) \rrbracket (\overline{S}(X)))_i \\ &= (\llbracket \beta(i, \sigma_2, i) \rrbracket (\llbracket \beta(i, \sigma_1, j) \rrbracket (\overline{S}(X))))_i \\ &= (\llbracket \beta(i, \sigma_2, i) \rrbracket (S_1(X), \dots, S_{i-1}(X), \llbracket \sigma_1 \rrbracket (S_j(X)), S_{i+1}(X), \dots, S_n(X)))_i \\ &\quad \text{From the definition of LRE semantics and induction hypothesis} \\ &= \llbracket \sigma_2 \rrbracket (\llbracket \sigma_1 \rrbracket (S_j(X)) \quad \text{Since } R(\beta(i, \sigma_2, i)) = \{i\}) \\ &= \llbracket \sigma_1 \circ \sigma_2 \rrbracket (S_j(X)) \quad \text{From the definition of RE semantics}\end{aligned}$$

Hence proved.

3. Let σ be $\sigma_1 ; \sigma_2$.

We know from the construction that

$$\begin{aligned}\lambda &= \beta(i, \sigma_1 ; \sigma_2, j) \\ &= (\beta(i, \sigma_1, j) + i : (\delta)_i) \circ (\beta(i, \sigma_2, i) + i : (\delta)_i)\end{aligned}$$

$$\begin{aligned}(\llbracket \lambda \rrbracket (\overline{S}(X)))_i &= (\llbracket (\beta(i, \sigma_1, j) + i : (\delta)_i) \circ (\beta(i, \sigma_2, i) + i : (\delta)_i) \rrbracket (\overline{S}(X)))_i \\ &= (\llbracket \beta(i, \sigma_2, i) + i : (\delta)_i \rrbracket (\llbracket \beta(i, \sigma_1, j) + i : (\delta)_i \rrbracket (\overline{S}(X))))_i \quad \text{From LRE semantics} \\ &= (\llbracket \beta(i, \sigma_2, i) + i : (\delta)_i \rrbracket ((S_1(X), \dots, \llbracket \sigma_1 \rrbracket (S_j(X)), S_{i+1}(X), \dots, S_n(X)) \sqcup (S_1(X), \dots, S_j(X), S_{i+1}(X), \dots, S_n(X))))_i \\ &\quad \text{From LRE semantics and induction hypothesis} \\ &= (\llbracket \beta(i, \sigma_2, i) + i : (\delta)_i \rrbracket (S_1(X), \dots, S_j(X) \vee \llbracket \sigma_1 \rrbracket (S_j(X)), S_{i+1}(X), \dots, S_n(X)))_i \\ &= (\llbracket \beta(i, \sigma_2, i) \rrbracket ((S_1(X), \dots, S_j(X) \vee \llbracket \sigma_1 \rrbracket (S_j(X)), S_{i+1}(X), \dots, S_n(X)) \sqcup \\ &\quad \llbracket i : (\delta)_i \rrbracket (S_1(X), \dots, S_j(X) \vee \llbracket \sigma_1 \rrbracket (S_j(X)), S_{i+1}(X), \dots, S_n(X))))_i \quad \text{From LRE semantics} \\ &= S_j(X) \vee \llbracket \sigma_1 \rrbracket (S_j(X)) \vee \llbracket \sigma_2 \rrbracket (S_j(X)) \vee \llbracket \sigma_2 \rrbracket (\llbracket \sigma_1 \rrbracket (S_j(X))) \quad \text{From LRE semantics, induction hypothesis and RE semantics} \\ &= \llbracket \delta + \sigma_1 + \sigma_2 + \sigma_1 \circ \sigma_2 \rrbracket (S_j(X)) \quad \text{From RE semantics} \\ &= \llbracket \sigma_1 ; \sigma_2 \rrbracket (S_j(X)) \quad \text{From RE semantics}\end{aligned}$$

Hence proved.

4. Let σ be (σ_1) .

Follows from the definition of RE and LRE semantics and induction hypothesis.

5. Let σ be $\neg\sigma_1$.

We know that

$$\begin{aligned}(\llbracket \neg\lambda_1 \rrbracket (\overline{S}(X)))_i &= \neg(\llbracket \lambda_1 \rrbracket (\overline{S}(X)))_i \quad \text{From the definition of LRE semantics} \\ &= \neg \llbracket \sigma_1 \rrbracket (S_j(X)) \quad \text{From the induction hypothesis} \\ &= \llbracket \neg\sigma_1 \rrbracket (S_j(X)) \quad \text{From the definition of RE semantics}\end{aligned}$$

Hence proved.

6. Let σ be $*\sigma_1$.

From the semantics of RE, we know that $\bigvee_{l=0}^{\infty} \llbracket \sigma_1^l \rrbracket (S_j(X))$. We proceed by induction.

Basis: When $l = 0$, we know that

$$\begin{aligned} \llbracket \sigma_1^0 \rrbracket (S_j(X)) &= \llbracket \delta \rrbracket (S_j(X)) \quad \text{From the definition of RE semantics} \\ &= S_j(X) \end{aligned}$$

Also,

$$\begin{aligned} (\llbracket \lambda_1^0 \rrbracket (\bar{S}(X)))_i &= (\llbracket i : (\delta)_j \circ (G(i, \sigma_1))^0 \rrbracket (\bar{S}(X)))_i \quad \text{From equation ??} \\ &= (\llbracket i : (\delta)_j \circ i : (\delta)_i \rrbracket (\bar{S}(X)))_i \quad \text{From the definition of LRE semantics} \\ &= (\llbracket i : (\delta)_j \rrbracket (\bar{S}(X)))_i \quad \text{From the definition of LRE semantics} \\ &= S_j(X) \quad \text{From the definition of LRE semantics} \end{aligned}$$

Hypothesis: Assume that the LRE λ_1^l is equivalent to σ_1^l , $0 \leq l \leq p$.

Induction Step: We know from LRE semantics that $\lambda_1^{l+1} = \lambda_1 \circ \lambda_1^l$. From induction hypothesis and case 2 this follows $\forall l \geq 0$.

Hence proved on the structure of σ .

Example 26. Let σ be $\Upsilon_1 + \Upsilon_2$.

To get the corresponding LRE, we need to specify both the set of states we want to apply the transition clusters on and also the labels where the results of the evaluation needs to be stored. Assume that there are 2 components of the State Set Vector $\bar{S}(X)$, and we want to evaluate σ on $S_2(X)$ and store the result in $S_1(X)$, i.e. Compute $S_1(X) = \llbracket \sigma \rrbracket (S_2(X)) = \llbracket \Upsilon_1 \rrbracket (S_2(X)) \vee \llbracket \Upsilon_2 \rrbracket (S_2(X)) = \text{Img}_{TR}(S_2(X), \Upsilon_1) \vee \text{Img}_{TR}(S_2(X), \Upsilon_2)$. The equivalent LRE λ would be

$$\begin{aligned} \lambda &= \beta(1, \sigma, 2) = 1 : (\delta)_2 \circ (G(1, \sigma)) \\ &= 1 : (\delta)_2 \circ (G(1, \Upsilon_1) + G(1, \Upsilon_2)) \\ &= 1 : (\delta)_2 \circ (1 : (\Upsilon_1)_1 + 1 : (\Upsilon_2)_1) \\ &= (1 : (\delta)_2 \circ 1 : (\Upsilon_1)_1) + (1 : (\delta)_2 \circ 1 : (\Upsilon_2)_1) \end{aligned}$$

We know from LRE semantics that,

$$\begin{aligned} \llbracket \lambda \rrbracket (S_1(X), S_2(X)) &= \llbracket 1 : (\delta)_2 \circ 1 : (\Upsilon_1)_1 \rrbracket (S_1(X), S_2(X)) \sqcup \\ &\quad \llbracket 1 : (\delta)_2 \circ 1 : (\Upsilon_2)_1 \rrbracket (S_1(X), S_2(X)) \\ &= (\text{Img}_{TR}(S_2(X), \Upsilon_1), S_2(X)) \sqcup (\text{Img}_{TR}(S_2(X), \Upsilon_2), S_2(X)) \\ &= (\text{Img}_{TR}(S_2(X), \Upsilon_1) \vee \text{Img}_{TR}(S_2(X), \Upsilon_2), S_2(X)) \end{aligned}$$

Hence $\llbracket \sigma \rrbracket (S_2(X)) = (\llbracket \lambda \rrbracket (S_1(X), S_2(X)))_1$.

15 Discovering strategies

We have seen earlier that RLREs allow us to express several search strategies. Given a State Set Vector, assume that we know which components refer to the view at the root of *curr*, views at the leaves of *next* and

views at all the nodes of acc . Let λ_{null} be an RLRE which nullifies all the state components referring to the views at the leaves of $next$ to False. Let m be a node in the $curr$ tree and let m_1, \dots, m_r be its children. Let λ_m be an RLRE that computes the image I of the view at $curr$'s root under the transition relation represented at node m of TT by invoking λ_{m_i} for each child m_i of m , and finally stores the image I in the node corresponding to m in the $next$ tree. λ_m may be thought of as sequentially composing λ_{m_i} for each child m_i , followed by an update of the corresponding node, say m' , in $next$ by applying the appropriate Boolean operation on the views at the children of m' . Let λ_{up} be an RLRE which (i) accumulates the view for all state components corresponding to leaves of $next$ in the corresponding leaves of acc , (ii) computes the view for every state component referring to every internal node of acc , (iii) copies the view at the state component for the root of acc to the state component for the root of $curr$, and (iv) nullifies all the state components referring to the views at the leaves of $next$ to False. The expressive power of RLREs allows us to express all of these computations easily. Since $W(\lambda_{null}) \subset W(\lambda_{up})$, $R(\lambda_{null}) = \emptyset$ and both λ_{null} and λ_{up} nullify the views of all leaf nodes of $next$, we have $\lambda_{up} \circ \lambda_{null} = \lambda_{up}$. The i^{th} step in the fixpoint computation involves $(\lambda_{null} \circ \lambda_r \circ \lambda_{up})^i$, where r is the root of TT . Using the previous property, this reduces to being $\lambda_{null} \circ (\lambda_r \circ \lambda_{up})^i$. Using semantics of RLREs, we now show that this results of this basic reachability analysis strategy can also be obtained using alternative RLRE's, thereby permitting the discovery of different search strategies.

With the definitions of λ_n and λ_{up} as given above, $\lambda_{null} \circ \lambda_m \circ \lambda_{up} = \lambda_{null} \circ (\lambda_m \circ \lambda_{up} + \hat{\delta})$ for every node m of TT . It can then be shown that $\lambda_{null} \circ \lim_{i \rightarrow \infty} (\lambda_m \circ \lambda_{up})^i = \lambda_{null} \circ *(\lambda_m \circ \lambda_{up})$ using LRE semantics. Also, since application of λ_m involves application of λ_{m_i} for each child m_i of m as discussed above, it can be shown that $(\lambda_{null} \circ \lambda_m \circ \lambda_{up}) \succeq (\lambda_{null} \circ \lambda_{m_i} \circ \lambda_{up})$. Hence,

$$\begin{aligned} \lambda_{null} \circ *(\lambda_m \circ \lambda_{up}) &\succeq \lambda_{null} \circ *(\lambda_{m_1} \circ \lambda_{up}) \\ \lambda_{null} \circ *(\lambda_m \circ \lambda_{up}) &\succeq \lambda_{null} \circ *(\lambda_{m_2} \circ \lambda_{up}) \end{aligned}$$

Hence from LRE semantics

$$\begin{aligned} (\lambda_{null} \circ *(\lambda_m \circ \lambda_{up}))^2 &\succeq (\lambda_{null} \circ *(\lambda_{m_1} \circ \lambda_{up})) \\ &\quad \circ (\lambda_{null} \circ *(\lambda_{m_2} \circ \lambda_{up})) \end{aligned}$$

$$\text{But, } (\lambda_{null} \circ *(\lambda_m \circ \lambda_{up}))^2 = \lambda_{null} \circ *(\lambda_m \circ \lambda_{up})$$

since $\lambda_{null} \circ \lambda_{up} = \lambda_{up}$ and by Property 22 case(2). Hence, from Property 22 case(3) and the fact that $\lambda_{null} \circ \lambda_{up} = \lambda_{up}$, it follows that

$$\begin{aligned} \lambda_{null} \circ *(\lambda_m \circ \lambda_{up}) &\succeq \lambda_{null} \circ *(\lambda_{m_1} \circ \lambda_{up}) \circ \\ &\quad *(\lambda_{m_2} \circ \lambda_{up}) \end{aligned}$$

Also, from LRE semantics, we get for any λ , $\hat{\delta} + * \lambda = * \lambda$. Hence, using Property 22 case(4) we get for any RLREs λ_1 and λ_2 , $*(*\lambda_1 \circ *\lambda_2) = *(*\lambda_1 ; *\lambda_2) = *(\lambda_1 ; \lambda_2)$. Hence,

$$\begin{aligned} &\lambda_{null} \circ *(*(\lambda_{m_1} \circ \lambda_{up}) \circ *(\lambda_{m_2} \circ \lambda_{up})) \\ &= \lambda_{null} \circ *(*((\lambda_{m_1} \circ \lambda_{up}) ; (\lambda_{m_2} \circ \lambda_{up}))) \\ &= \lambda_{null} \circ *(*((\lambda_{m_1} \circ \lambda_{up}) \circ (\lambda_{m_2} \circ \lambda_{up}))) \\ &= \lambda_{null} \circ *(\lambda_m \circ \lambda_{up}) \quad \text{From definition} \end{aligned}$$

This proves that $\lambda_{null} \circ *(\lambda_r \circ \lambda_{up}) = \lambda_{null} \circ *(*(\lambda_{r_1} \circ \lambda_{up}) \circ *(\lambda_{r_2} \circ \lambda_{up}))$ where r is the root of TT and r_1 and r_2 are its child nodes. Hence, this formalises our notion of a family of strategies arising from a negation-free decomposition of Y .

16 Discovering newer strategies

Given a negation-free decomposition of Y , use of distributivity laws of conjunction and disjunction can give a variety of TT representations. Pushing a conjunction as far down the TT with respect to disjunctions is a good idea as then the overapproximation involves smaller sets of states corresponding to the views of the leaves of $curr$. Additionally, use of the modified constrain operator instead of conjunction gives exact sets of states. We now discuss a new strategy arising from this observation.

It is often the case that only a subset of I occur in the support of the Y_i 's. This subset can then be used to constrain the image operation of the Y_i 's. We illustrate this with an example. Let $Y = Y_1 \wedge Y_2 \wedge (Y_3 \vee Y_4)$, where Y_3 and Y_4 are constraint sets on I such that $\neg Y_3 \rightarrow Y_4$. Let LREs λ_i and λ encode the image computations under Y_i and Y respectively. Let $\lambda_{ij} = \lambda_i \wedge \lambda_j$. There are two ways to express λ . We can either use $\lambda_1 = (\lambda_{13} + \lambda_{14}) \wedge (\lambda_{23} + \lambda_{24})$ or $\lambda_2 = (\lambda_{13} \wedge \lambda_{23}) + (\lambda_{14} \wedge \lambda_{24})$. It can be shown using LRE semantics that $\lambda_2 \preceq \lambda_1$ and hence gives a much tighter approximation.

In addition, the modified constrain operator can be used (in place of \wedge) for efficient image computations. The resulting transition tree is shown in Figure 5. We call this strategy as input constraining and have experimented with this approach integrated with the state of the art traversal algorithms.

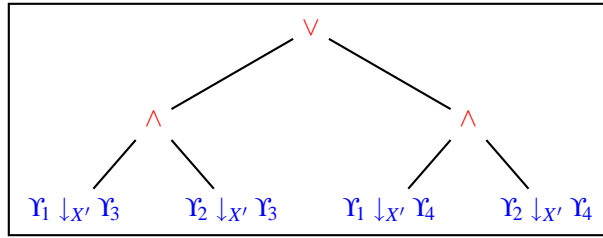


Fig. 5: Transition tree for $Y_1 \wedge Y_2 \wedge (Y_3 \vee Y_4)$

17 Encoding Traversal Methods

We first present details about encoding the state of the art traversal methods as LREs. We also establish correctness of the encoding by proving that LRE based encoding computes the same set of reachable states as the respective algorithms.

17.1 Notations

The encoding follows these notations:

1. \top stands for *true*
2. \perp stands for *false*
3. k denotes the number of projections of the state variables
4. d denotes the number of input constraints
5. $\alpha_i(S_0)$ projects the initial set of states S_0 onto the projection π_i
6. S denotes the State Set Vector of n components
7. S_i refers to the i^{th} components of S which is labeled with L_i
8. C_r refers to the r^{th} input constraint
9. $[L_{j, \forall j \neq i}]$ refers to a vector A of labels such that $L_i \notin A$ and $\forall j \neq i, L_j \in A$.

17.2 Encoding MBM

The input State Set Vector has $n = 2k$ components which are initialised as

$$\begin{aligned} \forall i \in \{1, \dots, k\}, L_i &\leftarrow \top \\ \forall i \in \{k+1, \dots, 2k\}, L_i &\leftarrow \alpha_{(i-k)}(S_0) \end{aligned}$$

It is intuitive to consider the components $\forall i \in \{1, \dots, k\}, L_i$ as the *reached_i* sets and the remaining components to store the (corresponding projections of the) initial sets of states. The order of choosing clusters is important here since for any $i, j, W(\text{traverse}_i) \cap R(\text{traverse}_j) \neq \emptyset$. So different cluster orderings could produce different sets of reachable states.

Hence the encoding of MBM becomes

$$*(\text{traverse}_1 \circ \text{traverse}_2 \circ \dots \circ \text{traverse}_k) \quad (3)$$

Also, the reachable set of states is $R_{mbm} = \bigwedge_{i=1}^{i=k} S_i$.

Lemma 24. *The reachable set of states computed by the encoding 3 is equal to the reached set of states computed by the BFS algorithm 1 i.e. $\alpha_i(\text{BFS}(\gamma(\alpha_1(S_0), \dots, \alpha_k(S_0)), [T(I, X)]_{\pi_i}))$ in the same number of iterations.*

Proof: We prove by induction on the number of iterations. We denote the set of states computed as reachable in the t^{th} iteration of the BFS algorithm 1 as *reached^t*.

Basis: When $t = 0$, i.e. at initialisation we have *reached⁰* = $\alpha_i(S_0)$. From the encoding, we get $S_i = \alpha_i(S_0)$. Hence *reached⁰* = S_i .

Hypothesis: Assume that at the end of the l^{th} iteration, $S_i = \text{reached}^l$.

Induction step: At the end of the $(l+1)^{\text{th}}$ iteration, for each projection i , the algorithm computes *reached^{l+1}* = *reached^l* \vee $\text{Img}(\text{reached}^l, \Upsilon_i)$. The encoding computes $S_i = S_i \cup \text{Img}(S_i, \Upsilon_i)$. From the induction hypothesis and computation of *reached* set of states, we get *reached^{l+1}* = S_i . Hence proved.

Lemma 25. *The reachable set of states computed by the encoding 3 is equal to the set of states computed by the MBM algorithm 3. Also, our encoding takes exactly the same number of iterations for convergence as the algorithm.*

Proof: Let *reached_i^j* denote the set of reachable states of the i^{th} projection at the end of the j^{th} iteration from the MBM algorithm. We prove by induction on the number of iterations that $R_{mbm} = \bigwedge_{i=1}^{i=k} \text{reached}_i^c$, where c denotes the iteration at which convergence was obtained.

Basis: When $j = 0$, i.e. at initialisation, we have from the encoding

$$\forall i \in \{1, \dots, k\}, L_i \leftarrow \top$$

Henced, $R_{mbm} = \bigwedge_{i=1}^{i=j} S_i = \top$. Also, from the MBM algorithm we know that $\forall i, \text{reached}_i^0 = \top$. Hence proved.

Hypothesis: Assume that at the end of the t^{th} iteration, we have $R_{mbm} = \bigwedge_{i=1}^{i=k} \text{reached}_i^t$.

Induction Step: Follows from Lemma 24.

17.3 Encoding RFBF

The input State Set Vector has $n = 2k$ components which are initialised as

$$\begin{aligned} \forall i \in \{1, \dots, k\}, L_i &\leftarrow \alpha_i(S_0) \\ \forall i \in \{k+1, \dots, 2k\}, L_i &\leftarrow \perp \end{aligned}$$

It is intuitive to consider the components $\forall i \in \{1, \dots, k\}$, L_i as the $reached_i$ sets and the remaining components to store intermediate computations. RFBF is encoded as

$$\lambda = *(traverse_1 \circ \dots \circ traverse_k \circ update_Labels) \quad (4)$$

$$\forall i, traverse_i = (L_{k+i} : (\Upsilon_i \downarrow [L_j \forall j \neq i, j \leq k])_{L_i}) + L_{k+i} : (\delta)_{L_i} \quad (5)$$

$$\text{And, } update_Labels = \circ_{i=1}^{i=k} (L_i : (\delta)_{L_{k+i}}) \quad (6)$$

The clusters can be chosen in any order as for any i, j , $W(traverse_i) \cap R(traverse_j) = \emptyset$. Also, the set of reachable states is computed as $R_{r_{f_{bf}}} = \bigwedge_{i=1}^{i=k} S_i$.

Lemma 26. *The reachable set of states computed by the encoding 5 and 6 is equal to the set of states computed by the RFBF algorithm 4 in the same number of iterations.*

Proof: We prove by induction on the number of iterations l .

Basis: When $l = 0$, we have $\forall i$, $reached_i = \alpha_i(S_0) = S_i$.

Hypothesis: Assume that at the end of the t^{th} iteration, we have $\forall i$, $reached_i = S_i$.

Induction Step: At the end of the $(t+1)^{th}$ iteration, we have from the RFBF algorithm $\forall i$, $reached_i = reached_i \vee \text{Img}(\gamma(reached_1^t, \dots, reached_k^t), \Upsilon_i)$. We know from induction hypothesis that $\forall i$, $reached_i = S_i$, and $\forall i$, $reached_i^t = S_{k+i}$. So the encoding also computes the same set after the $(t+1)^{th}$ iteration. Hence proved.

17.4 Encoding TFBF

TFBF incurs a huge memory overhead as every set of to states computed in each iteration prior to convergence has to be stored. Also, every convergence check involves conjunction of the to sets stored. This would result in huge memory blowup. In our framework, we encode only a specified number of TFBF iterations, say t without regard for convergence.

The input State Set Vector has $n = (t+1)k$ components which are initialised as

$$\forall i \in \{1, \dots, k\}, L_i \leftarrow \alpha_i(S_0) \quad (7)$$

$$\forall i \in \{k+1, \dots, tk+k\}, L_i \leftarrow \perp \quad (8)$$

It is intuitive to consider the components $\forall i \in \{1, \dots, k\}$, L_i as the $reached_i$ sets and the remaining components to store the to sets of states in each iteration. TFBF is encoded as

$$(traverse_1^1 \circ traverse_2^1 \circ \dots \circ traverse_k^1) \circ \dots \circ (traverse_1^t \circ traverse_2^t \circ \dots \circ traverse_k^t)$$

$$traverse_i^j = (L_{jk+i} : (\Upsilon_i \downarrow [L_{(j-1)k+l} \forall l \neq i])_{L_{(i-1)k+i}}) \circ (L_i : (\delta)_{L_i} + L_i : (\delta)_{L_{jk+i}})$$

The clusters can be chosen in any order because the write and read sets of the $traverse_k^i$ for any k , given an i are disjoint. Also, the set of reachable states is computed as $R_{tfbf} = \bigwedge_{i=1}^{i=k} S_i$.

Lemma 27. *The reachable set of states computed by the TFBF encoding upto t iterations is equal to the set of states computed by the TFBF algorithm 4 upto t iterations.*

Proof: We prove by induction on the number of iterations l .

Basis: When $l = 0$, we have $\forall i, reached_i = \alpha_i(S_0) = S_i$.

Hypothesis: Assume that at the end of the l^{th} iteration, we have $\forall i, reached_i = S_i$.

Induction Step: At the end of the $(l + 1)^{th}$ iteration, we have from the TFBF algorithm $\forall i, reached_i = reached_i \vee \text{Img}(\gamma(to_1^l, \dots, to_k^l), \Upsilon_i)$. We know from induction hypothesis that $\forall i, reached_i = S_i$, and $\forall i, to_i^l = S_{(l-1)k+i}$. Hence our encoding also computes the same set after the $(l + 1)^{th}$ iteration. Hence proved.

17.5 Encoding TMBM

TMBM is a hybrid traversal of TFBF and MBM. Its intuitive encoding description is

$$(TFBF_for_t_iterations) \circ (update_labels_for_MBM) \circ (MBM_till_convergence) \circ (compute_reachable_states) \quad (9)$$

$$update_labels_for_MBM = \circ_{i=1}^{i=k} (L_{(t+1)k+i} : (\delta)_{L_{(tk+i)}})$$

$$MBM_till_convergence = *(traverse_1 \circ \dots \circ traverse_k) \\ \forall i, traverse_i = (L_{(t+1)k+k+i} : (\delta)_{L_{(t+1)k+i}}) \circ \\ * \left(L_{(t+1)k+k+i} : (\Upsilon_i \downarrow [L_{(t+1)k+k+j} \forall j \neq i])_{L_{(t+1)k+k+i}} \right)$$

$$compute_reachable_states = \circ_{i=1}^{i=k} \left(L_i : (\delta)_{L_i} + L_i : (\delta)_{L_{(t+1)k+k+i}} \right)$$

The input State Set Vector has $n = 3k + tk$ components and the set of reachable states is given by $\bigwedge_{i=1}^{i=k} S_i$.

17.6 Encoding with Input Constraining

Let d be the number of input constraints.

18 Encoding MBM with input constraining

We use the encoding intuition of having

L_1 to L_d For the projection π_1 to store set of initial states
 L_{d+1} to L_{2d} For the projection π_2 to store set of initial states
 \vdots
 $L_{(k-1)d+1}$ to L_{kd} For the projection π_k to store set of initial states

And,

L_{kd+1} to L_{kd+d} For the projection π_1 to store computed set of reachable states
 L_{kd+d+1} to L_{kd+2d} For the projection π_2 to store computed set of reachable states
 \vdots
 $L_{kd+(k-1)d+1}$ to L_{2kd} For the projection π_k to store computed set of reachable states

We use an input State Set Vector of length $n = 2kd$, which is initialised as

$$\begin{aligned} \forall i \in \{1, \dots, k\}, \forall j \in \{1, \dots, d\} \quad L_{(i-1)d+j} &\leftarrow \alpha_i(S_0) \\ \forall i \in \{kd+1, \dots, 2kd\}, L_i &\leftarrow \top \end{aligned}$$

The encoding of the procedure is $*(traverse_1 \circ \dots \circ traverse_k)$, where

$$\begin{aligned} \forall i, \quad traverse_i &= \circ_{j=1}^{j=d} traverse_{ij} \\ \forall i, j, \quad traverse_{ij} &= L_{kd+(i-1)d+j} : (\delta)_{L_{(i-1)d+j}} \\ &\circ * \left(+_{m=1}^{m=d} (L_{kd+(i-1)d+j} : (\Upsilon_i \downarrow [C_j, L_{kd+(p-1)d+m, p \neq i}])_{L_{kd+(i-1)d+m}}) \right) \end{aligned}$$

The set of reachable states is given by $\bigvee_{j=1}^{j=d} \left(\bigwedge_{i=1}^{i=k} S_{kd+(i-1)d+j} \right)$.

18.1 Encoding RFBF with input constraining

We use the encoding intuition of having

$$\begin{aligned}
L_1 \text{ to } L_d & \text{ For the projection } \pi_1 \text{ to store current set of reachable states} \\
L_{d+1} \text{ to } L_{2d} & \text{ For the projection } \pi_2 \text{ to store current set of reachable states} \\
& \vdots \\
L_{(k-1)d+1} \text{ to } L_{kd} & \text{ For the projection } \pi_k \text{ to store current set of reachable states}
\end{aligned}$$

And,

$$\begin{aligned}
L_{kd+1} \text{ to } L_{kd+d} & \text{ For the projection } \pi_1 \text{ to store newly computed set of reachable states} \\
L_{kd+d+1} \text{ to } L_{kd+2d} & \text{ For the projection } \pi_2 \text{ to store newly computed set of reachable states} \\
& \vdots \\
L_{kd+(k-1)d+1} \text{ to } L_{2kd} & \text{ For the projection } \pi_k \text{ to store newly computed set of reachable states}
\end{aligned}$$

We use an input State Set Vector of length $n = 2kd$, which are initialised as

$$\forall i \in \{1, \dots, k\}, \forall j \in \{1, \dots, d\} L_{(i-1)d+j} \leftarrow \alpha_i(S_0)$$

The encoding of the procedure is $*(\text{traverse}_1 \circ \dots \circ \text{traverse}_k \circ \text{update}_1 \circ \dots \circ \text{update}_k)$, where

$$\begin{aligned}
\forall i, \text{traverse}_i &= \circ_{j=1}^{j=d} \text{traverse}_{ij} \\
\forall i, j, \text{traverse}_{ij} &= L_{kd+(i-1)d+j} : (\delta)_{L_{(i-1)d+j}} \\
&+ +_{m=1}^{m=d} \left(L_{kd+(i-1)d+j} : \left(\Upsilon_i \downarrow [C_j, L_{(p-1)d+m, \forall p \neq i}] \right)_{L_{(i-1)d+m}} \right) \\
\forall i, \text{update}_i &= \circ_{j=1}^{j=d} \left(L_{(i-1)d+j} : (\delta)_{L_{kd+(i-1)d+j}} \right)
\end{aligned}$$

The set of reachable states is given by $\bigvee_{j=1}^{j=d} \left(\bigwedge_{i=1}^{i=k} S_{(i-1)d+j} \right)$.

18.2 Encoding TFBBF with input constraining

We use the encoding intuition of having

$$\begin{aligned}
L_1 \text{ to } L_d & \text{ For the projection } \pi_1 \text{ to store current set of reachable states} \\
L_{d+1} \text{ to } L_{2d} & \text{ For the projection } \pi_2 \text{ to store current set of reachable states} \\
& \vdots \\
L_{(k-1)d+1} \text{ to } L_{kd} & \text{ For the projection } \pi_k \text{ to store current set of reachable states}
\end{aligned}$$

$$\begin{aligned}
L_{kd+(t-1)kd+(i-1)d+1} \text{ to } L_{kd+(t-1)kd+(i-1)d+d} & \text{ For the projection } \pi_i \text{ to store the } t\text{o} \text{ sets computed} \\
& \text{using the } j^{\text{th}} \text{ Input Constraint in the } t^{\text{th}} \text{ iteration}
\end{aligned}$$

The input State Set Vector has $n = kd + tkd$ components which are initialised as

$$\forall i \in \{1, \dots, k\}, \forall j \in \{1, \dots, d\} L_{(i-1)d+j} \leftarrow \alpha_i(S_0)$$

We claim that the encoding λ for t iterations of TFBBF with input constraining is

$$\begin{aligned} \lambda &= (\text{traverse}_1^1 \circ \dots \circ \text{traverse}_k^1) \circ \dots \circ (\text{traverse}_1^t \circ \dots \circ \text{traverse}_k^t) \quad \text{where,} \\ \forall i \in \{1, \dots, k\}, \text{traverse}_i^p &= \circ_{j=1}^{j=d} (\text{traverse}_{ij}^p) \quad \text{such that} \\ \text{traverse}_{ij}^p &= +_{m=1}^{m=d} \left(L_{kd+(p-1)kd+(i-1)d+j} : \left(\Upsilon_i \downarrow [C_j, L_{kd+(p-2)kd+(i-1)d+m, \forall c \neq i}] \right)_{L_{kd+(p-2)kd+(i-1)d+m}} \right) \\ &\quad \circ \left(L_{(i-1)d+k} : (\delta)_{L_{(i-1)d+j}} + L_{(i-1)d+j} : (\delta)_{L_{kd+(p-1)kd+(i-1)d+j}} \right) \end{aligned}$$

The set of reachable states is given by $\bigvee_{j=1}^{j=d} \left(\bigwedge_{i=1}^{i=k} S_{(i-1)d+j} \right)$.

18.3 Encoding TMBM with input constraining

As TMBM with input constraining is a hybrid traversal of TFBBF with input constraining and MBM with input constraining, we follow a similar intuitive encoding description as for TMBM

$$\begin{aligned} &(\text{TFBBF} - \text{IC_for_}t\text{_iterations}) \circ (\text{update_Labels_for_MBM} - \text{IC}) \circ (\text{MBM} - \text{IC_till_convergence}) \\ &\quad \circ (\text{compute_reachable_states}) \quad (10) \end{aligned}$$

Following the intuition of encoding TMBM given in Equation 10 and the general encoding intuition given in Equation ??, we formalise the description of traverse_i , update_Labels , $\text{update_Labels_for_MBM} - \text{IC}$ and $\text{compute_reachable_states}$ for TMBM based traversal with input constraining.

$$\text{update_Labels_for_MBM} - \text{IC} = \circ_{i=1}^{i=k} (L_{tkd+kd+i} : (\delta)_{L_{kd+(t-1)kd+i}})$$

$$\text{MBM} - \text{IC_till_convergence} = *(\text{traverse}_1 \circ \dots \circ \text{traverse}_k)$$

$$\forall i, \text{traverse}_i = \circ_{j=1}^{j=d} \text{traverse}_{ij}$$

$$\begin{aligned} \forall i, j, \text{traverse}_{ij} &= L_{tkd+2kd+(i-1)d+j} : (\delta)_{L_{tkd+kd+(i-1)d+j}} \\ &\quad \circ * \left(+_{m=1}^{m=d} (L_{tkd+2kd+(i-1)d+j} : (\Upsilon_i \downarrow [C_j, L_{tkd+2kd+(p-1)d+m, \forall p \neq i}])_{L_{tkd+2kd+(i-1)d+m}}) \right) \end{aligned}$$

$$\text{compute_reachable_states} = \circ_{i=1}^{i=k} \text{update}_i$$

$$\forall i, \text{update}_i = \circ_{j=1}^{j=d} \text{update}_{ij}$$

$$\forall i, j, \text{update}_{ij} = L_{tkd+2kd+(i-1)d+j} : (\delta)_{L_{tkd+2kd+(i-1)d+j}} + L_{tkd+2kd+(i-1)d+j} : (\delta)_{L_{kd+(t-1)kd+(i-1)d+j}}$$

The input State Set Vector has $n = tkd + 3kd$ components, and the set of reachable states is given by

$$\bigvee_{j=1}^{j=d} \left(\bigwedge_{i=1}^{i=k} S_{tkd+2kd+(i-1)d+j} \right).$$

19 Reachability Matrices

Given a transition system $B = (I, X, Q, S_0, \mathbf{T})$, a Reachability Matrix (RM) [22] is another framework which permits operations over State Set Vectors. An RM M of order $n \times n$ is a matrix with n^2 terms, each $(r_{ij}, \forall i, j)$ of which is an RE (possibly with \neg operator) over the set C . The REs are applied row-wise to the input State Set Vector (of length n) and are aggregated to get a component of the transformed State Set Vector (also of length n). Formally, let M be an RM which takes \bar{S} as an input State Set Vector and computes the transformed State Set Vector \bar{R} . The application of M on \bar{S} is denoted as $\llbracket M \rrbracket(\bar{S}(X))$ and is evaluated as shown

$$M = \begin{pmatrix} r_{11} & r_{12} & \dots & r_{1n} \\ r_{21} & r_{22} & \dots & r_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ r_{n1} & r_{n2} & \dots & r_{nn} \end{pmatrix}$$

$\llbracket M \rrbracket(S_1(X), S_2(X), \dots, S_n(X)) = (R_1(X), R_2(X), \dots, R_n(X))$ where $\forall i \in \{1, 2, \dots, n\}$
 $R_i(X) = \llbracket r_{i1} \rrbracket(S_1(X)) \vee \llbracket r_{i2} \rrbracket(S_2(X)) \vee \dots \vee \llbracket r_{in} \rrbracket(S_n(X))$.

19.1 Syntax of RM Expressions

Let $\{M_1, \dots, M_p\}$ be a finite set of $n \times n$ RMs over C_T . The syntax of an RM expression, henceforth called an RME over $C_{T,n}$ is a terminal string derived from the syntax

$$\begin{aligned} M &\rightarrow M + M \\ &| M \circ M \\ &| M ; M \\ &| *M \\ &| (M) \\ &| \neg M \\ &| \Delta \\ &| \Phi \\ &| M_1 | \dots | M_p \end{aligned}$$

19.2 Semantics of RMEs

An RME M of order $n \times n$ can be considered as a transformer of a State Set Vector of length n into another State Set Vector also of length n . The semantics of M is denoted as $\llbracket M \rrbracket_B : (2^Q)^n \rightarrow (2^Q)^n$, where Q is the set of states of B expressed as a disjunction of k transitions. We shall henceforth omit the subscript B when it is clear from the context. Let $\bar{S}(X)$ be characteristic function of a State Set Vector of length n .

We present an inductive definition of the semantics of RMEs.

- Φ is the Additive Identity RM where $\forall i, j, \Phi_{ij} = \theta$.
 Define $\llbracket \Phi \rrbracket(\bar{S}(X)) = (0, \dots, 0)$.

- Δ is the Multiplicative Identity RM where

$$\Delta_{ij} = \begin{cases} \delta & \text{if } i = j \\ \theta & \text{otherwise} \end{cases}$$

- such that $\llbracket \Delta \rrbracket (\bar{S}(X)) = \bar{S}(X)$.
- $\llbracket M_1 + M_2 \rrbracket (\bar{S}(X)) = \llbracket M_1 \rrbracket (\bar{S}(X)) \sqcup \llbracket M_2 \rrbracket (\bar{S}(X))$
- $\llbracket M_1 \circ M_2 \rrbracket (\bar{S}(X)) = \llbracket M_2 \rrbracket (\llbracket M_1 \rrbracket (\bar{S}(X)))$
- $\llbracket M_1 ; M_2 \rrbracket (\bar{S}(X)) = \llbracket (M_1 + \Delta) \circ (M_2 + \Delta) \rrbracket (\bar{S}(X))$
- $\llbracket (M) \rrbracket (\bar{S}(X)) = \llbracket M \rrbracket (\bar{S}(X))$
- Let $M^0 = \Delta$, and $\forall i \geq 1$, $M^i = M^{i-1} \circ M$, then $\llbracket *M \rrbracket (\bar{S}(X)) = \sqcup_{i=0}^{\infty} \llbracket (M)^i \rrbracket (\bar{S}(X))$
- $\llbracket \neg M \rrbracket (\bar{S}(X)) = \neg \llbracket M \rrbracket (\bar{S}(X))$ where \neg refers to component wise negation.

19.3 Operations on Reachability Matrices and Properties

We briefly describe some operations on RMEs taken from [22].

- Given two RMEs R and Q, define an RM $P = R + Q$, where

$$\forall i, \forall j, P_{ij} = R_{ij} + Q_{ij}$$

- Given two RMEs R and Q, define an RM $P = R \circ Q$, where

$$\forall i, \forall j, P_{ij} = +_{l=1}^n R_{il} \circ Q_{lj} \quad (11)$$

- Given two RMEs R and Q, define an RM $P = R ; Q$ to be $(R + \Delta) \circ (Q + \Delta)$.

- **Definition 35.** *Negation free RME:* An RME R is said to be negation - free if it has no occurrence of the \neg operator.

Given a negation free RME $*R$, it can be shown using Arden's Lemma [4] that it can be expressed as a unique RM P.

20 Expressive Equivalence of LREs and RMEs

In this section, we establish the expressive equivalence of LREs and RMEs by showing that every LRE λ over $\mathcal{T}_{\Upsilon, n}$ can be formulated as an equivalent RME M over $C_{\Upsilon, n}$ and vice-versa.

20.1 Preliminaries

Definition 36. *Equivalence of an LRE λ and an RME M*

An LRE λ over $\mathcal{T}_{\Upsilon, n}$ and an RME M over $C_{\Upsilon, n}$ are said to be **semantically equivalent** expressions, denoted $\lambda \simeq M$ iff for any transition system and any State Set Vector \bar{S} of n components, $\llbracket \lambda \rrbracket (\bar{S}(X)) = \llbracket M \rrbracket (\bar{S}(X))$.

Lemma 28. *Let λ be any LRE expression over $\mathcal{T}_{\Upsilon, n}$. There exists a semantically equivalent RME M over $C_{\Upsilon, n}$ such that $\lambda \simeq M$.*

Proof. We show by induction on the structure of λ that there exists a semantically equivalent RME M. Define a function

$$\rho : LRE \rightarrow RME$$

which maps an LRE to its equivalent RME $\rho(\lambda)$. We prove that $\lambda \simeq \rho(\lambda)$.

Basis:

1. If $\lambda = i : (\delta)_j$.

We define $\rho(\lambda) = M$ such that

$$M_{kl} = \begin{cases} \delta & \text{if } k = i \text{ and } l = j \\ \theta & \text{else if } k = i \text{ and } l \neq j \\ \delta & \text{else if } k = l \\ \theta & \text{otherwise} \end{cases}$$

We claim that $i : (\delta)_j \simeq \rho(i : (\delta)_j)$.

From the semantics of LRE, λ would be evaluated as

$$\llbracket i : (\delta)_j \rrbracket (S_1(X), S_2(X), \dots, S_i(X), \dots, S_n(X)) = (S_1(X), S_2(X), \dots, S_j(X), \dots, S_n(X)).$$

Also we know from RME semantics that

$$\llbracket \rho(i : (\delta)_j) \rrbracket (S_1(X), S_2(X), \dots, S_i(X), \dots, S_n(X)) = (S_1(X), S_2(X), \dots, S_j(X), \dots, S_n(X)).$$

We see that

$$\llbracket i : (\delta)_j \rrbracket (S_1(X), S_2(X), \dots, S_n(X)) = \llbracket \rho(i : (\delta)_j) \rrbracket (S_1(X), S_2(X), \dots, S_n(X))$$

Hence proved $i : (\delta)_j \simeq \rho(i : (\delta)_j)$.

Note: $(i : (\delta)_i = \hat{\delta}) \simeq \Delta$.

2. If $\lambda = i : (\theta)_j$.

We define $\rho(\lambda) = M$ such that

$$M_{kl} = \begin{cases} \theta & \text{if } k = l = i \\ \delta & \text{else if } k = l \neq i \\ \theta & \text{otherwise} \end{cases}$$

We claim that $i : (\theta)_j \simeq \rho(i : (\theta)_j)$.

From the semantics of LRE, λ would be evaluated as $\llbracket i : (\theta)_j \rrbracket (S_1(X), S_2(X), \dots, S_i(X), \dots, S_n(X)) = (S_1(X), S_2(X), \dots, 0, \dots, S_n(X))$.

Also we know from RME semantics that

$$\llbracket \rho(i : (\theta)_j) \rrbracket (S_1(X), S_2(X), \dots, S_i(X), \dots, S_n(X)) = (S_1(X), S_2(X), \dots, 0, \dots, S_n(X)).$$

We see that

$$\llbracket i : (\theta)_j \rrbracket (\bar{S}(X)) = \llbracket \rho(i : (\theta)_j) \rrbracket (\bar{S}(X))$$

Hence proved $i : (\theta)_j \simeq \rho(i : (\theta)_j)$.

3. If $\lambda = i : (\Upsilon)_k$.

We define $\rho(\lambda) = M$ such that

$$M_{kl} = \begin{cases} \Upsilon_j & \text{if } k = i \text{ and } l = j \\ \theta & \text{else if } k = i \text{ and } l \neq j \\ \delta & \text{else if } k = l \\ \theta & \text{otherwise} \end{cases}$$

We claim that $i : (\Upsilon)_k \simeq \rho(i : (\Upsilon)_k)$.

From the semantics of LRE, λ would be evaluated as $\llbracket i : (\Upsilon)_k \rrbracket (S_1(X), S_2(X), \dots, S_i(X), \dots, S_n(X)) = (S_1(X), S_2(X), \dots, \text{Img}_{TR}(S_k(X), \Upsilon_j), \dots, S_n(X))$.

Also we know from RME semantics that

$$\llbracket \rho(i : (\Upsilon_j)_k) \rrbracket (S_1(X), S_2(X), \dots, S_i(X), \dots, S_n(X)) = (S_1(X), S_2(X), \dots, \text{Img}_{TR}(S_k(X), \Upsilon_j), \dots, S_n(X)).$$

We see that

$$\llbracket i : (\Upsilon_j)_k \rrbracket (\bar{S}(X)) = \llbracket \rho(i : (\Upsilon_j)_k) \rrbracket (\bar{S}(X))$$

Hence proved $i : (\Upsilon_j)_k \simeq \rho(i : (\Upsilon_j)_k)$.

Hypothesis: Let λ_1 and λ_2 be two LREs over \mathcal{T} , then there exist semantically equivalent RMEs M_1 and M_2 over \mathcal{C}_Υ and a State Set Vector of length n , where $M_1 = \rho(\lambda_1)$ and $M_2 = \rho(\lambda_2)$ such that

$$\begin{aligned} \lambda_1 &\simeq M_1, \text{ and} \\ \lambda_2 &\simeq M_2 \end{aligned}$$

Induction Step:

We consider the following cases:

1. Let $\lambda = \lambda_1 + \lambda_2$.

We define $\rho(\lambda) = M$ such that

$$\begin{aligned} \forall k \leq n, \forall l \leq n \ M_{kl} &= (\rho(\lambda_1))_{kl} + (\rho(\lambda_2))_{kl} \\ &= (M_1)_{kl} + (M_2)_{kl} \end{aligned}$$

We claim that $\lambda \simeq \rho(\lambda)$.

$$\begin{aligned} \llbracket \lambda \rrbracket (\bar{S}(X)) &= \llbracket \lambda_1 + \lambda_2 \rrbracket (\bar{S}(X)) && \text{by definition of LRE semantics} \\ &= \llbracket \lambda_1 \rrbracket (\bar{S}(X)) \sqcup \llbracket \lambda_2 \rrbracket (\bar{S}(X)) && \text{by definition of LRE semantics} \\ &= \llbracket M_1 \rrbracket (\bar{S}(X)) \sqcup \llbracket M_2 \rrbracket (\bar{S}(X)) && \text{by induction hypothesis} \\ &= \llbracket M_1 + M_2 \rrbracket (\bar{S}(X)) && \text{Hence proved that} \\ &= \llbracket M \rrbracket (\bar{S}(X)) && \text{by definition of RME semantics} \\ & && \text{by definition of } \rho(\lambda) \end{aligned}$$

$\lambda \simeq \rho(\lambda)$.

2. Let $\lambda = \lambda_1 \circ \lambda_2$.

We define $\rho(\lambda) = M$ such that

$$\begin{aligned} \forall k \leq n, \forall l \leq n \ M_{kl} &= (\rho(\lambda_1) \circ \rho(\lambda_2))_{kl} \\ &= +_{i=1}^{i=n} [(\rho(\lambda_1))_{ki} \circ (\rho(\lambda_2))_{il}] \\ &= +_{i=1}^{i=n} [(M_1)_{ki} \circ (M_2)_{il}] \end{aligned}$$

We claim that $\lambda \simeq \rho(\lambda)$.

$$\begin{aligned} \llbracket \lambda \rrbracket (\bar{S}(X)) &= \llbracket \lambda_1 \circ \lambda_2 \rrbracket (\bar{S}(X)) && \text{by definition of LRE semantics} \\ &= \llbracket \lambda_2 \rrbracket (\llbracket \lambda_1 \rrbracket (\bar{S}(X))) && \text{by definition of LRE semantics} \\ &= \llbracket \lambda_2 \rrbracket (\bar{R}), \text{ where } \bar{R} = \llbracket \lambda_1 \rrbracket (\bar{S}(X)) = \llbracket M_1 \rrbracket (\bar{S}(X)) && \text{by induction hypothesis} \\ &= \llbracket M_2 \rrbracket (\bar{R}) && \text{by induction hypothesis} \\ &= \llbracket M_2 \rrbracket (\llbracket M_1 \rrbracket (\bar{S}(X))) && \\ &= \llbracket M_1 \circ M_2 \rrbracket (\bar{S}(X)) && \text{by definition of RME semantics} \\ &= \llbracket M \rrbracket (\bar{S}(X)) && \text{by definition of } \rho(\lambda) \end{aligned}$$

Hence proved that $\lambda \simeq \rho(\lambda)$.

3. Let $\lambda = \lambda_1 ; \lambda_2$.

Proof: Define $\rho(\lambda) = \rho((\lambda_1 + i : (\delta)_i) \circ (\lambda_2 + i : (\delta)_i))$. We claim that $\lambda \simeq \rho(\lambda)$. We know from the induction hypothesis that

$$\lambda_1 \simeq M_1$$

$$\lambda_2 \simeq M_2$$

Also, from the basis, for any $i, i : (\delta)_i \simeq \Delta$

From case 1

$$\lambda_1 + i : (\delta)_i \simeq M_1 + \Delta \quad (12)$$

$$\text{and, } \lambda_2 + i : (\delta)_i \simeq M_2 + \Delta \quad (13)$$

Also from case 2 and Equations 12 and 13,

$$(\lambda_1 + i : (\delta)_i) \circ (\lambda_2 + i : (\delta)_i) \simeq (M_1 + \Delta) \circ (M_2 + \Delta)$$

From the definition of RME semantics we get,

$$(\lambda_1 ; \lambda_2) \simeq (M_1 ; M_2)$$

Hence proved that $\lambda \simeq \rho(\lambda)$.

4. Let $\lambda = * \lambda_1$.

Define $\rho(\lambda) = * \rho(\lambda_1) = * M_1$ (From induction hypothesis). We claim that $\lambda \simeq \rho(\lambda)$. From the semantics of LRE, we know that $\llbracket * \lambda_1 \rrbracket (\overline{S}(X)) = \bigsqcup_{i=0}^{\infty} \llbracket \lambda_1^i \rrbracket (\overline{S}(X))$. We prove by induction on i .

Basis: When $i=0$, $\llbracket \lambda_1^0 \rrbracket (\overline{S}(X)) = \llbracket \hat{\delta} \rrbracket (\overline{S}(X))$. We know that the RME equivalent of $\hat{\delta}$ is Δ .

Hypothesis: Assume that the RME equivalent of λ_1^i is M_1^i , $0 \leq i \leq k$.

Induction step: By definition, we know that $\lambda_1^{k+1} = \lambda_1 \circ \lambda_1^k$. So,

$$\begin{aligned} \llbracket \lambda_1^{k+1} \rrbracket (\overline{S}(X)) &= \llbracket \lambda_1 \circ \lambda_1^k \rrbracket (\overline{S}(X)) \\ &= \llbracket \lambda_1^k \rrbracket (\llbracket \lambda_1 \rrbracket (\overline{S}(X))) \\ &= \llbracket M_1^k \rrbracket (\llbracket M_1 \rrbracket (\overline{S}(X))) \text{ by induction hypothesis} \\ &= \llbracket M_1 \circ M_1^k \rrbracket (\overline{S}(X)) \text{ from RME semantics} \\ &= \llbracket M_1^{k+1} \rrbracket (\overline{S}(X)) \text{ from RME semantics} \end{aligned}$$

So, for any i , if an assignment of s satisfies $(\llbracket * \lambda_1 \rrbracket (\overline{S}(X)))_i$, then there exists an integer $k (\geq 0)$ such that an assignment of s satisfies $(\llbracket (\lambda_1)^k \rrbracket (\overline{S}(X)))_i$. Since $(M_1)^l$ is the RME equivalent for the LRE $(\lambda_1)^l \forall l \geq 0$, we have $(\llbracket (M_1)^k \rrbracket (\overline{S}(X)))_i = (\llbracket (\lambda_1)^k \rrbracket (\overline{S}(X)))_i$. Hence, an assignment of s satisfies $(\llbracket * M_1 \rrbracket (\overline{S}(X)))_i$ as well. Hence proved.

5. Let $\lambda = (\lambda_1)$.

Define $\rho(\lambda) = M$ such that

$$\forall k, l, M_{kl} = (\rho(\lambda_1))_{kl}$$

We claim that $\lambda \simeq \rho(\lambda)$.

$$\begin{aligned} \llbracket \lambda \rrbracket (\overline{S}(X)) &= \llbracket (\lambda_1) \rrbracket (\overline{S}(X)) && \text{By definition of } \lambda \\ &= \llbracket \lambda_1 \rrbracket (\overline{S}(X)) && \text{From the definition of LRE semantics} \\ &= \llbracket M_1 \rrbracket (\overline{S}(X)) && \text{From induction hypothesis} \\ &= \llbracket (M_1) \rrbracket (\overline{S}(X)) && \text{From the definition of RME semantics} \\ &= \llbracket M \rrbracket (\overline{S}(X)) && \text{From the definition of } \rho(\lambda) \end{aligned}$$

Hence proved that $\lambda \simeq \rho(\lambda)$.

6. Let $\lambda = \neg(\lambda_1)$.

Define $\rho(\lambda) = \neg M$ such that

$$\forall k, l, M_{kl} = (\rho(\lambda_1))_{kl}$$

We claim that $\lambda \simeq \rho(\lambda)$.

$$\begin{aligned} \llbracket \lambda \rrbracket (\overline{S}(X)) &= \llbracket \neg(\lambda_1) \rrbracket (\overline{S}(X)) && \text{By definition of } \lambda \\ &= \neg \llbracket \lambda_1 \rrbracket (\overline{S}(X)) && \text{From the definition of LRE semantics} \\ &= \neg \llbracket M_1 \rrbracket (\overline{S}(X)) && \text{From induction hypothesis} \\ &= \llbracket \neg(M_1) \rrbracket (\overline{S}(X)) && \text{From the definition of RME semantics} \\ &= \llbracket M \rrbracket (\overline{S}(X)) && \text{From the definition of } \rho(\lambda) \end{aligned}$$

Hence proved that $\lambda \simeq \rho(\lambda)$.

7. Let $\lambda = i : (\Upsilon_j \downarrow_{X'} [C]_k)$.

Let $[C] = [r_1, r_2, \dots, r_p]$ be a vector of p labels.

We first prove that for any i, j, k , if $\lambda_1 = \neg i : (\delta)_i$ and $\lambda_2 = \neg i : (\delta)_j$, then

$$(\llbracket \neg\lambda_1 + \lambda_2 \rrbracket (\overline{S}(X)))_i = S_j(X) \wedge S_k(X).$$

$$\begin{aligned} (\llbracket \neg\lambda_1 + \lambda_2 \rrbracket (\overline{S}(X)))_i &= (\neg(\llbracket \neg\lambda_1 + \neg\lambda_2 \rrbracket (\overline{S}(X))))_i \\ &= (\neg(\llbracket \neg\lambda_1 \rrbracket (\overline{S}(X)) \sqcup \llbracket \neg\lambda_2 \rrbracket (\overline{S}(X))))_i && \text{From LRE semantics} \\ &= \neg \neg S_j(X) \vee \neg \neg S_k(X) && \text{From LRE semantics} \\ &= S_j(X) \wedge S_k(X) \end{aligned}$$

This can hence be extended to $p, p \geq 2$ collection of $\lambda_j, 1 \leq j \leq p$ where each $\lambda_j = i : (\delta)_{r_j}$, then

$$(\llbracket \neg(+_{j=1}^{j=p} \lambda_j) \rrbracket (\overline{S}(X)))_i = \bigwedge_{j=1}^{j=p} (S_{r_j}(X)). \text{ Hence for this case we know that } \llbracket \lambda \rrbracket (\overline{S}(X)) =$$

$$\begin{aligned} &\llbracket \neg(\neg(i : (\delta)_{r_1}) + \dots + \neg(i : (\delta)_{r_p}) + \neg(i : (\delta)_k)) \\ &\quad \circ (i : (\Upsilon_j)_i) \rrbracket (\overline{S}(X)) \end{aligned}$$

Construction of $\rho(\lambda)$ follows from the basis, induction hypothesis and proof cases 1 and 2. Hence proved that $\lambda \simeq \rho(\lambda)$.

Hence proved by induction on the structure of λ .

Lemma 29. Any $n \times n$ RM M over C_Υ can be expressed as a semantically equivalent LRE λ over $\mathcal{T}_{\Upsilon, n}$ given by the following translation

$$\lambda = +_{i=1}^{i=n} \lambda'_i \quad \text{where,} \tag{14}$$

$$\forall i, \lambda'_i = \lambda_i \circ (\circ_{\forall j \neq i} j : (\theta)_j) \tag{15}$$

$$\forall i, \lambda_i = \lambda_{i1} + \lambda_{i2} + \dots + \lambda_{in} \tag{16}$$

$$\lambda_{ij} = \beta(i, r_{ij}, j) = (i : (\delta)_j) \circ G(i, r_{ij}) \tag{17}$$

Proof. Let $\overline{R}(X) = \llbracket M \rrbracket (\overline{S}(X))$. Then by definition of RME semantics,

$$R_i(X) = \llbracket r_{i1} \rrbracket (S_1(X)) \vee \llbracket r_{i2} \rrbracket (S_2(X)) \vee \dots \vee \llbracket r_{in} \rrbracket (S_n(X))$$

where $\forall i, j, r_{ij}$ is an RE over C_Y . We claim that $M \simeq \lambda$.

Let $\overline{Q}(X) = \llbracket \lambda \rrbracket (\overline{S}(X))$. Then for all i , we will prove that $Q_i(X) = R_i(X)$. We know that

$$\begin{aligned} Q_i(X) &= (\llbracket \lambda \rrbracket (\overline{S}(X)))_i \\ &= (\llbracket \lambda'_1 + \lambda'_2 + \dots + \lambda'_n \rrbracket (\overline{S}(X)))_i \quad \text{From Equation 14} \\ &= (\llbracket \lambda'_1 \rrbracket (\overline{S}(X)) \sqcup \llbracket \lambda'_2 \rrbracket (\overline{S}(X)) \sqcup \dots \sqcup \llbracket \lambda'_n \rrbracket (\overline{S}(X)))_i \quad \text{From the definition of LRE semantics} \\ &= (\llbracket \lambda'_1 \rrbracket (\overline{S}(X)))_i \vee (\llbracket \lambda'_2 \rrbracket (\overline{S}(X)))_i \vee \dots \vee (\llbracket \lambda'_n \rrbracket (\overline{S}(X)))_i \quad \text{From the definition of LRE semantics} \end{aligned}$$

We will first prove that $\forall j \neq i, (\llbracket \lambda'_j \rrbracket (\overline{S}(X)))_j = 0$

$$\begin{aligned} (\llbracket \lambda'_i \rrbracket (\overline{S}(X)))_j &= (\llbracket \lambda_i \circ (\circ_{\forall j \neq i} j : (\theta)_j) \rrbracket (\overline{S}(X)))_j \quad \text{From Equation 15} \\ &= (\llbracket \circ_{\forall j \neq i} j : (\theta)_j \rrbracket (\llbracket \lambda_i \rrbracket (\overline{S}(X))))_j \quad \text{From the definition of LRE semantics} \\ &= (\overline{R}(X))_j \quad \text{where } \forall j, R_j(X) = (\llbracket \circ_{\forall j \neq i} j : (\theta)_j \rrbracket (\llbracket \lambda_i \rrbracket (\overline{S}(X))))_j \\ &= 0 \quad \text{From the definition of LRE semantics} \end{aligned}$$

Hence,

$$\begin{aligned} Q_i(X) &= (\llbracket \lambda'_i \rrbracket (\overline{S}(X)))_i \\ &= (\llbracket \lambda_i \rrbracket (\overline{S}(X)))_i \quad \text{Since } W(\circ_{\forall j \neq i} j : (\theta)_j) \cap \{i\} = \emptyset. \\ &= (\llbracket \lambda_{i1} + \lambda_{i2} + \dots + \lambda_{in} \rrbracket (\overline{S}(X)))_i \quad \text{From Equation 15} \\ &= (\llbracket \lambda_{i1} \rrbracket (\overline{S}(X)))_i \vee (\llbracket \lambda_{i2} \rrbracket (\overline{S}(X)))_i \vee \dots \vee (\llbracket \lambda_{in} \rrbracket (\overline{S}(X)))_i \quad \text{From the definition of LRE semantics} \\ &= \llbracket r_{i1} \rrbracket (S_1(X)) \vee \llbracket r_{i2} \rrbracket (S_2(X)) \vee \dots \vee \llbracket r_{in} \rrbracket (S_n(X)) \quad \text{From Equation 17} \\ &= R_i(X) \end{aligned}$$

Hence proved.

Lemma 30. *Every RME R over $C_{Y,n}$ can be expressed as a semantically equivalent LRE λ over $\mathcal{T}_{Y,n}$ such that for a State Set Vector $S(X)$ of length n and any transition system $\llbracket R \rrbracket (\overline{S}(X)) = \llbracket \lambda \rrbracket (\overline{S}(X))$.*

Proof. We prove by induction on the structure of RMEs.

Basis: For every $RM \in \{M_1, \dots, M_k\} \cup \{\Delta, \Phi\}$, the lemma follows from Lemma 29.

Induction hypothesis: Assume that RMEs R_1 and R_2 over $C_{Y,n}$ can be expressed as semantically equivalent LREs λ_1 and λ_2 respectively over $\mathcal{T}_{Y,n}$.

Induction Step: We consider the following cases:

1. If $R = R_1 + R_2$, then we prove that $\lambda = \lambda_1 + \lambda_2$.

$$\begin{aligned} \llbracket R \rrbracket (\overline{S}(X)) &= \llbracket R_1 + R_2 \rrbracket (\overline{S}(X)) && \text{by definition of RME semantics} \\ &= \llbracket R_1 \rrbracket (\overline{S}(X)) \sqcup \llbracket R_2 \rrbracket (\overline{S}(X)) && \text{by definition of RME semantics} \\ &= \llbracket \lambda_1 \rrbracket (\overline{S}(X)) \sqcup \llbracket \lambda_2 \rrbracket (\overline{S}(X)) && \text{by induction hypothesis} \\ &= \llbracket \lambda_1 + \lambda_2 \rrbracket (\overline{S}(X)) && \text{by definition of LRE semantics} \\ &= \llbracket \lambda \rrbracket (\overline{S}(X)) \end{aligned}$$

Hence proved.

2. If $R = R_1 \circ R_2$, then we prove that $\lambda = \lambda_1 \circ \lambda_2$.

$$\begin{aligned}
\llbracket R \rrbracket(\overline{S}(X)) &= \llbracket R_1 \circ R_2 \rrbracket(\overline{S}(X)) && \text{by definition of RME semantics} \\
&= \llbracket R_2 \rrbracket(\llbracket R_1 \rrbracket(\overline{S}(X))) && \text{by definition of RME semantics} \\
&= \llbracket R_2 \rrbracket(\llbracket \lambda_1 \rrbracket(\overline{S}(X))) && \text{by induction hypothesis} \\
&= \llbracket \lambda_2 \rrbracket(\llbracket \lambda_1 \rrbracket(\overline{S}(X))) && \text{by induction hypothesis} \\
&= \llbracket \lambda_1 \circ \lambda_2 \rrbracket(\overline{S}(X)) && \text{by definition of LRE semantics} \\
&= \llbracket \lambda \rrbracket(\overline{S}(X))
\end{aligned}$$

Hence proved.

3. If $R = \neg R_1$, then we prove that $\lambda = \neg \lambda_1$.

$$\begin{aligned}
\llbracket R \rrbracket(\overline{S}(X)) &= \llbracket \neg R_1 \rrbracket(\overline{S}(X)) && \text{by definition of RME semantics} \\
&= \neg \llbracket R_1 \rrbracket(\overline{S}(X)) && \text{by definition of RME semantics} \\
&= \neg \llbracket \lambda_1 \rrbracket(\overline{S}(X)) && \text{by induction hypothesis} \\
&= \llbracket \neg \lambda_1 \rrbracket(\overline{S}(X)) && \text{by definition of LRE semantics} \\
&= \llbracket \lambda \rrbracket(\overline{S}(X))
\end{aligned}$$

Hence proved.

4. If $R = (R_1)$, then we prove that $\lambda = (\lambda_1)$.

$$\begin{aligned}
\llbracket R \rrbracket(\overline{S}(X)) &= \llbracket (R_1) \rrbracket(\overline{S}(X)) && \text{by definition of RME semantics} \\
&= \llbracket (\lambda_1) \rrbracket(\overline{S}(X)) && \text{by induction hypothesis} \\
&= \llbracket \lambda \rrbracket(\overline{S}(X))
\end{aligned}$$

Hence proved.

5. If $R = *R_1$, we prove that $\lambda = *\lambda_1$.

From the semantics of RME, we know that $\llbracket *R_1 \rrbracket(\overline{S}(X)) = \bigsqcup_{i=0}^{\infty} \llbracket R_1^i \rrbracket(\overline{S}(X))$. We prove by induction on i .

Basis: When $i=0$, $\llbracket R_1^0 \rrbracket(\overline{S}(X)) = \llbracket \Delta \rrbracket(\overline{S}(X))$. We know that the LRE equivalent of Δ is $\hat{\delta}$.

Hypothesis: Assume that the LRE equivalent of R_1^i , $0 \leq i \leq k$ is λ_1^i .

Induction step: By definition, we know that $R_1^{k+1} = R_1 \circ R_1^k$. So,

$$\begin{aligned}
\llbracket R_1^{k+1} \rrbracket(\overline{S}(X)) &= \llbracket R_1 \circ R_1^k \rrbracket(\overline{S}(X)) \\
&= \llbracket R_1^k \rrbracket(\llbracket R_1 \rrbracket(\overline{S}(X))) \\
&= \llbracket \lambda_1^k \rrbracket(\llbracket \lambda_1 \rrbracket(\overline{S}(X))) && \text{by induction hypothesis} \\
&= \llbracket \lambda_1 \circ \lambda_1^k \rrbracket(\overline{S}(X)) && \text{from LRE semantics} \\
&= \llbracket \lambda_1^{k+1} \rrbracket(\overline{S}(X)) && \text{from LRE semantics}
\end{aligned}$$

So, for any i , if an assignment of s satisfies $(\llbracket *R_1 \rrbracket(\overline{S}(X)))_i$, then there exists an integer $k (\geq 0)$ such that an assignment of s satisfies $(\llbracket (R_1)^k \rrbracket(\overline{S}(X)))_i$. Since $(\lambda_1)^k$ is the LRE equivalent for the RME $(R_1)^k \forall l \geq 0$, we have $(\llbracket (M_1)^k \rrbracket(\overline{S}(X)))_i = (\llbracket (\lambda_1)^k \rrbracket(\overline{S}(X)))_i$. Hence, an assignment of s satisfies $(\llbracket *\lambda_1 \rrbracket(\overline{S}(X)))_i$ as well.

Hence proved.

Theorem 16. Every LRE λ over $\mathcal{T}_{\gamma,n}$ can be expressed as a semantically equivalent RME M over $C_{\gamma,n}$ and vice versa.

Proof. Follows from Lemma 28, Lemma 29 and Lemma 30. Hence, LREs and RMEs are expressively equivalent.

Corollary 5. *Every negation free and constrain free LRE λ over $\mathcal{T}_{\gamma,n}$ can be expressed as a semantically equivalent negation free RME M over $C_{\gamma,n}$ and vice versa.*

Proof. Follows from Theorem 16.

Lemma 31. *Every negation free and constrain free LRE λ over $\mathcal{T}_{\gamma,n}$ can be expressed as a semantically equivalent negation free RME M over $C_{\gamma,n}$ but there is atleast one RME M with negation which is not semantically equivalent to an RM M' over C .*

Proof. Follows from Corollary 5 and properties of RMEs listed in Section 19.3. The expressive equivalence of LREs and RMEs follows from Theorem 16. With the negation operator RMEs are strictly more expressive than RMs. We illustrate this with an example.

Consider a State Set Vector $\bar{S} = \{S_1, S_2\}$ with 2 components. Let

$$M_1 = \begin{pmatrix} \delta & \theta \\ \theta & \theta \end{pmatrix}$$

and

$$M_2 = \begin{pmatrix} \theta & \delta \\ \theta & \theta \end{pmatrix}$$

Consider the RME $\neg(\neg(M_1) + \neg(M_2))$. Evaluating this RME on \bar{S} gives

$$\begin{aligned} \llbracket \neg(\neg(M_1) + \neg(M_2)) \rrbracket(\bar{S}(X)) &= \neg(\llbracket \neg(M_1) \rrbracket(\bar{S}(X)) \sqcup \llbracket \neg(M_2) \rrbracket(\bar{S}(X))) \text{ From the definition of RME semantics} \\ &= \neg(\neg S_1(X), 1) \sqcup (\neg S_2(X), 1) \text{ From the definition of RME semantics} \\ &= \neg(\neg S_1(X) \vee \neg S_2(X), 1) \\ &= (S_1(X) \wedge S_2(X), 0) \end{aligned}$$

This RME $\neg(\neg(M_1) + \neg(M_2))$ cannot be expressed as a RM. Let R be a 2×2 RM such that $\llbracket R \rrbracket(\bar{S}(X)) = (S_1(X) \wedge S_2(X), 0)$. We want to determine $r_{ij}, \forall i, j$. In other words, determine

$$\llbracket r_{11} \rrbracket(S_1(X)) \vee \llbracket r_{12} \rrbracket(S_2(X)) = S_1(X) \wedge S_2(X) \quad \text{and,} \quad (18)$$

$$\llbracket r_{21} \rrbracket(S_1(X)) \vee \llbracket r_{22} \rrbracket(S_2(X)) = 0 \quad (19)$$

Equation 19 has a trivial solution: $r_{21} = r_{22} = \theta$. Equation 18 has no solution.

21 Effect of negation operator

It is more intuitive to view the framework of REs, LREs and RMEs as a mechanism to specify a sequence of image computations over certain sets of states. So these frameworks allow us to talk about sequences of image computations rather than a single image operation. This allows us to consider these frameworks as a language which permits image computation operations over subsets of Q . Each image computation sequence can be thought of as a word of a language whose alphabet contains the individual image operations which are used to for the sequence. For instance, the $+$ operator can be viewed as the union operator in any language, the \circ operator performs concatenation of two strings, while the $*$ operator performs Kleene closure. But with the negation operator, the effect of evaluating the negation is on the result obtained by the application

of that image sequence rather than over the image sequence itself. We illustrate this point with an example of the STD of a transition system shown in Figure 6. Given the characteristic function $S(X)$ of a set of states S , let $\tilde{\sigma}$ be the image computation sequence not allowed by σ , then in general

$$\begin{aligned} \llbracket \neg\sigma \rrbracket (S(X)) &\neq \llbracket \tilde{\sigma} \rrbracket (S(X)) \\ \llbracket \neg\sigma \rrbracket (S(X)) &= \neg \llbracket \sigma \rrbracket (S(X)) \quad \text{From the definition of RE semantics} \end{aligned}$$

For handling languages that allow the negation operation, we would require using a much richer theory of μ -calculus.

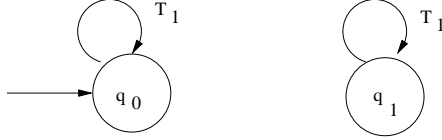


Fig. 6: Effect of Negation operator

Consider Figure 6, let the state markings refer to their characteristic functions. Suppose we want to evaluate $\neg(*Y_1)$ on the characteristic function q_0 of the initial set of states S_0 , we get,

$$\begin{aligned} \llbracket \neg(*Y_1) \rrbracket (q_0) &= \neg \llbracket *Y_1 \rrbracket (q_0) \\ &= q_1 \\ \text{Whereas } \llbracket *\tilde{Y}_1 \rrbracket (q_0) &= 0 \end{aligned}$$

22 Experimental results and Discussion

We have described how several strategies can be encoded as LREs. To evaluate the effectiveness of our approach, we have implemented an interpreter for LREs on top of the BDD-based reachability analysis engine NuSMV [14]. It takes as inputs: (i) a description of a boolean decomposition of the transition relation, (ii) an LRE, and (iii) an initial set of states. Figure 7 shows the Layered Architecture of our generic framework. The reachability engine of NuSMV at the backend is used for image computations. We believe that it should be possible to integrate our framework with other backend engines in future. The middle layer provides some book-keeping and set-theoretic functionality used to compute the set of reachable states. The topmost interpreter layer allows us to write *meta-programs* as LREs encoding search strategies.

Our framework allows more ease in experimenting with different search strategies. Implementing such a framework in NuSMV would require lot of familiarity with the functions available and parameter passing required. In comparison, we believe that learning to use LREs is much easier, more intuitive and less cumbersome. Additionally, every LRE can be shown to be equivalent to a set of equations in modal μ -calculus. Hence, a modal μ -calculus prover can be integrated to our framework to assist in checking certain kinds of relations (containment, entailment and equivalence) between the semantics of two LREs. This would also help to check if the semantics of an LRE is correct by checking its relation in this *prover* component with the *vanilla* strategies which are known to be computationally inefficient for large circuits. Our implementation currently does support this prover component.

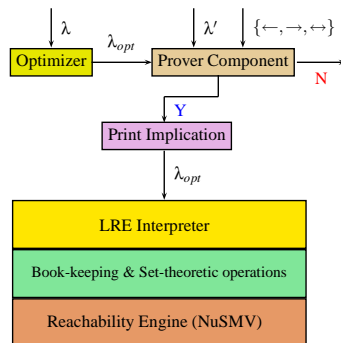


Fig. 7: Layered framework architecture

We report reachability analysis results of the algorithms from [12] using overlapping projections of the state variables on some large circuits of the publicly available ISCAS-89 benchmark suite. In addition, we also compare these with the results of using the new strategy of input constraining. All our experiments were conducted on a 1.66GHz Intel Pentium-IV processor with 512MB main memory, and running Fedora Core Linux 3.4.3-6.fc3. Table 1 presents details of reachability analysis performed using our tool on some

Circuit	PI	FF	projections	TMBM			TMBM-input constraining			Relative gain
				BDD	reach	time	BDD	reach	time	
s13207.v	31	669	47	219769	1.03e-115	1427.4	455164	1.39e-125	5844	1.35e-10
s15850.v	14	597	64	496117	2.81e-101	3095.3	851332	3.32e-110	6136.2	1.18e-09
s38584.v	12	1452	205	235198	4.93e-57	7996.9	1410362	5.25e-61	9343.4	1.06e-04
s35932.v	35	1728	54	1015639	3.99e-71	25811.4	2194369	2.93e-78	51362.6	0.73e-07

Table 1: Comparison of reachability analysis with and without input constraining

large circuits from ISCAS-89 benchmark suite. The columns of PI, FF and projections refer to the number of primary inputs, state variables and projections respectively. The projections were provided to us by the authors of [18] and are the same ones used by them. All the circuits were traversed using the TMBM [12] approach using 10 iterations of TFBF initially. We also encoded and experimented with other strategies discussed in [12], all of which show comparable reachable state sets, but involve overheads of time and BDD sizes. Setting up our experimental framework takes only around a couple of hours for each circuit in contrast to building custom reachability analyzers using either NuSMV or VIS which in our experience takes much longer time. In addition, our framework also allows us to ensure the correctness and soundness of the encoded strategies. We also report results of using the idea of input constraining on these circuits. Columns BDD, reach and time refer to the Peak BDD count, reachable fraction of the state space and traversal time (in seconds) respectively. The last column shows the relative gain of using input constraining. Without input constraining, our tool gives reachable fractions comparable to those reported in [18] except for the circuit s15850.v where use of a different cluster ordering gives us tighter overapproximation. For the largest circuit, s35932.v, Govindaraju *et al.* in [18] don't report any results. We observe that our tool

requires more BDD nodes than in [18] which can be attributed to the book-keeping operations involved, or use of a different BDD package in [18], or some heuristic optimizations unreported in [18]. As was expected, both the peak BDD count and time taken show sharp increase with the use of input constraining. Note that lower this fraction, tighter is the approximation of the reachable state space computed. We report orders of magnitude improvement for most of the circuits with the use of input constraining. The tool is available at <http://www.cfdvs.iitb.ac.in/~seetha/FMCAD07/nusmvp/>.

The dependency plot shown in Figure 8 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 similar next state equations as their corresponding groups in other boxes. Only the state variables belonging to the first box depend on the PIs.

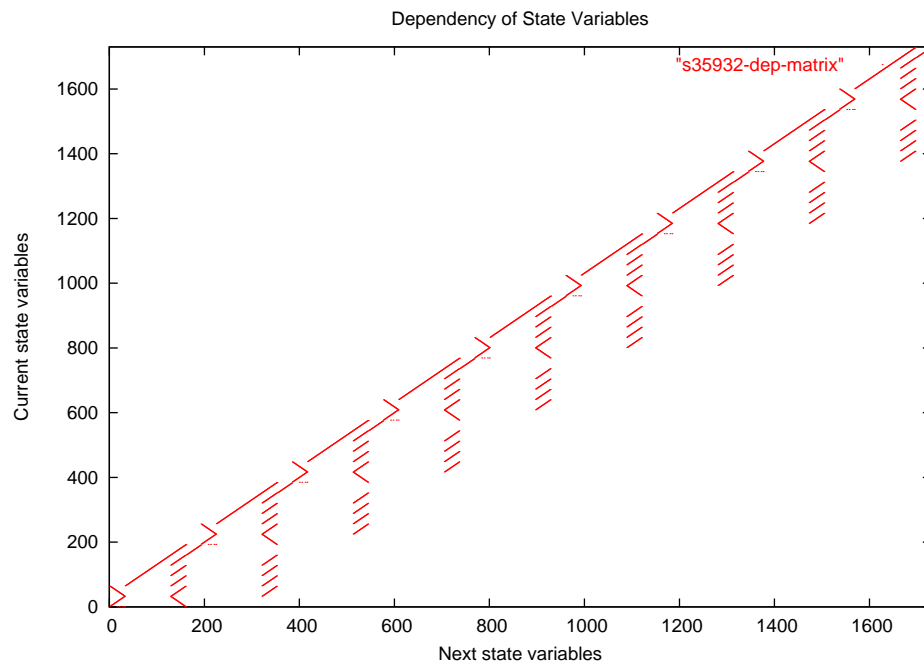


Fig. 8: Dependency Plot of s35932.v

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 [12] but like Govindaraju *et al.* in [18], we were also unable to generate 54 clusters based on the partitioning strategy proposed by Cho *et al.* in [12]. 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.

22.0.1 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 9

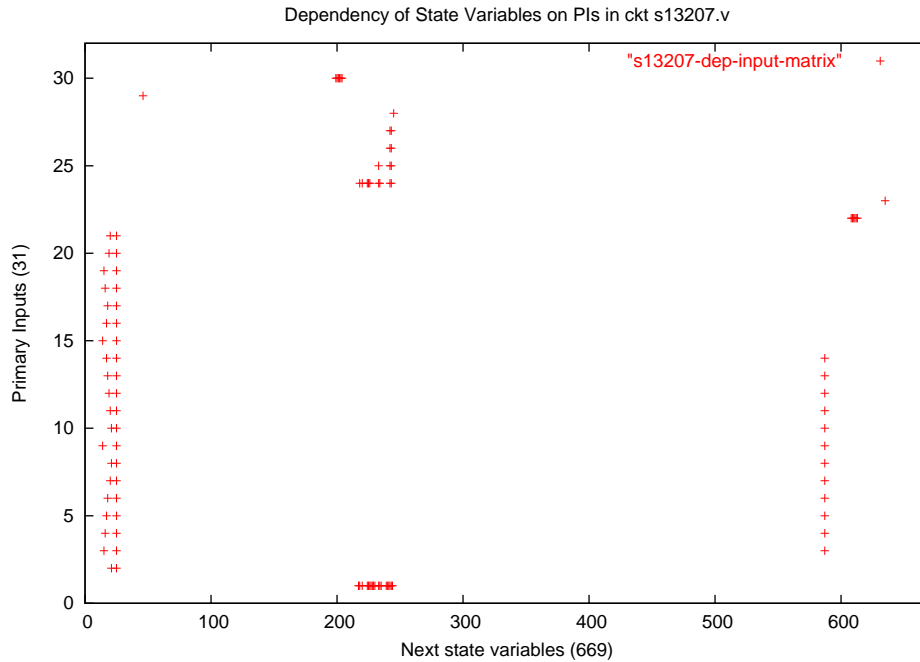


Fig. 9: Dependency on PI in s13207.v

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 9, 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.

23 Conclusions & Future work

In this report, we presented an extension of Thomas et al's work [32] on reachability expressions, and showed how negation-free decompositions of a circuit's transition relation can be used to systematically derive and discover LREs encoding strategies for computing the set of reachable states. Encoding such strategies as LREs gives the unique advantage of being able to do comparative reasoning in a formal setting. It also allows implementing these strategies quickly using our tool. Thus it achieves the dual goal of easy

implementation and formal reasoning about alternative search strategies. However, there are limitations of LREs as well. Since each LRE encodes a static order of computing images and combining them, we are currently unable to encode strategies that adapt on-the-fly. Similarly, the restriction of using only a finite pre-determined number of labels/tags imposes restrictions on encoding strategies like full-blown TFBB [12] or pure depth-first search. We intend to augment the expressive power of LREs to be able to capture such strategies as well, while being able to efficiently decide implication and equivalence checking between LREs. The latter is important in order to check the soundness and completeness of new strategies encoded as LREs. We hope that the framework of LREs will eventually help designers and verification engineers easily encode their intuitions into search strategies, and also help discover new provably correct strategies for reachability analysis.

24 Syntax Directed Translation and Reachability Operators

In this chapter, we detail the syntax directed translation of LREs using the reachability operators used by our implementation framework.

$$\begin{aligned}
LE &\rightarrow L : (E)_L && \{LE.code = \$_{L_1.index} : E.code(\$_{L_2.index})\} \\
&| LE + LE && \{LE.code = union(LE_1.code, LE_2.code)\} \\
&| LE \circ LE && \{LE.code = fix(fix(\$, LE_1.code, 1), LE_2.code, 1)\} \\
&| LE; LE && \{LE.code = fixU(fixU(\$, LE_1.code, 1), LE_2.code, 1)\} \\
&| \neg LE && \{LE.code = compl(LE_1.code)\} \\
&| (LE) && \{LE.code = LE_1.code\} \\
&| \#LE && \{LE.code = fixU(\$, LE_1.code, inf)\} \\
\\
E &\rightarrow Y_j && \{E.code(\$_i) = img(\$_i, j)\} \\
&| Y_j \downarrow [C] && \{E.code(\$_i) = imgmc(\$_i, C.code, j)\} \\
&| \delta && \{E.code(\$_i) = \$_i\} \\
&| \theta && \{E.code(\$_i) = \emptyset\} \\
\\
C &\rightarrow L, C && \{C.code = \$_{L.index}, C.code\} \\
&| L && \{C.code = \$_{L.index}\} \\
\\
L &\rightarrow L_i && \{L.index = i\}
\end{aligned}$$

24.1 Operators and Functions used

1. $fixU(S, op, count)$: Return value: $S \cup op(S) \cup op^2(S) \cup \dots \cup op^{count}(S)$.
2. $fix(S, op, count)$: Return value: $op^{count}(S)$.
3. $union(S_1, S_2)$: Return value: Computes component wise union of the State Set Vectors S_1 and S_2 .
4. $compl(S_1)$: Return value: $U \setminus S_1$, computes set difference of State Set Vector U (each component of which is Q) and S_1 .
5. $img(S, cnum)$: Returns the image of the set of states S computed using the BFV of the $cnum^{th}$ cluster.
6. $imgmc(S, S_vector, cnum)$: Returns the image of the implicit conjunction of the set of states S and a vector of sets of states under the BFV of the $cnum^{th}$ cluster using multiple constrain operator.
7. $null_vector()$: Return value: returns a State Set Vector all of whose components are \emptyset .

Note that we use $\$_j$ to refer to the j^{th} component of the context State Set Vector.

24.2 Evaluation algorithm for LREs

```

vector * eval(char * cmd, vector * S) {
    subcmd[] = parse(cmd);

    if(subcmd[0] == fixU) {
        count = subcmd[3];
        S_1 = eval(subcmd[1],S);
        do{
            if(count != infinity)
                count--;
            S_2 = eval(subcmd[2],S_1);
            S_1 = S_1 U S_2;
        }while(doesnt_change(S_1) & count >= 0)
        return S_1;
    }

    if(subcmd[0] == fix) {
        count = subcmd[3];
        S_1 = eval(subcmd[1],S);
        do {
            if(count != infinity)
                count--;
            S_2 = S_1;
            S_1 = eval(subcmd[2],S_2);
        }while(S_1 != S_2 & count >= 0)
        return S_1;
    }

    if(subcmd[0] == Union) {
        S_1 = eval(subcmd[1], S);
        S_2 = eval(subcmd[2], S);
        S_1 = S_1 U S_2;
        return S_1;
    }

    if(subcmd[0] == compl) {
        S_1 = eval(subcmd[1], S);
        S_2 = U \ S_1;
        return S_2;
    }

    if(subcmd[0] == $i: Img) {
        label = i;
        cluster = subcmd[2];
    }
}

```

```

    trans_vector = make_bfv_vector(cluster);
    S_1 = S;
    Scomp = eval_comp(subcmd[1], S);
    Simage = Image(Scomp, trans_vector);
    S_1[label] = Simage;
    return S_1;
}

if(subcmd[0] == $i: Imgmc) {
    label = i;
    S_1 = S;
    Scomp = eval_comp(subcmd[1], S);
    cluster = subcmd[subcmdlength-1];
    trans_vector = make_bfv_vector(cluster);
    index = 2;
    do {
        Sconstraint = eval_comp(subcmd[index], S);
        index++;
        add_to_list(constraint_list, Sconstraint);
    }while(index != subcmdlength-2)
    Simage = Imagemc(Scomp, constraint_list, trans_vector);
    S_1[label] = Simage;
    return S_1;
}

if(subcmd[0] == D) {
    return S;
}

if(subcmd[0] == B) {
    S_1 = null_vector();
    return S_1;
}

if(subcmd[0] == $i:-) {
    label = i;
    S_1 = S;
    S_1[label] = I_0;
    return S_1;
}

if(subcmd[0] == $i: $j) {
    label = i;
    S_1 = S;
    S_1[label] = S[j];
    return S_1;
}

```

```

    }

    if(subcmd[0] == $i: b) {
        label = i;
        S_1 = S;
        S_1[label] = set_empty();
        return S_1;
    }

    if(subcmd[0] == $i: t) {
        label = i;
        S_1 = S;
        S_1[label] = set_tautology();
        return S_1;
    }
}

set * eval_comp(char * subcmd, vector * S) {
    index = get_index(subcmd);
    return S[index];
}

```

References

1. Parosh Aziz Abdulla, Per Bjesse, and Niklas Eén. Symbolic reachability analysis based on sat-solvers. In *TACAS*, pages 411–425, 2000.
2. Parosh Aziz Abdulla, Per Bjesse, and Niklas Eén. Symbolic Reachability Analysis Based on SAT-Solvers. In *TACAS*, 2000.
3. Sheldon B. Akers. Binary Decision Diagrams. In *IEEE Transactions on Computers*, volume 6, pages 509–516, 1978.
4. D. N. Arden. Delayed logic and finite state machines. In *Theory of Computing Machine Design*, pages 1–35, 1960.
5. Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. *LNCS*, 1579:193–207, 1999.
6. Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, 35(8):677–691, 1986.
7. G. Cabodi, P. Camurati, and S. Quer. Can BDDs compete with SAT solvers on Bounded Model Checking? In *DAC*, pages 117–122, 2002.
8. Gianpiero Cabodi and Paolo Camurati. Symbolic fsm traversals based on the transition relation. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 16(5):448–457, 1997.
9. Gianpiero Cabodi, Paolo Camurati, Luciano Lavagno, and Stefano Quer. Disjunctive partitioning and partial iterative squaring: an effective approach for symbolic traversal of large circuits. In *DAC*, pages 728–733, 1997.
10. Gianpiero Cabodi, Sergio Nocco, and Stefano Quer. Improving sat-based bounded model checking by means of bdd-based approximate traversals. In *DATE*, page 10898, 2003.
11. P. Chauhan, E. Clarke, and D. Kroening. Using sat based image computation for reachability analysis. Technical Report CMU-CS-03-151, School of Computer Science, Carnegie Mellon University, 2003.
12. Hyunwoo Cho, Gary D. Hachtel, Enrico Macii, Bernard Plessier, and Fabio Somenzi. Algorithms for approximate fsm traversal. In *DAC*, pages 25–30, 1993.

13. Hyunwoo Cho, Gary D. Hachtel, Enrico Macii, Massimo Poncino, and Fabio Somenzi. Automatic state space decomposition for approximate fsm traversal based on circuit analysis. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 15(12):1451–1464, 1996.
14. Alessandro Cimatti, Edmund M. Clarke, Fausto Giunchiglia, and Marco Roveri. NUSMV: A new symbolic model verifier. In *CAV*, pages 495–499, 1999.
15. O. Coudert and J. Madre. A Unified Framework for the Formal Verification of Sequential Circuits. In *ICCAD*, pages 126–129, November 1989.
16. D. Geist and I. Beer. Efficient model checking by automated ordering of transition relation. In *CAV*, volume 818, pages 299–310, 1994.
17. Shankar G. Govindaraju, David L. Dill, and Jules P. Bergmann. Improved approximate reachability using auxiliary state variables. In *DAC*, pages 312–316, 1999.
18. Shankar G. Govindaraju, David L. Dill, Alan J. Hu, and Mark A. Horowitz. Approximate reachability with BDDs using overlapping projections. In *DAC*, pages 451–456, 1998.
19. Aarti Gupta, Zijiang Yang, Pranav Ashar, and Anubhav Gupta. Sat-based image computation with application in reachability analysis. In *FMCAD*, pages 354–371, 2000.
20. Youpyo Hong, Peter A. Beerel, Jerry R. Burch, and Kenneth L. McMillan. Sibling-substitution-based bdd minimization using don't cares. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 19(1):44–55, 2000.
21. 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 *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 1–33, 1990.
22. Varun Kanade. Guided Symbolic Reachability using Partitioning. Master's thesis, Dept. of C. S. E., IIT Bombay, 2006.
23. C. Y. Lee. Representation of Switching Circuits by Binary Decision Programs. Technical Report 38, Bell Systems Technical Journal, 1959.
24. K. L. McMillan. A Conjunctively Decomposed Boolean Representation for Symbolic Model Checking. In *CAV*, pages 13–25, 1990.
25. Kenneth L. McMillan. Interpolation and SAT-Based Model Checking. In *CAV*, pages 1–13, 2003.
26. In-Ho Moon, James H. Kukula, Kavitha Ravi, and Fabio Somenzi. To Split or to Conjoin: The Question in Image Computation. In *DAC*, pages 271–275, 2000.
27. Amit Narayan, Adrian J. Isles, Jawahar Jain, Robert K. Brayton, and Alberto L. Sangiovanni-Vincentelli. Reachability analysis using partitioned-robdds. In *ICCAD*, pages 388–393, 1997.
28. R. Ranjan, A. Aziz, R. Brayton, B. Plessier, and C. Pixley. Efficient bdd algorithms for fsm synthesis and verification, 1995.
29. Kavita Ravi and Fabio Somenzi. High-density reachability analysis. In *ICCAD*, pages 154–158, 1995.
30. Kavita Ravi and Fabio Somenzi. Hints to accelerate symbolic traversal. In *CHARME*, 1999.
31. Dina Thomas, Supratik Chakraborty, and Paritosh Pandya. Efficient Guided Symbolic Reachability using Reachability Expressions. Technical Report TR-06-19, CFDVS, IIT Bombay, 2006.
32. Dina Thomas, Supratik Chakraborty, and Paritosh K. Pandya. Efficient guided symbolic reachability using reachability expressions. In *TACAS*, pages 120–134, 2006.
33. Hervé J. Touati, Hamid Savoj, Bill Lin, Robert K. Brayton, and Alberto L. Sangiovanni-Vincentelli. Implicit state enumeration of finite state machines using bdds. In *ICCAD*, pages 130–133, 1990.
34. Poul Frederick Williams, Armin Biere, Edmund M. Clarke, and Anubhav Gupta. Combining decision diagrams and SAT procedures for efficient symbolic model checking. In *Computer Aided Verification*, pages 124–138, 2000.