Approximate Symbolic Reachability of Networks of Transition Systems

Sudeep Juvekar[1], Ankur Taly[1], Varun Kanade[2], and Supratik Chakraborty[1]

[1] Indian Institute of Technology, Bombay, India
{sjuvekar,ankur123,supratik}@cse.iitb.ac.in
[2] Georgia Institute of Technology, USA
varunk@cc.gatech.edu

Abstract. Symbolic reachability analysis of networks of state transition systems present special optimization opportunities that are not always available in monolithic state transition systems. These optimizations can potentially allow scaling of reachability analysis to much larger networks than can be handled using existing techniques. In this paper, we discuss a set of techniques for efficient approximate reachability analysis of large networks of small state transition systems with *local interactions*, and analyze their relative precision and performance in a BDD-based tool. We use overlapping projections to represent the state space, and discuss optimizations that significantly limit the set of variables in the support set of BDDs that must be manipulated to compute the image of each projection due to a transition of the system. The ideas presented in this paper have been implemented in a BDD-based symbolic reachability analyzer built using the public-domain symbolic model checking framework of NuSMV. We report experimental results on a set of benchmarks that demonstrate the effectiveness of our approach over existing techniques using overlapping projections.

1 Introduction

Large and complex systems are often built by interconnecting small and simple components. A large class of such systems can be behaviorally modeled as networks of interacting state transition systems, where each individual state transition system, or *component*, has a simple transition structure and involves only a few state variables. Examples of such systems abound in practice, e.g. circuits obtained by interconnecting logic gates and flip-flops, distributed control systems with interacting sensors, controllers and actuators, a collection of devices communicating through a shared bus and arbiter, etc. The reachable state space of such a system can be computed by starting from a specified initial state of all components and by non-deterministically choosing and atomically executing an enabled state transition from the individual components. This produces a change of state of one or more components, and hence of the overall system. Interactions between components can be modeled by sharing state variables and by executing synchronized transitions between components. The above process can then be repeated until all reachable states of the system are explored. If there are k components in the system, and if Σ_i denotes the set of state variables of the i^{th} component, the state variables of the overall system is given by $\Sigma = \cup_{i=1}^{k} \Sigma_i$. Even when $|\Sigma_i|$ is small for each i, $|\Sigma|$ can be large for large values of k. Since the complexity of reachability analysis grows exponentially with the number of state variables, reachability analysis of a large network of components is computationally far more difficult than searching the state spaces of individual components separately.

The additional computational effort needed to search the state space of a large network of components is primarily for reasoning about interactions between components. Interestingly, however, in a large class of practical systems, components primarily interact locally with a few other components in their "neighbourhood". More formally, state transitions of one component affect the state variables of only a few other components. This is not surprising since systems are often designed in a modular and compositional way, where individual components are required to interface and interact locally with a few other components in their spatial neighbourhood. While a few components may interact globally with other components, even these global interactions can often be modeled by synchronized local interactions, as we will see later in Section 2. Thus, interactions between components in a large class of practical systems are largely local in nature. This presents significant opportunities for optimization when performing reachability analysis. In this paper, we exploit these opportunities to design highly scalable Binary Decision Diagrams (BDD)-based [1] techniques for efficiently searching state spaces of large networks of state transition systems with local interactions.

If the behaviour of each component in a network is independent of that of others, the reachable state space of the overall system can be obtained by computing the Cartesian product of the reachable state spaces of individual components. Interactions between components however render such an analysis highly conservative in practice. Traditional symbolic reachability analyzers [4] therefore require the transition relations of individual components to be combined into a single system-wide transition relation involving all state variables in Σ. Since representing and manipulating BDDs with thousands of variables in the support set is computationally prohibitive even with state-of-the-art public-domain BDD packages like CUDD [8], BDD-based tools that work with system-wide transition relations do not scale well to large networks. To address this problem, earlier researchers have considered using partitions [3] and overlapping projections [6] of state variables. Govindaraju and Dill's approach [6] based on the *multiple constrain* operator is currently among the best BDD-based approaches for computing over-approximations of the reachable state space using *overlapping projections*. Yet other approaches [2, 7] have attempted to characterize BDDs on-the-fly during reachability analysis, and use appropriate approximations to achieve a trade-off between efficiency and accuracy. While these techniques have been used to efficiently compute good over-approximations of reachable state spaces of some large systems, they still require operations (e.g. the multiple constrain operation in [6]) on BDDs that have almost all variables in Σ in the support set. This makes it impractical to use these techniques for networks of transition systems with thousands of state variables. This difficulty was also observed during our experiments, where the technique of Govindaraju and Dill [6] could not compute an over-approximation of the reachable discrete-timed state space of an interconnection of timed logic gates having 6242 state variables in 60 minutes on a moderately powerful computing platform. In this work, we wish to address such scalability issues by designing approximate BDD-based reachability analysis techniques that scale to very large networks without compromising much on accuracy.

Existing techniques, including that of Govindaraju and Dill [6], use information about locality of interactions between different components to choose a good set of overlapping projections, but not to optimize the reachability analysis per se. In this paper, we wish to go a step further and exploit locality of interactions to optimize BDD-based reachability analysis using overlapping projections. While the method of Govindaraju and Dill provably gives the best over-approximation of the reachable state space using overlapping projections, we show that by exploiting locality of interactions during image computation, we can gain significantly in efficiency without suffering much precision-wise. Significantly, our technique permits scaling the analysis to networks of transition systems with variable counts at least an order of magnitude higher than those analyzable using Govindaraju and Dill's technique.

The remainder of the paper is organized as follows. Section 2 describes networks of state transition systems and presents a set of techniques to optimize reachability analysis of large networks using locality of interactions. Section 3 discusses experiments for evaluating the optimized analysis techniques, and compares the performance and accuracy of these techniques with each other as well as with Govindaraju and Dill's technique. Finally, Section 4 concludes the paper.

2 Networks of State Transition Systems

We represent a state transition system B as a $4-$tuple (Σ, Q, I, T), where Σ is a finite set of state variables, Q is the set of all states, $I : Q \rightarrow \{\mathsf{True}, \mathsf{False}\}$ is an initial state predicate, and $T : Q \times Q \rightarrow \{\mathsf{True}, \mathsf{False}\}$ is a transition relation predicate. Each state variable $s \in \Sigma$ has an associated *finite* domain $\mathcal{D}_s$, and Q is the Cartesian product of the finite domains corresponding to all variables in Σ. When describing a state transition, we use unprimed letters to refer to values of variables in the present state, and the corresponding primed letters to refer to their values in the next state. Let Σ' denote the set $\{s' \mid s \in \Sigma\}$. Thus, I is a predicate with free variables Σ, while T is a predicate with free variables $\Sigma \cup \Sigma'$. We will henceforth refer to these predicates as $I(\Sigma)$ and $T(\Sigma, \Sigma')$ respectively.

A *network* of state transition systems is a collection of state transition systems (with possibly overlapping sets of state variables), and a specification of synchronized transitions between them. Let $\mathcal{B} = \{B_1, \ldots B_m\}$ be a set of state transition systems that interact to form a network $\mathcal{N}$. Each B_i is a $4-$ tuple

$(\Sigma_i, Q_i, I_i, T_i)$ and is called a *component* of $\mathcal{N}$. Components B_i and B_j are said to execute state transitions *synchronously* iff every state transition of B_i occurs simultaneously with a state transition of B_j. Thus, state transitions of B_i and B_j cannot be interleaved. We specify synchronization between components by an undirected graph $\mathcal{H} = (\mathcal{B}, E_{\mathcal{H}})$, where $\mathcal{B}$ is the set of components and $(B_i, B_j) \in E_{\mathcal{H}}$ iff components B_i and B_j execute state transitions *synchronously.* It is easy to see that the binary relation on components defined by synchronous execution of transitions is an equivalence relation. Hence, the graph $\mathcal{H}$ consists of a set of disconnected cliques. If a clique consists of only a single component, we say that the corresponding component executes state transitions *asynchronously* with other components. Indeed, its state transitions can be interleaved arbitrarily with those of other components. The network $\mathcal{N}$ is formally defined by the tuple $(\mathcal{B}, \mathcal{H})$.

Given a network $\mathcal{N} = (\mathcal{B}, \mathcal{H})$, the overall state transition system corresponding to the network is given by $B_{\mathcal{N}} = (\Sigma_{\mathcal{N}}, Q_{\mathcal{N}}, I_{\mathcal{N}}, T_{\mathcal{N}})$, where $\Sigma_{\mathcal{N}} = \cup_{i=1}^{m} \Sigma_i$, $Q_{\mathcal{N}}$ is the Cartesian product of the finite domains corresponding to variables in $\Sigma_{\mathcal{N}}$, and $I_{\mathcal{N}}(\Sigma_{\mathcal{N}}) = \bigwedge_{i=1}^{m} I_i(\Sigma_i)$. In order to determine $T_{\mathcal{N}}(\Sigma_{\mathcal{N}}, \Sigma'_{\mathcal{N}})$, we need to model the effect of synchronous transitions of each clique in $\mathcal{H}$. Let $C = \{B_i, \ldots B_k\}$ be a clique in $\mathcal{H}$. Let $\Sigma_C = \Sigma_i \cup \ldots \Sigma_k$ and $\overline{\Sigma_C} = \Sigma_{\mathcal{N}} \setminus \Sigma_C$. We will henceforth use this notation to denote the set of variables corresponding to a collection of components, and the complement of a set of variables, respectively. We say that the network changes state from q to q' due to a synchronous transition of components in C iff (q, q') satisfies the predicate $\bigwedge_{B_i \in C} T_i(\Sigma_i, {\Sigma_i}')$ $\wedge$ $\bigwedge_{s \in \overline{\Sigma_C}} (s \leftrightarrow s')$. Let this predicate be called $\Upsilon_C(\Sigma_{\mathcal{N}}, \Sigma'_{\mathcal{N}})$. If $Cliques(\mathcal{H})$ denotes the set of cliques of $\mathcal{H}$, the transition predicate of the overall state transition system $B_{\mathcal{N}}$ is given by $T_{\mathcal{N}}(\Sigma_{\mathcal{N}}, \Sigma'_{\mathcal{N}}) = \bigvee_{C \in Cliques(\mathcal{H})} \Upsilon_C(\Sigma_{\mathcal{N}}, \Sigma'_{\mathcal{N}})$. For clarity of notation, we will henceforth omit the subscript $\mathcal{N}$ from Σ, Q, I and T whenever the network $\mathcal{N}$ is clear from the context.

2.1 Reachability analysis of networks of transition systems

Let $N2P$[3] be a predicate transformer that transforms a predicate $R(\Sigma, \Sigma')$ by replacing every occurrence of s' in R with the corresponding $s \in \Sigma$, for every $s' \in \Sigma'$. Formally,

$$N2P\left(R(\Sigma, \Sigma')\right) = \exists \Sigma' \left(R(\Sigma, \Sigma') \wedge \bigwedge_{s \in \Sigma} (s \leftrightarrow s') \right) \tag{1}$$

Now consider a predicate $R(\Sigma, \Sigma')$ that has at most one of s and s' (but not both) as free variable, for every $s \in \Sigma$. Such a predicate can be written as $R(\Sigma_*, \overline{\Sigma_*}')$, where Σ_* is the set of all variables $s \in \Sigma$ that appear as free variables of R, and $\overline{\Sigma_*} = \Sigma \setminus \Sigma_*$. The effect of transforming R by $N2P$ is given by $N2P(R(\Sigma_*, \overline{\Sigma_*}')) = \exists \Sigma' \left(R(\Sigma_*, \overline{\Sigma_*}') \wedge \bigwedge_{s \in \Sigma} (s \leftrightarrow s')\right) = R(\Sigma_*, \overline{\Sigma_*})$. Effectively, $N2P$ renames a subset of free variables of $R(\Sigma_*, \overline{\Sigma_*}')$. For predicates on

[3] $N2P$ stands for "next-to-present"

Boolean variables, such renaming can be efficiently performed in BDD packages like CUDD. For example, if the BDD for $R(\Sigma_*, \overline{\Sigma_*}')$ is given, the `bdd_permute` operation in CUDD achieves the effect of renaming variables. In the following discussion, whenever we apply $N2P$ to a predicate, the property that at most one of s and s' occurs as free variable, holds for the predicate. Therefore, asssuming that we are using a BDD package like CUDD that allows efficient renaming of variables, $N2P$ can be considered an efficiently computable operation.

Let $R(\Sigma)$ be the characteristic predicate of a set of states. Henceforth, we will refer to sets of states and their characteristic predicates interchangeably. The *image* of the set $R(\Sigma)$ under $T(\Sigma, \Sigma')$, denoted $Im(R(\Sigma), T(\Sigma, \Sigma'))$, can be obtained as $N2P(\exists\Sigma\,(R(\Sigma) \wedge T(\Sigma, \Sigma')))$. Given a network $\mathcal{N} = (\mathcal{B}, \mathcal{H})$ of state transition systems, the set of reachable states of $B_\mathcal{N}$ can be obtained by initializing the reachable set with the initial set of states, and by repeatedly computing the image of the current reachable set under $T(\Sigma, \Sigma')$ until no further new states are obtained. Since $T(\Sigma, \Sigma')$ is a disjunction of $\Upsilon_C(\Sigma, \Sigma')$ for $C \in Cliques(\mathcal{H})$, computing the image of a set of states $R(\Sigma)$ under $T(\Sigma, \Sigma')$ is equivalent to computing the image of $R(\Sigma)$ under each $\Upsilon_C(\Sigma, \Sigma')$ individually and then taking the union of the resulting sets of states. Note that in general, each $\Upsilon_C(\Sigma, \Sigma')$ has all variables in $\Sigma \cup \Sigma'$ as free variables. Since $|\Sigma|$ can indeed be large (several thousand variables) for a large network, computing the image of a set $R(\Sigma)$ under $\Upsilon_C(\Sigma, \Sigma')$ as $N2P(\exists\Sigma\,(R(\Sigma) \wedge \Upsilon_C(\Sigma, \Sigma')))$ does not scale well in BDD-based tools. However, there is an obvious optimization that can be done. On closer examination of the structure of $\Upsilon_C(\Sigma, \Sigma')$, i.e., $\bigwedge_{B_i \in C} T_i(\Sigma_i, {\Sigma_i}') \wedge \bigwedge_{s \in \overline{\Sigma_C}} (s \leftrightarrow s')$, we find that the values of all state variables in $\overline{\Sigma_C}$ are preserved in the next state. Since $\Sigma = \Sigma_C \cup \overline{\Sigma_C}$, we can write $R(\Sigma)$ as $R(\Sigma_C, \overline{\Sigma_C})$. Therefore, the image expression $N2P(\exists\Sigma\,(\bigwedge_{B_i \in C} T_i(\Sigma_i, \Sigma_i') \wedge \bigwedge_{s \in \overline{\Sigma_C}}\ (s \leftrightarrow s') \wedge R(\Sigma_C, \overline{\Sigma_C})))$ can be simplified to $N2P(\exists\Sigma_C\,(\bigwedge_{B_i \in C} T_i(\Sigma_i, \Sigma_i') \wedge R(\Sigma_C, \overline{\Sigma_C}')))$. Using the definiton of $N2P$ from equation (1), and noting that the quantification inside $N2P$ eliminates only variables in Σ_C, which is mutually disjoint with $\overline{\Sigma_C}$, we obtain the following equivalent expression for the image:

$$Im\,(R(\Sigma), \Upsilon_C(\Sigma, \Sigma')) = N2P\left(\exists\Sigma_C\left(\bigwedge_{B_i \in C} T_i(\Sigma_i, \Sigma_i') \wedge R(\Sigma_C, \overline{\Sigma_C})\right)\right) \quad (2)$$

Notice that the quantification in the final expression is over Σ_C which is potentially much smaller than Σ. Similarly, we have eliminated the potentially large conjunction $\bigwedge_{s \in \overline{\Sigma_C}} (s \leftrightarrow s')$ from the image computation step. In the following discussion, we will refer to this optimization as *"reducing non-transition variables"*. Unfortunately, even with this optimization, $R(\Sigma_C, \overline{\Sigma_C})$ involves all variables in Σ, and hence the scalability problem continues to exist.

To address this problem, we propose to exploit the fact that in a large class of practical systems, interactions between components are local in nature. Thus, state transitions of a component B_i change the state variables of only a small number of other components. Given a network $\mathcal{N} = (\mathcal{B}, \mathcal{H})$, we can capture this locality of interactions by an undirected graph $\mathcal{G} = (\mathcal{B}, E_G)$, where $\mathcal{B}$ is the set of components and $(B_i, B_j) \in E_G$ iff components B_i and B_j share some state variables, i.e., $\Sigma_i \cap \Sigma_j \neq \emptyset$. For every component B_i, we can then define its $k-$neighbourhood to be the set of all components that have a path of length at most k from B_i in $\mathcal{G}$. We denote this set as $B_i^{(k)}$. Formally, $B_i^{(0)} = \{B_i\}$ and $B_i^{(k)} = B_i \cup \{B_j^{(k-1)} \mid (B_i, B_j) \in E_G\}$ for all $i \geq 1$. A state transition of component B_i potentially changes some state variables of all components in $B_i^{(1)}$, but does not affect any state variable of any other component. Consequently, if C is a clique in the graph $\mathcal{H}$ representing synchronization between components, when computing the image of a set of states under the synchronized transition $\Upsilon_C(\Sigma, \Sigma')$, it is meaningful to update state variables of only components in $B_i^{(1)}$, where $B_i \in C$. This suggests that instead of considering state predicates on the entire set Σ of variables, it would be beneficial to consider state predicates on appropriately chosen subsets of Σ. In other words, reachability analysis using overlapping projections of states makes sense when analyzing large networks with local interactions. While the idea of using overlapping projections for approximate reachability was explored in detail by Govindaraju and Dill [6], their work considered reachability analysis of large sequential circuits instead of networks of small state transition systems. Consequently, they were unable to exploit locality of interactions to optimize the updation of various projections when a state transition happens. The primary contribution of this paper is to show that locality of interactions can indeed be exploited to significantly optimize updation of projections, enabling the design of BDD-based reachability analyzers that scale to much larger networks than those that can be handled by earlier techniques.

Let $\Pi_1, \ldots \Pi_p$ be subsets of state variables on which we choose to project the states of the overall system. Since we do not wish to ignore any state variable, we require that $\bigcup_{i=1}^{p} \Pi_p = \Sigma$. As in Govindaraju and Dill's approach, this gives rise to p projections, say $R_1(\Pi_1), \ldots, R_p(\Pi_p)$, of the set of reachable states. The collection of projections can be viewed as an abstraction of the actual reachable state set. The conjunction $\bigwedge_{i=1}^{p} R_i(\Pi_i)$ is the corresponding concretization that gives the best over-approximation of the reachable set from a given set of projections. However, computing the conjunction is computationally expensive in BDD-based tools since this involves all variables in Σ. Therefore, following Govindaraju and Dill's approach, we initialize the projections $R_1, \ldots, R_p$ with projections of the initial set of states on the sets of variables $\Pi_1, \ldots \Pi_p$, and update these projections each time the system executes a state transition. As discussed earlier, every transition of the network $\mathcal{N} = (\mathcal{B}, \mathcal{H})$ is a state transition of some clique $C \in \mathcal{H}$. Since a transition of C potentially changes all variables in Σ_C, every projection R_i such that $\Pi_i \cap \Sigma_C \neq \emptyset$ must be updated whenever a transition of C is taken. Conversely, all projections R_j such that $\Pi_j \cap \Sigma_C = \emptyset$ need not be updated when C transitions, since there is no component in C whose transitions change the values of variables in Π_j. Thus, by appropriately choosing $\Pi_1, \ldots \Pi_p$, it is possible to optimize the updation of projections every time a transition corresponding to a clique C is taken.

A transition of a clique C is basically a set of simultaneous transitions of all its components. Since the transition of an individual component B_i depends solely on the values of variables in Σ_i and potentially changes the values of only these variables, a good choice of $\Pi_1, \ldots \Pi_p$ is one where for each component B_i, there is at least one Π_j such that $\Sigma_i \subseteq \Pi_j$. Intuitively, such a choice would allow us to compute the effect of a transition of component B_i on the projection R_j with a high degree of accuracy. If $R_1(\Pi_1), \ldots R_p(\Pi_p)$ represent projections of the set of reachable states seen thus far, the image of $R_j(\Pi_j)$ under a synchronous transition of components in C can be computed as $\exists\overline{\Pi_j}\ (N2P(\exists\Sigma\ \Upsilon_C(\Sigma, \Sigma') \wedge \bigwedge_{i=1}^{p} R_i(\Pi_i)))$. Expanding $\Upsilon_C(\Sigma, \Sigma')$, we get

$$\exists\overline{\Pi_j}\left(N2P\left(\exists\Sigma \bigwedge_{B_i \in C} T_i(\Sigma_i, \Sigma_i') \wedge \bigwedge_{s \in \overline{\Sigma_C}} (s \leftrightarrow s') \wedge \bigwedge_{i=1}^{p} R_i(\Pi_i)\right)\right) \tag{3}$$

Coudert and Madre [5] have shown that the basic image computation in the above expression can be also be done using the constrain operator as follows.

$$\exists\overline{\Pi_j}\left(N2P\left(\exists\Sigma\left(\bigwedge_{B_i \in C} T_i(\Sigma_i, \Sigma_i') \wedge \bigwedge_{s \in \overline{\Sigma_C}} (s \leftrightarrow s')\right) \downarrow \bigwedge_{i=1}^{p} R_i(\Pi_i)\right)\right) \tag{4}$$

Since computing $\bigwedge_{i=1}^{p} R_i(\Pi_i)$ potentially involves all variables in Σ and is computationally expensive in BDD-based tools, Govindaraju and Dill proposed using a multiple constrain operator. This operator takes a Boolean predicate F and constrains it with a vector of predicates $\langle C_1, \ldots C_m\rangle$ iteratively, instead of conjoining C_1 through C_m first and then constraining F with the conjunction. Using this operator, the expression for the image can be written as

$$\exists\overline{\Pi_j}\left(N2P\left(\exists\Sigma\left(\bigwedge_{B_i \in C} T_i(\Sigma_i, \Sigma_i') \wedge \bigwedge_{s \in \overline{\Sigma_C}} (s \leftrightarrow s')\right) \downarrow_m \langle R_i(\Pi_i)\rangle\right)\right). \tag{5}$$

In the above expression, $\downarrow_m$ denotes the multiple constrain operator and $\langle R_i(\Pi_i)\rangle$ denotes the vector $\langle R_1(\Pi_1), \ldots R_p(\Pi_p)\rangle$ of all projections. Govindaraju and Dill showed experimentally that for a large class of sequential circuits, this significantly enhanced the efficiency of image computation compared to applying the single constrain operator as in expression (3). In the following discussion, we will refer to the technique of computing the image using the multiple constrain operator as the *"complete multiple constrain"* method.

Although updation using multiple constrain is more efficient that updation using single constrain, the computation of $\bigwedge_{B_i \in C} T_i(\Sigma_i, \Sigma_i') \wedge \bigwedge_{s \in \overline{\Sigma_C}} (s \leftrightarrow s')$ still involves all variables of Σ and can be computationally expensive. One might wonder if we can reduce non-transition variables, as discussed earlier, to simplify $\bigwedge_{B_i \in C} T_i(\Sigma_i, \Sigma_i') \wedge \bigwedge_{s \in \overline{\Sigma_C}} (s \leftrightarrow s')$ to $\bigwedge_{B_i \in C} T_i(\Sigma_i, \Sigma_i')$, and then apply the multiple constrain (or even Coudert and Madre's constrain) operator to the simplified predicate. Unfortunately, this is not possible in general. Indeed, reducing non-transition variables simplifies expression (3) to

$$\exists \overline{\Pi_j} \left(N2P \left(\exists \Sigma_C \bigwedge_{B_i \in C} T_i(\Sigma_i, \Sigma_i') \wedge \bigwedge_{i=1}^{p} R_i(\Pi_i) \right) \right) \tag{6}$$

While this expression is simpler than expression (3), we cannot directly apply Coudert and Madre's constrain operator and equate $(\exists \Sigma_C \bigwedge_{B_i \in C} T_i(\Sigma_i, \Sigma_i') \wedge \bigwedge_{i=1}^{p} R_i(\Pi_i))$ to $(\exists \Sigma_C\, (\bigwedge_{B_i \in C} T_i(\Sigma_i, \Sigma_i')) \downarrow \bigwedge_{i=1}^{p} R_i(\Pi_i))$. This is because the quantification in expression (6) is done over a subset of the variables in $\bigwedge_{i=1}^{p} R_i(\Pi_i)$. We will henceforth refer to the method of computing image using expression (6) as the *"reduced conjunction"* method. Note, however, that in expression (6), we must still compute $\bigwedge_{i=1}^{p} R_i(\Pi_i)$, which involves all variables in Σ. Hence the reduced conjunction method can be computationally expensive in BDD-based tools.

Fortunately, the reduced conjunction method can be improved further using locality of interactions. To exploit locality, we consider as many subsets of state variables as there are components in the network. For each component B_i, we choose a subset Π_i that includes state variables of all components in its $k-$ neighbourhood. Recall that when a component transitions, it affects the variables of only those components that are in its $1-$ neighbourhood. Thus $k = 1$ is a good initial choice of neighborhood for choosing subsets of state variables. As k increases, the subsets increase in size, and so does their overlap. Of course, if one subset is completely contained in another subset (as can happen for large values of k), only the larger subset is retained. As an extreme case, if d is the diameter of the graph $\mathcal{G} = (\mathcal{B}, E_G)$, and if we choose a neighbourhood of d, we obtain a single subset of state variables, $\Pi_1 = \Sigma$.

As the number of variables in each subset increases and as their overlap increases, the accuracy of computing projections of reachable states using expressions (3), (4), (5) or (6) is expected to increase. This is because large subsets with large overlaps can track correlations between state variables better than small subsets with small overlaps. However, having large subsets also entails increased computational effort in manipulating BDDs with large support sets when computing images of projections. By choosing an intermediate value of k, it is possible to ensure that the number of variables in each subset remains small, while there is adequate overlap between the subsets as well. While the best value of k might be domain dependent, in general, our experiments with discrete-timed circuits indicate that using small values like 1 or 2 often strikes a good balance between accuracy of image computations and computational efficiency.

2.2 Exploiting locality to optimize image computation

Let us now examine if computing the image of projection $R_j(\Pi_j)$ under a synchronous transition of components in a clique C can be simplified using locality of interactions. Expression (6) tells us how to compute the image. To simplify this expression, we first identify the set of transitioning components in C and the set of projections in $\{R_1, \ldots R_p\}$ that are intuitively "important" for computing the image of projection R_j. Expression (6) is then simplified by replacing transition relations of all other components and characteristic predicates of all other projections by True. This leads to an over-approximation of the image of projection $R_j(\Pi_j)$ due to a synchronous transition of components in C. We argue below that that this strategy allows us to scale reachability analysis to very large networks with local interactions, while remaining fairly accurate.

When a synchronous transition of components in clique C occurs, only those components that share state variable(s) with Π_j determine the image of projection $R_j(\Pi_j)$. Let the subset of components in C that share state variable(s) with Π_j be called D, and let Σ_D be the union of state variables of all components in D. Transitions of components in $C \setminus D$ neither read nor modify variables of Π_j, and hence do not affect the image of projection $R_j(\Pi_j)$ *directly*. Intuitively, the set of transitioning components D are "important" for determining the image of projection $R_j(\Pi_j)$ under a synchronous transition of components in C. Similarly, when a component $B_i \in D$ transitions, only those projections that share state variable(s) with B_i potentially constrain the transitions of B_i. Let the set of projections that share state variable(s) with at least one $B_i \in D$ be called $P1$, and let Σ_{P1} denote the union of all Π_k's, where projection R_k is in $P1$. Projections not in $P1$ cannot *directly* constrain transitions of B_i, since transitions of B_i are not guarded by conditions on state variables of these projections. Intuitively, projections in $P1$ are "important" for determining the synchronous transitions of components in D. If $|\Sigma_{P1}|$ is large, the set $P1$ can be further pruned by considering only those projections that share state variable(s) with Π_j and also with some $B_i \in D$. Let this set of projections be called $P2$ and let Σ_{P2} denote the corresponding set of state variables. Intuitively, projections in $P2$ not only directly constrain the transitions of components in D, but also allow projection $R_j(\Pi_j)$ to constrain transitions of components in D indirectly. Such an indirect constraining happens when the conjunction of $R_j(\Pi_j)$ and projections in $P2$ constrains transitions of components in D that would not have been constrained by $R_j(\Pi_j)$ alone. Hence, projections in $P2$ are very "important" for computing the image of $R_j(\Pi_j)$ under a synchronous transition of components in D. In the following discussion, we will use P to denote the set of "important" projections chosen for simplifying expression (6). While either $P1$ or $P2$ could be chosen for P, we have chosen $P2$ for our experiments since $|\Sigma_{P2}| \leq |\Sigma_{P1}|$.

We can now simplify expression (6) by over-approximating the transition relations of all components in $C \setminus D$ by True, and by over-approximating the conjunction of projections not in P by True. The simplified expression for the image of projection $R_j(\Pi_j)$ under a synchronous transition of components in C is then given by

$$\exists \overline{\Pi_j} \left(N2P \left(\exists \Sigma_D \bigwedge_{B_i \in D} T_i(\Sigma_i, \Sigma_i') \wedge \bigwedge_{\Pi_i \in P} R_i(\Pi_i) \right) \right) \tag{7}$$

Due to locality of interactions, the set P is much smaller than the entire set of projections, and similarly, D is a small subset of C, in general. This leads to significant gains in efficiency compared to computing the image of projection $R_j(\Pi_j)$ according to expression (6). One might suspect that this gain in efficiency is achieved at the cost of a significant loss of accuracy. However, as demonstrated by our experiments, the loss in accuracy due to these simplifications is not large, and the accuracy-efficiency trade-off is on the favourable side. We will call this technique that exploits locality of interactions and computes the image of projection $R_j(\Pi_j)$ according to expression (7) as *"partial reduced conjunction"*.

It is important to note that the proposed simplifications lead to over-approximations in the image of projection $R_j(\Pi_j)$ in the worst case. This happens when the complete conjunction $\bigwedge_{B_i \in C} T_i(\Sigma_i, \Sigma_i') \wedge \bigwedge_{i=1}^{p} R_i(\Pi_i)$ forbids a state transition by evaluating to False for a specific pair of present and next states, whereas the partial conjunction $\bigwedge_{B_i \in D} T_i(\Sigma_i, \Sigma_i') \wedge \bigwedge_{\Pi_i \in P} R_i(\Pi_i)$ evaluates to True for the same choice of present and next states.

Similar to the approach of Govindaraju and Dill, one might also consider optimizing the evaluation of expression (7) by using the multiple constrain operator. Let Σ_{PD} denote $\Sigma_P \cup \Sigma_D$ and $\Sigma_{P\setminus D}$ denote $\Sigma_P \setminus \Sigma_D$. The expression for the image of $R_j(\Pi_j)$ using the multiple constrain operator is then given by

$$\exists \overline{\Pi_j} \left(N2P \left(\exists \Sigma_{PD} \left(\bigwedge_{B_i \in D} T_i(\Sigma_i, \Sigma_i') \wedge \bigwedge_{s \in \Sigma_{P\setminus D}} (s \leftrightarrow s') \right) \downarrow_m \langle R_i(\Pi_i) \rangle_{\mathbf{P}} \right) \right). \tag{8}$$

In the above expression, $\downarrow_m$ denotes the multiple constrain operator and $\langle R_i(\Pi_i)\rangle_{\mathbf{P}}$ is a vector of projections belonging to the set P. Note that in order to apply the multiple constrain operator, we had to introduce the subexpression $\bigwedge_{s \in \Sigma_P \setminus \Sigma_D} (s \leftrightarrow s')$, which translates to increased computational cost in evaluating expression (8). As we will see in Section 3, experiments on a set of benchmarks indicate that this technique performs marginally worse than the partial reduced conjunction method. This shows that the benefits of the multiple constrain operator are more than offset by the additional computational cost of evaluating the subexpression $\bigwedge_{s \in \Sigma_P \setminus \Sigma_D} (s \leftrightarrow s')$. This technique of computing the image of projection $R_j(\Pi_j)$ using expression (8) will be called *"partial multiple constrain"* in our future discussion.

The set $\overline{\Pi_j}$ of variables that is finally quantified (corresponding to the leftmost existential quantifier) in expression (7) can be written as the union of $\overline{\Pi_j} \cap \overline{\Sigma_D}$ and $\overline{\Pi_j} \cap \Sigma_D$. Since the quantifier inside the $N2P$ operator eliminates only variables in Σ_D, all free variables of $\bigwedge_{B_i \in D} T_i(\Sigma_i, \Sigma_i') \wedge \bigwedge_{\Pi_i \in P} R_i(\Pi_i)$ that are in $\overline{\Pi_j} \cap \overline{\Sigma_D}$ are unaffected by this quantification. These variables, being present state variables, are not renamed by $N2P$ as well. Therefore, it is possible to push $\exists \overline{\Pi_j} \cap \overline{\Sigma_D}$ inside the $N2P$ and $\exists \Sigma_D$ operators in expression (7) to obtain the semantically equivalent expression

$$\exists(\overline{\Pi_j} \cap \Sigma_D)\left(N2P\left(\exists\Sigma_D\left(\bigwedge_{B_i \in D} T_i(\Sigma_i, \Sigma_i') \wedge \left(\exists(\overline{\Pi_j} \cap \overline{\Sigma_D}) \bigwedge_{\Pi_i \in P} R_i(\Pi_i)\right)\right)\right)\right) \tag{9}$$

In the above expression, the argument of $N2P$ is an expression whose free variables can be partitioned into the sets $(\Pi_j \cap \Sigma_D)'$, $(\overline{\Pi_j} \cap \Sigma_D)'$ and $(\Pi_j \cap \overline{\Sigma_D})$. Let these three mutually disjoint sets of variables be called Γ_1', Γ_2' and Γ_3, respectively, and let $\xi(\Gamma_1', \Gamma_2', \Gamma_3)$ denote the argument of $N2P$ in expression (9). Using the definition of $N2P$ from equation (1) and the notation introduced above, expression (9) can now be written as $\exists\Gamma_2\, \xi(\Gamma_1, \Gamma_2, \Gamma_3)$. Unfortunately, it is not straightforward to obtain $\xi(\Gamma_1, \Gamma_2, \Gamma_3)$. Specifically, we note from expression (9) that $\xi(\Gamma_1', \Gamma_2', \Gamma_3)$ is obtained by existentially quantifying variables in $\Gamma_1 \cup \Gamma_2$ from an expression, say ζ, with free variables $\Gamma_1 \cup \Gamma_2 \cup \Gamma_3 \cup \Gamma_1' \cup \Gamma_2'$. Therefore, simply substituting Γ_1 for Γ_1' and Γ_2 for Γ_2' in ζ and quantifying out $\Gamma_1 \cup \Gamma_2$ will not give $\xi(\Gamma_1, \Gamma_2, \Gamma_3)$. To overcome this problem, we use the fact that renaming bound variables does not change a quantified expression. Hence, the image expression $\exists\Gamma_2\, \xi(\Gamma_1, \Gamma_2, \Gamma_3)$ is equivalent to $\exists\Gamma_2'\, \xi(\Gamma_1, \Gamma_2', \Gamma_3)$, which in turn, is equivalent to $N2P(\exists\Gamma_2'\, \xi(\Gamma_1', \Gamma_2', \Gamma_3))$. Recalling that $\xi(\Gamma_1', \Gamma_2', \Gamma_3)$ is the argument of $N2P$ in expression (9), the following image expression, semantically equivalent to expression (9) but with different quantifier ordering, is obtained.

$$N2P\left(\exists\Sigma_D\left(\left(\exists(\overline{\Pi_j} \cap \Sigma_D)' \bigwedge_{B_i \in D} T_i(\Sigma_i, \Sigma_i')\right) \wedge \left(\exists(\overline{\Pi_j} \cap \overline{\Sigma_D}) \bigwedge_{\Pi_i \in P} R_i(\Pi_i)\right)\right)\right) \tag{10}$$

The evaluation of expression (10) can be further simplified, albeit with further loss of accuracy, by pushing the quantification of variables in $\overline{\Pi_j} \cap \overline{\Sigma_D}$ inside the conjunction $\bigwedge_{\Pi_i \in P} R_i(\Pi_i)$. Effectively, this amounts to over-approximating the existential projection of a conjunction by the conjunction of existential projections. As a first approximation, we observe that expression (10) computes the (approximate) image of projection $R_j(\Pi_j)$ due to a transition of components in clique C. Consequently, the most important variables in this expression are those in $\Sigma_D \cup \Pi_j$. Hence, the quantification of all other variables, i.e. variables in $\overline{\Pi_j} \cap \overline{\Sigma_D}$, may be pushed inside the conjunction to optimize the computation of expression (10). This gives rise to the optimized image expression

$$N2P\left(\exists\Sigma_D\left(\left(\exists(\overline{\Pi_j} \cap \Sigma_D)' \bigwedge_{B_i \in D} T_i(\Sigma_i, \Sigma_i')\right) \wedge \bigwedge_{\Pi_i \in P} \exists(\overline{\Pi_j} \cap \overline{\Sigma_D}) R_i(\Pi_i)\right)\right) \tag{11}$$

We will call image computation using expression (11) as *"partial approximate quantification"*. Note that in computing the image according to expression (11), the maximum number of variables involved in a BDD operation is $\max_{\Pi_i \in P}(|\Pi_i|)+$

$2.|\Sigma_D|$. This is far smaller than the number of variables involved in operations required for computing images by the earlier expressions. This contributes to the efficiency of using expression (11), which is also corroborated by our experiments. The loss of accuracy when computing the image according to expression (11) vis-a-vis when computing using expressions (4) or (5) or (6) stems from two sources: (i) approximating conjunctions by partial conjunctions, as in expression (7), and (ii) pushing quantifications inside conjunctions, as in expression (11). However, in both cases, the approximation is done with projections, components or variables that are intuitively less "important" in determining the image of projection $R_j(\Pi_j)$ under a transition of clique C. The variables, projections and transitions that are more "important" are not approximated as far as possible. Locality of interactions allows us to restrict this set of "important" variables, projections and transitions to a small set even in very large networks. This explains why our experiments indicate a high degree of efficiency, and a fair degree of accuracy when using expression (11).

It is easy to see from the nature of our approximations that in terms of accuracy of results, the ordering of the different methods is "complete multiple constrain" $\geq$ "partial reduced conjunction" = "partial multiple constrain" $\geq$ "partial approximate quantification". The degradation in accuracy from "complete multiple constrain" to "partial reduced conjunction" is due to the over-approximation of several "unimportant" transition relations and projections. The further degradation in accuracy in "partial approximate quantification" is attributable to the over-approximation of a projection of conjunctions by the conjunction of projections. Experimental results however show that the degradation in accuracy is not large for a range of timed circuit benchmarks. In terms of the number of variables involved in BDD operations when computing the imageo of a projection under a synchronized transition of components, it is clear from expressions (5), (8), (7) and (11) that the ordering of the different methods is "complete multiple constrain" $\geq$ "partial multiple constrain" $\geq$ "partial reduced conjunction" $\geq$ "partial approximate quantification". The variable count involved in BDD operations gives a rough indication of the relative computational time and memory requirements of BDD-based tools implementing these techniques. Our experimental results corroborate that this order is by and large correct.

2.3 Scalability issues

The technique of "partial approximate quantification" described above offers unique advantages in scaling our BDD-based approach to very large networks. We argue below that the maximum number of variables involved in any BDD operation during image computation using this technique is independent of the size of the transition system. Instead, it depends only on (i) the number of variables in each component, (ii) the degree of the graph $\mathcal{G} = (\mathcal{B}, E_G)$ that represents sharing of variables between components of the network, and (iii) the neighbourhood k used to determine projections. Thus, by carefully dumping BDDs to and from disk, it is never necessary to represent or manipulate BDDs with very large support sets in main memory. This can enable our BDD-based reachability analysis to scale up to very large networks with local interactions.

As has been described in Section 2.2, the number of variables involved in any image computation step of "partial approximate quantification" is bounded above by $\max_{\Pi_i \in P}(|\Pi_i|) + 2.|\Sigma_D|$. Here, $\max_{\Pi_i \in P}(|\Pi_i|)$ denotes the maximum number of variables in any projection, and depends on (i) the maximum degree of a node in the graph $\mathcal{G} = (\mathcal{B}, E_G)$, (ii) the neighbourhood k used to compute the projections, and (iii) the number of state variables in each component. The quantity $|\Sigma_D|$ depends on (i) the maximum number of components in a clique C that share variable(s) with the projection R_j being updated, and (ii) the number of state variables in a component. Given a network of transition systems, the maximum degree of a node in the graph $\mathcal{G}$ and the neighbourhood k used to compute projections uniquely define an upper bound on the maximum number of components in a clique C that potentially share variables with R_j. Interestingly, neither the degree of $\mathcal{G}$, nor the neighbourhood k, nor the maximum number of variables in a component depend on the total number of components in the network. Thus, if the maximum degree of $\mathcal{G}$ and the number of variables in each component is bounded by a small number, and if we choose a neighbourhood k for defining projections such that $\max_{\Pi_i \in P}(|\Pi_i|)+2.|\Sigma_D|$ is within the maximum variable count that a BDD-package can efficiently handle, it is possible to scale the analysis to very large networks. Of course, the number of projections and transition relations of components will increase with the size of the network. Hence the total number of BDDs that need to be stored will be large for large networks. However, as argued above, only a few of these, with a bounded number of variables in their support set, are required for image computation at any point of time. Hence by effectively dumping BDDs not currently needed to the disk and by re-loading them when needed, we envisage the possibility of BDD-based reachability analysis tools for arbitrarily large networks with local interactions. Such tools may require significant computational time for exploring the reachable state space of large networks, but will never run out of main memory due to BDD size explosion.

3 Experimental Results

In order to evaluate the effectiveness of our approach, we have implemented the strategies described in the previous section in the public domain reachability analysis engine `NuSMV`[4]. We have modified the reachability routine of `NuSMV` to perform reachability analysis on overlapping projections using asynchronous and synchronous transitions. We have used the resulting tool to explore the entire reachable space of a set of benchmarks, and report the reachable projections using our approach as well as using the multiple constrain approach of Govindaraju et al [6]. Our experimental results support our hypothesis that reachability analysis of large networks can be significantly optimized by exploiting locality of interactions.

Our benchmark suite consists of gate-level circuits with discrete-time delays. The motivation behind choosing these examples comes from their popularity in the domain of timed systems analysis and also their ease of scalability. Some of our examples consist of small combinational circuits, consisting of tens of gates, used in [9]. We also perform experiments on standard benchmark circuits from the `ISCAS-85` suite, which consists of larger combinational circuits.

3.1 Modeling of circuits

Every gate used in our experiments implements a combinational logic function and has an *inertial delay* and *bi-bounded pure delay*. The behaviour of a gate is modeled by having the following three parts: (i) a *Boolean logic block* that sets the Boolean value of the output to a function of the Boolean values of the inputs, (ii) the output of the logic block is fed to an *inertial delay element*, and (iii) the output of inertial delay element is fed to a *bi-bounded pure delay element*. We

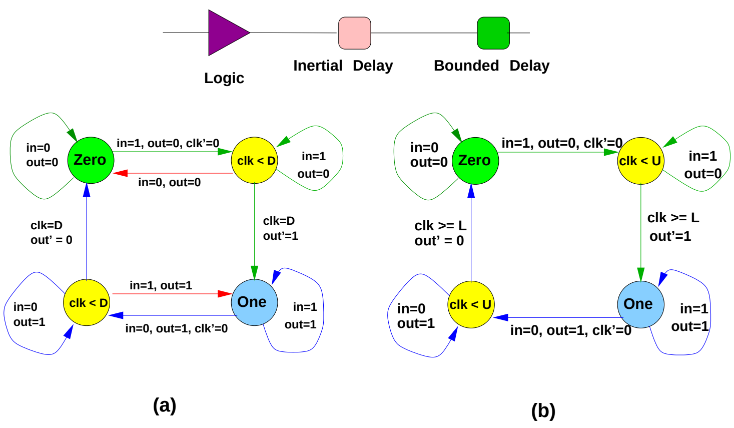

Fig. 1. (a) Inertial delay model (b) Bi-bounded pure delay model

model an inertial delay element having delay D by a timed automaton as shown in Fig. 1(a). Similarly, a bi-bounded pure delay element with lower and upper bounds l and u is modeled by a timed automaton as shown in Fig. 1(b). In these figures, `Zero` and `One` refer to stable states, where the Boolean values of *in* and *out* are the same.

Given the interconnection of gates representing a circuit, we compose the state transition behaviour of the logic block, inertial delay element and bi-bounded delay element of each gate to form a network of timed automata. To simplify the model, we assume that D, l and u are identical for all gates. To ensure that every pure delay element causes its output to change once between two consecutive changes at its input, we also assume that $u < D$. In fact, for all our experiments, the inertial delay (D) is set to 3, and the lower (l) and upper (u) bounds of bi-bounded pure delay elements are set to 1 and 2, respectively. When the output of a gate feeds the input of another gate, we ensure during composition that the corresponding output and input transitions occur simultaneously. Time is assumed to flow synchronously for all gates. For our experiments, the circuit inputs are modeled as signals that non-deterministically change their boolean values after a predefined delay $\Delta_{in} = 4$. We assume that time is discrete for all our experiments.

For an $n-$gate circuit modeled as above, the network of timed automata has two types of transitions: discrete (non-time-elapse) transitions for each logic function, inertial and bi-bounded delay element which execute asynchronously, and a global transition for synchronous advancement of time of all clocks. The global timed transition is modeled as the synchronous transition of a group of transition systems, where each transition system models advancement of time for one gate. It is easy to see that both the discrete and timed transitions corresponding to each gate affect the state variables of only those gates that are in its immediate fanout or fanin. By restricting the fanin and fanout of each gate in the circuit to a small number, we can therefore ensure that the locality of interactions is small and independent of the circuit size.

3.2 Comparing different techniques

We now present and compare experimental results obtained by application of the techniques referred to as "partial reduced conjunction", "partial approximate quantification", "partial multiple constrain", and "complete multiple constrain" in Section 2.1. We will call these techniques S_0, S_1, S_2 and S_3, respectively in the subsequent discussion. We compare the relative performance of these techniques in terms of BDD sizes, time taken and accuracy. All our experiments were performed on a 3 GHz Intel Pentium 686 processor with 1 GB of main memory, and running Fedora Core Linux 3.4.3-6.fc3.

In order to see how the various techniques scale up, we converted all large circuits (> 70 gates) to functionally equivalent circuits in which the fanout and fanin of each gate is bounded above by 3. As discussed in Section 2.3, this bounds the maximum degree of any node in the graph $\mathcal{G}$ and allows the "partial approximate quantification" technique to scale to large circuits. To reduce the fanout of a gate to 3, we used a linear chain of buffers at the output of the gate. To reduce the fanin of a gate, we used a tree of 2 input gates which combine to give the same Boolean function.

The specification for each circuit used in our experiments is given in Table 1. In this table, column *Ckt* gives the name of the circuit, and column *Gates* gives the number of gates, which is also the total number of discrete transitions for the circuit. Column *Additional gates* gives the number of additional gates that were added to obtain a functionally equivalent circuit with all gates having bounded fanin and bounded fanout, as discussed above. Columns *0-nbd, 1-nbd, 2-nbd* give the statistics for projections using neighbourhoods of 0, 1, and 2 respectively. Sub-column *Proj* gives the number of distinct projections obtained for each value of neighbourhood. A projection whose support set is a subset of the support set of another projection is discarded. Sub-column *Max* gives the maximum number of variables involved during any image computation. Finally, column *Variables* gives the total number of state variables of the network of transition systems. Note that this number is much larger than the total number of gates in the circuit, since encoding the behaviour of each gate requires several state variables. The plots in Table 2 show the time taken for searching the discrete-timed

Ckt	Gates	Additional gates	0-nbd		1-nbd		2-nbd		Variables
			Proj	Max	Proj	Max	Proj	Max	
2	10	0	17	38	13	78	9	123	274
3	11	0	19	59	17	115	13	143	306
6	21	0	29	59	23	115	18	157	626
7	31	0	39	59	33	115	29	151	946
17	11	0	17	54	11	131	8	138	274
74181	202	130	390	109	378	813	363	1782	6242
432	564	368	1092	108	1055	748	1013	1960	17574
499	724	481	1407	108	1345	711	1284	1864	22514
880	1182	739	2304	109	2191	811	2045	1883	36866

Table 1. Circuit characteristics

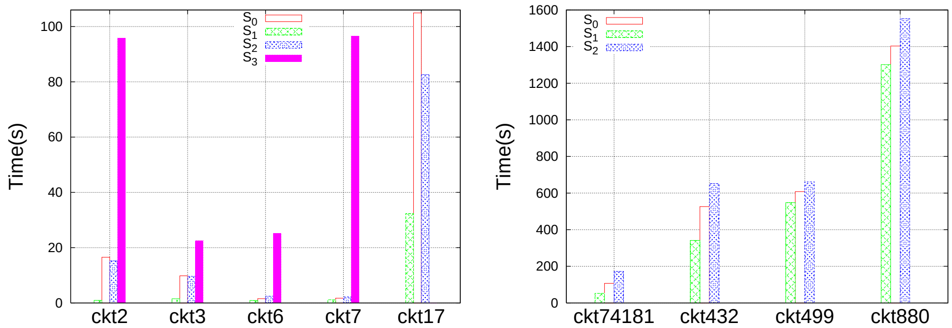

Table 2. Time plots for reachability analysis

reachable state space of each circuit using the techniques S_0, S_1, S_2 and S_3 referred to above. The results for small circuits and large circuits are given in two separate plots. The results presented in Table 2 were obtained with projections computed using a neighbourhood of 1. The maximum number of state variables of an individual component (discrete or timed transition system of a gate) is 28 in our experiments. In the plots of Table 2, the absence of a bar for a particular technique and circuit implies that for that circuit, reachability analysis using the specific technique did not terminate within 1 hour. We note that bars corresponding to “complete multiple constrain” are completely absent in the plots for all large circuits.

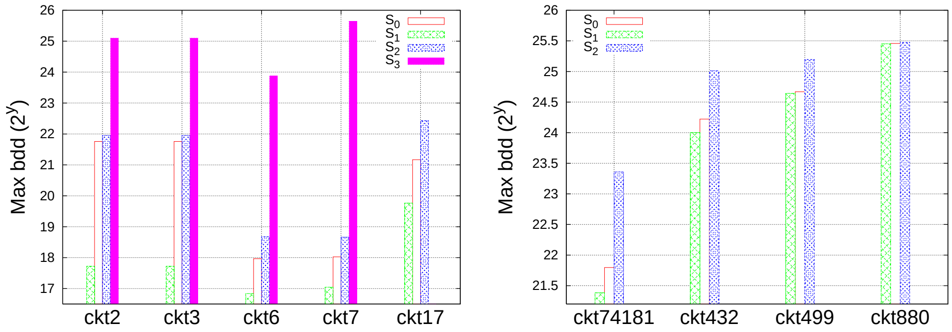

Table 3. Peak live BDD nodes for reachability analysis

The plots in Table 3 show on a $\log_2$ scale the peak live BDD node counts attained during reachability analysis using various techniques on the same circuits and using the same projections as in Table 2.

In order to compare the accuracy of the different techniques, we consider each circuit, and compute the ratio of the sizes of the projections of the reachable states using the different techniques. Ideally, we would have liked to compare the total count of reachable states for each circuit using various techniques. However, this requires conjoining the reachable state predicates for all the projections, and then counting the number of satisfying assignments for the resulting conjunction. Unfortunately, for large circuits, computing the conjunction of all projections involves constructing a BDD with almost all state variables of the entire network in its support set. This leads to a BDD size blowup and prevents us from computing the overall reachable state set. For each circuit, we therefore compute the ratios of sizes of projections of reachable sets using various techniques, and report the minimum, maximum and average values of these ratios considered over all projections. These ratios are summarized in Table 4. As discussed in Section 2.1, S_3 corresponding to “complete multiple constrain” gives the smallest sets of states for the projections, while “partial multiple constrain” (S_2) and “partial approximate quantification” (S_1) are comparable in accuracy to “partial reduced conjunction” (S_0). For the larger circuits, the “complete multiple constrain” technique does not terminate in 1 hour and hence the corresponding accuracy figures are not available. Note that the worst-case accuracy of technique S_0 (relative to technique S_3) is on ckt3, where it computes nearly 4 times the reachable states of a projection compared to what is computed by technique S_0. However, on an average, S_3 computes projections of reachable state sets that are 1.4 times larger than the projections computed by technique S_0 on ckt3.

Ckt	S_1/S_0			S_2/S_0			S_3/S_0		
	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min
ckt2	1	1	1	1	1	1	0.940254	1	0.72333
ckt3	1.00273	1.04646	1	1	1	1	0.721266	1	0.265875
ckt6	1	1	1	1	1	1	0.956729	1	0.834483
ckt7	1	1	1	1	1	1	0.969841	1	0.834483
ckt74181	1	1	1	1	1	1			
ckt432	1.00072	1.4	1	1	1	1			
ckt499	1	1	1	1	1	1			
ckt880	1	1	1	1	1	1			

Table 4. Accuracy comparison for reachability analysis

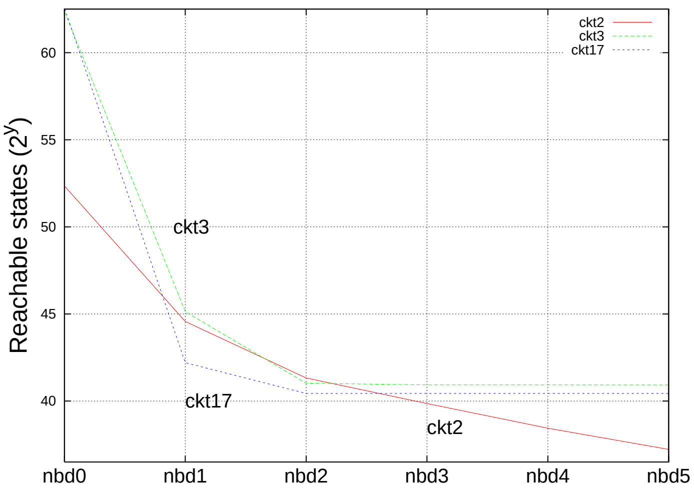

Fig. 2. Variation in over-approximation of reachable states with neighbourhood

In order to study the effect of increasing neighbourhoods when determining projections on the accuracy of reachability analysis, we carried out a set of experiments with small circuits in which we increased the neighbourhood from 0 to 5. The experiments were performed only on small circuits, since as explained above, it is difficult to conjoin the BDDs of all projections to obtain the complete set of reachable states for large circuits. Figure 2 shows a plot of the number of states in the over-approximation of the reachable set against the neighbourhood. The over-approximated reachable states in these plots were obtained using the technique S_0. The name of the circuit corresponding to each curve is written on top of the curve.

Increasing the neighbourhood when choosing projections results in larger projections with larger overlaps between them. The increased overlaps better facilitate tracking correlations between state variables, and leads to increased accuracy. This manifests itself as a monotone decrease in the size of the over-approximated reachable state set with increasing neighbourhood, as seen in Figure 2.

4 Discussion and Conclusion

From the bar charts in the previous section, we find that for small circuits, technique S_3 takes much more CPU time as compared to other techniques, and for larger circuits it does not terminate within 1 hour. This behaviour is expected as technique S_3 requires operating with BDDs that have the complete set of state variables in the support set for every image computation step. Indeed, the total count of such variables (ranging from 6000 to 30000 for our experimental circuits) is far beyond the reach of standard BDD packages like CUDD. In contrast, the other techniques require operating with BDDs that have much smaller support sets. This translates to reduced computation times and lower peak BDD sizes, when using these techniques. In addition, the accuracy of S_0 and S_1 do not suffer much compared to S_3, as measured in those cases where S_3 terminated. This gives concrete evidence in support of our claim that locality of interactions can be exploited to build efficient and scalable reachability analyzers, without compromising much on accuracy.

We also observe that for all circuits, technique S_1 gives the best results in terms of CPU time and peak BDD sizes. This can be qualitatively explained by the fact that the image computation step with this technique involves conjunction of BDDs with small support sets (10's of variables), followed by quantification of small sets of variables. In addition, technique S_1 retains the projections and transitions that are most "important" for computing the image of a projection under a transition, while ignoring the effects of other projections and transitions that do not directly affect the image of the projection under consideration. Since the accuracy obtained with this technique is fairly good, we conclude that exploiting locality of interactions in networks of transition systems has the potential to buy us a lot of efficiency without compromising much on accuracy.

Although in Section 2.1 we theoretically argued about the scalability of technique S_1 ("partial approximate quantification"), we were unable to analyze circuits with more than 1200 gates in the current set of experiments. With a fanin and fanout of 2 for each gate, even such large circuits are amenable to analysis using technique S_1, if BDDs can be dynamically swapped to and from disk during reachability analysis. The current implementation of our reachability analyzer does not implement such swapping of BDDs to and from disk, and consequently stores all BDDs in main memory. For large circuits, the total number of projections (and hence BDDs) becomes very large. While only a few of these are needed at any time for image computation, our current implementation suffers from memory bottlenecks since it stores all BDDs in main memory. We intend to implement swapping of BDDs to and from disk to allow scaling of our analysis to even larger circuits in the future.

The techniques presented in this paper effectively exploit locality of interactions in large networks of transition systems to scale reachability analysis to networks that are at least an order of magnitude larger than those amenable to existing tools. While other optimization techniques are being actively investigated for scaling reachability analysis, we believe locality of interactions is an important factor that, if properly exploited, can allow us to analyze very large networks of transition systems efficiently and fairly accurately. Our current work is a first step in this direction. We are also investigating ways to encode different search strategies in a meta-programming framework for reachability analysis that would allow one to mix and match different techniques for achieving a good balance of accuracy and efficiency.

References

1. R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.
2. G. Cabodi, P. Camurati, and S. Quer. Improving symbolic reachability analysis by means of activity profiles. *IEEE Transactions on Computers*, 19(9):1065–1075, 2000.
3. H. Cho, G. D. Hachtel, E. Macii, B. Plessier, and F. Somenzi. Algorithms for approximate FSM traversal based on state space decomposition. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 15(12):1465–1478, 1996.
4. A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV version 2: An opensource tool for symbolic model checking. In *Proceedings of CAV*, LNCS 2404, pages 359–364, 2002.
5. O. Coudert and J. C. Madre. A unified framework for the formal verification of sequential circuits. In *Proceedings of ICCAD*, pages 126–129, 1990.
6. Gaurishankar Govindaraju. *Approximate Symbolic Model Checking Using Overlapping Projections*. PhD thesis, Stanford University, August 2000.
7. I.-H. Moon, J. H. Kukula, K. Ravi, and F. Somenzi. To split or to conjoin: The question in image computation. In *Proceedings of DAC*, pages 23–28, 2000.
8. F. Somenzi. CUDD: Colorado University Decision Diagram Package Release 2.3.0., University of Colorado at Boulder, 1998.
9. D. Thomas, S. Chakraborty, and P. K. Pandya. Efficient guided symbolic reachability using reachability expressions. In *Proceedings of TACAS*, pages 120–134, 2006.