Reachability Analysis using Message Passing over Tree Decompositions.

Sriram Sankaranarayanan
University of Colorado, Boulder, CO.
Email: srirams@nameOfTheState.EDU

July 22, 2020

Abstract

In this paper, we study efficient approaches to reachability analysis for discrete-time nonlinear dynamical systems when the dependencies among the variables of the system have low treewidth. Reachability analysis over nonlinear dynamical systems asks if a given set of target states can be reached, starting from an initial set of states. This is solved by computing conservative over approximations of the reachable set using abstract domains to represent these approximations. However, most approaches must tradeoff the level of conservatism against the cost of performing analysis, especially when the number of system variables increases. This makes reachability analysis challenging for nonlinear systems with a large number of state variables. Our approach works by constructing a dependency graph among the variables of the system. The tree decomposition of this graph builds a tree wherein each node of the tree is labeled with subsets of the state variables of the system. Furthermore, the tree decomposition satisfies important structural properties. Using the tree decomposition, our approach abstracts a set of states of the high dimensional system into a tree of sets of lower dimensional projections of this state. We derive various properties of this abstract domain, including conditions under which the original high dimensional set can be fully recovered from its low dimensional projections. Next, we use ideas from message passing developed originally for belief propagation over Bayesian networks to perform reachability analysis over the full state space in an efficient manner. We illustrate our approach on some interesting nonlinear systems with low treewidth to demonstrate the advantages of our approach.

1 Introduction

Reachability analysis asks whether a target set of states is reachable over a finite or infinite time horizon, starting from an initial set for a dynamical system. This problem is fundamental to the verification of systems, and is known to be challenging for a wide variety of models. This includes cyber-physical systems, physical and biological processes. In this paper, we study reachability analysis algorithms for nonlinear, discrete-time dynamical systems. The key challenge in analyzing such systems arises from the difficulty of representing the reachable sets of these systems. As a result, we resort to over-approximations of reachable sets using tractable set representations such as intervals [16], ellipsoids, polyhedra [19], and low degree semi-algebraic sets [2]. Whereas these representations are useful for reachability analysis, they also trade off the degree of over-approximation in representing various sets against the complexity of performing operations such as intersections, unions, projections and image computations over these sets. The theory of abstract interpretation allows us to design various abstract domains that serve as representations for sets of

states in order explore these tradeoffs [18, 17, 34]. However, for nonlinear dynamical systems, these representations often become too conservative or too expensive as the number of state variables grow.

In this paper, we study reachability analysis using the idea of tree decompositions over the dependency graph of a dynamical system. Tree decompositions are a well-known idea from graph theory [37], used to study properties of various types of graphs. The treewidth of a graph is an intrinsic property of a graph that relates to how "far away" a given graph is from a tree. For instance, trees are defined to have a treewidth of 1. Many commonly occurring families of graphs such as *series-parallel graphs* have treewidth 2 and so on. Formally, a tree decomposition of a graph is a tree whose nodes are associated with subsets of vertices of the original graph along with some key conditions that will be described in Section 2. We use tree decompositions to build an abstract domain. The abstraction operation projects a set of states in the full system state space along each of the nodes of the tree, yielding various projections of this set. The concretization combines projections back into the high dimensional set. We study various properties of this abstract domain. First, we characterize abstract elements that can potentially be generated by projecting some concrete elements along the nodes of the tree (so called *canonical* elements, Def. 3.7). Next we characterize those sets which can be abstracted along the tree decomposition and reconstructed without any loss in information (tree decomposable sets, Def. 3.8). In this process, we also derive a *message passing* approach wherein nodes of the tree can exchange information to help refine sets of states in a sound manner. However, as we will demonstrate, the abstraction is "lossy" in general since projections of tree decomposable sets are not necessarily tree decomposable. We discuss some interesting ways in which precision can be regained by carefully analyzing this situation.

We combine these ideas together into an approach for reachability analysis of nonlinear systems using a grid domain that represents complex non convex sets as a union of fixed size cells using a gridding of the state-space. Although such a domain would be prohibitively expensive, we show that the tree decomposition abstract domain can drastically cut down on the complexity of computing reachable set overapproximations in this domain, yielding precise reachable set estimation for some nonlinear systems with low treewidth. We demonstrate our approach using a prototype implementation to show that for a restricted class of systems whose dependency graphs have low treewidth, our approach can be quite efficient and precise at the same time. Although some interesting systems have low treewidth property, it is easy to see that many systems will have treewidths that are too high for our approach. Our future work will consider how systems whose dependency graphs do not have sufficiently low treewidth can still be tackled in a conservative manner using some ideas from this paper.

1.1 Related Work

As mentioned earlier, the concept of tree decompositions and treewidth originated in graph theory [37]. The concept of treewidth gained popularity when it was shown that many NP-complete problems on graphs such as graph coloring could be solved efficiently for graphs with small treewidths [5]. Courcelle showed that the problem of checking if a given graph satisfies a formula in the monadic second order logic of graphs can be solved in linear time on graphs with bounded treewidth [15]. Several NP-complete problems such as 3-coloring can be expressed in this logic. Tree decompositions are also used to solve inference problems over Bayesian networks leading to representations of the Bayesian networks such as junction trees that share many of the properties of a tree decomposition [29]. In fact, belief propagation over junction trees is performed by passing messages that marginalize the probability distributions at various nodes of the tree. This is analogous to the message passing approach described here.

Tree decomposition techniques have been applied to model checking problems over finite state systems. For instance, Obdržálek show that the μ-calculus model checking problem can be solved in linear time in the size of a finite-state system whose graph has a bounded treewidth [35]. However, as Ferrara et al point out, requiring the state graph of a system to have a bounded treewidth is often restrictive [24]. Instead, they study concurrent finite state systems wherein the communication graph has a bounded tree width. However, they conclude that while it is more reasonable to assume that the communication graph has a bounded tree width, it does not confer much advantages to verification problems. For instance, they show that the unrolling of these systems over time potentially results in unbounded treewidth. In this paper, we consider a different approach wherein we study the treewidth of dependency graphs of the system. We find that many systems have small treewidth and exploit this property. At the same time, we note that some of the benchmarks studied have "sparse" dependency graphs but treewidths that are too large for our approach.

Tree decomposition techniques have also been studied in static analysis of programs. The control and data flow graphs of structured programs without goto-statements or exceptional control flow are known to have small treewidth that can be exploited to perform compiler optimizations such as register allocation quite efficiently [38]. Chatterjee et al have shown how to exploit small treewidth property of the control flow graphs of procedures in programs to perform interprocedural dataflow analysis by modeling the execution of programs with procedures as recursive state machines [11]. However, this approach seems restricted to control dominated properties such as sequence of function calls. In a followup work, they study control and data flow analysis problems for concurrent systems, wherein each component has constant treewidth [10]. In contrast, our approach studies dynamical system and consider tree decompositions of the data dependency graph.

The use of message passing in this paper closely resembles past work by Gulwani and Jojic [27]. Therein, a program verification problem involving the verification pre/post and intermediate assertions in a program is solved by passing messages that can propagate information between assertions along program paths in a randomized fashion. The approach is shown to be similar to loopy belief propagation used in Bayesian inference. The key differences are (a) we use data dependencies and tree decompositions rather than control flow paths to pass information along; and (b) we formally prove properties of the message passing algorithm.

Our approach is conceptually related to a well-known idea of speeding up static analysis of large programs using "packing" of program variables [28, 4]. This approach was used successfully in the Astreé static analyzer [4, 3, 21]. Therein, clusters of variables representing small sets of dependent local and global are extracted. The remaining program variables are abstracted away and the abstract interpretation process is carried out over just these variables. The usefulness of this approach has borne out in other abstract interpretation efforts, including Varvel [28]. The key idea in this paper can be seen as a formalization of the rather informal "clustering" approach using tree decompositions. We demonstrate theoretical properties as well as the ability to pass messages to improve the results of the abstract interpretation.

The use of the dependency graph structure to speed up reachability analysis approaches has been explored in the past for speeding up Hamilton-Jacobi-based approaches by Mo Chen et al [12] as well as flowpipe based approaches by Xin Chen et al [13]. Both approaches consider the directed dependency graph wherein x_i is connected to x_j if the former appears in the dynamical update equation of the latter variable. The approaches perform a strongly connected component (SCC) decomposition and analyze each SCC in a topological sorted order. However, this approach breaks as soon as the system has large SCCs, which is common. As a result, Xin Chen et al show how SCCs can themselves be broken into numerous subsets at the cost of a more conservative solution. In contrast, the tree decomposition approach can be applied to exploit sparsity even when the entire dependency graph is a single SCC.

2 Preliminaries

In this section, we will describe the system model under analysis, the dependency graph structure and the basics of tree decompositions. Let $X : \{x_1, \ldots, x_n\}$ be a set of *system variables* and $\sigma : X \mapsto \mathbb{R}$ represent a valuation to these system variables. Let D be the domain of all valuations of X, that describes the *state space* of the system. Also, let $W : \{w_1, \ldots, w_m\}$ represent disturbance variables and $\mathbf{w} : W \mapsto \mathbb{R}$ represent a vector of $m \geq 0$ external disturbance inputs that take values in some compact disturbance space $\mathcal{W}$.

Definition 2.1 (Dynamical Model)**.** A model Π is a tuple $\langle X, W, D, \mathcal{W}, f, X_0, U \rangle$, wherein $X, W, D, \mathcal{W}$ are as defined above, f is an arithmetic expression over variables in X, W describing the dynamics, X_0 is a set of possible initial valuations (states) and U is a designated set of unsafe states.

The dynamics are given by $\sigma' = \mathsf{eval}(f, \sigma, \mathbf{w})$, wherein eval evaluates a given an expression f, a set of valuations to the system variables $\sigma \in D$ and disturbances $\mathbf{w} \in \mathcal{W}$, and returns a new set of valuations for each variable in X, denoted by σ'.

For simplicity, we write $f(\sigma, \mathbf{w})$ to denote $\mathsf{eval}(f, \sigma, \mathbf{w})$ for a function expression f. As mentioned earlier, a state of the system is a valuation $\sigma : X \mapsto \mathbb{R}$ such that $\mathbf{x} \in D$. Given a finite sequence of disturbance inputs $\mathbf{w}_0, \ldots, \mathbf{w}_T$, for some $T \geq 0$ and $\mathbf{w}_i \in \mathcal{W}$ for all $i \in [0, T]$, an execution of the system is a sequence of states $\sigma_0, \ldots, \sigma_{T+1}$, such that: (a) $\sigma_0 \in X_0$, (b) $\sigma_t \in D$ for $t \in [0, T+1]$ and (c) $\sigma_{t+1} = f(\sigma_t, \mathbf{w}_t)$ for all $t \in [0, T]$. According to these semantics, the system may fail to have an execution for a given disturbance sequence $\mathbf{w}_t,\ t \in [0, T]$ and initial state σ_0 if for some state σ_t, we have $f(\sigma_t, \mathbf{w}_t) \notin D$.

A state σ_t is reachable (at time t) if there is an execution of the form $\sigma_0, \ldots, \sigma_t$, satisfying the conditions (a) - (c) above. We say that the unsafe state U is reachable iff some state $\sigma \in U$ is reachable. Furthermore, we say that U is reachable within a finite time horizon T, iff some state $\sigma \in U$ is reachable at time $t \in [0, T]$.

Example 2.1. Consider a nonlinear example of a dynamical model Π with state variables $X = \{x_1, x_2, x_3\}$ and $W = \{w_1\}$. The dynamics can be written as parallel assignments to the state variables:

$$x_1 \ := \ x_1 + 0.25x_2 - 0.05x_1 sin(x_2),\ x_2 \ := \ x_2 + w_1,\ x_3 \ := x_3 - 0.2x_3x_2\,,$$

The assignments are all evaluated in parallel to update the current state σ_t to a new state σ_{t+1}. The domain D is described by the constraints $\bigwedge_{i=1}^{3} x_i \in [-3, 3]$, and the disturbance $w_1 \in [-0.1, 0.1]$. The initial set X_0 is given by the constraints $x_1 \in [-0.2, 0.2] \ \wedge \ x_2 \in [-0.3, 0] \ \wedge \ x_3 \in [0, 0.4]$.

We will now define the dependency (hyper)graph of the system Π. For convenience, we write the update function (expression) f of a system Π in terms of individual updates $(f_1, \ldots, f_n)$, wherein $\sigma'(x_j) = f_j(\sigma, \mathbf{w})$. We say that system variable x_i (or disturbance variable w_j) is a *proper input* to the expression f_k if x_i (or w_j) occurs as a subterm in f_k. Let $\mathsf{inps}(f_k)$ denote the set of all proper input variables to the function (expression) f_k.

As an example, consider $X = \{x_1, \ldots, x_4\}$ and $W = \{w_1, w_2\}$ and the expression $f : x_1x_4 - w_1$. The proper inputs to f are $\{x_1, x_4, w_1\}$. We exclude cases such as $g : \frac{\sin^2(x_1)+\cos^2(x_1)}{\sin^2(x_2)+\cos^2(x_2)}$ that has $\{x_1, x_2\}$ as proper inputs. However a simplification using elementary trigonometric rules can eliminate them. We will assume that all expressions are simplified to involve the least number of variables.

Definition 2.2 (Dependency Hypergraph)**.** A dependency hypergraph of a system Π has vertices $V: X \cup W$, given by the union of the system and disturbance variables with hyperedge set $E \subseteq 2^V$ given by $E = \{e_1, \ldots, e_n\}$, wherein for each update $x_k := f_k(\mathbf{x}, \mathbf{w})$ $(k = 1, \ldots, n)$, we have the hyperedge $e_k : \{x_k\} \cup \mathsf{inps}(f_k)$. In other words, each update $x_k := f_k(\mathbf{x}, \mathbf{w})$ yields an edge that includes x_k along with all the system/disturbance variables that are proper inputs to f_k.

Example 2.2. The dependency hypergraph for the system from Example 2.1 has the vertices $V : \{x_1, x_2, x_3, w_1\}$ and the edges $\{e_1 : \{x_1, x_2\},\ e_2 : \{x_2, w_1\}$ and $e_3 : \{x_2, x_3\}\}$.

2.1 Tree Decomposition

We will now discuss tree decompositions and the associated concept of treewidth of a hypergraph $G : (V, E)$. The tree decomposition will be applied to the dependency hypergraphs (Def. 2.2) for systems Π (Def. 2.1).

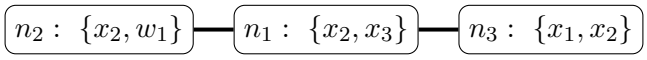
Definition 2.3 (Tree Decomposition and Treewidth)**.** Given a hypergraph $G : (V, E)$, a tree decomposition is a tree $T : (N, C)$ and a mapping $\textsc{verts} : N \mapsto 2^V$, wherein N is the set of tree nodes, C is the set of tree edges and $\textsc{verts}(\cdot)$ associates each node $u \in N$ with a set of graph vertices $\textsc{verts}(n) \subseteq V$. The tree decomposition satisfies the following conditions:

1. For vertex $v \in V$ there exists (at least one) $n \in N$ such that $v \in \textsc{verts}(n)$.
2. For each hyperedge $e \in E$ there exists (at least one) $n \in N$: $e \subseteq \textsc{verts}(n)$.
3. For each vertex v, for any two nodes n_1, n_2 such that $v \in \textsc{verts}(n_1)$ and $v \in \textsc{verts}(n_2)$, then $v \in \textsc{verts}(n)$ for each node n along the unique path between n_1 and n_2 in the tree. Stated another way, the subset of nodes $N_v : \{n \in N \mid v \in \textsc{verts}(n)\}$ induces a subtree of T (denoted T_v). This property is commonly called the *running intersection* property.

The width of a tree decomposition is given by $\max\{|\textsc{verts}(n)| \mid n \in N\} - 1$. In other words, we find the node n in the tree whose associated set of vertices has the largest cardinality. We subtract one from this maximal cardinality to obtain the treewidth. A tree decomposition is optimal for a graph G if no other tree decomposition exists with a strictly smaller width. The treewidth of a hypergraph G is given by width of an optimal tree decomposition.

It is easy to show that if the graph G is a tree, it has treewidth 1. Likewise, a cycle has tree width 2.

Example 2.3. The tree decomposition of the hypergraph G from Ex. 2.2 has three nodes $\{n_1, n_2, n_3\}$ with edges (n_1, n_2) and (n_2, n_3). The nodes along with the associated vertex sets are as follows:



Although the tree decomposition is not a rooted tree, we often designate an arbitrary node $r \in N$ as the root node, and consider the tree T as a rooted tree with root r.

Finding a Tree Decomposition: Interestingly, the problem of finding the treewidth of a graph is itself a NP-hard problem. However, many practical approaches exist for graphs with small treewidths. For instance, Bodlaender presents an algorithm that runs in time $O(k^{O(k^3)})$ to construct a tree decomposition of width at most k or conclude that the treewidth of the graph is at least $k + 1$ [6]. Such an approach can be quite useful if a given graph is suspected to have a small tree width in the first place. Besides this, many efficient algorithms exist to approximate the treewidth of a graph to some constant factor. A detailed survey of these results is available elsewhere [7, 8]. Open-source packages such as HTD can compute treewidth for graphs with thousands of nodes [1].

Finally, we note that if a tree decomposition of width k can be found, then one can be found with at most $|V|$ nodes.

Lemma 2.1. *Let T be a tree decomposition for a (multi)graph G with vertices V and treewidth k. There exists a tree decomposition $\hat{T}$ of G with the same treewidth k, and at most $|V|$ nodes.*

Proof. Let $T : (N, C)$ be a tree decomposition of tree width k. For each edge $(n_1, n_2) \in C$ such that $\textsc{verts}(n_1) \subseteq \textsc{verts}(n_2)$, merge n_1, n_2 into a single node $\hat{n}$ with $\textsc{verts}(\hat{n}) = \textsc{verts}(n_2)$. We verify that the resulting tree is still a valid tree decomposition of G with treewidth k. We repeat this process until no more edges exist with this property. Let $\hat{T}$ be the resulting tree at the end of the process: it continues to be a tree decomposition of G with treewidth k. As a result, for every edge (n_1, n_2) in $\hat{T}$, we have that $\textsc{verts}(n_1) \setminus \textsc{verts}(n_2)$ is nonempty. It remains to prove that $\hat{T}$ has at most as many nodes as the number of vertices in graph G.

Let us designate a node r as the root of $\hat{T}$ and consider it as a rooted tree. We will construct a one-to-one mapping β from each node n of $\hat{T}$ into a vertex G. To start with, arbitrarily select a vertex $\beta(r) \in \textsc{verts}(r)$. For each node n with parent $\hat{n}$, we select a $\beta(n) \in \textsc{verts}(n) \setminus \textsc{verts}(\hat{n})$. Such a selection can always be made due to the construction of $\hat{T}$. Now, we show that β is a one to one mapping. If not, let $\beta(n) = \beta(m) = v$ for some two nodes (m, n). By construction of β, we know that (m, n) cannot be an edge in the tree. Thus, consider the path in $\hat{T}$ from m to n. Let $\hat{m} \neq n$ be the first node along that path. We know that $v \in \textsc{verts}(m)$ and $v \in \textsc{verts}(n)$. However, we also know that $v \notin \textsc{verts}(\hat{m})$. This directly contradicts the fact that $\hat{T}$ is a tree decomposition. Thus β is one to one. Therefore, the number of nodes in $\hat{T}$ is atmost $|V|$. □

3 Abstract Domains Using Tree Decompositions

In this section, we will define abstract domains using tree decompositions of the dependency hypergraph of the system under analysis. Let Π be a transition system over system variables X. The concrete states are given by $\sigma \in D$, wherein $\sigma : X \mapsto \mathbb{R}$ maps each state variable $x_j \in X$ to its value $\sigma(x_j)$.

Definition 3.1 (Projections)**.** The projection of a state σ to a subset of state variables $J \subseteq X$, denoted as $\mathsf{proj}(\sigma, J)$, is a valuation $\hat{\sigma} : J \mapsto \mathbb{R}$ such that $\hat{\sigma}(x_i) = \sigma(x_i)$ for all $x_i \in J$. For a set of states $S \subseteq D$ and a subset of state variables $J \subseteq X$, we denote the projection of S along (the dimensions of) J as $\mathsf{proj}(S, J) : \{\mathsf{proj}(\sigma, J) \mid \sigma \in S\}$.

Definition 3.2 (Extensions)**.** Let R be a set of states involving just the variables in the set $J_1 \subseteq X$, i.e, $R \subseteq \mathsf{proj}(D, J_1)$. We define the *extension* of R into a set of variables $J_2 \supseteq J_1$ as $\mathsf{ext}_{J_2}(R) : \{\sigma \in \mathsf{proj}(D, J_2) \mid \mathsf{proj}(\sigma, J_1) \in R\}$.

In other words, the extension of a set embeds each element in the larger dimensional space defined by J_2 allowing "all possible values" for the dimensions in $J_2 \setminus J_1$.

We will use the notation $\mathsf{ext}(S)$ to denote the set $\mathsf{ext}_X(S)$, i.e, its extension to the entire set of state variables X. For a state σ_S, we will use $\mathsf{ext}(\sigma_S)$ denote $\mathsf{ext}(\{\sigma_S\})$.

Definition 3.3 (Product (Join) of Sets)**.** Let $R_1 \subseteq \mathsf{proj}(D, J_1)$ and $R_2 \subseteq \mathsf{proj}(D, J_2)$. We define $R_1 \otimes R_2 : \{\sigma : J_1 \cup J_2 \mapsto \mathbb{R} \mid \mathsf{proj}(\sigma, J_1) \in R_1 \text{ and } \mathsf{proj}(\sigma, J_2) \in R_2\}$.

Let $T : (N, C)$ be a tree decomposition of the dependency hypergraph of the system. Recall that for each node $n \in N$ we associate a set of system/disturbance variables denoted by $\textsc{verts}(n)$. Let $\textsc{verts}_X(n)$ denote the set of system variables: $\textsc{verts}(n) \cap X$. We say that an update function $x_k := f_k(\sigma, \mathbf{w})$ is associated with a node n in the tree iff $\{x_k\} \cup \mathsf{inps}(f_k) \subseteq \textsc{verts}(n)$.

Lemma 3.1. *For every system variable x_k, its update $x_k := f_k(\sigma, \mathbf{w})$ is associated with at least one node $n \in N$.*

Proof. This follows from those of a tree decomposition that states that every hyperedge in the dependency hypergraph must belong to $\textsc{verts}(n)$ for at least one node $n \in N$. □

3.1 Abstraction and Concretization

We consider subsets of the concrete states for the system Π, i.e, the set 2^D, ordered by set inclusion as our *concrete domain*. Given a tree decomposition, T, we define an abstract domain through projection of a concrete set along $\textsc{verts}(n)$ for each node n of T. For technical reasons, we will assume that the domain D is a cartesian product $D := D_1 \otimes \cdots \otimes D_n$ such that $\sigma \in D$ iff $\sigma(x_j) \in D_j$. While this condition can be relaxed considerably, we note that the assumption above is sufficient for most systems we will work with, in practice.

Definition 3.4 (Abstract Domain)**.** Each element s of the abstract domain $\mathbb{A}_T$ is a mapping that associates each node $n \in N$ with a set $s(n) \subseteq \mathsf{proj}(D, \textsc{verts}_X(n))$.

For $s_1, s_2 \in \mathbb{A}_T$, $s_1 \sqsubseteq s_2$ iff $s_1(n) \subseteq s_2(n)$ for each $n \in N$.

We will use the notation $\mathsf{proj}(S, n)$ for a node $n \in N$ to denote $\mathsf{proj}(S, \textsc{verts}_X(n))$.

Definition 3.5 (Abstraction Map)**.** Given a tree decomposition T, the abstraction map α_T takes a set of states $S \subseteq D$ and produces a mapping that associates tree node $n \in N$ to a projection of S along the variables $\textsc{verts}_X(n)$. Formally,

$$\alpha_T(S) : \ \lambda n : N.\ \mathsf{proj}(S, n)\,.$$

Thus, an abstract state s is a map that associates each node n of the tree to a set $s(n) \subseteq D_n$. We now define the concretization map γ_T.

Definition 3.6 (Concretization Map)**.** The concretization $\gamma_T(s)$ of an abstract state is defined as $\gamma_T(s) : \bigcap_{n \in N} \mathsf{ext}(s(n))$. In other words, we take $s(n)$ for every node $n \in N$, extend it to the full dimensional space of all system variables and intersect the result over all nodes $n \in N$.

Example 3.1. Consider a simple tree decomposition T with 2 nodes n_1, n_2 and a single edge (n_1, n_2). Let $\textsc{verts}(n_1) : \{x_1, x_2\}$ and $\textsc{verts}(n_2) : \{x_2, x_3\}$. Let the domain D be the set $\{\sigma \mid \sigma(x_i) \in \{1, 2, 3\}\}$ for $i = 1, 2, 3$. We use the notation $(\overset{x_1}{v_1}, \overset{x_2}{v_2}, \overset{x_3}{v_3})$ to denote a state σ that maps x_1 to the value v_1, x_2 to the value v_2 and so on.

Now consider the set $S = \{(\overset{x_1}{1}, \overset{x_2}{1}, \overset{x_3}{1}), (\overset{x_1}{1}, \overset{x_2}{1}, \overset{x_3}{2}), (\overset{x_1}{1}, \overset{x_2}{2}, \overset{x_3}{3})\}$. We have that $s : \alpha(S)$ is the mapping that projects S onto the dimensions (x_1, x_2) for node n_1 and (x_2, x_3) for node n_2:

$$n_1 \mapsto \{(\overset{x_1}{1}, \overset{x_2}{1}), (\overset{x_1}{1}, \overset{x_2}{2})\},\ n_2 \mapsto \{(\overset{x_2}{1}, \overset{x_3}{1}), (\overset{x_2}{1}, \overset{x_3}{2}), (\overset{x_2}{2}, \overset{x_3}{3})\}\,.$$

Likewise, we verify that the concretization map $\gamma(s)$ will yields us:

$$\gamma(s) : \{(\overset{x_1}{1}, \overset{x_2}{1}, \overset{x_3}{1}), (\overset{x_1}{1}, \overset{x_2}{1}, \overset{x_3}{2}), (\overset{x_1}{1}, \overset{x_2}{2}, \overset{x_3}{3})\}\,.$$

For convenience, if the tree T is clear from the context, we will drop the subscripts to simply write α and γ for the abstraction and concretization map, respectively.

Theorem 3.2. *For any tree decomposition T, the maps α and γ form a Galois connection. I.e, for all $S \subseteq D$ and $s \in \mathbb{A}_T$: $\alpha(S) \sqsubseteq s$ iff $S \subseteq \gamma(s)$.*

Proof. Let S, s be such that $\alpha(S) \sqsubseteq s$. Therefore, $\mathsf{proj}(S, n) \subseteq s(n)\ \forall n \in N$ by the definition of $\sqsubseteq$. Pick any, $\sigma \in S$. First, $\mathsf{proj}(\sigma, n) \in \mathsf{proj}(S, n)$ and therefore, $\mathsf{proj}(\sigma, n) \in s(n)$ for all $n \in N$. Thus, $\sigma \in \mathsf{ext}(s(n))$ for all $n \in N$. Therefore, $\sigma \in \bigcap_{n \in N} \mathsf{ext}(s(n))$. Thus, $\sigma \in \gamma(s)$, by defn. of γ. We conclude that $S \subseteq \gamma(s)$.

Conversely, assume $S \subseteq \gamma(s)$. Since $\gamma(s) = \bigcap_{n \in N} \mathsf{ext}(s(n))$ (from Def. 3.6). Therefore, $S \subseteq \mathsf{ext}(s(n))$ forall $n \in N$. Therefore, for all $\sigma \in S$, $\mathsf{proj}(\sigma, n) \in s(n)$. Therefore, $\mathsf{proj}(S, n) \subseteq s(n)$ for every $n \in N$. Finally, this yields $\alpha(S) \sqsubseteq s$. □

The meet operation is defined as $s_1 \sqcap s_2 :\ \lambda n : N.\ s_1(n) \cap s_2(n)$, and likewise, the join is defined as $s_1 \sqcup s_2 :\ \lambda n : N.\ s_1(n) \cup s_2(n)$. We recall two key facts that follow from Galois connection between α and γ.

1. For any set $S \subseteq D$, we have $S \subseteq \gamma(\alpha(S))$. Abstracting a concrete set and concretizing it back again "loses information". To see why, we start from $\alpha(S) \sqsubseteq \alpha(S)$ and apply the Galois connection to derive $S \subseteq \gamma(\alpha(S))$.
2. Likewise, for any abstract domain object $s \in \mathbb{A}$, we have $\alpha(\gamma(s)) \sqsubseteq s$. I.e, for any element s, taking its concretization and abstracting it "gains information". To prove this, we start from $\gamma(s) \subseteq \gamma(s)$, and conclude that $\alpha(\gamma(s)) \sqsubseteq s$.

Example 3.2. Returning back to Ex. 3.1, now consider the set

$$\hat{S} = \{(\overset{x_1}{1}, \overset{x_2}{1}, \overset{x_3}{2}), (\overset{x_1}{1}, \overset{x_2}{2}, \overset{x_3}{3}), (\overset{x_1}{2}, \overset{x_2}{1}, \overset{x_3}{2}), (\overset{x_1}{2}, \overset{x_2}{2}, \overset{x_3}{4})\}\,.$$

Its abstraction $\hat{s} : \alpha(\hat{S})$ is given by the mapping:

$$n_1 \mapsto \{(\overset{x_1}{1}, \overset{x_2}{1}), (\overset{x_1}{1}, \overset{x_2}{2}), (\overset{x_1}{2}, \overset{x_2}{1}), (\overset{x_1}{2}, \overset{x_2}{2})\},\ n_2 \mapsto \{(\overset{x_2}{1}, \overset{x_3}{2}), (\overset{x_2}{2}, \overset{x_3}{3}), (\overset{x_2}{2}, \overset{x_3}{4})\}\,.$$

We note that $\gamma(\hat{s})$ is the set: $\{(\overset{x_1}{1}, \overset{x_2}{1}, \overset{x_3}{2}), (\overset{x_1}{1}, \overset{x_2}{2}, \overset{x_3}{3}), (\overset{x_1}{1}, \overset{x_2}{2}, \overset{x_3}{4}), (\overset{x_1}{2}, \overset{x_2}{1}, \overset{x_3}{2}), (\overset{x_1}{2}, \overset{x_2}{2}, \overset{x_3}{3}), (\overset{x_1}{2}, \overset{x_2}{2}, \overset{x_3}{4})\}$. Thus $\hat{S} \subseteq \gamma(\hat{s})$. Notice that $(\overset{x_1}{2}, \overset{x_2}{2}, \overset{x_3}{3})$ and $(\overset{x_1}{1}, \overset{x_2}{2}, \overset{x_3}{4})$ are part of $\gamma(\hat{s})$ but not the original set $\hat{S}$. Similarly, consider the abstract element s_1: $n_1 \mapsto \{(\overset{x_1}{1}, \overset{x_2}{1}), (\overset{x_1}{1}, \overset{x_2}{2})\},\ n_2 \mapsto \{(\overset{x_2}{1}, \overset{x_3}{3})\}$. We note that $\gamma(s_1) : \{(\overset{x_1}{1}, \overset{x_2}{1}, \overset{x_3}{3})\}$ and therefore $\alpha(\gamma(s_1))$ yields the abstract element $s_2 \sqsubseteq s_1$: $n_1 \mapsto \{(\overset{x_1}{1}, \overset{x_2}{1})\},\ n_2 \mapsto \{(\overset{x_2}{1}, \overset{x_3}{3})\}$.

3.2 Canonical Elements and Message Passing

In the tree decomposition, various nodes share information about the subsets of vertices associated with each node. Since the subsets have elements in common, it is possible that a node n_1 has information about a variable x_2 that is also present in some other node n_2 of the tree. We will now see how to take an abstract element s and refine each $s(n)$ by exchanging information between nodes in a systematic manner.

For each edge $(n_1, n_2) \in C$ of the tree, define the set of variables in common as $\mathrm{CV}(n_1, n_2) : \textsc{verts}(n_1) \cap \textsc{verts}(n_2)$ and $\mathrm{CV}_X(n_1, n_2) :\ \textsc{verts}_X(n_1) \cap \textsc{verts}_X(n_2)$

Definition 3.7 (Canonical Elements)**.** An abstract element s is said to be *canonical* if and only if for each edge $(n_1, n_2) \in C$ in the tree:

$$\mathsf{proj}(s(n_1), \mathrm{CV}_X(n_1, n_2)) = \mathsf{proj}(s(n_2), \mathrm{CV}_X(n_1, n_2))\,.$$

In other words, if we took the common variables $\textsc{verts}_X(n_1) \cap \textsc{verts}_X(n_2)$, the set $s(n_1)$ projected along these common variables is equal to the projection of $s(n_2)$ along the common variables.

Example 3.3. Consider the abstract element s_1 from Ex. 3.2: $n_1 \mapsto \{(\overset{x_1}{1}, \overset{x_2}{1}), (\overset{x_1}{1}, \overset{x_2}{2})\}$, $n_2 \mapsto \{(\overset{x_2}{1}, \overset{x_3}{3})\}$. $\mathsf{proj}(s_1(n_1), \mathrm{CV}(n_1, n_2))$ is the set $\{\overset{x_2}{1}, \overset{x_2}{2}\}$ whereas $\mathsf{proj}(s_1(n_2), \mathrm{CV}(n_1, n_2))$ is simply $\{\overset{x_2}{1}\}$. Therefore, s_1 fails to be canonical.

The key theorem of tree decomposition is that a canonical element in the abstract domain can be seen as the projection of a concrete set S along $\textsc{verts}_X(n)$ for each node n of the tree.

Theorem 3.3. *An element s is canonical (Def.3.7) if and only if $s = \alpha(S)$ for some concrete set S.*

To prove this fact, we will first establish a few useful lemmas.

Consider a labeling that associates a valuation $\pi_n : \mathsf{proj}(D, n)$ with every node $n \in N$ of the tree decomposition. First, we will establish the equivalent of Theorem 3.3 when each node in the tree is labeled with a singleton valuation.

Lemma 3.4. *Given a state labeling π_n for $n \in N$, the following are equivalent:*
***(A)** There exists (a unique) $\sigma \in D$ such that $\mathsf{proj}(\sigma, n) = \pi_n$ for each $n \in N$.*
***(B)** For each edge (m, n) of the tree decomposition, the labeling satisfies*

$$\mathsf{proj}(\pi_n, \mathrm{CV}(m, n)) = \mathsf{proj}(\pi_m, \mathrm{CV}(m, n))\,. \tag{1}$$

Note that the lemma above is akin to the canonicity condition but for a concrete set containing just a single state.

Proof. It is trivial to see how (A) implies (B). We prove that (B) implies (A).

We arbitrarily designate a node $r \in N$ as the "root" of the tree and having done so, consider T as a *rooted* tree.

Let π be a labeling that satifies (1). We will construct a $\sigma \in D$ such that $\mathsf{proj}(\sigma, n) = \pi_n$ for each $n \in N$.

The construction of $\sigma \in D$ proceeds by induction on the subtrees of the rooted tree starting from the leaves. For each subtree T_j rooted at node n_j, let us define the set $X_j = \bigcup_{m \in T_j} \textsc{verts}_X(m)$. X_j collects all variables that occur in some node of the subtree T_j. Our goal is to construct a state $\sigma_j \in \mathsf{proj}(D, X_j)$ for the subtree T_j such that for any node $p \in T_j$, $\mathsf{proj}(\sigma_j, p) = \pi_p$.

Each leaf l of the tree is its own subtree. We associate it with the state $\sigma_l = \pi_l$. As a result, $\mathsf{proj}(\sigma_l, l) = \pi_l$, trivially holds at the leaves.

Proceeding by induction, let m be a node with subtrees $T_1, \ldots, T_k$ rooted at nodes $m_1, \ldots, m_k$, which have corresponding sets $X_1, \ldots, X_k$ (as defined above) and states $\sigma_1, \ldots, \sigma_k$ associated with the respective subtrees. Furthermore, we note that by induction hypothesis $\mathsf{proj}(\sigma_i, p_i) = \pi_{p_i}$ for any node $p_i \in T_i$. We will construct the required state σ_m for the subtree rooted at m as follows:

$$\sigma_m(x_l) = \begin{cases} \pi_m(x_l) & \text{if } x_l \in \textsc{verts}_X(m) \\ \sigma_j(x_l) & \text{if } x_l \in X_j \setminus \textsc{verts}_X(m),\ j \in 1, \ldots, k \end{cases}$$

Note that $X_m = \textsc{verts}_X(m) \cup \bigcup_{j=1}^{k} X_k$. Note that $\pi_m \in \mathsf{proj}(D, m)$ and assuming inductively that $\sigma_j \in \mathsf{proj}(D, X_j)$, we can establish from assumption that $D : D_1 \otimes \cdots \otimes D_n$ that $\sigma_m \in \mathsf{proj}(D, X_m)$.

We are required to prove that $\mathsf{proj}(\sigma_m, p) = \pi_p$ for any node $p \in T_m$. Pick any node $p \in T_m$. There are two cases to consider.

Case-1: $p = m$. Since $\sigma_m(x_l) = \pi_m(x_l)$ for all $x_l \in \textsc{verts}_X(m)$, we conclude that $\mathsf{proj}(\sigma_m, p) = \pi_p$.

Case-2: $p \in T_j$ for some subtree T_j for $1 \leq j \leq k$. First, we will prove that $\mathsf{proj}(\sigma_m, X_j) = \sigma_j$. To see why, we consider any variable $x_l \in X_j$ and in turn split on two cases:

Case-2a: $x_l \in \textsc{verts}_X(m)$. In this case, by running intersection property, we conclude that $x_l \in \textsc{verts}_X(m_j)$. Therefore, $x_l \in \textsc{verts}_X(m_j) \cap \textsc{verts}_X(m)$. Thus, by (1), we note that $\pi_{m_j}(x_l) = \pi_m(x_l)$. Therefore,

$$\sigma_m(x_l) = \pi_m(x_l) = \pi_{m_j}(x_l) = \sigma_j(x_l)$$

The last equality is given by the induction hypothesis $\mathsf{proj}(\sigma_j, m_j) = \pi_{m_j}$ and $x_l \in \textsc{verts}_X(m_j)$.

Case-2b $x_l \notin \textsc{verts}_X(m)$. Thus, $x_l \in X_j \setminus \textsc{verts}_X(m)$. Hence, $\sigma_m(x_l) = \sigma_j(x_l)$.

Combining cases 2a and 2b, we conclude that $\mathsf{proj}(\sigma_m, X_j) = \sigma_j$. Therefore, for any $p \in T_j$, we have $\mathsf{proj}(\sigma_m, p) = \mathsf{proj}(\mathsf{proj}(\sigma_m, X_j), p)$ since $\textsc{verts}_X(p) \subseteq X_j$. However, $\mathsf{proj}(\sigma_j, p) = \pi_p$ by induction hypothesis.

Combining cases 1 and 2, we conclude that $\mathsf{proj}(\sigma_m, p) = \pi_p$ for any $p \in T_m$.

Finally, at the root, we have a state $\sigma : \sigma_r$ and the subtree T_r is now the entire tree. This concludes the required existence of σ. Uniqueness of σ follows by the following simple argument. Let σ_1, σ_2 be two different states such that $\sigma_1(x_i) \neq \sigma_2(x_i)$ for some $x_i \in X$. Since $x_i \in \textsc{verts}_X(n_j)$ for some n_j (by condition for being a tree decomposition), we have that $\mathsf{proj}(\sigma_1, n_j) = \pi_j = \mathsf{proj}(\sigma_2, n_j)$. However, this directly contradicts the assumption that $\sigma_1(x_i) \neq \sigma_2(x_i)$.

□

Lemma 3.5. *For every canonical element $s \in \mathbb{A}$, node $n \in N$ and element $\sigma_n \in s(n)$, we have that $\mathsf{ext}(\sigma_n) \cap \gamma(s) \neq \emptyset$.*

Stated another way, the lemma claims that for any canonical s, any $\sigma_n \in s(n)$ can be extended to form some element of $\gamma(s)$.

Proof. Let s be canonical. Pick any $n \in N$ and $\sigma_n \in s(n)$. We will construct a state $\sigma \in \gamma(s)$ such that $\mathsf{proj}(\sigma, n) = \sigma_n$. This immediately shows the result since, $\sigma \in \mathsf{ext}(\sigma_n)$ and $\sigma \in \gamma(s)$. We will achieve this construction through a process called (downward) "message passing" in the tree.

First, we convert the tree into a rooted tree designating the node n as the root.

Next, for every child $\hat{n}$ of n, we send a "message" $\mathsf{msg}(n, \hat{n}) : \mathsf{proj}(\sigma_n, \mathrm{CV}_X(\hat{n}, n))$. Once any node m receives a message from its parent $\hat{m}$, it chooses $\sigma_m \in s(m)$ such that $\mathsf{proj}(\sigma_m, \mathrm{CV}_X(m, \hat{m})) = \mathsf{msg}(\hat{m}, m)$.

Such a choice can always be made at each node by canonicity of s. This is because we have (a) $\mathsf{proj}(s(m), \mathrm{CV}_x(m, \hat{m})) = \mathsf{proj}(s(\hat{m}), \mathrm{CV}_x(m, \hat{m}))$, (b) $\mathsf{msg}(\hat{m}, m) \in \mathsf{proj}(s(\hat{m}), \mathrm{CV}_x(m, \hat{m}))$ and therefore $\mathsf{msg}(\hat{m}, m) \in \mathsf{proj}(s(m), \mathrm{CV}_x(m, \hat{m}))$.

Therefore, each node m that receives a message from its parent makes a choice $\sigma_m \in s(m)$ as noted above and sends a message to each of its children by projecting σ_m along the common dimensions. The process stops when the messages reach the leaves of the rooted tree.

Once the process stops, each node $m \in N$ is labeled by an element $\sigma_m \in s(m)$. Furthermore, for any edge $(n_1, n_2) \in T$, we have the property that $\mathsf{proj}(\sigma_{n_1}, \mathrm{CV}_X(n_1, n_2)) = \mathsf{proj}(\sigma_{n_2}, \mathrm{CV}_X(n_1, n_2))$ by construction. Applying Lemma 3.4, we note that there exists $\sigma \in D$ such that $\mathsf{proj}(\sigma, \textsc{verts}_X(m)) = \sigma_m$ for all nodes m.

To complete the proof, we note that $\sigma \in \mathsf{ext}(s(m))$ for every node $m \in N$ and therefore $\sigma \in \gamma(s)$. At the same time, $\mathsf{proj}(\sigma, \textsc{verts}_x(n)) = \sigma_n$ as required. □

Proof of Theorem 3.3

Proof. Let s be any given canonical abstract domain element. We will now prove that $S : \gamma(s)$ will satisfy the required condition $s = \alpha(S)$. In other words, $\alpha(\gamma(s)) = s$. Assuming otherwise, we conclude that $\alpha(S) \neq s$. Let $\hat{s}$ denote $\alpha(S)$. We know from the discussion above that $\hat{s} = \alpha(S) = \alpha(\gamma(s)) \sqsubseteq s$. Combining the two facts, we note that for all $n \in N$, $\hat{s}(n) \subseteq s(n)$ but there exists a node $r \in N$ such that $\hat{s}(r) \subset s(r)$. Hence, there exists $\sigma_r \in s(r)$ such that $\sigma_r \notin \hat{s}(r)$. In other words, $\sigma_r \notin \mathsf{proj}(S, r)$. Thus, $\mathsf{ext}(\sigma_r) \cap S = \emptyset$. However, using lemma 3.5 we note that for a canonical s, and for every $\sigma_r \in s(r)$, $\mathsf{ext}(\sigma_r) \cap \gamma(s) \neq \emptyset$. This is a direct contradiction. Therefore, we conclude that $\alpha(S) = s$. Thus, $s = \alpha(\gamma(s))$.

To prove the other direction, let us take a set S and compute $s : \alpha(S)$. Note that $s(n) = \mathsf{proj}(S, \textsc{verts}_X(n))$ by definition. We need to show that s is canonical. Consider an edge of the tree (m, n). We have that

$$\begin{array}{rcll} \mathsf{proj}(s(n), \mathrm{CV}_X(m,n)) & = & \mathsf{proj}(\mathsf{proj}(S, \textsc{verts}_X(n)), \mathrm{CV}_X(m,n)) & \\ & = & \mathsf{proj}(S, \mathrm{CV}_X(m,n)) & \because \mathrm{CV}_X(m,n) \subseteq \textsc{verts}_X(n) \\ & = & \mathsf{proj}(\mathsf{proj}(S, \textsc{verts}_X(m)), \mathrm{CV}_X(m,n)) & \because \mathrm{CV}_X(m,n) \subseteq \textsc{verts}_X(m) \\ & = & \mathsf{proj}(s(m), \mathrm{CV}_X(m,n)) & \end{array}$$

This establishes that s is canonical. □

Ideally, in abstract interpretation, we would like to work with abstract domain objects that satisfy $s = \alpha(\gamma(s))$. One way to ensure that is to take any given domain element s_0 and simply calculate out $\alpha(\gamma(s_0))$ by applying the maps. However, $\gamma(s_0)$ in our domain takes lower dimensional projections and reconstructs a set in the full state space. It may thus be too expensive to compute. Fortunately, canonical objects satisfy the equality $s = \alpha(\gamma(s))$. Therefore, given any object $s \in \mathbb{A}$ that is not necessarily canonical, we would like to make it canonical: I.e, we seek an object $\hat{s}$ such that $\gamma(\hat{s}) = \gamma(s)$, but $\hat{s}$ is canonical. As mentioned earlier, directly computing $\hat{s} = \alpha(\gamma(s))$ can be prohibitively expensive, depending on the domain. We now describe a *message passing* approach.

First, we convert the tree T to a rooted tree by designating an arbitrary node $r \in N$ as the root of the tree. The choice of the node r does not matter for what follows. Let $\mathbf{p}(m)$ denote the parent of node m in this rooted tree for $m \neq r$.

Message Passing along Edges: Let (n_1, n_2) be an edge of the tree and s be an abstract element. A message from n_1 to n_2 is defined as the set $\mathsf{msg}(s, n_1 \to n_2) : \mathsf{proj}(s(n_1), \mathrm{CV}(n_1, n_2))$. In other words, we project the set $s(n_1)$ along the dimensions that are common to (n_1, n_2).

Once a node n_2 receives $M : \mathsf{msg}(s, n_1 \to n_2)$, it processes the message by updating $s(n_2)$ as $s(n_2) := s(n_2) \cap \mathsf{ext}_{\textsc{verts}(n_2)}(M)$. In other words, it intersects the message (extended to the dimensions in n_2) with the current set that is associated with n_2.

Example 3.4. Consider a tree decomposition with three nodes $\{n_1, n_2, n_3\}$ and the edges (n_1, n_2) and (n_2, n_3). Let $\textsc{verts}(n_1) : \{x_1, x_2\}$, $\textsc{verts}(n_2) : \{x_2, x_4\}$ and $\textsc{verts}(n_3) : \{x_2, x_3\}$. Let D be the domain $\{1, 2, 3, 4\}^4$. Consider the abstract element s:

$$n_1 \mapsto \{(\overset{x_1}{1}, \overset{x_2}{2}), (\overset{x_1}{3}, \overset{x_2}{3}), (\overset{x_1}{1}, \overset{x_2}{4})\},\ n_2 \mapsto \{(\overset{x_2}{1}, \overset{x_4}{1}), (\overset{x_2}{2}, \overset{x_4}{2}), (\overset{x_2}{3}, \overset{x_4}{3}), (\overset{x_2}{4}, \overset{x_4}{4})\},\ n_3 \mapsto \{(\overset{x_2}{4}, \overset{x_3}{4}), (\overset{x_2}{2}, \overset{x_3}{3})\}\,.$$

A message $\mathsf{msg}(s, n_1 \to n_2)$ is given by the set $\mathsf{proj}(s(n_1), \{x_2\}) : \{\overset{x_2}{2}, \overset{x_2}{3}, \overset{x_2}{4}\}$. This results in the new abstract object s' wherein the element $(\overset{x_2}{1}, \overset{x_4}{1})$ is removed from $s(n_2)$:

$$n_1 \mapsto \{(\overset{x_1}{1}, \overset{x_2}{2}), (\overset{x_1}{3}, \overset{x_2}{3}), (\overset{x_1}{1}, \overset{x_2}{4})\},\ n_2 \mapsto \{\cancel{(\overset{x_2}{1}, \overset{x_4}{1})}, (\overset{x_2}{2}, \overset{x_4}{2}), (\overset{x_2}{3}, \overset{x_4}{3}), (\overset{x_2}{4}, \overset{x_4}{4})\},\ n_3 \mapsto \{(\overset{x_2}{4}, \overset{x_3}{4}), (\overset{x_2}{2}, \overset{x_3}{3})\}\,.$$

Upwards Message Passing: The upwards message passing works from leaves up to the root of the tree according to the following two rules:

1. First, each leaf of the tree n passes a message to its parent $n_p : \mathbf{p}(n)$. The parent node n_p intersects its current value $s(n_p)$ with the message to update its current set.
2. After a node has received (and processed) a message from all its children, it passes a message up to its parent, if one exists.

The upwards message passing terminates at the root since it does not have a parent to send a message to.

Example 3.5. Going back to Ex. 3.4, we designate n_2 as the root and the upwards pass sends the messages $\mathsf{msg}(s, n_1 \to n_2)$ and $\mathsf{msg}(s, n_3 \to n_2)$. This results in the following updated element:

$$n_1 \mapsto \{(\overset{x_1}{1}, \overset{x_2}{2}), (\overset{x_1}{3}, \overset{x_2}{3}), (\overset{x_1}{1}, \overset{x_2}{4})\},\ n_2 \mapsto \{(\cancel{\overset{x_2}{1}, \overset{x_4}{1}}), (\overset{x_2}{2}, \overset{x_4}{2}), (\cancel{\overset{x_2}{3}, \overset{x_4}{3}}), (\overset{x_2}{4}, \overset{x_4}{4})\},\ n_3 \mapsto \{(\overset{x_2}{4}, \overset{x_3}{4}), (\overset{x_2}{2}, \overset{x_3}{3})\}\,.$$

Downwards Message Passing: The downwards message passing works from the root down to the leaves.

1. To initialize, the root sends a message to all its children.
2. After a node has received (and processed) a message from its parent, it sends a message to all its children.

The overall procedure to make a given abstract object s canonical is as follows: (a) perform an upwards message passing phase and (b) perform a downwards message passing phase.

Example 3.6. Going back to Ex. 3.5, the downward message passing phase sends messages from $n_2 \to n_1$ and $n_2 \to n_3$. The resulting element $\hat{s}$ is

$$n_1 \mapsto \{(\overset{x_1}{1}, \overset{x_2}{2}), (\cancel{\overset{x_1}{3}, \overset{x_2}{3}}), (\overset{x_1}{1}, \overset{x_2}{4})\},\ n_2 \mapsto \{(\cancel{\overset{x_2}{1}, \overset{x_4}{1}}), (\overset{x_2}{2}, \overset{x_4}{2}), (\cancel{\overset{x_2}{3}, \overset{x_4}{3}}), (\overset{x_2}{4}, \overset{x_4}{4})\},\ n_3 \mapsto \{(\overset{x_2}{4}, \overset{x_4}{4}), (\overset{x_2}{2}, \overset{x_4}{3})\}\,.$$

On the other hand, it is important to perform message passing upwards first and then downwards second. Reversing this does not yield a canonical element. For instance going back to Ex. 3.4, if we first performed a downwards pass from n_2, the result is unchanged:

$$n_1 \mapsto \{(\overset{x_1}{1}, \overset{x_2}{2}), (\overset{x_1}{3}, \overset{x_2}{3}), (\overset{x_1}{1}, \overset{x_2}{4})\},\ n_2 \mapsto \{(\overset{x_2}{1}, \overset{x_4}{1}), (\overset{x_2}{2}, \overset{x_4}{2}), (\overset{x_2}{3}, \overset{x_4}{3}), (\overset{x_2}{4}, \overset{x_4}{4})\},\ n_3 \mapsto \{(\overset{x_2}{4}, \overset{x_3}{4}), (\overset{x_2}{2}, \overset{x_3}{3})\}\,.$$

Performing an upwards pass now yields the element s_2:

$$n_1 \mapsto \{(\overset{x_1}{1}, \overset{x_2}{2}), (\overset{x_1}{3}, \overset{x_2}{3}), (\overset{x_1}{1}, \overset{x_2}{4})\},\ n_2 \mapsto \{(\cancel{\overset{x_2}{1}, \overset{x_4}{1}}), (\overset{x_2}{2}, \overset{x_4}{2}), (\cancel{\overset{x_2}{3}, \overset{x_4}{3}}), (\overset{x_2}{4}, \overset{x_4}{4})\},\ n_3 \mapsto \{(\overset{x_2}{4}, \overset{x_4}{4}), (\overset{x_2}{2}, \overset{x_4}{3})\}\,.$$

However this is not canonical, since the element $(\overset{x_1}{3}, \overset{x_3}{3})$ in $s_2(n_1)$ violates the requirement over the edge (n_1, n_2).

We first establish an useful fact of message passing that shows that each step of message passing preserves the concretization.

Lemma 3.6. *Let s be an abstract element and s' be the result of a single message passed along edge (m, n). It follows that $\gamma(s) = \gamma(s')$.*

Proof. As a result of the message passed, we have $s'(m) = s(m)$ for all $m \neq n$ and $s'(n) = s(n) \cap \mathsf{proj}(s(m), \mathrm{CV}(m, n))$.

We have that $\gamma(s') = \cap_{r \in N} \mathsf{ext}(s'(r)) = \cap_{r \in N \setminus \{n\}} \mathsf{ext}(s(r)) \cap \mathsf{ext}(s'(n))$. This is obtained by expanding out the intersection and noting that $s'(r) = s(r)$ for $r \neq n$.

We note that $s'(n) = s(n) \cap \mathsf{proj}(s(m), \mathrm{CV}(m,n))$ and thus,

$$\mathsf{ext}(s'(n)) = \mathsf{ext}(s(n)) \cap \mathsf{ext}(\mathsf{proj}(s(m), \mathrm{CV}(m,n)))\,.$$

However, the second term $\mathsf{ext}(\mathsf{proj}(s(m), \mathrm{CV}(m,n))) \supseteq \mathsf{ext}(s(m))$. Thus, $\mathsf{ext}(s'(n)) = \mathsf{ext}(s(n)) \cap R$, where $R \supseteq \mathsf{ext}(s(m))$.

Therefore, $\gamma(s') = \cap_{r \in N \setminus \{m,n\}} \mathsf{ext}(s(r)) \cap \mathsf{ext}(s(n)) = \gamma(s)$.

□

Let $\hat{s}$ be the resulting abstract object after the message passing procedure finishes.

Theorem 3.7. *The result of message passing $\hat{s}$ is a canonical object, and it satisfies $\gamma(\hat{s}) = \gamma(s)$.*

To prove this theorem, we will first establish two important properties:

An abstract element s is said to have the "downward containment property" for an edge (m,n) wherein $n = \mathbf{p}(m)$, denoted as $s \downarrow (m,n)$, iff $\mathsf{proj}(s(n), \mathrm{CV}(m,n)) \subseteq \mathsf{proj}(s(m), \mathrm{CV}(m,n))$. We write $s \downarrow C$ to denote that s satisfies the downward containment property for all tree edges in the set C.

An abstract element s is said to have the "upward containment property" for an edge (m,n) wherein $n = \mathbf{p}(m)$, denoted as $s \uparrow (m,n)$, iff $\mathsf{proj}(s(m), \mathrm{CV}(m,n)) \subseteq \mathsf{proj}(s(n), \mathrm{CV}(m,n))$. We write $s \uparrow C$ to denote that s satisfies the downward containment property for all tree edges in the set C.

Note that an abstract object s is canonical iff $s \downarrow C$ and $s \uparrow C$. Let (m,n) be an edge in the tree with $n = \mathbf{p}(m)$. Let s_i be any abstract element obtained *after* the message from m to n has been sent during the upwards pass.

Lemma 3.8. *For any abstract value s_i we encounter* after *the message from $m \to n$ has been sent in the upwards message passing phase, $s_i \downarrow (m,n)$ holds.*

Proof. It is easy to see that the property $s_i \downarrow (m,n)$ holds for the abstract object obtained immediately after the message from m to n has been sent:

$$\mathsf{proj}(s_i(n), \mathrm{CV}(m,n)) \subseteq \mathsf{proj}(s_i(m), \mathrm{CV}(m,n))\,.$$

Next, suppose the property held for object s_i encountered after the message from m to n has been sent. Let us assume that some message (p,q) is sent along an edge with $q : \mathbf{p}(p)$. Let s_i' be the resulting object. We wish to show that $s_i' \downarrow (m,n)$. To do so, we distinguish two cases, (a) $q \notin \{m,n\}$, (b) $q = m$ or (c) $q = n$. As a result of case (a), $s_i'(m) = s_i(m)$ and $s_i'(n) = s_i(n)$. Therefore, the property continues to hold for s_i'. Case (b) is impossible since all messages involving m's children need to be passed before m passes a message to n in the upwards phase. Finally, as a result of case (c), we have $s_i'(n) \subseteq s_i(n)$. This means that property $s_i' \downarrow (m,n)$ continues to hold:

$$\mathsf{proj}(s_i'(n), \mathrm{CV}(m,n)) \subseteq \mathsf{proj}(s_i(n), \mathrm{CV}(m,n)) \subseteq \mathsf{proj}(s_i(m), \mathrm{CV}(m,n))\,.$$

However, $\mathsf{proj}(s_i'(m), \mathrm{CV}(m,n)) = \mathsf{proj}(s_i(m), \mathrm{CV}(m,n))$. Therefore, $s_i' \downarrow (m,n)$ is concluded. □

We note that the downward containment property is maintained by "downwards" message passing. Let us assume that $s \downarrow C$ holds for an abstract domain object s.

Lemma 3.9. *Let us pass a message $n \to m$ from the parent node $n : \mathit{parent}(m)$ down to m, resulting in object s'. It follows that $s' \downarrow C$.*

Proof. Note that the message from n to m, modifies $s'(m)$. For all other nodes, $s'(p) = s(p)$ $(p \neq m)$. Therefore, $s' \downarrow (p, q)$ for all $p \neq m$. Since $s \downarrow (m, n)$, we have $\mathsf{proj}(s(n), \mathrm{CV}(m, n)) \subseteq \mathsf{proj}(s(m), \mathrm{CV}(m, n))$. As a result of the message passing, we have that $\mathsf{proj}(s'(n), \mathrm{CV}(m, n)) = \mathsf{proj}(s'(m), \mathrm{CV}(m, n))$. In other words, $s' \downarrow (m, n)$ holds. Combining, we conclude $s' \downarrow C$. □

Let s_j be any abstract object obtained after a downward message from $n \to m$ has been sent during the downward message passing phase.

Lemma 3.10. *$s_j \uparrow (m, n)$ holds.*

Proof. As a proof, note that $s_j \uparrow (m, n)$ holds immediately after the message $n \to m$ has been sent.

Next, note that subsequent to the message (m, n) any other message $p \to q$ must satisfy the criterion that $q \neq m$ because m has a unique parent n, and $q \neq n$, since the message from n to m is sent only after n itself has already received a message from its own parent (if any). As a result, we note that if $s_j \uparrow (m, n)$ holds before the message from $p \to q$, then it continues to hold after, during the downwards message passing phase. □

Proof of Theorem 3.7:

Proof. We conclude that $\hat{s} \downarrow C$, and $\hat{s} \uparrow C$ by combining Lemmas 3.8, 3.9, and 3.10. As a result, $\hat{s}$ is canonical.

Next to prove that $\gamma(\hat{s}) = \gamma(s)$, we show that each message passing iteration across an edge does not affect the concretization (Lemma 3.6). There are a total of at most $2|N|$ message passing steps. We prove that $\gamma(s) = \gamma(\hat{s})$ by induction over the number of messages passed. □

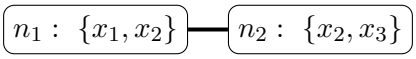
3.3 Decomposable Sets and Post-Conditions

We have already noted that for any concrete set over $S \subseteq D$, the process of abstracting it by projecting into nodes of a tree T, and re-concretizing it is "lossy": I.e, $S \subseteq \gamma(\alpha(S))$. In this section, we study "tree decomposable" concrete sets S for which $\gamma(\alpha(S)) = S$. Ideally, we would like to prove that if a set S is tree decomposable then so is the set $\mathsf{post}(S, \Pi)$ of next states. However, we will disprove this by showing a counterexample. Nevertheless, we will present an analysis of why this fact fails and suggest approaches that can "manage" this loss in precision.

Definition 3.8 (Decomposable Sets)**.** We say that a set S is tree decomposable given a tree T iff $\gamma(\alpha(S)) = S$.

This is in fact a "global" definition of decomposability. In fact, a nice "local" definition can be provided that is reminiscent of the notion of conditional independence in graphical models. We will defer this discussion to an extended version of this paper due to space limitations.

Example 3.7. Consider set $S : \{(\overset{x_1}{1}, \overset{x_2}{2}, \overset{x_3}{1}), (\overset{x_1}{2}, \overset{x_2}{2}, \overset{x_3}{2})\}$ and tree T below:



We wish to check if S is T-decomposable. We have $s : \alpha(S)$ as

$$s(n_1) : \mathsf{proj}(S, n_1) : \{(\overset{x_1}{1}, \overset{x_2}{2}), (\overset{x_1}{2}, \overset{x_2}{2})\}\ s(n_2) : \mathsf{proj}(S, n_2)\{(\overset{x_2}{2}, \overset{x_3}{1}), (\overset{x_2}{2}, \overset{x_3}{2})\}\,.$$

Now, $\gamma(s) : \{(\overset{x_1}{1}, \overset{x_2}{2}, \overset{x_3}{1}), (\overset{x_1}{1}, \overset{x_2}{2}, \overset{x_3}{2}), (\overset{x_1}{2}, \overset{x_2}{2}, \overset{x_3}{1}), (\overset{x_2}{2}, \overset{x_2}{2}, \overset{x_2}{2})\,.\}$. We note that the set S is not tree decomposable. On the other hand, one can verify that the set $S_1 : \{(\overset{x_1}{1}, \overset{x_2}{2}, \overset{x_3}{2}), (\overset{x_1}{2}, \overset{x_2}{2}, \overset{x_3}{2})\}$ is tree decomposable.

Lemma 3.11. *Let $R \subseteq \mathsf{proj}(D, J)$ for a subset of variables J and n be a node in the tree such that $J \subseteq \textsc{verts}(n)$. The set $\mathsf{ext}(R)$ is tree decomposable.*

Proof. Let $S : \mathsf{ext}(R)$ and $s = \alpha(S)$. We note that $s(n) = \mathsf{proj}(S, n) = \mathsf{ext}_{\textsc{verts}(n)}(R)$, since $R \subseteq \mathsf{proj}(D, J)$. Therefore, $\mathsf{ext}(s(n)) \subseteq \mathsf{ext}(R)$. $\gamma(s) = \bigcup_{m \in N} \mathsf{ext}(s(m)) \subseteq \mathsf{ext}(R)$. Therefore, $\gamma(\alpha(S)) \subseteq S$, showing that $\mathsf{ext}(R)$ is tree decomposable. □

Lemma 3.12. *Let S_1, S_2 be tree decomposable sets over T. Their intersection is tree decomposable.*

Proof. Let S_1, S_2 satisfy the condition $\gamma(\alpha(S_i)) = S_i$ for $i = 1, 2$. Let $s_1 = \alpha(S_1)$ and $s_2 = \alpha(S_2)$. Let $s_3 : \alpha(S_1 \cap S_2)$. We have that $s_3(n) = \mathsf{proj}(S_1 \cap S_2, n) \subseteq \mathsf{proj}(S_1, n) \cap \mathsf{proj}(S_2, n)$. Therefore $s_3(n) \subseteq s_1(n) \cap s_2(n)$ for each $n \in N$.

$$\begin{array}{rcl} \gamma(s_3) & = & \bigcap_{n \in N} \mathsf{ext}(s_3(n)) \\ & \subseteq & \bigcap_{n \in N} \mathsf{ext}(s_1(n) \cap s_2(n)) \\ & \subseteq & \bigcap_{n \in N} \mathsf{ext}(s_1(n)) \cap \bigcap_{n \in N} \mathsf{ext}(s_2(n)) \\ & \subseteq & \gamma(S_1) \cap \gamma(S_2) \\ & = & S_1 \cap S_2 \end{array}$$

We have shown that $\gamma(\alpha'(S_3)) \subseteq S_3$ and from Galois connection we can already conclude the other direction of containment holds. As a result S_3 is tree decomposable. □

Let Π be a transition system over system variables in $\sigma \in D$. For a given set $S \subseteq D$, us define the post-condition $\mathsf{post}(S, \Pi)$ to be the set of states reachable in one step starting from some state in S:

$$\mathsf{post}(S, \Pi) : \{\sigma' \mid \sigma \in S,\ \sigma' = \mathsf{eval}(f, \sigma)\}.$$

Let us also consider a transition relation R over pairs of states $(\sigma, \sigma') \in D \otimes D$:

$$R = \{(\sigma, \sigma') \mid \sigma, \sigma' \in D \text{ and } \sigma' = \mathsf{eval}(f, \sigma)\}.$$

The relation R can be viewed as the intersection of n relations: $R : \bigcap_{x_j \in X} R_j$, wherein

$$R_j : \{(\sigma, \sigma') \mid \sigma, \sigma' \in D \text{ and } \sigma'_j = \mathsf{eval}(f_j, \sigma)\}.$$

In other words, R_j is a component of R that models the update of the system variable x_j. Also for each $x_j \in X$, let $e_j : \mathsf{inps}(f_j) \cup x_j$ be the inputs to the update function f_j and the node x_j itself.

Given the tree T, we define the extended tree T' as having the same node set N and edge set C as T. However, $\textsc{verts}_{T'}(n) = \textsc{verts}_T(n) \cup \{x'_j \mid x_j \in \textsc{verts}_T(n)\}$. Note that T' with the labeling $\textsc{verts}_{T'}$ satisfies all the condition of a tree decomposition for a graph G save the addition of vertices x'_i in each node of the tree. We will write $\textsc{verts}'(n)$ to denote the set $\textsc{verts}_{T'}(n)$.

Lemma 3.13. *The transition relation R of a system Π is tree T' decomposable.*

Proof. Note that $R : \bigcap_{x_j \in X} R_j$. We simply show that each R_j is tree decomposable and appeal to Lemma 3.12. Each R_j involves variables in the set e_j and $e_j \subseteq \textsc{verts}(n_j)$ for some node n_j of the tree. As a result, we note that $R_j : \mathsf{ext}(\hat{R}_j)$, wherein $\hat{R}_j$ is a relation that just involves variables in e_j, e'_j which are in $\textsc{verts}_{T'}(n)$. From Lemma 3.11, any set written in this form is tree decomposable. □

First, we show the negative result that the image of a tree (T) decomposable set under a tree (T') decomposable transition relation is not tree decomposable, in general.

Example 3.8. Let $X = \{x_1, x_2, x_3\}$ and consider again the tree decomposition from Ex. 3.7. Let S be the set $\{(\overset{x_1}{*}, \overset{x_2}{*}, \overset{x_3}{*})\}$, wherein we use the wild card character as notation that can be substituted for any element in the set $\{1, 2\}$. Therefore, we take S to be a set with 8 elements. Clearly S is tree decomposable in the tree T from Ex. 3.7.

Consider the transition relation R that will be written as the intersection of three transition relations:

$$R_1 : \{(X, X') \mid x'_1 = x_2\},\ R_2 : \{(X, X') \mid x'_2 \in \{1, 2\}\},\ R'_3 :\ \{(X, X') \mid x'_3 = x_2\}\,.$$

Clearly R is tree T' decomposable. We can now compute the post-condition of S under this relation. The reader can verify the post-condition $\hat{S} :\ \{(\overset{x_1}{1}, \overset{x_2}{*}, \overset{x_3}{1}), (\overset{x_1}{2}, \overset{x_2}{*}, \overset{x_3}{2})\}$. However, $\hat{S}$ is not tree decomposable. We note that $\hat{s} :\ \alpha(\hat{S})$ is the set $\hat{s}(n_1) :\ \{(\overset{x_1}{*}, \overset{x_2}{*})\}$ and $\hat{s}(n_2) :\ \{(\overset{x_1}{*}, \overset{x_2}{*})\}$. Therefore $\gamma(\hat{s})$ is the set $\{(\overset{x_1}{*}, \overset{x_2}{*}, \overset{x_3}{*})\}$.

As noted above, the set R is tree T' decomposable. If S is tree decomposable, we can extend S to a set $S' :\ \mathsf{ext}_{X'}(S)$ that is now defined over $X \cup X'$ and is also tree decomposable. As a result $S' \cap R$ is also tree decomposable. However, the postcondition of S is the set $\mathsf{proj}(S' \cap R, X')$. Thus, the key operation that failed was the projection operation involved in computing the post-condition. This suggests a possible solution to this issue albeit an expensive one: at each step, we maintain the reachable states using both current and next state variables, thus avoiding projection. In effect, the reachable states at the i^{th} step will be entire trajectories of the system expressed over variables $X_0 \cup X_1 \cup \cdots X_i$. This is clearly not practical. However, a more efficient solution is to note that some of the current state variables can be projected out without losing the tree decomposability property. Going back to Ex. 3.8, we note that we can safely project away $\{x_1, x_3\}$, while maintaining the new reachable set in terms of (x_2, x'_1, x'_2, x'_3). In this way, we may recover the lost precision back.

In conclusion, we note that tree decompositions may lose precision over post-conditions. However, the loss in precision can be avoided if carefully selected "previous state variables" are maintained as the computation proceeds. The question of how to optimally maintain this information will be investigated in the future.

4 Grid-Based Interval Analysis

We now combine the ideas to create a disjunctive interval analysis using tree decompositions. The main idea here is to apply tree decompositions not to the concrete set of states but to an abstraction of the concrete domain by grid-based intervals.

We will now describe the interval-based abstraction of sets of states dynamical system Π in order to perform over-approximate reachability analysis. Let us fix a system $\Pi : \langle \mathbf{x}, \mathbf{w}, D, W, f, X_0, U \rangle$ as defined in Def. 2.1. We will assume that the domain of state variables D is a hyper-rectangle given by $D : [L(x_1), U(x_1)] \times \cdots \times [L(x_n), U(x_n)]$ for $L(x_j), U(x_j) \in \mathbb{R}$ and $L(x_j) \leq U(x_j)$ for each $j = 1, \ldots, n$. In other words, each system variable x_j lies inside the interval $[L(x_j), U(x_j)]$. Likewise, we will assume that $W : \prod_{k=1}^{m} [L(w_k), U(w_k)]$ such that $L(w_k) \leq U(w_k)$ and $L(w_k), U(w_k) \in \mathbb{R}$.

We will consider a *uniform* cell decomposition wherein each dimension is divided into some natural number $M > 0$ of equal sized subintervals. The i^{th} *subinterval* of variable x_j is denoted as $\mathsf{subInt}(x_j, i)$, and is given by $[L(x_j) + i\delta_j, L(x_j) + (i+1)\delta_j]$ for $i = 0, \ldots, M-1$ and $\delta_j : \frac{(U(x_j) - L(x_j))}{M}$. Similarly, we will define $\mathsf{subInt}(w_k, i)$ for disturbance variables w_k whose domains are also divided into M subdivisions. The overall domain $D \times \mathcal{W}$ is therefore divided into M^{m+n} cells wherein

each cell is indexed by a tuple of natural numbers $\mathbf{i} : \langle i_1, \ldots, i_n, i_{n+1}, \ldots, i_{n+m} \rangle$, such that $i_j \in \{0, \ldots, M-1\}$ and the cell corresponding to $\mathbf{i}$ is given by:

$$\gamma_C(\mathbf{i}) : \prod_{j=1}^{n} \mathsf{subInt}(x_j, \mathbf{i}_j) \times \prod_{k=1}^{m} \mathsf{subInt}(w_k, \mathbf{i}_{n+k}) \tag{2}$$

Definition 4.1 (Grid-Based Abstract Domain)**.** The grid based abstract domain is defined by the set $\mathcal{C} : \mathcal{P}(\mathbf{i} \in \{0, \ldots, M\}^{m+n})$, wherein each abstract domain element is a set of grid cells. The sets are ordered simply by set inclusion $\subseteq$ between sets of grid cells. The abstraction map $\alpha_C : \mathcal{P}(D) \to \mathcal{C}$ is defined as follows:

$$\alpha_C(S) : \{\mathbf{i} \in \mathcal{C} \mid \gamma_C(\mathbf{i}) \cap S \neq \emptyset\}.$$

The concretization map γ_C is defined above in (2).

Definition 4.2 (Interval Propagator)**.** An interval propagator (IP) is a higher order function that takes in the description of a function f with k real-valued inputs and p real valued outputs, and an interval $I : [l_1, u_1] \times \cdots \times [l_k, u_k]$ and outputs an interval (hyperrectangle over $\mathbb{R}^p$) $\textsc{IntvlProp}(f, I)$ such that the following soundness guarantees hold:

$$(\forall \mathbf{x} \in D) \bigwedge_{j=1}^{k} \mathbf{x}_j \in [l_j, u_j] \;\Rightarrow\; \mathsf{eval}(f, \mathbf{x}) \in \textsc{IntvlProp}(f, I).$$

In practice, interval arithmetic approaches have been used to build sound interval propagators [33]. However, they suffer from issues such as the *wrapping effect* that make their outputs too conservative. This can be remedied by either (a) performing a finer subdivision of the inputs (i.e, increasing M) to ensure that the intervals I being input into the IntvlProp are sufficiently small to guarantee tight error bounds; or (b) using higher order arithmetics such as affine arithmetic or Taylor polynomial arithmetic [25, 32].

The interval propagator serves to define an abstract post-condition operation over sets of cells $\hat{S} \subseteq \mathcal{C}$. Given such a set, $\hat{S}$, we compute the post condition in the abstract domain. Informally, the post condition is given (a) by iterating over each cell in S; and (b) computing the possible next cells using IntvlProp. Formally, we define the abstract post operation as follows:

$$\mathsf{post}_C(\hat{S}, \Pi) : \bigcup_{\mathbf{i} \in \hat{S}} \alpha_C(\textsc{IntvlProp}(f, \gamma_C(\mathbf{i}))).$$

Given this machinery, an abstract T-step reachability analysis is performed in the standard manner: (a) abstract the initial state; (b) compute post condition for T steps; and (c) check for intersections of the abstract states with the abstraction of the unsafe set. We can also define and use widening operators to make the sequence of iterates converge. The grid based abstract domain can offer some guarantees with respect to the quality of the abstraction. For instance, we can easily bound the Hausdorff distance between the underlying concrete set and the abstraction as a function of the discretization sizes δ_j. However, the desirable properties come at a high computational cost since the number of cells grows exponentially in the number of system and disturbance variables.

4.1 Tree Decomposed Analysis

We now consider a tree-decomposed approach based on the concept of nodal abstractions. The key idea here is to perform the grid-based abstraction not on the full set of system and disturbance variables, but instead on individual *nodal* abstractions over a tree decomposition T.

Definition 4.3 (Nodal Abstractions)**.** A nodal abstraction $\textsc{NodalAbstraction}(\Pi, n)$ corresponding to a node $n \in N$ is defined as follows:

1. The set of system variables are given by $X_n : \textsc{verts}_X(n)$ with domain given by $D_n : \mathsf{proj}(D, X_n)$.
2. The initial states are given by $\mathsf{proj}(X_0, X_n)$.
3. The unsafe set is given by $\mathsf{proj}(U, X_n)$.
4. The set of disturbance variables are $Y_n : \textsc{verts}_W(n)$ with domain given by $W_n : \mathsf{proj}(W, W_n)$.
5. The updates are described by a *relation* $R(X_n, X'_n)$ that relate the possible current states X_n and next states X'_n. The relation is constructed as a conjunction of assertions over variables x_i, x'_i wherein $x_i \in X_n$.
   (a) If the update $x_i := f_i(\mathbf{x}, \mathbf{w})$ is associated with the node n, we add the conjunct $x'_i = f_i(X_n, W_n)$, noting that the proper inputs to f_i are contained in $\textsc{verts}(n)$.
   (b) Otherwise, $x'_i \in \mathsf{proj}(D, \{x_i\})$ that simply states that the next state value of the variable x_i is some value in its domain.

Given a system Π, the nodal abstraction is a conservative abstraction, and therefore, it preserves reachability properties.

Lemma 4.1. *For any reachable state* $\mathbf{x}$ *of* Π *at time* t*, its projection* $\mathsf{proj}(\mathbf{x}, X_n)$ *is a reachable state of* $\textsc{NodalAbstraction}(\Pi, n)$ *at time* t*.*

Since each nodal abstraction involves at most $\omega + 1$ variables, the abstraction at each node can involve at most $M^{\omega+1}$ cells where ω is the tree width. Also, note that a tree decomposition can be found with tree width ω that has at most $|X| + |W|$ nodes. This implies that the number of nodal abstractions can be bounded by $(|X| + |W|)$.

Let $\Pi(n) : \textsc{NodalAbstraction}(\Pi, n)$ be the nodal abstraction for tree node $n \in N$. For each node $n \in N$, we instantiate a grid based abstract domain for $\Pi(n)$ ranging over the variables $\textsc{verts}_X(n)$. At the i^{th} step of the reachability analysis, we maintain a map s_i each node n to a set of grid cells $s_i(n)$ defined over $\textsc{verts}(n)$.

1. Compute $\hat{s}_i(n) : \mathsf{post}_C(s_i(n), \Pi(n))$.
2. Make $\hat{s}_i$ canonical using message passing between nodes to obtain s_{i+1}.

The message passing is performed not over projections of concrete states but over cells belonging to the grid based abstract domain. Nevertheless, we can easily extend the soundness guarantees in theorem 3.7 to conclude soundness of the composition.

Once again, we can stop this process after T steps or use widening to force convergence. We now remark on a few technicalities that arise due to the way the tree decomposition is constructed.

Intersections with Unsafe Sets: Checking for a non-empty intersection with the unsafe sets may require constructing concrete cells over the full dimensional space if the unsafe sets are not tree decomposable for the tree T. However in many cases, the unsafe states are specified as intervals over individual variables, which yields a tree decomposable set. In such cases, we need to intersect the abstraction at each node with the unsafe set and perform message passing to make it canonical before checking for emptiness.

Handling Guards and Invariants: We have not discussed guards and invariants. It is assumed that such guards and invariants are tree decomposable over the tree T. In this case, we can check which abstract cells have a non-empty intersection with the guard using message passing. The handling of transition systems with guards and invariants will be discussed as part of future extensions.

Table 1: Results on benchmark examples. $|X|$: Number of state variables, $|W|$: number of disturbance variables, Tree Decomp.: reachability using tree decompositions, Monolithic: reachability analysis without tree decompositions. SAPO: number of directions ($|L|$), number of bundles ($|T|$) and running time. All timings are reported in seconds on a Macbook pro laptop running MacOS 10.14 with 16GB RAM and 3.4 GHz Intel core i7 processor. Reachability analysis was carried out for 15 time steps.

Name	$\|X\|$	$\|W\|$	Tree Width	Tree Decomp.		Monolithic		SAPO	
				Time	# Cells	Time	# Cells	$(\|L\|, \|T\|)$	Time
System # 1	3	1	1	14.4	0.22M	1047.6	7.6M	-n/a-	
System # 2	4	1	2	2.7	24K	652	3.1M	-n/a-	
SIR [23, 40]	3	0	1	4.1	95K	143	2M	(3,1)	0.1
1D-Lattice-10 [39]	10	0	2	99	1.1M	TO (1.5 hours)		(16,6)	679
ebola-epidemic [14]	5	0	2	799.4	1.9M	TO (1.5 hours)		(5,5)	0.02
p53-gene-reg [31]	6	0	2	135.8	98K	TO (1.5 hours)		-n/a-	
influenza-epidemic [22]	4	0	2	517.9	1.4M	TO (1.5 hours)		(7,4)	0.1
coupled-vanderpol	6	0	2	10.5	0.1M	TO (1.5 hours)		(10,5)	2.5
Laub-Loomis [30, 20]	7	0	3	1755.1	2.6M	TO (1.5 hours)		(12,6)	1.8
Honeybee* [23, 9]	6	4	3	206.1	2.1M	TO (1.5 hours)		(8,4)	0.7
Phosporelay [22]	7	0	3	1566.2	7.5M	TO (1.5 hours)		(10,4)	1.2
Coord. Vehicles (1)	5	1	2	150.2	0.5M	TO (1.5 hours)		-n/a-	
Coord. Vehicles (2)	10	2	2	1175.2	2M	TO (1.5 hours)		-n/a-	
Coord. Vehicles (4)	20	4	2	2206.7	3.9M	TO (1.5 hours)		-n/a-	

5 Experimental Evaluation

In this section, we describe an experimental evaluation of our approach over a set of benchmark problems. Our evaluation is based on a C++-based prototype implementation that can read in the description of a nonlinear dynamical system over a set of system and disturbance variables. The dynamics can currently include polynomials, rational functions and trigonometric functions. Our implementation uses the MPFI library to perform interval arithmetic over the grid cells [36]. We use the HTD library to compute tree decompositions [1]. The system then computes a time-bounded reachable set over the first T steps of the system's execution. Currently, we plot the results and compare the reachable set estimates against simulation data. We also compare the reachable sets computed by the tree decomposition approach against an approach without using tree decompositions. However, we note that the latter approach timed out on systems beyond 4 state variables.

Table 1 presents the results over a small set of challenging nonlinear systems benchmarks along with a comparison to two other approaches (a) the approach without tree decomposition and (b) the tool SAPO [22] which computes time bounded reachable sets for polynomial systems using the technique of parallelotope bundles described by Dreossi et al [23]. The benchmarks range in number of system variables from 3 to 20 state variables. We describe the sources for each benchmark where appropriate. Note that the SAPO tool does not handle nonpolynomial dynamics or time varying disturbances at the time of writing.

The treewidths range from 1 for the simplest system (Ex. 2.1) to 3 for the 7-state Laub Loomis oscillator example [30]. We note that the tree decomposition was constructed within 0.01 seconds for all the examples. We also note that systems with as many as 20 state variables are handled

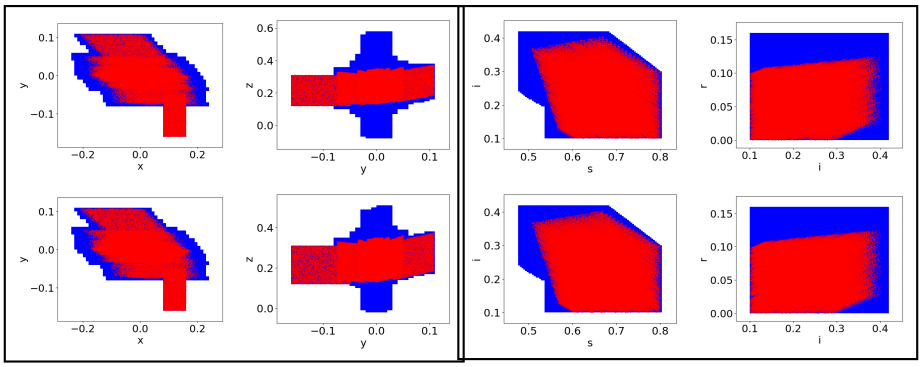


Figure 1: Reachable set projections (shaded blue) for System# 2 (left) and the SIR model [22] (right). Top : tree decomposition approach and Bottom: monolithic approach without tree decompositions. Reachable sets are identical for the SIR model. Note the difference in range of z for the system #2. The red dots show the results of simulations.

by our approach whereas the monolithic approach cannot handle systems beyond 4 state variables. We now compare the results of our approach to that of the monolithic approach on the two cases where the latter approach completed.

System # 1: Consider again the system from Ex. 2.1 with 3 state variables and 1 disturbance. We have already noted a tree decomposition of tree width 1 for this example.

System # 2: In this example, we consider a system over 4 state variables $\{x, y, z, w\}$ and one disturbance variable w_1.

$$x \;:=\; 0.5x + y + 0.05xy - w_1,\; y \;:=\; -0.7y - 0.03x,\; z \;:=\; z - 0.4y,\; w \;:=\; w - 0.05xw$$

The domains include $(x, y, z, w) \in [-1, 1]^4$ and divided into 16×10^8 grid cells (200 for each state variable). The disturbance $w_1 \in [-0.1, 0.1]$. The initial conditions are $x \in [0.08, 0.16], y \in [-0.16, -.05], z \in [0.12, 0, 31]$ and $w \in [-0.15, -0.1]$. We obtain a tree decomposition of width 2, wherein the nodes include $n_1 : \{x, y, w_1\}$, $n_2 : \{y, z\}$ and $n_3 : \{x, w\}$ with the edges (n_1, n_2) and (n_1, n_3).

Figure 1 compares the resulting reachable sets for the tree decomposed reachability analysis versus the monolithic approach. We note differences between the two reachable sets but the loss in precision is not significant.

Coordinated Vehicles: In this example, we study nonlinear vehicle models of vehicles executing coordinated turns. Each vehicle has states $(x_i, y_i, v_{x,i}, v_{y,i}, \omega)$, representing positions, velocities and the rate of change in the yaw angle, respectively, with a disturbance w_i. The dynamics are given by

$$x_i := x_i + 0.1v_{x,i},\; y_i := y_i + 0.1v_{y,i},\; v_{x,i} = v_{x,i} + 0.1v_{x,i}\cos(0.1\omega_i) - 0.1v_{y,i}\sin(0.1\omega_i)$$
$$\omega_i = 0.5\omega_i + 0.5\omega_0 + 0.1w_i$$

The vehicles are loosely coupled with ω_i representing the turn rate of the i^{th} vehicle and ω_0 that of the "lead" vehicle. The i^{th} vehicle tries to gradually align its turn rate to that of the lead vehicle. This model represents a simple scenario of loosely coupled systems that interact using a small set of state variables. applications including models of cardiac cells that are also loosely coupled through shared action potentials [26]. The variables x_i, y_i are set in the domain $[-15, 15]$ and subdivided into 300 parts along each dimension. Similarly, the velocities range over $[-10, 10]$ and are subdivided into 500 parts each and the yaw rate ranges over $[-0.2, 0.2]$ radians/sec and

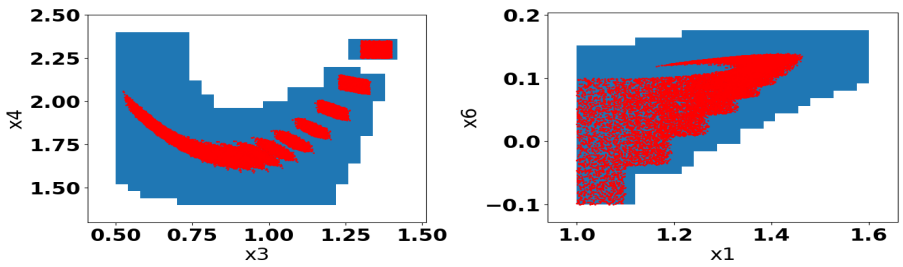


Figure 2: Projection of reachable sets for the Laub-Loomis model.

subdivided into 25 parts. The disturbance ranges over $[-0.1, 0.1]$. Table 1 reports results from models involving $1, 2$ and 4 vehicles. Since they are loosely coupled, the treewidth of these models is 2.

Laub-Loomis Model: The Laub-Loomis model is a molecular network that produces spontaneous oscillations for certain values of the model parameters. The model's description was taken from Dang et al [20]. The system has 7 state variables each of which was subdivided into 100 cells yielding a large state space with 10^{14} cells. We note that the tree width of the graph is 3, yielding nodes with upto 4 variables in them.

Figure 2 shows two projections of the reachable set computed by our approach against numerical simulations.

Comparison With SAPO SAPO is a state-of-the-art tool that uses polytope bundles and Bernstein polynomials to represent and propagate reachable sets for polynomial dynamical systems [22, 23]. We compare our approach directly on SAPO for identical models and initial sets. Note that SAPO does not currently handle non-polynomial models or models with time-varying disturbances. Table 1 shows that SAPO is orders of magnitude faster on all the models, with the sole exception of the 1D-Lattice-10 model. Figure 3 shows the comparison of the reachable sets computed by our approach (shaded blue region) against those computed by SAPO (black rectangles) for five different models. We note that for three of the models compared, neither reachable set is contained in the other. For the one dimensional lattice model, SAPO produces a better reachable set, whereas our approach is better for the influenza model. We also note that both for our approach the precision can be improved markedly by increasing the number of subdivisions, albeit at a large computational cost that depends on the treewidth of the model. The same is true for SAPO, where the number of directions and the template sizes have a non-trivial impact on running time.

Models with Large Treewidths We briefly report on a few models that we attempted with large treewidths. For such models, our approach of decomposing the space into cells becomes infeasible due to the curse of dimensionality.

A model of how honeybees select between different sites [9, 23] has 6 variables and its tree width is 5 with a single tree node containing all state variables. However, the large treewidth is due to two terms in the model which are replaced by disturbance variables that overapproximate their value. This brings down the treewidth to 3, making it tractable for our approach. Details of this transformation are discussed in our extended version. Treewidth reduction using abstractions is an interesting topic for future work.

We originally proposed to analyze a 2D grid lattice model taken from Vleck et al [39]. However, a 2D 10×10 lattice model has a dependency hypergraph that forms a 10×10 grid with treewidth 10.

Likewise, the 17-state crazyflie benchmark for SAPO [22] could not be analyzed by our approach since its treewidth is too large.

6 Conclusions

We have shown how tree decompositions can define an abstract domain that projects concrete sets along the various subsets of state variables. We showed how message passing can be used to exchange information between these subsets. We analyze the completeness of our approach and show that the abstraction is lossy due to the projection operation. We show that for small tree width models, a gridding-based analysis of nonlinear system can be used whereas such approaches are too expensive when applied in a monolithic fashion. For the future, we plan to study tree decompositions for abstract domains such as disjunctions of polyhedra, parallelotope bundles and Taylor models. The process of model abstraction to reduce treewidth is another interesting future possibility.

Acknowledgments: This work was supported by US NSF under award number CPS 1836900, CCF 1815983 and the US Air Force Research Laboratory (AFRL). The author acknowledges Profs. Mohamed Amin Ben Sassi and Fabio Somenzi for helpful discussions, and the anonymous reviewers for their comments.

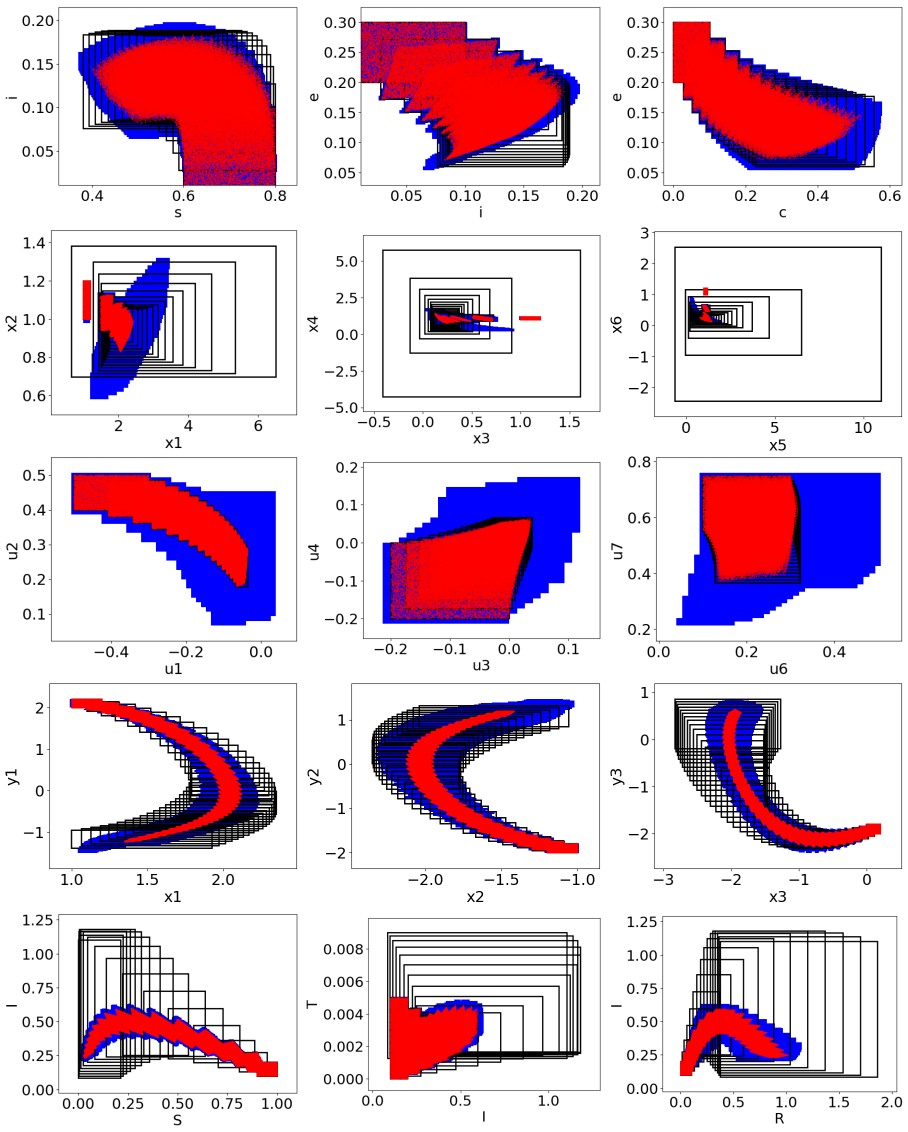


Figure 3: Comparison of various projections of the reachable sets computed by our approach shown in blue, the reachable set computed by SAPO shown as black rectangles and states obtained through random simulation shown in red dots. Top row: ebola model, second row: phosporelay, third row: 1d-lattice-10, fourth row: vanderpol (35 steps) and bottom row: influenza model.

References

[1] Abseher, M., Musliu, N., Woltran, S.: htd – a free, open-source framework for(customized) tree decompositions andbeyond. In: Integration of AI and OR Techniques in Constraint Programming. pp. 376–386. Springer International Publishing, Cham (2017)

[2] Adjé, A., Gaubert, S., Goubault, E.: Coupling policy iteration with semi-definite relaxation to compute accurate numerical invariants in static analysis. In: Programming Languages and Systems. pp. 23–42. Springer Berlin Heidelberg (2010)

[3] Blanchet, B., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., Rival, X.: A static analyzer for large safety-critical software. In: Prog. Lang. Design & Implementation. pp. 196–207. ACM Press (2003)

[4] Blanchet, B., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., Rival, X.: Design and implementation of a special-purpose static program analyzer for safety-critical real-time embedded software (invited chapter). In: In The Essence of Computation: Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones. LNCS, vol. 2566, pp. 85–108. Springer (2005)

[5] Bodlaender, H.L.: Dynamic programming on graphs with bounded treewidth. In: Automata, Languages and Programming. pp. 105–118. Springer (1988)

[6] Bodlaender, H.L.: A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM Journal on Computing **25**(6), 1305–1317 (1996)

[7] Bodlaender, H.L.: Fixed-Parameter Tractability of Treewidth and Pathwidth, pp. 196–227. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)

[8] Bodlaender, H.L., Koster, A.M.: Treewidth computations i. upper bounds. Information and Computation **208**(3), 259 – 275 (2010)

[9] Britton, N.F., Franks, N.R., Pratt, S.C., Seeley, T.D.: Deciding on a new home: how do honeybees agree? Proceedings of the Royal Society of London. Series B: Biological Sciences **269**(1498), 1383–1388 (2002)

[10] Chatterjee, K., Ibsen-Jensen, R., Goharshady, A.K., Pavlogiannis, A.: Algorithms for algebraic path properties in concurrent systems of constant treewidth components. ACM Trans. Program. Lang. Syst. **40**(3) (Jul 2018)

[11] Chatterjee, K., Ibsen-Jensen, R., Pavlogiannis, A., Goyal, P.: Faster algorithms for algebraic path properties in recursive state machines with constant treewidth. In: Principles of Programming Languages (POPL). p. 97109. Association for Computing Machinery, New York, NY, USA (2015)

[12] Chen, M., Herbert, S., Tomlin, C.: Exact and efficient Hamilton-Jacobi-based guaranteed safety analysis via system decomposition. In: IEEE International Conference on Robotics and Automation (ICRA) (2017), to appear, also at arXiv:1609.05248

[13] Chen, X., Sankaranarayanan, S.: Decomposed reachability analysis for nonlinear systems. In: 2016 IEEE Real-Time Systems Symposium (RTSS). pp. 13–24 (Nov 2016)

[14] Chowell, G., Hengartner, N., Castillo-Chavez, C., Fenimore, P., Hyman, J.: The basic reproductive number of ebola and the effects of public health measures: the cases of congo and uganda. Journal of Theoretical Biology **229**(1), 119 – 126 (2004)

[15] Courcelle, B.: The monadic second-order logic of graphs iii: Treewidth, forbidden minors and complexity issues. Informatique Théorique **26**, 257286 (1992)

[16] Cousot, P., Cousot, R.: Static determination of dynamic properties of programs. In: Proc. ISOP'76. pp. 106–130. Dunod, Paris, France (1976)

[17] Cousot, P., Cousot, R.: Comparing the Galois connection and widening/narrowing approaches to Abstract interpretation, invited paper. In: PLILP '92. LNCS, vol. 631, pp. 269–295. springer (1992)

[18] Cousot, P., Cousot, R.: Abstract Interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: ACM Principles of Programming Languages. pp. 238–252 (1977)

[19] Cousot, P., Halbwachs, N.: Automatic discovery of linear restraints among the variables of a program. In: POPL'78. pp. 84–97 (Jan 1978)

[20] Dang, T., Dreossi, T.: Falsifying oscillation properties of parametric biological models. In: Hybrid Systems Biology (HSB). EPTCS, vol. 125, pp. 53–67 (2013)

[21] Delmas, D., Souyris, J.: Astrée: from research to industry. In: Proc. Static Analysis Symposium, SAS. LNCS, vol. 4634, pp. 437–451. Springer, Berlin (2007)

[22] Dreossi, T.: Sapo: Reachability computation and parameter synthesis of polynomial dynamical systems. In: Hybrid Systems:Computation and Control (HSCC). pp. 29–34. ACM (2017)

[23] Dreossi, T., Dang, T., Piazza, C.: Parallelotope bundles for polynomial reachability. In: Hybrid Systems: Computation and Control (HSCC). pp. 297–306. ACM (2016)

[24] Ferrara, A., Pan, G., Vardi, M.Y.: Treewidth in verification: Local vs. global. In: Logic for Programming, Artificial Intelligence, and Reasoning. pp. 489–503. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)

[25] de Figueiredo, L.H., Stolfi, J.: Self-validated numerical methods and applications. In: Brazilian Mathematics Colloquium monograph. IMPA, Rio de Janeiro, Brazil (1997)

[26] Grosu, R., Batt, G., Fenton, F.H., Glimm, J., Guernic, C.L., Smolka, S.A., Bartocci, E.: From cardiac cells to genetic regulatory networks. In: Computer Aided Verification CAV. Lecture Notes in Computer Science, vol. 6806, pp. 396–411. Springer (2011)

[27] Gulwani, S., Jojic, N.: Program verification as probabilistic inference. In: POPL. p. 277289. POPL 07, Association for Computing Machinery (2007)

[28] Ivancic, F., Balakrishnan, G., Gupta, A., Sankaranarayanan, S., Maeda, N., Imoto, T., Pothengil, R., Hussain, M.: Scope bounded software verification in varvel. Journal of Automated Software Engineering (J. ASE) pp. 1–14 (2014)

[29] Koller, D., Friedman, N.: Probabilistic Graphical Models. The MIT Press (2009)

[30] Laub, M.T., Loomis, W.F.: A molecular network that produces spontaneous oscillations in excitable cells of dictyostelium. Molecular biology of the cell **9**(12), 3521–3532 (1998)

[31] Leenders, G., Tuszynski, J.A.: Stochastic and deterministic models of cellular p53 regulation. Frontiers in Oncology **3**(64) (2013)

[32] Makino, K., Berz, M.: Taylor models and other validated functional inclusion methods. International Journal of Pure and Applied Mathematics **4**(4), 379–456 (2003)

[33] Moore, R.E., Kearfott, R.B., Cloud, M.J.: Introduction to Interval Analysis. SIAM (2009)

[34] Nielson, F., Nielson, H.R., Hankin, C.: Principles of Program Analysis. Springer (1999)

[35] Obdržálek, J.: Fast mu-calculus model checking when tree-width is bounded. In: Computer Aided Verification. pp. 80–92. Springer Berlin Heidelberg, Berlin, Heidelberg (2003)

[36] Revol, N., Rouillier, F.: Motivations for an arbitrary precision interval arithmetic and the mpfi library. Reliable Computing **11**, 275290 (2005)

[37] Robertson, N., Seymour, P.: Graph minors. iii. planar tree-width. Journal of Combinatorial Theory, Series B **36**(1), 49 – 64 (1984)

[38] Thorup, M.: All structured programs have small tree width and good register allocation. Information and Computation **142**(2), 159 – 181 (1998)

[39] Vleck, E.S.V., Mallet-Paret, J., Cahn, J.W.: Traveling wave solutions for systems of odes on a two-dimensional spatial lattice. SIAM Journal of Applied Mathematics **59**, 455–493 (1998)

[40] Weisstein, E.W.: SIR model, from MathWorld–A Wolfram Web Resource. https://mathworld.wolfram.com/SIRModel.html. Accessed May 2020.