Fixed Point Iteration for Computing the Time Elapse Operator

Sriram Sankaranarayanan[1,2], Henny B. Sipma[2], Zohar Manna[2] *

[1] NEC Laboratories America, Princeton, NJ
srirams@nec-labs.com
[2] Computer Science Department, Stanford University
(sipma,zm)@theory.stanford.edu

Abstract. We investigate techniques for automatically generating symbolic approximations to the time solution of a system of differential equations. This is an important primitive operation for the safety analysis of continuous and hybrid systems. In this paper we design a *time elapse operator* that computes a symbolic over-approximation of time solutions to a continous system starting from a given inital region. Our approach is iterative over the cone of functions (drawn from a suitable universe) that are non negative over the initial region. At each stage, we iteratively remove functions from the cone whose Lie derivatives do not lie inside the current iterate. If the iteration converges, the set of states defined by the final iterate is shown to contain all the time successors of the initial region. The convergence of the iteration can be forced using abstract interpretation operations such as widening and narrowing.
We instantiate our technique to linear hybrid systems with piecewise-affine dynamics to compute polyhedral approximations to the time successors. Using our prototype implementation TimePass, we demonstrate the performance of our technique on benchmark examples.

1 Introduction

An invariant is a predicate that holds on every reachable state of the system. By generating invariants, it is possible to prove a system safe or find potential bugs in systems. For discrete systems, the generation of invariants can be performed by a static analysis of the system; forward propagation is used to explore the reachable states of the system starting from the initial states of the system until an over-approximation of the reach set is generated, excluding the unsafe region. This idea has been explored for restricted classes of hybrid systems by popular tools such as Hytech, DDT, CheckMate and Charon.

To apply the forward propagation scheme to hybrid systems, we need a *time elapse operator*; an operator that, given an initial region Θ and a vector field D describing the continuous dynamics, computes an over-approximation of the

* This research was supported in part by NSF grants CCR-01-21403, CCR-02-20134, CCR-02-09237, CNS-0411363, and CCF-0430102, by ARO grant DAAD19-01-1-0723, and by NAVY/ONR contract N00014-03-1-0939.

time successors of Θ under D for a potentially infinite time horizon. The construction of the reachable set consists of alternate applications of such a time elapse operator for each mode, along with the standard *post condition (image)* operator for discrete mode changes. To be useful, this time elapse operator must be as accurate as possible. This paper presents a novel method to construct such a time elapse operator.

Our method iteratively constructs a set of functions $\{f_1, \ldots, f_m\}$ over the system variables drawn from a given universe of functions U, such that the corresponding assertion $f_1 \geq 0 \wedge \ldots \wedge f_m \geq 0$ holds for all time successors of Θ. We start with the set of all functions in U that are nonnegative over Θ, and then iteratively remove those functions whose Lie derivative with respect to the system's vector field D does not support the corresponding invariant assertion until a fixed point is reached. We show that the fixed point is guaranteed to correspond to an invariant assertion. Standard techniques from abstract interpretation such as widening and narrowing [6, 7] are used to force convergence in a finite number of steps to a set of functions that are guaranteed to be nonnegative on all the time successors of Θ.

The method is presented as a general framework, parameterized by an abstract *refinement operator* that performs the removal of functions from the set at each iteration. Specialization of the refinement operator allows the method to be applied to different function domains, thus generating different types of inequalities. To illustrate the method we describe a concrete instance of the framework for the domain of affine functions, providing an alternative way of polyhedral analysis. We have implemented this approach in our prototype tool TimePass with encouraging results over benchmark examples.

Related Work

The time elapse operator can be analytically computed for polyhedral initial regions and piecewise constant dynamics. The computation is *hard* for linear systems and *harder* for nonlinear systems. The polyhedral flowpipe approximation approach of Krogh et al. can solve the bounded time elapse problem for arbitrary differential equations. The approach has been implemented in their tool CheckMate [19] and used for complex systems with both linear and nonlinear dynamics. The DDT system due to Dang et al. uses orthogonal polyhedra and face lifting to compute the time elapse [1]. The PHAVer tool due to Frehse [10] presents a technique for the safety analysis of linear system using a sophisticated flowpipe construction for linear differential equations. Nevertheless, these technique can approximate flowpipes only upto a time bound. They also rely on numerical integration using ODE solvers to solve a hard non convex optimization problem numerically. Piazza et al. [14] and Ratschan et al. [16] propose approximations to the time-elapse based on quantifier elimination over the reals along with Taylor series expansions.

The time solutions can be symbolically computed for certain affine systems. However, the solution typically contains terms involving exponentiations, sines and cosines. It is computationally expensive to draw inferences from these results.

Extracting polyhedral over-approximations from the solution of linear systems is a formidable challenge. The work of Lafferriere et al. [13] and Tiwari [20] present interesting techniques for proving safety by integrating the dynamics of the system. Recently, symbolic techniques for generating invariants without the use of an explicit time elapse operator have been proposed, including the generation of nonlinear equality invariants for systems with polynomial dynamics [18, 21, 17]. These techniques can handle interesting nonlinear systems beyond the reach of traditional automatic techniques, but the theory has so far been restricted to equality invariants. Prajna and Jadbabaie [15] propose a method for the synthesis of barrier functions (inequalities) to justify invariants of nonlinear systems using convex optimization. These barrier functions are generated by solving equations on the unknown coefficients of a parametric polynomial; in contrast, in this paper we iteratively compute a set of functions starting from the initial region.

2 Preliminaries

Let $\mathcal{R}$ denote the set of reals. A function $f : \mathcal{R}^n \mapsto \mathcal{R}$ is said to be *smooth* if it is continuous and differentiable to any degree. Examples of such functions include polynomials and other analytical functions. Throughout this paper, we consider assertions $\varphi : \bigwedge_{i=1}^m f_i \geq 0$, such that each $f_i : \mathcal{R}^n \mapsto \mathcal{R}$ is smooth.

Let $\varphi : \bigwedge_{i=1}^m f_i \geq 0$ be such an assertion. We denote the set of values satisfying φ by $[[\varphi]]$, i.e, $[[\varphi]] = \{\boldsymbol{x} \in \mathcal{R}^n \mid \varphi(\boldsymbol{x})\}$. An assertion φ_1 *semantically entails* φ_2, written $\varphi_1 \models \varphi_2$ iff $[[\varphi_1]] \subseteq [[\varphi_2]]$.

Definition 1 (Cone). *Let $G = \{f_1, \ldots, f_i, \ldots\}$ be a set of smooth functions. The cone generated by G is given by*

$$Cone(G) = \left\{ \lambda_0 + \sum_{i=1}^{N} \lambda_i f_i \mid \lambda_i \geq 0,\ 0 \leq i \leq N,\ N > 0 \right\}.$$

Each $f_i \in G$ is said to be a generator *of the cone. The cone I is said to be finitely generated iff $I = Cone(G)$ for some finite set G. Given an assertion $\varphi : \bigwedge_{i=1}^m f_i \geq 0$, the expression $Cone(\varphi)$ denotes $Cone(\{f_1, \ldots, f_m\})$.*

A cone I defines a region $[[I]] = \{\boldsymbol{x} \in \mathcal{R}^n \mid f_i(\boldsymbol{x}) \geq 0,\ \forall f_i \in I\}$. Given cones I, J, note that $I \subseteq J$ iff $[[J]] \subseteq [[I]]$.

Lemma 1. *Given $\varphi : \bigwedge_{i=1}^m f_i \geq 0$, if $g \in Cone(\varphi)$ then $\varphi \models (g \geq 0)$.*

Proof. If $\boldsymbol{x} \in [[\varphi]]$, then $f_i(\boldsymbol{x}) \geq 0$ for $1 \leq i \leq m$. Also, if $g \in Cone(\varphi)$, then $g = \lambda_0 + \sum_i \lambda_i f_i$ for $\lambda_i \geq 0$. Hence, $g(\boldsymbol{x}) = \sum_i \lambda_i f_i(\boldsymbol{x}) \geq 0$.

Example 1. Consider $J = Cone(\varphi : x \geq 0 \wedge y \geq 0)$. We note that $3x + 4y \in J$. Thus $\varphi \models 3x + 4y \geq 0$. On the other hand, $\varphi \models x^2 + y \geq 0$. However, $x^2 + y \notin J$; $Cone(\varphi)$ is not necessarily a complete set of consequences.

The intersection of two cones is also a cone. However, the union of two cones fails to be a cone. We define the *conic hull* $I_1 \uplus I_2$, to be the smallest cone containing $I_1 \cup I_2$. Let $I_1 = Cone(f_1, \ldots, f_k)$ and $I_2 = Cone(g_1, \ldots, g_m)$, then $I_1 \uplus I_2 = Cone(f_1, \ldots, f_k, g_1, \ldots, g_m)$. Therefore, $Cone(\varphi_1 \wedge \varphi_2) = Cone(\varphi_1) \uplus Cone(\varphi_2)$

Continuous and Hybrid Systems

A vector field D over $\mathcal{R}^n$ associates each point $\boldsymbol{x}$ with a direction $D(\boldsymbol{x}) \in \mathcal{R}^n$. Given a system of differential equations of the form $\dot{x_i} = f_i(x_1, \ldots, x_n)$, we associate a vector field $D(\boldsymbol{x}) = \langle f_1(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x})\rangle$.

Definition 2 (Continuous System). *A continuous system $\langle V, D(\cdot), X, \Theta\rangle$ consists of a set of real-valued continuous variables V, such that $|V| = n$; a vector field $D(\cdot)$ over $\mathcal{R}^n$ defining the dynamics of the system; an invariant predicate (domain) X restricting the state space of the system and an initial region Θ such that $[[\Theta]] \subseteq [[X]]$.*

A *time trajectory* of a continuous system is a function $\tau : [0, \delta) \mapsto \mathcal{R}^n$ for some time $\delta > 0$ such that

(a) $\tau(0) \in [[\Theta]]$,
(b) $\tau(t) \in [[X]]$, for all $t \in [0, \delta)$, and
(c) $\dot{\tau}(t) = D(\tau(t))$.

Definition 3 (Lie Derivative). *Let $V(\boldsymbol{x}) = \langle p_1(\boldsymbol{x}), \ldots, p_n(\boldsymbol{x})\rangle$ be a vector field in $\mathcal{R}^n$. Let $f : \mathcal{R}^n \mapsto \mathcal{R}$ be continuous and differentiable. The* Lie derivative *of f over V is given by $\mathcal{L}_V(f) = (\nabla f) \cdot V(\boldsymbol{x}) = \sum_{i=1}^{n} \frac{\partial f}{\partial x_i} \cdot p_i(\boldsymbol{x})$.*

Let τ be some time trajectory of a continuous system with dynamics given by $D(\cdot)$. Consider the function $u(t) = f(\tau(t))$. The time derivative $\dot{u}(t)$ is given by the Lie derivative $\mathcal{L}_D(f)$ evaluated at $\boldsymbol{x} = \tau(t)$.

Hybrid systems generalize continuous systems by providing finitely many modes, each with possibly different dynamics and discrete mode changes. A state is *reachable* if it occurs in some computation. The set of all reachable states of a hybrid system is denoted $\mathsf{Reach}(H)$. The safety analysis problem given a safe set S, asks if $\mathsf{Reach}(H) \subseteq S$. Alternatively, the reachability problem given an unsafe set U, decides $\mathsf{Reach}(H) \cap U = \emptyset$.

The safety analysis problem is undecidable for a general hybrid system. In practice, it is solved by generating an over-approximation of the set $\mathsf{Reach}(H)$, also known as an *invariant.* The standard technique for generating invariants is based on a symbolic simulation of the system using assertions to represent sets of states. These techniques require the fundamental primitive of computing *time elapse* on a given region.

Definition 4 (Time Elapse Problem). *Given a system $\langle V, D(\cdot), X, \Theta\rangle$, compute an assertion ψ that contains all the time trajectories of the system starting from any state $\boldsymbol{x}_0 \in [[\Theta]]$.*

This problem is hard to solve in general. However, for restricted cases such as piecewise constant differential equations and polyhedral assertions, there have been many successful approaches to approximating the time elapse operation. In this paper, we provide a general iterative approach to computing the time elapse operator.

3 Algorithm

We first present the general framework to construct the time elapse operator without the use of invariant regions. After specializing this framework for the domain of affine functions, we refine the general framework to the case of systems with invariant regions. Proofs of theorems have been omitted in this version. They may be obtained in an extended version of this paper.

3.1 General Framework

Let $\boldsymbol{x}$ be a vector of system variables, $\Theta: \ f_1 \geq 0 \wedge \ \cdots \ \wedge \ f_m \geq 0$ be the initial region. Differential equations $\dot{x_i} = p_i(\boldsymbol{x}),\ 1 \leq i \leq n$ specify the dynamics. Recall that the dynamics induce a vector field D such that $D(\boldsymbol{x}) = \langle p_1(\boldsymbol{x}), \ldots, p_n(\boldsymbol{x})\rangle$. Assume $p_1, \ldots, p_n$ Lipschitz continuous.

Let U be a class of continuous and differentiable functions. Typically, U is suggested by the class of inequalities $f_i \geq 0$ that appear in the system description and those inequalities sought as potential invariants. We assume that U is a vector space of functions, i.e, closed under addition of functions and scaling by a real. Examples of U include the set of all affine functions over $\boldsymbol{x}$, the set of all polynomials of degree at most k, and the set of all polynomials.

We shall begin by formulating the notion of invariants that over-approximate the reachable states of the continuous system. The time elapse operator that we seek is nothing but a process of computing such invariants automatically.

Definition 5 (Bounding Invariant). *An assertion* $\varphi: \ g_1 \geq 0 \ \wedge \ \cdots \wedge g_m \geq 0$ *is a* bounding invariant *iff (a)* $\Theta \models \varphi$ *and (b)* $g_1 \geq 0 \ \wedge \cdots \wedge \ \mathbf{g_i} = \mathbf{0} \ \wedge \ \cdots \wedge \ g_m \geq 0 \models \mathcal{L}_D(g_i) > 0$*, for all* $1 \leq i \leq m$.

Bounding invariants contain the time trajectories of the system starting from Θ.

Lemma 2 (Soundness). *If* φ *is a bounding invariant then all time trajectories starting from* $\boldsymbol{x}_0 \in [[\varphi]]$ *satisfy* φ*.*

Proof. Sketch: Consider a time trajectory $\tau(t)$ such that $g_j(\tau(t)) < 0$ for some $t > 0$. Initially, each $g_i(\tau(0)) \geq 0$ for each i. A *violation* for g_i (if it exists) is a time interval $(t_i, t_i + \Delta_i)$ such that $g_i(\tau(t)) \geq 0$ for $0 \leq t \leq t_i$ and $g_i(\tau(t)) < 0$ for $t_i < t < t_i + \Delta$.

Let g_j be the "earliest" violation, i.e. t_j is minimal (simultaneous violations are not a problem). Thus, at time t_j, $g_k(\tau(t_j)) \geq 0$ for all k. Since $g_j(\tau(t))$ is smooth, we obtain $g_j(\tau(t_j)) = 0$, and since $g_j(\tau(t_j + \Delta t)) < 0$, we have that $\dot{g_j}(\tau(t_j)) \leq 0$, leading to a contradiction.

The set of all functions that are bounding invariants need not be convex. For instance, if $g_1 \geq 0$ and $g_2 \geq 0$ are bounding invariants, then $g_1 + g_2 \geq 0$ is invariant but not necessarily a bounding invariant. The notion of a *relaxed invariant* provides a stronger condition that is convex.

Definition 6 (Relaxed Invariant). *An assertion* $\varphi : \bigwedge_{i=1}^{m} g_i \geq 0$ *is a relaxed* invariant *for a* scale factor $\lambda \in \mathcal{R}$ *iff (a)* $\Theta \models \varphi$ *and (b)* $\varphi \models \mathcal{L}_D(g_i) + \lambda g_i > 0$, *for each* $1 \leq i \leq m$.

Lemma 3. *If* φ *is a relaxed invariant then it is also a bounding invariant.*

Proof. Since $\varphi \models \mathcal{L}_D(g_i) + \lambda g_i > 0$, it follows that $(\varphi \ \wedge \ g_i = 0) \models \mathcal{L}_D(g_i) > 0$.

We now extend the notion of a relaxed invariant to a cone of functions.

Definition 7 (Invariant Cone). *Let* $I = Cone(\{g_1, \ldots, g_m\})$ *be a finitely generated cone of functions such that* $I \subseteq U$. *Let* $\lambda \in \mathcal{R}$ *be a scale factor. We say that* I *is an* invariant cone *iff it satisfies the initiation and closure condition*

(1) Initiation: $I \subseteq Cone(\Theta)$, *(thus* $[[\Theta]] \subseteq [[I]]$*),*

(2) Lie derivative closure: $(\forall\ f \in I)\ (\exists \epsilon > 0)\ (\mathcal{L}_D(f) + \lambda f - \epsilon \in I)$.

Lemma 4 (Soundness). *Let* $I = Cone(\{g_1, \ldots, g_m\})$ *be an invariant cone for scale factor* λ. *The assertion* $\varphi :\ g_1 \geq 0 \wedge \ldots \wedge g_m \geq 0$ *is a relaxed invariant.*

Proof. Sketch: For each g_i, $\mathcal{L}(g_i) + \lambda g_i - \epsilon \in I$ for some $\epsilon > 0$, we have that $\varphi :\ \bigwedge_i g_i \geq 0 \models \mathcal{L}(g_i) + \lambda g_i - \epsilon \geq 0 \models \mathcal{L}(g_i) + \lambda g_i > 0$. Thus φ is a relaxed invariant, and hence an invariant.

Lemma 5. *If* I_λ *is an invariant cone for scale factor* λ, *then it is also invariant for any scale factor* $\mu \geq \lambda$.

Proof. For any $f \in I_\lambda$, $\mathcal{L}(f) + \lambda f - \epsilon \in I_\lambda$, for $\epsilon > 0$. Also, for $\mu \geq \lambda$, $(\mu - \lambda) f \in I_\lambda$. Therefore, $\mathcal{L}(f) + \mu f - \epsilon \in I_\lambda$ and thus I_λ is invariant for any scale factor $\mu \geq \lambda$.

The key computational step in our scheme is that of a refinement operator:

Definition 8 (Refinement Operator). *Given a cone* I*, a vector field* D *and a* scale factor λ*, we define the set*

$$\partial_\lambda I = \{ f \in U \mid \lambda f + \mathcal{L}_D(f) - \epsilon \in I,\ \epsilon > 0 \}\ .$$

Thus, $\partial_\lambda I$ *consists of all the functions* $f \in U$ *such that* $\mathcal{L}_D(f) + \lambda f - \epsilon \in I$.

The notion of an invariant may be recast as follows: A cone I is invariant for scale factor λ iff $I \subseteq Cone(\Theta) \cap \partial_\lambda I$. Consider the monotonic function $\mathfrak{F}_\lambda$ over cones, defined by $\mathfrak{F}_\lambda(I) = I \cap \partial_\lambda I \cap Cone(\Theta)$. A cone I is said to be a *fixed point* for $\mathfrak{F}_\lambda$ iff $\mathfrak{F}_\lambda(I) = I$.

Theorem 1. *Given a cone I, and the refinement operator ∂_λ,*

1. *$\partial_\lambda I$ is a cone.*
2. *The function $\mathfrak{F}_\lambda(I) = I \cap (\partial_\lambda I) \cap (Cone(\Theta))$ is monotonic and decreasing in the lattice of cones ordered by set inclusion, i.e., $\mathfrak{F}_\lambda(I) \subseteq I$.*
3. *If $\mathfrak{F}_\lambda(I) = I$, i.e., I is a fixed point of $\mathfrak{F}_\lambda$, then I is an invariant cone.*
4. *If $\lambda \leq \mu$ then $\partial_\lambda I \subseteq \partial_\mu I$. Thus, $\mathfrak{F}_\lambda(I) \subseteq \mathfrak{F}_\mu(I)$.*

The space of all cones $I \subseteq U$ forms a complete lattice and furthermore, $\mathfrak{F}_\lambda$ is a monotonic function. Tarski's theorem (see [8]) guarantees the existence of a greatest fixed point of $\mathfrak{F}_\lambda$: $I_\lambda^* = \bigcap_{i \geq 0} \partial_\lambda^i Cone(\Theta)$.

If I_λ^* is finitely generated then its generators correspond to an invariant assertion. Note that $I_\lambda^* \subseteq I_\mu^*$ for $\mu \geq \lambda$. Thus, it follows that $[[I_\mu^*]] \subseteq [[I_\lambda^*]]$. A larger value of λ, yields a stronger invariant.

In practice, the greatest fixed point is frequently not computable and even when it can be analytically computed, it may not be finitely generated. Therefore, we seek fixed points that are not necessarily the greatest fixed points. Note that such fixed points are also guaranteed to be invariant cones. This is performed by under-approximating I_λ^* as the limit of the following iteration:

$$\begin{aligned} I^{(0)} &= U \\ I^{(i+1)} &= \mathfrak{F}_\lambda(I^{(i)}) = I^{(i)} \cap (\partial_\lambda I^{(i)}) \cap Cone(\Theta) \end{aligned}$$

It follows from the monotonicity of $\mathfrak{F}_\lambda$ that each $I^{(i+1)} \subseteq I^{(i)}$. The iteration converges if $I^{(i+1)} = I^{(i)}$. If convergence occurs in finitely many steps then the result is a fixed point. Additionally, if the result is finitely generated then it is also an invariant. On the other hand, convergence is not guaranteed in all domains. Therefore, we use the narrowing operator $\triangle$ to force convergence [6, 7].

Definition 9 (Narrowing Operator [6, 7]). *Let I_1, I_2 be two cones such that $I_1 \supseteq I_2$. The narrowing $I_1 \triangle I_2$ is a cone defined as follows*

1. *$I_1 \triangle I_2 \subseteq I_1 \cap I_2 \subseteq I_2$.*
2. *Given any monotonically decreasing sequence $I_0 \supseteq I_1 \supseteq \cdots$, the sequence $J_0 = I_0$, $J_i = J_{i-1} \triangle I_i$ converges in finitely many steps.*

The convergence of the iterative strategy to some fixed point can now be ensured by repeated application of narrowing. For instance, consider the strategy

$$\begin{aligned} I_0 &= U \\ I_{j+1} &= \mathfrak{F}_\lambda(I_j), \qquad \text{if } 0 \leq j \leq K \\ I_{k+1} &= I_k \triangle \mathfrak{F}(I_k), \text{ if } k > K \end{aligned}$$

This strategy known as the *naive iteration* computes the regular iteration sequence until a fixed limit K. If convergence is not achieved within this bound, the repeated application of the narrowing operator guarantees convergence. Starting from a finitely generated cone $Cone(\Theta)$, and forcing convergence in finitely many steps (either naturally or through narrowing), we are guaranteed a finitely generated invariant cone I.

3.2 Polyhedral Analysis of Affine Systems

As a concrete instance of the framework defined in Section 3.1, we now present algorithms for the special case when the universe is the set of all affine expressions $\boldsymbol{c}^T\boldsymbol{x} + c_0$, the initial set Θ is a polyhedron of the form $A\boldsymbol{x} + \boldsymbol{b} \geq 0$, and the dynamics are affine, of the form $\dot{\boldsymbol{x}} = P\boldsymbol{x} + \boldsymbol{q}$. The Lie derivative is given by $\mathcal{L}_D(\boldsymbol{c}^T\boldsymbol{x} + c_0) = \boldsymbol{c}^T P\boldsymbol{x} + \boldsymbol{c}^T\boldsymbol{q}$.

Definition 10 (Finitely generated (polyhedral) cones). *A cone $I \subseteq U$ is said to be* finitely generated *iff $I = Cone(g_1, \ldots, g_m)$. The functions $g_1, \ldots, g_m$ are said to be its* generators*. Let I be a finitely generated cone of affine expressions. We may represent I in the form of a polyhedron $I = \{\boldsymbol{c}^T\boldsymbol{x} + c_0 \mid A\boldsymbol{c} \geq 0\}$, for a $m \times (n+1)$ matrix A.*

More generally, the coefficients of each expression in I satisfy a linear constraint of the form $A\boldsymbol{c} \geq 0$. Note that the vector $\boldsymbol{c}$ contains coefficient c_i for variables x_i along with the coefficient c_0 for the constant term.

Example 2. Consider the set of all affine expressions with nonnegative coefficients. We may represent such a set as

$$N = \{\boldsymbol{c}^T\boldsymbol{x} + c_0 |\ c_0 \geq 0 \ \wedge \ c_1 \geq 0 \ \wedge \ \cdots \ c_n \geq 0\} \ .$$

This set is finitely generated by the expressions $\{x_1, x_2, \ldots, x_n\}$. Consider the assertion $\Theta : \ x = 0 \ \wedge \ y \geq 0 \ \wedge \ y \leq 1$. We may represent $Cone(\Theta)$ in two ways:

$$\begin{aligned} Cone(\Theta) &= Cone(\{x, \ -x, \ y, \ 1-y\}) \\ &= \{c_0 + c_1 x + c_2 y \mid c_0 \geq 0 \ \wedge \ c_0 + c_2 \geq 0\} \end{aligned}$$

Conversion between representations is achieved through a vertex enumeration.

The refinement operator can be computed in a straightforward manner for finitely generated cones of affine expressions, as suggested by the following lemma

Lemma 6. *Let $I = \{\boldsymbol{c}^T\boldsymbol{x} + c_0 \mid \varphi(\boldsymbol{c}, c_0)\}$ and $\epsilon > 0$. The refinement $\partial_\lambda I$ for a field $D(\boldsymbol{x}) = P\boldsymbol{x} + \boldsymbol{q}$ is a finitely generated cone given by*

$$\partial_\lambda I = \{\boldsymbol{c}^T\boldsymbol{x} + c_0 \mid \varphi(P^T\boldsymbol{c} + \lambda\boldsymbol{c}, \ \lambda c_0 + \boldsymbol{q}^T\boldsymbol{c} - \epsilon)\} \, .$$

Proof. The Lie derivative $\mathcal{L}_D(\boldsymbol{c}^T\boldsymbol{x} + c_0) = \boldsymbol{c}^T P\boldsymbol{x} + \boldsymbol{q}^T\boldsymbol{c}$. Therefore, given an expression $f : \ \boldsymbol{c}^T\boldsymbol{x} + c_0 \in U$, $(\mathcal{L}_D(f) + \lambda f - \epsilon) \in I : \ \{\boldsymbol{c}^T\boldsymbol{x} + c_0 \mid \varphi(\boldsymbol{c}, c_0)\}$ iff $\psi : \ \varphi(\lambda\boldsymbol{c} + P^T\boldsymbol{c}, \ \lambda c_0 + \boldsymbol{q}^T\boldsymbol{c} - \epsilon)$ holds.

Note: The set $I = \{\boldsymbol{c}^T\boldsymbol{x} + c_0 \mid A(\boldsymbol{c}, c_0)^T + \boldsymbol{b} \geq 0\}$ is convex but not a cone unless $\boldsymbol{b} = 0$. If ϵ were given a fixed value such as 0.001 in our theory, the resulting constraints after refinement are not homogeneous. In theory, we introduce ϵ as a new variable and eliminate it from the final result. This is common in polyhedral libraries implementing strict inequalities.

The intersection of two sets $\{\boldsymbol{c}^T\boldsymbol{x} + c_0 \mid \varphi_1\}$ and $\{\boldsymbol{c}^T\boldsymbol{x} + c_0 \mid \varphi_2\}$ is given by $\{\boldsymbol{c}^T\boldsymbol{x} + c_0 \mid \varphi_1 \wedge \varphi_2\}$. We have now defined all the basic primitives needed to carry out the fixed point iteration for this domain.

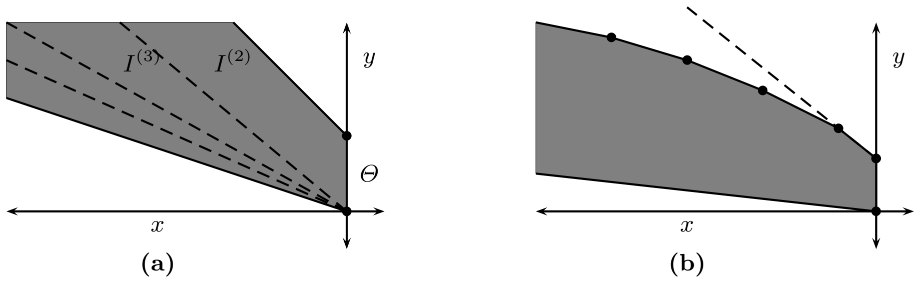

Fig. 1. Fixed points for Example 3: (a) $\lambda = 0$, (b) $\lambda = 1$. The shaded figure represents the final fixed point (not to scale). Dashed line in (b) represents upper solid line in (a).

Table 1. Iterates for Examples 3 for $\lambda = 0$.

#	constraints	generators
1	$c_0 \geq 0,\ c_0 + c_2 \geq 0$	$x = 0,\ 0 \leq y \leq 1$
2	$c_0 \geq 0,\ c_0 + c_2 \geq 0,\ 2c_2 - c_1 \geq \epsilon$	$x \leq 0, y \geq 0, 0 \leq 2x + y \leq 1$
3	$c_0 \geq 0,\ c_0 + c_2 \geq 0,$ $2c_2 - c_1 \geq \epsilon,\ 5c_2 - 4c_1 \geq 0$	$x \leq 0, y \geq 0,$ $2x + y \leq 1, 4y + 5x \geq 0$
4	$c_0 \geq 0,\ c_0 + c_2 \geq 0,$ $2c_2 - c_1 \geq \epsilon,\ 14c_2 - 13c_1 \geq 0$	$x \leq 0, y \geq 0,$ $2x + y \leq 1, 13y + 14x \geq 0$
$\vdots$	$\vdots$	$\vdots$
∞	$c_0 \geq 0,\ c_0 + c_2 \geq 0,$ $2c_2 - c_1 \geq \epsilon, c_2 - c_1 \geq 0$	$x \leq 0, y \geq 0,$ $2x + y \leq 1,$ $y + x \geq 0$

Example 3. Consider the system $\dot{x} = 2x - y,\ \dot{y} = -x + 2y$. We perform the iterator for scale factor $\lambda = 0$. The Lie derivative of an expression $c_0 + c_1 x + c_2 y$ is given by $(2c_1 - c_2)x + (2c_2 - c_1)y$. Consider the initial region $\Theta :\ x = 0,\ 0 \leq y \leq 1$.

$$Cone(\Theta) = \{c_0 + c_1 x + c_2 y \mid c_0 \geq 0 \ \wedge\ c_0 + c_2 \geq 0\}\ .$$

Let $I^{(0)} = U$, the set of all affine expressions. It follows that $\partial_0 I^{(0)} = U$. Therefore, $I^{(1)} = \mathfrak{F}(I) = Cone(\Theta) \cap (\partial_0 I^{(0)}) \cap I^{(0)} = Cone(\Theta)$.

$$\begin{aligned} \partial_0 I^{(1)} &= \partial_0 \{c_0 + c_1 x + c_2 y \mid c_0 \geq 0 \ \wedge\ c_0 + c_2 \geq 0\} \\ &= \{c_0 + c_1 x + c_2 y \mid 2c_2 - c_1 \geq \epsilon\} \\ I^{(2)} &= \{c_0 + c_1 x + c_2 y \mid c_0 \geq 0 \ \wedge\ c_0 + c_2 \geq 0 \ \wedge\ 2c_2 - c_1 \geq \epsilon,\ \epsilon > 0\} \end{aligned}$$

Table shows the cones encountered along the iteration, visualized in Figure 1(a). The fixed point I_∞ is not reached in finitely many iterations. The following table shows the fixed points for different values of the scale factor λ. Convergence was forced by the narrowing heuristics described below. Figure 1(b) depicts the fixed point for the case $\lambda = 1$.

λ	I_λ fixed point generators
-1	$x + y \geq 0,\ x \leq 0$
1	$x + y \geq 0,\ x \leq 0,\ 2x + y \leq 1,\ 7x + 5y \leq 8,$ $13x + 11y \leq 32, 25x + 23y \leq 128, 49x + 47y \leq 512$
2	$x + y \geq 0, x \leq 0, 2x + y \leq 1, 6x + 4y \leq 5,$ $56x + 44y \leq 75, 536x + 464y \leq 1125$

Note that the invariant for a larger value of λ subsumes that for a smaller value. In general, the iteration does not necessarily terminate in a finite number of steps. Furthermore, the resulting cone I may not be finitely generated. Therefore, approximations in the form of narrowing are required to force termination in a finite number of steps. For any two finitely generated cones, it is possible to define a *standard narrowing* by dropping generators [6].

Definition 11 (Standard Narrowing). *Consider two cones $I_1 = Cone(g_1, \ldots, g_m)$ and $I_2 = Cone(h_1, \ldots, h_k)$ such that $I_1 \supseteq I_2$. The standard narrowing $I = I_1 \triangle I_2$ is defined as $I = Cone(g_i \mid g_i \in I_2)$. In other words, the standard narrowing drops from I_1 all those generators that do not belong to the cone I_2.*

Each application of the standard narrowing results either in convergence or the removal of at least one generator from the first argument. This guarantees convergence of the naive iteration strategy in finitely many steps.

Example 4. Consider two successive iterates from Example 3.

$$I_1 = \begin{pmatrix} c_0 \geq 0,\ c_0 + c_2 \geq 0, \\ 2c_2 - c_1 \geq 0,\ 11c_2 - 10c_1 \geq 0 \end{pmatrix} = Cone(x, -y, 2x + y - 1, 10y + 11x)$$

$$I_2 = \begin{pmatrix} c_0 \geq 0,\ c_0 + c_2 \geq 0, \\ 2c_2 - c_1 \geq 0,\ 33c_2 - 32c_1 \geq 0 \end{pmatrix} = Cone(x, -y, 2x + y - 1, 32x + 33y)$$

Note that all but one generator $(10x + 11y)$ in I_1 also belong to I_2. Therefore, narrowing drops this generator resulting in

$$I_1 \triangle I_2 = \begin{pmatrix} c_0 \geq 0,\ c_1 \leq 0, \\ c_0 + c_2 \geq 0,\ 2c_2 - c_1 \geq 0 \end{pmatrix} = Cone(x, -y, 2x + y - 1)$$

3.3 Adding Invariant Regions

We now extend the general framework by considering the evolution restricted to an invariant region of the form $X : h_1 \geq 0\ \wedge\ h_2 \geq 0\ \cdots h_k \geq 0$ such that $\Theta \models X$. Let $J = Cone(X) = Cone(h_1, \ldots, h_k)$. As in Section 3.1, we assume that U is a universe, Θ is the initial condition and D is a differential field, with the refinement operator ∂_λ. Furthermore, we assume that $h_1, \ldots, h_k \in U$.

Definition 12 (Invariant Cone). *Let $I = Cone(\{g_1, \ldots, g_m\})$ be a finitely generated cone of functions such that $I \subseteq U$. We fix a scale factor $\lambda \in \mathcal{R}$. We say that I is an* invariant cone *under the invariant region $J = Cone(X)$ iff it satisfies the initiation and closure condition*

Algorithm 1 Algorithmic scheme for computing Time Elapse

K: Number of steps of initial iteration. λ: Appropriate value of λ for refinement.
Narrow: function implementing narrowing scheme for forcing convergence
function **ComputeTimeElapse**(Θ : predicate, D: dynamics, ψ: invariant)
$I(0) := \mathsf{ConeOfConsequences}(\Theta)$ { Form the initial cone by dualization }
$J := \mathsf{ConeOfConsequences}(\psi)$ { Form the cone for the invariant region}
for $i = 1$ to K **do**
  {Initial iteration for K steps}
  $I(i) := (I(i-1) \cap \mathsf{Refinement}(I(i-1), D, \lambda)) \uplus J$
end for
$I := I(K)$
{Start Narrowing to enforce convergence}
repeat
  $I' := (I \cap \mathsf{Refinement}(I, D, \lambda)) \uplus J$ {Refine I w.r.t. dynamics. $\mathsf{assert}(I' \subseteq I)$}
  $I := \mathsf{Narrow}(I, I')$ { Narrow successive iterations. $\mathsf{assert}(I \subseteq I')$}
until $I' \equiv I$
$\mathsf{return}[[I]]$

(1) Initiation: $I \subseteq Cone(\Theta) \uplus J$,

(2) Lie derivative closure: $(\forall\ f \in I)\ (\exists\ \epsilon > 0)\ (\mathcal{L}_D(f) + \lambda f - \epsilon \in I \uplus J)$.

Lemma 7 (Soundness). *Let* $\tau : [0, \delta)$ *be any time trajectory starting from* $\boldsymbol{x}_0 \in [[\Theta]]$*, under the vector field* D *and the invariant region* X*. Let* I *be an invariant cone (under* X*). It follows that for all* $t \in [0, \delta)$*,* $g_i(\tau(t)) \geq 0$.

We now extend the iterative solution in the presence of an invariant region cone J. Let us assume a fixed scale factor λ. Let $\mathfrak{F}_X(I) = J \uplus (\mathfrak{F}_\lambda(I))$. Also, let $I^{(0)} = U$ be the initial iterate. We refine each iterate using

$$I^{(i+1)} = \mathfrak{F}_X(I^{(i)}) = J \uplus (I^{(i)} \cap (\partial_\lambda I^{(i)}) \cap Cone(\Theta))\,.$$

$\mathfrak{F}_X$ is monotonic, and furthermore, its fixed point is an invariant under the region X. Therefore, as before, we may use the iterative technique with heuristic narrowing to force convergence. Algorithm 1 depicts the computation.

Given two cones $I_1 = \{\boldsymbol{c}^T\boldsymbol{x} + c_0 | A_1\boldsymbol{c} \geq 0\}$ and $I_2 = \{\boldsymbol{c}^T\boldsymbol{x} + c_0 | A_2\boldsymbol{c} \geq 0\}$, their union is given by $I_1 \uplus I_2 = \{\boldsymbol{c}^T\boldsymbol{x} + c_0 \mid (A_1\boldsymbol{c} \geq 0) \sqcup (A_2\boldsymbol{c} \geq 0)\}$, where $\sqcup$ denotes the polyhedral convex hull of two polyhedra.

Hybrid Systems Analysis The time elapse operator presented so far can be used as a primitive to perform approximate reachability analysis of hybrid systems. While the time elapse operator is used inside each mode to compute the time successors, the standard *post condition* operator is used to compute the image of a set of reachable states under a discrete transition. Algorithm 2 presents the use of our time elapse operator for the analysis of hybrid system. This algorithm is widely used in the analysis of hybrid systems [12]. Note that the convergence of this algorithm is not guaranteed for all hybrid systems. It is possible to modify

Algorithm 2 Compute reach-set for hybrid system using time elapse operator

list *wlist* : worklist consisting of unprocessed modes and predicates.
map *reachmap* : maps each location to its current *reachability predicate.*

function **analyze-hybrid-system** { compute reachable state (predicates) }
{initialize the worklist and reachmap}
$wlist := \{\langle m_{initial}, \Theta \rangle\}$ {add initial mode and start predicate to the worklist}
($\forall$ mode m) $reahmap(m) := false$
{initial reachable region is empty for each mode}
while $wlist \neq \emptyset$ **do**
  $\langle m, \varphi \rangle := \mathsf{pop}(wlist)$ {pop an unprocessed mode/predicate from worklist.}
  if $[[\varphi]] \not\subseteq [[reachmap(m)]]$ **then**
    {if the states in the popped predicate have not already been visited}
    $\mathsf{visit}(m, \varphi)$ {process unvisited states}
  end if
end while
{No more unprocessed predicates. Hence, all states have been visited.}
end function {analyze-hybrid-system}

function **visit**(m :mode, φ : predicate) {mode m is entered with state set φ}
$\varphi' := \mathsf{computeTimeElapse}(\varphi, \mathsf{inv}(m), \mathsf{dynamics}(m))$
{apply time elapse operator to φ}
$reachmap(m) := reachmap(m) \vee \varphi'$
{add φ' to reachmap. If φ' is polyhedral, $\vee$ may be approximated by convex hull. }
for all $\tau : m \to m'$ outgoing discrete transitions of mode m **do**
  $\psi' := \mathsf{post}(\varphi', \tau)$ {compute post condition}
  $wlist := wlist \cup \langle m', \psi' \rangle$ { enqueue new $\langle$location, predicate$\rangle$ pair}
end for
end function {visit}

Algorithm 2 to use widening/narrowing along the lines of Algorithm 1 (see [11]). However the loss in precision due to widening could make it less useful in practice. We have implemented the algorithm for systems with affine dynamics and polyhedral guards/invariants maps in our prototype tool TimePass.

3.4 Discussion

The technique, so far, has many parameters that need to be adjusted for its proper working. We list a few important issues that arise in practice.

Narrowing. Repeated applications of standard narrowing guarantees the termination of the iteration. However, the standard narrowing is a poor strategy for forcing convergence. More heuristic strategies such as extrapolation are based on "guessing" the ultimate limits of an iteration. For instance, the evolution of a generator across successive iterations $4y + 5x,\ 13y + 14x,\ 40y + 41x,\ \cdots$ observed in Example 3 suggests the limit $x+y$, leading us to the actual fixed point. The problem of designing precise narrowing/widening operators for polyhedral

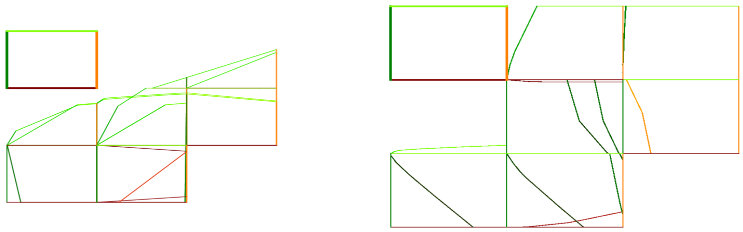

Fig. 2. Reachable regions for benchmarks NAV-04 (left) and NAV-06(right). Top left rectangle shows the unsafe region.

iterates has received a lot of attention in the (discrete) program analysis community [4, 11, 2]. Our own narrowing strategy maps generators across successive iterates using a distance metric such as the euclidean distance. It then guesses the ultimate limit as a weighted sum of the mapped pairs of generators rounded to a fixed precision limit.

Iteration Scheme. There has been a significant amount of work in the program analysis community on choosing iteration schemes for fixed point iterations. In this paper, however, we choose the "naive iteration" scheme (Algorithm 1). The scheme uses a pre-determined number (K) of initial iteration steps followed by narrowing/extrapolation until convergence. The value of K needs to be sufficiently large to allow our extrapolation scheme to guess the right limit to the iteration. However, higher values of K lead to large and complex polyhedra.

Choosing λ. In theory, a larger value of λ produces a stronger invariant. This may fail to hold due to the approximate nature of the narrowing operator. Nevertheless, the result holds in most examples encountered in practice. Unfortunately, a larger value of λ yields extremely complex cones with large coefficients in its representation. In practice, we perform many time elapses in stages, starting from a coarser grained approximation with smaller values of λ and improving using larger values. Such an approach provides means of focusing our narrowing heuristic at each stage to perform no worse than the previous one. For linear systems, using $\lambda = 0$ discovers the rays (infinite directions) of the time elapse operator, useful for computing time elapses over infinite time horizons.

4 Applications

Our tool TIMEPASS implements the algorithms described in this paper for the case of hybrid automata with affine dynamics and updates. Mode invariants, transition guards and initial regions are all assumed polyhedral. Our implementation is based on the Parma Polyhedral Library [2]. The library uses exact arithmetic to represent the coefficients of polyhedra. By default, the reachable set is represented as a list of polyhedra. However, it is possible to speed convergence of Algorithm 2 by using a single polyhedron per location.

Table 2. Resource utilization for the NAV benchmark examples.

#	$\lambda = 0, 10$			$\lambda = 0, 100$		
	Time	Mem (Mb)	Proved	Time	Mem (Mb)	Proved
NAV-01	4.4s	2.1	Yes	1m28s	5.2	Yes
NAV-02	1m13s	5.2	Yes	20m12s	18	Yes
NAV-03	1m18s	5	Yes	17m51s	16	Yes
NAV-04	19m51s	16	Yes	$\geq 45m$	≥ 60	No
NAV-05	2m39s	8.5	No	11m49s	30	No
NAV-06	$\geq 45m$	≥ 35	No	12m14s	21	No

Example 5. We consider the NAV benchmark examples standardized by Fehnker and Ivančić [9]. These benchmarks consist of an object moving through rectangular cells on a plane, each with different target velocities. Instances of these benchmarks have been standardized and are available online[3]. We refer the reader to this online repository for a detailed description. For each benchmark, we allowed our tool 45 minutes to converge. Figure 2 depicts the final reach sets computed for NAV-04 and NAV-06. In each case, the square at the top left corner is the forbidden region whose unreachability needs to be proven. Table 2 shows the running times and memory consumption recorded on an Intel Pentium III laptop with 512 Mb RAM. We were able to prove unreachability of the forbidden region for benchmarks NAV-01 to NAV-04. For NAV-06, the entire forbidden region but for the right most corner of the forbidden region is unreachable. As expected, there is a performance penalty for a higher value of λ. However for the case of NAV-06, we observe a reversal of this trend. A more accurate time elapse operator forces convergence of Algorithm 2 faster for this case.

5 Conclusion

We have presented a general framework for over approximating the flowpipe of a continuous system given a starting region. We have provided an instance of this framework for affine systems and polyhedral approximations. Our technique is entirely symbolic and works by computing a greatest fixed point in the space of finitely generated cones. As an advantage, our technique can handle unbounded domains and construct approximations that hold without any time bounds. Our approach is independent of the eigenstructure of the equation. On the other hand, the technique presents many parameters, chiefly the "scale factor" involved in the iteration. A larger value provably yields a more precise answer at the cost of performance.

We have engineered a prototype TimePass for the analysis of affine hybrid systems using polyhedra to represent sets of states. Our initial results with

[3] see `http://www.cse.unsw.edu.au/~ansgar/benchmark/nav_inst.txt`

benchmarks results are encouraging. Better narrowing strategies and careful engineering should improve its performance on these benchmarks. We are also looking into other representations of cones such as ellipsoids and quadratic forms for handling non-linear systems. We hope to extend our technique to provide a stronger framework based on sum-of-squares and positive semidefinite cones rather than polyhedral cones.

Acknowledgments. We are grateful Franjo Ivančić and the reviewers for their comments. Thanks to the developers of the PPL library [3] and Aaron Bradley for the Mathematica interface to PPL.

References

1. Asarin, E., Dang, T., and Maler, O. The d/dt tool for verification of hybrid systems. In *Proc. 14^{th} Intl. Conference on Computer Aided Verification* (2002), vol. 2404 of *Lecture Notes in Computer Science*, Springer–Verlag, pp. 365–370.
2. Bagnara, R., Hill, P. M., Ricci, E., and Zaffanella, E. Precise widening operators for convex polyhedra. In *Static Analysis Symposium* (2003), vol. 2694 of *Lecture Notes in Computer Science*, Springer–Verlag, pp. 337–354.
3. Bagnara, R., Ricci, E., Zaffanella, E., and Hill, P. M. Possibly not closed convex polyhedra and the Parma Polyhedra Library. In *Static Analysis Symposium* (2002), vol. 2477 of *Lecture Notes in Computer Science*, Springer–Verlag, pp. 213–229.
4. Besson, F., Jensen, T., and Talpin, J.-P. Polyhedral analysis of synchronous languages. In *Static Analysis Symposium* (1999), vol. 1694 of *Lecture Notes in Computer Science*, pp. 51–69.
5. Chutinan, A., and Krogh, B. Computing polyhedral approximations to flow pipes for dynamic systems. In *Proceedings of IEEE Conference on Decision and Control* (1998), IEEE press.
6. Cousot, P., and 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* (1977), pp. 238–252.
7. Cousot, P., and Cousot, R. Comparing the Galois connection and widening/narrowing approaches to Abstract interpretation, invited paper. In *PLILP '92* (1992), vol. 631 of *Lecture Notes in Computer Science*, Springer–Verlag, pp. 269–295.
8. Davey, B. A., and Priestly, H. A. *Introduction to Lattices and Order.* Cambridge University Press, 1990.
9. Fehnker, A., and Ivančić, F. Benchmarks for hybrid systems verification. In *Hybrid Systems: Computation and Control (HSCC 2004)* (2004), vol. 2993 of *Lecture Notes in Computer Science*, Springer–Verlag, pp. 326–341.
10. Frehse, G. PHAVer: Algorithmic verification of hybrid systems past HyTech. In *Hybrid Systems: Computation and Control (HSCC 2005)* (2005), vol. 2289 of *Lecture Notes in Computer Science*, Springer–Verlag, pp. 258–273.
11. Halbwachs, N., Proy, Y., and Roumanoff, P. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design 11*, 2 (1997), 157–185.
12. Henzinger, T., and Ho, P.-H. Algorithmic analysis of nonlinear hybrid systems. In *Computer-Aided Verification*, P. Wolper, Ed., vol. 939 of *Lecture Notes in Computer Science*. Springer–Verlag, 1995, pp. 225–238.

13. Lafferriere, G., Pappas, G., and Yovine, S. Symbolic reachability computation for families of linear vector fields. *J. Symbolic Computation 32* (2001), 231–253.
14. Piazza, C., Antoniotti, M., Mysore, V., Policriti, A., Winkler, F., and Mishra, B. Algorithmic algebraic model checking i: Challenges from systems biology. In *Computer-aided Verification* (2005), vol. 3576 of *Lecture Notes in Computer Science*, Springer–Verlag, pp. 5–19.
15. Prajna, S., and Jadbabaie, A. Safety verification using barrier certificates. In *Hybrid Systems: Computation and Control* (2004), vol. 2993 of *Lecture Notes in Computer Science*, Springer–Verlag, pp. 477–492.
16. Ratschan, S., and She, Z. Safety verification of hybrid systems by constraint propagation based abstraction refinement. In *HSCC* (2005), vol. 3414 of *Lecture Notes in Computer Science*, Springer–Verlag, pp. 573–589.
17. Rodriguez-Carbonell, E., and Tiwari, A. Generating polynomial invariants for hybrid systems. In *Hybrid Systems: Computation and Control, HSCC 2005* (2005), vol. 3414 of *LNCS*, Springer, pp. 590–605.
18. Sankaranarayanan, S., Sipma, H. B., and Manna, Z. Constructing invariants for hybrid systems. In *Hybrid Systems: Computation and Control (HSCC 2004)* (march 2004), vol. 2993 of *Lecture Notes in Computer Science*, Springer–Verlag, pp. 539–555.
19. Silva, B., Richeson, K., Krogh, B. H., and Chutinan, A. Modeling and verification of hybrid dynamical system using checkmate. In *ADPM 2000* (2000). available online from `http://www.ece.cmu.edu/~webk/checkmate`.
20. Tiwari, A. Approximate reachability for linear systems. In *Hybrid Systems: Computation and Control HSCC* (2003), vol. 2623 of *Lecture Notes in Computer Science*, Springer–Verlag, pp. 514–525.
21. Tiwari, A., and Khanna, G. Non-linear systems: Approximating reach sets. In *Hybrid Systems: Computation and Control* (2004), vol. 2993 of *Lecture Notes in Computer Science*, Springer–Verlag, pp. 477–492.