Space-time Interpolants

Goran Frehse[1], Mirco Giacobbe[2], and Thomas A. Henzinger[2]

[1] Univ. Grenoble Alpes, CNRS, Grenoble INP, VERIMAG
[2] IST Austria

Abstract. Reachability analysis is difficult for hybrid automata with affine differential equations, because the reach set needs to be approximated. Promising abstraction techniques usually employ interval methods or template polyhedra. Interval methods account for dense time and guarantee soundness, and there are interval-based tools that overapproximate affine flowpipes. But interval methods impose bounded and rigid shapes, which make refinement expensive and fixpoint detection difficult. Template polyhedra, on the other hand, can be adapted flexibly and can be unbounded, but sound template refinement for unbounded reachability analysis has been implemented only for systems with piecewise constant dynamics. We capitalize on the advantages of both techniques, combining interval arithmetic and template polyhedra, using the former to abstract time and the latter to abstract space. During a CEGAR loop, whenever a spurious error trajectory is found, we compute additional space constraints and split time intervals, and use these *space-time interpolants* to eliminate the counterexample. Space-time interpolation offers a lazy, flexible framework for increasing precision while guaranteeing soundness, both for error avoidance and fixpoint detection. To the best of out knowledge, this is the first abstraction refinement scheme for the reachability analysis over *unbounded* and *dense* time of affine hybrid systems, which is both *sound* and *automatic*. We demonstrate the effectiveness of our algorithm with several benchmark examples, which cannot be handled by other tools.

1 Introduction

Formal verification techniques can be used to either provide rigorous guarantees about the behaviors of a critical system, or detect instances of violating behavior if such behaviors are possible. Formal verification has become widely used in the design of software and digital hardware, but has yet to show a similar success for physical and cyber-physical systems. One of the reasons for this is a scarcity of suitable algorithmic verification tools, such as model checkers, which are formally sound, precise, and scale reasonably well. In this paper, we propose a novel verification algorithm that meets these criteria for systems with piecewise affine dynamics. The performance of the approach is illustrated experimentally on a number of benchmarks. Since systems with affine dynamics have been studied before, we first describe why the available methods and tools do not handle this class of systems sufficiently well, and then describe our approach and its core contributions.

Previous approaches The algorithmic verification of systems with continuous or discrete-continuous (hybrid) dynamics is a hard problem both in theory and practice. For piecewise constant dynamics (PCD), the continuous successor states (a.k.a. flow pipe) can be computed exactly, and the complexity is exponential in the number of variables [17, 19]. While in principle, any dynamics can be approximated arbitrarily well by PCD systems using an approach called hybridization [20], this requires partitioning of the state space, which often leads to prohibitive computational costs. For piecewise affine dynamics (PWA), one-step successors can be computed approximately using complex set representations. However, all published approaches suffer either from a possibly exponential increase in the complexity of the set representation, or from a possibly exponential increase in the approximation error as the considered time interval increases; this will be argued in detail in Sect. 4.

In addition to these theoretical obstacles, we note the following practical obstacles for the available tools and their performance in experiments. The only available model checkers that are (i) *sound* (i.e., they compute provable dense-time overapproximations), (ii) *unbounded* (i.e., they overapproximate the flowpipe for an infinite time horizon), and (iii) *arbitrarily precise* (i.e., they support precision refinement) are, with one exception, limited to PCD systems, namely, HyTech [18], PHAVer [13], and Lyse [7]. The tool Ariadne [6] can deal with affine dynamics and is sound, unbounded, and precise. However, Ariadne discretizes the reachable state space with a rectangular grid. This invariably leads to an exponential complexity in terms of the number of variables. Other tools that are applicable to PWA systems do not meet our criteria in that they are either not formally sound (e.g., CORA [2], SpaceEx [15]), not arbitrarily precise because of templates or particular data structures (e.g., SpaceEx, Flow* [8], CORA), or limited to bounded model checking (e.g., dReach [24], Flow*). All the above tools exhibit fatal limitations in scalability or precision on standard PWA benchmarks; they typically work only on well-chosen examples. Note that while these tools do not meet the criteria we advance in this paper, they of course have strengths in other areas handling nonlinear and nondeterministic dynamics.

Our approach We view iterative abstraction refinement as critical for soundness and precision management, and fixpoint detection as critical for evaluating unbounded properties. We implement, for the first time, a CEGAR (counterexample-guided abstraction refinement) scheme in combination with a fixpoint detection criterion for PWA systems. Our abstraction refinement scheme manages complexity and precision trade-offs in a flexible way by decoupling time from space: the dense timeline is partitioned into a sequence of intervals that are refined individually and lazily, by splitting intervals, to achieve the necessary precision and detect fixpoints; state sets are overapproximated using template polyhedra that are also refined individually and lazily, by adding normal directions to templates; and both refinement processes are interleaved for optimal results, while maintaining soundness with each step. A similar approach was recently proposed for the limited class of PCA systems [7]; this paper can be seen as an extension of the approach to the class of piecewise affine dynamics.

With each iteration of the CEGAR loop, a spurious counterexample is removed by computing a proof of infeasibility in terms of a sequence of linear constraints in space and interval constraints in time, which we call a sequence of *space-time interpolants*. We use linear programming to construct a suitable sequence of space-time intervals and check for fixpoints. If a fixpoint check fails, we increase the time horizon by adding new intervals. The separation of time from space gives us the flexibility to explore different refinement strategies. Fine-tuning the iteration of space refinement (adding template directions), time refinement (splitting intervals), and fixpoint checking (adding intervals), we find that it is generally best to prefer fewer time intervals over fewer space constraints. Based on performance evaluation, we even expand individual intervals time when this is possible without sacrificing the necessary precision for removing a counterexample.

2 Motivating Example

The ordinary differential equation over the variables x and y

$$\begin{aligned} \dot{x} &= 0.1x - y + 1.8 \\ \dot{y} &= x + 0.1y - 2.2 \end{aligned} \tag{1}$$

moves counterclockwise around the point $(2, 2)$ in an outward spiral. We center a box B (of side 0.92) on the same point and place a diagonal segment S close to the bottom right corner of B, without touching it (between $(2, 1)$ and $(3.5, 2)$; see Fig. 1). Then, we consider the problem of proving that every trajectory starting from any point in S never hits B. This is a time-unbounded reachability problem for a hybrid automaton with piecewise affine dynamics and two control modes. The first mode has the dynamics above (Eq. 1) and S as initial region. It has a transition to a second mode, which in its turn has B as invariant. The second mode is a bad mode, which all trajectories indeed avoid.

We tackle the reachability problem by abstraction refinement. In particular, we aim at automatically constructing an enclosure for the flowpipe —i.e., for the set of trajectories from S— which (i) avoids the bad state B and (ii) covers the continuous timeline up to infinity. Figure 1 shows three abstractions that result from different strategies for refining an initial space partition (i.e., template) and time partition (i.e., sequence of time intervals). All three refinement schemes start by enclosing S with an initial template polyhedron P, and then transforming P into a sequence of abstract flowpipe sections $\text{intflow}^{[\underline{t},\overline{t}]}(P)$, one for each interval $[\underline{t}, \overline{t}]$ of an initial partitioning of the unbounded timeline. The computation of new flowpipe sections stops when a fixpoint is reached, —i.e., we reach a time threshold t^* whose flowpipe section closes a cycle with $\text{intflow}^{t^*}(P) \subseteq P$, sufficient condition for any further flowpipe section to be contained within the union of previously computed sections.

Refinement scheme (a) sticks to a fixed octagonal template P —i.e., to the normals of a regular octagon— and iteratively halves all time intervals until every flowpipe section avoids the bad set B. This is achieved at interval width $1/64$, but

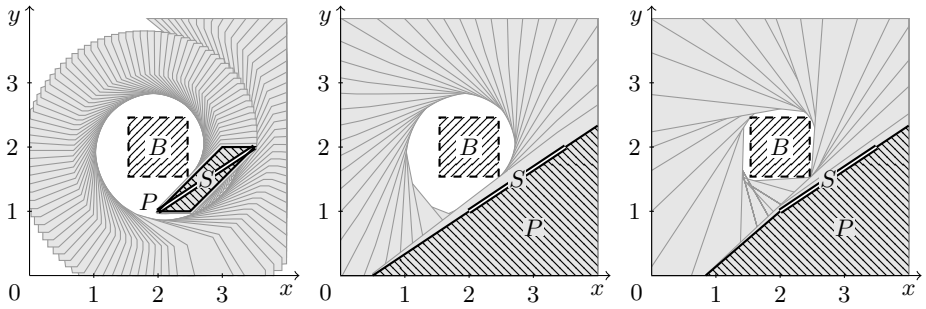

Fig. 1. Comparison of abstraction refinement methods for the ODE in Eq. 1, the segment S as initial region, and the box B as bad region. The polyhedron P is the template polyhedron of S, and the gray polyhedra are the flowpipe sections $\text{intflow}^{[\underline{t},\overline{t}]}(P)$.

the computation does not terminate because no fixpoint is reached. Refinement scheme (b) splits time similarly but also computes a different, more accurate template for every iteration: first, an interval $[\underline{t},\overline{t}]$ is halved until it admits a halfspace interpolant —i.e., a halfspace H that $S \subseteq H$ and $\text{intflow}^{[\underline{t},\overline{t}]}(H) \cap B = \emptyset$; then, a maximal set of linearly independent directions is chosen as template from the normals of the obtained halfspaces. Refinement scheme (b) succeeds at interval width $1/16$ to avoid B and reach a fixpoint; the latter at time 6.25, with $\text{intflow}^{6.25}(P) \subseteq P$. Refinement scheme (c) modifies (b) by optimizing the refinement of the time partition: instead of halving time intervals, the maximal intervals which admit halfspace interpolants are chosen. This scheme produces a nonuniform time partitioning with an average interval width of about $1/8$, discovers five template directions, and finds a fixpoint in fewer steps.

Each iteration of the abstraction refinement loop consists of first abstracting the initial region into a template polyhedron, second solving the differential equation into a sequence of interval matrices, and finally transforming the template polyhedron using each of the interval matrices. We represent each transformation symbolically, by means of its support function. Then, we verify (i) the separation between every support function and the bad region, and (ii) the containment of any support function in the initial template polyhedron. The separation problem amounts to solving one LP, and the inclusion problem amounts to solving an LP in each template direction. If the separation fails, then we independently bisect each time that does not admit halfspace interpolants and expand each that does, until all are proven separated. Together, these halfspace interpolants form an infeasibility proof for the counterexample: a space-time interpolant. We forward the resulting new time intervals and halfspaces to the abstraction generator, and repeat, using the refined partitioning and the augmented template. If the inclusion fails, then we increase the time horizon by some amount Δ, and repeat. Once we succeed with both separation and inclusion, the system is proved safe.

This example shows the advantage of lazily refining *both* the space partitioning (i.e., the template) by adding directions, and the time partitioning, by splitting intervals.

3 Hybrid Automata with Piecewise Affine Dynamics

A hybrid automaton with piecewise affine dynamics consists of an n-dimensional vector x of real-valued variables and a finite directed multigraph (V, E), the control graph. We call it the control graph, the vertices $v \in V$ the control modes, and the edges $e \in E$ the control switches. We decorate each mode $v \in V$ with an initial condition $Z_v \subseteq \mathbb{R}^n$, a nonnegative invariant condition $I_v \subseteq \mathbb{R}^n_{\geq 0}$, and a flow condition given by the system of ordinary differential equations

$$\dot{x} = A_v x + b_v. \tag{2}$$

We decorate each switch $e \in E$ with a guard condition $G_e \subseteq \mathbb{R}^n$ and an update condition given the difference equations $x := R_e x + s_e$. All constraints I, G, and Z are conjuctions of rational linear inequalities, A and R are constant matrices, and b and s constant vectors of rational coefficients. In this paper, whenever an indexing of modes and switches is clear from the context, we index the respective constraints and transformations similarly, e.g., we abbreviate A_{v_i} with A_i.

A trajectory is a possibly infinite sequence of states $(v, x) \in V \times \mathbb{R}^n$ repeatedly interleaved first by a switching time $t \in \mathbb{R}_{\geq 0}$ and then by a switch $e \in E$

$$(v_0, x_0) t_0 (v_0, y_0) e_0 (v_1, x_1) t_1 (v_1, y_1) e_1 \ldots \tag{3}$$

for which there exists a sequence of solutions $\psi_0, \psi_1, \ldots : \mathbb{R} \to \mathbb{R}_n$ such that $\psi_i(0) = x_i$, $\psi_i(t_i) = y_i$ and they satisfy (i) the invariant conditions $\psi_i(t) \in I_i$ and (ii) the flow conditions $\dot{\psi}_i(t) = A_i \psi_i(t) + b_i$, for all $t \in [0, t_i]$. Moreover, $x_0 \in Z_0$, every switch e_i has source v_i, destination v_{i+1}, and and the respective states satisfy (i) the guard condition $y_i \in G_i$ and (ii) the update $x_{i+1} = R_i y_i + s_i$. The maximal set of its trajectories is the semantics of the hybrid automaton, which is safe if none of them contains a special bad mode.

Every hybrid automaton with affine dynamics can be transformed into an equivalent hybrid automaton with linear dynamics, i.e., the special case where $b = 0$ on every mode. We obtain such transformation by adding one extra variable y, rewriting the flow of every mode into $\dot{x} = Ax + by$, and forcing y to be always equal to 1, i.e., invariant $y = 1$ and flow $\dot{y} = 0$ on every mode and update $y' = y$ on every switch. For this reason, in the following sections we discuss w.l.o.g. the reachability analysis of hybrid automata with linear dynamics.

4 Time Abstraction using Interval Arithmetic

We abstract the reach set of the hybrid automaton with a union of convex polyhedra. In particular, we abstract the states that are reachable in a mode

using a finite sequence of images of the initial region over a *time partitioning*, until a completeness threshold is reached. Thereafter, we compute the *template polyhedron* of each of the images that can take a switch. Then, we repeat in the destination mode and we continue until a fixpoint is found.

Precisely, a time partitioning T is a (possibly infinite) set of disjoint closed time intervals whose union is a single (possibly open) interval. For a finite set of directions $D \subseteq \mathbb{R}^n$, the D-polyhedron of a closed convex set X is the tightest polyhedral enclosure whose facets normals are in D. In the following, we associate every mode v to a template D_v and a time partitioning T_v of the time axis $\mathbb{R}_{\geq 0}$, we employ interval arithmetic for abstracting the continuous dynamics (Sec. 4.1), and on top of it we develop a procedure for hybrid dynamics (Sec. 4.2).

4.1 Continuous Dynamics

We consider w.l.o.g. a mode with ODE reduced to the linear form $\dot{x} = A_v x$, invariant I_v, and a given time interval $[\underline{t}, \overline{t}]$. Every linear ODE $\dot{x} = Ax$ has the unique solution

$$\psi(t) = \exp(At)\psi(0). \tag{4}$$

It follows (see also [16]) that the set of states reachable in v after exactly t time units from an initial region X is

$$\mathrm{flow}_v^t(X) \stackrel{\text{def}}{=} \exp(A_v t)X \cap \bigcap_{0 \leq \tau \leq t} \exp\big(A_v(t-\tau)\big) I_v, \tag{5}$$

Then, the flowpipe section over the time interval $[\underline{t}, \overline{t}]$ is

$$\mathrm{flow}_v^{[\underline{t},\overline{t}]}(X) \stackrel{\text{def}}{=} \cup\{\mathrm{flow}_v^t(X) \mid t \in [\underline{t}, \overline{t}]\}. \tag{6}$$

We note three straightforward but consequential properties of the reach set: (i) The accuracy of any convex abstraction depends on the size of the time interval: While $\mathrm{flow}_v^t(X)$ is convex for convex X, this is generally not the case for $\mathrm{flow}_v^{[\underline{t},\overline{t}]}(X)$. (ii) We can prune the time interval whenever we detect that the reach set no longer overlaps with the invariant: If for any $t^* \geq 0$, $\mathrm{flow}_v^{t^*}(X) = \emptyset$, then for all $\overline{t} \geq t^*$, $\mathrm{flow}_v^{\overline{t}}(X) = \emptyset$ and $\mathrm{flow}_v^{[\underline{t},\overline{t}]}(X) = \mathrm{flow}_v^{[\underline{t},t^*]}(X)$. (iii) We can prune the time interval whenever we detect containment in the initial states: If $\mathrm{flow}_v^{t^*}(X) \subseteq X$, then $\mathrm{flow}_v^{[\underline{t},\infty]}(X) = \mathrm{flow}_v^{[\underline{t},t^*]}(X)$.

For given A and t, the matrix $\exp(At)$ can be computed with arbitrary, but only finite, accuracy. We resolve this problem by computing a rational interval matrix $[\underline{M}, \overline{M}]$, which we denote $\mathrm{intexp}(A, \underline{t}, \overline{t})$, such that for all $t \in [\underline{t}, \overline{t}]$ we have element-wise that

$$\exp(At) \in \mathrm{intexp}(A, \underline{t}, \overline{t}). \tag{7}$$

This interval matrix can be derived efficiently with a variety of methods [25], e.g., using a guaranteed ODE solver or using interval arithmetic. The width of the interval matrix can be made arbitrarily small at the price of increasing the number of computations and the size of the representation of the rational

numbers. In our approach, we do not rely in a fixed accuracy of the interval matrix. Instead, we require that the accuracy increases as the width of the time interval goes to zero. That way, we don't need to introduce an extra parameter. To ensure progress in our refinement loop, we require that the interval matrix decreases monotonically when we split the time interval. Formally, if $[\underline{t}, \overline{t}] \subseteq [\underline{u}, \overline{u}]$ we require the element-wise inclusion $\text{intexp}(A, \underline{t}, \overline{t}) \subseteq \text{intexp}(A, \underline{u}, \overline{u})$. This can be ensured by intersecting the interval matrices with the original interval matrix after time splitting.

While the mapping with interval matrices is in general not convex [29], we can simplify the problem by assuming that all points of X are in the positive orthant. As long as X is bounded from below, this condition can be satisfied by inducing an appropriate coordinate change. Under the assumption that $X \subseteq \mathbb{IR}^n_{\geq 0}$,

$$[\underline{M}, \overline{M}](X) = \left\{ y \in \mathbb{IR}^n \;\middle|\; \underline{M}x \leq y \leq \overline{M}x \text{ and } x \in X \right\}. \tag{8}$$

Combining the above results, we obtain a convex abstraction of the flowpipe over a time interval as

$$\text{intflow}_v^{[\underline{t},\overline{t}]}(X) \stackrel{\text{def}}{=} \text{intexp}(A, \underline{t}, \overline{t})X \cap I_v. \tag{9}$$

The abstraction is conservative in the sense that $\text{flow}_v^{[\underline{t},\overline{t}]}(X) \subseteq \text{intflow}_v^{[\underline{t},\overline{t}]}(X)$. On the other hand, the longer is the time interval, the coarser is the abstraction. For this reason, we construct an abstraction of the flowpipe in terms of a union of convex approximations over a time partitioning. The abstract flowpipe over the time partitioning T is

$$\text{intflow}_v^T(X) \stackrel{\text{def}}{=} \cup\{\text{intflow}_v^{[\underline{t},\overline{t}]}(X) \mid [\underline{t}, \overline{t}] \in T\}. \tag{10}$$

Again, this is conservative w.r.t. the concrete flowpipe, i.e., for all time partitionings T it holds that $\text{flow}_v^{\cup T}(X) \subseteq \text{intflow}_v^T(X)$. Moreover, it is conservative w.r.t. any refinement of T, i.e., the time partitioning U refines T if $\cup U = \cup T$ and $\forall [\underline{u}, \overline{u}] \in U \colon \exists [\underline{t}, \overline{t}] \in T \colon [\underline{u}, \overline{u}] \subseteq [\underline{t}, \overline{t}]$, then $\text{intflow}_v^U(X) \subseteq \text{intflow}_v^T(X)$.

4.2 Hybrid Dynamics

We embed the flowpipe abstraction routine into a reachability algorithm that accounts for the switching induced by the hybrid automaton. The discrete post operator is the image of a set $Y \subseteq \mathbb{IR}^n$ through a switch $e \in E$

$$\text{jump}_e(Y) \stackrel{\text{def}}{=} R_e(Y \cap G_e) \oplus \{s_e\}. \tag{11}$$

We explore the hybrid automaton constructing a set of abstract trajectories, namely sequences abstract states interleaved by time intervals and switches

$$(v_0, X_0)[\underline{t}_0, \overline{t}_0](v_0, Y_0)e_0(v_1, X_1)[\underline{t}_1, \overline{t}_1](v_1, Y_1)e_1 \ldots \tag{12}$$

where $X_0, Y_0, \cdots \subseteq \mathbb{IR}^n$ are nonempty sets of states that comply with template $\{D_v\}$ and partitioning $\{T_v\}$ in the following sense. First, $X_0 = Z_0$ and $X_{i+1} =$

input : Template $\{D_v\}$ and partitioning $\{T_v\}$ indexed by V
output : Optionally an abstract trajectory (counterexample)

foreach $v \in V$ *with nonempty* Z_v **do**
  push $(v, Z_v)[0, \Delta]$ into the stack W;
  add the D_v-polyhedron of Z_v to P_v;
while W *is not empty* **do**
  pop $\dots (v, X)[\underline{t}, \overline{t}]$ from W;
  $P \leftarrow D_v$-polyhedron of X;
  if v *is bad and* $P \cap I_v$ *is nonempty* **then** `// check counterexample`
    return $\dots (v, X)$;
  foreach $t^* \in \{\underline{t} + \delta, \underline{t} + 2\delta, \dots, \overline{t}\}$ **do** `// find completeness threshold`
    if $\text{intflow}_v^{t^*}(P) \subseteq P_v$ **then break**;
  if $t^* = \overline{t}$ *and* $\text{intflow}_v^{\overline{t}}(P) \not\subseteq P_v$ **then** `// otherwise extend time horizon`
    push $\dots (v, X)[\overline{t}, \overline{t} + \Delta]$ into W;
  foreach $[\underline{u}, \overline{u}] \in T_v$ *and* $[\underline{u}, \overline{u}] \cap [\underline{t}, t^*] \neq \emptyset$ **do** `// construct flowpipe`
    $Y \leftarrow \text{intflow}_v^{[\underline{u}, \overline{u}]}(P)$;
    foreach $e \in E$ *with source* v *and destination* v' **do**
      $X' \leftarrow \text{jump}_e(Y)$;
      if $X' \subseteq P_{v'}$ **then continue**;
      push $\dots (v, X)[\underline{u}, \overline{u}](v, Y)e(v', X')[0, \Delta]$ into W;
      add the $D_{v'}$-polyhedron of X' to $P_{v'}$;

Algorithm 1: Reachability procedure.

$\text{jump}_i(Y_i)$ for all $i \geq 0$. Second, $Y_i = \text{intflow}_i^{[\underline{t}_i, \overline{t}_i]}(P_i)$ for all $i \geq 0$, where P_i is the D_i-polyhedron of X_i and $[\underline{t}_i, \overline{t}_i] \in T_i$. The maximal set of abstract trajectories, the abstract semantics induced by $\{D_v\}$ and $\{T_v\}$, overapproximates the concrete semantics in the sense that every concrete trajectory (see Eq. 3) has an abstract trajectory that subsumes it, i.e., modes and switches match, $x_i \in X_i$, $t_i \in [\underline{t}_i, \overline{t}_i]$, and $y_i \in Y_i$, for all $i \geq 0$.

Computing the abstraction involves several difficulties. First, the trajectories might be not finitary. Indeed, this is unsolvable in theory, because the reachability problem is undecidable [21]. Second, the post operators are hard to compute. In particular, obtaining the sets X and Y in terms of conjunctions of linear inequalities in $\mathbb{R}^n$ requires eliminating quantifiers. In Alg. 1, we present a procedure (which does not necessarily terminate) for tackling the first problem. In the next section, we show how to tackle the second using support functions.

We employ Alg. 1 to explore the tree of abstract trajectories. We store in the stack W the leaves to process $\dots (v, X)$, followed by a candidate interval $[\underline{t}, \overline{t}]$. For each leaf, we retrieve P, the template polyhedron of X. If it leads to a bad mode, we return, otherwise we search for a completeness threshold t^* between $\underline{t}$ excluded and $\overline{t}$, checking for inclusion in the union of visited polyhedra P_v. In case of failure, we extend the time horizon of Δ and push the next candidate to the stack. Then, we partition the time between $\underline{t}$ and t^*, construct the flowpipe, and process switching. Upon each successful switch, we augment $P_{v'}$ with the $D_{v'}$-polyhedron of the switching region X', avoiding to store redundant polyhedra. Notably, the latter operation is efficient because all polyhedra comply with the same template. For the same reason, we obtain efficient inclusion checks, which

we implement by first computing the template polyhedron of the left hand side, and then comparing the constant terms of the respective linear inequalities.

In conclusion, this reachability procedure that takes a template $\{D_v\}$ and a partitioning $\{T_v\}$ and constructs a tree of reachable sets of states X and Y. It manipulates them through the post operators and overapproximate them into template polyhedra. In the next section, we discuss how to efficiently represent X and Y, so to efficiently compute their template polyhedra. In Sec. 6 we discuss how to discover appropriate $\{D_v\}$ and $\{T_v\}$, so to eliminate spurious counterexamples.

5 Space Abstraction using Support Functions

Abstracting away time left us with the task of representing the state space of the hybrid automaton, namely the space of its variable valuations. Such sets consists of polyhedra emerging from operations such as intersections, Minkowski sums, and linear maps with simple or interval matrices. In this section, we discuss how to represent precisely all sets emerging from any of these operations by means of their support functions (Sec. 5.1) and then how to abstract them into template polyhedra (Sec. 5.2). In the next section, we discuss how to refine the abstraction.

5.1 Support Functions

The support function of a closed convex set $X \subseteq \mathbb{R}^n$ in direction $d \in \mathbb{R}^n$ consists of the maximizer scalar product of d over X

$$\rho_X(d) = \sup\{d^\mathsf{T} x \mid x \in X\}, \tag{13}$$

and, indeed, uniquely represents any closed convex set [28]. Classic work on the verification of hybrid automata with affine dynamic have posed a framework for the construction of support functions from basic set operations, but under the assumption of unboundedness and nonemptiness of the represented set, and with approximated intersection [16]. Indeed, if the set is empty then its support function is $-\infty$, while if it is unbounded an d points toward a direction of recession is $+\infty$, making the framework end up into undefined values. Such conditions turn out to be limiting in our context, first because we find desirable to represent unbounded sets so to accelerate the convergence to a fixpoint of the abstraction procedure, but most importantly because when encoding support functions for long abstract trajectories we might be not aware whether its concretization is infeasible. Checking this is a crucial element of a counterexample-guided abstraction refinement routine.

Recent work on the verification of hybrid automata with constant dynamics, i.e., with flows defined by constraints on the derivative only, provides us with a generalization of the classic support function framework which relaxes away the assumptions of boundedness and nonemptiness and yields precise intersection [7]. The framework encodes combinations of convex sets of states into LP (linear

programs) which enjoy strong duality with their support function. Similarly, we encode the support function in direction d of any set X into the LP

$$\begin{array}{l} \text{minimize} \;\; c^{\mathsf{T}}\lambda \\ \text{subject to } A\lambda = Bd, \end{array} \tag{14}$$

over the nonnegative vector of variables λ. The LP is dual to $\rho_X(d)$, which is to say that if the LP is infeasible then X is unbounded in direction d, and if the LP is unbounded then X is the empty set. Moreover, if the LP has bounded solution so does $\rho_X(d)$ and the solutions coincide.

The construction is inductive on operations between sets. For the base case, we recall that from duality of linear programming the support function of a polyhedron given by a system of inequalities $Px \leq q$ is dual to the LP over $\lambda \geq 0$

$$\begin{array}{l} \text{minimize} \;\; q^{\mathsf{T}}\lambda \\ \text{subject to } P^{\mathsf{T}}\lambda = d. \end{array} \tag{15}$$

Then, inductively, we assume that for the set $X \subseteq \mathbb{R}^n$ we are given an LP with the coefficients A_X, B_X, and c_X, and similarly for the set $Y \subseteq \mathbb{R}^n$. For the support functions of $X \oplus Y$, MX, and $X \cap Y$ we respectively construct the following LP over the nonnegative vectors of variables λ, μ, α, and β:

$$\begin{array}{l} \text{minimize} \;\; c_X^{\mathsf{T}}\lambda + c_Y^{\mathsf{T}}\mu \\ \text{subject to } A_X\lambda = B_X d \text{ and } A_Y\mu = B_Y d, \end{array} \tag{16}$$

$$\begin{array}{l} \text{minimize} \;\; c_X^{\mathsf{T}}\lambda \\ \text{subject to } A_X\lambda = B_X M^T d, \text{ and} \end{array} \tag{17}$$

$$\begin{array}{l} \text{minimize} \;\; c_X^{\mathsf{T}}\lambda + c_Y^{\mathsf{T}}\mu \\ \text{subject to } A_X\lambda - B_X(\alpha - \beta) = 0 \text{ and} \\ \qquad\qquad A_Y\mu + B_Y(\alpha - \beta) = B_Y d. \end{array} \tag{18}$$

Such construction follows as a special case of [7], which we extend with the support function of a map through an interval matrix.

The time abstraction of Sec. 4 additionally requires us to represent the map of sets of states through interval matrices. Precisely, we are given convex set of nonnegative values $X \subseteq \mathbb{R}^n_{\geq 0}$, the coefficients for the respective LP, an interval matrix $[\underline{M}, \overline{M}] \subseteq \mathbb{R}^{n \times n}$, and we aim at computing the support function of all values in X mapped by all matrices in $[\underline{M}, \overline{M}]$. To this end, we define the LP

$$\begin{array}{l} \text{minimize} \;\; c_X^{\mathsf{T}}\lambda \\ \text{subject to } A_X\lambda + B_X(\underline{M}^{\mathsf{T}}\mu - \overline{M}^{\mathsf{T}}\nu) = 0 \text{ and} \\ \qquad\qquad -\mu + \nu = d, \end{array} \tag{19}$$

over the vectors λ, μ, and ν of nonnegative variables. This linear program corresponds to the the dual of the interval matrix map in Eq. 8.

5.2 Computing Template Polyhedra

We represent all space abstractions X and Y in our procedure by their support functions. In particular, whenever set operations are applied, instead of solving

the operation by removing quantifiers, we construct an LP. We delay solving it until we need to compute a template polyhedron. In that case, we compute the D-polyhedron of the set X by computing its support function in each of the directions in D, and constructing the intersection of halfspaces $\cap\{d^\mathsf{T} x \leq \rho_X(d) \mid d \in D\}$.

6 Abstraction Refinement using Space-time Interpolants

The reachability analysis of hybrid automata by means of the combination of interval arithmetic and support functions presented in Sec. 4 and 5 builds an overapproximation of the system dynamics. It is always sound for safety, but it may produce spurious counterexamples, due to an inherent lack of precision of the time abstraction and the polyhedral approximation. The level of precision is given by two factors, namely the choice of time partitioning and the choice of template directions, excluding the parameters for approximation of the exponential function, which we assume constant (see Sec. 4.1). In the following, we present a procedure to extract infeasibility proofs from spurious counterexamples. We produce them in the form of time partitions and bounding polyhedra, which we call space-time interpolants. Space-time interpolants can then be used to properly refine time partitioning and template directions.

Consider the bounded path $v_0, e_0, v_1, e_1, \ldots, v_k, e_k, v_{k+1}$ over the control graph and a sequence of dwell time intervals $[\underline{t}_0, \overline{t}_0], [\underline{t}_1, \overline{t}_1], \ldots, [\underline{t}_k, \overline{t}_k]$ emerging from an abstract trajectory. We aim at extracting a sequence $X_0, X_1, \ldots, X_{k+1}$ of (possibly nonconvex) polyhedra and a sequence $T_0, T_1, \ldots, T_k$ of refinements of the respective dwell times such that $Z_0 \subseteq X_0$, $\mathrm{jump}_0 \circ \mathrm{intflow}_0^{T_0}(X_0) \subseteq X_1$, $\ldots$, $\mathrm{jump}_k \circ \mathrm{intflow}_k^{T_k}(X_k) \subseteq X_{k+1}$, and $X_{k+1} \cap I_{k+1}$ is empty. In other words, we want every X_{i+1} to contain all states that can enter mode v_{i+1} after dwelling on v_i between $\underline{t}_i$ and $\overline{t}_i$ time, and the last to be separated from the invariant of mode v_{k+1}. Containment is to hold inductively, namely X_{i+1} has to contain what is reachable from X_i, and the time refinements T are to be chosen in such a way that containment holds in the abstraction. Then, we call the sequence $X_0, T_0, X_1, T_1, \ldots, X_k, T_k, X_{k+1}$ a sequence of space-time interpolants for the path and the dwell times above.

We compute a sequence of space-time interpolants by alternating multiple strategies. First, for the given sequence of dwell times, we attempt to extract a sequence of halfspace interpolants using linear programming (Sec. 6.1). In case of failure, we iteratively partition the dwell times in sets of smaller intervals, separating nonswitching from switching times and until every combination of intervals along the path admits halfspace interpolants (Sec. 6.2). We accumulate all halfspaces to form a sequence of unions of convex polyhedra that, together with the obtained time partitionings, will form a valid sequence of space-time interpolants. Finally, we refine the abstraction using the time partitionings and the outwards pointing directions of all computed halfspaces, in order to eliminate the spurious counterexample (Sec. 6.3).

6.1 Halfspace Interpolation

Halfspace interpolants are the special case of space-time interpolants where every polyhedron in the sequence is defined by a single linear inequality [1]. Indeed, they are the simplest kind of space-time interpolants, and, for the same reason, the ones that best generalize the reachable states along the path. Unfortunately, not all paths admit halfspace interpolants, but, if one such sequence exists, then it can be extrapolated from the solution of a linear program.

Consider a path $v_0, e_0, \ldots, v_{k+1}$ with the respective dwell times $[\underline{t}_0, \overline{t}_0], \ldots, [\underline{t}_k, \overline{t}_k]$. A sequence of halfspace interpolants consists of a sequence of sets $H_0, \ldots, H_{k+1}$ among either any halfspace, or the empty set, or the universe, such that $Z_0 \subseteq H_0$, $\mathrm{jump}_0 \circ \mathrm{intflow}_0^{[\underline{t}_0, \overline{t}_0]}(H_0) \subseteq H_1$, $\ldots$, $\mathrm{jump}_k \circ \mathrm{intflow}_k^{[\underline{t}_k, \overline{t}_k]}(H_k) \subseteq H_{k+1}$, and $H_{k+1} \cap I_{k+1}$ is empty. In contrast with general space-time interpolants, every time partition consists of a single time interval and therefore the support function of every post operator $\mathrm{jump} \circ \mathrm{intflow}^{[\underline{t}, \overline{t}]}$ can be encoded into a single LP (see Sec. 5). We exploit the encoding for extracting halfspace interpolants, similarly to a recent interpolation technique for PCD systems [7].

We encode the support function in direction d of the closure of the image of the post operators along the path, i.e., the set $\mathrm{jump}_k \circ \mathrm{intflow}_k^{[\underline{t}_k, \overline{t}_k]} \circ \cdots \circ \mathrm{jump}_0 \circ \mathrm{intflow}_0^{[\underline{t}_0, \overline{t}_0]}(Z_0)$, intersected with the invariant I_{k+1}. We obtain the following LP over the free vectors $\alpha_0, \ldots, \alpha_{k+1}$ and the nonnegative vectors β, $\delta_0, \ldots, \delta_k$, $\gamma_0, \ldots, \gamma_{k+1}$, $\mu_0, \ldots, \mu_k$, and $\nu_0, \ldots, \nu_k$:

$$\begin{array}{lll}
\text{minimize} & q_{Z_0}^\mathsf{T}\beta + \sum_{i=0}^{k}(q_{I_i}^\mathsf{T}\gamma_i + q_{G_i}^\mathsf{T}\delta_i + s_i^\mathsf{T}\alpha_{i+1}) + q_{I_{k+1}}^\mathsf{T}\gamma_{k+1} & \\
\text{subject to} & P_{Z_0}^\mathsf{T}\beta = \alpha_0, & \\
& \underline{M}_i^\mathsf{T}\mu_i - \overline{M}_i^\mathsf{T}\nu_i = -\alpha_i & \text{for each } i \in [0..k], \\
& -\mu_i + \nu_i + P_{I_i}^\mathsf{T}\gamma_i + P_{G_i}^\mathsf{T}\delta_i = R_i^\mathsf{T}\alpha_{i+1} & \text{for each } i \in [0..k], \\
& P_{I_{k+1}}^\mathsf{T}\gamma_{k+1} = -\alpha_{k+1} + d, &
\end{array} \tag{20}$$

where every system of inequalities $Px \leq q$ corresponds to the constraints of the respective init, guard, or invariant, every $R_i x + s_i$ is an update equation, and every interval matrix $[\overline{M}_i, \underline{M}_i] = \mathrm{intexp}(A_i, \underline{t}_i, \overline{t}_i)$. In general, one can check whether the closure is contained in a halfspace $a^\mathsf{T}x \leq b$ by setting the direction to its linear term $d = a$ and checking whether the objective function can equal its constant term b. In particular, we check for emptiness, which we pose as checking inclusion in $0x \leq -1$. Therefore, we set $d = 0$ and the objective function to equal -1. Upon affirmative answer, from the solution $\alpha_0^\star, \alpha_1^\star, \ldots, \nu_k^\star$ we obtain a valid sequence of halfspace interpolants whose i-th linear term is given by $\alpha_i^\star$ and i-th constant term is given by $q_{Z_0}^\mathsf{T}\beta^\star + \sum_{j=0}^{i-1}(q_{I_j}^\mathsf{T}\gamma_j^\star + q_{G_j}^\mathsf{T}\delta_j^\star + s_j^\mathsf{T}\alpha_{j+1}^\star)$.

6.2 Time Partitioning

Halfspace interpolation attempts to compute a sequence of enclosures that are convex for a sequence of sets that are not necessarily convex. Specifically, it requires each halfspace to enclose the set of solutions of a linear differential equation,

input : sequence of intervals $[\underline{u}_0, \overline{u}_0], \ldots, [\underline{u}_j, \overline{u}_j]$
output : set of intervals

$b \leftarrow \underline{u}_j$;
while $b < \overline{u}_j$ **do**
  $a \leftarrow b$;
  $b \leftarrow b + \varepsilon$;
  $c \leftarrow \overline{u}_j$;
  if $[\underline{u}_0, \overline{u}_0], \ldots, [\underline{u}_{j-1}, \overline{u}_{j-1}], [a, b]$ *does not admit halfspace interpolants* **then**
    continue;
  if $[\underline{u}_0, \overline{u}_0], \ldots, [\underline{u}_{j-1}, \overline{u}_{j-1}], [a, c]$ *admits halfspace interpolants* **then**
    push $[a, c]$ to the output;
    return;
  while $c - b > \varepsilon$ **do**
    if $[\underline{u}_0, \overline{u}_0], \ldots, [\underline{u}_{j-1}, \overline{u}_{j-1}], [a, \varepsilon \lfloor \frac{b+c}{2\varepsilon} \rfloor]$ *admits halfspace interpolants* **then**
      $b \leftarrow \varepsilon \lfloor \frac{b+c}{2\varepsilon} \rfloor$;
    else
      $c \leftarrow \varepsilon \lfloor \frac{b+c}{2\varepsilon} \rfloor$;
  push $[a, b]$ to the output;

Algorithm 2: Nonswitching time partitioning.

which is nonconvex, by enclosing its convex overapproximation along a whole time interval. As a result, large time intervals produce large overapproximations, on which halfspace interpolation might be impossible. Likewise, shorter intervals produce tighter overapproximations, which are more likely to admit halfspace interpolants. In this section, we exploit such observation to enable interpolation over large time intervals. In particular, we properly partition the time into smaller subintervals and we treat each of them as a halfspace interpolation problem. Later, we combine the results to refine the abstraction.

Time partitioning is a delicate task in the whole abstraction refinement loop. In fact, while template refinement affects linearly the performance of the abstractor, partitioning time intervals that can switch induces branching in the search, possibly leading to an exponential blowup. For this reason, we partition time by narrowing down the switching time, for incremental precision, until no more is left. In particular, we use Alg. 2 to compute a set N of maximal intervals that admit halfspace interpolants, by enlarging or narrowing them of ε amounts. We embed this procedure in Alg. 3 which, along the sequence, excludes the time in N, constructing a set of intervals S that overapproximate the switching time. In particular, we construct the set with the widest possible intervals that are disjoint from N. Algorithm 3 succeeds when no more intervals are left, otherwise we half ε and reapply it to the sequences that are left to process.

6.3 Abstraction Refinement

The procedures above construct sequences of time intervals $[\underline{u}_0, \overline{u}_0], \ldots, [\underline{u}_j, \overline{u}_j]$ that are included in $[\underline{t}_0, \overline{t}_0], \ldots, [\underline{t}_k, \overline{t}_k]$ and that, with the respective halfspace interpolants, this constitutes a proof of infeasibility for the counterexample. Yet, it does not form a sequence of space-time interpolants $X_0, T_0, \ldots, X_{k+1}$.We form each partitioning T_i by splitting $[\underline{t}_i, \overline{t}_i]$ in such a way each element of T_i is either

```
input  : sequence of intervals [t̲_0, t̄_0], ..., [t̲_k, t̄_k]
output : set of sequences of intervals
push [t̲_0, t̄_0] to the queue Q;
while Q is not empty do
    pop [u̲_0, ū_0], ..., [u̲_j, ū_j] from Q;
    N ← nonswitching time partitioning of [u̲_0, ū_0], ..., [u̲_j, ū_j];
    foreach [a̲, ā] ∈ N do
        push [u̲_0, ū_0], ..., [u̲_{j-1}, ū_{j-1}], [a̲, ā] to the output;
    if j = k then
        assert [u̲_j, ū_j]\ ∪ N = ∅;
        continue;
    S ← choose set of intervals that cover [u̲_j, ū_j]\ ∪ N;
    foreach [b̲, b̄] ∈ S do
        push [u̲_0, ū_0], ..., [u̲_{j-1}, ū_{j-1}], [b̲, b̄], [t̲_{j+1}, t̄_{j+1}] to Q;
```

Algorithm 3: Dwell time partitioning.

contained in $[\underline{u}_i, \overline{u}_i]$ or disjoint from it, for all intervals $[\underline{u}_i, \overline{u}_i]$. Then, we refine the partitioning of mode v_i similarly. Each polyhedron X_i is a union of convex polyhedra, each of which is the intersection of all halfspaces H_i corresponding to some sequence $[\underline{u}_0, \overline{u}_0], \ldots, [\underline{u}_i, \overline{u}_i]$. Nevertheless, to refine the abstraction we do not need to construct X_i, but just to take the outward point directions of all H_i and add them to the template of v_i.

7 Experimental Evaluation

We implemented our method in `C++` using GMP and Eigen for multiple precision linear algebra, Arb for interval arithmetic, and PPL for linear programming [23, 5]. In particular, all libraries we are using are meant to provide guaranteed solutions, as well as our implementation. We evaluate it on several instances of a *filtered oscillator* and a *rod reactor*, which are both parametric in the number of variables, and the latter in the number of modes too [15, 35]. We record several statistics from every execution of our tool: the number #cex of counterexamples found during the CEGAR loop, the number #dir of linearly independent directions and the average width of the time partitionings extracted from all space-time interpolants. Moreover, we independently measure three times. First, the time spent in finding counterexamples, namely the total time taken by inconclusive abstractions which returned a spurious counterexample. Second, the refinement time, that is the total time consumed by computing space-time interpolants. Finally, the verification time, that is the time spend in the last abstraction of the CEGAR loop, which terminates with a fixpoint proving the system safe. We compare the outcome and the performance of our tool against Ariadne which, to the best of our knowledge, is the only verification tool available that is numerically sound and time-unbounded [11].

The filtered oscillator is hybrid automaton with four modes that smoothens a signal x into a signal z. It has $k+2$ variables and a system of $k+2$ affine ODE, where k is the order of the filter. Table 1 shows the results, for a scaling of k

	# vars	# modes	# cex	# dirs	avg. width	cex. time	ref. time	ver. time	tot. time	Ariadne
`filtosc_1st_ord`	3	4	7	13	0.55	0.57	0.96	0.13	**1.66**	27.56
`filtosc_2nd_ord`	4	4	7	15	0.55	0.83	1.78	0.20	**2.81**	150.7
`filtosc_3rd_ord`	5	4	7	16	0.55	1.28	4.65	0.32	**6.25**	oot
`filtosc_4th_ord`	6	4	7	18	0.55	1.53	11.39	0.37	**13.29**	oot
`filtosc_5th_ord`	7	4	7	19	0.55	2.61	26.60	0.70	**29.37**	-
`filtosc_6th_ord`	8	4	7	18	0.55	4.56	101.8	1.29	**107.7**	-
`filtosc_7th_ord`	9	4	7	18	0.55	4.36	109.9	1.13	**114.6**	-
`filtosc_8th_ord`	10	4	7	17	0.55	5.92	150.9	1.54	**158.4**	-
`filtosc_9th_ord`	11	4	7	16	0.55	6.49	383.1	1.83	**391.3**	-
`filtosc_10th_ord`	12	4	7	17	0.55	12.84	428.87	3.73	**445.4**	-
`filtosc_11th_ord`	13	4	7	17	0.55	15.10	525.2	4.38	**544.6**	-
`reactor_1_rod`	2	4	11	3	0.11	5.24	10.64	1.59	**17.47**	oot
`reactor_2_rods`	3	5	9	7	0.79	5.68	5.36	2.33	**13.37**	oot
`reactor_3_rods`	4	6	12	13	1.07	14.46	13.94	13.13	**41.53**	-
`reactor_4_rods`	5	7	15	29	1.67	45.50	42.47	111.5	**199.9**	-
`reactor_5_rods`	6	8	16	31	1.81	73.77	27.36	696.46	**797.5**	-

Table 1. Statistics for the benchmark examples (oot when > 1000s).

up to the 11-th order. The first observation is that the CEGAR loop behaves quite similarly on all scalings: number of counterexamples, number of directions, and time partitionings are almost identical. On the other hand, the computation times show a growth, particularly in the refinement phase which dominates over abstraction and verification. This suggests us that our procedure exploits efficiently the symmetries of the benchmark. In particular, time partitioning seems unaffected. What affects the performance is linear programming, whose size depends on the number of variables of the system.

The rod reactor consists of a heating reactor tank and k rods each of which cools the tank for some amount of time, excluding each other. The hybrid automaton has one variable x for the temperature, k clock variables, one heating mode, one error mode, and k cooling modes. If the temperature reaches a critical threshold and no rod can intervene, it goes into an error. For this benchmark, we start with a simple template, the interval around x, and we discover further directions. Table 1 highlights two fundamental differences with the previous benchmark. First, the average width grows with the model size. This is because the heating mode requires finer time partitioning than the cooling modes. The cooling modes increase with the number of rods, and so does the average width over all time partitions. Second, while with the filtered oscillator the difficulty laid at interpolation, for the rod reactor interpolation is rather easy as well as finding counterexamples. Most of the time is spent in the verification phase, where all fixpoint checks must be concluded, without being interrupted by a counterexample. This shows the advantage of our lazy approach, which first processes the counterexamples and finally proves the fixpoint.

Our method outperforms Ariadne on all benchmarks. On the other hand, tools like Flow* and SpaceEx can be dramatically faster [9]. For instance, they analyze `filtosc_8th_ord` in resp. 9.1s and 0.36s (time horizon of 4 and jump depth of 10). This is hardly surprising, as our method has primarily been designed to comply with soundness and time-unboundedness, and pays the price for that.

8 Related Work

There is a rich literature on CEGAR approaches for hybrid automata, either abstracting to a purely discrete system [3, 10, 27, 33, 34] or to a hybrid automaton with simpler dynamics [22, 30]. Both categories exploit the principle that the verification step is easier to carry out in the abstract domain. The abstraction entails a considerable loss of precision that can only be counteracted by increasing the number of abstract states. This leads to a state explosion that severely limits the applicability of such approaches. In contrast, our approach allows us to increase the precision by adding template directions, which does not increase the number of abstract states. The only case where we incur additional abstract states is when partitioning the time domain. This is a direct consequence of the nonconvexity of flowpipes of affine systems, and therefore seems to be unavoidable when using convex sets in abstractions. In [26], the abstraction consists of removing selected ODE entirely. This reduces the complexity, but does not achieve any fine-tuning between accuracy and complexity. Template reachability has been shown to be very effective in both scaling up reachability tasks to more efficient successor computations [32, 31, 15] and achieving termination even over unbounded time horizons [12]. The drawback of templates is the lack of accuracy, which may lead to an approximation error that accumulates excessively. Efforts to dynamically refine templates have so far not scaled well for affine dynamics [14]. A single-step refinement was proposed in [4], but as was illustrated in [7], the refinement needs to be inductive in order to exclude counterexamples in a CEGAR scheme.

9 Conclusion

We have developed an abstraction refinement scheme that combines the efficiency and scalability of template reachability with just enough precision to exclude all detected paths to the bad states. At each iteration of the refinement loop, only one template direction is added per mode and time-step. This does not increase the number of abstract states. Additional abstract states are only introduced when required by the nonconvexity of flowpipes of affine systems, a problem that we consider unavoidable. In contrast, existing CEGAR approaches for hybrid automata tend to suffer from state explosion, since refining the abstraction immediately requires additional abstract states. As our experiments confirm, our approach results in templates over very low complexity and terminates with an unbounded proof of safety after a relatively small number of iterations. Further research is required to extend this work to nondeterministic and nonlinear dynamics.

Acknowledgments

We thank Luca Geretti for helping us setting up Ariadne. This research was supported in part by the Austrian Science Fund (FWF) under grants S11402-N23(RiSE/SHiNE) and Z211-N23 (Wittgenstein Award), by the European Commission under grant 643921 (UnCoVerCPS).

References

[1] Albarghouthi, A., McMillan, K.L.: Beautiful interpolants. In: Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings. pp. 313–329 (2013)

[2] Althoff, M.: An introduction to cora 2015. In: Frehse, G., Althoff, M. (eds.) ARCH14-15. 1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems. EPiC Series in Computer Science, vol. 34, pp. 120–151. EasyChair (2015)

[3] Alur, R., Dang, T., Ivančić, F.: Counterexample-guided predicate abstraction of hybrid systems. Theoretical Computer Science 354(2), 250–271 (2006)

[4] Asarin, E., Dang, T., Maler, O., Testylier, R.: Using redundant constraints for refinement. In: International Symposium on Automated Technology for Verification and Analysis. pp. 37–51. Springer (2010)

[5] Bagnara, R., Hill, P.M., Zaffanella, E.: The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. Science of Computer Programming 72(1–2), 3–21 (2008)

[6] Benvenuti, L., Bresolin, D., Collins, P., Ferrari, A., Geretti, L., Villa, T.: Assume-guarantee verification of nonlinear hybrid systems with Ariadne. Int. J. Robust. Nonlinear Control 24(4), 699–724 (2014)

[7] Bogomolov, S., Frehse, G., Giacobbe, M., Henzinger, T.A.: Counterexample-guided refinement of template polyhedra. In: Tools and Algorithms for the Construction and Analysis of Systems , TACAS 2017. pp. 589–606 (2017)

[8] Chen, X., Ábrahám, E., Sankaranarayanan, S.: Taylor model flowpipe construction for non-linear hybrid systems. In: RTSS'12. pp. 183–192 (2012)

[9] Chen, X., Schupp, S., Makhlouf, I.B., Ábrahám, E., Frehse, G., Kowalewski, S.: A benchmark suite for hybrid systems reachability analysis. In: NASA Formal Methods Symposium. pp. 408–414. Springer (2015)

[10] Clarke, E., Fehnker, A., Han, Z., Krogh, B., Ouaknine, J., Stursberg, O., Theobald, M.: Abstraction and counterexample-guided refinement in model checking of hybrid systems. International journal of foundations of computer science 14(04), 583–604 (2003)

[11] Collins, P., Bresolin, D., Geretti, L., Villa, T.: Computing the evolution of hybrid systems using rigorous function calculus. In: Proc. of the 4th IFAC Conference on Analysis and Design of Hybrid Systems (ADHS12). pp. 284–290. Eindhoven, The Netherlands (June 2012)

[12] Dang, T., Gawlitza, T.M.: Template-based unbounded time verification of affine hybrid automata. In: Asian Symposium on Programming Languages and Systems. pp. 34–49. Springer (2011)

[13] Frehse, G.: PHAVer: algorithmic verification of hybrid systems past HyTech. STTT 10(3), 263–279 (2008)

[14] Frehse, G., Bogomolov, S., Greitschus, M., Strump, T., Podelski, A.: Eliminating spurious transitions in reachability with support functions. In: Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control. pp. 149–158. ACM (2015)

[15] Frehse, G., Le Guernic, C., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: SpaceEx: Scalable verification of hybrid systems. In: CAV'11. pp. 379–395 (2011)

[16] Guernic, C.L., Girard, A.: Reachability analysis of hybrid systems using support functions. In: Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings. pp. 540–554 (2009)

[17] Halbwachs, N., Proy, Y.E., Raymond, P.: Verification of linear hybrid systems by means of convex approximations. In: International Static Analysis Symposium, SAS'94. Namur (Belgium) (September 1994)
[18] Henzinger, T., Ho, P.H., Wong-Toi, H.: HyTech: A model checker for hybrid systems. Software Tools for Technology Transfer 1, 110–122 (1997)
[19] Henzinger, T.A.: The theory of hybrid automata. In: Verification of Digital and Hybrid Systems, pp. 265–292. Springer (2000)
[20] Henzinger, T.A., Ho, P.H., Wong-Toi, H.: Algorithmic analysis of nonlinear hybrid systems. IEEE Transactions on Automatic Control 43, 540–554 (1998)
[21] Henzinger, T.A., Kopke, P.W., Puri, A., Varaiya, P.: What's decidable about hybrid automata? In: Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, 29 May-1 June 1995, Las Vegas, Nevada, USA. pp. 373–382 (1995)
[22] Jha, S.K., Krogh, B.H., Weimer, J.E., Clarke, E.M.: Reachability for linear hybrid automata using iterative relaxation abstraction. In: International Workshop on Hybrid Systems: Computation and Control. pp. 287–300. Springer (2007)
[23] Johansson, F.: Arb: efficient arbitrary-precision midpoint-radius interval arithmetic. IEEE Transactions on Computers 66, 1281–1292 (2017)
[24] Kong, S., Gao, S., Chen, W., Clarke, E.: dreach: δ-reachability analysis for hybrid systems. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 200–205. Springer (2015)
[25] Moler, C., Van Loan, C.: Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. SIAM review 45(1), 3–49 (2003)
[26] Nellen, J., Ábrahám, E., Wolters, B.: A cegar tool for the reachability analysis of plc-controlled plants using hybrid automata. In: Formalisms for Reuse and Systems Integration, pp. 55–78. Springer (2015)
[27] Ratschan, S., She, Z.: Safety verification of hybrid systems by constraint propagation-based abstraction refinement. ACM Transactions on Embedded Computing Systems (TECS) 6(1), 8 (2007)
[28] Rockafellar, R.T.: Convex Analysis. Princeton University Press (1970)
[29] Rohn, J.: Systems of linear interval equations. Linear algebra and its applications 126, 39–78 (1989)
[30] Roohi, N., Prabhakar, P., Viswanathan, M.: Hybridization based cegar for hybrid automata with affine dynamics. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 752–769. Springer (2016)
[31] Sankaranarayanan, S., Dang, T., Ivančić, F.: Symbolic model checking of hybrid systems using template polyhedra. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 188–202. Springer (2008)
[32] Sankaranarayanan, S., Sipma, H.B., Manna, Z.: Scalable analysis of linear systems using mathematical programming. In: International Workshop on Verification, Model Checking, and Abstract Interpretation. pp. 25–41. Springer (2005)
[33] Segelken, M.: Abstraction and counterexample-guided construction of ω-automata for model checking of step-discrete linear hybrid models. In: International Conference on Computer Aided Verification. pp. 433–448. Springer (2007)
[34] Sorea, M.: Lazy approximation for dense real-time systems. In: Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, pp. 363–378. Springer (2004)
[35] Vaandrager, F.: Hybrid systems. Images of SMC Research pp. 305–316 (1996)