Hybridization based CEGAR for Hybrid Automata with Affine Dynamics

Nima Roohi[1], Pavithra Prabhakar[2], and Mahesh Viswanathan[1]

[1] Department of Computer Science,
University of Illinois at Urbana-Champaign, USA
{roohi2,vmahesh}@illinois.edu
[2] Department of Computing & Information Sciences,
Kansas State University, USA
pprabhakar@ksu.edu

Abstract. We consider the problem of safety verification for hybrid systems, whose continuous dynamics in each mode is affine, $\dot{X} = AX + b$, and invariants and guards are specified using rectangular constraints. We present a counter-example guided abstraction refinement framework (CEGAR), which abstract these hybrid automata into simpler ones with rectangular inclusion dynamics, $\dot{x} \in \mathcal{I}$, where x is a variable and $\mathcal{I}$ is an interval in $\mathbb{R}$. In contrast to existing CEGAR frameworks which consider discrete abstractions, our method provides highly efficient abstraction construction, though model-checking the abstract system is more expensive. Our CEGAR algorithm has been implemented in a prototype tool called `HARE` (Hybrid Abstraction-Refinement Engine), that makes calls to `SpaceEx` to validate abstract counterexamples. We analyze the performance of our tool against standard benchmark examples, and show that its performance is promising when compared to state-of-the-art safety verification tools, `SpaceEx`, `PHAVer`, `SpaceEx AGAR`, and `HSolver`.

1 Introduction

The safety verification of cyber-physical systems is a computationally challenging problem that is in general undecidable [1, 3, 22, 27, 35]. Thus, verifying realistic designs often involves crafting an abstract model with simpler dynamics that is amenable to automated analysis. The success of the abstraction based method depends on finding the right abstraction, which can be difficult. One approach that tries to address this issue is the counterexample guided abstraction refinement (CEGAR) technique [12] that tries to automatically discover the right abstraction through a process of progressive refinement based on analyzing spurious counterexamples in abstract models. CEGAR has been found to be useful in a number of contexts [5, 13, 23, 24], including hybrid systems [2, 10, 11, 15, 17, 25, 33, 34].

There are two principal CEGAR approaches in the context of verifying hybrid system that differ primarily on the space of abstract models considered. The first approach [2, 10, 11, 32–34] tries to abstract hybrid models into finite state, discrete transition systems that have no continuous dynamics. The second approach [15,25,29] abstracts a hybrid automaton by another hybrid automaton with simpler dynamics. Using hybrid automata as abstractions has the advantage that constructing abstract models is computationally easier.

In this paper, we present a CEGAR framework for verifying cyber-physical systems, where the concrete and abstract models are both hybrid automata. We consider hybrid automata with affine dynamics and rectangular constraints (affine hybrid automata for short) which are a subclass of hybrid automata, where invariants, guards, and resets are given by rectangular constraints (conjunctions of constraints comparing variables to constants), but the continuous flow in control locations is given by linear differential equations of the form $\dot{X} = AX + b$; here X is the vector of continuous variables, A is a rational matrix, and b is a vector of rational numbers. The safety verification problem for such automata is challenging — not only is the problem undecidable, but it is even unknown whether the problem of checking if the states reachable within a time bound t (without taking any discrete transitions) intersects a polyhedral unsafe region is decidable. We abstract such affine hybrid automata by rectangular hybrid automata. Rectangular hybrid automata are similar to affine hybrid automata except that the continuous dynamics is given by rectangular differential inclusions (i.e., dynamics of each variable is of the form $\dot{x} \in [a, b]$) as opposed to linear differential equations. Our results extend previous hybrid automata based CEGAR algorithms [15,25,29] to a richer class of hybrid models (from concrete automata that have rectangular dynamics to automata that have affine dynamics).

We establish a few basic results about our CEGAR framework. First we show that any spurious counterexample will be detected during the counterexample validation step. This result is not obvious because it is unknown whether the bounded time reachability problem is decidable for affine hybrid automata. Hence validation cannot be carried out "exactly". Our proof relies on the observation that the sets computed during counterexample validation are bounded, and uses the fact that continuous time bounded posts of affine hybrid automata can be approximated with arbitrary precision. Next, we show that our refinement algorithm makes progress. More precisely, we prove that any abstract counterexample, if it appears sufficiently many times, is eventually eliminated. Progress is proved by observing that, for a bounded time, linear dynamics can be approximated with arbitrary precision by rectangular dynamics [31].

We have extended our CEGAR-based tool `HARE` (Hybrid Abstraction Refinement Engine) to verify affine hybrid automata; the previous `HARE` implementation only handled rectangular hybrid automata. Furthermore, we found existing tools for model checking rectangular hybrid automata (`HyTech` [21], `PHAVer` [19], `SpaceEx` [20], and `FLOW*` [8]) inadequate for our purposes (see Section 5 for explanations). So we implemented a new model checker for rectangular hybrid automata that uses the Parma Polyhedral Library (`PPL`) [4] and `Z3` [14]. Counterexample validation is carried out by making calls to `SpaceEx` and `PPL`.

We have compared the performance of the new version of our tool `HARE` against `SpaceEx` with the `Supp` and `PHAVer` scenarios, `SpaceEx AGAR` [7], and `HSolver` [32] on standard benchmark examples. `SpaceEx` is the state-of-the-art symbolic state space explorer for affine hybrid automata that over-approximates the reachable set, and may occasionally converge to a fixpoint in the process. `SpaceEx AGAR` is a CEGAR-based tool that merges different locations and over-approximates their dynamics. `HSolver` is a another CEGAR-based tool that abstracts hybrid automata into finite-state, discrete abstractions (as opposed to other hybrid automata). `HSolver` failed to terminate within a reasonable time on almost all of our examples. The running time of `HARE` was roughly comparable to `SpaceEx` and `SpaceEx AGAR` (details in Section 5), with each tool beating the other on different examples. But we found that `HARE` was more *accurate*. On quite a few examples, `SpaceEx` (and `SpaceEx AGAR`) fails to prove safety either because it does not converge to a fixpoint or because it over-approximates the reach set too much. A virtual machine containing the scripts to run all the examples reported in the paper on the 5 tools considered can be found at `https://uofi.box.com/cegar-hare-tacas-2016`.

2 Related Work

Doyen *et al.* consider rectangular abstractions for safety verification of affine hybrid systems in [16]. However, their refinement is not guided by counter-example analysis. Instead, a reachable unsafe location in the abstract system is determined, and the invariant of the corresponding concrete location is split to ensure certain optimality criteria on the resulting rectangular dynamics. This, in general, may not lead to abstract counter-example elimination, as in our CEGAR algorithm. We belive that the refinement algorithms of the two papers are incomparable — one may perform better than the other on certain examples. Empirical evaluations could provide some insights into the merits of the approaches, however, the implementation of the algorithm in [16] was not available for comparison at the time of writing the paper.

Bogomolov *et al.* consider polyhedral inclusion dynamics as abstract models of affine hybrid systems for CEGAR in [7]. Their abstraction merges the locations, and refinement corresponds to splitting the locations. Hence, the CEGAR loop ends with the original automaton in a finite number of steps, if safety is not proved by then. Our algorithm splits the invariants of the locations, and hence, explores finer abstractions. Our method is orthogonal to that of [7], and can be used in conjunction with [7] to further refine the abstractions.

Nellen *et al.* use CEGAR in [28] to model check chemical plants controlled by programmable logic controllers. They assume that the dynamics of the system in each location is given by *conditional* ODEs, and their abstraction consists of choosing a subset of these conditional ODEs. The refinement consists of adding some of these conditional ODEs based on a unsafe location in a counter-example. The methods does not ensure counter-example elimination in successive iterations. Their prototype tool does not automate the refinement step, in that the inputs to the refinements need to be provided manually. Hence, we did not experimentally compare with this tool.

Zutshi *et al.* propose a CEGAR-based search in [36] to find violations of safety properties. Here they consider the problem of finding a concrete counter-example and use CEGAR to guide the search of the same. We instead use CEGAR to prove safety — the absence of such concrete counter-examples.

3 Preliminaries

Numbers. Let $\mathbb{N}$, $\mathbb{Q}$, and $\mathbb{R}$ denote the set of *natural*, *rational*, and *real* numbers, respectively. Similarly, $\mathbb{N}_+$, $\mathbb{Q}_+$, and $\mathbb{R}_+$ are respectively the set of *positive* natural, rational, and real numbers, and $\mathbb{Q}_{\geq 0}$ and $\mathbb{R}_{\geq 0}$ are respectively the set of *non-negative* rational and real numbers. For any $n \in \mathbb{N}$ we define $[n] = \{0, 1, \ldots, n-1\}$.

Sets and Functions. For any sets A and B, $|A|$ is the size of A (the number of elements in A), $\mathcal{P}(A)$ is the power set of A, $A \times B$ is the Cartesian product of A and B, and $[A \to B]$ is the set of all (total) functions from A to B. A^B is a vector of elements in A indexed by elements in B (we treat an element of A^B as a function from B to A). In order to make the notations simpler, for any $n, m \in \mathbb{N}$, by A^n and $A^{n \times m}$, we mean $A^{[n]}$ and $A^{[n]\times[m]}$. The latter represents matrices of dimension $n \times m$ with elements from A. For any $f \in [A \to B]$ and set $C \subseteq A$, $f(C) = \{f(c) \mid c \in C\}$. Similarly, for any $\pi = a_1, a_2, \ldots, a_n$, a sequence of elements in A, we define $f(\pi)$ to be $f(a_1), f(a_2), \ldots, f(a_n)$.

Distance and Intervals. When A and B are non-empty subsets of a normed space with norm $[\![.]\!]$, we define their *Hausdorff distance* $\mathtt{dist}_{\mathtt{H}}(A, B)$ by

$$\max\{\sup_{a\in A}\inf_{b\in B}[\![a-b]\!], \sup_{b\in B}\inf_{a\in A}[\![a-b]\!]\}$$

An *interval* is any subset of real numbers of the form $[a, b]$, $(a, b]$, $[a, b)$, or (a, b). We denote the set of all intervals by $\mathcal{I}$ and the set of all closed-bounded intervals by $\mathcal{I}_\circ$.

3.1 Hybrid Automata

In this section, we present a hybrid automaton model for representing hybrid systems.

Definition 1. *A hybrid automaton H is a tuple* $(\mathtt{Q}, \mathtt{X}, \mathtt{I}, \mathtt{F}, \mathtt{E}, \mathtt{Q}^{\mathsf{init}}, \mathtt{Q}^{\mathsf{bad}})$*, where*
- $\mathtt{Q}$ *is a finite non-empty set of (discrete)* locations*.*
- $\mathtt{X}$ *is a finite set of* variables*. A valuation $\nu \in \mathbb{R}^{\mathtt{X}}$ assigns a value to each variable in* $\mathtt{X}$*. We denote the set of all valuations by* $\mathtt{V}$*.*
- $\mathtt{I} \in [\mathtt{Q} \to \mathcal{I}_\circ^{\mathtt{X}}]$ *maps each location q to a closed bounded rectangular region as its* invariant*. We denote* $\mathtt{I}(q)(x)$ *by* $\mathtt{I}(q, x)$*.*

– $\mathtt{F} \in [\mathtt{Q} \times \mathtt{V} \to \mathcal{P}(\mathtt{V})]$ *maps each location* q *and valuation* ν *to a set of possible derivatives of the trajectories in that location and valuation.*
– $\mathtt{E}$ *is a finite set of* edges e *of the form* (s, d, g, j, r) *where:*
  - $s, d \in \mathtt{Q}$ *are* source *and* destination *locations, respectively.*
  - $g \in \mathcal{I}_{\circ}^{\mathtt{X}}$ *is* guard *of* e *and specifies the set of possible values for each variable in order to traverse* e*.*
  - $j \in \mathcal{P}(\mathtt{X})$ *is the set of variables whose values* change *after traversing* e*.*
  - $r \in \mathcal{I}_{\circ}^{j}$ *is* reset *of* e *and specifies the set of possible values for each variable in* j *after traversing* e*.*

  We write $\mathtt{S}e$, $\mathtt{D}e$, $\mathtt{G}e$, $\mathtt{J}e$, *and* $\mathtt{R}e$ *to denote different elements of an edge* e*, respectively. Also we denote* $(\mathtt{G}e)(x)$ *and* $(\mathtt{R}e)(x)$ *respectively by* $\mathtt{G}(e, x)$ *and* $\mathtt{R}(e, x)$*.*
– $\mathtt{Q}^{\mathsf{init}}, \mathtt{Q}^{\mathsf{bad}} \subseteq \mathtt{Q}$ *are respectively the set of* initial *and* unsafe *locations.*

For all hybrid automata H*, we display elements of* H *by* $\mathtt{Q}_H$, $\mathtt{X}_H$, $\mathtt{I}_H$, $\mathtt{F}_H$, $\mathtt{E}_H$, $\mathtt{S}_H$, $\mathtt{D}_H$, $\mathtt{G}_H$, $\mathtt{J}_H$, $\mathtt{R}_H$, $\mathtt{Q}_H^{\mathsf{init}}$, $\mathtt{Q}_H^{\mathsf{bad}}$, *and* $\mathtt{V}_H$*. We may omit the subscript when it is clear from the context.*

We define the semantics of a hybrid automaton by a transition system it represents. Hence, we first define transition systems.

Definition 2. *A transition system* T *is a tuple* $(\mathtt{S}, \Sigma, \to, \mathtt{S}^{\mathsf{init}}, \mathtt{S}^{\mathsf{bad}})$ *in which*
1. $\mathtt{S}$ *is a (possibly infinite) set of states,*
2. Σ *is a (possibly infinite) set of labels,*
3. $\to \subseteq \mathtt{S} \times \Sigma \times \mathtt{S}$ *is a transition relation,*
4. $\mathtt{S}^{\mathsf{init}} \subseteq \mathtt{S}$ *is the set of initial states, and*
5. $\mathtt{S}^{\mathsf{bad}} \subseteq \mathtt{S}$ *is the set of unsafe states.*

We write $s \xrightarrow{\alpha} s'$ *instead of* $(s, \alpha, s') \in \to$*. Also, we write* $s \to s'$ *as a shorthand for* $\exists \alpha \in \Sigma \bullet s \xrightarrow{\alpha} s'$*, and* $\to^*$ *denotes the reflexive transitive closure of* $\to$*. Finally, for any* $s \in \mathtt{S}$ *we define* $\mathsf{reach}_T(s)$ *to be the set* $\{s' \in \mathtt{S} | s \to^* s'\}$*, and* $\mathsf{reach}(T)$ *to be* $\bigcup_{s \in \mathtt{S}^{\mathsf{init}}} \mathsf{reach}_T(s)$*.*

For all transition systems T*, we denote the elements of* T *by* $\mathtt{S}_T$, Σ_T, $\to_T$, $\mathtt{S}_T^{\mathsf{init}}$, $\mathtt{S}_T^{\mathsf{bad}}$*. In addition, whenever it is clear, we drop the subscript* T *to make the notation simpler.*

The semantics of a hybrid automaton $H = (\mathtt{Q}, \mathtt{X}, \mathtt{I}, \mathtt{F}, \mathtt{E}, \mathtt{Q}^{\mathsf{init}}, \mathtt{Q}^{\mathsf{bad}})$ can be defined as a transition system $[\![H]\!] = (\mathtt{S}, \Sigma, \to, \mathtt{S}^{\mathsf{init}}, \mathtt{S}^{\mathsf{bad}})$ in which
– $\mathtt{S} = \mathtt{Q} \times \mathtt{V}$,
– $\Sigma = \mathtt{E} \cup \mathbb{R}_{\geq 0}$,
– $\mathtt{S}^{\mathsf{init}} = \{(q, \nu) \in \mathtt{S} \mid q \in \mathtt{Q}^{\mathsf{init}}\}$,
– $\mathtt{S}^{\mathsf{bad}} = \{(q, \nu) \in \mathtt{S} \mid q \in \mathtt{Q}^{\mathsf{bad}}\}$,
– $\to = \to_1 \cup \to_2$ where
  - $\to_1$ is the set of time transitions and for all $t \in \mathbb{R}_{\geq 0}$ $(q, \nu) \xrightarrow{t}_1 (q', \nu')$ iff $q = q'$ and there exists a differentiable function $f \in [[0, t] \to \mathtt{V}]$ such that 1. $f(0) = \nu$, 2. $f(t) = \nu'$, 3. $\forall t' \in [0, t] \bullet f(t') \in \mathtt{I}(q)$, and 4. $\dot{f}(t') \in \mathtt{F}(q, f(t'))$.
  - $\to_2$ is the set of jump transitions and $(q, \nu) \xrightarrow{e}_2 (q', \nu')$ iff 1. $q = \mathtt{S}e$, 2. $q' = \mathtt{D}e$, 3. $\nu \in \mathtt{I}(q) \cap \mathtt{G}e$, 4. $\nu' \in \mathtt{I}(q')$, and 5. $\forall x \in \mathtt{X} \bullet\ x \in \mathtt{J}e \Rightarrow \nu'(x) \in \mathtt{R}(e, x)$ and $x \notin \mathtt{J}e \Rightarrow \nu(x) = \nu'(x)$.

In this paper, we deal with two subclasses of hybrid automata:

1. An affine hybrid automaton is a hybrid automaton in which for every location $q \in \mathtt{Q}$ there exists a matrix $M \in \mathbb{Q}^{\mathtt{X}^2}$ and a vector $b \in \mathbb{Q}^{\mathtt{X}}$ such that for every valuation $\nu \in \mathtt{V}$ we have $\mathtt{F}(q, \nu) = \{M\nu + b\}$. This is the class of hybrid automata we intend to analyse for safety.
2. A rectangular automaton is a hybrid automaton in which for every location $q \in \mathtt{Q}$ there exists a rectangular region $f \in \mathcal{I}^{\mathtt{X}}$ such that for every valuation $\nu \in \mathtt{V}$ we have $\mathtt{F}(q, \nu) = f$. We may write $\mathtt{F}(q, x)$ to denote the set of possible flows for variable x at location q. We use this class to represent abstract hybrid automata in our CEGAR algorithm.

For a hybrid automaton H, a *path* is defined to be a finite sequence $e_1, e_2, \ldots, e_n$ of edges in $\mathtt{E}$ such that $\mathtt{D}e_i = \mathtt{S}e_{i+1}$ for all $0 < i < n$. A *timed path* π is a finite sequence of the form $(t_1, e_1), (t_2, e_2), \ldots, (t_n, e_n)$ such that $e_1, \ldots, e_n$ is a path in H and $t_i \in \mathbb{R}_{\geq 0}$ for all $0 < i \leq n$. A *run ρ from s_0 to s_n* is a finite sequence $s_0, (t_1, e_1), s_1, (t_2, e_2), \ldots, (t_n, e_n), s_n$ such that 1. $(t_1, e_1), \ldots, (t_n, e_n)$ is a timed path in H, 2. for all $0 \leq i \leq n$ we have $s_i \in \mathtt{S}_{[\![H]\!]}$, and 3. for all $0 < i \leq n$ there exists a state $s'_i \in \mathtt{S}_{[\![H]\!]}$ for which $s_{i-1} \xrightarrow{t_i} s'_i \xrightarrow{e_i} s_i$. We will denote the first and last elements of ρ respectively by ρ_0 and ρ_{lst}.

For any hybrid automaton H, the *reachability problem* asks whether or not H has a run ρ such that $\rho_0 \in \mathtt{S}^{\mathsf{init}}_{[\![H]\!]}$ and $\rho_{\mathsf{lst}} \in \mathtt{S}^{\mathsf{bad}}_{[\![H]\!]}$. If the answer is positive, we say the H is *unsafe*. Otherwise, we say the H is *safe*.

For any hybrid automaton H, set of states $S \subseteq \mathtt{S}_{[\![H]\!]}$, and edge $e \in \mathtt{E}_H$ we define the following functions:

- $\mathsf{dpost}^e_H(S) = \{s' \mid \exists s \in S \bullet s \xrightarrow{e} s'\}$. Discrete post of S in H with respect to e is the set of states reachable from S after taking e.
- $\mathsf{dpre}^e_H(S) = \{s \mid \exists s' \in S \bullet s \xrightarrow{e} s'\}$. Discrete pre of S in H with respect to e is the set of states that can reach a state in S after taking e.
- $\mathsf{cpost}_H(S) = \{s' \mid \exists s \in S, t \in \mathbb{R}_{\geq 0} \bullet s \xrightarrow{t} s'\}$. Continuous post of S in H is the set of states reachable from S in an arbitrary amount of time using dynamics specified for the source states.
- $\mathsf{cpre}_H(S) = \{s \mid \exists s' \in S, t \in \mathbb{R}_{\geq 0} \bullet s \xrightarrow{t} s'\}$ Continuous pre of S in H is the set of states that can reach a state in S in an arbitrary amount of time using dynamics specified for the source states.

4 CEGAR Algorithm for Safety Verification of Affine Hybrid Automata

Every CEGAR-based algorithm has four main parts [9]: 1. abstracting the concrete system, 2. model checking the abstract system, 3. validating the abstract counterexample, and 4. refining the abstract system. We explain parts of our algorithm regarding each of these parts in this section. Before that, Algorithm 1 shows at a very high level what the steps of our algorithm are.

Algorithm 1 High level steps of our CEGAR algorithm

Input: C an affine hybrid automaton ▷ C is called concrete hybrid automaton. Def 1
Output: Whether or not C is safe ▷ this is the reachability problem. Sec 3

Add a trivial self loop to every location of C ▷ Sec 4.2
$P \leftarrow$ the initial partition of invariants in C ▷ Sec 4.2
$A \leftarrow \alpha(C, P)$ ▷ A is called abstract hybrid automaton. Def 4
$\rho = O^{\mathtt{RHA}}(A)$ ▷ $O^{\mathtt{RHA}}$ model checks rectangular automata. Sec 4.3
▷ ρ is an annotated counterexample. Sec 4.3
while $\rho \neq \emptyset$ **do** ▷ while abstract system is unsafe
  if ρ is valid in C **then return** 'unsafe' ▷ Sec 4.4
  $(q, p) \leftarrow$ abstract location that should be split ▷ Sec 4.5
  $p_1, p_2 \leftarrow$ sets that should be separated in (q, p) ▷ Sec 4.5
  refine $P(q)$ such that p_1 and p_2 gets separated ▷ Sec 4.5
  $A \leftarrow \alpha(C, P)$ ▷ Sec 4.2
  $\rho = O^{\mathtt{RHA}}(A)$ ▷ Sec 4.3
end while
return 'safe'

4.1 Time-Bounded Transitions

A step of every CEGAR algorithm is to validate a counterexample of an abstract system returned by the model-checking phase (Section 4.4). We do validation by running the counterexample of the abstract model checker against the concrete hybrid automaton. In our discussion, we will assume that for affine hybrid automata one can compute the continuous post of a set of states for an arbitrary amount of time. But this is not completely true. What we can do is to only compute approximations of the continuous post of a set of states. In addition, bounded error approximations can be computed only for a finite amount of time. Hence, we convert a hybrid automaton H to another hybrid automaton H' with the same reachability information and with the additional property that in H', there is no time transition with a label larger than t, for some parameter $t \in \mathbb{R}_+$. With this transformation, we can compute bounded error approximations of the unbounded time post, since it is actually a continuous post over a bounded time t. Appendix B shows how the new automaton is formally constructed.

4.2 Abstraction

Input to our algorithm is an affine hybrid automaton C which we call the *concrete* hybrid automaton. The first step is to construct an *abstract* hybrid automaton A which is a rectangular automaton. The abstract hybrid automaton A is obtained from the concrete hybrid automaton C, by splitting the invariant of any location $q \in \mathtt{Q}_C$ into a finite number of cells of type $\mathcal{I}_\circ^{\mathtt{X}}$ and defining an abstract location for each of these cells which over-approximates the linear dynamics in the cell by a rectangular dynamics. Definition 3 and Definition 4 formalizes the way an abstraction A is constructed from C.

Definition 3 (Invariant Partitions). *For any hybrid automaton C and function $P \in [\mathtt{Q} \rightarrow \mathcal{P}(\mathcal{I}_\circ^{\mathtt{X}})]$ we say P partitions invariants of C iff the following conditions hold for any location $q \in \mathtt{Q}$:*

- $\bigcup P(q) = \mathtt{I}(q)$, *which means union of cells in* $P(q)$ *covers invariant of* q.
- $\forall p_1, p_2 \in P(q), x \in \mathtt{X}$ *at least one of the following conditions are true:*
  - $|p_1(x) \cap p_2(x)| = 0$
  - $|p_1(x) \cap p_2(x)| = 1$
  - $p_1(x) = p_2(x)$

Definition 4 (Abstraction Using Invariant Partitioning). *For any affine hybrid automaton* C *and invariant partition* $P \in [\mathtt{Q} \to \mathcal{P}(\mathcal{I}_\circ^{\mathtt{X}})]$, $\alpha(C, P)$ *returns rectangular automaton* A *which is defined below:*

- $\mathtt{Q}_A = \{(q, p) \mid q \in \mathtt{Q}_C \wedge p \in P(q)\}$,
- $\mathtt{X}_A = \mathtt{X}_C$,
- $\mathtt{Q}_A^{\mathsf{init}} = \{(q, p) \in \mathtt{Q}_A \mid q \in \mathtt{Q}_C^{\mathsf{init}}\}$,
- $\mathtt{I}_A((q, p)) = p$,
- $\mathtt{Q}_A^{\mathsf{bad}} = \{(q, p) \in \mathtt{Q}_A \mid q \in \mathtt{Q}_C^{\mathsf{bad}}\}$,
- $\mathtt{E}_A = \{((s, p_1), (d, p_2), g, j, r) \mid (s, d, g, j, r) \in \mathtt{E}_C \wedge (s, p_1), (d, p_2) \in \mathtt{Q}_A\}$, *and*
- $\mathtt{F}_A((q, p), \nu) = \mathsf{recthull}(\bigcup_{\nu \in p} \mathtt{F}_C(q, \nu))$, *where for any set* $S \subset \mathbb{R}^{\mathtt{X}}$, $\mathsf{recthull}(S)$ *is the smallest possible element of* $\mathcal{I}_\circ^{\mathtt{X}}$ *such that* $\forall \nu \in S \bullet \nu \in \mathsf{recthull}(S)$.

In addition, we define function γ_A *to map 1. every state in* $[\![A]\!]$ *to a state in* $[\![C]\!]$, *and 2. every edge in* $\mathtt{E}_A$ *to an edge in* $\mathtt{E}_C$. *Formally, for any* $s = ((q, p), \nu) \in \mathtt{S}_{[\![A]\!]}$ *and* $e = ((q_1, p_1), (q_2, p_2), g, j, r) \in \mathtt{E}_A$, *we define* $\gamma_A(s)$ *to be* (q, ν) *and* $\gamma_A(e)$ *to be* (q_1, q_2, g, j, r).

For each concrete location we will have one or more abstract locations. By making invariants of abstract locations small (and thus increasing the number of abstract locations) we want to be able to make behavior of A as close as required to the behavior of C. This requires trajectories to be always able to jump between two abstract locations when they correspond to a single concrete location. But we did not add any such edge to A in Definition 4. Although defining abstract system in this way just imposes an additional initial step to our algorithm, we find it very convenient not to introduce any edge in the abstract hybrid automata that corresponds to no edge in the concrete hybrid automata. Nonetheless, it is easy to see that if for every location $q \in \mathtt{Q}_C$, $\mathtt{E}_C$ contains a trivial edge (*i.e.* an edge with no guard and no reset) from q to itself, abstracting C using Definition 4 will produce a trivial edge between all abstract locations corresponding to a single concrete location. One can easily add these edges to C in an initial step, so in the rest of this paper, wlog. we assume every location of C has a trivial self loop. Finally, it is easy to see that these trivial self loops along with Definition 3 and Definition 4 introduce Zeno behavior in the abstract system (*i.e.* the abstract system can make an infinite number of discrete transitions in a finite amount of time), but our model checker can easily handle it. In fact since we check for a fixed-point, we believe our tool is not considerably affected by this type of behavior.

Proposition 5 (Over-Approximation). *For any affine hybrid automaton* C *and invariant partition* P, $A = \alpha(C, P)$ *is a rectangular automaton which* over-approximates C, *that is,* $\mathsf{reach}(C) \subseteq \gamma_A(\mathsf{reach}(A))$.

It is clear that if A is safe then C is also safe. Also, one can easily see that if P is defined as $P(q) = \{\mathtt{I}_C(q)\}$ (for all $q \in \mathtt{Q}_C$), it is a valid invariant partition of C. It is actually what our algorithm always uses as the initial invariant partitioning (initially we do not partition any invariant).

4.3 Counterexample and Model Checking Rectangular Automata

After an abstract hybrid automaton is constructed (initially and after any refinement), we have to model check it. In this section we define the notion of a counterexample and annotation of a counterexample, which we assume is returned by the abstract model checker $O^{\mathtt{RHA}}$ when it finds that the input hybrid automaton is unsafe.

Definition 6. *For any hybrid automaton H, a* counterexample *is a path $e_1, \ldots,$ e_n such that $\mathtt{S}e_1 \in \mathsf{Q}^{\mathsf{init}}$ and $\mathtt{D}e_n \in \mathsf{Q}^{\mathsf{bad}}$.*

Definition 7. *A counterexample π is called* valid *in H iff H has a run ρ and ρ has the same path as π. A counterexample that is not valid is called* spurious.

Definition 8. *An* annotation for a counterexample $\pi = e_1, \ldots, e_n$ *of hybrid automaton H is a sequence $\rho = S_0 \to S'_0 \xrightarrow{e_1} S_1 \to S'_1 \xrightarrow{e_2} \cdots \xrightarrow{e_n} S_n \to S'_n$ such that the following conditions hold:*

1. $\forall 0 \leq i \leq n \bullet \emptyset \neq S_i, S'_i \subseteq \mathsf{S}_{[\![H]\!]}$,
2. $\forall 0 \leq i \leq n \bullet S_i = \mathsf{cpre}_H(S'_i)$,
3. $\forall 0 \leq i < n \bullet S'_i = \mathsf{dpre}_H^{e_{i+1}}(S_{i+1})$,
4. $S'_n = \mathsf{S}^{\mathsf{bad}}_{[\![H]\!]} \cap (\{\mathtt{D}e_n\} \times \mathtt{V}_H)$.

Condition 1 means that each S_i and S'_i in ρ are a non-empty set of states. Conditions 2 and 3 mean that sets of states in ρ are computed using backward reachability. Finally, condition 4 means that S'_n is the set of unsafe states in destination of e_n. Note that these conditions completely specify $S_0, \ldots, S_n$ and $S'_0, \ldots, S'_n$ from $e_1, \ldots, e_n$ and H. Also, every S_i and S'_i is a subset of states corresponding to exactly one location.

In this paper, we assume to have access to an oracle $O^{\mathtt{RHA}}$ that can correctly answer reachability problems when the hybrid automata are restricted to be rectangular automata. If no unsafe location of A is reachable from an initial location of it, $O^{\mathtt{RHA}}(A)$ returns 'safe'. Otherwise, it returns an annotated counterexample of A.

4.4 Validating Abstract Counterexamples

For any invariant partition P and affine hybrid automaton C, if $O^{\mathtt{RHA}}(A)$ (for $A = \alpha(C, P)$) returns 'safe', we know C is safe. So the algorithm returns C is 'safe' and terminates. On the other hand, if $O^{\mathtt{RHA}}$ finds A to be unsafe it returns an annotated counterexample ρ of A. Since A is an over-approximation of C, we cannot be certain at this point that C is also unsafe. More precisely, if π is the path in ρ, we do not know whether $\gamma_A(\pi)$ is a valid counterexample in C or it is spurious. Therefore, we need to validate ρ in order to determine if it corresponds to any actual run from an initial location to an unsafe location in C.

To validate ρ, an annotated counterexample of $A = \alpha(C, P)$, we run ρ on C. More precisely, we create a sequence $\rho' = R_0 \to R'_0 \xrightarrow{e'_1} R_1 \to \cdots \xrightarrow{e'_n} R_n \to R'_n$ where

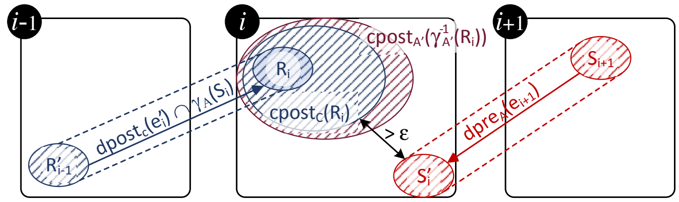


Fig. 1: Validation and Refinement. There are three locations: $i-1$, i, and $i+1$. S_{i+1} and S'_i are elements of annotated counterexample ρ. R'_{i-1}, R_i, and $\mathsf{cpost}_C(R_i)$ are computed when ρ is validated. i is the smallest index for which $\mathsf{cpost}_C(R_i)$ and $\gamma_A(S'_i)$ are separated. Hence we need to refine A in location i. Refinement should be done in such a way that for the result of refinement A' we have $\mathsf{cpost}_{A'}(\gamma_{A'}^{-1}(R_i)) \cap \gamma_{A'}(S'_i) = \emptyset$.

1. $e'_i = \gamma_A(e_i)$,
2. $R_0 = \gamma_A(S_0)$,
3. $R'_i = \mathsf{cpost}_C(R_i) \cap \gamma_A(S'_i)$,
4. $R_i = \mathsf{dpost}_C^{e'_i}(R'_{i-1}) \cap \gamma_A(S_i)$.

Condition 1 states that edges in ρ' correspond to the edges in ρ as defined by the function γ_A in Definition 4. Condition 2 states that R_0 is just concrete states corresponding to S_0. Note that R_0 is never empty. Condition 3 states that each R'_i is the intersection of two sets: 1. concrete states corresponding to abstract states in S'_i, and 2. continuous post of R_i. Condition 4 states that each R_i is the intersection of two sets: 1. concrete states corresponding to abstract states in S_i, and 2. discrete post of R'_{i-1} using e'_i. It is easy to see that for any i if R_i or R'_i becomes empty then for all $j > i$ both R_j and R'_j will be empty. Also, if R_i is empty then R'_i is empty too. Figure 1 depicts the situation when the counterexample is spurious and R'_i is the first empty set we reach during our validation. Proposition 9 proves that the first empty set (if any) is always R'_i for some i and not R_i.

Proposition 9. *$R'_n = \emptyset$ in ρ' implies there exists i such that 1. $R'_i = \emptyset$, 2. $R_i \neq \emptyset$, 3. $\forall j < i \bullet R_j, R'_j \neq \emptyset$, and 4. $\mathsf{cpost}_C(R_i)$ and $\gamma_A(S'_i)$ are nonempty disjoint sets.*

Lemma 10. *The counterexample $\pi' = e'_1, \ldots, e'_n$ of C is valid iff $R'_n \neq \emptyset$.*

Proposition 9 tells us that two sets $\mathsf{cpost}_C(R_i)$ and $\gamma_A(S'_i)$ are disjoint. Lemma 11 states a stronger result that there is a minimum distance $\epsilon > 0$ between those two sets, by exploiting the compactness of the two sets.

Lemma 11. *There exists $\epsilon \in \mathbb{R}_+$ such that $\mathtt{dist_H}(\mathsf{cpost}_C(R_i), \gamma_A(S'_i)) > \epsilon$.*

4.5 Refinement

Let us fix a concrete automaton C, an invariant partition P, and an abstract automaton $A = \alpha(C, P)$. Suppose model checking A reveals a counterexample π and its annotation ρ. If ρ is found to be spurious by the validation algorithm (in Section 4.4), then we need to refine the model A by refining the invariant partition P. We will do this by refining the invariant of only a single location of A. In this section we describe how to do this.

Since ρ is spurious, there is a smallest index i such that $R'_i = \emptyset$ (where the sets R_i, R'_i are as defined in Section 4.4); we will call this the *point of refinement* and denote it as $\mathtt{por}_{C,A}(\rho)$. We will refine the location $(q,p) = \mathtt{De}_i$ of A by refining its invariant p. We know from Proposition 9, $\mathsf{cpost}_C(R_i) \cap \gamma_A(S'_i) = \emptyset$. However, the coresponding sets in the abstract system A are not disjoint, that is, $\mathsf{cpost}_A(\gamma_A^{-1}(R_i)) \cap S'_i \neq \emptyset$. Our refinement strategy is to find a partition for the location (q,p) such that in the refined model $R = \alpha(C, P')$ (for some P'), S'_i is not reachable from R_i. In order to define the actual refinement, and to make this condition precise, we need to introduce some definitions.

Let C, A, R_i, S'_i, and (q,p) be as above. Let us denote by $C_{q,p}$ the restriction of C to the single location q with invariant p, *i.e.*, $C_{q,p}$ has only one location q whose flow and invariant is the same as that of (q,p) in A, and only transitions whose source and destination is q. We will say that an invariant partition P_r of $C_{q,p}$ *separates* R_i from S'_i iff in the automaton $A_1 = \alpha(C_{q,p}, P_r)$, $\mathsf{reach}_{A_1}(\gamma_{A_1}^{-1}(R_i)) \cap \gamma_{A_1}^{-1}(\gamma_A(S'_i)) = \emptyset$. In other words, the states corresponding to S'_i in A_1 are not reachable from $\gamma_{A_1}^{-1}(R_i)$ in A_1.

Refinement Strategy. Let P_r be an invariant partition of $C_{q,p}$ that separates R_i from S'_i. Define the invariant partition P' of C as follows: $P'(q') = P(q')$ if $q' \neq q$, and $P'(q) = (P(q) \setminus \{p\}) \cup P_r(q)$. The new abstract automaton will be $R = \alpha(C, P')$. Observe that R is a refinement of A (since the invariant partition is refined), and the relationship between the locations and edges of the two automata is characterized by a function $\alpha_{R,A}(\cdot)$ defined as follows. For a location (q', p'), $\alpha_{R,A}(q', p') = (q', p')$ if either $q' \neq q$, or $p' \not\subseteq p$, and $\alpha_{R,A}(q', p') = (q,p)$ otherwise. Having defined the mapping between locations, the mapping between edges is its natural extension:

$$\begin{gathered}\alpha_{R,A}((q_1,p_1),(q_2,p_2),g,j,r) = \\ (\alpha_{R,A}(q_1,p_1),\alpha_{R,A}(q_2,p_2),g,j,r).\end{gathered}$$

The goal of the refinement strategy outlined above is to ensure that a given counterexample π is eventually eliminated, if the abstract model checker generates it sufficiently many times. To make this statement precise and to articulate the nature of progress we need to first identify when a counterexample of R corresponds to a counterexample of A. Observe that a path π of A can "correspond" to a longer path π' in R, where previous sojourn in location (q,p) in π, now corresponds to a path in π' that traverses the newly created locations by partitioning p. Recall that we are assuming that $\mathtt{por}_{C,A}(\rho) = i$, where ρ is the annotation corresponding to π. We will say that a counterexample $\pi' = e'_1, e'_2, \ldots e'_m$ *corresponds* to counterexample $\pi = e_1, e_2, \ldots e_n$, if there exists k, $0 \leq k \leq m - i$, such that 1. for all $j \leq i$, $\alpha_{R,A}(e'_j) = e_j$, 2. for all $j > i + k$, $\alpha_{R,A}(e'_j) = e_{j-k}$, and 3. for all $i < j \leq i + k$, source and destination of $\alpha_{R,A}(e'_j)$ is (q,p). If π' corresponds to π, we will call k its witness. Using this notion of correspondence, we are ready to state what our refinement achieves.

Proposition 12. *Let π be a counterexample of A and ρ its annotation. Let R be the refinement constructed by our strategy after ρ is found to be spurious. Let*

π' be a counterexample of R that corresponds to π, and let ρ' be its annotation. Then, $\texttt{por}_{C,R}(\rho') < \texttt{por}_{C,A}(\rho)$.

The above proposition implies that a counterexample π can appear only finitely many times in the CEGAR loop. This is because the point of refinement of any π' in R corresponding to π in A is strictly smaller.

Next, we claim that a partition satisfying the refinement strategy always exists. It relies on the following observation from [30] which states that the reach set of a linear dynamical system can be approximated to within any ϵ by a rectangular hybridization over a bounded time interval.

Theorem 13 ([30]). *Let H be a linear hybrid automaton with a single location such that there is a bound T on the time for which the system can evolve in the location. Then, for any $\epsilon > 0$, there exists an invariant partition P of H such that $\texttt{dist}_{\texttt{H}}(\mathsf{reach}(H), \mathsf{reach}(\alpha(H, P))) < \epsilon$.*

Corollary 14 (Existence of Refinement). *There always exists a partition P' that separates R_i and S'_i.*

4.6 Validation Approximation

In order to validate a counterexample, we assumed to be able to exactly compute continuous post of a set of states in the affine hybrid automaton for a finite amount of time. But the best one can actually hope for is computing over and under approximation of this set. In this section we show that being able to approximate the continuous post is enough for our algorithm. For any hybrid automaton H, set of states $S \subseteq \texttt{S}_{[\![H]\!]}$, edge $e \in \texttt{E}_H$, and parameter $\epsilon \in \mathbb{R}_+$ we define the following functions:

- $\mathsf{cpost}^{\epsilon}_{\mathsf{over}}(S)$ is an over-approximation of $\mathsf{cpost}(S)$. Formally, if $\mathsf{cpost}^{\epsilon}_{\mathsf{over}}(S)$ returns S' then we know $\mathsf{cpost}(S) \subseteq S'$ and $\texttt{dist}_{\texttt{H}}(S', \mathsf{cpost}(S)) < \epsilon$.
- $\mathsf{cpost}^{\epsilon}_{\mathsf{under}}(S)$ is an under-approximation of $\mathsf{cpost}(S)$. Formally, if $\mathsf{cpost}^{\epsilon}_{\mathsf{under}}(S)$ returns S' then we know $\mathsf{cpost}(S) \supseteq S'$ and $\texttt{dist}_{\texttt{H}}(S', \mathsf{cpost}(S)) < \epsilon$.

During the validation procedure, instead of computing ρ' we compute ρ_o and ρ_u. They are computed exactly as ρ', except that in ρ_o and ρ_u, instead of cpost, we respectively use $\mathsf{cpost}^{\epsilon}_{\mathsf{over}}$ and $\mathsf{cpost}^{\epsilon}_{\mathsf{under}}$. Let us denote the last elements of ρ_o and ρ_u respectively by R'_n and U'_n. If U'_n is non-empty, we know ρ represents at least one valid counterexample. Therefore, the algorithm outputs 'unsafe' and terminates. If U'_n is empty but R'_n is non-empty, it means ϵ is too big. Therefore, the algorithm repeats itself using $\frac{\epsilon}{2}$. If R'_n is empty, it means all counterexamples in ρ are spurious. Therefore, too much over-approximation is deployed in A and it needs to be refined as stated in Section 4.5.

Lemma 15. *Given a counterexample π of A, if $\gamma_A(\pi)$ is spurious, then there exists an $\epsilon > 0$ for which R'_n is empty.*

The above lemma states that if the abstract counterexample is spurious, then the same will be detected by our algorithm. This is a direct consequence of Lemma 11.

5 Experimental Results

Our tool (Hybrid Abstraction Refinement Engine or HARE, for short) is implemented in Scala. The CEGAR framework relies on a model checker that analyzes an abstract model and produce a counterexample if the abstract model violates the safety requirement. In our case this is a model checker for rectangular hybrid automata that produces counterexamples. The only model checkers for rectangular automata that produce counterexample that we are aware of are HyTech [21] and the old version of HARE [29] [3]. Unfortunately, because HyTech is not being actively maintained, it does not have support for numbers of arbitrary size, and so in our experiments we frequently ran into overflow problems. Also, we decided not to use the old version of HARE to model check rectangular automata for two reasons: 1. we wanted to only study the effects of the abstraction techniques introduced in this paper, and not have our results compromised by other simplification steps introduced in [29] like merging control locations and transitions, and ignoring variables. 2. The old version of HARE internally calls HyTech, hence, the overflow error happens when the size of the automaton becomes large as a result of refinements. Therefore, we implemented a new model checker for rectangular hybrid automata. Our implementation uses the Parma Polyhedral Library (PPL) [4] to compute the discrete and continuous pre in rectangular hybrid automata [4], and Z3 [14] to check for fixpoints or intersection with initial states. Starting from the unsafe states, we iteratively compute pre until either a fixed point is found or we reach an initial state. Both of these libraries can handle numbers of arbitrary size. Validation of counterexamples requires computing posts in the concrete affine hybrid automata. For discrete post we use the PPL library, and for the continuous post we call SpaceEx [20] with either Supp or PHAVer [19] scenario. Note that SpaceEx only computes an over-approximation of the continuous post and does not have support for computing under-approximations. Therefore, currently in our tool, we stop when an abstract counterexample is validated using the over-approximation implemented by SpaceEx. Finally, in the current implementation, in order to refine a location we simply halve its invariant along some variable at the point of refinement.

We evaluate our tool against four suites of examples that have been proposed by the community [2,6,18] as benchmarks for model checkers of hybrid systems. Each of these suites is qualitatively different and tests different aspects of the performance of a model checker. They are Tank, Satellite, Heater, and Navigation benchmarks. A short description of each of the benchmarks appears in the Appendix C.

We ran different instances of the above examples on 4 different tools, in addition to HARE — SpaceEx, PHAVer (*i.e.* SpaceEx using PHAVer scenario),

[3] Note that FLOW* produces counterexamples and can even handle non-linear ODEs. But it does not support differential *inclusions* and therefore it is incapable of handling rectangular automata.

[4] Technically, we first convert the problem of computing pre to an equivalent problem of computing post, and then use PPL to find the solution.

	Example Size			HARE		SpaceEx			PHAVer			SpaceEx AGAR				HSolver
Name	Dim.	Locs.	Trns.	Time	Safe	Time	FP.	Safe	Time	FP.	Safe	Merged Locs	Time	FP.	Safe	Time
Tank 14	7	7	12	4	No	4	No	No	56	Yes	No	3	10	Yes	No	---
Tank 16	3	3	6	< 1	Yes	3	No	No	1414	No	Yes	2	1133	No	Yes	---
Tank 17	3	3	6	< 1	Yes	5	No*	Yes	1309	No	Yes	2	1041	No	Yes	---
Satellite 03	4	64	198	91	No	< 1	No	No	1804	No	No	28	> 600	---	---	---
Satellite 04	4	100	307	< 1	Yes	< 1	No*	Yes	< 1	Yes	Yes	91	49	Yes	Yes	---
Satellite 11	4	576	1735	1	Yes	< 1	No*	Yes	< 1	Yes	Yes	449	> 600	---	---	---
Satellite 15	4	1296	3895	2	Yes	< 1	No*	Yes	< 1	Yes	Yes	264	> 600	---	---	---
Heater 01	3	4	6	< 1	No	< 1	No*	No	< 1	Yes	No	---	---	---	---	> 600
Heater 02	3	4	6	< 1	No	10	No	No	< 1	Yes	No	---	---	---	---	> 600
Nav 01	4	25	80	9	Yes	< 1	Yes	Yes	< 1	Yes	Yes	21	5	Yes	Yes	> 600
Nav 08	4	16	48	7	Yes	685	No	Yes	< 1	Yes	Yes	10	< 1	Yes	Yes	> 600
Nav 09	4	9	16	8	Yes	< 1	No	No	< 1	Yes	No	4	< 1	Yes	No	> 600
Nav 13	4	9	18	8	Yes	< 1	No*	Yes	< 1	Yes	Yes	4	< 1	Yes	Yes	> 600
Nav 20	4	33	97	29	Yes	2	No*	Yes	< 1	Yes	Yes	11	< 1	Yes	Yes	> 600

Table 1: Experimental Results. Columns Dim., Locs., and Trns. specify number of respectively variables (dimension), locations, and transitions in each benchmark. Five different Time columns specify amount of time each tool took to solve a problem. Times are all in seconds. ‘< 1’ means less than a second and ‘> 600’ means time out (more than 10 minutes). Also, ‘---’ means one of the following: 1) it could not be run on `HSolver` because of specific features the model has, 2) it could not be run on `SpaceEx AGAR` because we could not find any set of locations that can be merged without causing the tool to terminate unexpectedly, 3) we do not have the data because of `SpaceEx AGAR`’s time out. Four different Safe columns specify the output of each tool. Note that all tools perform some kind of over-approximation. Three FP. columns mean whether or not the corresponding tool reached a fixed-point in its reachability computation. No* in the FP. column of `SpaceEx` means that the tool reached a fixed-point, but it also generates the following warning which invalidates the reliability of its “safe” answer: `WARNING (incomplete output) Reached time horizon without exhausting all states, result is incomplete.`

`SpaceEx AGAR` [7], and `HSolver` [32]. We do not compare with the older version of `HARE`, since it implements a CEGAR algorithm for rectangular hybrid automata and not for affine hybrid automata.

Table 1 shows the results on some of the instances we ran the tools on [5]. All examples were run on a laptop with Intel i5 2.50GHz CPU, 6GB of RAM, and Ubuntu 14.10. The salient observations, based on the experiments reported in Table 1, are summarized below.

1. The Satellite benchmark shows that `HARE` scales up to automata with a large control structure.
2. `HARE` often beats the `SpaceEx` scenario in terms of proving safety or running time. For 4 problems, `HARE` performed faster. For 3 problems both tools have the same time, but in one of them only `HARE` proved safety. For 5 out of the remaining 7 problems in which `SpaceEx` performed faster, only `HARE` proved safety.
3. The `PHAVer` scenario is often faster but there are cases where `HARE` beats `PHAVer`. There are only 4 instances in which `HARE` performed faster, but in 7 examples `PHAVer` performed faster. Also there are 3 cases (including one in which `PHAVer` performed faster) where only `HARE` proved safety.
4. `HARE` often beats `SpaceEx AGAR` in terms of proving safety or running time. In 2 problems, we could not find any two locations such that merging them does not cause `SpaceEx AGAR` to encounter internal error. In 7 problems, `HARE` performed faster. In the remaining 5 problems `SpaceEx AGAR` performed

[5] Due to space limits, the table of full results (including all 65 examples along with quite a few metrics on `HARE`) can be found at the following link: https://uofi.box.com/cegar-hare-tacas-2016. The link also points to a virtual machine containing different scripts to run all the examples on each of 5 tools.

faster, but there is one problem among them for which only HARE proved safety.

5. In some instances, SpaceEx, PHAVer, and SpaceEx AGAR failed to prove safety while HARE did not. There are two reasons for it. Sometimes those three tools fail to reach a fixpoint in the reachability computation. Examples of this are Tank 16-17, Satellite 4,11,15, and Nav 8,9,13,20 for SpaceEx, and Tank 16-17 for both PHAVer and SpaceEx AGAR. The other reason is that sometimes those three tools over-approximate too much. Examples of this is Nav 9 for PHAVer and SpaceEx AGAR. Furthermore, it seems merging locations is a very expensive task in SpaceEx AGAR, which we believe is the main reason for the time outs of this tool.
6. On all our examples, HSolver either timed out or the specific constraints in the model made them unamenable to analysis by HSolver. HSolver is an abstraction based tool that abstracts hybrid automata into finite state, discrete transition systems. It can handle models with non-linear dynamics, and so applies to automata more general than what HARE, SpaceEx, and PHAVer analyze. This suggests that HSolver's algorithm makes certain decisions that are not effective for affine hybrid automata.

6 Conclusion

We presented a new algorithm for model checking safety problems of hybrid automata with affine dynamics and rectangular constraints in a counterexample guided abstraction refinement framework. We show that our algorithm is sound and have implemented it in a tool named HARE. We also compared the performance of our tool with a few state-of-the-art tools. Results show that performance of our tool is promising compared to the other tools (SpaceEx, PHAVer, and HSolver).

In the future, we intend to incorporate certain improvements to our implementation. In particular, we would like to integrate an algorithm for computing an under-approximation of the continuous post. The will allow us to definitively validate abstract counterexamples. Theoretically, we would like to explore the completeness of our algorithm, in terms of finding a concrete counterexample when the concrete system is unsafe. This may require a novel notion of counterexample in the abstract system, which is shortest in terms of the number of edges in the concrete system which do not correspond to self-loops. Our broad future goal is to extend the hybrid abstraction refinement method for non-linear hybrid systems.

7 Acknowledgement

The authors would like to thank Sergiy Bogolomov for help with using the SpaceEx AGAR. We gratefully ackowledge the support of the following grants — Nima Roohi was partially supported by NSF CNS 1329991; Pavithra Prabhakar was partially supported by EU FP7 Marie Curie Career Integration Grant no. 631622 and NSF CAREER 1552668; and Mahesh Viswanathan was partially supported by NSF CCF 1422798 and AFOSR FA9950-15-1-0059.

References

1. Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A., Ho, P.H., Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: The algorithmic analysis of hybrid systems. TCS 138(1), 3–34 (1995)
2. Alur, R., Dang, T., Ivančić, F.: Predicate abstraction for reachability analysis of hybrid systems. ACM Trans. Embed. Comput. Syst. 5(1), 152–199 (Feb 2006)
3. Asarin, E., Maler, O., Pnueli, A.: Reachability analysis of dynamical systems having piecewise-constant derivatives. TCS 138(1), 35–65 (1995)
4. 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)
5. Ball, T., Rajamani, S.: Bebop: A symbolic model checker for Boolean programs. In: Proc. of the SPIN. pp. 113–130 (2000)
6. Bogomolov, S., Donze, A., Frehse, G., Grosu, R., Johnson, T.T., Ladan, H., Podelski, A., Wehrle, M.: Guided search for hybrid systems based on coarse-grained space abstractions. International Journal on Software Tools for Technology Trans fer (Oct 2014)
7. Bogomolov, S., Frehse, G., Greitschus, M., Grosu, R., Pasareanu, C.S., Podelski, A., Strump, T.: Assume-guarantee abstraction refinement meets hybrid systems. In: 10th International Haifa Verification Conference. pp. 116–131 (2014)
8. Chen, X., Ábrahám, E., Sankaranarayanan, S.: Flow*: An analyzer for non-linear hybrid systems. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification, Lecture Notes in Computer Science, vol. 8044, pp. 258–263. Springer Berlin Heidelberg (2013)
9. Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings. pp. 154–169 (2000)
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. JFCS 14(4), 583–604 (2003)
11. Clarke, E., Fehnker, A., Han, Z., Krogh, B., Ouaknine, J., Stursberg, O., Theobald, M.: Verification of Hybrid Systems Based on Counterexmple-Guided Abstraction Refinement. In: TACAS. pp. 192–207 (2003)
12. Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-Guided Abstraction Refinement. In: CAV. pp. 154–169 (2000)
13. Corbett, J., Dwyer, M., Hatcliff, J., Laubach, S., Pasareanu, C., Robby, Zheng, H.: Bandera: Extracting finite-state models from Java source code. In: ICSE. pp. 439–448 (2000)
14. De Moura, L., Bjørner, N.: Z3: An efficient smt solver. In: Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 337–340. TACAS'08/ETAPS'08, Springer-Verlag, Berlin, Heidelberg (2008)
15. Dierks, H., Kupferschmid, S., Larsen, K.: Automatic Abstraction Refinement for Timed Automata. In: FORMATS. pp. 114–129 (2007)

16. Doyen, L., Henzinger, T., Raskin, J.F.: Automatic rectangular refinement of affine hybrid systems. In: Pettersson, P., Yi, W. (eds.) Formal Modeling and Analysis of Timed Systems, Lecture Notes in Computer Science, vol. 3829, pp. 144–161. Springer Berlin Heidelberg (2005)
17. Fehnker, A., Clarke, E., Jha, S., Krogh, B.: Refining Abstractions of Hybrid Systems using Counterexample Fragments. In: HSCC 2005. pp. 242–257 (2005)
18. Fehnker, A., Ivancic, F.: Benchmarks for hybrid systems verification. In: In Hybrid Systems: Computation and Control. pp. 326–341 (2004)
19. Frehse, G.: Phaver: Algorithmic verification of hybrid systems past hytech. In: Hybrid Systems: Computation and Control, 8th International Workshop, HSCC 2005, Zurich, Switzerland, March 9-11, 2005, Proceedings. pp. 258–273 (2005)
20. 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: Proc. 23rd International Conference on Computer Aided Verification (2011)
21. Henzinger, T.A., Ho, P.H., Wong-Toi, H.: Hytech: a model checker for hybrid systems. International Journal on Software Tools for Technology Transfer (STTT) 1, 110–122 (1997)
22. Henzinger, T.A., Kopke, P.W., Puri, A., Varaiya, P.: What's decidable about hybrid automata? In: Journal of Computer and System Sciences. pp. 373–382. ACM Press (1995)
23. Henzinger, T., Jhala, R., Majumdar, R., Sutre, G.: Lazy Abstraction. In: POPL 2002. pp. 58–70 (2002)
24. Holzmann, G., Smith, M.: Automating software feature verification. Bell Labs Technical Journal 5(2), 72–87 (2000)
25. Jha, S.K., Krogh, B.H., Weimer, J.E., Clarke, E.M.: Reachability for linear hybrid automata using iterative relaxation abstraction. In: HSCC 2007. pp. 287–300 (2007)
26. Johnson, T.T., Green, J., Mitra, S., Dudley, R., Erwin, R.S.: Satellite rendezvous and conjunction avoidance: Case studies in verification of nonlinear hybrid systems. In: Proceedings of the 18th International Conference on Formal Methods (FM 2012) (2012)
27. Mysore, V., Pnueli, A.: Refining the undecidability frontier of hybrid automata. In: FSTTCS. pp. 261–272 (2005)
28. 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, vol. 346, pp. 55–78 (2015)
29. Prabhakar, P., Duggirala, P.S., Mitra, S., Viswanathan, M.: Hybrid automata-based CEGAR for rectangular hybrid systems. In: VMCAI. pp. 48–67 (2013)
30. Puri, A., Borkar, V.S., Varaiya, P.: Epsilon-approximation of differential inclusions. In: Hybrid Systems III: Verification and Control. pp. 362–376 (1995)
31. Puri, A., Varaiya, P., Borkar, V.: ϵ-approximation of differential inclusions (1995)
32. Ratschan, S., She, Z.: Safety verification of hybrid systems by constraint propagation based abstraction refinement. ACM Transactions in Embedded Computing Systems 6(1) (2007)
33. Segelken, M.: Abstraction and Counterexample-guided Construction of Omega-Automata for Model Checking of Step-discrete linear Hybrid Models. In: CAV. pp. 433–448 (2007)
34. Sorea, M.: Lazy approximation for dense real-time systems. In: Proc. of FORMATS/FTRFTS. pp. 363–378 (2004)
35. Vladimerou, V., Prabhakar, P., Viswanathan, M., Dullerud, G.: STORMED hybrid systems. In: ICALP. pp. 136–147 (2008)

36. Zutshi, A., Deshmukh, J.V., Sankaranarayanan, S., Kapinski, J.: Multiple shooting, CEGAR-based falsification for hybrid systems. In: Proceedings of the 14th International Conference on Embedded Software (2014)

A Need for Annotated Counterexamples in Refinement

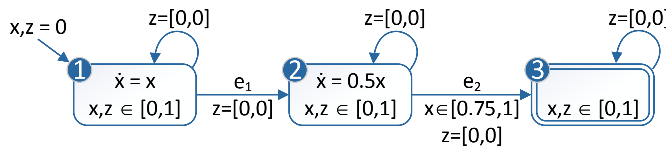
In a CEGAR framework, validating a counterexample typically involves just performing a forward search (with respect to the transitions in the counterexample) in the concrete system; henceforth called the *standard validation algorithm*. However, in the validation algorithm outlined in Section 4.4 is different. The forward search in the concrete system uses a sequence of sets of abstract states computed using backward reachability in abstract model (i.e., the annotated counterexample). Thus, validation for us involves doing a backward search in the abstract model, *and* a forward search in the concrete model. The reason for doing this is because the standard validation algorithm fails to identify the correct refinement step in our case, where the counterexample corresponds to infinitely many executions.

We illustrate this through an example affine hybrid automaton C shown in Figure 2a. The automaton has 3 states $\{1, 2, 3\}$, with 1 as the only initial location, and 3 as the only unsafe location. The automaton has two variables x and z. z is a clock that is used to ensure that a discrete transition is taken every 1 unit of time; invariants of z in each location, and its guards and resets on every transition are set to ensure this. The dynamics of variable x in 1 are given by the equation $\dot{x} = x$, while that in location 2 are given by $\dot{x} = 0.5x$; the dynamics in 3 is not important as that is the unsafe location. The invariant in all locations is $x \in [0, 1]$ and $z \in [0, 1]$. There are 5 transitions: self loop on every location, transition e_1 from 1 to 2 and the transition e_2 from 2 to 3. All transitions reset z to 0, and leave x unchanged. All transitions except transition e_2 are always enabled; the transition e_2 is enabled when $x \in [0.75, 1]$. Assuming that the initial value of x is 0 in location 1 [6], the automaton is safe. This is because no matter how many discrete transitions are taken, the value of x remains 0 in both locations 1 and 2, and so the transition from 2 to 3 is never taken.

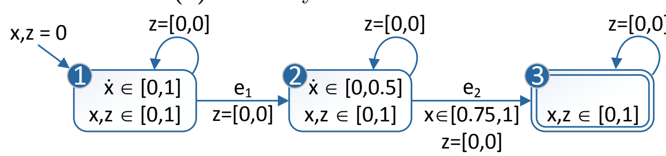
Consider the trivial invariant partition P for this automaton that leaves the invariant for each location intact. The rectangular automaton $A = \alpha(C, P)$ is shown in Figure 2b. The only difference between A and C is in the dynamics for variables x in locations 1 and 2 — the dynamics in 1 is given by $\dot{x} \in [0, 1]$ and in 2 by $\dot{x} \in [0, \frac{1}{2}]$. Observe that A is unsafe because of the execution

$$\begin{aligned}(1, x = 0) &\xrightarrow{0.5} (1, x = 0.5) \xrightarrow{e_1} (2, x = 0.5) \\ &\xrightarrow{0.5} (2, x = 0.75) \xrightarrow{e_2} (3, x = 0.75)\end{aligned}$$

[6] Our model does not have initial values for continuous variables. But having initial values can easily be ensured by adding a new initial location, where the invariant for the variables is constrained to be the initial value. We did not do this here to keep the number of locations small.



(a) Affine hybrid automaton C



(b) Rectangular automaton $A = \alpha(C, P)$

Fig. 2: Need for annotated counterexamples in refinement. $\dot{z} = 1$ in all locations.

In the above execution, we have skipped the value of variable z, since it does not play an important role. Thus, the abstract counterexample is $\pi = e_1, e_2$. The annotated counterexample ρ corresponding to π is

$$(1, x \in [0,0]) \xrightarrow{\leq 1} (1, x \in [0.25, 1]) \xrightarrow{e_1} (2, x \in [0.25, 1]) \\ \xrightarrow{\leq 1} (2, x \in [0.75, 1]) \xrightarrow{e_2} (3, x \in [0.75, 1])$$

Now doing the standard validation algorithm for the counterexample π using forward search in C results in the following sequence.

$$(1, x \in [0,0]) \xrightarrow{\leq 1} (1, x \in [0,0]) \xrightarrow{e_1} (2, x \in [0,0]) \\ \xrightarrow{\leq 1} (2, x \in [0,0]) \xrightarrow{e_2} (3, x \in \emptyset)$$

The counterexample π is spurious because the set $(2, x \in [0,0])$ does not intersect the guard of e_2 suggesting that the dynamics in location 2 needs to be refined. How do we refine? We need to ensure that in the new abstract model (after refinement), there is no way to reach the guard of e_2 from the state $(2, x = 0)$, which is the entry state for location 2. But notice that the set of states reachable in A from $(2, x = 0)$ is $(2, x \in [0, 0.5])$ which is already disjoint from guard of e_2. So no matter how we refine A will not make any progress from the standpoint of eliminating the counterexample π.

On the other hand, our validation algorithm will correctly identify that what needs to be refined is the dynamics in location 1 (and not 2). Recall that our validation algorithm does a forward search in C, and each post computation is intersected with the corresponding set in the annotated counterexample ρ. This will result in the following sequence.

$$(1, x \in [0,0]) \xrightarrow{\leq 1} (1, x \in \emptyset) \xrightarrow{e_1} (2, x \in \emptyset) \\ \xrightarrow{\leq 1} (2, x \in \emptyset) \xrightarrow{e_2} (3, x \in \emptyset)$$

Thus, the reason π is spurious is because the set of states reachable in C from $(1, x = 0)$ (without any discrete transitions) is $(1, x = 0)$ which is disjoint from the set of all abstract states $(1, x \in [0.25, 1])$ that can exhibit the transitions e_1, e_2. Thus, we need to refine the dynamics in location 1 in such a way that the set of states reachable from $(1, x = 0)$ (initial state) is disjoint from the set $(1, x \in [0.25, 1])$. This is easily achieved by splitting the invariant of location 1 into $[0, 0.2]$ and $[0.2, 1]$.

B Time-Bounded Transitions - Formal Construction

Section 4.1 talks about the requirement of bounding duration of time transitions in hybrid automata. We mentioned that for any hybrid automaton H and $t \in \mathbb{R}_+$, one can always construct hybrid automaton H' with the same reachability information and with the additional property that no time transition in H' can take longer than t units of time [7]. To construct H', we add a new variable z to H. It will be a clock, initially 0, with no guard but always reset to 0 on transitions. Furthermore, its invariant puts an upper bound on possible values of z which makes duration of any continuous trajectory finite. More precisely, H' is constructed in the following way:

- $\mathtt{X}_{H'} = \mathtt{X}_H \cup \{z\}$ assuming $z \notin \mathtt{X}_H$,
- $\mathtt{Q}^{\mathsf{init}}_{H'} = \mathtt{Q}^{\mathsf{init}}_{H}$,
- $\mathtt{Q}_{H'} = \mathtt{Q}_H$,
- $\mathtt{Q}^{\mathsf{bad}}_{H'} = \mathtt{Q}^{\mathsf{bad}}_{H}$,
- $\forall q, x \bullet \mathtt{I}_{H'}(q, x) = \mathtt{I}_H(q, x)$ if $x \neq z$ and $[0, t]$ otherwise,
- $\forall q, x, \nu \bullet \mathtt{F}_{H'}(q, \nu)(x) = \mathtt{F}_H(q, \nu)(x)$ if $x \neq z$ and $\{1\}$ otherwise, and
- $\forall e \in \mathtt{E}_H \bullet (\mathtt{S}_H e, \mathtt{D}_H e, g, j, r) \in \mathtt{E}_D$ where
  - $g = \mathtt{G}_H e \cup \{(z, (-\infty, \infty))\}$
  - $r = \mathtt{R}_H e \cup \{(z, [0, 0])\}$
  - $j = \mathtt{J}_H e \cup \{z\}$

Because z is a clock and its invariant restricts its value to be always between 0 and t, we know that unbounded time continuous post becomes equivalent to continuous post for at most t units of time.

C Benchmarks

In this section we provide a short description of each of four class of benchmarks we considered in this paper.

Tank Benchmark [6]. Each problem in this benchmark consists of some $N \in \mathbb{N}$ tanks. Each tank $i \in \{1, \ldots, N\}$ loses volume x_i at some constant flow rate v_i. Hence, dynamics of tank i is $\dot{x}_i = -v_i$ for a rational constant $v_i \geq 0$. Furthermore, one of the tanks is filled from an external inlet at some constant flow rate w which makes its dynamics $\dot{x}_i = w - v_i$, for a rational constant $w \geq 0$. The volume lost by each tank simply vanishes and does not move from one tank to another.

Satellite Benchmark [6]. These examples model two satellites orbiting the earth with nonlinear dynamics described by Kepler's laws (see [26] for details).

[7] One can merge new edges in this construction with trivial self loops mentioned in Section 4.2.

The nonlinear dynamics were hybridized in [6] to generate an affine hybrid automaton. The size of the problems varies from 36 to 1296 locations and so this benchmark can test the scalability of the tool. The safety property being checked is collision avoidance, *i.e.*, whether there is a trajectory in which satellites come too much close to each other.

Heater Benchmark [6]. There are three rooms with three heaters. For each room, we have one automaton with two states modeling heater being on and off in that room. Composition of these three room automata gives us a heater system.

Navigation Benchmark [2,18]. This benchmark considers a robot moving in the $\mathbb{R}^2$ plane. There is a desired velocity v_d that is determined by the current location of the object in an $n \times m$ grid. Each grid has one of the 8 possible desired velocities pointing to the usual 8 possible directions in the plane. Dynamics of object's velocity is determined by $\dot{v} = A(v - v_d)$ where $A \in \mathbb{Q}^2$. There are two special type of cells. Those that are unsafe and those that are blocked. Some of the problems in this class use the following variation: For an small value $\epsilon \in \mathbb{Q}_+$, neighbor cells overlap with each other. This introduces non-nondeterminism into the model.

D Proofs

Proposition 9. *$R'_n = \emptyset$ in ρ' implies there exists i such that 1. $R'_i = \emptyset$, 2. $R_i \neq \emptyset$, 3. $\forall j < i \bullet R_j, R'_j \neq \emptyset$, and 4. $\mathsf{cpost}_C(R_i)$ and $\gamma_A(S'_i)$ are nonempty disjoint sets.*

Proof. We know that for all i neither S_i nor S'_i is empty. We prove that if $R'_{i-1} \neq \emptyset$ then $R_i \neq \emptyset$. By definition we have 1. $R'_{i-1} \subseteq \gamma_A(S'_{i-1})$, and 2. $\forall \nu_1 \in S'_{i-1} \bullet \exists \nu_2 \in S_i \bullet \nu_1 \xrightarrow{e_i} \nu_2$. Note that we do not change guards and resets in abstraction. Therefore, R'_{i-1} is a subset of states that can take edge e'_i and after transition they go to $\gamma_A(S_i)$, which means $R_i \neq \emptyset$. This proves the first three parts. Let i be such that those parts hold. Note that $\gamma_A(S'_i) \neq \emptyset$, and if $R_i \neq \emptyset$ then $\mathsf{cpost}_C(R_i) \neq \emptyset$. Furthermore, $\gamma_A(S'_i)$ and $\mathsf{cpost}_C(R_i)$ are disjoint, since, otherwise R'_i would be non-empty. □

Lemma 11. *There exists $\epsilon \in \mathbb{R}_+$ such that $\mathtt{dist_H}(\mathsf{cpost}_C(R_i), \gamma_A(S'_i)) > \epsilon$.*

Proof. Invariants, guards, and resets of the affine hybrid automaton C are all closed and bounded (by definition). Hence, the invariants, guards, resets, and flow of the abstract rectangular automaton $A = \alpha(C, P)$ are also closed and bounded. Therefore, each of the sets S_i, S'_i and R_i, R'_i are compact (closed and bounded). Since, $\mathsf{cpost}_C(R_i)$ and $\gamma_A(S'_i)$ are compact sets, there exists a minimum distance between them. □

Proposition 12. *Let π be a counterexample of A and ρ its annotation. Let R be the refinement constructed by our strategy after ρ is found to be spurious. Let π' be a counterexample of R that corresponds to π, and let ρ' be its annotation. Then, $\mathtt{por}_{C,R}(\rho') < \mathtt{por}_{C,A}(\rho)$.*

Proof. Let $i = \mathtt{por}_{C,A}(\rho)$, and let R_i and S'_i be sets as defined in the validation algorithm. Let k be the witness for the correspondence between π' and π. Observe that our refinement strategy ensures that the set of states that can reach S'_i through the path $\pi'[i, i+k]$ is disjoint from $\gamma_A^{-1}(R_i)$. Hence if the sequence $U_0, U'_0, U_1, \dots U_m$ is computed by the validation algorithm for ρ', we know that $U_i = \emptyset$. Using Proposition 9, we can conclude that $\mathtt{por}_{C,R}(\rho') < i$. □

Corollary 14 (Existence of Refinement)**.** *There always exists a partition P' satisfying the conditions outlined in the refinement strategy.*

Proof. The result follows from Theorem 13 and Lemma 11. □