Counter-Example Guided Predicate Abstraction of Hybrid Systems*

Rajeev Alur[1], Thao Dang[2], and Franjo Ivančić[1]

[1] University of Pennsylvania
[2] VERIMAG

Abstract. Predicate abstraction has emerged to be a powerful technique for extracting finite-state models from infinite-state systems, and has been recently shown to enhance the effectiveness of the reachability computation techniques for hybrid systems. Given a hybrid system with linear dynamics and a set of linear predicates, the verifier performs an on-the-fly search of the finite discrete quotient whose states correspond to the truth assignments to the input predicates. The success of this approach depends on the choice of the predicates used for abstraction. In this paper, we focus on identifying these predicates automatically by analyzing spurious counter-examples generated by the search in the abstract state-space. We present the basic techniques for discovering new predicates that will rule out closely related spurious counter-examples, optimizations of these techniques, implementation of these in the verification tool, and case studies demonstrating the promise of the approach.

1 Introduction

Inspired by the success of model checking in hardware verification and protocol analysis [12, 20], there has been increasing research on developing tools for automated verification of hybrid (mixed discrete-continuous) models of embedded controllers [1, 4, 6, 9, 16, 19, 22]. Model checking requires the computation of the set of reachable states of a model, and in presence of continuous dynamics, this is typically undecidable. Consequently, contemporary tools for model checking of hybrid systems, such as CheckMate[9] and `d/dt`[6], approximate the set of reachable states by polyhedra. It has been shown that effectiveness of the reachability computation for hybrid systems can be enhanced using predicate abstraction [3]. Predicate abstraction is a powerful technique for extracting finite-state models from complex, potentially infinite-state, discrete systems (see, for instance, [14, 23]), and tools such as Bandera [13], SLAM [7], and Feaver [21] have used it for analysis of C or Java programs. The input to our verification tool consists of the concrete system modeled by a hybrid automaton, the safety property to be verified, and a finite set of predicates over system variables to

* This research was supported in part by ARO URI award DAAD19-01-1-0473, DARPA Mobies award F33615-00-C-1707, NSF award ITR/SY 0121431, and European IST project CC (Computation and Control).

be used for abstraction. For the sake of efficiency, we require that all invariants, guards, and discrete updates of the hybrid automaton are specified by linear expressions, the continuous dynamics is linear, possibly with bounded input, and the property as well as the abstraction predicates are linear. An abstract state is a valid combination of truth values to the predicates, and thus, corresponds to a polyhedral set of the concrete state-space. The verifier performs an on-the-fly search of the abstract system by symbolic manipulation of polyhedra.

The core of the verifier is the computation of the transitions between abstract states that capture both discrete and continuous dynamics of the original system. Computing discrete successors is relatively straightforward, and involves computing weakest preconditions, and checking non-emptiness of intersection of polyhedral sets. For computing continuous successors of an abstract state A, we use a strategy inspired by the techniques used in CheckMate and `d/dt`. However, while tools such as `d/dt` are designed to compute a "good" approximation of the continuous successors of A, we are interested in checking if this set intersects with a new abstract state permitting many optimizations. Postulating the verification problem for hybrid systems as a search problem in the abstract system has many benefits compared to the traditional approach of computing approximations of reachable sets, and our experiments indicate significant improvements in time and space requirements compared to a tool such as `d/dt`.

The success of our scheme crucially depends on the choice of the predicates used for abstraction. In this paper, we focus on identifying such predicates automatically by analyzing spurious counter-examples generated by the search in the abstract state-space. Counter-example guided refinement of abstractions has been used in multiple contexts before, for instance, to identify the relevant timing constraints in verification of timed automata [5], to identify the relevant boolean predicates in verification of C programs [7], and to identify the relevant variables in symbolic model checking [11]. We present the basic techniques for analyzing counter-examples, techniques for discovering new predicates that will rule out spurious counter-examples, optimizations of these techniques, implementation of these in our verifier, and case studies demonstrating the promise of the approach. Counter-example guided refinement of abstractions for hybrid systems is being independently explored by the hybrid systems group at CMU [10].

The abstract counter-example consists of a sequence of abstract states leading from an initial state to a state violating the property. The analysis problem is to check if the corresponding sequence can be traversed in the concrete system. We perform a forward search from the initial abstract state following the given counter-example. The analysis relies on techniques for polyhedral approximations of the reachable sets under continuous dynamics. We also implemented a local test that checks for feasibility of *pairwise transitions*, and this proves to be effective in many cases. If the counter-example is found to be infeasible, then we wish to identify new predicates that would rule out this sequence in the refined abstract space. This reduces to the problem of finding predicates that *separate* two sets of polyhedra. We present a greedy strategy for identifying such predicates. After discovering new predicates, we include these to the set of predicates used before, and rerun the search in the refined abstract state-space. We demonstrate the feasibility using three case studies. The first one involves verification of a parametric version of Fischer's protocol. The second one involves analysis of a thermostat model, and the third analyzes a model of an adaptive cruise controller. In each of these cases, we show how counter-example analysis can be effective in discovering the predicates that are needed for establishing safety.

2 Predicate Abstraction for Linear Hybrid Systems

In this section, we briefly recap the definitions of predicate abstraction for linear hybrid systems and the search strategy in the abstract space as outlined in [3].

2.1 Mathematical Model

We denote the set of all n-dimensional linear expressions $l : \mathbb{R}^n \to \mathbb{R}$ with Σ_n and the set of all n-dimensional linear predicates $\pi : \mathbb{R}^n \to \mathbb{B}$, where $\mathbb{B} := \{0, 1\}$, with $\mathcal{L}_n$. A linear predicate is of the form $\pi(x) := \sum_{i=1}^{n} a_i x_i + a_{n+1} \sim 0$, where $\sim \in \{\geq, >\}$ and $\forall i \in \{1, \ldots, n+1\} : a_i \in \mathbb{R}$. The set of finite sets of n-dimensional linear predicates is denoted by $\mathcal{C}_n$, where an element of $\mathcal{C}_n$ represents the conjunction of its elements.

Definition 1 (Linear Hybrid System). *An n-dimensional* **linear hybrid system** *is a tuple $H = (\mathcal{X}, L, X_0, I, f, T)$ with the following components:*

- *$\mathcal{X} \subseteq \mathbb{R}^n$ is a convex polyhedron representing the* **continuous state-space***.*
- *L is a finite set of* **locations***. The* **state-space** *of H is $X = L \times \mathcal{X}$. Each state thus has the form (l, x), where $l \in L$ is the discrete part of the state, and $x \in \mathcal{X}$ is the continuous part.*
- *$X_0 \subseteq X$ is the set of* **initial states***. It is assumed that for all locations $l \in L$, the set $\{x \in \mathcal{X} \mid (l, x) \in X_0\}$ is a convex polyhedron.*
- *$I : L \to \mathcal{C}_n$ assigns to each location $l \in L$ a finite set of linear predicates $I(l)$ defining the* **invariant** *conditions that constrain the value of the continuous part of the state while the discrete location is l. The hybrid automaton can only stay in location l as long as the continuous part of the state x satisfies $I(l)$, i.e. $\forall \pi \in I(l) : \pi(x) = 1$. We write $\mathcal{I}_l$ for the invariant set of location l, that is the set of all points x satisfying all predicates in $I(l)$.*
- *$f : L \to (\mathbb{R}^n \to \mathbb{R}^n)$ assigns to each location $l \in L$ a* **continuous vector field** *$f(l)$ on x. While at location l the evolution of the continuous variable is governed by the differential equation $\dot{x} = f(l)(x)$. We restrict our attention to hybrid automata with linear continuous dynamics, that is, for every location $l \in L$, the vector field $f(l)$ is linear, i.e. $f(l)(x) = A_l x$ where A_l is an $n \times n$ matrix. The analysis can also be applied to systems having linear continuous dynamics with uncertain, bounded input of the form $\dot{x} = A_l x + B_l u$.*
- *$T \subseteq L \times L \times \mathcal{C}_n \times (\Sigma_n)^n$ is a relation capturing discrete transition jumps between two discrete locations. A transition $(l, l', g, r) \in T$ consists of an initial location l, a destination location l', a set of* **guard** *constraints g and*

a linear **reset** *mapping* r*. From a state* (l, x) *where all predicates in* g *are satisfied the hybrid automaton can jump to location* l' *at which the continuous variable* x *is reset to a new value* $r(x)$*. We write* $\mathcal{G}_t \subseteq \mathcal{I}_l$ *for the guard set of a transition* $t = (l, l', g, r) \in T$ *which is the set of points satisfying all linear predicates of* g *and the invariant of the location* l*.*

2.2 Transition System Semantics

We define the semantics of a hybrid automaton by formalizing its underlying transition system. For simplicity we consider the system $\dot{x} = A_l x$, and we denote the flow of this system with $\Phi_l(x, t) = e^{A_l t} x$. The underlying transition system of H is $T_H = \{X, \rightarrow, X_0\}$. The state-space of the transition system is the state-space of H, i.e. $X = L \times \mathcal{X}$. The transition relation $\rightarrow \subseteq X \times X$ between states of the transition system is defined as the union of two relations $\rightarrow_C, \rightarrow_D \subseteq X \times X$. The relation $\rightarrow_C$ describes transitions due to continuous flows, whereas $\rightarrow_D$ describes transitions due to discrete jumps.

$$
\begin{aligned}
(l, x) \rightarrow_C (l, y) \ &\text{ iff } \ \exists t \in \mathbb{R}_{\geq 0} : \Phi_l(x, t) = y \wedge \forall t' \in [0, t] : \Phi_l(x, t') \in \mathcal{I}_l. \\
(l, x) \rightarrow_D (l', y) \ &\text{ iff } \ \exists (l, l', g, r) \in T : x \in \mathcal{G}_t \wedge y = r(x) \wedge y \in \mathcal{I}_{l'}.
\end{aligned}
$$

2.3 Discrete Abstraction

We define a discrete abstraction of the hybrid system $H = (\mathcal{X}, L, X_0, I, f, T)$ with respect to a given k-dimensional vector of n-dimensional linear predicates $\Pi = (\pi_1, \pi_2, \ldots, \pi_k) \in (\mathcal{L}_n)^k$. We can partition the continuous state-space $\mathcal{X} \subseteq \mathbb{R}^n$ into at most 2^k states, corresponding to the 2^k possible boolean evaluations of Π; hence, the infinite state-space X of H is reduced to $|L|2^k$ states in the abstract system. From now on, we refer to the hybrid system H as the *concrete system* and its state-space X as the *concrete state-space*.

Definition 2 (Abstract state-space). *Given an* n*-dimensional hybrid system* $H = (\mathcal{X}, L, X_0, f, I, T)$ *and a* k*-dimensional vector* $\Pi \in (\mathcal{L}_n)^k$ *of* n*-dimensional linear predicates we can define an* **abstract state** *as a tuple* $(l, \boldsymbol{b})$*, where* $l \in L$ *and* $\boldsymbol{b} \in \mathbb{B}^k$*. The abstract state-space for a* k*-dimensional vector of linear predicates hence is* $Q_\Pi := L \times \mathbb{B}^k$*. We define a* **concretization function** $C_\Pi : \mathbb{B}^k \rightarrow 2^{\mathcal{X}}$ *for a vector of linear predicates* $\Pi = (\pi_1, \ldots, \pi_k) \in (\mathcal{L}_n)^k$ *as* $C_\Pi(\boldsymbol{b}) := \{x \in \mathcal{X} \mid \forall i \in \{1, \ldots, k\} : \pi_i(x) = b_i\}$*.*

Definition 3 (Discrete Abstraction). *Given a hybrid system* $H = (\mathcal{X}, L, X_0, f, I, T)$*, we define its abstract system with respect to a vector of linear predicates* Π *as the transition system* $H_\Pi = (Q_\Pi, \xrightarrow{\Pi}, Q_0)$ *where*

- *the set of initial states is* $Q_0 = \{(l, \boldsymbol{b}) \in Q_\Pi \mid \exists x \in C_\Pi(\boldsymbol{b}) : (l, x) \in X_0\}$*;*
- *the abstract transition relation* $\xrightarrow{\Pi} \subseteq Q_\Pi \times Q_\Pi$ *is defined as the union of the following two relations* $\xrightarrow{\Pi}_D, \xrightarrow{\Pi}_C \subseteq Q_\Pi \times Q_\Pi$*. The relation* $\xrightarrow{\Pi}_D$ *represents*

transitions in the abstract state-space due to discrete jumps, whereas $\stackrel{\Pi}{\to}_C$ *represents transitions due to continuous flows:*

$$\begin{aligned}(l,\boldsymbol{b}) \stackrel{\Pi}{\to}_D (l',\boldsymbol{b'}) \;\; &\textit{iff} \;\; \exists(l,l',g,r) \in T, x \in C_\Pi(\boldsymbol{b}) \cap \mathcal{G}_t : \\ &\qquad (l,x) \to_D (l', r(x)) \wedge r(x) \in C_\Pi(\boldsymbol{b'}); \\ (l,\boldsymbol{b}) \stackrel{\Pi}{\to}_C (l,\boldsymbol{b'}) \;\; &\textit{iff} \;\; \exists x \in C_\Pi(\boldsymbol{b}), t \in \mathbb{R}_{\geq 0} : \Phi_l(x,t) \in C_\Pi(\boldsymbol{b'}) \wedge \\ &\qquad \forall t' \in [0,t] : \Phi_l(x,t') \in \mathcal{I}_l.\end{aligned}$$

2.4 Searching the Abstract State-Space

Given a hybrid system H we want to verify certain safety properties. We define a property by specifying a set of *unsafe locations* $U \subseteq L$ and a convex set $\mathcal{B} \subseteq \mathcal{X}$ of *unsafe continuous states*. The property is said to hold for the hybrid system H iff there is no valid trace that leads to some state in $\mathcal{B}$ while in an unsafe location. We implemented an on-the-fly search of the abstract state-space giving priority to computing discrete successors rather than continuous successors, as this is generally much faster. Computing discrete successors is relatively straightforward, and involves computing weakest preconditions, and checking non-emptiness of intersection of polyhedral sets. For computing continuous successors of an abstract state A, we compute the polyhedral slices of states reachable at fixed times $r, 2r, 3r, \ldots$ for a suitably chosen r, and then, take convex-hull of all these polyhedra to over-approximate the set of all states reachable from A. We are only interested in checking if this set intersects with a new abstract state. This approach has many benefits compared to the traditional approach of computing approximations of reachable sets, one of them being the fact that the expensive operation of computing continuous successors is applied only to abstract states, and not to intermediate polyhedra of unpredictable shapes and complexities.

We include an optimization technique in the search strategy. For each concrete counter-example in the concrete hybrid system, there exists an equivalent counter-example that has the additional constraint that there are no two consecutive transitions due to continuous flow. This is due to the additivity of flows of hybrid systems, namely $(l,x) \to_C (l,x') \wedge (l,x') \to_C (l,x'') \Rightarrow (l,x) \to_C (l,x'')$. We are hence searching only for counter-examples in the abstract system that do not have two consecutive transitions due to continuous flow.

3 Counter-Example Analysis

An abstract counter-example consists of a sequence of abstract states and transitions leading from an initial state to a state violating the property. The analysis problem is to check if the corresponding sequence of modes and discrete switches can be traversed in the concrete system. The analysis relies on techniques for polyhedral approximations of the reachable sets under continuous dynamics. To speed up the feasibility analysis, we also implemented a local test that checks for feasibility of *pairwise transitions*, and this often proves to be effective.

3.1 Global Analysis Algorithm

We denote the set of transition labels Σ_T as $\Sigma_T = T \cup \{C\}$, denoting that either a discrete transition or a continuous transition occurred. For the subsequent definitions of counter-examples in the abstract state-space we use the following notation for transitions due to discrete jumps for $t = (l, l', g, r) \in T$:

$$(l, \boldsymbol{b}) \overset{\Pi}{\rightarrow}_t (l', \boldsymbol{b'}) \text{ iff } \exists x \in C_\Pi(\boldsymbol{b}) \cap \mathcal{G}_t : (l, x) \rightarrow_D (l', r(x)) \wedge r(x) \in C_\Pi(\boldsymbol{b'}).$$

Definition 4. *An* **abstract path** *p in the abstract state-space given by the vector of predicates Π of length $n \geq 0$ is a pair $(\boldsymbol{a}, \boldsymbol{t}) \in (Q_\Pi)^{n+1} \times (\Sigma_T)^n$, such that $\boldsymbol{a} = (a_0, \ldots, a_n)$ and $\boldsymbol{t} = (t_0, \ldots, t_{n-1})$ with $t_i \in \Sigma_T$, $a_0 = (l_0, \boldsymbol{b_0}) \in Q_0$, and $\forall 0 \leq i \leq n-1 : a_i \overset{\Pi}{\rightarrow}_{t_i} a_{i+1}$. The set of abstract paths of length n given by the vector of predicates Π is denoted by $\mathcal{P}_n^\Pi$. A* **counter-example** *is an abstract path $p = (\boldsymbol{a}, \boldsymbol{t}) = ((a_0, \ldots, a_n), (t_0 \,, \ldots, t_{n-1}))$, such that a_n is a violation of the property to be proven. We call the sequence of abstract states $\boldsymbol{a} = (a_0, \ldots, a_n)$ of a counter-example $p = (\boldsymbol{a}, \boldsymbol{t})$ an unlabeled counter-example.*

The counter-example analysis problem is twofold. The first objective is to check whether a counter-example in the abstract system corresponds to a counter-example in the concrete system. In case that the analysis finds that this particular counter-example cannot be traversed in the concrete system, we want the analysis to identify one or more new predicates that would rule out *closely related* counter-examples in the refined abstract state-space. The refined abstract state-space is defined by adding these predicates to the previous set of predicates used in the abstract state-space search. We define the notion of refinement between abstract paths to formalize the concept of closely related abstract paths.

Definition 5. *A vector of predicates $\Pi' \in (\mathcal{L}_n)^{k'}$ refines a vector of predicates $\Pi \in (\mathcal{L}_n)^k$, if its corresponding set of predicates includes all predicates in Π.*

Definition 6. *An abstract state $a' = (l', \boldsymbol{b'}) \in Q_{\Pi'}$ for the vector of predicates Π' refines another abstract state $a = (l, \boldsymbol{b}) \in Q_\Pi$ for the vector of predicates Π, iff $l = l'$ and $C_{\Pi'}(\boldsymbol{b'}) \subseteq C_\Pi(\boldsymbol{b})$.*

Definition 7. *An abstract path $p' = ((a'_0, \ldots, a'_n), (t'_0, \ldots, t'_{n-1})) \in \mathcal{P}_n^{\Pi'}$ for a vector of predicates Π' refines another abstract path $p = ((a_0, \ldots, a_n), (t_0, \ldots, t_{n-1})) \in \mathcal{P}_n^\Pi$ for a vector of predicates Π, with $a_i = (l_i, \boldsymbol{b_i})$ and $a'_i = (l'_i, \boldsymbol{b'_i})$, iff Π' refines Π, $\forall 0 \leq i \leq n : a'_i$ refines a_i, and $\forall 0 \leq i \leq n-1 : t'_i = t_i$.*

During the counter-example analysis we define $\texttt{Pre} : Q_\Pi \times \Sigma_T \times Q_\Pi \rightarrow 2^{\mathcal{X}}$ and $\texttt{Post} : 2^{\mathcal{X}} \times \Sigma_T \times Q_\Pi \rightarrow 2^{\mathcal{X}}$ functions that consider only the abstract states or the concretely reachable state space rather than the whole continuous state-space $\mathcal{X}$. The computation of these takes into consideration the concretization of the abstract state, as well as the invariants and guards of the system. We define the functions Pre and Post with $a = (l, \boldsymbol{b})$ and $a' = (l', \boldsymbol{b'})$ as:

$$\texttt{Pre}(a, t, a') = \begin{cases} \left\{ \begin{array}{c} x \in C_{\Pi}(\boldsymbol{b}) \cap \mathcal{G}_t | \\ r(x) \in C_{\Pi}(\boldsymbol{b'}) \cap \mathcal{I}_{l'} \end{array} \right\} & : \quad t = (l, l', g, r); \\ \left\{ \begin{array}{c} x \in C_{\Pi}(\boldsymbol{b}) \cap \mathcal{I}_l | \exists \tau \in \mathbb{R}_{\geq 0} \\ \Phi(x, \tau) \in C_{\Pi}(\boldsymbol{b'}) \cap \mathcal{I}_{l'} \\ \wedge \forall \tau' \in [0, \tau] : \Phi(x, \tau') \in \mathcal{I}_{l'} \end{array} \right\} & : \quad t = C. \end{cases}$$

$$\texttt{Post}(X, t, a') = \begin{cases} \texttt{Post}\left(\left\{ \begin{array}{c} x \in C_{\Pi}(\boldsymbol{b'}) \cap \mathcal{I}_{l'} | \\ \exists y \in \mathcal{G}_t \cap X : x = r(y) \end{array} \right\}, C, a' \right) & : \quad t = (l, l', g, r); \\ \left\{ \begin{array}{c} x \in C_{\Pi}(\boldsymbol{b'}) \cap \mathcal{I}_{l'} | \exists \tau \in \mathbb{R}_{\geq 0} \\ \exists y \in X : \Phi(y, \tau) = x \\ \wedge \forall \tau' \in [0, \tau] : \Phi(y, \tau') \in \mathcal{I}_{l'} \end{array} \right\} & : \quad t = C. \end{cases}$$

Our counter-example analysis algorithm is presented in algorithm 1. The set R_0 is the part of the initial state-space that is covered by the abstract state $(l, \boldsymbol{b_0})$. We then compute the concretely reachable state-space of each abstract state of the counter-example. For each $1 \leq i \leq n$ we compute R_i as the reachable region after i transitions according to the counter-example. It is hence clear that if $R_i = \emptyset$ for some i then the counter-example is spurious.

Algorithm 1 Analyzing a counter-example $p \in \mathcal{P}_n^{\Pi}$

$R_0 = C_{\Pi}(b_0) \cap \{x \in \mathcal{I}_{l_0} | (l_0, x) \in X_0\}$
for $1 \leq i \leq n$ **do**
  $R_i = \texttt{Post}(R_{i-1}, t_{i-1}, a_i)$
  if $R_i = \emptyset$ **then**
    return "Counter-example is spurious!"
  end if
end for
return "Counter-example is concrete!"

In the case that we found that the counter-example is spurious, we want to use the counter-example to find new predicates. Consider a counter-example $p \in \mathcal{P}_n^{\Pi}$, such that $R_{k+1} = \emptyset$ and $R_k \neq \emptyset$ for $0 \leq k < n$. We call t_k the failing transition of the counter-example p. Then we can prove the following lemma.[1]

Lemma 1. *Given a counter-example* $p = (\boldsymbol{a}, \boldsymbol{t}) = ((a_0, \ldots, a_n), (t_0, \ldots, t_{n-1})) \in \mathcal{P}_n^{\Pi}$ *where* t_k *is the failing transition, we have* $R_k \cap \texttt{Pre}(a_k, t_k, a_{k+1}) = \emptyset$.

We want to add new predicates to the vector Π, so that the refined vector Π' does not allow a refined (unlabeled) counter-example of p to reappear. Consider a strategy that adds predicates to the set Π that correspond to a separation of R_k from $\texttt{Pre}(a_k, t_k, a_{k+1})$ for a failing transition t_k. This means that we are looking for a refined set of predicates Π' of Π, such that every refined abstract

[1] Please note, that we omit all the proofs in this paper for the sake of brevity.

state intersects at most with one of the two sets R_k and $\mathtt{Pre}(a_k, t_k, a_{k+1})$. We define the notion of separation in terms of polyhedral sets, since we approximate the set of reachable states by polyhedral slices in the implementation of the tool. It should be noted here that we use under-approximations of the reachable sets of states during the analysis of counter-examples while we over-approximated the reachable sets of states during the search in the abstract state-space.

Definition 8 (Separating predicates). *Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ and $\mathcal{Q} = \{Q_1, \ldots, Q_m\}$ be two disjoint sets of convex polyhedra. We denote by $\bigcup \mathcal{P}$ and $\bigcup \mathcal{Q}$ the union of all polyhedra in $\mathcal{P}$ and $\mathcal{Q}$. A finite vector of linear predicates $\Pi = (\pi_1, \pi_2, \ldots, \pi_k)$ separates $\mathcal{P}$ and $\mathcal{Q}$ iff for all $\boldsymbol{b} \in \mathbb{B}^k$, at least one of the two sets $(C_\Pi(\boldsymbol{b}) \cap \bigcup \mathcal{P})$ and $(C_\Pi(\boldsymbol{b}) \cap \bigcup \mathcal{Q})$ is empty.*

The predicates in Π are called *separating predicates*. Note that such a vector Π always exists[2], but it is often not unique.

Theorem 1. *Assume a counter-example $p \in \mathcal{P}_n^\Pi$ for a vector of predicates Π such that t_k is the failing transition. If Π' refines Π and additionally contains predicates corresponding to a separation of R_k from $\mathtt{Pre}(a_k, t_k, a_{k+1})$, and we find a refined counter-example $p' \in \mathcal{P}_n^{\Pi'}$ of p, then there exists a failing transition t_j in p', such that $j < k$.*

As a single counter-example p is of finite length, the above theorem assures us that after a finite number of iterations, a refinement of p will not be possible.

3.2 Locally Infeasible Abstract States

In this section we present a second counter-example analysis algorithm, which checks a counter-example quickly for a common cause of spurious counter-examples. We also show that this analysis produces new predicates with stronger implications for subsequent searches in the refined abstract state-space.

Definition 9. *For a path $p = (\boldsymbol{a}, \boldsymbol{t}) \in \mathcal{P}_{n+1}^\Pi$ given the vector of predicates Π, with $\boldsymbol{a} = (a_0, \ldots, a_{n+1}) = ((l_0, \boldsymbol{b_0}), \ldots, (l_{n+1}, \boldsymbol{b_{n+1}}))$ and $\boldsymbol{t} = (t_0, \ldots, t_n)$, we say that an abstract state a_i for $1 \leq i \leq n$ is* **locally infeasible***, iff*

$$\mathtt{Post}(C_\Pi(\boldsymbol{b_{i-1}}), t_{i-1}, a_i) \cap \mathtt{Pre}(a_i, t_i, a_{i+1}) = \emptyset.$$

The detection of locally infeasible abstract states can be implemented in a straight-forward fashion. In addition, we can easily compute new predicates that disallow refined counter-examples. If a state a_i is locally infeasible, then we can use the fact that our implemented optimization technique guarantees that either t_{i-1} or t_i is a discrete transition. If t_{i-1} is discrete, one reasonable choice is to use the predicates corresponding to the constraints of the polyhedral sets representing $\mathtt{Post}(C_\Pi(\boldsymbol{b_{i-1}}), t_{i-1}, a_i)$ in the refined search. Otherwise, a possible

[2] It is easy to see that we can simply take the linear constraints of all polyhedra from P or from Q to determine Π.

approach is to use the predicates corresponding to $\texttt{Pre}(a_i, t_i, a_{i+1})$ in the refined search. We denote this strategy of picking new predicates by `LocalStrategy`.

We can now prove the following two theorems about using the strategy `LocalStrategy` in case we find a locally infeasible abstract state. The theorems formalize that by using this strategy we can assure that a refinement of the (unlabeled) counter-example will not be found in subsequent searches.

Theorem 2. *Assume a counter-example $p \in \mathcal{P}_n^{\Pi}$ for a vector of predicates Π, such that there is a locally infeasible abstract state a_i in p. A search in the refined abstract state-space given by the strategy* `LocalStrategy` *to find new predicates will not find a counter-example that is a refinement of p.*

Theorem 3. *Assume a hybrid system $H = (\mathcal{X}, L, X_0, I, f, T)$ with the properties that $\forall l \in L : (l, l', g, r) \in T \Rightarrow l \neq l'$, and $\forall (l, l', g_1, r_1), (l, l', g_2, r_2) \in T : g_1 = g_2 \wedge r_1 = r_2$. Additionally assume a counter-example $p = (\boldsymbol{a}, \boldsymbol{t}) \in \mathcal{P}_n^{\Pi}$ for a vector of predicates Π, such that there is a locally infeasible abstract state a_i in p. Then a search in the refined abstract state-space obtained using the strategy* `LocalStrategy` *to refine the set of predicates Π will not produce a refined unlabeled counter-example.*

4 Computing separating predicates

In the previous section we described two counter-example analysis algorithms. If the counter-example is found to be infeasible, then we wish to identify one or more new predicates that would rule out this sequence in the refined abstract space. This reduces to the problem of finding one or more predicates that *separate* two sets of polyhedra. We present a greedy strategy for identifying the separating predicates. After discovering new predicates, we then include these predicates to the set of predicates used before, and rerun the search in the refined abstract state-space defined by the enriched predicate set.

4.1 Separating two disjoint convex polyhedra

Let P and Q be two disjoint convex polyhedra. To separate them, we define the distance between P and Q as follows: $d(P, Q) = \inf\{d(p - q) \mid p \in P \wedge q \in Q\}$ where $d(\cdot)$ denotes the Euclidean distance. Since P and Q are disjoint, $d(P, Q)$ is positive. Let $p^* \in P$ and $q^* \in Q$ be points that form a pair of closest points. We denote by $s(p^*, q^*)$ the line segment with extreme points p^* and q^*. The half-space $\mathcal{H}$ which is normal to $s(p^*, q^*)$ and has q^* as a supporting point can be written as: $\mathcal{H} = \{x \mid \langle p^* - q^*, x\rangle \geq \langle p^* - q^*, q^*\rangle\}$. We denote by $\overline{\mathcal{H}}$ the complement of $\mathcal{H}$.

Lemma 2. *The polyhedron Q is contained in $\mathcal{H}$ and P is contained in $\overline{\mathcal{H}}$.*

We remark that Lemma 2 also holds for any half-space which is normal to $s(p^*, q^*)$ and passes through an arbitrary point in $s(p^*, q^*)$. Hence, any such half-space can be used to define a separating predicate. To compute $d(P, Q)$ as well as p^* and q^*, there exist efficient algorithms [8] which take time $\mathcal{O}(K_P + K_Q)$ where K_P and K_Q are the number of vertices of P and Q.

4.2 Separating two disjoint sets of convex polyhedra

We proceed with the problem of finding a set Π of separating predicates for two sets of convex polyhedra $\mathcal{P}_1$ and $\mathcal{P}_2$. In order to keep the size of the abstract state space as small as possible, we want to find Π with the smallest number of predicates. Many related polyhedral separation problems have been considered in literature [17, 18]. However, the solutions proposed in these works are only for two and three dimensional polyhedra. On the other hand, even in low dimensions most separation problems were shown to be intractably hard. In three dimensions the problem of finding a minimum facet-separator for two polyhedral solids is NP-complete [15]. Our objective is not to find an optimal solution but to develop methods which are effective on the problem of separating reachable sets of hybrid systems for abstraction refinement purposes.

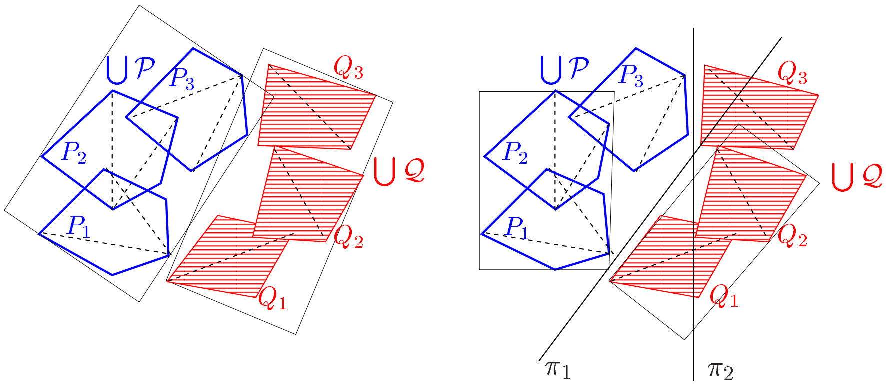

Fig. 1. Subdividing the sets $\mathcal{P} = \{P_1, P_2, P_3\}$ and $\mathcal{Q} = \{Q_1, Q_2, Q_3\}$ respectively into $\{P_1, P_2\}$, $\{P_3\}$ and $\{Q_1, Q_2\}$, $\{Q_3\}$ allows to find two separating predicates π_1 and π_2.

Our solution is based on the following observation. Given two set of polyhedra $\mathcal{P}_1$ and $\mathcal{P}_2$, if the convex hulls of $\mathcal{P}_1$ and $\mathcal{P}_2$ are disjoint, then one can apply the method presented in the previous section to find a separating predicate. If the convex hulls intersect, it is clear that $\mathcal{P}_1$ and $\mathcal{P}_2$ cannot be separated by a single hyperplane. The main idea is to divide $\mathcal{P}_1$ and $\mathcal{P}_2$ into subsets of polyhedra such that their convex hulls do not intersect allowing to find a separating predicate. The procedure of subdivision can be done in a hierarchical way. We begin with all polyhedra in $\mathcal{P}_1$ and $\mathcal{P}_2$ and recursively subdivide until the convex hulls are pairwise disjoint. Moreover, for efficiency purposes, instead of convex hulls, we can use approximations by non-axis-aligned bounding boxes which are easier to compute and test for overlaps (see figure 1). One way to compute tight fitting bounding boxes is to align the axes of the box in the directions along which the vertices of the polyhedra tend to lie. From the vertices of the polyhedra we can determine the matrix of covariance and take its largest eigenvectors to define the orientation of the box.

Our method for computing separating predicates is summarized in algorithm 2. We denote by $\mathcal{H}(\pi)$ the half-space defined by predicate π. Given a set $\mathcal{P}$ of polyhedra, $chull(\mathcal{P})$ and $bbox(\mathcal{P})$ are respectively the convex hull and the non-axis-aligned bounding box of $\mathcal{P}$. The set $\mathcal{S}(\mathcal{P}, \pi) = \{s \in \mathcal{P} \mid s \subseteq \mathcal{H}(\pi)\}$ is the largest subset of $\mathcal{P}$ lying entirely inside $\mathcal{H}(\pi)$, and $Int(\mathcal{P}, \pi) = \{s \cap \mathcal{H}(\pi) \mid s \in \mathcal{P} \ \wedge \ s \cap \mathcal{H}(\pi) \neq \emptyset\}$ is the intersection of $\bigcup \mathcal{P}$ with $\mathcal{H}(\pi)$. The core of the algorithm is a procedure, called *sep*, which computes a separating predicate for two disjoint polyhedra using the method presented in section 4.1. Two sets of polyhedra $\mathcal{P}_1$ and $\mathcal{P}_2$ are called *separable* if $conv\{\mathcal{P}_1\} \cap conv\{\mathcal{P}_2\} \neq \emptyset$ where *conv* is a convex-approximation operation which can be *chull* or *bbox*. The notation $separable(\mathcal{P}_1, \mathcal{P}_2)$ in the algorithm indicates that $\mathcal{P}_1$ and $\mathcal{P}_2$ are separable.

Algorithm 2 Separating$(\mathcal{P}_1, \mathcal{P}_2)$

Step 1. If $separable(\mathcal{P}_1, \mathcal{P}_2)$, compute $\pi = sep(chull\{\mathcal{P}_1\}, chull\{\mathcal{P}_2\})$ and return π.
Step 2. Divide $\mathcal{P}_1$ and $\mathcal{P}_2$ into subsets $\mathcal{P}_{11}$, $\mathcal{P}_{12}$ and $\mathcal{P}_{21}$, $\mathcal{P}_{22}$, respectively.
Step 3. Compute separating predicates for pairs of one set and a subset of the other:

$$\Pi_t = \{\pi = sep(chull\{\mathcal{P}_i\}, chull\{\mathcal{P}_{jk}\}) \mid separable(\mathcal{P}_i, \mathcal{P}_{jk}), 1 \leq i \neq j, k \leq 2\}.$$

If $\Pi_t \neq \emptyset$, go to step 4; otherwise, continue with pairs of subsets:

$$\Pi_t = \{\pi = sep(chull\{\mathcal{P}_{1i}\}, chull\{\mathcal{P}_{2j}\}) \mid separable(\mathcal{P}_{1i}, \mathcal{P}_{2k}), 1 \leq i, j \leq 2\}.$$

If $\Pi_t = \emptyset$, repeat the algorithm for all pairs $(\mathcal{P}_{1i}, \mathcal{P}_{2j}), 1 \leq i, j \leq 2$.
Step 4. Pick $\pi_m \in \Pi_t$ that maximizes $|\mathcal{S}(\mathcal{P}_1, \pi)| + |\mathcal{S}(\mathcal{P}_2, \neg\pi)|$.
Step 5. Compute pairs $(Int(\mathcal{P}_1, \pi_m), Int(\mathcal{P}_2, \pi_m))$, $(Int(\mathcal{P}_1, \neg\pi_m), Int(\mathcal{P}_2, \neg\pi_m))$. For each pair, if both sets are non-empty, repeat the algorithm for the pair.

We briefly sketch the proof of the correctness of algorithm 2. As one can see from step 4, we use a greedy strategy to choose separating predicates, that is we select the one that can separate the largest number of polyhedra. The goal of step 5 is to exclude the subsets of $\bigcup \mathcal{P}_1$ and $\bigcup \mathcal{P}_2$ that the selected predicate π_m can separate. Indeed, if one of the sets $Int(\mathcal{P}_1, \pi_m)$ and $Int(\mathcal{P}_2, \pi_m)$ is empty, then either $\mathcal{P}_1$ or $\mathcal{P}_2$ lies entirely outside the half-space $\mathcal{H}(\pi_m)$. This means that the predicate π_m can separate a part of one set from the other, and we only need to continue with the remaining part.

One factor that determines the number of separating predicates is the subdivision in step 2. The way we subdivide the sets $\mathcal{P}_1$ with view of avoiding interference of the resulting subsets with $\mathcal{P}_2$ is as follows. We first try to split $\mathcal{P}_1$ into two subsets such that one contains all the polyhedra entirely outside $conv(\mathcal{P}_2)$. If this subset is empty, then we split $\mathcal{P}_1$ with respect to a hyperplane which is perpendicular to the longest side of $bbox(\mathcal{P}_1)$ and passes through its centroid. Another option for the normal of the splitting hyperplane is the line passing through the two most distant points.

We now briefly discuss the bound on the number of predicates algorithm 2 can produce. It is easy to see that the upper bound corresponds to the case where no splitting can produce separable subsets, which requires to consider all pairs of polyhedra. In other words, in this case, each time algorithm 2 finds a separating predicate π_m, $|(Int(\mathcal{P}_1, \pi_m)| = |\mathcal{P}_1| - 1$ and $|(Int(\mathcal{P}_2, \pi_m)| = |\mathcal{P}_2|$; similarly, $|(Int(\mathcal{P}_1, \neg\pi_m)| = |\mathcal{P}_1|$ and $|(Int(\mathcal{P}_2, \neg\pi_m)| = |\mathcal{P}_2| - 1$. This means that in each side of the half-space of π_m, two sets of polyhedra remain to be separated and the size of one set is decreased by 1. For example, if $|\mathcal{P}_1| = |\mathcal{P}_2| = K$, we can prove that algorithm 2 produces in worst case $2^{K+1} - 1$ predicates. It is important to note that this worst case typically happens when the polyhedra in each set are all disjoint and intertwine with those in the other set. However, in this context, reachable sets to be separated are often connected. Hence, in many practical cases, convex hulls or non-axis-aligned bounding boxes are relatively good approximations and the number of separating predicates produced by algorithm 2 is often much smaller than this bound. Finally, we can use the following lemma to achieve better efficiency.

Lemma 3. *If a set of predicates Π separates the boundaries of $\mathcal{P}_1$ and $\mathcal{P}_2$ then it separates $\mathcal{P}_1$ and $\mathcal{P}_2$.*

To prove the lemma, we remark that Π separates $\mathcal{P}_1$ and $\mathcal{P}_2$ iff any line segment between a point in $\mathcal{P}_1$ and another point in $\mathcal{P}_2$ intersects with the hyperplane of at least one predicates in Π. Hence if Π separates the boundaries of $\mathcal{P}_1$ and $\mathcal{P}_2$ then it separates $\mathcal{P}_1$ and $\mathcal{P}_2$ since any line segment connecting points in the interior of two disjoint sets must cross the boundaries of both sets. Using lemma 3 we can consider only some boundary layer of $\mathcal{P}_1$ and $\mathcal{P}_2$, which allows to obtain tighter convex approximations and thus reduces splitting.

5 Implementation and Experimentation

We presented foundations for automated verification of safety properties of hybrid systems by combining the ideas of counter-example guided predicate abstraction and polyhedral approximation of reachable sets of linear continuous dynamics. The presented counter-example analysis tool extends previous work on predicate abstraction of hybrid systems [3]. Our current prototype implementation of the predicate abstraction model checking and the counter-example analysis tool are both implemented in C++ using library functions of the hybrid systems reachability tool `d/dt` [6]. We implemented a translation procedure from Charon [2] source code to the predicate abstraction input language which is based on the `d/dt` input language. Our tool uses the polyhedral libraries CDD and QHull. We have implemented the global analysis algorithm, the local feasibility check, as well as the computation of separating predicates as part of the counter-example analysis tool.

5.1 Fischer's Mutual Exclusion

We first look at an example of mutual exclusion which uses time-based synchronization in a multi-process system. We want to implement a protocol that allows a shared resource to be used exclusively by at most one of two processes at any given time. The state machines for the two processes are shown in figure 2. The example is small enough to be used effectively for an illustration of our approach.

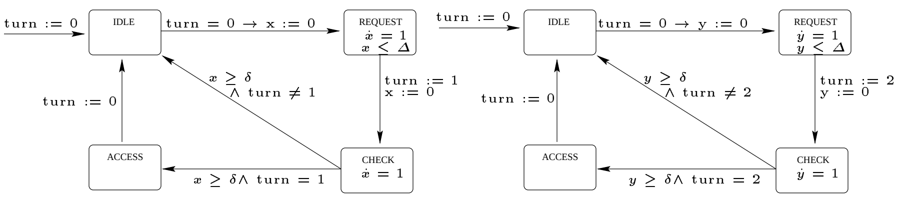

Fig. 2. The two processes for the mutual exclusion example

The possible execution traces depend on the two positive parameters Δ and δ. If the parameters are such that $\Delta \geq \delta$ is true, we can find a counter-example that proves the two processes may access the shared resource at the same time. On the other hand, if $\delta > \Delta$, then the system preserves mutual exclusive use of the shared resource. We use this example to illustrate the use of the local feasibility check of counter-examples for the case that $\delta > \Delta$. Consider the abstract system defined by the predicates used in the description of the 2-process Fischer's mutual exclusion protocol. These are: $x \geq \delta, y \geq \delta, x \leq \Delta, y \leq \Delta, \delta > \Delta, \Delta > 0, \delta > 0, x \geq 0$ and $y \geq 0$. The search in the abstract state-space finds a counter-example of length nine. The third abstract state a_3 in the counter-example has both processes in their respective `Request` locations, $\texttt{turn} = 0$, and $0 \leq x \leq \Delta, 0 \leq y \leq \Delta$. The following state a_4 can be reached by a discrete transition t_d, and the first process is now in its `Check` location, while $\texttt{turn} = 1$ and $0 \leq x \leq \Delta, 0 \leq y \leq \Delta$. The fifth abstract state a_5 can then be reached by a continuous transition t_c, so that the locations and the `turn` variable are unchanged, but now we have $x > \delta, 0 \leq y \leq \Delta$. Then, a_4 is locally infeasible, as shown by projection onto the variables x and y:

$$\texttt{Post}_{|(x,y)}(a_3, t_d, a_4) = \{(x, y)^T \in \mathbb{R}^2 \mid 0 \leq x \leq y \leq \Delta < \delta\} \text{ and}$$
$$\texttt{Pre}_{|(x,y)}(a_4, t_c, a_5) = \{(x, y)^T \in \mathbb{R}^2 \mid 0 \leq y < x \leq \Delta < \delta\};$$

hence, we know that $\texttt{Post}(a_3, t_d, a_4) \cap \texttt{Pre}(a_4, t_c, a_5) = \emptyset$. Following the strategy `LocalStrategy` we include the only one new predicate $x \leq y$ to the set of predicates. In the next iteration with this refinement of the abstract state-space, we obtain a symmetrical locally infeasible counter-example. The strategy `LocalStrategy` then suggests the symmetric predicate $y \leq x$. The subsequent reachability analysis finds 54 reachable abstract states in the refined abstract state-space, which all maintain the mutual exclusion property.

5.2 Thermostat

We have also successfully applied our counter-example guided predicate abstraction technique to verify a thermostat example. We present this case study to illustrate the global counter-example analysis algorithm as well as the procedure to separate two disjoint sets of polyhedra. The single hybrid automaton in this case study is shown in figure 3. It contains a timer `clock` and a temperature `Temp`. The initial location is `Heat` and the initial set is $\texttt{clock} = 0, 5 \leq \texttt{Temp} \leq 10$, and the bad set is $\texttt{Temp} < 4.5$ in any location. The model can be proven correctly by adding the predicate $\texttt{clock} \leq 0$ to the predicates that are already mentioned in the model as transition guards and location invariants. In this case, the abstraction uses ten predicates and finds 35 reachable abstract states.

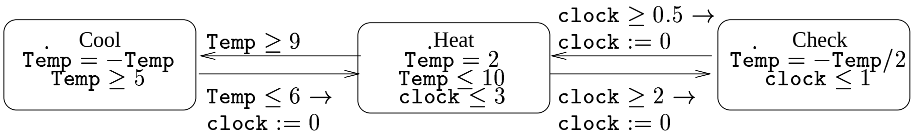

Fig. 3. The Thermostat example. We omit the differential equation $\dot{\texttt{clock}} = 1$ in all three locations.

For purposes of illustration, we start the verification using our counter-example guided predicate abstraction toolkit with the predicates mentioned in the model; that means, we are searching for a set of predicates that can be used to refine this initial set to be able to prove the safety of the given model. Note that picking the aforementioned predicate $\texttt{clock} \leq 0$ would suffice. In addition, to illustrate the global analysis algorithm and the separating routine only, we skip the local feasibility checking algorithm. The first iteration of our algorithm produces a spurious counter-example of length 7 after 11 abstract states have been discovered by the search of the abstract state-space. The separation routine suggests the following four linear predicate to refine the abstract state-space:

```
  0.979265*Temp +  0.202584*clock <=  9.34423
  0.872555*Temp +  0.488515*clock <=  8.16961
  0.428587*Temp +  0.9035  *clock <=  4.11184
-0.0680518*Temp +  0.997682*clock <= -0.439659
```

Please notice the last suggested predicate and its similarity to the predicate mentioned before. The model designer may have been able to use this suggested set of predicates to refine the abstract state space by adding the predicate $\texttt{clock} \leq 0$.

Following our example, after refining the predicates with the help of these four predicates, the system still finds a spurious counter-example, and suggest four more predicates. In a third round after discovering another spurious counter-example, the system generates eleven more predicates, one of which is `0.0139043*Temp + 0.999903*clock <= 0.152558`. The total set of 28 predicates is in the following iteration enough to prove the thermostat example safe. The search in the abstract state-space finds 358 reachable abstract states.

5.3 Coordinated Adaptive Cruise Control

We have also successfully applied our predicate abstraction technique to verify a model of the *Coordinated Adaptive Cruise Control* mode of a vehicle-to-vehicle coordination system. This case study is provided by the PATH project (see `http://www-path.eecs.berkeley.edu`). We first briefly describe the model omitting a more detailed discussion for the sake of brevity. The goal of this mode is to maintain the car at some desired speed v_d while avoiding collision with a car in front. Let x and v denote the position and velocity of the car. Let x_l, v_l and a_l denote respectively the position, velocity and acceleration of the car in front. Since we want to prove that no collision happens regardless of the behavior of the car in front, this car is treated as disturbance, more precisely, the derivative of its acceleration is modeled as uncertain input ranging in $[da_{lmin}, da_{lmax}]$.

The closed-loop system can be modeled as a hybrid automaton with 5 continuous variables and 8 locations.The invariants of the locations and the transition guards are specified by the operation regions and switching conditions of the controller together with the bounds on the speed and acceleration. In order to prove that the controller can guarantee that no collision between the cars can happen, we specify an unsafe set as $x_l - x \leq 0$ in all locations. To define initial predicates, in addition to the constraints of the invariants and guards, we use the predicate of the bad set allowing to distinguish safe and unsafe states and predicates representing the initial set. Assuming that the follower car is faster than the preceding car, and a too small initial separation of the two cars, the tool finds a counter-example that corresponds to a real trace in the concrete system. On the other hand, if the two cars start with a large enough initial separation, the combined verification approach enabled us to prove safety of the abstract system which implies safety of the concrete system.

References

1. R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.
2. R. Alur, T. Dang, J. Esposito, Y. Hur, F. Ivančić, V. Kumar, I. Lee, P. Mishra, G. Pappas, and O. Sokolsky. Hierarchical modeling and analysis of embedded systems. *Proceedings of the IEEE*, 91(1), January 2003.
3. R. Alur, T. Dang, and F. Ivančić. Reachability analysis of hybrid systems via predicate abstraction. In *Hybrid Systems: Computation and Control, Fifth International Workshop*, LNCS 2289. Springer-Verlag, 2002.
4. R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
5. R. Alur, A. Itai, R.P. Kurshan, and M. Yannakakis. Timing verification by successive approximation. *Information and Computation*, 118(1):142–157, 1995.

6. E. Asarin, O. Bournez, T. Dang, and O. Maler. Approximate reachability analysis of piecewise-linear dynamical systems. In *Hybrid Systems: Computation and Control, Third International Workshop*, LNCS 1790, pages 21–31. 2000.
7. T. Ball and S. Rajamani. Bebop: A symbolic model checker for boolean programs. In *SPIN 2000 Workshop on Model Checking of Software*, LNCS 1885. 2000.
8. S. Cameron. A comparison of two fast algorithms for computing the distance between convex polyhedra. *IEEE Transactions on Robotics and Automation*, 13(6):915–920, 1997.
9. A. Chutinan and B.K. Krogh. Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations. In *Hybrid Systems: Computation and Control, Second International Workshop*, LNCS 1569, pages 76–90. 1999.
10. E. Clarke, A. Fehnker, Z. Han, B. Krogh, O. Stursberg, and M. Theobald. Verification of hybrid systems based on counterexample-guided abstraction refinement. In *Tools and Algorithms for the Construction and Analysis of Systems*, 2003.
11. E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Computer Aided Verification*, pages 154–169, 2000.
12. E.M. Clarke and R.P. Kurshan. Computer-aided verification. *IEEE Spectrum*, 33(6):61–67, 1996.
13. J.C. Corbett, M.B. Dwyer, J. Hatcliff, S. Laubach, C.S. Pasareanu, Robby, and H. Zheng. Bandera: Extracting finite-state models from Java source code. In *Proceedings of 22nd International Conference on Software Engineering*. 2000.
14. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM Symposium on Principles of Programming Languages*, 1977.
15. G. Das and D. Joseph. The complexity of minimum convex nested polyhedra. In *Canadian Conference on Computational Geometry*, 1990.
16. C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In *Hybrid Systems III: Verification and Control*, LNCS 1066. Springer-Verlag, 1996.
17. D. Dobkin and D. Kirkpatrick. Determining the separation of preprocessed polyhedra - a unified approach. In *Proc. of ICALP'90*, pages 400–413, 1990.
18. H. Edelsbrunner and F.P. Preparata. Minimum polygon separation. *Information and Computation*, 77:218–232, 1987.
19. T.A. Henzinger, P. Ho, and H. Wong-Toi. HyTech: the next generation. In *Proceedings of the 16th IEEE Real-Time Systems Symposium*, pages 56–65, 1995.
20. G.J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.
21. G.J. Holzmann and M.H. Smith. Automating software feature verification. *Bell Labs Technical Journal*, 5(2):72–87, 2000.
22. K. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *Springer International Journal of Software Tools for Technology Transfer*, 1, 1997.
23. C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design Volume 6, Issue 1*, 1995.