Reachability Analysis of Hybrid Systems via Predicate Abstraction

Rajeev Alur, Thao Dang, and Franjo Ivančić

Department of Computer and Information Science
University of Pennsylvania
`http://www.seas.upenn.edu/hybrid/`

Abstract. Predicate abstraction has emerged to be a powerful technique for extracting finite-state models from infinite-state discrete programs. This paper presents algorithms and tools for reachability analysis of hybrid systems by combining the notion of predicate abstraction with recent techniques for approximating the set of reachable states of linear systems using polyhedra. Given a hybrid system and a set of user-defined boolean predicates, we consider the finite discrete quotient whose states correspond to all possible truth assignments to the input predicates. The tool performs an on-the-fly exploration of the abstract system. We demonstrate the feasibility of the proposed technique by analyzing a parametric timing-based mutual exclusion protocol and safety of a simple controller for vehicle coordination.

1 Introduction

Inspired by the success of model checking in hardware verification and protocol analysis [8,17], there has been increasing research on developing techniques for automated verification of hybrid (mixed discrete-continuous) models of embedded controllers [1,3,15]. The state-of-the-art computational tools for model checking of hybrid systems are of two kinds. Tools such as KRONOS [11], UPPAAL [20], and HYTECH [16] limit the continuous dynamics to simple abstractions such as rectangular inclusions (e.g. $\dot{x} \in [1, 2]$), and compute the set of reachable states exactly and effectively by symbolic manipulation of linear inequalities. On the other hand, emerging tools such as CHECKMATE [7], `d/dt` [5], and level-sets method [14,21], approximate the set of reachable states by polyhedra or ellipsoids [19] by optimization techniques. Even though these tools have been applied to interesting real-world examples after appropriate abstractions, scalability remains a challenge.

In the world of program analysis, predicate abstraction has emerged to be a powerful and popular technique for extracting finite-state models from complex, potentially infinite state, discrete systems [10,13]. A verifier based on this scheme requires three inputs, the (concrete) system to be analyzed, the property to be verified, and a finite set of boolean predicates over system variables to be used for abstraction. An abstract state is a valid combination of truth values to the boolean predicates, and thus, corresponds to a set of concrete states.

There is an abstract transition from an abstract state A to an abstract state B, if there is a concrete transition from some state corresponding to A to some state corresponding to B. The job of the verifier is to compute the abstract transitions, and to search in the abstract graph for a violation of the property. If the abstract system satisfies the property, then so does the concrete system. If a violation is found in the abstract system, then the resulting counter-example can be analyzed to test if it is a feasible execution of the concrete system. This approach, of course, does not solve the verification problem by itself. The success crucially depends on the ability to identify the "interesting" predicates, and on the ability of the verifier to compute abstract transitions efficiently. Nevertheless, it has led to opportunities to bridge the gap between code and models and to combine automated search with user's intuition about interesting predicates. Tools such as Bandera [9], SLAM [6], and Feaver [18] have successfully applied predicate abstraction for analysis of C or Java programs.

Inspired by these two trends, we develop algorithms for invariant verification of hybrid systems using discrete approximations based on predicate abstractions. Consider a hybrid automaton with n continuous variables and a set L of locations. Then the continuous state-space is $L \times \mathbb{R}^n$. For the sake of efficiency, we restrict our attention where all invariants, switching guards, and discrete updates of the hybrid automaton are specified by linear expressions, and the continuous dynamics is linear, possibly with bounded input. For the purpose of abstraction, the user supplies initial predicates $p_1 \dots p_k$, where each predicate is a polyhedral subset of $\mathbb{R}^n$. In the abstract program, the n continuous variables are replaced by k discrete boolean variables. A combination of values to these k boolean variables represents an abstract state, and the abstract state space is $L \times \mathbb{B}^k$. Our 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 an 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`. The basic strategy computes the polyhedral slices of states reachable from A at fixed times $r, 2r, 3r, \dots$ for a suitably chosen r, and then, takes the convex-hull of all these polyhedra to over-approximate the set of all states reachable from A. However, while tools such as CHECKMATE and `d/dt` are designed to compute a "good" approximation of the continuous successors of A, we are interested in testing if this set intersects with a new abstract state. Consequently, our implementation differs in many ways. For instance, it checks for nonempty intersection with other abstract states of each of the polyhedral slices, and omits steps involving approximations using orthogonal polyhedra and termination tests.

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 of hybrid systems. First, the expensive operation of computing continuous successors is applied only to abstract states, and not to intermediate polyhedra of unpredictable shapes and complexities. Second, we can prematurely terminate the computation of continuous successors whenever new abstract transitions are discovered. Finally, we can explore with different search strategies aimed at making progress in the abstract graph. For instance, our implementation always prefers computing discrete transitions over continuous ones. Our early experiments indicate that improvements in time and space requirements are significant compared to a tool such as `d/dt`.

We demonstrate the feasibility of our approach using two case studies. The first one involves verification of a parametric version of Fischer's protocol for timing-based mutual exclusion. The correctness of the protocol depends on two parameters δ and Δ. Traditional tools can analyze such problems very efficiently for fixed values of these parameters. Recently, there have been some results for parametric versions of the problem [4]. In our analysis, $b = (\delta > \Delta)$ is used as one of the initial predicates. The abstract search nicely reveals that the bad states are reachable precisely from those initial states in which the predicate b is false. The second example involves verification of a longitudinal controller for the leader car of a platoon from the IVHS projects [12]. Our concrete model consists of 4 continuous variables, linear dynamics with one bounded input, and 17 initial predicates. The verifier could establish absence of collisions without using any significant computational resources.

2 Hybrid Automata

2.1 Syntax

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 $\forall i \in \{1, \ldots, n+1\} : a_i \in \mathbb{R}$. Additionally, we denote the set of finite conjunctions of n-dimensional linear predicates by $\mathcal{C}_n$.

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

- *$\mathcal{X} \subseteq \mathbb{R}^n$ is a* **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.*
- *$X_0 \subseteq \mathcal{X}$ is the set of* **initial states**.
- *$I : L \to \mathcal{C}_n$ assigns to each location $l \in L$ a finite conjunction of linear predicates $I(l)$ defining the* **invariant** *conditions that constrain the value of the continuous part of the state while the discrete part is l. The hybrid automaton can only stay in location l as long as the continuous variable x satisfies $I(l)$, i.e. $\forall i \in I(l) : i(x) = 1$. We will write $\mathcal{I}_l$ for the invariant set of location l, that is the set of all points x satisfying all predicates in $I(l)$. In other words, $\mathcal{I}_l := \{x \in \mathcal{X} \mid \forall i \in I(l) : i(x) = 1\}$.*

- $f : L \rightarrow (\mathbb{R}^n \rightarrow \mathbb{R}^n)$ *assigns to each location* $l \in V$ *a* **continuous vector field** $f(l)$ *on* x*. While staying at location* l *the evolution of the continuous variable is governed by the differential equation* $\dot{x} = f(l)(x)$*.*
- $T : L \rightarrow 2^{\mathcal{C}_n \times L \times (\Sigma_n)^n}$ *is a function capturing discrete transition jumps between two discrete locations. A transition* $(g, l', r) \in T(l)$ *consists of an initial location* l*, a destination location* l'*, a set of* **guard** *constraints* g *and a* **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 will write* $\mathcal{G}_{ll'}$ *for the guard set of a transition* $(g, l', r) \in T(l)$ *which is the set of points satisfying all linear predicates of* g*, that is,* $\mathcal{G}_{ll'} := \{x \in \mathcal{X} \mid \forall e \in g : e(x) = 1\}$*.*

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. As we shall see later in Section 4.2, our reachability analysis can also be applied to hybrid systems having linear continuous dynamics with uncertain, bounded input of the form: $\dot{x} = A_l x + B_l u$.

2.2 Semantics

We will explain the semantics of a hybrid automaton by defining its underlying transition system. Let $\Phi_l(x, t) = e^{A_l t} x$ denote the flow of the system $\dot{x} = A_l x$.

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 flow, whereas $\rightarrow_D$ describes the transitions due to discrete jumps.

$$(l, x) \rightarrow_C (l, y) :\Leftrightarrow \exists t \in \mathbb{R}_{\geq 0} : \Phi_l(x, t) = y \wedge \forall t' \in [0, t] : \Phi_l(x, t') \in \mathcal{I}_l. \quad (1)$$

$$(l, x) \rightarrow_D (l', y) :\Leftrightarrow \exists (g, l', r) \in T(l) : x \in \mathcal{G}_{ll'} \wedge y = r(x). \quad (2)$$

We introduce now some basic reachability notation. We define the set of *continuous successors* of a set of states (l, P) where $l \in L$ and $P \subseteq \mathcal{X}$, denoted by $\texttt{Post}_C(l, P)$ as: $\texttt{Post}_C(l, P) := \{(l, y) \in X \mid \exists x \in P\ (l, x) \rightarrow_C (l, y)\}$. Similarly, we define the set of *discrete successors* of (l, P), denoted by $\texttt{Post}_D(l, P)$, as: $\texttt{Post}_D(l, P) := \{(l', y) \in X \mid \exists x \in P\ (l, x) \rightarrow_D (l', y)\}$.

2.3 Discrete Abstraction

Let us now define a discrete abstraction of the hybrid system H with respect to a given k-dimensional vector of linear predicates $\Pi = (\pi_1, \pi_2, \ldots, \pi_k)$. We can partition the continuous state space $\mathcal{X}$ 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 will 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 a 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 $Q := L \times \mathbb{B}^k$.*

Definition 3 (Concretization function). *We define a* **concretization function** $C_\Pi : \mathbb{B}^k \to 2^{\mathcal{X}}$ *for a vector of linear predicates $\Pi = (\pi_1, \dots, \pi_k) \in (\mathcal{L}_n)^k$ as follows: $C_\Pi(\boldsymbol{b}) := \{x \in \mathcal{X} \mid \forall i \in \{1, \dots, k\} : \pi_i(x) = b_i\}$. Denote a vector $\boldsymbol{b} \in \mathbb{B}^k$ as* **consistent** *with respect to a vector of linear predicates $\Pi \in (\mathcal{L}_n)^k$, if $C_\Pi(\boldsymbol{b}) \neq \emptyset$. We say that an abstract state $(l, \boldsymbol{b}) \in Q$ is* **consistent** *with respect to a vector of linear predicates Π, if $\boldsymbol{b}$ is consistent with respect to Π.*

If all predicates in Π are indeed linear predicates, it can be shown that $C_\Pi(\boldsymbol{b})$ is a convex polyhedron for any $\boldsymbol{b} \in \mathbb{B}^k$.

Definition 4 (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 $H_\Pi = (Q, \xrightarrow{\Pi}, Q_0)$ where*

- *the abstract transition relation $\xrightarrow{\Pi} \subseteq Q \times Q$ is defined as follows:*

$$(l, \boldsymbol{b}) \xrightarrow{\Pi} (l', \boldsymbol{b'}) :\Leftrightarrow \exists x \in C_\Pi(\boldsymbol{b}), y \in C_\Pi(\boldsymbol{b'}) : (l, x) \to (l', y);$$

- *the set of initial states is $Q_0 = \{(l, \boldsymbol{b}) \in Q \mid \exists x \in C_\Pi(\boldsymbol{b}) : (l, x) \in X_0\}$.*

To be able to distinguish transitions in the abstract state space due to a discrete jump in the concrete state space from those transitions that are due to continuous flows, we introduce the following two relations $\xrightarrow{\Pi}_D, \xrightarrow{\Pi}_C \subseteq Q \times Q$:

$$\begin{aligned}(l, \boldsymbol{b}) \xrightarrow{\Pi}_D (l', \boldsymbol{b'}) :\Leftrightarrow\ & \exists (g, l', r) \in T(l), x \in C_\Pi(\boldsymbol{b}) \cap \mathcal{G}_{ll'} : \\ & (l, x) \to_D (l', r(x)) \wedge r(x) \in C_\Pi(\boldsymbol{b'}), \qquad (3)\\ (l, \boldsymbol{b}) \xrightarrow{\Pi}_C (l, \boldsymbol{b'}) :\Leftrightarrow\ & \exists x \in C_\Pi(\boldsymbol{b}), t \in \mathbb{R}_{\geq 0} : \Phi_l(x, t) \in C_\Pi(\boldsymbol{b'}) \wedge \\ & \forall t' \in [0, t] : \Phi_l(x, t') \in \mathcal{I}_l. \qquad (4)\end{aligned}$$

We can now define the successors of an abstract state $(l, \boldsymbol{b})$ by discrete jumps and by continuous flows, denoted respectively by $\mathtt{Post}_D^\Pi(l, \boldsymbol{b})$ and $\mathtt{Post}_C^\Pi(l, \boldsymbol{b})$, as: $\mathtt{Post}_D^\Pi(l, \boldsymbol{b}) := \{(l', \boldsymbol{b'}) \in Q \mid (l, \boldsymbol{b}) \xrightarrow{\Pi}_D (l', \boldsymbol{b'})\}$, and $\mathtt{Post}_C^\Pi(l, \boldsymbol{b}) := \{(l, \boldsymbol{b'}) \in Q \mid (l, \boldsymbol{b}) \xrightarrow{\Pi}_C (l, \boldsymbol{b'})\}$.

3 Reachability Analysis

For the reachability analysis, it could be assumed that all guards and invariants of a hybrid automaton H are included in the vector of linear predicates Π which will be used for our abstract state space reachability exploration. On the other hand, one may reduce the state space of the abstract system by not including all guards and invariants, but rather only include linear predicates that are important for the verification of the given property.

3.1 Computing Discrete Successors of the Abstract System

Given an abstract state $(l, \boldsymbol{b}) \in Q$ and a particular transition $(g, l', r) \in T(l)$ we want to compute all abstract states that are reachable. A transition $(g, l', r) \in T(l)$ is *enabled* in an abstract state $(l, \boldsymbol{b})$ with respect to Π, if $C_\Pi(\boldsymbol{b}) \cap \mathcal{G}_{ll'} \neq \emptyset$.

We define a tri-valued logic using the symbols $\mathcal{T} := \{0, 1, *\}$. 0 denotes that a particular linear predicate is always false for a given abstract state, 1 that it is always true, whereas $*$ denotes the case that a linear predicate is true for part of the abstract state, and false for the rest. We can define a function $t : \mathbb{B}^k \times (\mathcal{L}_n)^k \times \mathcal{L}_n \to \mathcal{T}$ formally as:

$$t(\boldsymbol{b}, \Pi, e) = \begin{cases} 1\,, & \text{if } C_\Pi(\boldsymbol{b}) \neq \emptyset \wedge \forall x \in C_\Pi(\boldsymbol{b}) : \ e(x) = 1; \\ 0\,, & \text{if } C_\Pi(\boldsymbol{b}) \neq \emptyset \wedge \forall x \in C_\Pi(\boldsymbol{b}) : \ e(x) = 0; \\ *\,, & \text{otherwise.} \end{cases}$$

As will be described shortly, we can use this tri-valued logic to reduce the size of the set of feasible abstract successor states. For later use, let us define the number of positions in a k-dimensional vector $\boldsymbol{t} \in \mathcal{T}^k$ with element $*$ as $||\boldsymbol{t}||_*$.

Given a particular transition $(g, l', r) \in T(l)$ and a given linear predicate $e : \mathbb{R}^n \to \mathbb{B}$, we need to compute the boolean value of a linear predicate e after the reset r, which is $e(r(x))$. It can be seen that $e \circ r : \mathbb{R}^n \to \mathbb{B}$ is another linear predicate. It should be noted that given e and r we can easily compute $e \circ r$. If we generalize this for a vector of predicates $\Pi = (\pi_1, \ldots, \pi_k)^T \in (\mathcal{L}_n)^k$ by $\Pi \circ r := (\pi_1 \circ r, \ldots, \pi_k \circ r)^T$, the following lemma immediately follows.

Lemma 1. *Given a k-dimensional vector $\boldsymbol{b} \in \mathbb{B}^k$, a vector of n-dimensional linear predicates $\Pi \in (\mathcal{L}_n)^k$, and a reset mapping $r \in (\Sigma_n)^n$, we have:*

$$x \in C_{\Pi \circ r}(\boldsymbol{b}) \Leftrightarrow r(x) \in C_\Pi(\boldsymbol{b}).$$

We can now compute the possible successor states of an enabled transition $(g, l', r) \in T(l)$ from a consistent abstract state $(l, \boldsymbol{b})$ with respect to a vector of linear predicates $\Pi = (\pi_1, \ldots, \pi_k)$ as $(l', \boldsymbol{b'})$, where $\boldsymbol{b'} \in \mathcal{T}^n$ and each component b'_i is given by: $b'_i = t(\boldsymbol{b}, \Pi, \pi_i \circ r)$. If $b'_i = 1$, then we know that the corresponding linear predicate $\pi_i \circ r$ is true for all points $x \in C_\Pi(\boldsymbol{b})$. This means that all states in $C_\Pi(\boldsymbol{b})$ after the reset r will make π_i true. Similarly, if $b'_i = 0$ we know that the linear predicate will always be false. Otherwise, if $b'_i = *$, then there exist concrete continuous states in $C_\Pi(\boldsymbol{b})$ that after the reset r force π_i to become true, as well as other concrete continuous states that make π_i to become false. Hence, the tri-valued vector $\boldsymbol{b'} \in \mathcal{T}^n$ represents $2^{||\boldsymbol{b'}||_*}$ many possibilities, which combined with location l' make up at most $2^{||\boldsymbol{b'}||_*}$ many abstract states. We define $c : \mathcal{T}^k \to 2^{\mathbb{B}^k}$ as: $c(\boldsymbol{t}) := \{\boldsymbol{b} \in \mathbb{B}^k \mid \forall i \in \{1, \ldots, k\} : t_i \neq * \Rightarrow t_i = b_i\}$.

An abstract state $(l', \boldsymbol{e}) \in Q$ is a discrete successor of $(l, \boldsymbol{b})$, if $\boldsymbol{e} \in c(\boldsymbol{b'})$ and $C_{\Pi \circ r}(\boldsymbol{e})$ intersects with $C_\Pi(\boldsymbol{b})$ and the guard of the corresponding transition, which is formulated in the following theorem.

Theorem 1. *Given an abstract state $(l, \boldsymbol{b}) \in Q$ with respect to a k-dimensional vector of n-dimensional linear predicates Π, a transition $(g, l', r) \in T(l)$ and the corresponding guard set $\mathcal{G}_{ll'}$, we have $\forall \boldsymbol{v} \in \mathbb{B}^k$:*

$$C_{\Pi \circ r}(\boldsymbol{v}) \cap C_{\Pi}(\boldsymbol{b}) \cap \mathcal{G}_{ll'} \neq \emptyset \Leftrightarrow (l, \boldsymbol{b}) \xrightarrow{\Pi}_D (l', \boldsymbol{v}).$$

Proof: If $C_{\Pi \circ r}(\boldsymbol{v}) \cap C_{\Pi}(\boldsymbol{b}) \cap \mathcal{G}_{ll'}$ is not empty, we can pick a point $x \in C_{\Pi \circ r}(\boldsymbol{v}) \cap C_{\Pi}(\boldsymbol{b}) \cap \mathcal{G}_{ll'}$. As $x \in \mathcal{G}_{ll'}$, we found a discrete transition in the concrete state space $(l, x) \rightarrow_D (l', r(x))$. We know that $x \in C_{\Pi}(\boldsymbol{b})$. Additionally we know that $x \in C_{\Pi \circ r}(\boldsymbol{v})$ and by using Lemma 1 we have $r(x) \in C_{\Pi}(\boldsymbol{v})$. Hence, this corresponds to a transition in the abstract state space $(l, \boldsymbol{b}) \xrightarrow{\Pi}_D (l', \boldsymbol{v})$.

If, on the other hand, we have $(l, \boldsymbol{b}) \xrightarrow{\Pi}_D (l', \boldsymbol{v})$ for a discrete transition $(g, l', r) \in T(l)$, we must have: $\exists x \in C_{\Pi}(\boldsymbol{b}) : x \in \mathcal{G}_{ll'} \wedge r(x) \in C_{\Pi}(\boldsymbol{v})$. Using Lemma 1, this means that $\exists x \in C_{\Pi}(\boldsymbol{b}) : x \in \mathcal{G}_{ll'} \wedge x \in C_{\Pi \circ r}(\boldsymbol{v})$. Hence we found that $C_{\Pi \circ r}(\boldsymbol{v}) \cap C_{\Pi}(\boldsymbol{b}) \cap \mathcal{G}_{ll'} \neq \emptyset$. ■

Note, that if we assume that all the linear predicates of the guard $g \in \mathcal{C}_n$ are part of the k-dimensional vector of linear predicates Π, then we can skip the additional check, whether $C_{\Pi \circ r}(\boldsymbol{v}) \cap C_{\Pi}(\boldsymbol{b})$ intersects with the guard set $\mathcal{G}_{ll'}$. In addition, we can restrict the search for non-empty intersections to $\boldsymbol{v} \in c(\boldsymbol{b'})$ instead of the full space $\mathbb{B}^k$ due to the aforementioned observations.

3.2 Computing Continuous Successors of the Abstract System

Our procedure for computing continuous successors of the abstract system A_{Π} is based on the following observation. By definition, the abstract states $(l, \boldsymbol{b'})$ is reachable from $(l, \boldsymbol{b})$ if the following condition is satisfied

$$PostC(l, C_{\boldsymbol{\Pi}}(\boldsymbol{b})) \cap C_{\boldsymbol{\Pi}}(\boldsymbol{b'}) \neq \emptyset, \tag{5}$$

where $\texttt{Post}_c$ is the successor operator of the concrete system H. Intuitively, the above condition means that while staying at location l the concrete system admits at least one trajectory from a point $x \in C_{\boldsymbol{\Pi}}(\boldsymbol{b})$ to a point $y \in C_{\boldsymbol{\Pi}}(\boldsymbol{b'})$. Therefore, the set of continuous successors of an abstract state $(l, \boldsymbol{b})$ can be written as follows: $\texttt{Post}_C^{\Pi}(l, \boldsymbol{b}) = \{(l, \boldsymbol{b'}) \mid \texttt{Post}_c(l, C_{\boldsymbol{\Pi}}(\boldsymbol{b})) \cap C_{\boldsymbol{\Pi}}(\boldsymbol{b'}) \neq \emptyset\}$. The test of the condition (5) requires the computation of continuous successors of the concrete system, and for this purpose we will make use of a modified version of the reachability algorithm implemented in the verification tool `d/dt` [5]. For a clear understanding, let us first recap this algorithm.

The approach used by `d/dt` works directly on the continuous state space of the hybrid system and uses orthogonal polyhedra to represent reachable sets, which allows to perform all operations, such as boolean operations and equivalence checking, required by the verification task. Basically, the computation of reachable sets is done on a step-by-step basis, that is each iteration k computes an over-approximation of the reachable set for the time interval $[kr, (k+1)r]$ where r is the time step. Suppose P is the initial convex polyhedron. The set P_r of successors at time r of P is the convex hull of the successors at time r

of its vertices. To over-approximate the successors during the interval $[0, r]$, the convex hull $C = conv(P \cup P_r)$ is computed and then enlarged by an appropriate amount. Finally, the enlarged convex hull is over-approximated by an orthogonal polyhedron. To deal with invariant conditions that constrain the continuous evolution at each location, the algorithm intersects P_r with the invariant set and starts the next iteration from the resulting polyhedron.

It is worth emphasizing that the goal of the orthogonal approximation step in the reachability algorithm of `d/dt` is to represent the reachable set after successive iterations as a unique orthogonal polyhedron, which facilitates termination checking and the computation of discrete successors. However, in our predicate abstraction approach, to compute continuous successors of the abstract system we will exclude the orthogonal approximation step for the following reasons. First, checking condition (5) does not require accumulating concrete continuous successors. Moreover, although operations on orthogonal polyhedra can be done in any dimension, they become expensive as the dimension grows. This simplification allows us to reduce computation cost in the continuous phase and thus be able to perform different search strategies so that the violation of the property can be detected as fast as possible. In the sequel, for simplicity, we will use an informal notation $\mathtt{APost}_C^{\Pi}(l, P, [0, r])$ to denote the above described computation of an over-approximation of concrete continuous successors of (l, P) during the time interval $[0, r]$ and the outcome of $\mathtt{APost}_C^{\Pi}$ is indeed the enlarged convex hull mentioned earlier. The algorithm for over-approximating continuous successors of the abstract system is given below. It terminates if the reachable set of the current iteration is included in that of the precedent iteration. This termination condition is easy to check but obviously not sufficient, and hence in some cases the algorithm is not guaranteed to terminate. An illustration of Algorithm 1 is shown in Figure 1.

Algorithm 1 Over-Approximating the Abstract Continuous-Successors of $(l, \boldsymbol{b})$

```
$R_c \leftarrow \emptyset$;  $P^0 \leftarrow C_{\boldsymbol{\Pi}}(\boldsymbol{b})$;  $k \leftarrow 0$ ;
repeat
  $P^{k+1} \leftarrow \mathtt{APost}_C^{\Pi}(l, P^k, [0, r])$;
  for all $(l, \boldsymbol{b}') \in Q \setminus R_c$ do
    $P' \leftarrow C_{\boldsymbol{\Pi}}(\boldsymbol{b}')$;
    if $P^{k+1} \cap P' \neq \emptyset$ then
      $R_c := R_c \cup (l, \boldsymbol{b}')$ ;
  $k \leftarrow k + 1$ ;
until $P^{k+1} \subseteq P^k$
return $R_c$ ;
```

In each iteration k, to avoid testing all unvisited abstract states $(l, \boldsymbol{b}')$, we will use a similar idea to the one described in the computation of discrete successors. We can determine the tri-valued result of the intersection of the time slice P^k with the half-space corresponding to each predicate in Π, allowing us to eliminate the abstract states which do not intersect with P^k.

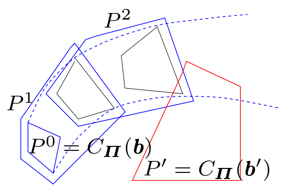
P^2
P^1
$P^0 = C_{\boldsymbol{\Pi}}(\boldsymbol{b})$
$P' = C_{\boldsymbol{\Pi}}(\boldsymbol{b}')$

Fig. 1. Illustration of the computation of continuous successors. After two iterations new abstract state $(l, \boldsymbol{b}')$ is reachable.

3.3 Search Strategy

The search in the abstract state space can be performed in a variety of ways. Our goal is to make the discovery of counter-examples in the abstract state space given a reachability property as fast as possible. In the case that the property is true we need to search the entire reachable abstract sub-space.

We perform a DFS, which usually does not find a shortest counter-example possible. On the other hand, it only stores the current trace of abstract states from an initial abstract state on a stack. In case we find an abstract state that violates the property, the stack contents represent the counter-example. This is generally much more memory efficient than BFS.

We give a priority to computing discrete successors rather than continuous successors. This decision is based on the fact that computing a discrete successor is generally much faster than computing a continuous one.

During the computation of continuous successors we abort or interrupt the computation when a new abstract state is found. Not running the fixpoint computation of continuous successors to completion may result in a substantial speed-up when discovering a counter-example, if one exists. A simplified sketch of the algorithm is given as Algorithm 2.

Algorithm 2 Abstract State Space Reachability Analysis via DFS

```
stack ← new Stack () ;
push initial state onto stack ; {for simplicity assume, there is only one such state}
repeat
  if stack.top().violatesProperty() then
    return stack ;
  if $\texttt{Post}_D^{\Pi}$(stack.top()) $\neq \emptyset$ then
    push one state in $\texttt{Post}_D^{\Pi}$(stack.top()) onto the stack ;
  else if $\texttt{Post}_C^{\Pi}$(stack.top()) $\neq \emptyset$ then
    push one state in $\texttt{Post}_C^{\Pi}$(stack.top()) onto the stack ;
  else
    stack.pop() ;
until stack.isEmpty()
return "Property is guaranteed!" ;
```

Using the aforementioned approach we can prove the following theorem which states the soundness of Algorithm 2.

Theorem 2. *If Algorithm 2 terminates and reports that the abstract system is safe, then the corresponding concrete system is also safe.*

We additionally include an optimization technique in the search strategy. Consider a real 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 in the equivalent counter-example. This is due to the so-called semi-group property of hybrid systems, namely: $(l, x) \rightarrow_C (l, x') \wedge (l, x') \rightarrow_C (l, x'') \Rightarrow (l, x) \rightarrow_C (l, x'')$. We are hence searching only for counter-examples in the abstract system that do not have two consecutive transitions due to continuous flow. By enforcing this additional constraint we eliminate some fake counter-examples that could have been found otherwise in the abstract transition system. The fake counter-examples that are eliminated are due to the fact that $(l, \boldsymbol{b}) \overset{\Pi}{\rightarrow}_C (l, \boldsymbol{b'})$ and $(l, \boldsymbol{b'}) \overset{\Pi}{\rightarrow}_C (l, \boldsymbol{b''})$ does *not* imply that $(l, \boldsymbol{b}) \overset{\Pi}{\rightarrow}_C (l, \boldsymbol{b''})$. Hence, we are in fact not computing the whole relation $\overset{\Pi}{\rightarrow}_C$ as it was defined above, but only a part of it without compromising the conservativeness of our approach. We illustrate this optimization technique in Figure 2.

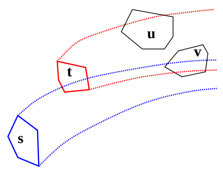

Fig. 2. An optimization technique in the search strategy: If the abstract state t can only be reached by continuous transitions, we will not explore its continuous successors by the same continuous dynamics. Hence, in the above example the abstract state u will not be regarded as reachable. On the other hand, v will be reached by continuous flow from s.

Also, in order to force termination of the continuous search routine, we can limit the number of iterations k to some value $K_{\max}$. This way we can bound the computation of $\mathtt{Post}_C^{\Pi}(l, \boldsymbol{b})$ for an abstract state $(l, \boldsymbol{b})$.

4 Implementation and Experimentation

4.1 Mutual Exclusion with Time-Based Synchronization

We will first look at an example of mutual exclusion which uses time-based synchronization in a multi-process system. The state machines for the two processes are shown in Figure 3. We will use this example for two reasons. The first reason is that it is small enough to be used effectively for an illustration of our approach. A second reason is that it is similar to examples that have been used as case-studies in other verification tools as well. Tools like KRONOS [11] and UPPAAL[20] for example have solved this example for specific values for the positive parameters δ and $\varDelta$. We want to solve the parametric version of the problem, that is using parameters without specifying values for δ and $\varDelta$, which has not been done previously.

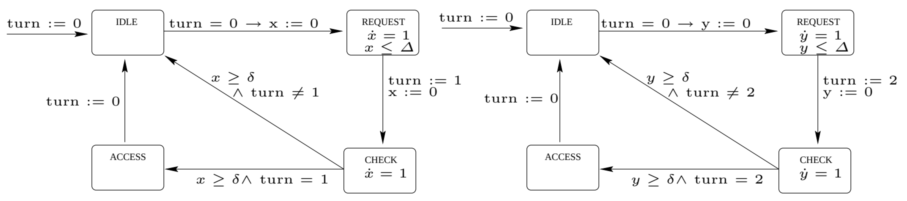

Fig. 3. The two processes for the mutual exclusion example

The possible execution traces depend on the two positive parameters $\varDelta$ and δ. If the parameters are such that $\varDelta \geq \delta$ is true, we can find a counter-example that proves the two processes may access the shared resource at the same time. The trace of abstract states that represents a valid counter-example in the original system is given in Figure 4.

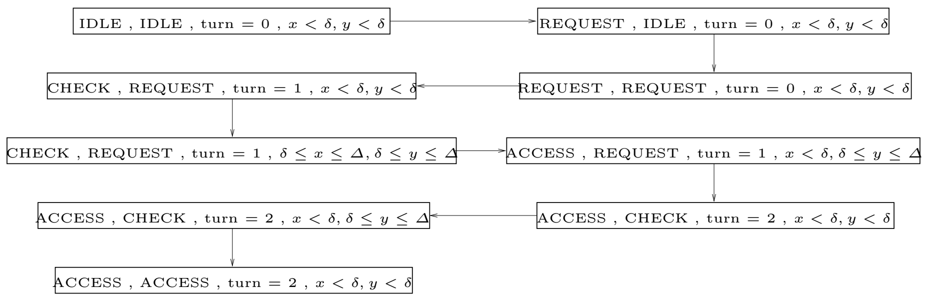
IDLE , IDLE , turn = 0 , x < δ, y < δ
REQUEST , IDLE , turn = 0 , x < δ, y < δ
REQUEST , REQUEST , turn = 0 , x < δ, y < δ
CHECK , REQUEST , turn = 1 , x < δ, y < δ
CHECK , REQUEST , turn = 1 , δ ≤ x ≤ Δ, δ ≤ y ≤ Δ
ACCESS , REQUEST , turn = 1 , x < δ, δ ≤ y ≤ Δ
ACCESS , CHECK , turn = 2 , x < δ, y < δ
ACCESS , CHECK , turn = 2 , x < δ, δ ≤ y ≤ Δ
ACCESS , ACCESS , turn = 2 , x < δ, y < δ

Fig. 4. A counter-example trace for the mutual exclusion problem for the parameter setting $\varDelta \geq \delta$. The predicates that were used to find this counter-example are the predicates given by the guards and invariants of the composed hybrid system. These are: $x \geq \delta, y \geq \delta, x \leq \varDelta$ and $y \leq \varDelta$. The states do not show the constantly true linear predicates over the parameters $\varDelta \geq \delta$, $\varDelta > 0$ and $\delta > 0$.

On the other hand, if $\delta > \Delta$, then the system preserves mutual exclusive use of the shared resource. In order to prove this using our predicate abstraction algorithm, we needed to include the intuitive predicates $x \geq y$ and $y \geq x$ in addition to the predicates already mentioned in the caption of Figure 4. The reachability analysis finds 54 reachable abstract states, which all maintain the mutual exclusion property. The computation of this result takes a few seconds, and the memory requirements are negligible.

4.2 Vehicle Coordination

We have also successfully applied our predicate abstraction technique to verify a longitudinal controller for the leader car of a platoon moving in an Intelligent Vehicle Highway System (IVHS). Let us first briefly describe this system. Details on the design process can be found in [12]. In the leader mode all the vehicles inside a platoon follow the leader. We consider a platoon i and its preceding platoon $(i-1)$. Let v_i and a_i denote respectively the velocity and acceleration of platoon i, and d_i is its distance to platoon $(i-1)$. The most important task of a controller for the leader car of each platoon i is to maintain the distance d_i equal to a safety distance $D_i = \lambda_a a_i + \lambda_v v_i + \lambda_p$ (in the nominal operation $\lambda_a = 0$, $\lambda_v = 1sec$, and $\lambda_p = 10m$). Other tasks the controller should perform are to track an optimal velocity and trajectories for certain maneuvers. The dynamics of the system are as follows: $\dot{d}_i = v_{i-1} - v_i$, $\dot{v}_{i-1} = a_{i-1}$, $\dot{v}_i = a_i$, and $\dot{a}_i = u$, where u is the control. Without going into details, the controller for the leader car of platoon i proposed in [12] consists of 4 control laws u which are used in different regions of the state space. These regions are defined based on the values of the relative velocity $v_i^e = 100(v_{i-1} - v_i)/v_i$ and the error between the actual and the safe inter-platoon distances $e_i = d_i - D_i$. When the system changes from one region to another, the control law should change accordingly. The property we want to verify is that a collision between platoons never happens, that is, $d_i \geq 0$. Here, we focus only on two regions which are critical from a safety point of view: "track optimal velocity" ($v_i^e < 0$ and $e_i > 0$) and "track velocity of previous car" ($v_i^e < 0$ and $e_i < 0$). The respective control laws u_1 and u_2 are as follows:

$$u_1 = -d_i - 3v_{i-1} + (-3 - \lambda_v)v_i + -3a_i, \tag{6}$$

$$u_2 = 0.125d_i + 0.75v_{i-1} + (-0.75 + -0.125\lambda_v)v_i - 1.5a_i. \tag{7}$$

Note that these regions correspond to situations where the platoon in front moves slower and, moreover, the second region is particularly safety critical because the inter-platoon distance is smaller than desired.

We model this system by a hybrid automaton with 4 continuous variables (d_i, v_{i-1}, v_i, a_i) and two locations corresponding to the two regions. The continuous dynamics of each location is linear as specified above, with u specified by (6) and (7). To prove that the controller of the leader car of platoon i can guarantee that no collision happens regardless of the behavior of platoon $(i-1)$, a_{i-1} is treated as *uncertain input* with values in the interval $[a_{min}, a_{max}]$ where a_{min} and a_{max}

are the maximal deceleration and acceleration. The invariants of the locations are defined by the constraints on e_i and v_i^e and the bounds on the velocity and acceleration. The guards of the transitions between locations are the boundary of the invariants, i.e. $e_i = 0$ (for computational reasons the guards are "thickened", more precisely, the transitions are enabled when e_i is in some small interval $[-\varepsilon, \varepsilon]$). The bad set is specified as $d_i < 0$. To construct the discrete abstract system, in addition to the predicates of the invariants and guards we use two predicates $d_i \leq 0$ and $d_i \geq 2$ which allow to separate safe and unsafe states, and the total number of initial predicates are 17. For the initial set specified as $5 \leq d_i \leq 1000 \ \wedge \ 5 \leq v_{i-1} \leq 15 \ \wedge \ 18 \leq v_i \leq 30$, the tool found 16 reachable abstract states and reported that the system is safe. The computation took 4 minutes on a Pentium 2. For *individual continuous modes* this property has been proven in [22] using optimal control techniques.

5 Conclusions

We have presented foundations and an initial prototype implementation for automated verification of safety properties of hybrid systems by combining ideas from predicate abstraction and polyhedral approximation of reachable sets of linear continuous systems. We are excited about the promise of this approach, but additional research and experiments are needed to fully understand the scope of the method. Two research directions are of immediate importance. First, the number of abstract states grows exponentially with the number of linear predicates used for abstraction. We have presented a couple of heuristics that avoid examination of all abstract states while computing the successors of a given abstract state. However, more sophisticated methods that exploit the structure of abstract states are needed. Also, heuristics for guiding the search can be useful to avoid the expensive computation of successors. Second, we have solely relied on the intuition of the designer to obtain the initial set of predicates. If the predicate abstraction verification algorithm returns a counter-example, we cannot be sure that the found counter-example corresponds to a real counter-example in the concrete system. It may be a fake counter-example instead, due to missing predicates important for the verification of the property at hand. We are working on a second module that will check the validity of a counter-example, and analyze it to introduce additional predicates, if needed. We are also incorporating the verifier with the tool Charon, a modeling and analysis environment for hierarchical hybrid systems [2].

Acknowledgments. This research was supported in part by DARPA ITO Mobies award F33615-00-C-1707 and by NSF ITR award. We thank the members of Penn's hybrid systems group for discussions and feedback.

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, R. Fierro, Y. Hur, F. Ivančić, V. Kumar, I. Lee, P. Mishra, G. Pappas, and O. Sokolsky. Hierarchical hybrid modeling of embedded systems. In *Embedded Software, First Intern. Workshop*, LNCS 2211. 2001.
3. R. Alur, T. Henzinger, G. Lafferriere, and G. Pappas. Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, 2000.
4. A. Annichini, E. Asarin, and A. Bouajjani. Symbolic techniques for parametric reasoning about counter and clock systems. In *Comp. Aided Verification*, 2000.
5. 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.
6. 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.
7. 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.
8. E.M. Clarke and R.P. Kurshan. Computer-aided verification. *IEEE Spectrum*, 33(6):61–67, 1996.
9. 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 Intern. Conf. on Software Engineering*. 2000.
10. S. Das, D. Dill, and S. Park. Experience with predicate abstraction. In *Computer Aided Verification, 11th International Conference*, LNCS 1633, 1999.
11. C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In *Hybrid Systems III: Verification and Control*, LNCS 1066, 1996.
12. D. Godbole and J. Lygeros. Longitudinal control of a lead card of a platoon. *IEEE Transactions on Vehicular Technology*, 43(4):1125–1135, 1994.
13. S. Graf and H. Saidi. Construction of abstract state graphs with PVS. In *Proc. 9th Conference on Computer Aided Verification*, volume 1254, 1997.
14. M. Greenstreet and I. Mitchell. Reachability analysis using polygonal projections. In *Hybrid Systems: Computation and Control, Second Intern. Workshop*, LNCS 1569. 1999.
15. N. Halbwachs, Y. Proy, and P. Raymond. Verification of linear hybrid systems by means of convex approximations. In *Intern. Symposium on Static Analysis*, LNCS 864. 1994.
16. 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.
17. G.J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.
18. G.J. Holzmann and M.H. Smith. Automating software feature verification. *Bell Labs Technical Journal*, 5(2):72–87, 2000.
19. A. Kurzhanski and P. Varaiya. Ellipsoidal techniques for reachability analysis. In *Hybrid Systems:Computation and Control, Third Intern. Workshop*, LNCS 1790. 2000.
20. K. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *Springer International Journal of Software Tools for Technology Transfer*, 1, 1997.
21. I. Mitchell and C. Tomlin. Level set methods for computation in hybrid systems. In *Hybrid Systems:Computation and Control, Third Intern. Workshop*, volume LNCS 1790. 2000.
22. A. Puri and P. Varaiya. Driving safely in smart cars. Technical Report UBC-ITS-PRR-95-24, California PATH, University of California in Berkeley, July 1995.