Progress on Reachability Analysis of Hybrid Systems using Predicate Abstraction*

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. To compute the transitions out of an abstract state, the tool needs to compute the set of discrete and continuous successors, and find out all the abstract states that this set intersects with. The complexity of this computation grows exponentially with the number of abstraction predicates. In this paper we present various optimizations that are aimed at speeding up the search in the abstract state-space, and demonstrate their benefits via case studies. We also discuss the completeness of the predicate abstraction technique for proving safety of hybrid systems.

1 Introduction

Automated verification of hybrid systems offers the promise of revealing subtle errors in high-level models of embedded controllers [1, 6, 9, 14, 17, 19]. Verification tools compute symbolic representations of the set of reachable states of a model. Dealing with continuous dynamics is a major computational challenge. Contemporary tools for verification of hybrid systems, such as CheckMate [9] and `d/dt` [6], approximate the set of reachable states by polyhedra. Recently, we have shown that effectiveness of these techniques can be enhanced using predicate abstraction [4], a powerful technique for extracting finite-state models from complex, potentially infinite state, discrete systems (see, for instance, [8, 11, 16]). This paper presents various optimizations to the abstraction and search strategy, discusses completeness of the technique, and presents experimental results.

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 be used for abstraction. We require that all

* This research is also presented in [3] and 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).

invariants, switching 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 corresponding 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, where the computation of continuous successors of abstract states uses strategies inspired by the techniques used in CheckMate and `d/dt`. There are two significant benefits of postulating the verification problem as a search problem in the abstract system compared to the traditional approach of computing approximations of reachable sets of hybrid systems. First, the continuous reachability computation is applied only to an abstract state, instead of intermediate sets of arbitrary complexity generated during fixpoint computation. Second, while tools such as `d/dt` are designed to compute a "good" approximation of the continuous successors, we are interested only in checking if this set intersects with a new abstract state, permitting many optimizations. If the initial choice of predicates is too coarse, the search finds abstract counter-examples that are infeasible in the original hybrid system. We have also shown how to analyze such counter-examples to discover new predicates that will rule out related spurious counter-examples [5]. This strategy of iterative refinements of abstractions guided by counter-examples has also been incorporated in our verification tool, which is an integrated component of the modeling and analysis toolkit Charon [2].

After reviewing the previous work in Section 2, we present a variety of optimizations of the abstraction and search strategy in Section 3. If the original hybrid system has m locations and we are using k predicates for abstraction, the abstract state-space has at most $m \cdot 2^k$ states. To compute the abstract successors of an abstract state A, we need to compute the discrete and the continuous successor-set of A, and check if this set intersects with any of the abstract states. This can be expensive as the number of abstraction predicates grows, and our heuristics are aimed at speeding up the search in the abstract space. The first optimization implements a search constraint based on the additivity of flows of hybrid systems. A second optimization uses the BSP (Binary space partition) technique to impose a tree structure on abstract states so that invalid states (that is, inconsistent combinations of truth values to linear predicates) can be detected easily. A third optimization implements a guided search strategy. Since initial abstraction is typically coarse, the abstract search is likely to reach the target (i.e. unsafe states). During depth-first search, after computing the abstract successors of the current state, we choose to examine the abstract state whose distance to the target is the smallest according to an easily computable metric. We have experimented with a variety of natural metrics that are based on the shortest path in the discrete location graph of the hybrid system as well as the Euclidean distance between the polyhedra corresponding to abstract states and the target. Such a search improves the efficiency significantly in the initial iterations. Another optimization allows a location-specific choice of predicates for abstraction. Instead of having a global pool of abstraction predicates, each location is tagged with a relevant set of predicates, thereby reducing the size of the abstract state-space. The final optimization uses qualitative analysis of vector fields to rule out reachability of certain abstract states from a given abstract state *a priori* before applying the continuous reachability computation.

In Section 4, we address the completeness of our abstraction-based verification strategy for hybrid systems. Given a hybrid system H with linear dynamics, an initial set X_0, and a target set of unsafe states $\mathcal{B}$, the verification problem is to determine if there is an execution of H starting in X_0 and ending in $\mathcal{B}$. If there is such an execution, then even simulation can potentially demonstrate this fact. On the other hand, if the system is safe (i.e., $\mathcal{B}$ is unreachable), a symbolic algorithm that computes the set of reachable states from X_0 by iteratively computing the set of states reachable in one discrete or continuous step, cannot be guaranteed to terminate after a bounded number of iterations. Consequently, for completeness, we are interested in errors introduced by, first, approximating reachable sets in one continuous step using polyhedra, and second, due to predicate abstraction. We show that if the original system stays at least δ distance away from the target set for any execution involving at most n discrete switches and up to total time τ, then there is a choice of predicates such that the search in the abstract-space proves that the target set is not reached up to those limits. This shows that predicate abstraction can be used at least to prove bounded safety, that is, safety for all executions with a given bound on total time and a bound on the number of discrete switches.

In Section 5, we present case studies and experimental results. Our first example concerns a parametric version of Fischer's timing-based protocol for mutual exclusion. This model has 4 continuous variables, 23 locations, and we use 7 predicates for abstraction. We show that during initial iterations, guided search works quite well, and improves the time and space requirements. On the other hand, for establishing safety, location-specific choice of abstraction predicates reduces the number of reachable abstract states from 54 to 24. Our second example is an adaptive cruise controller that maintains a safe distance between cars based on communicated acceleration. The model has 5 continuous variables, 8 locations, and we use 17 predicates for abstraction. It can be completely analyzed using our verifier. We are also applying the tool to a design of an electronic throttle controller from DARPA's MoBIES project. The model has 9 continuous variables and 18 locations. Using all the 29 predicates mentioned in the model for the purpose of abstraction, our tool finds counter-examples. Since the model is incomplete, rigorous analysis of this example was not yet possible, but its size is a good indicator of the complexity that our tool can handle.

2 Predicate Abstraction for Hybrid Systems

In this section, we briefly recap the definitions of predicate abstraction for hybrid systems and the search strategy in the abstract state-space as outlined in [4].

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

- *$\mathcal{X} \subset \mathbb{R}^n$ 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***. We assume 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 reachability analysis can also be applied to hybrid systems having linear continuous dynamics with uncertain, bounded input of the form: $\dot{x} = A_l x + B_l u$.*
- *$T \subseteq L \times L \times \mathcal{C}_n \times (\Sigma_n)^n$ 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 linear hybrid system by formalizing its underlying transition system. The underlying transition system of H is $T_H = \{X, \to, X_0\}$. The state-space of the transition system is the state-space of H, i.e. $X = L \times \mathcal{X}$. The transition relation $\to \subseteq X \times X$ between states of the transition system is defined as the union of two relations $\to_C, \to_D \subseteq X \times X$. The relation $\to_C$ describes transitions due to continuous flows, whereas $\to_D$ describes transitions due to discrete jumps.

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

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 $\mathit{\Pi} = (\pi_1, \pi_2, \ldots, \pi_k) \in (\mathcal{L}_n)^k$. 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 $\mathit{\Pi}$; 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 $\mathit{\Pi} \in (\mathcal{L}_n)^k$ of n-dimensional linear predicates, an* **abstract state** *is a tuple $(l, \boldsymbol{b})$, where $l \in L$ and $\boldsymbol{b} \in \mathbb{B}^k$. The abstract state-space is $Q_\mathit{\Pi} := L \times \mathbb{B}^k$. The* **concretization function** *$C_\mathit{\Pi} : \mathbb{B}^k \to 2^\mathcal{X}$ for a vector of linear predicates $\mathit{\Pi} = (\pi_1, \ldots, \pi_k) \in (\mathcal{L}_n)^k$ is defined as $C_\mathit{\Pi}(\boldsymbol{b}) := \{x \in \mathcal{X} \mid \forall i \in \{1, \ldots, k\} : \pi_i(x) = b_i\}$. If $C_\mathit{\Pi}(\boldsymbol{b}) = \emptyset$, then the vector $b \in \mathbb{B}^k$ is infeasible with respect to $\mathit{\Pi}$.*

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

- *the abstract transition relation $\overset{\mathit{\Pi}}{\to} \subseteq Q_\mathit{\Pi} \times Q_\mathit{\Pi}$ is defined as the union of the following two relations $\overset{\mathit{\Pi}}{\to}_D, \overset{\mathit{\Pi}}{\to}_C \subseteq Q_\mathit{\Pi} \times Q_\mathit{\Pi}$. The relation $\overset{\mathit{\Pi}}{\to}_D$ represents transitions in the abstract state-space due to discrete jumps, whereas $\overset{\mathit{\Pi}}{\to}_C$ represents transitions due to continuous flows:*

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

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

A **trace** *in the abstract state-space is a sequence of abstract states $a_0, a_1, \ldots, a_n$, such that $a_0 \in Q_0$, and $a_i \overset{\mathit{\Pi}}{\to} a_{i+1}$ for $0 \leq i < n$.*

2.4 Searching the Abstract State-Space

Given a hybrid system H we want to verify 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 from an initial state to some state in $\mathcal{B}$ while in an unsafe location. We implemented an on-the-fly DFS search of the abstract state-space. In case we find an abstract state that violates the property, the current trace stored on a stack represents a counter-example. If the abstract system satisfies the property, then so does the concrete system. However, if a violation is found in the abstract system, then the resulting counter-example may or may not correspond to a counter-example in the concrete state-space.

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. The search strategy as outlined in [4] gives a priority to computing discrete successors rather than continuous successors, as the computation of discrete successors is generally much faster. 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. If the search of the abstract state-space finds that the abstract system is safe, then the concrete system is also safe. However, if the search finds a counter-example in the abstract state-space, then this counter-example may or may not correspond to a counter-example in the concrete state-space.

2.5 Counter-Example Analysis

If the predicate abstraction routine returns a counter-example, then this counter-example may not correspond to a counter-example in the concrete hybrid system. Such a counter-example is called *spurious*. We can analyze a counter-example and check whether it corresponds to a concrete one. If we find that the counter-example is indeed spurious, we can compute a set of new predicates based on this counter-example that ensures that a *refined* counter-example is not possible in the refined abstract state-space. Refinement of abstract states is based on the inclusion of the continuous concretizations, and refinement of paths additionally on following the same transitions. For more details about the analysis of counter-examples we refer the reader to [5].

3 Optimizations

If the original hybrid system has m locations and we are using k predicates for abstraction, the abstract state-space has $m \cdot 2^k$ abstract states. To compute the abstract successors of an abstract state A, we need to compute its discrete and continuous successors, and check if this set intersects with the other abstract states. This can be expensive as the number of abstraction predicates grows. We present optimizations in this section that are aimed at speeding up the discovery of counter-examples in the abstract state-space given a reachability property.

We include an optimization technique in the search strategy. Consider a 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: $(l, x) \rightarrow_C (l, x') \wedge (l, x') \rightarrow_C (l, x'') \Rightarrow (l, x) \rightarrow_C (l, x'')$. 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 spurious counter-examples that could have been found otherwise in the abstract transition system.

Another optimization concerns the construction of the abstract state-space. Since the predicates decompose the continuous state-space into polyhedral regions, instead of computing a polyhedron for each abstract state independently, we can use the Binary Space Partition (BSP) technique to incrementally construct the abstract state-space. The polyhedra resulting from partitioning the continuous state-space by one predicate after another are stored in a BSP tree as follows. First, the root of the tree is associated with the whole set $\mathcal{X}$. A predicate π_i is chosen from Π to partition $\mathcal{X}$ into 2 convex polyhedral subsets and create two child nodes: a left node is used to store the intersection of $\mathcal{X}$ with the half-space $\mathcal{H}(\pi_i)$ (which contains all points in $\mathcal{X}$ satisfying π_i) and a right node to store the intersection with the half-space $\overline{\mathcal{H}(\pi_i)}$. Then the non-empty polyhedra are partitioned recursively at the new nodes. Once all the predicates in Π have been considered, the non-empty polyhedra at the leaves of the tree correspond to the closure of the concretizations of all possible consistent abstract states. This construction is illustrated by figure 1 where the continuous state-space $\mathcal{X}$ is a rectangle in two dimensions and the vector of initial predicates $\Pi = (\pi_1, \pi_2, \pi_3)$. The predicate π_1 partitions $\mathcal{X}$ into 2 polygons P_1 and $P_{\bar{1}}$. Next, splitting P_1 and $P_{\bar{1}}$ by the predicate π_2 gives P_{12}, $P_{1\bar{2}}$ and $P_{\bar{1}2}$, $P_{\bar{1}\bar{2}}$. Then, only the interior of the polygon $P_{\bar{1}\bar{2}}$ intersects with the hyperplane of the predicate π_3 while all other polygons in the current decomposition lie entirely inside $\overline{\mathcal{H}(\pi_3)}$; therefore, only $P_{\bar{1}\bar{2}}$ is split. This BSP tree provides simultaneously a geometric representation of the state-space and a search structure. Note that the amount of splitting depends on the order of predicates and the abstract states. This order is determined by the search strategy, more precisely, the tree is built on-the-fly, based on the decision which abstract state to explore next. This BSP construction allows fast detection of combinations of predicates that give inconsistent abstract states and thus saves a significant amount of polyhedral computations.

3.1 Guided Search

The predicate abstraction implementation performs an on-the-fly depth-first search. Since an abstract state has many successors, the performance of the

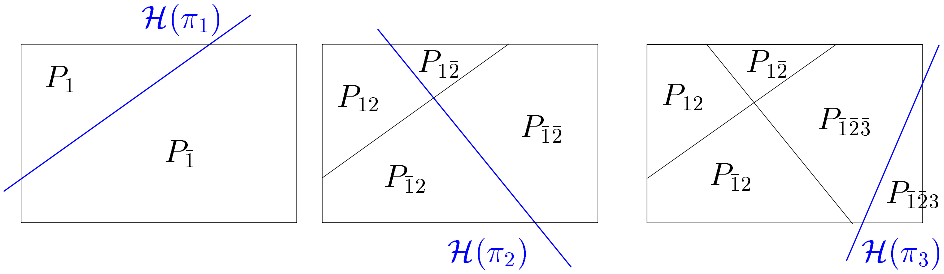

Fig. 1. BSP-based construction of the abstract state-space

search depends on which successor is examined next to continue the search at every step. We present three guided search strategies that we have recently implemented in our tool. In each case, we define a priority function $\rho : Q_\Pi \to \mathbb{R}$ that tells us how "close" each abstract state is to the set of unsafe states. As we are trying to minimize the time it takes to discover a counter-example, we prefer states that are "closer" to the set of unsafe states.

Given a hybrid system $H = (\mathcal{X}, L, X_0, I, f, T)$, we can define a graph $G_H = (V, E)$ such that $V = L$ and $(l, l') \in E$ iff $\exists (l, l', g, r) \in T$. Given the set of unsafe locations $U \subseteq L$, we define a priority function $\rho_D : L \to \mathbb{N}$ on locations as:

$$\rho_D(l) = \begin{cases} 0 & : \quad l \in U, \\ \text{shortest path length from } l \text{ to } U \text{ in } G_H & : \quad l \notin U \wedge \text{path exists}, \\ \infty & : \quad \text{otherwise.} \end{cases}$$

It is clear that $\forall l \in L : \rho_D(l) \neq \infty \Rightarrow 0 \leq \rho_D(l) \leq |L| - 1$. We use ρ_D in all three guided search strategies that we introduce in the following.

Mask Priority. The mask priority guided search strategy is based on the boolean vector representation of the continuous part of an abstract state. We define a mask $\boldsymbol{m} \in \mathbb{T}^k$ that represents a compact description of the continuous part of all abstract states that intersect with the set of unsafe continuous states $\mathcal{B}$. Given a predicate π, $\mathcal{H}(\pi)$ denotes the half-space defined by π and $\overline{\mathcal{H}(\pi)}$ denotes the complement of $\mathcal{H}(\pi)$. Then we define $\boldsymbol{m} = (m_1, \ldots, m_k)$ as:

$$m_i = \begin{cases} 1 & : \quad \mathcal{B} \subseteq \mathcal{H}(\pi_i), \\ 0 & : \quad \mathcal{B} \subseteq \overline{\mathcal{H}(\pi_i)}, \\ * & : \quad \text{otherwise.} \end{cases}$$

We then define a comparator function $\delta : \mathbb{B} \times \mathbb{T} \to \mathbb{B}$ as

$$\delta(b, t) = \begin{cases} 1 & : \quad t = 1 \wedge b = 0 \vee t = 0 \wedge b = 1, \\ 0 & : \quad \text{otherwise,} \end{cases}$$

and a priority function $\rho_1 : \mathbb{B}^k \to \mathbb{N}$ as Clearly, $\forall \boldsymbol{b} \in \mathbb{B}^k : 0 \leq \rho_1(\boldsymbol{b}) \leq k$. The value $\rho_1(\boldsymbol{b})$ represents the number of positions in the vector representation $\boldsymbol{b}$ that contradict the corresponding position in the mask $\boldsymbol{m}$. To combine ρ_1 with ρ_D to form a priority function $\rho_M : Q_\Pi \to \mathbb{N}$ over abstract states we define

$$\rho_M(l, \boldsymbol{b}) = \begin{cases} \infty & : \quad \rho_D(l) = \infty, \\ (k+1)\rho_D(l) + \rho_1(\boldsymbol{b}) & : \quad \text{otherwise.} \end{cases}$$

Euclidean Distance Priority. The Euclidean distance priority guided search strategy differs from the mask priority one because it does not rely on the boolean vector representation enforced by the chosen predicates for the abstraction. Instead, it measures the Euclidean distance from the continuous part of the abstract state to the set of unsafe states. To do so, we define the distance between two non-empty convex polyhedral sets $P \subseteq \mathcal{X}$ and $Q \subseteq \mathcal{X}$ as follows: $d(P, Q) = \inf\{d(p-q) \mid p \in P \wedge q \in Q\}$ where $d(\cdot)$ denotes the Euclidean distance. Then the priority function $\rho_2 : \mathbb{B}^k \to \mathbb{R}$ can be computed as $\rho_2(\boldsymbol{b}) = d(C_\Pi(\boldsymbol{b}), \mathcal{B})$. As $\mathcal{X}$ is bounded, we can compute the limit for any two non-empty convex subsets of $\mathcal{X}$, and we denote that value with $d_\mathcal{X}$. We can then combine ρ_2 with ρ_D to form the priority function $\rho_E : Q_\Pi \to \mathbb{R}$ by defining:

$$\rho_E(l, \boldsymbol{b}) = \begin{cases} \infty & : \quad \rho_D(l) = \infty, \\ (d_\mathcal{X}+1)\rho_D(l) + \rho_2(\boldsymbol{b}) & : \quad \text{otherwise.} \end{cases}$$

Reset Distance Priority. The Euclidean distance priority does not consider the effects of resets of the continuous variable x enforced by switches in the concrete system. The reset distance priority guided search strategy favors abstract states in any location which are close to the set of unsafe states in an unsafe location after appropriate resets are taken into consideration. Appropriate resets are those that lead the current abstract state to an unsafe location. We define the reset distance priority function $\rho_R : Q_\Pi \to \mathbb{R}$ by $\rho_R(l, \boldsymbol{b}) = \rho_3(l, C_\Pi(\boldsymbol{b}))$ and $\rho_3 : L \times 2^\mathcal{X} \to \mathbb{R}$ by

$$\rho_3(l, X) = \begin{cases} d(X, \mathcal{B}) & : \quad \rho_D(l) = 0, \\ \min_{\substack{(l,l',g,r) \in T \\ \rho_D(l') = \rho_D(l) - 1}} \rho_3(l', r(X)) & : \quad \text{otherwise.} \end{cases}$$

The reset distance represents the smallest Euclidean distance of the current abstract state to the unsafe set in a shortest path to an unsafe location, if no more transitions due to continuous flow occur.

3.2 Generalized Predicate Abstraction

We present a formal framework of abstractions of predicate abstraction –generalized predicate abstraction, which allows clustering of abstract states. For illustrative purposes, we use a location-specific predicate abstractor: The main idea of the location-specific predicate abstraction routine is the fact that certain predicates are only important in certain locations. Consider for example guards and invariants. A specific predicate representing an invariant may be important in one location of the hybrid automaton, but may not be relevant in the other locations. Considering this predicate only in the location it is really needed, may reduce the number of reachable abstract states considerably. This is similar to optimizations in predicate-abstraction based tools for model checking of C programs, such as Cartesian predicate abstraction [7]. We define a tri-valued domain $\mathbb{T} := \{0, 1, *\}$, and a function $c : \mathbb{T}^k \to 2^{\mathbb{B}^k}$ as $c(\boldsymbol{t}) = \{\boldsymbol{b} \in \mathbb{B}^k | t_i \neq * \Rightarrow b_i = t_i\}$.

Definition 4. *The generalized predicate abstract state-space is defined as* $\hat{Q}_\Pi := L \times \mathbb{T}^k$*, such that* $(l, \boldsymbol{t}) \to_G (l', \boldsymbol{t'})$ *iff* $\exists \boldsymbol{b} \in c(\boldsymbol{t}), \boldsymbol{b'} \in c(\boldsymbol{t'}) : (l, \boldsymbol{b}) \overset{\Pi}{\to} (l', \boldsymbol{b'})$*. The set of initial abstract states is* $\hat{Q}_0 := \{(l, \boldsymbol{t}) \in \hat{Q}_\Pi | \exists \boldsymbol{b} \in c(\boldsymbol{t}) : (l, \boldsymbol{b}) \in Q_0\}$.

The above definition allows a concrete state $(l, x) \in \mathcal{X}$, as well as an abstract state $(l, \boldsymbol{b}) \in Q_\Pi$, to be represented by many states in $\hat{Q}_\Pi$. Hence, we restrict our attention to a subset of $\hat{Q}_\Pi$ that is both a partition and a cover of $\mathbb{B}^k$.

Definition 5. *A subset of abstract states* $Q \subseteq \hat{Q}_\Pi$ *is called* **location-specific***, iff* $\forall l \in L, \boldsymbol{b} \in \mathbb{B}^k \exists \boldsymbol{t} \in \mathbb{T}^k : \boldsymbol{b} \in c(\boldsymbol{t}) \wedge (l, \boldsymbol{t}) \in Q$*, and* $\forall (l, \boldsymbol{t_1}), (l, \boldsymbol{t_2}) \in Q : \boldsymbol{t_1} \neq \boldsymbol{t_2} \Rightarrow c(\boldsymbol{t_1}) \cap c(\boldsymbol{t_2}) = \emptyset$*. The set of transitions for a location-specific* Q *is the restriction of* $\to_G$ *to* Q*. The set of initial states is the restriction of* $\hat{Q}_0$ *to* Q*.*

The search in the generalized abstract state-space needs only slight modifications. The computation of the continuous successor-set of a generalized abstract state does not need any alteration, as transitions due to continuous flow do not change the location of the states and, therefore, the set of predicates remains the same. On the other hand, we need to modify the computation of the discrete successor-set. The weakest precondition computation for a particular discrete switch needs to accommodate for the fact that the set of predicates in the locations before and after the switch are not necessarily the same anymore. The following theorem stating the soundness of this approach is based on the soundness of the predicate abstraction algorithm [4].

Theorem 1. *If the generalized predicate abstraction routine terminates and reports that the system is safe, then the corresponding concrete system is also safe.*

3.3 Vector Field Analysis

In order to construct the discrete abstraction of a hybrid system, we need to compute the continuous successors of an abstract state, and check if this set intersects with the other abstract states. In this section we present a method, based on a qualitative analysis of the vector fields, that avoids the test for feasibility of some transitions. This allows to obtain a first rough over-approximation of the transition relation which is then refined using reachability computations. Similar ideas of qualitative analysis of vector fields have been used in [18].

Geometrically speaking, the concretizations $C_\Pi(\boldsymbol{b})$ for all $\boldsymbol{b} \in \mathbb{B}^k$ form a convex decomposition of the concrete state-space $\mathcal{X}$. Hence, for any two non-empty abstract states $(l, \boldsymbol{b})$ and $(l, \boldsymbol{b}')$, the closures of their concretizations $cl(C_\Pi(\boldsymbol{b}))$ and $cl(C_\Pi(\boldsymbol{b}'))$ are either disjoint or have only one common facet. We now focus on the latter case and denote by F the common facet. We assume that F is a $(n-1)$-dimensional polyhedron. Let $\boldsymbol{n}_F$ be the normal of F which points from $C_\Pi(\boldsymbol{b}')$ to $C_\Pi(\boldsymbol{b})$. If for all points on the face F the projection of f_l on $\boldsymbol{n}_F$ is non-negative, that is,

$$\forall x \in F \ \langle f_l(x), \boldsymbol{n}_F \rangle \geq 0, \tag{1}$$

then there exists a trajectory by continuous dynamics f_l from $C_\Pi(\boldsymbol{b}')$ to $C_\Pi(\boldsymbol{b})$. Moreover, any trajectory from $C_\Pi(\boldsymbol{b})$ to $C_\Pi(\boldsymbol{b}')$ by f_l, if one exists, must cross another polyhedron $C_\Pi(\boldsymbol{b}'')$. In the context of predicate abstraction, this means that the transition from $(l, \boldsymbol{b})$ to $(l, \boldsymbol{b}')$ is feasible. Furthermore, we need not consider the transition by f_l from $(l, \boldsymbol{b})$ to $(l, \boldsymbol{b}')$ because this transition, if possible, can be deduced from the transitions via some other intermediate states. Note that when the dynamics f_l is affine, in order for the condition (1) to hold, it suffices that $\langle f_l(x), \boldsymbol{n}_F \rangle$ is non-negative at all the vertices of F.

On the other hand, if the dynamics f_l is stable, we can use the standard Lyapunov technique for linear dynamics to rule out some abstract states that cannot be reached from $(l, \boldsymbol{b})$ as follows. Let P be the solution of the Lyapunov equation of the dynamics f_l and $\mathcal{E}$ be the smallest ellipsoid of the form $\mathcal{E} = \{x \mid x^T P x \leq \alpha\}$ that contains the polyhedron $C_\Pi(\boldsymbol{b})$. We know that $\mathcal{E}$ is invariant in the sense that all trajectories from points inside $\mathcal{E}$ remain in $\mathcal{E}$. Consequently, all the abstract states $(l, \boldsymbol{b}')$ such that $C_\Pi(\boldsymbol{b}') \cap \mathcal{E} = \emptyset$ cannot be reached from $(l, \boldsymbol{b})$ by continuous dynamics f_l.

4 Bounded Completeness

Given a hybrid system H with linear dynamics, an initial set X_0, and a target set $\mathcal{B} \subseteq X$, the verification problem is to determine if there is an execution of H starting in X_0 and ending in $\mathcal{B}$. If there is such an execution, then even simulation can potentially demonstrate this fact. On the other hand, if the system is safe (i.e., $\mathcal{B}$ is unreachable), a *complete* verification strategy should be able to demonstrate this. However, a symbolic algorithm that computes the set of reachable states from X_0 by iteratively computing the set of states reachable in one discrete or continuous step, cannot be guaranteed to terminate after a bounded number of iterations. Consequently, for completeness, we focus on errors introduced by approximating reachable sets in one continuous step using polyhedra, as well as due to predicate abstraction. We show that predicate abstraction is complete for establishing bounded safety; that is, unreachability of unsafe states for a specified number of discrete switches and time duration.

4.1 Completeness for Continuous Systems

We can present a completeness result if we focus on purely continuous systems first. We use two additional assumptions for this result. We only consider systems that exhibit a separation of the reachable state-space and the unsafe states. In addition we use the knowledge of the optimization of the search strategy which prohibits multiple successive continuous successors.

We assume a purely continuous system such that we can specify the initial convex region $\mathcal{X}_0 := \{x \in \mathcal{X} | (l_0, x) \in X_0\}$ and the set of unsafe states $\mathcal{B}$ respectively using the conjunction of a finite set of predicates. In addition, assume a separation of the set $\texttt{Post}(\mathcal{X}_0)$ of continuous states reachable from $\mathcal{X}_0$, and $\mathcal{B}_{\mathcal{X}}$, the projection of $\mathcal{B}$ onto $\mathcal{X}$, that is $d(\texttt{Post}(\mathcal{X}_0), \mathcal{B}_{\mathcal{X}}) \geq \epsilon$. Following [12], we know that we can find a small enough time-step that ensures that the over-approximation error due to the computation of convex hulls will not result in an overlap of the over-approximation of $\texttt{Post}(\mathcal{X}_0)$ with $\mathcal{B}_{\mathcal{X}}$. Additionally, we assume that the set of predicates used for predicate abstraction entails all the predicates corresponding to the linear constraints needed to specify the polyhedral sets $\mathcal{X}_0$ and $\mathcal{B}_{\mathcal{X}}$. Given the optimization of our search strategy it is clear that any abstract refinement of $\mathcal{B}$ will not be declared reachable by the search.

4.2 (n, τ, δ)-Safety

We can prove that our predicate abstraction model checker is complete to establish safety up to a fixed number of discrete switches and time duration. Note that the recent research on *bounded model checking* [10] can be viewed as establishing safety of discrete systems up to a fixed number of transitions. We first define this notion of bounded safety for hybrid systems formally. For this purpose, we define a distance function $d : X \times X \to \mathbb{R}_{\geq 0}$ on X as

$$d((l,x),(l',x')) = \begin{cases} d(x,x') & : \quad l = l', \\ \infty & : \quad \text{else}; \end{cases}$$

and generalize over sets of states by $d(S, S') = \min_{(l,x) \in S, (l',x') \in S'} d((l,x),(l',x'))$.

Definition 6. *A hybrid system $H = (\mathcal{X}, L, X_0, I, f, T)$ is called (n, τ, δ)-***safe** *for the unsafe set $\mathcal{B}$, iff the set of states $R_{(n,\tau)}(X_0)$ that is reachable using at most n discrete switches and combined flow of at most τ time-units from the initial states X_0 has a distance of at least δ to the set of unsafe states $\mathcal{B}$: $d(R_{(n,\tau)}(X_0), \mathcal{B}) > \delta$.*

The proof, which is omitted here for the sake of brevity, shows that if the original system stays at least δ distance away from the target set for the first n discrete switches and up to total time τ, then there is a choice of predicates such that the search in the abstract space proves that the target set is not reached up to those limits. This shows that predicate abstraction can be used at least to prove bounded safety, that is, safety for all execution with a given bound on total time and a bound on discrete switches.

Theorem 2 (Bounded Completeness). *The predicate abstract model checker is complete for the class of (n, τ, δ)-safe hybrid systems.*

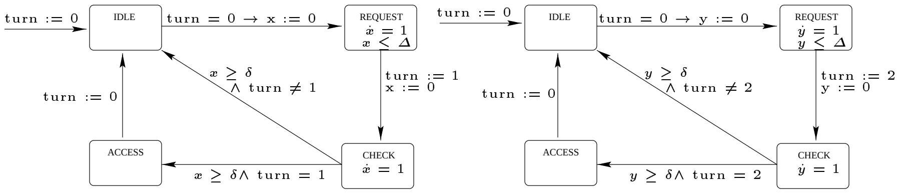

Fig. 2. The two processes for the mutual exclusion example

5 Implementation and Experimentation

The optimizations discussed in this paper have been incorporated in our verification tool. The current prototype implementation of the predicate abstraction model checking tool is 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. A detailed overview of a verification case study starting from Charon source code is given in [15]. Our tool uses the polyhedral libraries CDD and QHull. It also includes a counter-example analysis and predicate discovery tool as described in more detail in [5].

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 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 present a flattened version of the two-process protocol in Table 1. We use the flat model in the following sections to illustrate the mask priority guided search strategy and the generalized predicate abstraction.

Mask Priority. We consider Fischer's two-process protocol example for the case that $\Delta \geq \delta$. As the set of unsafe states corresponds to any continuous state in location 22 (see Table 1), we have $\rho_M(l, \boldsymbol{b}) = \rho_D(l)$. Starting in the abstract state "$l = 0, 0 \leq x < \delta \leq \Delta, 0 \leq y < \delta$" with priority 6, the guided search tries to find a path that leads to a state with priority 0 by reducing the priority as much as possible at each step. In this example, this means that the search tries to reduce the priority of the next abstract state by exactly one at each step. In the case that this is not possible, a continuous transition is considered as this does not affect the priority. It can easily be seen that a valid counter-example

l	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
P_1	I	R	C	A	I	R	C	A	I	R	C	A	C	I	R	C	I	C	C	A	C	R	A
P_2	I	I	I	I	R	R	R	R	C	C	C	C	C	A	A	A	C	I	A	C	R	C	A
`turn`	0	0	1	1	0	0	1	1	2	2	2	2	1	2	2	2	0	0	1	1	0	0	—
$\rho_D(l)$	6	5	8	7	5	4	3	2	8	3	9	1	9	7	2	8	7	7	1	8	6	6	0

Table 1. The two-process Fischer's protocol as a flat model: The locations l are numbered from 0 to 22. P_1 specifies the local location of the first process: I represents the `Idle` location, R the `Request` location, C the `Check` location, and A the `Access` location. P_2 specifies the location of the second process, whereas `turn` specifies the value of the `turn` variable in the composed system. Location 0 is the initial location. Location 22 violates the mutual exclusion property regardless of the value of the `turn` variable.

can be constructed just by following this decrease of ρ_D. This is in contrast to our previous search algorithm, which always prefers discrete transitions over continuous ones. This leads the search away from a shortest path to a counterexample. Even in this small example, the previous search finds more than 10 other abstract states first.

Generalized Predicate Abstraction. We also consider the verification of Fischer's protocol to illustrate the advantage of the location-specific predicate abstraction routine. The verification using the regular predicate abstraction technique finds 54 reachable abstract states (see [4]), whereas, if we use the location-specific predicates as described in Table 2, we only reach 24 abstract states.

l	Π	l	Π	l	Π
0	—	8	$y \geq \delta$	16	$y \geq \delta$
1	$x \leq \Delta$	9	$x \leq \Delta, y \geq \delta, x \geq y$	17	$x \geq \delta$
2	$x \geq \delta$	10	$x \geq \delta, y \geq \delta, x \geq y$	18	$x \geq \delta$
3	—	11	$y \geq \delta$	19	$y \geq \delta$
4	$y \leq \Delta$	12	$x \geq \delta, y \geq \delta, x \leq y$	20	$x \geq \delta, y \leq \Delta$
5	$x \leq \Delta, y \leq \Delta$	13	—	21	$x \leq \Delta, y \geq \delta$
6	$x \geq \delta, y \leq \Delta, x \leq y$	14	$x \leq \Delta$	22	—
7	$y \leq \Delta$	15	$x \geq \delta$		

Table 2. Location-specific predicates for the 2-process Fischer's protocol example. The predicates $0 \leq \Delta, 0 \leq \delta, 0 \leq x, 0 \leq y, \Delta < \delta$ are supposed to be present in all locations.

5.2 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. Let us first briefly describe the model (see [13] for a detailed description). 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 the interval $[da_{lmin}, da_{lmax}]$. The dynamics of the system is described by the following differential equations: $\dot{x} = v, \quad \dot{v} = u, \quad \dot{x}_l = v_l, \quad \dot{v}_l = a_l, \quad \dot{a}_l \in [da_{lmin}, da_{lmax}]$, where u is the input that controls the acceleration of the car. In this mode, the controller consists of several modes. The control law to maintain the desired speed is as follows:

$$u_1 = \begin{cases} 0.4\varepsilon_v & : \quad a_{cmin} \leq 0.4\,\varepsilon_v \leq a_{cmax}, \\ a_{cmin} & : \quad 0.4\,\varepsilon_v < a_{cmin}, \\ a_{cmax} & : \quad 0.4\,\varepsilon_v > a_{cmax}, \end{cases}$$

where $\varepsilon_v = v - v_d$ is the error between the actual and the desired speed; a_{cmin} and a_{cmax} are the maximal comfort deceleration and acceleration.

In addition, in order for the car to follow its preceding car safely, another control law is designed as follows. A safety distance between cars is defined as $D = \max\{\, G_c\, v_l,\ D_d\,\}$ where G_c is the time gap parameter; D_d is the desired sensor range given by $D_d = 0.5\, v_l^2\, (-1/a_{min} + 1/a_{lmin}) + 0.02\, v_l$; a_{min} and a_{lmin} are the maximal decelerations of the cars. Then, the control law allowing to maintain the safety distance with the car in front is given by $u_{follow} = a_l + (v_l - v) + 0.25\,(x_l - x - 5 - D)$. Since the acceleration of the car is limited by its maximal breaking capacity, the control law to avoid collision is indeed $u_2 = \max\{\, a_{min},\ u_{follow}\,\}$. The combined switching control law is given by $u = \min\{u_1, u_2\}$. This means that the controller uses the control law u_1 to maintain the desired speed if the car in front is far and travels fast enough, otherwise it will switch to u_2.

The closed-loop system is modeled as a hybrid automaton with 5 continuous variables (x, v, x_l, v_l, a_l) and 8 locations corresponding to the above described switching control law. 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 $x_l - x \leq 0$. To define initial predicates, in addition to the constraints of the invariants and guards, we use the predicate of the unsafe set allowing to distinguish safe and unsafe states and another predicate on the difference between the speed and acceleration of the cars. The total number of the initial predicates used to construct the discrete abstraction is 17. For an initial set specified as $x_l - x \geq 100 \wedge v \geq 5$, the tool found 55 reachable abstract states and reported that the system is safe. For this model, in a preprocessing step using the Binary Space Partition technique, the tool found that the chosen set of initial predicates partitions the continuous state space into 785 polyhedral regions, and this enables to reduce significantly the computation time.

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 using counter-example guided predicate abstraction. Technical Report MS-CIS-02-34, University of Pennsylvania, November 2002.
4. 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, pages 35–48, March 2002.
5. R. Alur, T. Dang, and F. Ivančić. Counter-example guided predicate abstraction of hybrid systems. In *Ninth International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, April 2003.
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. Springer, 2000.
7. T. Ball, A. Podelski, and S. Rajamani. Boolean and Cartesian abstraction for model checking C programs. In *Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 2031. Springer, 2001.
8. 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.
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. Springer, 1999.
10. E.M. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded model checking using satisfiability solving. *Formal Methods in Systems Design*, 19(1):7–34, 2001.
11. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. of the 4th ACM Symposium on Principles of Programming Languages*, 1977.
12. T. Dang. *Verification and Synthesis of Hybrid Systems.* PhD thesis, Institut National Polytecnique de Grenoble, 2000.
13. A.R. Girard. *Hybrid System Architectures for Coordinated Vehicle Control.* PhD thesis, University of California at Berkeley, 2002.
14. 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.
15. F. Ivančić. Report on verification of the MoBIES vehicle-vehicle automotive OEP problem. Technical Report MS-CIS-02-02, University of Pennsylvania, March 2002.
16. 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.
17. I. Mitchell and C. Tomlin. Level set methods for computation in hybrid systems. In *Hybrid Systems: Computation and Control*, LNCS 1790. Springer, 2000.
18. O. Stursberg, S. Kowalewski, and S. Engell. Generating timed discrete models of continuous systems. In *Proc. 2nd IMACS Symposium on Mathematical Modeling*, pages 203–209, 1997.
19. A. Tiwari and G. Khanna. Series of abstractions for hybrid automata. In *Hybrid Systems: Computation and Control, Fifth Intern. Workshop*, LNCS 2289, 2002.