Predicate Abstraction for Reachability Analysis of Hybrid Systems

RAJEEV ALUR
University of Pennsylvania
and
THAO DANG
VERIMAG
and
FRANJO IVANČIĆ
NEC Laboratories America

Embedded systems are increasingly finding their way into a growing range of physical devices. These embedded systems often consist of a collection of software threads interacting concurrently with each other and with a physical, continuous environment. While continuous dynamics have been well studied in control theory, and discrete and distributed systems have been investigated in computer science, the combination of the two complexities leads us to the recent research on *hybrid systems*. This paper addresses the formal analysis of such hybrid systems.

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 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 present the basic techniques for guided search in the abstract state-space, optimizations of these techniques, implementation of these in our verifier, and case studies demonstrating the promise of the approach. We also address the completeness of our abstraction-based verification strategy by showing that predicate abstraction of hybrid systems can be used to prove bounded safety.

Categories and Subject Descriptors: D.2.4 [**Software**]: Software Engineering—*Software/Program Verification*; J.6 [**Computer Applications**]: Computer-Aided Engineering

General Terms: Algorithms, Reliability, Verification

Additional Key Words and Phrases: Rechability Analysis, Hybrid Systems, Predicate Abstraction

1. INTRODUCTION

Embedded systems are increasingly finding their way into a growing range of physical devices [Computer Science and Telecommunications Board 2001]. An embedded system typically consists of a collection of software threads interacting concurrently with each other and with a physical, continuous environment through sensors and actuators. They are becoming ever more sophisticated with respect to their software requirements, as the usage and demand for such systems evolve. Therefore, the need for a structured approach for developing embedded software is becoming increasingly urgent.

Traditionally, control theory and related engineering disciplines have addressed

the problem of designing robust control laws to ensure optimal performance of physical processes with continuous dynamics. This approach to system design has largely ignored the problem of implementing such control laws in software. Therefore, issues related to concurrency and communication have not been addressed appropriately in this setting. Computer science and software engineering on the other hand have an entirely discrete view of the world, which abstracts from the physical characteristics of the environment to which the software is reacting. Therefore, this approach is typically unable to guarantee safety or a suitable performance of the embedded device as a whole. An embedded system consisting of sensors, actuators, plant, and control software, then, is best viewed as a hybrid (mixed discrete-continuous) system. Hybrid modeling combines the two approaches and is natural for the specification of embedded systems.

Inspired by the success of model checking in hardware verification and protocol analysis [Clarke and Kurshan 1996; Holzmann 1997], there has been increasing research on developing algorithms and tools for automated verification of hybrid models of embedded controllers [Alur and Dill 1994; Halbwachs et al. 1994; Alur et al. 1995; Henzinger et al. 1995; Daws et al. 1996; Bengsston et al. 1998; Chutinan and Krogh 1999; Alur et al. 2000; Asarin et al. 2000; Mitchell and Tomlin 2000; Tiwari and Khanna 2002]. Model checking requires the computation of the set of reachable states of a model, and in presence of continuous dynamics, this is typically undecidable. The state-of-the-art computational tools for model checking of hybrid systems are of two kinds. Tools such as KRONOS [Daws et al. 1996], UPPAAL [Bengsston et al. 1998], and HYTECH [Henzinger et al. 1995] 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 [Chutinan and Krogh 1999], `d/dt` [Asarin et al. 2000], and level-sets method [Greenstreet and Mitchell 1999; Mitchell and Tomlin 2000], approximate the set of reachable states by polyhedra or ellipsoids [Kurzhanski and Varaiya 2000] using 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 [Cousot and Cousot 1977; Loiseaux et al. 1995; Graf and Saidi 1997; Das et al. 1999; Ball and Rajamani 2000]. 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 counterexample 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 [Corbett et al. 2000], SLAM [Ball and Rajamani 2000], Feaver [Holzmann and Smith 2000] and BLAST [Henzinger et al. 2002] 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 to systems 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. As elaborated in Section 2, a combination of values to these k Boolean variables represents an abstract state corresponding to a set of continuous states, 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 verification tool is integrated into the modeling and analysis toolkit CHARON [Alur et al. 2003].

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, which is described in Section 3. Computing discrete successors is relatively straightforward, and involves computing weakest preconditions, and checking nonemptiness of an intersection of polyhedral sets. To compute 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. Our early experiments indicate that improvements in time and space requirements are significant compared to a tool such as `d/dt`.

In Section 4 we present a variety of optimizations of the abstraction and search strategy. If the original hybrid system has m locations and we are using k predicates for abstraction, the abstract state-space has $m \cdot 2^k$ 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 eliminates some spurious counterexamples in the abstract state-space by requiring that a counterexample is not permitted to consist of two or more consecutive continuous transitions in the abstract state-space. The optimization is based on the fact that you can always find an equivalent valid path in the concrete state-space that does not contain two or more consecutive continuous transitions. 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. The third optimization uses qualitative analysis of vector fields to rule out reachability of certain abstract states from a given abstract states *a priori* before applying the continuous reachability computation. Another 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 shortest distance between the polyhedra corresponding to the abstract states. Such a priority-based search improves the efficiency significantly in the initial iterations. The final 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. Again, this strategy is shown to be effective in speeding up the computation in our case studies.

We also address the completeness of our abstraction-based verification strategy for hybrid systems, which is described in Section 5. Given a hybrid system H with linear dynamics, an initial set X_0, and a target set $\mathcal{B}$, 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.

We demonstrate the feasibility of our approach using five case studies in Section 6. The first one involves verification of a parametric version of Fischer's protocol for timing-based mutual exclusion. The second and third ones involve analysis of different models of automotive cruise controller. The fourth example is based on a navigation benchmark for hybrid systems proposed in [Fehnker and Ivančić 2004]. The fifth example is a thermostat example that is used to illustrate the concepts throughout this paper. In each of these cases, we show how predicate abstraction can be effective in establishing safety of the system.

We conclude this paper with some final remarks. In particular, we address the issue of finding appropriate predicates to be used for the abstraction of the considered system. For the sake of brevity, we briefly review the notion of counterexample guided predicate abstraction for hybrid systems as described in [Alur et al. 2003a].

Related Work. In the following, we give a brief overview of the state-of-the-art computation tools for model checking hybrid systems. A detailed description of the various tools can be found in [Silva et al. 2001].

The tool UPPAAL is an environment to model, simulate, and verify systems represented as networks of timed automata [Bengsston et al. 1998]. Timed automata are hybrid systems where each continuous variable x is a clock and thus follows the differential equation $\dot{x} = 1$ [Alur and Dill 1994]. UPPAAL additionally allows data variables and synchronization mechanisms to model communication between concurrent timed automata. It can analyze reachability properties and simple liveness properties. The timed automata are internally represented in a compact form using *clock difference diagrams* [Behrmann et al. 1999]. Additional information about UPPAAL can be found at `www.docs.uu.se/docs/rtmv/uppaal/`. KRONOS is another tool for the analysis of timed automata. More information about KRONOS can be found in [Daws et al. 1996] and online at `www-verimag.imag.fr/TEMPORISE/kronos`.

The tool HYTECH analyzes a class of hybrid systems called *linear hybrid automata* [Alur et al. 1996]. The tool allows flows of the form $A\dot{x} \leq b$. HYTECH can analyze a set of concurrent automata and can perform parametric analysis since it uses a symbolic model checking approach. A counterexample trace is generated if verification of a property fails. Details about HYTECH can be found in [Henzinger et al. 1997] and online at `www-cad.eecs.berkeley.edu/~tah/HyTech/`.

CHECKMATE is a MATLAB-based tool for simulation and verification of *threshold-event driven hybrid systems* (TEDHS) [Cury et al. 1998]. In a TEDHS the changes in the discrete state can occur only when continuous state variables encounter specified thresholds represented by hyperplanes. The TEDHS model specified in MATLAB is converted into a *polyhedral-invariant hybrid automaton* (PIHA) used for verification [Chutinan and Krogh 2000]. PIHA are automata with invariants defined by the hyperplanes defining guards for the transitions leaving modes. The resulting PIHA is equivalent to the original TEDHS within a bounded region of the

continuous state-space. CheckMate can analyze properties expressed in ACTL [Clarke et al. 1993]. The tool computes a finite-state approximation using general polyhedral over-approximations to the sets of reachable states for the continuous dynamics called *flowpipes*. The tool then performs a search in the completely constructed transition system. Recently, there has been work in adding a counter-example guided refinement procedure to the tool [Clarke et al. 2003; Clarke et al. 2003].

The tool `d/dt` performs verification and control synthesis for hybrid systems. It computes the reachable sets for models with linear continuous dynamics with uncertain, bounded input. The continuous dynamics are of the form $\dot{x} = Ax + Bu, u \in U$, where U is a bounded set of inputs. Reachable sets are represented by *orthogonal* polyhedra computed by performing so-called *face-lifting* to create efficient over-approximations [Dang and Maler 1998]. We review the reachability computations used by `d/dt` in more detail in Section 3.2.

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 exclude the orthogonal approximation step for the following reasons. First, we do not require to accumulate concrete continuous successors in our predicate abstraction search. 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.

Recently, there has been increased interest in applying abstraction techniques to the verification of hybrid systems. In [Tiwari and Khanna 2002] the authors propose the use of data abstraction techniques for the analysis of hybrid systems with polynomial continuous dynamics. For the purposes of abstraction the authors use a set of polynomials that partitions the continuous state-space into sign-invariant zones. A prototype tool has been implemented in the SAL environment [Bensalem et al. 2000; de Moura et al. 2004]. The abstract system is completely constructed using logical reasoning in the theory of reals. The resulting finite abstract transition system can then be passed to a traditional discrete model checker. A disadvantage of this approach is that the complete abstract transition system is constructed beforehand and cannot be searched on-the-fly. Although refinements of the discrete system are possible by adding new polynomials, there is no automatic refinement generator.

2. LINEAR HYBRID SYSTEMS

In this section, we define the class of hybrid system that we consider. We define the class of linear hybrid systems (LHS), which are hybrid systems with linear continuous dynamics with uncertain, bounded input. This class of hybrid systems should not be confused with so-called linear hybrid automata [Henzinger et al. 1997] used in the HyTech tool. As mentioned in the introduction, linear hybrid automata

allow continuous flows of the form $\dot{x} \leq Ab$ which is a small fragment of the continuous flows allowed by LHS. The various case studies discussed in Section 6 show that the class of linear hybrid systems is a powerful enough framework for many applications. Hybrid systems with non-linear dynamics can also be appropriately abstracted in this framework by splitting discrete locations into multiple ones with dynamics that conform to LHS. In addition, the model allows abstraction of non-linear features of the continuous dynamics by over-approximation using uncertain, bounded input in each location as described below.

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. Therefore, $\emptyset \in \mathcal{C}_n$ represents the predicate `true`.

Definition 2.1 *Linear Hybrid Systems.* An n-dimensional **linear hybrid system (LHS)** 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 linear hybrid system 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 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, $\mathcal{I}_l := \{x \in \mathcal{X} \mid \forall \pi \in I(l) : \pi(x) = 1\}$.

—$f : L \to (\mathcal{X} \times \mathbb{R}^m \to \mathbb{R}^n)$ assigns to each location $l \in L$ a **continuous vector field** $f(l)$ on the continuous state $x \in \mathcal{X}$ given an input $u \in \mathbb{R}^m$. While at location l the evolution of the continuous variable is governed by the differential equation $\dot{x} = f(l)(x, u)$. We restrict our attention to hybrid systems with linear continuous dynamics and uncertain, bounded input, that is, for every location $l \in L$, the vector field $f(l)$ is linear, i.e. $f(l)(x, u) = A_l x + B_l u$ where A_l is an $n \times n$ matrix, B_l is an $n \times m$ matrix, and the input $u \in \mathcal{U}$ where $\mathcal{U}$ consists of piecewise continuous functions of the form $u : \mathbb{R}_{\geq 0} \to U$ such that $U \subset \mathbb{R}^m$ is a bounded convex set. We assume that the function $f(l)$ is globally Lipschitz in

x and continuous in u. This assumption guarantees existence and uniqueness of the solution of the differential equation.

—$T \subseteq L \times L \times \mathcal{C}_n \times (\Sigma_n)^n$ 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 linear hybrid system can jump to location l' at which the continuous variable x is reset to a new value $r(x)$. We will 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, that is, $\mathcal{G}_t := \{x \in \mathcal{I}_l \,|\, \forall \pi \in g : \pi(x) = 1\}$.

We illustrate the definition of a linear hybrid system using a simple thermostat model given in Figure 1. The thermostat model consists of three locations, that is $L = \{\texttt{Heat, Cool, Check}\}$. It contains two continuous variables, namely a clock $t \in \mathbb{R}_{\geq 0}$ and a temperature $T \in \mathbb{R}_{\geq 0}$. In this particular example we can limit the continuous state-space such that both the clock t and the temperature T are within the interval $[0, 100]$ without loss of accuracy of our analysis. The continuous state thus is $(t, T) \in [0, 100]^2$.[1] We write $(\texttt{Heat}, (2, 8))$ to denote the state $t = 2 \wedge T = 8$ while in location `Heat`. The continuous dynamics of the clock t is $\dot{t} = 1$ in all locations. The thermostat is switched on in the `Heat` location, so that the temperature increases by $\dot{T} = 2$. The invariant in the `Heat` location is $T \leq 10 \wedge t \leq 3$, that is, the system cannot remain in this location when the temperature exceeds ten and the clock exceeds three time-units. The control can switch to the `Cool` location, which models that the thermostat is switched off, when the guard $T \geq 9$ is enabled. This means, the switch from `Heat` to `Cool` can happen non-deterministically at any time when the temperature T is in the interval $[9, 10]$. The control remains in the `Cool` location, until the temperature is in the interval $[5, 6]$, when it switches back to the `Heat` location. This transition has a reset, which resets the clock $t := 0$. The third location, `Check`, models a self-checking mode of the thermostat controller. The invariant in the `Check` location guarantees that the control will return to the `Heat` location after at most one time-unit. During this time, the temperature drops, but this happens slower than in the `Cool` location. We assume that initially the thermostat is in its `Heat` location with $t = 0$ and $5 \leq T \leq 10$. This example is used throughout this paper to illustrate some of the concepts defined.

2.2 Transition System Semantics and Verification Problem

We define the semantics of a linear hybrid system by formalizing its underlying transition system. Assume an admissible set $\mathcal{U}$ of input functions $\mu : \mathbb{R}_{\geq 0} \to U$. We can then denote the flow of the system $\dot{x}(t) = A_l x(t) + B_l \mu(t)$ in location $l \in L$ as $\Phi_l(x, t, \mu)$ for an input function $\mu \in \mathcal{U}$ with initial condition $\Phi_l(x, 0, \mu) = x$.

The underlying transition system of a hybrid system 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

[1] For stylistic purposes we sometimes use $(t, T) \in \mathbb{R}^2_{\geq 0}$ instead of $(t, T) \in [0, 100]^2$.

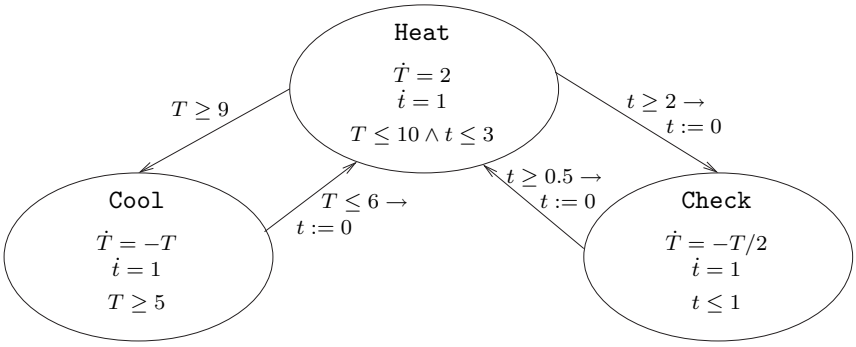

Fig. 1. A simple hybrid system model of a thermostat

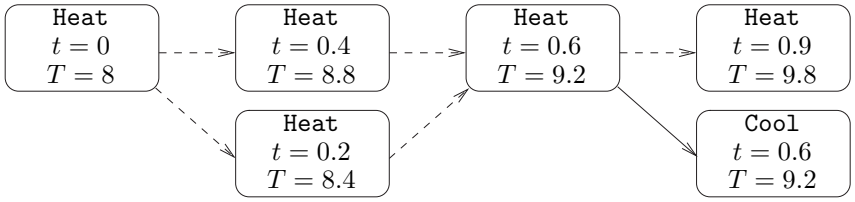

Fig. 2. Some traces of the thermostat model

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

$$(l, x) \rightarrow_C (l, y) \;:\Leftrightarrow\; \exists t \in \mathbb{R}_{\geq 0}, \mu \in \mathcal{U} : \Phi_l(x, t, \mu) = y \wedge \forall t' \in [0, t] : \Phi_l(x, t', \mu) \in \mathcal{I}_l.$$
$$(l, x) \rightarrow_D (l', y) \;:\Leftrightarrow\; \exists (l, l', g, r) \in T : x \in \mathcal{G}_t \wedge y = r(x) \wedge y \in \mathcal{I}_{l'}.$$

Figure 2 illustrates transitions between states in the underlying transition system T_H for the thermostat model (see Figure 1). From an initial state $t = 0 \wedge T = 8$ while in location `Heat`, we can perform infinitely many different continuous transitions, two of which are shown. Transitions in the figure with a dashed arrow denote transitions due to continuous flow, while a solid arrow implies a transition due to a discrete switch. Figure 2 shows a discrete switch from the state $t = 0.6 \wedge T = 9.2$ in location `Heat` to location `Cool` with the same continuous state. A *trace* of a hybrid system is a sequence of states starting in an initial state, such that there exists a transition in the underlying transition system between each consecutive pair of states.

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 $\texttt{Post}_C(l, P)$, and the continuous successors of a set of states $S \subseteq X$ denoted by $\texttt{Post}_C(S)$ as:

$$\texttt{Post}_C(l, P) \;:=\; \{(l, y) \in X \mid \exists x \in P : (l, x) \rightarrow_C (l, y)\};$$
$$\texttt{Post}_C(S) \;:=\; \{(l, y) \in X \mid \exists (l, x) \in S : (l, x) \rightarrow_C (l, y)\}.$$

Similarly, we define the set of *discrete successors* of (l, P) and S, denoted by

$\texttt{Post}_D(l, P)$ and $\texttt{Post}_D(S)$ respectively, as:

$$\begin{aligned}\texttt{Post}_D(l,P) &:= \{(l',y) \in X \mid \exists x \in P : (l,x) \rightarrow_D (l',y)\}.\\ \texttt{Post}_D(S) &:= \{(l',y) \in X \mid \exists (l,x) \in S : (l,x) \rightarrow_D (l',y)\}.\end{aligned}$$

For the thermostat example (see Figure 1), and a set $S = \{(\texttt{Heat}, (t,T)) \in X \mid 1.5 \leq t \leq 2.5 \wedge 8.5 \leq T \leq 9.5\}$, we have

$$\begin{aligned}\texttt{Post}_D(S) = \; &\{(\texttt{Cool}, (t,T)) \in X \mid 1.5 \leq t \leq 2.5 \wedge 9 \leq T \leq 9.5\} \cup \\ &\{(\texttt{Check}, (t,T)) \in X \mid t = 0 \wedge 8.5 \leq T \leq 9.5\},\end{aligned}$$

$$\begin{aligned}\texttt{Post}_C(S) = \{(\texttt{Heat}, (t,T)) \in X \mid \; &1.5 \leq t \leq 3 \wedge 8.5 \leq T \leq 10 \wedge \\ &2(t-2.5)+8.5 \leq T \leq 2(t-1.5)+9.5\}.\end{aligned}$$

Given a hybrid system H we want to verify certain safety properties. We define a property by specifying a set of *unsafe locations* $L_u \subseteq L$ and a convex set $\mathcal{B} \subseteq \mathcal{X}$ of *unsafe 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. For our thermostat example, we will define the set of unsafe states $\mathcal{B}$ as the set of states when the temperature drops below 4.5, that is:

$$\mathcal{B} = \{(t,T) \in (\mathbb{R}_{\geq 0})^2 \mid T \leq 4.5\}.$$

We define the set of unsafe locations $L_u = \{\texttt{Check}\}$, as the invariant in location `Cool` provides that we cannot reach $\mathcal{B}$ in the `Cool` location. We also do not include the location `Heat` into L_u, as the dynamics provide that $\mathcal{B}$ will not be reached while in the `Heat` location unless we are initially in $\mathcal{B}$.

Definition 2.2 *Verification problem.* Given a hybrid system $H = (\mathcal{X}, L, X_0, I, f, T)$, the set of reachable states $\texttt{Reach} \subseteq X$ is defined as

—$\texttt{Reach}^{(0)} := X_0 \cap \{(l,x) \in X \mid x \in \mathcal{I}_l\}$;
—$\texttt{Reach}^{(i+1)} := \texttt{Post}_C(\texttt{Reach}^{(i)}) \cup \texttt{Post}_D(\texttt{Reach}^{(i)}) \; \forall i \geq 0$; and
—$\texttt{Reach} := \bigcup_{i=0}^{\infty} \texttt{Reach}^{(i)}$.

Given a set of *unsafe locations* $L_u \subseteq L$ and a convex set $\mathcal{B} \subseteq \mathcal{X}$, we can define $\mathcal{B}_X := \{(l,x) \in X \mid l \in L_u \wedge x \in \mathcal{B}\}$. The **verification problem** then is:

$$\texttt{Reach} \cap \mathcal{B}_X \stackrel{?}{=} \emptyset.$$

In [Alur et al. 1995], it was shown that the verification problem for general hybrid systems is undecidable. In many practical situations though, model checking of hybrid systems can be used to verify certain properties of systems or to discover bugs in implementations. We now prove a property of the $\texttt{Reach}^{(i)}$ sets that will be used in later proofs.

LEMMA 2.3. *Given a hybrid system $H = (\mathcal{X}, L, X_0, I, f, T)$, the following holds $\forall i \in \mathbb{N}$:*

$$\texttt{Reach}^{(i)} \subseteq \texttt{Reach}^{(i+1)}.$$

PROOF. We first prove by induction, that $\forall i \in \mathbb{N}$

$$\texttt{Reach}^{(i)} \subseteq \{(l, x) \in X \,|\, x \in \mathcal{I}_l\} :$$

The statement is true for $\texttt{Reach}^{(0)}$ by definition. Now assume it holds for $\texttt{Reach}^{(i)}$. Then, for each state $(l, x) \in \texttt{Post}_C(\texttt{Reach}^{(i)})$, we have $(l, x) \in \{(l, x) \in X \,|\, x \in \mathcal{I}_l\}$ per definition of $\rightarrow_C$. Analogously, it holds for $\texttt{Post}_D(\texttt{Reach}^{(i)})$, which in turn means that the statement holds for $\texttt{Reach}^{(i+1)}$.

Given the fact that $\rightarrow_C$ is reflexive for states $(l, x) \in X$ where $x \in \mathcal{I}_l$, we have $\forall i \in \mathbb{N}$

$$\texttt{Reach}^{(i)} \subseteq \texttt{Post}_C(\texttt{Reach}^{(i)}),$$

which proves our lemma. □

2.3 Discrete Abstraction

We define a discrete abstraction of the hybrid system $H = (\mathcal{X}, L, X_0, I, f, T)$ with respect to a given k-dimensional vector of n-dimensional linear predicates $\Pi = (\pi_1, \pi_2, \ldots, \pi_k) \in (\mathcal{L}_n)^k$. We can partition the continuous state-space $\mathcal{X} \subseteq \mathbb{R}^n$ into at most 2^k states, corresponding to the 2^k possible Boolean truth 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 2.4 *Abstract state-space.* Given an n-dimensional hybrid system $H = (\mathcal{X}, L, X_0, f, I, T)$ and a k-dimensional vector $\Pi \in (\mathcal{L}_n)^k$ of n-dimensional linear predicates an **abstract state** is defined 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 therefore is $Q_\Pi := L \times \mathbb{B}^k$.

Figure 3 illustrates the abstraction of the continuous state-space for the thermostat example of Figure 1. We use ten predicates for the abstraction, namely:

$$\Pi = (t \leq 0, t \geq 0.5, t \leq 1, t \geq 2, t \leq 3, T \leq 4.5, T \geq 5, T \leq 6, T \geq 9, T \leq 10). \quad (1)$$

For the sake of simplicity these predicates all involve only one continuous variable, that is they correspond to hyperplanes parallel to some axis, though this is not necessary. The abstract continuous state-space consists of 36 non-empty states, which means that the size of the relevant abstract state-space Q_Π is $3 \cdot 36 = 108$.

For each vector $\boldsymbol{b} \in \mathbb{B}^k$ for a vector of linear predicates Π we can compute the set of states of the continuous state-space that it represents given the following definition. For example, the vector $(0, 1, 0, 1, 1, 0, 1, 0, 0, 1)$ represents the set $\{(t, T) \in \mathbb{R}^2 \,|\, 2 \leq t \leq 3 \wedge 6 < T < 9\}$ given the vector of predicates Π as specified in Equation (1).

Definition 2.5 *Concretization function.* We define a **concretization function** $C_\Pi : \mathbb{B}^k \rightarrow 2^{\mathbb{R}^n}$ for a vector of linear predicates $\Pi = (\pi_1, \ldots, \pi_k) \in (\mathcal{L}_n)^k$ as follows:

$$C_\Pi(\boldsymbol{b}) := \{x \in \mathbb{R}^n \,|\, \forall i \in \{1, \ldots, k\} : \pi_i(x) = b_i\}.$$

We denote a vector $\boldsymbol{b} \in \mathbb{B}^k$ as **consistent** with respect to a vector of linear predicates $\Pi \in (\mathcal{L}_n)^k$, iff $C_\Pi(\boldsymbol{b}) \neq \emptyset$. We say that an abstract state $(l, \boldsymbol{b}) \in Q_\Pi$ is

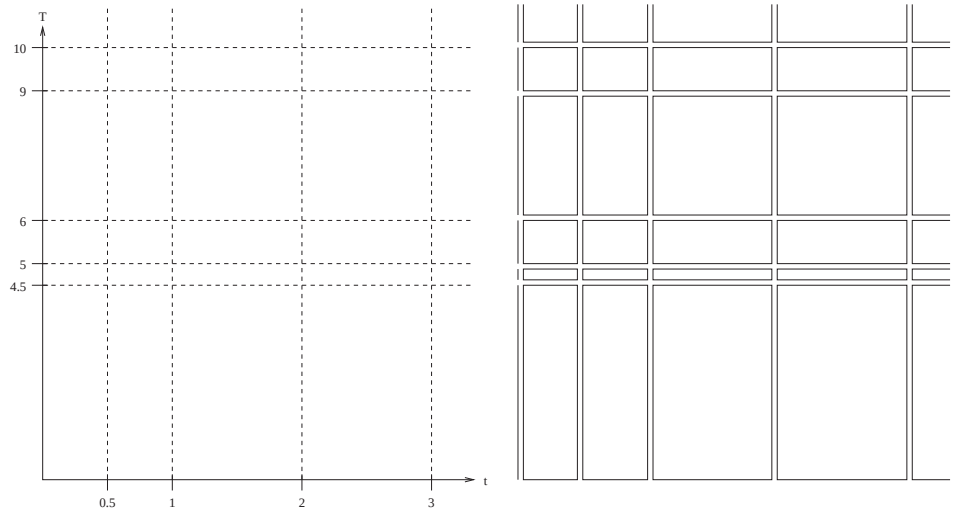

Fig. 3. Discrete abstraction of the continuous state-space for the thermostat model: Each box or line on the right hand side corresponds to a consistent vector $\boldsymbol{b} \in \mathbb{B}^{10}$ for the predicates as specified in equation (1).

consistent with respect to a vector of linear predicates Π, iff $\boldsymbol{b}$ is consistent with respect to Π.

As mentioned before, the set of abstract states has at most size $|L|2^k$ for k linear predicates. Often though the set of consistent abstract states is actually much smaller due to the fact that many predicates are redundant, that is they may be parallel, or do not cross inside the relevant continuous state-space $\mathcal{X}$. Figure 3 provides such an example. The abstract state-space consists only of 108 consistent abstract states, although there are $3 \cdot 2^{10} = 3072$ possible abstract states.

Our implementation is based on the fact that abstract states in the continuous state-space form a convex partition of the continuous state-space, which is formulated in the following lemma, and can be proven easily.

LEMMA 2.6. *Given a set of linear predicates $\Pi \in (\mathcal{L}_n)^k$ and a convex polyhedron $\mathcal{X}$, then for any $\boldsymbol{b} \in \mathbb{B}^k$ $C_\Pi(\boldsymbol{b})$ and $C_\Pi(\boldsymbol{b}) \cap \mathcal{X}$ represent convex polyhedra.*

Definition 2.7 *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 Π is defined as the transition system $H_\Pi = (Q_\Pi, \overset{\Pi}{\rightarrow}, Q_0)$ where

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

transitions due to continuous flows:

$$(l,\boldsymbol{b})\xrightarrow{\Pi}_D(l',\boldsymbol{b'}) :\Leftrightarrow \exists t=(l,l',g,r)\in T, x\in C_\Pi(\boldsymbol{b})\cap\mathcal{G}_t, y\in C_\Pi(\boldsymbol{b'})\cap\mathcal{I}_{l'}: y=r(x);$$

$$(l,\boldsymbol{b})\xrightarrow{\Pi}_C(l,\boldsymbol{b'}) :\Leftrightarrow \exists x\in C_\Pi(\boldsymbol{b}), t\in\mathbb{R}_{\geq 0}, \mu\in\mathcal{U}: \Phi_l(x,t,\mu)\in C_\Pi(\boldsymbol{b'})\wedge\forall t'\in[0,t]:\Phi_l(x,t',\mu)\in\mathcal{I}_l;$$

—the set of initial states is

$$Q_0=\{(l,\boldsymbol{b})\in Q_\Pi \mid \exists x\in C_\Pi(\boldsymbol{b})\cap\mathcal{I}_l:(l,x)\in X_0\}.$$

We can now define the successors of an abstract state $(l,\boldsymbol{b})\in Q_\Pi$ and a set of abstract states $S\subseteq Q_\Pi$ by discrete jumps and by continuous flows, denoted respectively by $\texttt{Post}_D(l,\boldsymbol{b})$, $\texttt{Post}_D(S)$, $\texttt{Post}_C(l,\boldsymbol{b})$, and $\texttt{Post}_C(S)$ as:

$$\begin{aligned}
\texttt{Post}_D(l,\boldsymbol{b}) &:= \{(l',\boldsymbol{b'})\in Q_\Pi \mid (l,\boldsymbol{b})\xrightarrow{\Pi}_D(l',\boldsymbol{b'})\},\\
\texttt{Post}_D(S) &:= \{(l',\boldsymbol{b'})\in Q_\Pi \mid \exists(l,\boldsymbol{b})\in S:(l,\boldsymbol{b})\xrightarrow{\Pi}_D(l',\boldsymbol{b'})\},\\
\texttt{Post}_C(l,\boldsymbol{b}) &:= \{(l,\boldsymbol{b'})\in Q_\Pi \mid (l,\boldsymbol{b})\xrightarrow{\Pi}_C(l,\boldsymbol{b'})\},\text{ and}\\
\texttt{Post}_C(S) &:= \{(l,\boldsymbol{b'})\in Q_\Pi \mid \exists(l,\boldsymbol{b})\in S:(l,\boldsymbol{b})\xrightarrow{\Pi}_C(l,\boldsymbol{b'})\}.
\end{aligned}$$

For our thermostat example, consider the abstract state $1<t<2\wedge 9\leq T\leq 10$ while in location `Heat`, which is represented by the abstract state $(l,\boldsymbol{b})=(\texttt{Heat},(0,1,0,0,1,0,1,0,1,1))$ given Π as specified in Equation (1). Then we have:

$$\texttt{Post}_D(l,\boldsymbol{b})=\{(\texttt{Cool},\boldsymbol{b})\},\text{ and}$$

$$\texttt{Post}_C(l,\boldsymbol{b})=\{(l,\boldsymbol{b}),(l,(0,1,0,1,1,0,1,0,1,1))\},$$

where $(0,1,0,1,1,0,1,0,1,1)$ represents $2\leq t\leq 3\wedge 9\leq T\leq 10$.

The verification problem in the abstract state-space can then be stated as described in the following definition:

Definition 2.8 *Abstract verification problem.* Given a hybrid system $H=(\mathcal{X},L,X_0,I,f,T)$ and a vector of linear predicates Π, we can define the set of reachable abstract states $\texttt{Reach}_\Pi$ as:

—$\texttt{Reach}_\Pi^{(0)}:=Q_0$;

—$\texttt{Reach}_\Pi^{(i+1)}:=\texttt{Post}_D(\texttt{Reach}_\Pi^{(i)})\cup\texttt{Post}_C(\texttt{Reach}_\Pi^{(i)})\,\forall i\geq 0$; and

—$\texttt{Reach}_\Pi:=\bigcup_{i\geq 0}\texttt{Reach}_\Pi^{(i)}$.

Given a set of unsafe locations $L_u\subseteq L$ and a convex set $\mathcal{B}\subseteq\mathcal{X}$, we can define $\mathcal{B}_\Pi:=\{(l,\boldsymbol{b})\in Q_\Pi \mid l\in L_u\wedge C_\Pi(\boldsymbol{b})\cap\mathcal{B}\neq\emptyset\}$. The verification problem then is:

$$\texttt{Reach}_\Pi\cap\mathcal{B}_\Pi\stackrel{?}{=}\emptyset.$$

Lemma 2.9. *Given a hybrid system $H=(\mathcal{X},L,X_0,I,f,T)$ and a vector of linear predicates Π, the following holds $\forall i\in\mathbb{N}$:*

$$\texttt{Reach}_\Pi^{(i)}\subseteq\texttt{Reach}_\Pi^{(i+1)}.$$

PROOF. We first prove by induction that $\forall i \in \mathbb{N}$

$$\forall (l, \boldsymbol{b}) \in \mathtt{Reach}_{\Pi}^{(i)} : C_{\Pi}(\boldsymbol{b}) \cap \mathcal{I}_l \neq \emptyset :$$

The statement is true for $\mathtt{Reach}_{\Pi}^{(0)}$ by definition of Q_0. Now assume it is true for $\mathtt{Reach}_{\Pi}^{(i)}$. Then, for each state $(l, \boldsymbol{b}) \in \mathtt{Post}_C(\mathtt{Reach}_{\Pi}^{(i)})$, we have $C_{\Pi}(\boldsymbol{b}) \cap \mathcal{I}_l \neq \emptyset$ by definition of $\xrightarrow{\Pi}_C$. Analogously, it holds for $\mathtt{Post}_D(\mathtt{Reach}_{\Pi}^{(i)})$, which in turn means that the statement holds for $\mathtt{Reach}_{\Pi}^{(i+1)}$.

Given the fact that $\xrightarrow{\Pi}_C$ is reflexive for all states $(l, \boldsymbol{b}) \in Q_{\Pi}$ where $C_{\Pi}(\boldsymbol{b}) \cap \mathcal{I}_l \neq \emptyset$, we have $\forall i \in \mathbb{N}$

$$\mathtt{Reach}_{\Pi}^{(i)} \subseteq \mathtt{Post}_C(\mathtt{Reach}_{\Pi}^{(i)}),$$

which proves our lemma. □

We can now prove that predicate abstraction of hybrid systems computes an over-approximation of the set of reachable states of the concrete system. This is formalized in the following lemma:

LEMMA 2.10. *Given a hybrid system $H = (\mathcal{X}, L, X_0, I, f, T)$ and a vector of linear predicates Π, the following holds:*

$$\mathtt{Reach} \subseteq \{(l, x) \in X \mid \exists (l, \boldsymbol{b}) \in \mathtt{Reach}_{\Pi} : x \in C_{\Pi}(\boldsymbol{b}) \cap \mathcal{I}_l\}.$$

PROOF. The proof of Lemma 2.3 already guarantees that for any state $(l, x) \in \mathtt{Reach} : x \in \mathcal{I}_l$. We only need to prove the following statement, which we will prove by induction:

$$\forall i \in \mathbb{N} : \mathtt{Reach}^{(i)} \subseteq \{(l, x) \in X \mid \exists (l, \boldsymbol{b}) \in \mathtt{Reach}_{\Pi}^{(i)} : x \in C_{\Pi}(\boldsymbol{b})\}.$$

—The statement holds for $i = 0$ by definition of $\mathtt{Reach}_{\Pi}^{(0)}$.

—Assume, that the statement holds for $i \in \mathbb{N}$. Then, given Lemma 2.3, we only need to show that:

$$\forall (l, x) \in \mathtt{Reach}^{(i+1)} \setminus \mathtt{Reach}^{(i)} \; \exists (l, \boldsymbol{b}) \in \mathtt{Reach}_{\Pi}^{(i+1)} : x \in C_{\Pi}(\boldsymbol{b}).$$

We consider two cases independently. The first case covers the scenario that the state $(l, x) \in \mathtt{Reach}^{(i+1)} \setminus \mathtt{Reach}^{(i)}$ was produced by a state $(l_i, x_i) \in \mathtt{Reach}^{(i)}$ through a discrete transition, that is $(l_i, x_i) \rightarrow_D (l, x)$. The induction hypothesis guarantees that there exists a state $(l_i, \boldsymbol{b_i}) \in \mathtt{Reach}_{\Pi}^{(i)}$ such that $x_i \in C_{\Pi}(\boldsymbol{b_i})$. It is clear then, that there also exists a transition $(l_i, \boldsymbol{b_i}) \xrightarrow{\Pi}_D (l, \boldsymbol{b})$ such that $x \in C_{\Pi}(\boldsymbol{b})$, which proves this case. Analogously, we can prove the other case for a transition $(l_i, x_i) \rightarrow_C (l, x)$, which completes the proof.

□

3. REACHABILITY ANALYSIS

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. To compute continuous successors of an abstract state A, we use a strategy inspired by the techniques used in CheckMate [Chutinan and Krogh 1999] and d/dt [Asarin et al. 2000]. In this section we describe the computation of the abstract transitions in more detail. Additionally, we describe a possible search strategy in the abstract state-space, and prove soundness of the algorithm.

For the reachability analysis it is generally a good initial guess to include all guards and invariants of a hybrid automaton H 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. We discuss the choice of abstraction predicates in detail in [Alur et al. 2003a], and assume here that Π is user-specified.

3.1 Computing Abstract Discrete Successors

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

We define a tri-valued logic using the symbols $\mathbb{T} := \{0, 1, *\}$. 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 $t^\Pi_{\mathcal{X}} : \mathbb{B}^k \times \mathcal{L}_n \to \mathbb{T}$ formally as:

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

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, we define the number of positions in a k-dimensional vector $\boldsymbol{t} \in \mathbb{T}^k$ with element $*$ as $||\boldsymbol{t}||_*$.

Given a particular transition $(l, l', g, r) \in T$ and a 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. If we generalize this for a vector of predicates $\Pi = (\pi_1, \ldots, \pi_k) \in (\mathcal{L}_n)^k$ by $\Pi \circ r := (\pi_1 \circ r, \ldots, \pi_k \circ r)$, the following lemma immediately follows.

Lemma 3.1. *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 $(l, l', g, r) \in T$ 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 $\boldsymbol{b'} \in \mathbb{T}^n$ and each component b'_i is given by: $b'_i = t^\Pi_{\mathcal{X}}(\boldsymbol{b}, \pi_i \circ r)$. If $b'_i = 1$, then we know that the corresponding linear predicate $\pi_i \circ r$ is true for all points $x \in C_\Pi(\boldsymbol{b}) \cap \mathcal{X}$. This means that all states in $C_\Pi(\boldsymbol{b}) \cap \mathcal{X}$ 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 either

$C_\Pi(\boldsymbol{b}) \cap \mathcal{X} = \emptyset$ or there exist concrete continuous states in $C_\Pi(\boldsymbol{b}) \cap \mathcal{X}$ 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 $\boldsymbol{b'} \in \mathbb{T}^n$ represents $2^{||\boldsymbol{b'}||_*}$ many possibilities, which combined with location l' make up at most[2] $2^{||\boldsymbol{b'}||_*}$ many abstract states. We additionally define $c : \mathbb{T}^k \to 2^{\mathbb{B}^k}$ 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 $(l', \boldsymbol{e}) \in Q_\Pi$ 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, and the reset hits the invariant of the next location l', which is formulated in the following theorem.

THEOREM 3.2. *Given an abstract state $(l, \boldsymbol{b}) \in Q_\Pi$ with respect to a k-dimensional vector of n-dimensional linear predicates Π, a transition $(l, l', g, r) \in T$ and the corresponding guard set $\mathcal{G}_t$, we have $\forall \boldsymbol{v} \in \mathbb{B}^k$:*

$$C_{\Pi \circ r}(\boldsymbol{v}) \cap C_\Pi(\boldsymbol{b}) \cap \mathcal{G}_t \cap \mathtt{Pre}_r(\mathcal{I}_{l'}) \neq \emptyset \Leftrightarrow (l, \boldsymbol{b}) \xrightarrow{\Pi}_D (l', \boldsymbol{v}),$$

where $\mathtt{Pre}_r : 2^{\mathbb{R}^n} \to 2^{\mathbb{R}^n}$ with $\mathtt{Pre}_r(P) = \{x \in \mathbb{R}^n \mid r(x) \in P\}$.

PROOF. If $C_{\Pi \circ r}(\boldsymbol{v}) \cap C_\Pi(\boldsymbol{b}) \cap \mathcal{G}_t \cap \mathtt{Pre}_r(\mathcal{I}_{l'})$ is not empty, we can pick a point $x \in C_{\Pi \circ r}(\boldsymbol{v}) \cap C_\Pi(\boldsymbol{b}) \cap \mathcal{G}_t \cap \mathtt{Pre}_r(\mathcal{I}_{l'})$. As $x \in \mathcal{G}_t \cap \mathtt{Pre}_r(\mathcal{I}_{l'})$, we found a discrete transition in the concrete state-space $(l, x) \to_D (l', r(x))$, as we know that $r(x) \in \mathcal{I}_{l'}$. Additionally, we know that $x \in C_\Pi(\boldsymbol{b})$ and that $x \in C_{\Pi \circ r}(\boldsymbol{v})$. By using Lemma 3.1 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, we have $(l, \boldsymbol{b}) \xrightarrow{\Pi}_D (l', \boldsymbol{v})$ for some discrete transition $(l, l', g, r) \in T$, we must have: $\exists x \in C_\Pi(\boldsymbol{b}) : x \in \mathcal{G}_t \wedge r(x) \in C_\Pi(\boldsymbol{v}) \wedge r(x) \in \mathcal{I}_{l'}$. Using Lemma 3.1, this means that $\exists x \in C_\Pi(\boldsymbol{b}) : x \in \mathcal{G}_t \wedge x \in C_{\Pi \circ r}(\boldsymbol{v}) \wedge x \in \mathtt{Pre}_r(\mathcal{I}_{l'})$. Hence we found that $C_{\Pi \circ r}(\boldsymbol{v}) \cap C_\Pi(\boldsymbol{b}) \cap \mathcal{G}_t \cap \mathtt{Pre}_r(\mathcal{I}_{l'}) \neq \emptyset$. □

The computation of $\mathtt{Pre}_r(\mathcal{I}_{l'})$ can be performed using Lemma 3.1, as we can also write it as $C_{I(l') \circ r}(1, \ldots, 1)$ assuming $I(l')$ represents a vector of linear predicates rather than a set. If we assume that all the linear predicates of the guard $g \in \mathcal{C}_n$ and the invariants $I(l)$ and $I(l')$ 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 $\mathcal{G}_t$ and the set $\mathtt{Pre}_r(\mathcal{I}_{l'})$. 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 and the following lemma:

LEMMA 3.3. *Given an abstract state $(l, \boldsymbol{b}) \in Q_\Pi$ with respect to a k-dimensional vector of n-dimensional linear predicates $\Pi = (\pi_1, \ldots, \pi_k)$, assume that $\boldsymbol{b'} \in \mathbb{T}^k$ such that each component b'_i is given by $b'_i = t^\Pi_{\mathcal{X}}(\boldsymbol{b}, \pi_i \circ r)$ for a transition $t = (l, l', g, r) \in T$. Then the following statement holds:*

$$(l, \boldsymbol{b}) \xrightarrow{\Pi}_D (l', \boldsymbol{v}) \Rightarrow \boldsymbol{v} \in c(\boldsymbol{b'}).$$

[2]Note, that $C_\Pi(\boldsymbol{b}) \cap \mathcal{X}$ may be empty for some $\boldsymbol{b}$, and is hence meaningless.

PROOF. Assume the contrary. Then, there has to be a component index $1 \leq i \leq k$, such that $t^{\Pi}_{\mathcal{X}}(\boldsymbol{b}, \pi_i \circ r) \neq *$ and $v_i \neq t^{\Pi}_{\mathcal{X}}(\boldsymbol{b}, \pi_i \circ r)$. We consider only the case that $v_i = 0 \wedge t^{\Pi}_{\mathcal{X}}(\boldsymbol{b}, \pi_i \circ r) = 1$, since the other case is analogous. The fact that $t^{\Pi}_{\mathcal{X}}(\boldsymbol{b}, \pi_i \circ r) = 1$ means that $\forall x \in C_{\Pi}(\boldsymbol{b}) \cap \mathcal{X} : \pi_i \circ r(x) = 1$. Therefore, there is no state in $C_{\Pi}(\boldsymbol{b}) \cap \mathcal{X}$ that makes $\pi_i \circ r$ false, which implies that no state in $C_{\Pi}(\boldsymbol{b}) \cap \mathcal{G}_t$ makes $\pi_i \circ r$ false. This contradicts that $v_i = 0$. □

Consider the thermostat example of Figure 1, and in particular the transition from the `Cool` to the `Heat` location. Assume that we are interested in computing the discrete successors of the abstract state $(l, \boldsymbol{b}) = (\texttt{Cool}, (0,1,1,0,1,0,1,1,0,1))$, which represents the continuous state-space $0.5 \leq t \leq 1 \wedge 5 \leq T \leq 6$ given the vector of predicates Π as specified in Equation (1). The guard for this transition is $T \leq 6$, while the reset is $t := 0$ and $T := T$. Therefore, we have $\Pi \circ r$ as

$$\Pi \circ r = (0 \leq 0, 0 \geq 0.5, 0 \leq 1, 0 \geq 2, 0 \leq 3, T \leq 4.5, T \geq 5, T \leq 6, T \geq 9, T \leq 10).^3$$

We can now compute $\boldsymbol{b'} \in \mathbb{T}^{10}$ where each component b'_i is given by $b'_i = t^{\Pi}_{\mathcal{X}}(\boldsymbol{b}, \pi_i \circ r)$. In this example we have

$$\boldsymbol{b'} = (1,0,1,0,1,0,1,1,0,1) \in \mathbb{T}^{10}.$$

As $||\boldsymbol{b'}||_* = 0$, there is only one possible discrete successor for $(l, \boldsymbol{b})$, namely $(\texttt{Heat}, (1,0,1,0,1,0,1,1,0,1))$ representing $t \leq 0 \wedge 5 \leq T \leq 6$ in the continuous state-space. We now use Theorem 3.2 to check whether $(l', \boldsymbol{v}) = (\texttt{Heat}, (1,0,1,0,1,0,1,1,0,1))$ is indeed a valid successor of $(l, \boldsymbol{b})$. We have

$$\begin{aligned} C_{\Pi \circ r}(\boldsymbol{v}) &= \{(t,T) \in \mathbb{R}^2 \mid 5 \leq T \leq 6\}, \\ C_{\Pi}(\boldsymbol{b}) &= \{(t,T) \in \mathbb{R}^2 \mid 0.5 \leq t \leq 1 \wedge 5 \leq T \leq 6\}, \\ \mathcal{G}_t &= \{(t,T) \in [0,100]^2 \mid 5 \leq T \leq 6\}, \text{ and} \\ \texttt{Pre}_r(\mathcal{I}_{l'}) &= \{(t,T) \in [0,100]^2 \mid T \leq 10\}. \end{aligned}$$

Therefore, we have

$$C_{\Pi \circ r}(\boldsymbol{v}) \cap C_{\Pi}(\boldsymbol{b}) \cap \mathcal{G}_t \cap \texttt{Pre}_r(\mathcal{I}_{l'}) = \{(t,T) \in [0,100]^2 \mid 0.5 \leq t \leq 1 \wedge 5 \leq T \leq 6\} \neq \emptyset,$$

which implies that indeed $(l, \boldsymbol{b}) \xrightarrow{\Pi}_D (l', \boldsymbol{v})$.

3.2 Computing Abstract Continuous Successors

To compute 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 the convex-hull of all these polyhedra to over-approximate the set of all states reachable from A. We are only interested in testing if this set intersects with a new abstract state. This approach has many benefits compared to the traditional approach of computing approximations of reachable sets for hybrid systems. First, the expensive operation of computing continuous successors is applied only to sets of states that initially correspond to abstract states with a fixed number of faces corresponding to the number of predicates. This is in stark contrast to earlier approaches that needed to consider initial sets of states corresponding to polyhedra of unpredictable shapes and complexities due to successive

[3] Clearly, the first five linear predicates are equivalent to `true, false, true, false, true`.

computations of continuous and discrete successors. Second, as a heuristics, we can prematurely terminate the computation of continuous successors whenever a new abstract state is discovered. If a counterexample is discovered using this heuristics, we can claim to have found this particular counterexample faster than we would have been able to without prematurely stopping the continuous successor computation. However, we may need to return to the point where the computation has been prematurely terminated in case a counterexample could not be found using this particular prefix-trace. Finally, we can explore with different search strategies aimed at making progress in the abstract graph.

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

$$\texttt{Post}_C(l, C_\Pi(\boldsymbol{b}) \cap \mathcal{I}_l) \cap \{(l, x) \mid x \in C_\Pi(\boldsymbol{b'}) \cap \mathcal{I}_l\} \neq \emptyset, \tag{2}$$

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 $x \in C_\Pi(\boldsymbol{b}) \cap \mathcal{I}_l$ to a point $y \in C_\Pi(\boldsymbol{b}') \cap \mathcal{I}_l$. The test of the condition (2) 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` [Asarin et al. 2000]. 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. 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 $C = \text{conv}(P \cup P_r)$ 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 (2) 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

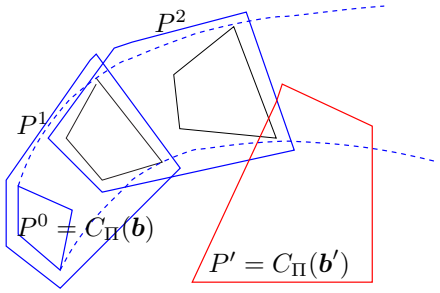

Fig. 4. Illustration of the computation of continuous successors. After two iterations the new abstract state $(l, \boldsymbol{b}')$ is reachable from $(l, \boldsymbol{b})$.

notation $\texttt{ApproxPost}_C^\Pi(l, P, [0, r])$ 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 $\texttt{ApproxPost}_C^\Pi$ 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 preceding 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 4.

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

$R_c \leftarrow \emptyset$;{stores reachable abstract states}
$P^0 \leftarrow C_\Pi(\boldsymbol{b}) \cap \mathcal{X}$;
$k \leftarrow 0$;
repeat
  $P^{k+1} \leftarrow \texttt{ApproxPost}_C^\Pi(l, P^k, [0, r]) \cap \mathcal{X}$;
  for all $(l, \boldsymbol{b}') \in Q_\Pi \setminus R_c$ **do**
    if $P^{k+1} \cap C_\Pi(\boldsymbol{b}') \neq \emptyset$ **then**
      $R_c := R_c \cup (l, \boldsymbol{b}')$;
    end if
  end for
  $k \leftarrow k + 1$;
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.

The algorithm terminates if the reachable set of the current iteration is included in that of the preceding iteration, or after some predetermined maximal number $K_{\max}$ of iterations. The value of $K_{\max}$ is determined by a high-level procedure which handles search order.

We give a brief illustration of the computation of continuous successors for the thermostat example of Figure 1. Assume the current abstract state is $(l, \boldsymbol{b}) = (\texttt{Cool}, (0,1,1,0,1,0,1,0,1,1))$ which represents $0.5 \leq t \leq 1 \wedge 9 \leq T \leq 10$ in the continuous state-space. The flow in the `Cool` location is specified by the differential equations $\dot{t} = 1$ and $\dot{T} = -T$. Assume that $P(0)$ represents the set $P(0) = \{(t,T) \in \mathbb{R}^2 \,|\, 0.5 \leq t \leq 1 \wedge 9 \leq T \leq 10\}$. Then we can compute the image of $P(0)$ after Δt time-units as

$$P(\Delta t) = \{(t,T) \in \mathbb{R}^2 \,|\, 0.5 + \Delta t \leq t \leq 1 + \Delta t \wedge 9 \cdot e^{-\Delta t} \leq T \leq 10 \cdot e^{-\Delta t}\}.$$

For a sample time step $\Delta t = 0.3$, we thus have

$$P(0.3) = \{(t,T) \in \mathbb{R}^2 \,|\, 0.8 \leq t \leq 1.3 \wedge 9 \cdot e^{-0.3} \leq T \leq 10 \cdot e^{-0.3}\},$$

where $9 \cdot e^{-0.3} \approx 6.667$ and $10 \cdot e^{-0.3} \approx 7.408$. We thus proved that $(l, \boldsymbol{b}) \xrightarrow{\Pi}_C (l, (0,1,1,0,1,0,1,0,0,1))$ and $(l, \boldsymbol{b}) \xrightarrow{\Pi}_C (l, (0,1,0,0,1,0,1,0,0,1))$, which represent $0.5 \leq t \leq 1 \wedge 6 < T < 9$ and $1 < t < 2 \wedge 6 < T < 9$ respectively in the continuous state-space. Note that we did not illustrate the computation of convex hulls for the computation of successor states due to continuous flow. For the two aforementioned abstract states it is enough though to compute the slice after 0.3 time-units to determine that they are indeed successors of $(l, \boldsymbol{b})$.

3.3 Searching the Abstract State-Space

We implemented an on-the-fly search of the abstract state-space, which means that we do not pre-compute the abstract transition first. If a counterexample exists in the abstract transition system, we may be able to discover it quickly by computing only a small fraction of the abstract transition system which often is the bottleneck for the verification.

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

We perform a DFS, which usually does not find a shortest counterexample 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 counterexample. 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 suspend or interrupt the computation when a new abstract state is found. We can then compute the successors of this abstract state and hope to explore quickly more of the reachable abstract state-space. Not running the fix-point computation of continuous successors to completion may result in a substantial speed-up when discovering a counterexample, if one exists. However, we may need to backtrack the computation to this point and continue the search for further reachable continuous successors if no counterexample has been found in the previously suspended computation. We store already visited abstract states in

a hash table as shown in Algorithm 2.

Algorithm 2 ABSTRACT STATE-SPACE REACHABILITY ANALYSIS VIA DFS

```
1: hashTable ← new HashTable () ; {stores already visited states}
2: while Q_0\hashTable ≠ ∅ do
3:   stack ← new Stack () ; {stores current search path}
4:   pick (l, b) ∈ Q_0\hashTable ;
5:   push (l, b) onto stack ; add (l, b) to hashTable ;
6:   repeat
7:     if stack.top().violatesProperty() then
8:       return stack ; {stack represents trace to violation of property}
9:     end if
10:    if Post_D(stack.top())\hashTable ≠ ∅ then {check discrete successors}
11:      pick (l, b) ∈ Post_D(stack.top())\hashTable ;
12:      push (l, b) onto stack ; add (l, b) to hashTable ;
13:    else if Post_C(stack.top())\hashTable ≠ ∅ then {check cont. successors}
14:      pick (l, b) ∈ Post_C(stack.top())\hashTable ;
15:      push (l, b) onto stack ; add (l, b) to hashTable ;
16:    else
17:      stack.pop() ; {this state is not on any path to a property violation}
18:    end if
19:  until stack.isEmpty() ;
20: end while
21: return "Property is guaranteed!" ;
```

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

THEOREM 3.4. *If Algorithm 2 terminates and reports that the abstract system is safe, then the corresponding concrete system is also safe.*

PROOF. We need to show that

$$\texttt{Reach}_\Pi \cap \mathcal{B}_\Pi = \emptyset \Rightarrow \texttt{Reach} \cap \mathcal{B}_X = \emptyset.$$

Lemma 2.10 shows the over-approximation of $\texttt{Reach}$ by $\texttt{Reach}_\Pi$. It is also clear, that $\mathcal{B}_X$ is over-approximated by $\mathcal{B}_\Pi$. Hence, if $\texttt{Reach}_\Pi \cap \mathcal{B}_\Pi$ is empty, so is $\texttt{Reach} \cap \mathcal{B}_X$. □

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

4. 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 of A, and check if this set intersects with each of the other

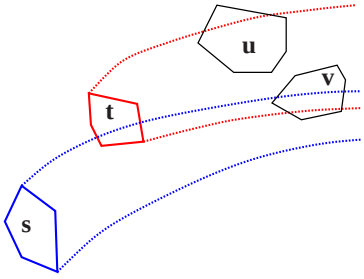

Fig. 5. An optimization technique in the search strategy

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 search in the abstract state-space.

4.1 Search Constraints

We include an optimization technique in the search strategy. Consider a real counterexample in the concrete hybrid system. There exists an equivalent counterexample that has the additional constraint that there are no two consecutive transitions due to continuous flow in the equivalent counterexample. 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 therefore searching only for counterexamples in the abstract system that do not have two consecutive transitions due to continuous flow. By enforcing this additional constraint we eliminate some spurious counterexamples that could have been found otherwise in the abstract transition system. The spurious counterexamples that are eliminated are due to the fact that $(l, \boldsymbol{b}) \stackrel{\Pi}{\rightarrow}_C (l, \boldsymbol{b'})$ and $(l, \boldsymbol{b'}) \stackrel{\Pi}{\rightarrow}_C (l, \boldsymbol{b''})$ does *not* imply that $(l, \boldsymbol{b}) \stackrel{\Pi}{\rightarrow}_C (l, \boldsymbol{b''})$. Therefore, we are in fact not following every possible path according to the relation $\stackrel{\Pi}{\rightarrow}$ as it is presented in Algorithm 2, but only a part of it without compromising the conservativeness of our approach. We illustrate this optimization technique in Figure 5. If the abstract state t can only be reached by transitions due to continuous flow, we will not explore its continuous successors by the same continuous dynamics. Therefore, in the example illustrated in Figure 5, the abstract state u will not be regarded as reachable. On the other hand, v will be reached by continuous flow from s. For an example of the use of this search constraint, we refer the reader to Section 4.4.1.

4.2 Binary Space Partition Tree

We now describe another optimization concerning 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 $\mathcal{X}$ by one

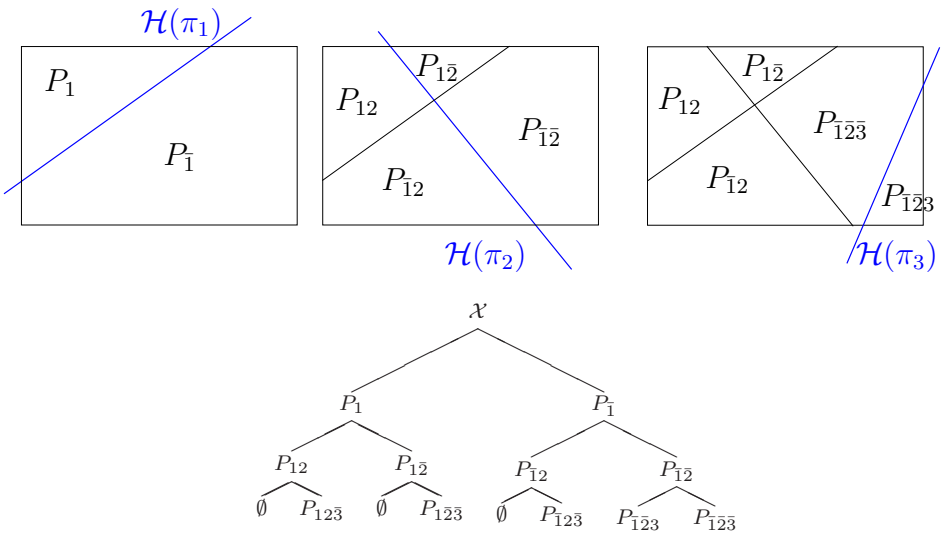

Fig. 6. BSP-based construction of the abstract state-space

predicate after another are stored in a BSP tree as follows. First, the root of the tree is associated with the whole set $\mathcal{X}$. We choose a predicate π_i 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 $\overline{\mathcal{H}(\pi_i)}$. We proceed to recursively partition the non-empty polyhedra 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 6 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. As we shall see in the next section, 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. As an example, the model of a vehicle coordination system discussed in section 6 has 17 initial predicates, and using this technique we found 785 consistent abstract states while the number of possible abstract states is 2^{17}.

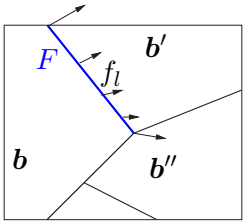

Fig. 7. Vector field analysis

4.3 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 each of 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.

Geometrically speaking, the $\mathcal{X}$-bounded concretizations $C_\Pi(\boldsymbol{b}) \cap \mathcal{X}$ for all $\boldsymbol{b} \in \mathbb{B}^k$ form a convex decomposition of the concrete state-space $\mathcal{X}$. Therefore, for any two non-empty abstract states $(l, \boldsymbol{b})$ and $(l, \boldsymbol{b}')$, the closures of their concretizations $cl(C_\Pi(\boldsymbol{b}) \cap \mathcal{X})$ and $cl(C_\Pi(\boldsymbol{b}') \cap \mathcal{X})$ 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 $C_\Pi(\boldsymbol{b}') \cap \mathcal{X}$ to $C_\Pi(\boldsymbol{b}) \cap \mathcal{X}$. If for all points on the face F the projection of f_l on $\boldsymbol{n}_F$ is non-negative, that is,

$$\forall x \in F \ \langle f(l)(x), \boldsymbol{n}_F \rangle \geq 0, \tag{3}$$

then there exists a trajectory by continuous dynamics $f(l)$ from $C_\Pi(\boldsymbol{b}') \cap \mathcal{X}$ to $C_\Pi(\boldsymbol{b}) \cap \mathcal{X}$. Moreover, any trajectory from $C_\Pi(\boldsymbol{b}) \cap \mathcal{X}$ to $C_\Pi(\boldsymbol{b}') \cap \mathcal{X}$ by $f(l)$, if one exists, must cross another polyhedron $C_\Pi(\boldsymbol{b}'') \cap \mathcal{X}$ (see Figure 7). 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 transitions via some other intermediate states[4]. Note that when the dynamics $f(l)$ is affine, in order for the Condition (3) to hold, it suffices that $\langle f(l)(x), \boldsymbol{n}_F \rangle$ is non-negative at all 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 $\mathcal{E} = \{x \,|\, x^T P x \leq \alpha\}$ that contains the polyhedron $C_\Pi(\boldsymbol{b}) \cap \mathcal{X}$. 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 $C_\Pi(\boldsymbol{b}') \cap \mathcal{X} \cap \mathcal{E} = \emptyset$ cannot be reached from $(l, \boldsymbol{b})$ by continuous

[4]It should be noted though that this technique of eliminating computations cannot be used in conjunction with the search strategy as described in Section 4.1.

dynamics $f(l)$.

4.4 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 search depends on which successor is examined next to continue the search at every step. If there exists a counterexample to a safety property in the concrete system, then we want to identify a corresponding abstract counterexample, which must exist, as fast as and as direct as possible.

We present three guided search strategies that we have 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 $\mathcal{B}_\Pi$. As we are trying to minimize the time it takes to discover a counterexample, we will 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

$$(l, l') \in E :\Leftrightarrow \exists (l, l', g, r) \in T.$$

Given the set of unsafe locations $L_u \subseteq L$, we define a priority function $\rho_D : L \to \mathbb{N}$ on locations as:

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

It is clear that $\forall l \in L : \rho_D(l) \neq \infty \Rightarrow 0 \leq \rho_D(l) \leq |L| - 1$. Also notice that we do not need to consider any location $l \in L$ with $\rho_D(l) = \infty$ during the reachability analysis, as there is no way to reach the set of unsafe states once we are in location l. Hence, eliminating those locations and all adjacent transitions does not impact the reachability result. We use ρ_D in all three guided search strategies that we introduce in the following.

For the thermostat example of Figure 1, we defined $L_u = \{\texttt{Check}\}$. Thus, we have $\rho_D(\texttt{Check}) = 0$, $\rho_D(\texttt{Heat}) = 1$, and $\rho_D(\texttt{Cool}) = 2$.

4.4.1 *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 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 $\delta : \mathbb{B} \times \mathbb{T} \to \mathbb{N}$ as

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

and a priority function $\rho_1 : \mathbb{B}^k \to \mathbb{N}$ as

$$\rho_1(\boldsymbol{b}) = \sum_{i=1}^{k} \delta(b_i, m_i).$$

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 $\boldsymbol{m}$.

One way of combining ρ_1 with ρ_D to form a priority function $\rho_M : Q_\Pi \to \mathbb{N}$ over abstract states is in the following manner:

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

The multiplication of $\rho_D(l)$ by $k+1$ guarantees that abstract states with a smaller distance to L_u in G_H are always preferred compared to abstract states where the location is "further away" from the unsafe locations.

We now go back to the thermostat example as described in Figure 1. Figure 8 illustrates our guided search including the search constraint optimization (see Section 4.1) using the predicates as defined in Equation (1). All 35 reachable abstract states are safe, which, following Theorem 3.4, means that the concrete hybrid system is also safe.

Figure 8 represents the graph of reachable abstract states. We included a running number for each abstract state, which corresponds to the chronological ordering of the reachable states found during the search of the abstract state-space. Dashed arrows correspond to transitions between abstract states that are taken due to continuous flow of time, whereas solid arrows represent transitions due to discrete jumps between locations. We are not trying to compute all possible arrows in this graph, instead we are trying to compute all reachable abstract states. Therefore, we only include arrows that represent the first time a new abstract state is found. However, there are two solid arrows (representing a transition due to a discrete jump) in the graph that are included although the destination abstract state has been reached before. These are the transitions between the abstract states with chronological number 15 and 10, and the transition from 12 to 22. These transitions are important, as they lead the search to a previously visited abstract search that had so far only been visited by an abstract transition due to continuous flow. Due to the aforementioned search constraint optimization, we have not explored the continuous successors of these two abstract states. As they have been reached by an abstract transition due to a discrete jump now, we have to compute the continuous successors of these abstract states.

In addition, for the stability of the computation, we thickened the bounded continuous interval for the timer t to $[-\varepsilon, 100]$ instead of $[0, 100]$ for a small $\varepsilon > 0$. This is due to the fact that the predicate $t \leq 0$, if true, would produce a lower-dimensional polyhedron, which tends to be numerically unstable. Therefore, the predicate $t \leq 0$ if true stands for $-\varepsilon \leq t \leq 0$ instead of $t = 0$. Thus the tool finds the transition from the abstract state with chronological number 5 to the abstract state 10 in Figure 8.

As mentioned, we also included a running number for each abstract state rep-

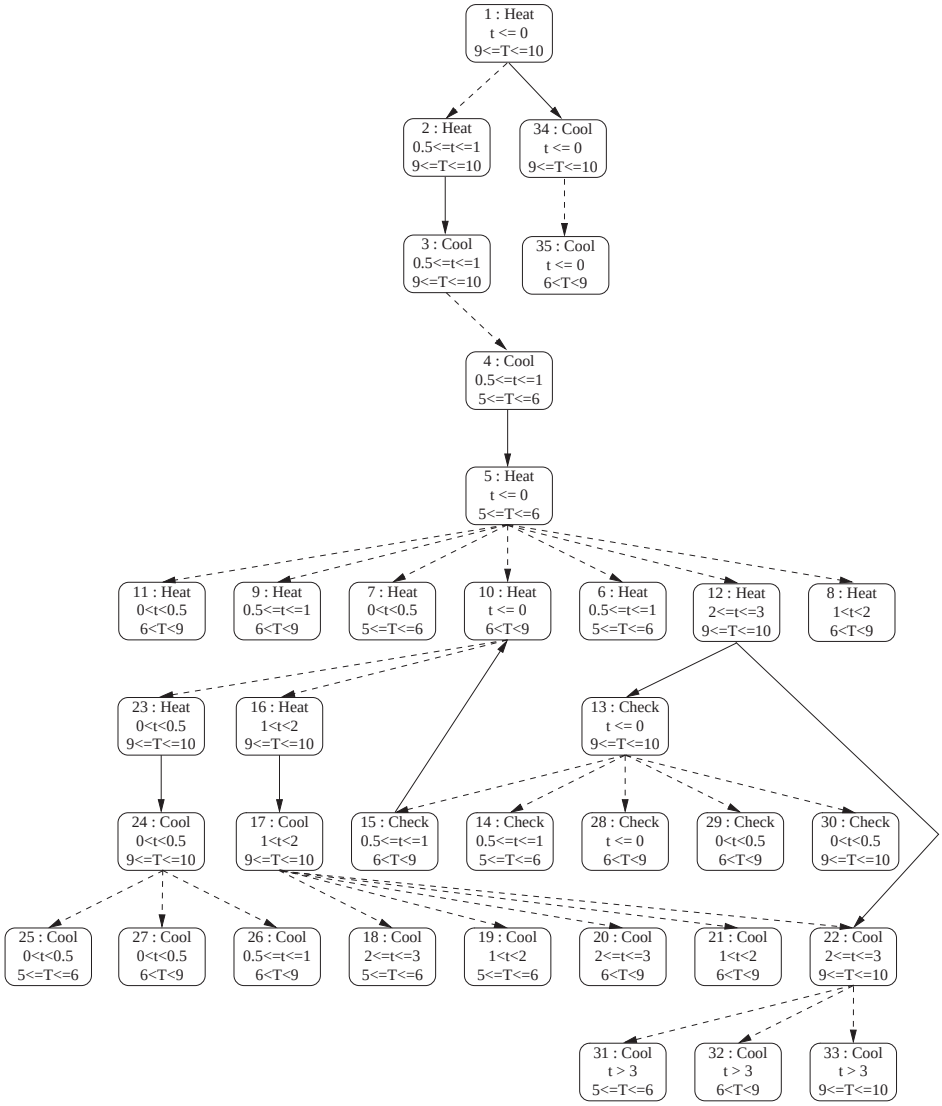

Fig. 8. The graph of reachable abstract states for the thermostat model

resenting the chronological order of choosing the state during the search using a mask priority guided search. It can be seen that transitions are preferred that reach abstract states with location `Check` and those transitions due to continuous flow that reach abstract states that are closer in the continuous state-space to the unsafe states. Therefore, abstract states where the temperature is in a lower interval are chosen first before other abstract states with the same location are considered. Consider for example the abstract state with chronological number 1. As $\rho_D(\texttt{Heat}) = 1$ and $\rho_D(\texttt{Cool}) = 2$, we know that the mask priority of the abstract state with chronological number 2 is lower than that of 34. Thus the transition in

the abstract state-space due to continuous flow is chosen before the transition due to a discrete switch.

4.4.2 *Euclidean Distance Priority.* This 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 $\rho_2(\boldsymbol{b}) = d(C_\Pi(\boldsymbol{b}) \cap \mathcal{X}, \mathcal{B})$. 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 $\rho_E : Q_\Pi \to \mathbb{R}$ as

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

4.4.3 *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 on a shortest path to an unsafe location. Assume we generalize the reset function r to $r : 2^{\mathbb{R}^n} \to 2^{\mathbb{R}^n}$ as $r(X) = \bigcup_{x \in X}\{r(x)\}$. We then define the reset distance priority function $\rho_R : Q_\Pi \to \mathbb{R}$ by $\rho_R(l, \boldsymbol{b}) = \rho_3(l, C_\Pi(\boldsymbol{b}) \cap \mathcal{X})$ and $\rho_3 : L \times 2^\mathcal{X} \to \mathbb{R}$ as:

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

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.

It should be noted that there is a trade-off between efficiency and quality of the distance measure when choosing which particular guided search strategy described above to pick. The reset distance measure provides the most accurate distance computation amongst the three choices; however, this accuracy comes with an increased computation time of the distance measure. Since the guided search is supposed to speed up the discovery of a counter-example, it is counter-productive to spend too much time in computing this guiding measure. In most cases, it was observed that the mask priority measurement provides a reliable enough distance measure while being extremely fast. This is due to the fact that it only considers the Boolean vector representation and does not need to compute Euclidean distances of the corresponding sets.

4.5 Generalized Predicate Abstraction

We present a formal framework for coarse predicate abstraction — generalized predicate abstraction — which allows clustering of abstract states. We will use a location-specific predicate abstractor: The main idea of the location-specific pred-

icate 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 linear hybrid system, 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 localization of predicates as discussed in [Henzinger et al. 2004; Jain et al. 2005].

For a hybrid system H and a given set of predicates Π, we will define an abstraction of the abstract state-space Q_Π.

Definition 4.1. The generalized predicate abstract state-space is defined as

$$\hat{Q}_\Pi := L \times \mathbb{T}^k,$$

such that

$$(l, \boldsymbol{t}) \overset{\Pi}{\rightarrow}_G (l', \boldsymbol{t'}) \;:\Leftrightarrow\; \exists \boldsymbol{b} \in c(\boldsymbol{t}), \boldsymbol{b'} \in c(\boldsymbol{t'}) : (l, \boldsymbol{b}) \overset{\Pi}{\rightarrow} (l', \boldsymbol{b'}).$$

The set of initial abstract states is

$$\hat{Q}_0 := \{(l, \boldsymbol{t}) \in \hat{Q}_\Pi \mid \exists \boldsymbol{b} \in c(\boldsymbol{t}) : (l, \boldsymbol{b}) \in Q_0\}.$$

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 $\hat{Q}_\Pi$. Therefore, we restrict our attention to a subset of $\hat{Q}_\Pi$. In this context we can think of $* \in \mathbb{T}$ as a *don't care* term.

Definition 4.2. A subset of abstract states $Q \subseteq \hat{Q}_\Pi$ is called **location-specific**, iff

(1) $\forall l \in L, \boldsymbol{b} \in \mathbb{B}^k \exists \boldsymbol{t} \in \mathbb{T}^k : \boldsymbol{b} \in c(\boldsymbol{t}) \wedge (l, \boldsymbol{t}) \in Q$, and
(2) $\forall (l, \boldsymbol{t_1}), (l, \boldsymbol{t_2}) \in Q : \boldsymbol{t_1} \neq \boldsymbol{t_2} \Rightarrow c(\boldsymbol{t_1}) \cap c(\boldsymbol{t_2}) = \emptyset$.

The set of transitions for a location-specific Q is the restriction of $\overset{\Pi}{\rightarrow}_G$ to Q, and 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 an 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. For an example of the use of generalized predicate abstraction in our tool we refer the reader to Section 6.1. It should be noted that multi-valued logic has been used in the context of predicate abstraction before, such as [Godefroid et al. 2001]. The following theorem stating the soundness of this approach is based on the soundness of the predicate abstraction algorithm [Alur et al. 2002].

THEOREM 4.3. *If the generalized predicate abstraction routine terminates and reports that the system is safe, then the corresponding concrete system is also safe.*

PROOF. The soundness of the generalized predicate abstraction algorithm follows from the soundness of the predicate abstraction algorithm. Similarly to the proof of Theorem 3.4, we can show that if the set of reachable states and the set of unsafe states in the location-specific abstract system do not intersect, then the reachable states in the predicate abstract state-space do not intersect with the unsafe states. Following Theorem 3.4 this assures us that the concrete system is also safe. □

5. BOUNDED COMPLETENESS

Given a linear hybrid system H, an initial set X_0, and an unsafe set $\mathcal{B}_X$, the verification problem is to determine if there is an execution of H starting in X_0 and ending in $\mathcal{B}_X$. 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., $\mathcal{B}_X$ 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 will focus on errors introduced by approximating reachable sets in one continuous step using polyhedra, as well as due to predicate abstraction. We will 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. 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 this to a distance function $d : 2^X \times 2^X \to \mathbb{R}_{\geq 0}$ by

$$d(S,S') = \min_{(l,x)\in S,(l',x')\in S'} d((l,x),(l',x')).$$

5.1 Completeness for Continuous Systems

We 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; that is we assume that the unsafe states are not reachable and that there is a minimal constant distance between the set of reachable states and the unsafe states. In addition we use the knowledge of the optimization of the search strategy which prohibits multiple successive continuous successors as described in Section 4.1.

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 $\mathcal{B}_X$ respectively using the conjunction of a finite set of predicates. In addition, assume a separation of the set `Reach` of states reachable from $\mathcal{X}_0$, and $\mathcal{B}_X$, that is $d(\texttt{Reach}, \mathcal{B}_X) \geq \epsilon$. Following [Dang 2000], we know that we can find a small enough time-step that will ensure that the over-approximation error due to the computation of convex hulls will not result in an overlap of the over-approximation of `Reach` with $\mathcal{B}_X$. Figure 9 illustrates this idea. Starting from an initial polyhedron P^0 representing $\mathcal{X}_0$, we compute an over-approximation of the reachable set `Reach` using subsequent convex hulls $P^1, P^2, \ldots$, as it was described in Section 3.2.

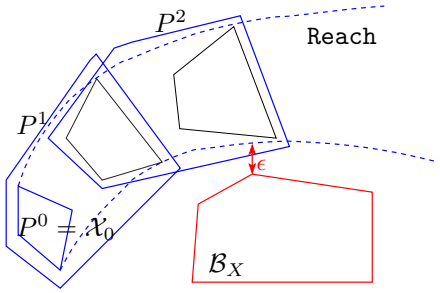

Fig. 9. Proving completeness for purely continuous systems

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 $\mathcal{B}_X$. Given the optimization of our search strategy it is clear that any abstract refinement of $\mathcal{B}_X$ will not be declared reachable by the search.

5.2 (n, τ, δ)-Safety

We also 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* (BMC) [Clarke et al. 2001] can be viewed as establishing safety of discrete systems up to a fixed number of transitions. BMC can also be used to analyze liveness properties in timed automata [Sorea 2002]. We first define the notion of bounded safety for hybrid systems formally.

Definition 5.1. For a hybrid system $H = (\mathcal{X}, L, X_0, I, f, T)$ we define the set of states $\texttt{Reach}^{(n,\tau)}$ that are reachable using at most n discrete switches and a combined flow of at most τ time-units from the initial states X_0 as

—$\texttt{Reach}^{(0,0)} := X_0$,

—$\texttt{Reach}^{(i+1,t)} := \texttt{Post}_D(\texttt{Reach}^{(i,t)}) \cup \texttt{Reach}^{(i,t)} \; \forall i \geq 0$, and

—$\texttt{Reach}^{(i,t+\Delta t)} := \left\{ \begin{array}{c} (l,y) \in X | \exists (l,x) \in \texttt{Reach}^{(i,t)}, 0 \leq t' \leq \Delta t, \mu \in \mathcal{U} : \\ \Phi_l(x,t',\mu) = y \wedge \forall 0 \leq t'' \leq t' : \Phi_l(x,t'',\mu) \in \mathcal{I}_l \end{array} \right\} \forall \Delta t > 0.$

A hybrid system $H = (\mathcal{X}, L, X_0, I, f, T)$ is called (n, τ, δ)**-safe** for the unsafe set $\mathcal{B}_X$, iff the set of states $\texttt{Reach}^{(n,\tau)}$ has a distance of at least δ to the set of unsafe states $\mathcal{B}_X$:

$$d(\texttt{Reach}^{(n,\tau)}, \mathcal{B}_X) > \delta.$$

The proof 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 5.2 BOUNDED COMPLETENESS. *The predicate abstraction algorithm is complete for the class of* (n, τ, δ)*-safe hybrid systems.*

PROOF. We will first compute the maximal approximation error for the over-approximation of the reachable state set using our polyhedral over-approximation strategy. We only consider over-approximation errors and not numerical errors. The over-approximation errors that we consider here are due to the computation of $\texttt{ApproxPost}_C^{\Pi}$ as discussed in Section 3.2, while numerical errors refer to rounding errors inherent to using floating point rational numbers. Following [Dang 2000], we know that the error of the over-approximation of the reachable set by continuous flow in terms of the Hausdorff distance is bounded by

$$2M||A||r + O(r^2),$$

with r being the slicing time step, M being an upper bound on $||x||$ and A being the linear matrix of the dynamics. If we assume that $A_{\max}$ is the matrix of the continuous dynamics in the hybrid system H that has the maximal norm, we define $m := 2M||A_{\max}||$.

As we do not consider numerical errors, we can assume that we do not add any error during discrete transitions. Additionally to the aforementioned over-approximation error during continuous flow, an error ϵ evolves under the continuous dynamics f, in the worst case, as:

$$\epsilon(t) = e^{L_f t}\epsilon$$

for $t \in \mathbb{R}_{\geq 0}$, where L_f is a Lipschitz constant of the dynamics f.

We now consider the approximation error after k instances of continuous flow. After the first instance of continuous flow, the approximation error ϵ_1 can be bounded by:

$$\epsilon_1 \leq mr + O(r^2).$$

Be L the largest Lipschitz constant for all dynamics in the hybrid system H. After the second instance of continuous flow, the approximation error ϵ_2 can be bounded by:

$$\begin{aligned} \epsilon_2 &\leq e^{L\tau}\epsilon_1 + mr + O(r^2), \\ &\leq e^{L\tau}mr + mr + e^{L\tau}O(r^2). \end{aligned}$$

Similarly, we can show that the approximation error ϵ_k after k instances of continuous flow can be bounded by:

$$\epsilon_k \leq \sum_{i=0}^{k} e^{iL\tau}mr + e^{kL\tau}O(r^2).$$

We only need to consider paths in the abstract state-space of n transitions. Given our optimization technique of eliminating consecutive continuous transitions, we can have at most $n+1$ instances of continuous flow. We can then choose a time step slicing size r, such that $\epsilon_{n+1} < \delta$.

The last thing we need to show is that we can find a finite set of predicates such that a (n, τ, δ)-safe hybrid system can be proven correct using the predicate abstraction model checker. The set of predicates will include the finite many predicates describing the initial region and the unsafe set. We will also include all guards and invariants specified in the hybrid system.

We have shown above that we can over-approximate the set of reachable states using polyhedra. If we were to include all predicates that correspond to faces of these polyhedra, we can use this finite set of predicates to define our abstract state-space. This set will provide the predicate abstraction model checker with predicates to prove the (n, τ, δ)-safe hybrid system safe. □

6. IMPLEMENTATION AND EXPERIMENTATION

We have presented foundations for automated verification of safety properties of hybrid systems by combining the ideas of predicate abstraction and polyhedral approximation of reachable sets of linear continuous dynamics. Our current prototype implementation of the predicate abstraction model checking and the counter-example analysis tool are both implemented in C++ using library functions of the hybrid systems reachability tool `d/dt` [Asarin et al. 2000]. We implemented a translation procedure from CHARON [Alur et al. 2003] source code to the predicate abstraction input language which is based on the `d/dt` input language. A detailed overview of a hybrid systems verification case study starting from CHARON source code is given in [Ivančić 2002]. Our tool uses the polyhedral libraries CDD [Fukuda 2001] and QHull [Barber et al. 1996].

In addition to the thermostat example of Figure 1, we demonstrate the feasibility of our approach using four other case studies in this section. The first one involves verification of a parametric version of Fischer's protocol for timing-based mutual exclusion. The second and third one involves analysis of different models of cruise controller. The fourth case study is based on a navigation benchmark for hybrid systems proposed in [Fehnker and Ivančić 2004]. In each of these cases, we show how predicate abstraction can be effective in establishing safety of the system.

6.1 Mutual Exclusion with Time-Based Synchronization

We 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 10. The variable `turn` is used to establish right of access in the model. The system starts with `turn` $= 0$ and both agents are in their respective `Idle` locations. Once process $i \in \{1, 2\}$ moves to its respective `Request` location, it will take at most Δ time-units to assign i to `turn`, establishing its wish to access the shared resource, and switch to its `Check` location. The process is required to stay in its `Check` location for at least δ time-units before it can test the value of `turn`. If `turn` still holds the value i it will access the shared resource, otherwise it does not access the resource this time and moves back to its `Idle` location. We use this example to illustrate the use of various techniques presented in earlier sections.

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 counterexample that proves the two processes may access the shared resource at the same time. The trace of abstract states that represents a valid counterexample in the original system is given in Figure 11.

On the other hand, if $\delta > \Delta$, then the system preserves mutual exclusive use of the shared resource. In this case, a process that is already in its `Check` location will be sure that if the other process moved to its `Request` location afterwards, that it would have also advanced to its `Check` location. That switch would have caused a

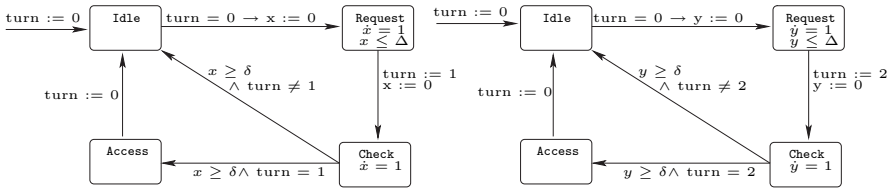

Fig. 10. The two processes for the mutual exclusion example. We omit the constraints $\dot{x} = 0$ and $\dot{y} = 0$ in the respective `Idle` and `Check` locations.

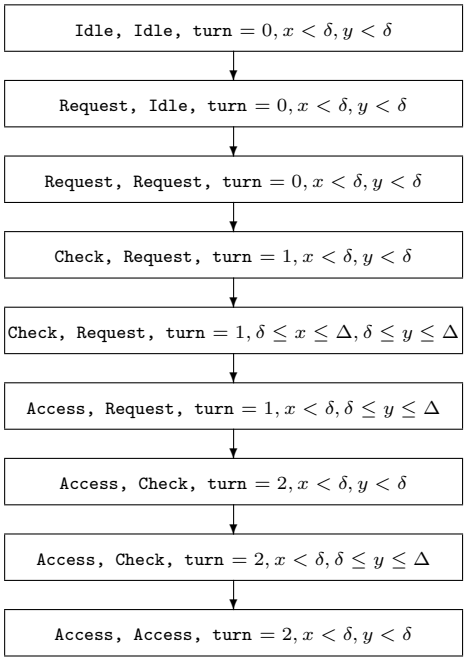

Fig. 11. A counterexample for the mutual exclusion problem for the parameter setting $\Delta \geq \delta$. The predicates that were used to find this counterexample are the predicates given by the guards and invariants of the composed hybrid system. These are: $x \geq \delta, y \geq \delta, x \leq \Delta$ and $y \leq \Delta$. The states do not show the constantly true linear predicates over the parameters $\Delta \geq \delta$, $\Delta > 0$ and $\delta > 0$.

change in the value of the shared variable `turn`. Hence, the first process to enter its `Check` location will not be able to access the shared resource.

It should be noted that our tool supports parameters indirectly. In this example, we can add the two parameters to the state-space of the system, thus considering a 4-dimensional problem, where the differential equations for the parameters δ and Δ simply are $\dot{\delta} = \dot{\Delta} = 0$.

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

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 I. 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 omitted in all locations $l \in L$.

finds 54 reachable abstract states (see [Alur et al. 2002]), whereas, if we use the location-specific predicates as described in Table I, we only reach 24 abstract states. The graph of reachable abstract states for this example using the predicates as specified in Table I is presented in Figure 12. This table can be obtained by starting off with the effective guards and invariants for each location. Subsequent refinement of the table can be performed by the counterexample analysis procedure (see [Alur et al. 2003a]), where we add new predicates only in the respective location.

6.2 Vehicle Coordination

We have 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 briefly describe this system. Details on the design can be found in [Godbole and Lygeros 1994]. 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 $\lambda_a = 0s^2$, $\lambda_v = 1s$, 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:

$$\begin{cases} \dot{d}_i = v_{i-1} - v_i, \\ \dot{v}_{i-1} = a_{i-1}, \\ \dot{v}_i = a_i, \\ \dot{a}_i = u; \end{cases} \tag{4}$$

where u is the control. Without going into details, the controller for the leader car of platoon i proposed in [Godbole and Lygeros 1994] 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, $d_i > 0m$. Here, we focus only on two regions which are critical from a safety point of view: "track optimal velocity" ($v_i^e \leq -10$ and $e_i \geq -1m - \epsilon$)

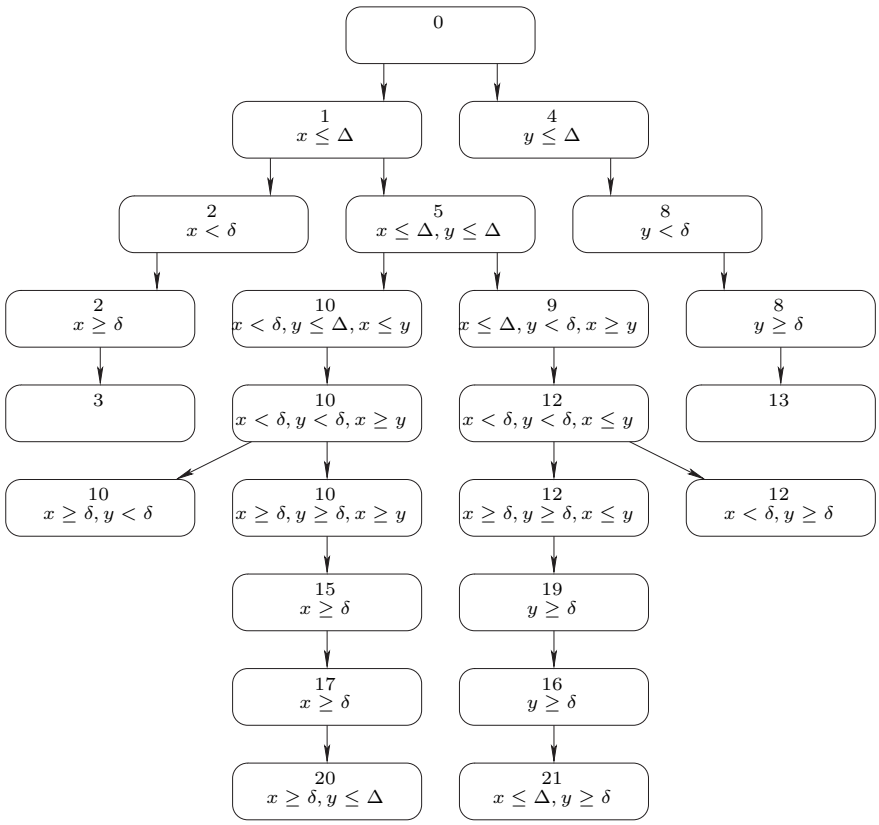

Fig. 12. Reachable abstract transition graph for Fischer's protocol using the location-specific predicate abstraction routine: We only reach 24 abstract states using the predicates as specified in Table I, whereas the original predicate abstraction routine discovers 54 reachable states. The transitions shown correspond to a BFS-search of the abstract state space. For clarity, we only draw the transitions that correspond to the first occurrence of an abstract state. Note that the predicates $0 \leq \Delta, 0 \leq \delta, 0 \leq x, 0 \leq y, \Delta < \delta$ are supposed to be true in all locations.

and "track velocity of previous car" ($v_i^e \leq -10$ and $e_i \leq -1m$). We include a thickening parameter $\epsilon > 0m$ into the model to add non-determinism to it. The two regions under consideration overlap allowing the controller to either use the "track optimal velocity" controller or the "track velocity of previous car" controller in this ϵ-thick region. Besides adding some non-determinism to the model, it also provides improved numerical stability to the simulation and reachability computation, as it is numerically hard to determine the exact time at which a switch occurs.

The respective control laws u_1 and u_2 are as follows:

$$u_1 = 0.125d_i + 0.75v_{i-1} - (0.75 + 0.125\lambda_v)v_i - 1.5a_i - 0.125\lambda_p, \quad (5)$$

$$u_2 = d_i + 3v_{i-1} - (3 + \lambda_v)v_i - 3a_i - \lambda_p. \quad (6)$$

Note that these regions correspond to situations where the platoon in front moves considerably 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 linear hybrid system 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 given by (4) as specified above, with u specified by (5) and (6). 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 unsafe set is specified as $d_i \leq 0$. To construct the discrete abstract system, in addition to the predicates of the invariants and guards we use four predicates $d_i \leq 0, d_i \geq 2, d_i \geq 10$ and $d_i \geq 20$ which allow to separate safe and unsafe states, and the total number of initial predicates is 11. For the initial set specified as $20 \leq d_i \leq 100 \wedge -1 \leq a_i \leq 1 \wedge 15 \leq v_{i-1} \leq 18 \wedge 20 \leq v_i \leq 25$, the tool found 14 reachable abstract states and reported that the system is safe. For *individual continuous modes* this property has been proven in [Puri and Varaiya 1995] using optimal control techniques.

6.3 Coordinated Adaptive Cruise Control

We have also successfully applied our predicate abstraction technique to verify a model of the *Coordinated Adaptive Cruise Control* mode of a vehicle-to-vehicle coordination system. This case study is provided by the PATH project. Let us first briefly describe the model (see [Girard 2002] 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 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 can be modeled as a linear hybrid system 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.

6.4 Navigation Benchmark

The navigation benchmark, which was proposed in [Fehnker and Ivančić 2004] for the evaluation of hybrid systems verification tools, deals with a single object – it can be thought of a vehicle or robot although the dynamics are not vehicle dynamics – moving around in the $\mathbb{R}^2$ plain. The object is trying to avoid certain obstacles in the modeled world while trying to reach certain targets in the world. The object is modeled using its center position $x \in \mathbb{R}^2$ and velocity $v \in \mathbb{R}^2$. A desired velocity $v_d \in \mathbb{R}^2$ is determined by the position of the object in an $n \times m$ grid, and the desired velocities may take values for $i \in \{0, 1, \ldots, 7\}$ using

$$v_d = \begin{pmatrix} \sin(i \cdot \pi/4) \\ \cos(i \cdot \pi/4) \end{pmatrix}, \tag{7}$$

It is assumed that the width of each cell is $1 + 2\epsilon$ for $0 \leq \epsilon < 0.5$, and that the lower left corner of the grid is located at $(-\epsilon, -\epsilon)^T$. The square grid cells are arranged such that neighboring cells overlap for 2ϵ. First consider the case that $\epsilon = 0$. Then each cell can be described by its lower left corner at some $(j, k)^T \in \mathbb{R}^2$ and its upper right corner at $(j + 1, k + 1)^T$ for $j, k \in \mathbb{N}$. However, if each cell has width $1 + 2\epsilon$ and overlaps as described above, then it can be described by having its lower left corner at $(j - \epsilon, k - \epsilon)^T \in \mathbb{R}^2$ and its upper right corner at $(j + 1 + \epsilon, k + 1 + \epsilon)^T$. Neighboring cells are overlapping, which allows modeling of non-determinism and also improves numerical stability of various analyses since determining the exact time of the switch of the moving object between the cells may be numerically hard to compute.

Each square grid cell can be labelled by a number $i \in \{0, 1, \ldots, 7\}$ describing its desired velocity v_d as specified by Equation (7). In addition, the grid contains also cells labelled `A` representing cells that are *attracting* the object and shall be reached, and cells labelled by `B` representing a *bad region* that ought to be avoided. Thus,

all obstacles in the model are squares, as are the targets.

Given v_d the behavior of the velocity is determined by the differential equation $\dot{v} = A(v - v_d)$, where $A \in \mathbb{R}^{2\times 2}$ is assumed to have eigenvalues with strictly negative real part only. This guarantees that the velocity will converge to the desired velocity. In the following, we assume the matrix A to be specified by

$$A = \begin{pmatrix} -1.2 & 0.1 \\ 0.1 & -1.2 \end{pmatrix}. \tag{8}$$

An instance of this benchmark is also characterized by the initial conditions on the position x of the object and its velocity v, by the matrix A in the differential equation for v and by the map of the grid, which can be represented by a $n \times m$ matrix with elements from $\{0, 1, \ldots, 7\} \cup \{\mathtt{A}, \mathtt{B}\}$. This matrix can be written as

$$\begin{pmatrix} \mathtt{B} & 2 & 4 \\ 4 & 3 & 4 \\ 2 & 2 & \mathtt{A} \end{pmatrix}. \tag{9}$$

Each instance of this benchmark contains 4 continuous dimensions, and has linear dynamics without uncertain input. On the other hand, it can easily be extended in the number of discrete locations by manipulating the size and the content of the map of the grid. It can therefore be applied to analyze the influence of the number of discrete locations on the performance on various methods. Note that the other benchmarks proposed in [Fehnker and Ivančić 2004] allow scaling of the continuous part of the system also.

In the following, a few experimental results will be presented that have been obtained for instances of the navigation benchmark. The base instance considered is the one described earlier using the matrix of Equation (8) and the grid cell map of Equation (9) for $\epsilon = 0.1$. Additionally, the initial set of conditions needs to be specified. In the following, the initial position of the object is the grid cell just above the attracting cell labelled `A`. This set of instances of the benchmark example tests the abstraction-based verifier of the CHARON toolkit [Alur et al. 2000] with respect to its adaptiveness to verification tasks where the number of locations grows substantially.

There are two sets of experiments conducted based on this model. First, the effects of varying initial conditions are tested. The variation is in the set of initial conditions for the velocity of the object. Secondly, instances of increasing size in the number of discrete locations are considered without significant impact on the set of reachable states. The instances are both run using a non-abstraction based tool as well as the predicate abstraction based verifier.

First, varying initial conditions on the starting velocity of the object are considered. The set of initial conditions are described in Table II. The first instance (I) described in the table is an easily verifiable instance of the benchmark model, since the object has an initial starting velocity pointing towards the attracting cell. However, the second and third instance are of somewhat higher complexity since the object may start off with an initial velocity that is pointing away from the attracting cell directly towards the bad cell.

The experiments were performed both with the CHARON based verifier and a non-abstraction based tool. As expected, both tools were able to verify instance (I)

Instance	Initial Conditions
(I)	$\texttt{vel\_x} \in [-0.3, 0.3], \texttt{vel\_y} \in [-0.3, 0]$
(II)	$\texttt{vel\_x} \in [-0.3, 0.3], \texttt{vel\_y} \in [-0.3, 0.3]$
(III)	$\texttt{vel\_x} \in [-0.4, 0.4], \texttt{vel\_y} \in [-0.4, 0.4]$

Table II. Varying initial conditions on the velocity of the moving object

without any significant user-guidance in a few seconds. In fact, it turned out that for this instance, the other tool outperformed the CHARON based tool with respect to the computation time. This is due to the fact that certain initialization steps of the CHARON based tool are not needed when directly computing reachable concrete states. These steps, for example, include some computations on the set of predicates specified by the user. Similarly, iteration over possible successor states in such a simple example may take longer than just computing the reachable sets directly.

When trying to verify instance (II), both tools were able to complete the verification task and prove safety (avoidance of the bad cell). However, during the verification of this instance several parameters needed to be adjusted in both tools to complete the task. The verification of instance (III), however, was proven in the CHARON based verifier with the same set of parameters, while the other tool was not able to complete the verification task. Either, the time-step was too large, and safety could not be proven, or the verification task was not completed due to a memory overflow. For the verification of instance (III) 55 linear predicates were used in the CHARON based verifier. The search took approximately 30 minutes, and found nearly 1700 reachable abstract states, while discovering more than 4500 abstract states to be non-consistent. The fact that the non-abstraction based tool could not complete the verification for this example underscores the memory savings that the predicate abstraction approach brings to hybrid systems verification.

A second set of instances based on this example is considered. The base instance is the second instance (II) of Table II. As discussed in the previous paragraph, both verification tool were able to verify this instance. Since both tools are computing the reachable sets on-the-fly, it is expected that the inclusion of additional – non-reachable – locations does not have a significant impact on the performance of the tools. A few bigger instances are created by adding more grid cells to the left of the grid cell map of Equation (9). The maps used will take the form

$$\begin{pmatrix} 4 & 4 & \cdots & 4 & 4 & \texttt{B} & 2 & 4 \\ 4 & 4 & \cdots & 4 & 4 & 4 & 3 & 4 \\ 4 & 4 & \cdots & 4 & 4 & 2 & 2 & \texttt{2} \end{pmatrix}.$$

Scripts were produced that create descriptions of such models as described earlier in this chapter for a map of size $n \times 3$ following the above guidelines. The output can then be used to create CHARON models and then generate descriptions in the verifier input format. The first considered case was the 3×3 map described earlier. A second instance is the map of size 33×3, since the obtained version of the other tool has a static limit of maximal 100 locations. As expected the CHARON based verifier was able to complete the verification task in about the same amount of time as for the first instance of this problem. It was expected that the other tool would also be able to complete the verification task using the same set of

Instance	Regular			Generalized		
	Search	Setup	Memory	Search	Setup	Memory
(a)	34	0				
(b)	150	3	68MB	149	3	68MB
(c)	145	7	180MB	151	8	180MB
(d)	452	103	550MB			
(e)	1305	1624	1675MB	4711	1473	1675MB

Table III. Performance Results for various instances of the navigation benchmark

parameters. Surprisingly though, it was not possible to complete the verification task for various sets of settings. It was not possible to clearly establish a reason for this behavior.

To consider the scaling behavior of the predicate abstraction based model checking tool, more instances of this sort were tested. In addition to the aforementioned instances of size (a) 3×3 and (b) 33×3, the tool was also run for examples of the size (c) 103×3, (d) 333×3, and (e) 1003×3. A few interesting statistics about the two bigger instances follow. The automatically generated CHARON model for instance (d) contains more than $16,000$ lines, and has a size of 420KB. The CHARON converter that prints out a model in the input language of the model checking tool runs for around 90 seconds and uses 87MB of working memory for the conversion. The converted model contains more than $63,000$ lines and has a size of 1.06MB. The verification search algorithm uses 774 linear predicates for the abstraction, most of which are parallel due to the gridding of the map. For the larger instance (e), the CHARON model contains more than $49,000$ lines and has a size of 1.24MB. It takes more than seven minutes to compute the model using more than 241MB of working memory for the conversion. The generated model contains more than $190,000$ lines, has a size of 3.2MB, and the predicate abstraction based verification uses 2084 linear predicates.

For all the instances (a) - (e) the predicate abstraction based model checker was able to complete the analysis – both with the original framework as well as in the generalized predicate abstraction framework. The performance results are presented in Table 6.4 which presents the performance of the verification algorithm for varying instances. The verification was performed on a Sun Enterprise 3000 ($4 \times$ 250MHz UltraSPARC) with 1GB of memory and 4GB of swap memory by Sun Microsystems Inc. running SunOS 5.8. Since the particular workload can vary between verification runs, these results cannot necessarily accurately reflect the runtime behavior of the verification. The table shows the performance results for each instance in terms of how long (in real-time seconds) the search routine took (Search), how much time was spend reading in the model and initializing all variables before the search started (Setup), and how much memory was used during the verification. For some instances, the table also shows the same performance results for a search using the generalized predicate abstraction approach. The additional predicates that are added to the abstraction due to the increase of the map are excluded in this search.

The performance result presented in Table 6.4 that correspond to the regular search routine are graphically also shown in Figure 13. It can be seen that both the memory requirements and the run-time of the verification grow approximately

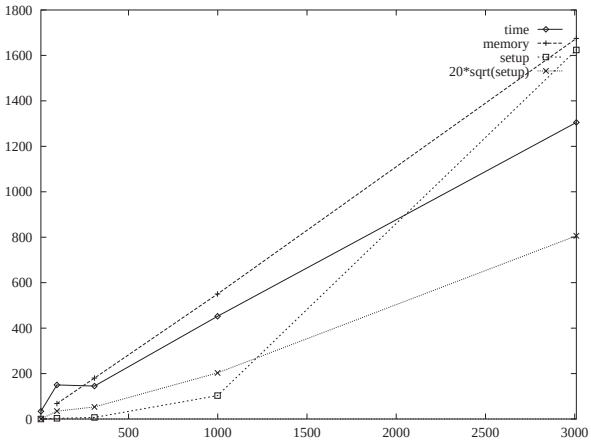

Fig. 13. Performance results for the navigation benchmark

linearly with the size of the grid cell map representing discrete locations of the flat hybrid automaton. For memory requirements this is clearly due to the fact that the size of the model description grows linearly with the number of discrete locations. The amount of memory used during the search in this case remains constant over all instances.

The run-time requirements of the verification search also seem to be linear in the size of the problem description. Some of this can be attributed to the fact that loading the model takes linear time. However, there is one instance that does not follow the trend of the other verification instances which may, for example, have been caused by run-time issues due to the initial use of swap memory.

The amount of time spend to initialize the system grows approximately quadratically with the size of the considered instance. This can be seen in Figure 13 by considering the graph labelled `setup` and a parametric version of it labelled `20*sqrt(setup)`.[5] The results for the generalized predicate abstraction approach show that the memory requirements in both cases are roughly the same for this instance - which is due to the fact that its mainly an issue of loading the model into main memory.

The above set of instances represent hypothetical examples where only a few locations out of many possible locations are really reachable. However, such cases can occur when flattening concurrent systems. For example, five processes of four locations each (such as Fischer's mutual exclusion example of Section 6.1) ends up being 1024 flattened locations, most of which are not reachable.

7. CONCLUSIONS

In this paper we presented 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.

[5]This plot represents the 20-times scaled square root of the time used for initialization.

Our tool applies to hybrid systems with linear continuous dynamics and uncertain, bounded input. The approach presented here extends to non-linear systems and non-linear abstraction predicates in principle. However, in terms of tool development, this would require a complete overhaul of our implementation.

A verifier based on this abstraction 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 counterexample can be analyzed to test if it is a feasible execution of the concrete system.

The success of our scheme crucially depends on the choice of the predicates used for abstraction. We identify such predicates automatically by analyzing spurious counterexamples generated by the search in the abstract state-space. Counterexample guided refinement of abstractions has been used in multiple contexts before, for instance, to identify the relevant timing constraints in verification of timed automata [Alur et al. 1995], to identify the relevant boolean predicates in verification of C programs [Ball and Rajamani 2000] and timed automata [Möller et al. 2002], and to identify the relevant variables in symbolic model checking [Clarke et al. 2000]. In [Alur et al. 2003a] we present the basic techniques for analyzing counterexamples and techniques for discovering new predicates that will rule out spurious counterexamples. Counterexample guided refinement of abstractions for hybrid systems is being independently explored by the hybrid systems group at CMU [Clarke et al. 2003].

The abstract counterexample consists of a sequence of abstract states leading from an initial state to a state violating the property. The analysis problem, then, is to check if the corresponding sequence of modes and discrete switches can be traversed in the concrete system. We perform a forward search from the initial abstract state following the given counterexample in the abstract state-space. The analysis relies on techniques for polyhedral approximations of the reachable sets under continuous dynamics. To speed up the feasibility analysis, we have also implemented a local test that checks for feasibility of pair-wise transitions, and this proves to be effective in many cases. If the counterexample is found to be infeasible, then we wish to identify one or more new predicates that would rule out this sequence in the refined abstract space. This reduces to the problem of finding one or more predicates that *separate* two sets of polyhedra. We present a greedy strategy for identifying the separating predicates. After discovering new predicates, we then include these predicates to the set of predicates used before, and rerun the search in the refined abstract state-space defined by the enriched predicate set. For sake of brevity, we omit a more detailed presentation of our counterexample-guided predicate abstraction approach which can be found in [Alur et al. 2003a].

ACKNOWLEDGMENTS

We wish to thank Ansgar Fehnker and Bruce Krogh for fruitful discussions, and the members of the CHARON group at the University of Pennsylvania for their help. We would also like to thank the anonymous reviewers for their insightful and detailed comments. This research was partially supported 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). Preliminary versions of this paper have appeared in *Hybrid Systems: Computation and Control* 2002 and 2003 (see [Alur et al. 2002; 2003b]).

REFERENCES

ALUR, R., COURCOUBETIS, C., HALBWACHS, N., HENZINGER, T., HO, P., NICOLLIN, X., OLIVERO, A., SIFAKIS, J., AND YOVINE, S. 1995. The algorithmic analysis of hybrid systems. *Theoretical Computer Science 138*, 3–34.

ALUR, R., DANG, T., ESPOSITO, J., HUR, Y., IVANČIĆ, F., KUMAR, V., LEE, I., MISHRA, P., PAPPAS, G., AND SOKOLSKY, O. 2003. Hierarchical modeling and analysis of embedded systems. *Proceedings of the IEEE 91,* 1 (January).

ALUR, R., DANG, T., AND IVANČIĆ, F. 2002. Reachability analysis of hybrid systems via predicate abstraction. In *Hybrid Systems: Computation and Control, Fifth International Workshop*, C. Tomlin and M. Greenstreet, Eds. LNCS 2289. Springer-Verlag, 35–48.

ALUR, R., DANG, T., AND IVANČIĆ, F. 2003a. Counter-example guided predicate abstraction of hybrid systems. In *Ninth International Conference on Tools and Algorithms for the Construction and Analysis of Systems*.

ALUR, R., DANG, T., AND IVANČIĆ, F. 2003b. Progress on reachability analysis of hybrid systems via predicate abstraction. In *Hybrid Systems: Computation and Control, Sixth International Workshop*.

ALUR, R. AND DILL, D. 1994. A theory of timed automata. *Theoretical Computer Science 126*, 183–235.

ALUR, R., GROSU, R., HUR, Y., KUMAR, V., AND LEE, I. 2000. Modular specifications of hybrid systems in CHARON. In *Hybrid Systems: Computation and Control, Third International Workshop*. Vol. LNCS 1790. 6–19.

ALUR, R., HENZINGER, T., AND HO, P.-H. 1996. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering 22,* 3, 181–201.

ALUR, R., HENZINGER, T., LAFFERRIERE, G., AND PAPPAS, G. 2000. Discrete abstractions of hybrid systems. *Proceedings of the IEEE*.

ALUR, R., ITAI, A., KURSHAN, R., AND YANNAKAKIS, M. 1995. Timing verification by successive approximation. *Information and Computation 118,* 1, 142–157.

ASARIN, E., BOURNEZ, O., DANG, T., AND MALER, O. 2000. Approximate reachability analysis of piecewise-linear dynamical systems. In *Hybrid Systems: Computation and Control, Third International Workshop*. LNCS 1790. Springer Verlag, 21–31.

BALL, T. AND RAJAMANI, S. 2000. Bebop: A symbolic model checker for boolean programs. In *SPIN 2000 Workshop on Model Checking of Software*. LNCS 1885. Springer, 113–130.

BARBER, C., DOBKIN, D., AND HUHDANPAA, H. 1996. The Quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software 22,* 4 (December), 469–483.

BEHRMANN, G., LARSEN, K., PEARSON, J., WEISE, C., AND YI, W. 1999. Efficient timed reachability analysis using clock difference diagrams. In *Computer Aided Verification, 11th International Conference*. LNCS, vol. 1633. Springer-Verlag.

BENGSSTON, J., LARSEN, K., LARSSON, F., PETTERSSON, P., YI, W., AND WEISE, C. 1998. New generation of UPPAAL. In *Proceedings of International Workshop on Software Tools for Technology Transfer*.

BENSALEM, S., GANESH, V., LAKHNECH, Y., NOZ, C. M., OWRE, S., RUESS, H., RUSHBY, J., RUSU, V., SAÏDI, H., SHANKAR, N., SINGERMAN, E., AND TIWARI, A. 2000. An overview of SAL. In

LFM 2000: Fifth NASA Langley Formal Methods Workshop, C. M. Holloway, Ed. Hampton, VA, 187–196.

Chutinan, A. and Krogh, B. 1999. Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations. In *Hybrid Systems: Computation and Control, Second International Workshop*. LNCS 1569. Springer-Verlag, 76–90.

Chutinan, A. and Krogh, B. 2000. Approximate quotient transition systems for hybrid systems. In *Proceedings of the 2000 American Control Conference*.

Clarke, E., Biere, A., Raimi, R., and Zhu, Y. 2001. Bounded model checking using satisfiability solving. *Formal Methods in Systems Design 19,* 1, 7–34.

Clarke, E., Fehnker, A., Han, Z., Krogh, B., Ouaknine, J., Stursberg, O., and Theobald, M. 2003. Abstraction and counterexample-guided refinement of hybrid systems. *Intern. Journal of Foundations of Computer Science 14,* 4 (August), 583–604.

Clarke, E., Fehnker, A., Han, Z., Krogh, B., Stursberg, O., and Theobald, M. 2003. Verification of hybrid systems based on counterexample-guided abstraction refinement. In *Ninth International Conference on Tools and Algorithms for the Construction and Analysis of Systems*.

Clarke, E., Grumberg, O., Jha, S., Lu, Y., and Veith, H. 2000. Counterexample-guided abstraction refinement. In *Computer Aided Verification*. 154–169.

Clarke, E., Grumberg, O., and Long, D. 1993. Verification tools for finite-state concurrent systems. In *Decade of Concurrency – Reflections and Perspectives (Proceedings of REX School)*. LNCS 803. Springer-Verlag, 124–175.

Clarke, E. and Kurshan, R. 1996. Computer-aided verification. *IEEE Spectrum 33,* 6, 61–67.

Computer Science and Telecommunications Board. 2001. *Embedded, Everywhere: A Research Agenda for Networked Systems of Embedded Computers*.

Corbett, J., Dwyer, M., Hatcliff, J., Laubach, S., Pasareanu, C., Robby, and Zheng, H. 2000. Bandera: Extracting finite-state models from Java source code. In *Proceedings of 22nd International Conference on Software Engineering*. 439–448.

Cousot, P. and Cousot, R. 1977. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM Symposium on Principles of Programming Languages*. 238–252.

Cury, J., Krogh, B., and Niinomi, T. 1998. Synthesis of supervisory controller for hybrid systems based on approximating automata. *IEEE Transaction on Automatic Control 43,* 4 (April), 564–569.

Dang, T. 2000. Verification and synthesis of hybrid systems. Ph.D. thesis, Institut National Polytecnique de Grenoble.

Dang, T. and Maler, O. 1998. Reachability analysis via face-lifting. In *Hybrid Systems: Computation and Control*. LNCS, vol. 1386. Springer-Verlag, 96–109.

Das, S., Dill, D., and Park, S. 1999. Experience with predicate abstraction. In *Computer Aided Verification, 11th International Conference*. LNCS 1633. Springer, 160–171.

Daws, C., Olivero, A., Tripakis, S., and Yovine, S. 1996. The tool kronos. In *Hybrid Systems III: Verification and Control*. LNCS 1066. Springer-Verlag, 208–219.

de Moura, L., Owre, S., Ruess, H., Rushby, J., Shankar, N., Sorea, M., and Tiwari, A. 2004. SAL 2. In *Computer-Aided Verification, CAV 2004*, R. Alur and D. Peled, Eds. Lecture Notes in Computer Science, vol. 3114. Springer-Verlag, Boston, MA, 496–500.

Fehnker, A. and Ivančić, F. 2004. Benchmarks for hybrid systems verification. In *Hybrid Systems: Computation and Control*, R. Alur and G. Pappas, Eds. Lecture Notes in Computer Science, vol. 2993. Springer, 326–341.

Fukuda, K. 2001. cddlib reference manual, cddlib version 092a. Tech. rep., McGill University.

Girard, A. 2002. Hybrid system architectures for coordinated vehicle control. Ph.D. thesis, University of California at Berkeley.

Godbole, D. and Lygeros, J. 1994. Longitudinal control of a lead card of a platoon. *IEEE Transactions on Vehicular Technology 43,* 4, 1125–1135.

Godefroid, P., Huth, M., and Jagadeesan, R. 2001. Abstraction-based model checking using modal transition systems. In *12th International Conference on Concurrency Theory*. Lecture Notes in Computer Science, vol. 2154. 426–440.

Graf, S. and Saidi, H. 1997. Construction of abstract state graphs with PVS. In *Proc. 9th International Conference on Computer Aided Verification (CAV'97)*. Vol. 1254. Springer Verlag, 72–83.

Greenstreet, M. and Mitchell, I. 1999. Reachability analysis using polygonal projections. In *Hybrid Systems: Computation and Control, Second International Workshop*. LNCS 1569. Springer-Verlag, 103–116.

Halbwachs, N., Proy, Y., and Raymond, P. 1994. Verification of linear hybrid systems by means of convex approximations. In *International Symposium on Static Analysis*. LNCS 864. Springer-Verlag.

Henzinger, T., Ho, P., and Wong-Toi, H. 1995. HyTech: the next generation. In *Proceedings of the 16th IEEE Real-Time Systems Symposium*. 56–65.

Henzinger, T., Ho, P., and Wong-Toi, H. 1997. HyTech: a model checker for hybrid systems. *Software Tools for Technology Transfer 1*.

Henzinger, T., Jhala, R., Majumdar, R., and McMillan, K. 2004. Abstractions from proofs. In *Symposium on Principles of Programming Languages* . ACM Press, 232–244.

Henzinger, T., Jhala, R., Majumdar, R., and Sutre, G. 2002. Lazy abstraction. In *Symposium on Principles of Programming Languages*. 58–70.

Holzmann, G. 1997. The model checker SPIN. *IEEE Transactions on Software Engineering 23,* 5, 279–295.

Holzmann, G. and Smith, M. 2000. Automating software feature verification. *Bell Labs Technical Journal 5,* 2, 72–87.

Ivančić, F. 2002. Report on verification of the MoBIES vehicle-vehicle automotive OEP problem. Tech. Rep. MS-CIS-02-02, University of Pennsylvania. March.

Jain, H., Ivančić, F., Gupta, A., and Ganai, M. 2005. Localization and register sharing for predicate abstraction. In *11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. LNCS, vol. 3340. Springer, 397–412.

Kurzhanski, A. and Varaiya, P. 2000. Ellipsoidal techniques for reachability analysis. In *Hybrid Systems: Computation and Control, Third International Workshop*. LNCS 1790. Springer-Verlag, 202–214.

Loiseaux, C., Graf, S., Sifakis, J., Bouajjani, A., and Bensalem, S. 1995. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design Volume 6, Issue 1*.

Mitchell, I. and Tomlin, C. 2000. Level set methods for computation in hybrid systems. In *Hybrid Systems: Computation and Control, Third International Workshop*. Vol. LNCS 1790. Springer-Verlag, 310–323.

Möller, M. O., Ruess, H., and Sorea, M. 2002. Predicate abstraction for dense real-time systems. *Electronic Notes in Theoretical Computer Science 65,* 6.

Puri, A. and Varaiya, P. 1995. Driving safely in smart cars. Tech. Rep. UBC-ITS-PRR-95-24, California PATH, University of California in Berkeley. July.

Silva, B., Stursberg, O., Krogh, B., and Engell, S. 2001. An assessment of the current status of algorithmic approaches to the verification of hybrid systems. In *40th Conference on Decision and Control*.

Sorea, M. 2002. Bounded model checking for timed automata. *Electronic Notes in Theoretical Computer Science 68,* 5.

Tiwari, A. and Khanna, G. 2002. Series of abstractions for hybrid automata. In *Hybrid Systems: Computation and Control, Fifth International Workshop*, C. Tomlin and M. Greenstreet, Eds. LNCS 2289. Springer-Verlag, 465–478.