Counter-Example Guided Predicate Abstraction of Hybrid Systems ⋆

Rajeev Alur

University of Pennsylvania, 3330 Walnut Street, Philadelphia, PA 19104

Thao Dang

VERIMAG, Centre Équation, 2, avenue de Vignate, 38610 Gières, France

Franjo Ivančić

NEC Laboratories America, Inc., 4 Independence Way, Princeton, NJ 08540

Abstract

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

1 Introduction

Inspired by the success of model checking in hardware verification and protocol analysis [17,27], there has been increasing research on developing tools for automated verification of

⋆ This research was supported in part by ARO URI award DAAD19-01-1-0473, NSF award ITR/SY 0121431, and European IST project CC (Computation and Control).

hybrid (mixed discrete-continuous) models of embedded controllers [1,7,9,13,21,25,32]. Model checking requires the computation of the set of reachable states of a model, and in presence of continuous dynamics, this is typically undecidable. Consequently, contemporary tools for model checking of hybrid systems, such as CheckMate[13] and `d/dt`[9], approximate the set of reachable states by polyhedra. We have recently shown that effectiveness of the reachability computation for hybrid systems can be enhanced using predicate abstraction [3]. Predicate abstraction is a powerful technique for extracting finite-state models from complex, potentially infinite-state, discrete systems (see, for instance, [19,33]), and tools such as Bandera [18], SLAM [10], and Feaver [28] have used it for analysis of C or Java programs. The input to our verification tool consists of the concrete system modeled by a hybrid automaton, the safety property to be verified, and a finite set of predicates over system variables to be used for abstraction. For the sake of efficiency, we require that all invariants, guards, and discrete updates of the hybrid automaton are specified by linear expressions, the continuous dynamics is linear, possibly with bounded input, and the property as well as the abstraction predicates are linear. An abstract state is a valid combination of truth values to the predicates, and thus, corresponds to a polyhedral set of the concrete state-space. The verifier performs an on-the-fly search of the abstract system by symbolic manipulation of polyhedra.

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

The success of our scheme crucially depends on the choice of the predicates used for abstraction. In this paper, we focus on identifying such predicates automatically by analyzing spurious counter-examples generated by the search in the abstract state-space. Counter-example guided refinement of abstractions has been used in multiple contexts before, for instance, to identify the relevant timing constraints in verification of timed automata [8], to identify the relevant boolean predicates in verification of C programs [10], and to identify the relevant variables in symbolic model checking [16]. We present the basic techniques for analyzing counter-examples, techniques for discovering new predicates that will rule out spurious counter-examples, optimizations of these techniques, implementation of these in our verifier, and case studies demonstrating the promise of the approach. Counter-example guided abstraction refinement (CEGAR) for hybrid systems is being independently explored by the hybrid systems group at CMU [14].

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

2 Predicate Abstraction for Linear Hybrid Systems

In this section, we briefly recap the definitions of predicate abstraction for linear hybrid systems and the search strategy in the abstract space as outlined in [3]. The class of linear hybrid systems is formally introduced, which are hybrid systems, where the continuous dynamics are linear with uncertain, bounded input and all guards, invariants and reset actions are linear. Note that this class of hybrid systems is more general than the so-called linear hybrid automata [26]. It should also be noted that the theory of abstraction and counter-example analysis developed in this paper can be applied to more general classes. The focus on linear hybrid systems here is purely due to implementation considerations.

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 expression is of the form $l(x) := \sum_{i=1}^{n} a_i x_i + a_{n+1}$, and 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, the set of finite sets of n-dimensional linear predicates is denoted by $\mathcal{C}_n$, where an element of $\mathcal{C}_n$ represents the conjunction of its elements.

Definition 1 (Linear Hybrid 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 has the form* (l, x), *where* $l \in L$ *is the discrete part of the state, and* $x \in \mathcal{X}$ *is the continuous part.*
- $X_0 \subseteq X$ *is the set of* **initial states**. *It is assumed that for all locations* $l \in L$, *the set* $\{x \in \mathcal{X} \mid (l, x) \in X_0\}$ *is a convex polyhedron.*
- $I : L \to \mathcal{C}_n$ *assigns to each location* $l \in L$ *a finite set of linear predicates* $I(l)$ *defining the* **invariant** *conditions that constrain the value of the continuous part of the state while the discrete location is* l. *The linear hybrid system can only stay in location* l *as long as the continuous part of the state* x *satisfies* $I(l)$, *i.e.* $\forall \pi \in I(l) : \pi(x) = 1$. *The notation* $\mathcal{I}_l$ *is used 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* $\dot{x} = f(l)(x, u)$. *The continuous dynamics is restricted 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* $u : \mathcal{T} \to U$ *such that* $U \subset \mathbb{R}^m$ *is a bounded convex set. It is assumed 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 state* x *is reset to a new value* $r(x)$. *The notation* $\mathcal{G}_t \subseteq \mathcal{I}_l$ *is used to represent 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,* $\mathcal{G}_t := \{x \in \mathcal{I}_l \mid \forall \pi \in g : \pi(x) = 1\}$.

The simple thermostat model of figure 1 is a linear hybrid system according to this definition. All the guards and invariants of the system are linear predicates, the resets are linear, and the continuous dynamics also follow the aforementioned constraints of linearity. The thermostat model consists of three locations, that is $L = \{\texttt{Heat}, \texttt{Cool}, \texttt{Check}\}$. 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 the continuous state-space can be limited such that both the clock t and the temperature T are within the interval $[0, 100]$ without loss of accuracy of the analysis. The continuous state thus is $(t, T) \in \mathcal{X} = [0, 100]^2$.

A state is denoted with $(\texttt{Heat}, (2, 8))$ representing $t = 2 \wedge T = 8$ 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, $\mathcal{I}_{\texttt{Heat}} = \{(t, T) \in [0, 100]^2 \mid T \leq 10 \wedge t \leq 3\}$. The thermostat system, therefore, cannot remain in the `Heat` location when the temperature exceeds ten or the clock exceeds three time-units. The control can switch to the `Cool` location,

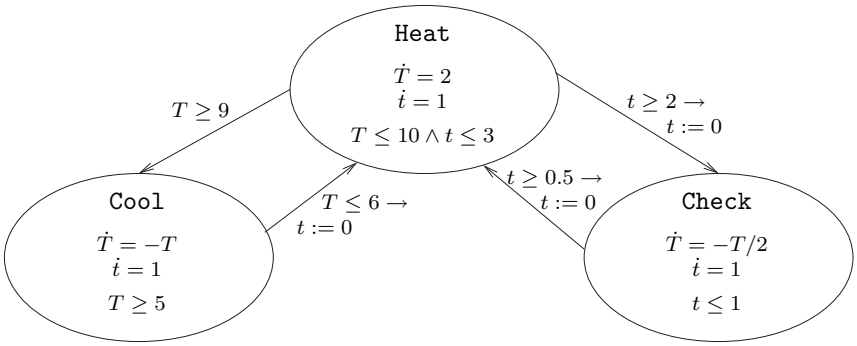

Fig. 1. A simple hybrid system model of a thermostat

which models that the thermostat is switched off, when the guard $T \geq 9$ is enabled. The guard set $\mathcal{G}$ of this transition therefore is $\mathcal{G} = \{(t, T) \in [0, 100]^2 \mid t \leq 3 \wedge 9 \leq T \leq 10\}$. This means, the switch from the `Heat` location to the `Cool` location 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. It is assumed 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 for illustrative purposes.

2.2 Transition System Semantics and Verification Problem

The semantics of a linear hybrid system can be formalized by describing its underlying transition system. We first define the notion of transition systems and traces used throughout this paper.

Definition 2 (Transition Systems) *A transition system is a quadruple $TS = (Q, Q_0, \Sigma, \delta)$ with the following components:*

- *Q is a (possibly infinite) set of* **states***;*
- *$Q_0 \subseteq Q$ is a (possibly infinite) set of* **initial states***;*
- *Σ is a (possibly infinite) set of* **labels***; and*
- *$\delta \subseteq Q \times \Sigma \times Q$ is a (possible infinite) relation capturing* **transitions***.*

A trace of a transition system $TS = (Q, Q_0, \Sigma, \delta)$ *is a sequence $\sigma : \mathbb{N} \rightarrow Q$, such that $\sigma(0) \in Q_0$, and $\forall k \geq 0\, \exists t \in \Sigma : (\sigma(k), t, \sigma(k+1)) \in \delta$. The notations $q \xrightarrow{t} q'$ and $q \rightarrow_t q'$ are often used instead of $(q, t, q') \in \delta$.*

The semantics of a linear hybrid system can now be formalized assuming an admissible set $\mathcal{U}$ of input functions $\mu : \mathcal{T} \rightarrow U$. The flow of the system $\dot{x}(t) = A_l x(t) + B_l \mu(t)$ in

location $l \in L$ can then be denoted by $\Phi_l(x, t, \mu)$ 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 $T_H = (X, X_0', T \cup \mathcal{T}, \delta)$ with $X_0' := \{(l, x) \in X_0 \mid x \in \mathcal{I}_l\}$. For notational convenience, a transition relation $\rightarrow \subseteq X \times X$ between states of the transition system is defined as the union of two relations $\rightarrow_C, \rightarrow_D \subseteq X \times X$. The relation $\rightarrow_C$ describes transitions due to continuous flows, whereas $\rightarrow_D$ describes the transitions due to discrete jumps.

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

Some basic reachability notation is introduced next. 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 $\texttt{Post}_C(S)$ can be defined as: $\texttt{Post}_C(l, P) := \{(l, y) \in X \mid \exists x \in P : (l, x) \rightarrow_C (l, y)\}$; and $\texttt{Post}_C(S) := \{(l, y) \in X \mid \exists (l, x) \in S : (l, x) \rightarrow_C (l, y)\}$. Similarly, the set of *discrete successors* of (l, P) and S, denoted by $\texttt{Post}_D(l, P)$ and $\texttt{Post}_D(S)$ respectively, can be defined as: $\texttt{Post}_D(l, P) := \{(l', y) \in X \mid \exists x \in P : (l, x) \rightarrow_D (l', y)\}$; and $\texttt{Post}_D(S) := \{(l', y) \in X \mid \exists (l, x) \in S : (l, x) \rightarrow_D (l', y)\}$. For the thermostat example (see figure 1), and a set S with

$$S = \{(\texttt{Heat}, (t, T)) \in X \mid 1.5 \leq t \leq 2.5 \wedge 8.5 \leq T \leq 9.5\},$$

it can thus be computed that

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

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

Safety properties of systems are usually specified by partitioning the set of all states into safe and unsafe states. A system satisfies the safety properties if an unsafe state cannot be reached. We proceed to formalize this notion for linear hybrid systems. A property can be specified by a set of *unsafe locations* $L_u \subseteq L$ and a convex set $\mathcal{B} \subseteq \mathcal{X}$ of *unsafe continuous states*. The property is said to hold for the hybrid system H iff there is no valid trace from an initial state to some state in $\mathcal{B}$ while in an unsafe location. For the thermostat example, the set of unsafe continuous states $\mathcal{B}$ is defined as the set of states when the temperature drops below 4.5, that is: $\mathcal{B} = \{(t, T) \in [0, 100]^2 \mid T \leq 4.5\}$. The set of unsafe locations L_u is defined as $L_u = \{\texttt{Check}\}$, as the invariant in location `Cool` provides that the system cannot reach $\mathcal{B}$ in the `Cool` location. The `Heat` location is also not included in L_u, as the dynamics provide that $\mathcal{B}$ will not be reached while in the `Heat` location unless the system starts initially in $\mathcal{B}$.

Definition 3 (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*

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

Given a set of unsafe locations $L_u \subseteq L$ *and a convex set* $\mathcal{B} \subseteq \mathcal{X}$*, the set of unsafe states* $\mathcal{B}_X$ *can be defined as* $\mathcal{B}_X := \{(l, x) \in X \,|\, l \in L_u \wedge x \in \mathcal{B}\}$*. The* **verification problem** *then is:* $\texttt{Reach} \cap \mathcal{B}_X \stackrel{?}{=} \emptyset$.

In [1], 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.

2.3 Discrete Abstraction

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

Definition 4 (Abstract state-space) *Given an n-dimensional linear 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 $(l, \vec{b})$, where $l \in L$ and $\vec{b} \in \mathbb{B}^k$. The abstract state-space for a k-dimensional vector of linear predicates therefore is $Q_\Pi := L \times \mathbb{B}^k$.*

Figure 2 illustrates the abstraction of the continuous state-space for the thermostat example of figure 1. Ten predicates are used for the abstraction, namely:

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

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. Each box or line on the right hand side of figure 1 corresponds to a vector $\vec{b} \in \mathbb{B}^{10}$ for the predicates as specified in equation 1. 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 $\vec{b} \in \mathbb{B}^k$ for a vector of linear predicates Π the set of states of the continuous state-space that it represents can be computed given the following definition. For example, the vector $(0, 1, 0, 1, 1, 0, 1, 0, 0, 1)$ represents the set of predicates $t > 0, t \geq 0.5, t > 1, t \geq 2, t \leq 3, T > 4.5, T \geq 5, T > 6, T < 9$, and $T \leq 10$ given the vector of predicates Π as specified in equation (1), which represents the continuous state-space $\{(t, T) \in \mathbb{R}^2 \,|\, 2 \leq t \leq 3 \wedge 6 < T < 9\}$.

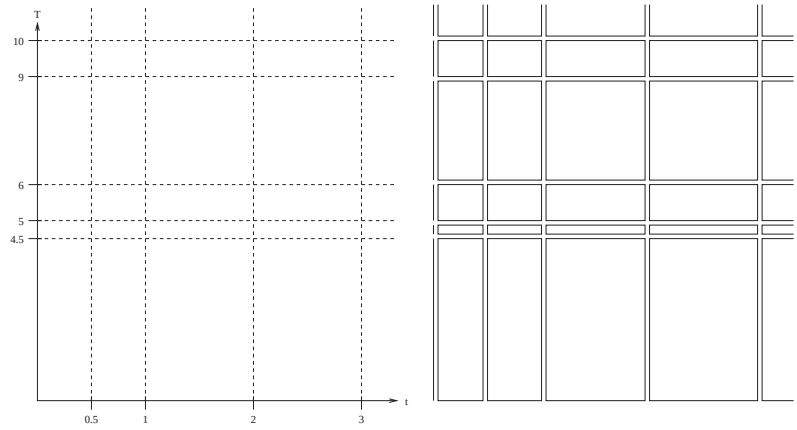

Fig. 2. Discrete abstraction of the continuous state-space for the thermostat model

Definition 5 (Concretization function) *A* **concretization function** C_Π *for a vector of linear predicates* $\Pi = (\pi_1, \ldots, \pi_k) \in (\mathcal{L}_n)^k$ *with* $C_\Pi : \mathbb{B}^k \to 2^{\mathbb{R}^n}$ *is defined as follows:* $C_\Pi(\vec{b}) := \{x \in \mathbb{R}^n \,|\, \forall i \in \{1, \ldots, k\} : \pi_i(x) = b_i\}$. *A vector* $\vec{b} \in \mathbb{B}^k$ *is said to be* **consistent** *with respect to a vector of linear predicates* $\Pi \in (\mathcal{L}_n)^k$*, iff* $C_\Pi(\vec{b}) \neq \emptyset$. *An abstract state* $(l, \vec{b}) \in Q_\Pi$ *is said to be* **consistent** *with respect to a vector of linear predicates* Π*, iff* $\vec{b}$ *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. However, 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 2 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.

The implementation of the verification tool 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 1 *Given a set of linear predicates* $\Pi \in (\mathcal{L}_n)^k$ *and a convex polyhedron* $\mathcal{X}$*, then for any* $\vec{b} \in \mathbb{B}^k$ $C_\Pi(\vec{b})$ *and* $C_\Pi(\vec{b}) \cap \mathcal{X}$ *represent convex polyhedra.*

The following definition formalizes the discrete transition system of a hybrid system using predicate abstraction.

Definition 6 (Discrete Abstraction) *An abstract system given a linear hybrid system* $H = (\mathcal{X}, L, X_0, f, I, T)$ *with respect to a vector of linear predicates* Π *is defined as the transition system* $H_\Pi = (Q, Q_0, \Sigma_T, \delta)$ *where*

- *the state-space of* H_Π *is* $Q = Q_\Pi$*;*
- *the set of initial states are those abstract states that overlap with concrete initial abstract states and the relevant invariant:* $Q_0 = \{(l, \vec{b}) \in Q_\Pi \mid \exists x \in C_\Pi(\vec{b}) \cap \mathcal{I}_l : (l, x) \in X_0\}$;
- *the set of labels of the transition system* Σ_T *is the union of the transition function* T

denoting transitions due to discrete switches and the symbol C denoting transitions due to continuous flow, i.e. $\Sigma_T = T \cup \{C\}$; *and*

- *the transition relation* $\delta \subseteq Q_\Pi \times \Sigma_T \times Q_\Pi$ *between states of the transition system includes both transitions due to discrete switches and due to continuous flow. It thus can be defined using the following two cases:*

$$\begin{aligned}(l,\vec{b}) \xrightarrow{\Pi}_C (l,\vec{b'}) :\iff\ & \exists t \in \mathcal{T}, \mu \in \mathcal{U}, x \in C_\Pi(\vec{b}) : \Phi_l(x,t,\mu) \in C_\Pi(\vec{b'}) \wedge \\ & \forall t' \in [0,t] : \Phi_l(x,t',\mu) \in I(l).\\ (l,\vec{b}) \xrightarrow{\Pi}_t (l',\vec{b'}) :\iff\ & t = (l,l',g,r) \in T \wedge \exists x \in C_\Pi(\vec{b}) : x \in g \wedge \\ & r(x) \in C_\Pi(\vec{b'}) \wedge r(x) \in I(l').\end{aligned}$$

For notational convenience, the abstract transition relation $\xrightarrow{\Pi} \subseteq Q_\Pi \times Q_\Pi$ is defined as the union of the following two relations $\xrightarrow{\Pi}_D, \xrightarrow{\Pi}_C \subseteq Q_\Pi \times Q_\Pi$. The relation $\xrightarrow{\Pi}_D$ represents transitions in the abstract state-space due to discrete jumps:

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

The successors of an abstract state $(l,\vec{b}) \in Q_\Pi$ and a set of abstract states $S \subseteq Q_\Pi$ by discrete jumps and by continuous flows, denoted respectively by $\texttt{Post}^\Pi_D(l,\vec{b})$, $\texttt{Post}^\Pi_D(S)$, $\texttt{Post}^\Pi_C(l,\vec{b})$, and $\texttt{Post}^\Pi_C(S)$ can be defined as:

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

Consider the abstract state $1 < t < 2 \wedge 9 \leq T \leq 10$ in location `Heat` for the thermostat model of figure 1, which is represented by the abstract state $(l,\vec{b}) = (\texttt{Heat}, (0,1,0,0,1,0,1,0,1,1))$ given Π as specified in equation (1). In this case, the following holds: $\texttt{Post}^\Pi_D(l,\vec{b}) = \{(\texttt{Cool},\vec{b})\}$, and $\texttt{Post}^\Pi_C(l,\vec{b}) = \{(l,\vec{b}),(l,(0,1,0,1,1,0,1,0,1,1))\}$, 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 7 (Abstract verification problem) *Given a linear hybrid system* $H = (\mathcal{X}, L, X_0, I, f, T)$ *and a vector of linear predicates* Π*, the set of reachable abstract states* $\texttt{Reach}_\Pi$ *is defined as:*

- $\texttt{Reach}^{(0)}_\Pi := Q_0$*;*
- $\texttt{Reach}^{(i+1)}_\Pi := \texttt{Post}^\Pi_D(\texttt{Reach}^{(i)}_\Pi) \cup \texttt{Post}^\Pi_C(\texttt{Reach}^{(i)}_\Pi)\ \forall i \geq 0$*; and*
- $\texttt{Reach}_\Pi := \bigcup_{i \geq 0} \texttt{Reach}^{(i)}_\Pi$.

Given a set of unsafe locations $L_u \subseteq L$ and a convex set $\mathcal{B} \subseteq \mathcal{X}$, the set $\mathcal{B}_\Pi$ is defined as $\mathcal{B}_\Pi := \{(l, \vec{b}) \in Q_\Pi \mid l \in L_u \wedge C_\Pi(\vec{b}) \cap \mathcal{B} \neq \emptyset\}$. The verification problem is: $\texttt{Reach}_\Pi \cap \mathcal{B}_\Pi \stackrel{?}{=} \emptyset$.

It can be proven that predicate abstraction of linear hybrid systems computes an over-approximation of the set of reachable states of the concrete system. This is formalized in the following lemma which is proven in [30]:

Lemma 2 *Given a linear hybrid system $H = (\mathcal{X}, L, X_0, I, f, T)$ and a vector of linear predicates Π, the following holds: $\texttt{Reach} \subseteq \{(l, x) \in X \mid \exists (l, \vec{b}) \in \texttt{Reach}_\Pi : x \in C_\Pi(\vec{b}) \cap \mathcal{I}_l\}$.*

2.4 Searching the Abstract State-Space

We implemented an on-the-fly search of the abstract state-space. The search in the abstract state-space can be performed in a variety of ways. Our goal is to make the discovery of counter-examples in the abstract state-space given a reachability property as fast as possible. In the case that the property is true we need to search the entire reachable abstract sub-space. We perform a DFS, which usually does not find a shortest counter-example possible. On the other hand, it only stores the current trace of abstract states from an initial abstract state on a stack. In case we find an abstract state that violates the property, the stack contents represent the counter-example. This is generally much more memory efficient than BFS.

We give a priority to computing discrete successors rather than continuous successors, as this is generally much faster. Computing discrete successors is relatively straightforward, and involves computing weakest preconditions, and checking non-emptiness of intersection of polyhedral sets. For computing continuous successors of an abstract state A, we compute the polyhedral slices of states reachable at fixed times $r, 2r, 3r, \ldots$ for a suitably chosen r, and then, compute the convex-hull of all these polyhedra to over-approximate the set of all states reachable from A. We are only interested in checking if this set intersects with a new abstract state. This approach has many benefits compared to the traditional approach of computing approximations of reachable sets, one of them being the fact that the expensive operation of computing continuous successors is applied only to abstract states, and not to intermediate polyhedra of unpredictable shapes and complexities. In [4] we proved soundness of our search algorithm:

Theorem 1 *If the search algorithm terminates and reports that the abstract system is safe, then the corresponding concrete system is also safe.*

We include various optimization techniques in the search strategy. In the following, we describe one optimization that is being exploited during the counter-example analysis. For each concrete counter-example in the concrete hybrid system, there exists an equivalent counter-example that has the additional constraint that there are no two consecutive transitions due

to continuous flow. This is due to the additivity of flows of hybrid systems, namely

$$(l, x) \rightarrow_C (l, x') \wedge (l, x') \rightarrow_C (l, x'') \Rightarrow (l, x) \rightarrow_C (l, x'').$$

We are hence searching only for counter-examples in the abstract system that do not have two consecutive transitions due to continuous flow. By enforcing this additional constraint we eliminate some spurious counter-examples that could have been found otherwise in the abstract transition system. The spurious counter-examples that are eliminated are due to the fact that $(l, \vec{b}) \xrightarrow{\Pi}_C (l, \vec{b'})$ and $(l, \vec{b'}) \xrightarrow{\Pi}_C (l, \vec{b''})$ does *not* imply that $(l, \vec{b}) \xrightarrow{\Pi}_C (l, \vec{b''})$. Hence, we are in fact not computing the whole relation $\xrightarrow{\Pi}_C$ as it was defined above, but only a part of it without compromising the conservativeness of our approach. As shown in [4], the ten predicates specified in equation (1) can be used to prove safety of the thermostat controller. The search discovers 35 reachable abstract states. For the sake of brevity we omit the details of the reachability analysis and various other optimization techniques in this paper. A more detailed description can be found in [6].

3 Counter-Example Analysis

The success of the predicate abstraction scheme as outlined in the previous section crucially depends on the choice of the predicates used for abstraction. This and the following section describe methods to identify such predicates automatically by analyzing spurious counter-examples generated by the search in the abstract state-space. Counter-example guided refinement of abstractions has been used in multiple contexts before, for instance, to identify the relevant timing constraints in verification of timed automata [8], to identify the relevant boolean predicates in verification of C programs [10], and to identify the relevant variables in symbolic model checking [16]. In the following the basic techniques for analyzing counter-examples and techniques for discovering new predicates that will rule out spurious counter-examples are presented.

3.1 Forward Analysis

An abstract counter-example consists of a sequence of abstract states and transitions leading from an initial state to a state violating the property. The analysis problem, then, is to check if the corresponding sequence of locations and continuous states can be traversed in the concrete system. This analysis problem is solved by a forward search from the initial abstract state following the given counter-example in the abstract state-space. The analysis relies on techniques for polyhedral approximations of the reachable sets under continuous dynamics. We first define the notion of abstract paths.

Definition 8 (Abstract path) *An abstract path p of length $n \geq 0$ in the abstract state-space given by the vector of predicates Π is a pair $(\vec{a}, \vec{t}) \in (Q_\Pi)^{n+1} \times (\Sigma_T)^n$, such that:*

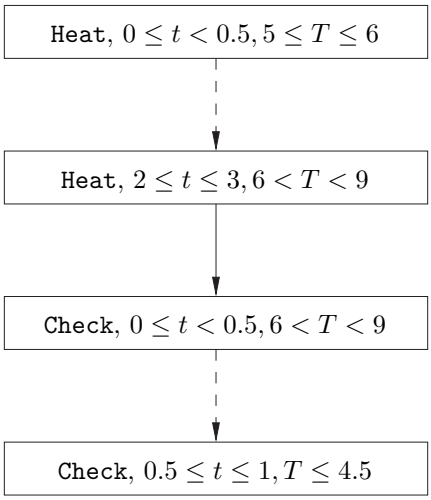

Fig. 3. An abstract path for the thermostat model of figure 1 using only those predicates mentioned in the model description

$\vec{a} = (a_0, \ldots, a_n)$ *and* $\vec{t} = (t_0, \ldots, t_{n-1})$ *with* $t_i \in \Sigma_T$, $a_0 = (l_0, \vec{b_0}) \in Q_0$, *and* $\forall 0 \leq i \leq n-1 : a_i \xrightarrow{\Pi}_{t_i} a_{i+1}$. *The set of abstract paths of length* n *given by the vector of predicates* Π *is denoted by* $\mathcal{P}_n^{\Pi}$.

Consider figure 3 which illustrates an abstract path of length 3 for the thermostat example of figure 1 in the abstract state-space defined by the predicates mentioned in the model. That is, the abstract state-space is partitioned according to all predicates mentioned in equation (1) except $t \leq 0$. The abstract path in figure 3 contains three transitions, two of which are due to continuous flow and graphically represented by a dashed edge, and one transition due to a discrete switch drawn by a solid edge.

Given a linear hybrid system H, a set of unsafe locations L_u and a set of unsafe continuous states $\mathcal{B} \subseteq \mathcal{X}$, it can now be formally defined what a counter-example in the abstract state-space is.

Definition 9 (Counter-example) *A counter-example of length* n *is an abstract path* $p = (\vec{a}, \vec{t}) = ((a_0, \ldots, a_n), (t_0\ , \ldots, t_{n-1}))$ *of length* n, *such that* $a_n = (l_n, \vec{b_n})$ *is a violation of the property to be proven; that is,* $l_n \in L_u \wedge C_{\Pi}(\vec{b_n}) \cap \mathcal{B} \neq \emptyset$. *The sequence of abstract states* $\vec{a} = (a_0, \ldots, a_n)$ *of a counter-example* $p = (\vec{a}, \vec{t})$ *is called an unlabeled counter-example.*

Consider again the abstract path in figure 3. As the path ends in an abstract state with a continuous state-space $T \leq 4.5$, which is unsafe, while in the unsafe location `Check` $\in L_u$, this path is therefore a counter-example in the abstract state-space defined by the predicates mentioned in the model. However, as mentioned earlier, the thermostat model is safe. The compressed set of predicates considered here is not enough though to prove the thermostat model safe. As the thermostat system is safe, it is clear that this counter-example has to be spurious, which will be shown subsequently.

The counter-example analysis problem is twofold. The first objective is to check whether a counter-example in the abstract state-space corresponds to a counter-example in the concrete state-space. In case that this analysis finds that this particular counter-example cannot be traversed in the concrete system, the analysis procedure should identify one or more new predicates that would rule out *closely related* counter-examples in the refined abstract state-space. The refined abstract state-space is defined by adding these predicates to the previous set of predicates used in the abstract state-space search. We would like some guarantee of convergence that the same counter-example is not discovered repeatedly. The notion of refinement between abstract paths is defined to formalize the concept of closely related abstract paths. First, the notion of refinement is defined for vectors of predicates.

Definition 10 (Refinement of a vector of predicates) *A vector of predicates* $\Pi' = (\pi'_1, \ldots, \pi'_{k'}) \in (\mathcal{L}_n)^{k'}$ *refines another vector of predicates* $\Pi = (\pi_1, \ldots, \pi_k) \in (\mathcal{L}_n)^k$*, iff* $\forall 1 \leq i \leq k \, \exists j \in \{1, \ldots, k'\} : \pi_i = \pi'_j$.

From now on, when the thermostat model is used for illustrative purposes in this paper two vectors of predicates will be considered. The vector of predicates as defined in equation (1) is denoted by Π, whereas $\hat{\Pi}$ denotes only those predicates that are mentioned in the model of the thermostat example in figure 1 itself. Therefore, $\hat{\Pi}$ contains all predicates in Π except for $t \leq 0$. Thus, Π is a refinement of $\hat{\Pi}$ according to the above definition. Next, the notion of refinement is defined for abstract states. An abstract state is considered a refinement of another abstract state, if the two have the same locations, and the concretization of the former is covered by the latter.

Definition 11 (Refinement of abstract states) *An abstract state* $a' = (l', \vec{b'}) \in Q_{\Pi'}$ *for the vector of predicates* Π' *refines another abstract state* $a = (l, \vec{b}) \in Q_{\Pi}$ *for the vector of predicates* Π*, iff* $l = l'$ *and* $C_{\Pi'}(\vec{b'}) \subseteq C_{\Pi}(\vec{b})$.

The abstract state $t = 0 \wedge 5 \leq T \leq 6$ in the location `Heat` location for the vector of predicates Π is a refinement of $0 \leq t < 0.5 \wedge 5 \leq T \leq 6$ in the same location for the vector of predicates $\hat{\Pi}$. The previous two definitions of refinement are now combined to define refinements of abstract paths. It is required that each abstract state on the path is refined while following the same transitions. Formally, it is defined as follows:

Definition 12 (Refinement of abstract paths) *An abstract path* $p' = (\vec{a'}, \vec{t'}) = ((a'_0, \ldots, a'_n), (t'_0, \ldots, t'_{n-1})) \in \mathcal{P}_n^{\Pi'}$ *for a vector of predicates* Π' *refines another abstract path* $p = (\vec{a}, \vec{t}) = ((a_0, \ldots, a_n), (t_0, \ldots, t_{n-1})) \in \mathcal{P}_n^{\Pi}$ *for a vector of predicates* Π*, with* $a_i = (l_i, \vec{b_i})$ *and* $a'_i = (l'_i, \vec{b'_i})$*, iff* Π' *refines* Π*,* $\forall 0 \leq i \leq n : a'_i$ *refines* a_i*, and* $\forall 0 \leq i \leq n-1 : t'_i = t_i$.

During the counter-example analysis specialized $\texttt{Pre} : Q_\Pi \times \Sigma_T \times Q_\Pi \to 2^{\mathcal{X}}$ and $\texttt{Post} : 2^{\mathcal{X}} \times \Sigma_T \times Q_\Pi \to 2^{\mathcal{X}}$ functions are defined that will either only consider a particular abstract state or the concretely reachable state-space rather than the whole continuous state-space $\mathcal{X}$. The computation of these takes into consideration the concretization of the abstract state, as well as the invariants and guards of the system. The functions $\texttt{Pre} : Q_\Pi \times \Sigma_T \times Q_\Pi \to 2^{\mathcal{X}}$

and $\texttt{Post} : 2^{\mathcal{X}} \times \Sigma_T \times Q_\Pi \to 2^{\mathcal{X}}$ with $a = (l, \vec{b})$ and $a' = (l', \vec{b'})$ are defined as:

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

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

The counter-example analysis algorithm is presented in Algorithm 1. The set R_0 is the part of the initial state-space X_0 that is covered by the abstract state $(l, \vec{b_0})$. Then, the concretely reachable state-space of each abstract state of the counter-example of length n in the abstract state-space is computed. This process is illustrated in figure 4. For each $1 \leq i \leq n$ the analysis computes R_i as the reachable region after i transitions according to the counter-example. It is hence clear that if $R_i = \emptyset$ for some i then the counter-example is spurious. The shaded sub-spaces in figure 4 represent the concretely reachable regions R_i.

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

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

Following algorithm 1 for the counter-example in figure 3 for the thermostat model of figure 1, the concretely reachable sub-spaces of the abstract states following this particular counter-example are computed. The first abstract state $a_0 = (l_0, \vec{b_0})$ in the counter-example represents the continuous state-space $0 \leq t < 0.5 \wedge 5 \leq T \leq 6$ while in location `Heat`. Given the constraints on the initial sets however, only $t = 0 \wedge 5 \leq T \leq 6$ are concretely possible in a_0. Thus, the following holds: $R_0 = \{(\texttt{Heat}, (0, T)) \in X \,|\, 5 \leq T \leq 6\}$. Consider now the continuous transition that leads from a_0 to a_1, which is $2 \leq t \leq 3 \wedge 6 < T < 9$ in the continuous state-space while in location `Heat`. It is thus evident that a_1 cannot be reached from R_0, that is $R_1 = \emptyset$. This proves that the counter-example of figure 3 is indeed spurious.

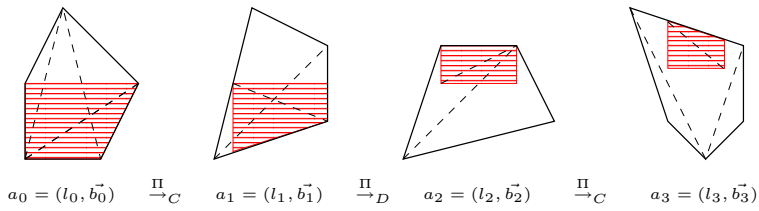

Fig. 4. A counter-example of length 3. For each abstract state $a_i = (l_i, \vec{b_i})$ the concrete continuous state-space $C_\Pi(\vec{b_i})$ that it represents is illustrated.

In case that the analysis finds that the counter-example is spurious, the counter-example is then used to find new predicates. These new predicates should be added to the current set of predicates used in the predicate abstraction model checker, in order to disallow closely related counter-examples to reappear. Consider a counter-example $p \in \mathcal{P}_n^\Pi$, such that $R_{k+1} = \emptyset$ and $R_k \neq \emptyset$ for $0 \leq k < n$. The transition t_k of the counter-example p is called the *failing transition*. Then the following lemma can be proven (see [30]):

Lemma 3 *Given a counter-example* $p = (\vec{a}, \vec{t}) = ((a_0, \ldots, a_n), (t_0, \ldots, t_{n-1})) \in \mathcal{P}_n^\Pi$ *where* t_k *is the failing transition, the following holds:* $R_k \cap \mathtt{Pre}(a_k, t_k, a_{k+1}) = \emptyset$.

New predicates are supposed to be added to the vector Π, so that the refined vector Π' does not allow a refined (unlabeled) counter-example of p to reappear. Consider a strategy that adds predicates to the set Π that correspond to a separation of R_k from $\mathtt{Pre}(a_k, t_k, a_{k+1})$ for the failing transition t_k. This means that the analysis is looking for a refined set of predicates Π' of Π, such that every refined abstract state intersects at most with one of the two sets R_k and $\mathtt{Pre}(a_k, t_k, a_{k+1})$. A notion of separation is defined in terms of polyhedral sets, since the set of reachable states is approximated by polyhedral slices in the implementation of the tool. It should be noted here that under-approximations of the reachable sets of states are used during the analysis of counter-examples while over-approximations of the reachable sets of states are used during the search in the abstract state-space.

Definition 13 (Separating predicates) *Assume that* $\mathcal{P} = \{P_1, \ldots, P_n\}$ *and* $\mathcal{Q} = \{Q_1, \ldots, Q_m\}$ *denote two disjoint sets of convex polyhedra. The union of all polyhedra in* $\mathcal{P}$ *and* Q *are respectively denoted by* $\bigcup\mathcal{P}$ *and* $\bigcup\mathcal{Q}$. *A finite vector of linear predicates* $\Pi = (\pi_1, \pi_2, \ldots, \pi_k)$ *separates* $\mathcal{P}$ *and* $\mathcal{Q}$ *iff for all* $\vec{b} \in \mathbb{B}^k$, *at least one of the two sets* $(C_\Pi(\vec{b}) \cap \mathcal{X} \cap \bigcup\mathcal{P})$ *and* $(C_\Pi(\vec{b}) \cap \mathcal{X} \cap \bigcup\mathcal{Q})$ *is empty.*

The predicates in Π are called *separating predicates*. Note that such a vector Π always exists, [1] but it is not unique.

Theorem 2 *Assume a counter-example* $p \in \mathcal{P}_n^\Pi$ *for a vector of predicates* Π *such that* t_k *is the failing transition. If* Π' *refines* Π *and additionally contains predicates corresponding to a separation of* R_k *from* $\mathtt{Pre}(a_k, t_k, a_{k+1})$, *and a refined counter-example* $p' \in \mathcal{P}_n^{\Pi'}$ *of* p *is*

[1] It is easy to see that the analysis can simply take the linear constraints of all polyhedra from P or from Q to determine Π. However, as the size of the refined abstract state-space grows exponentially with the number of new predicates, it is advantageous to try to include as few predicates as possible.

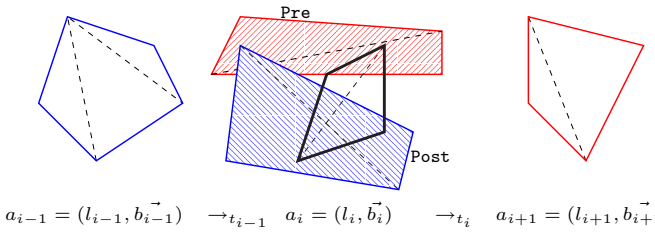

Fig. 5. The abstract state a_i is locally infeasible

found, then there exists a failing transition t_j in p', such that $j < k$.

As a single counter-example p is of finite length, the above theorem guarantees that after a finite number of iterations, a refinement of p will not be possible anymore. The proof is omitted for the sake of brevity and can be found in [30].

3.2 Locally Infeasible Abstract States

This section presents a second counter-example analysis algorithm. The purpose of this algorithm is to check a counter-example quickly for a common cause of spurious counter-examples. It is also shown that this analysis produces new predicates with stronger implications for subsequent searches in the refined abstract state-space.

Definition 14 *For a path $p = (\vec{a}, \vec{t}) \in \mathcal{P}^{\Pi}_{n+1}$ given the vector of predicates Π, with $\vec{a} = (a_0, \ldots, a_{n+1}) = ((l_0, \vec{b_0}), \ldots, (l_{n+1}, \vec{b_{n+1}}))$ and $\vec{t} = (t_0, \ldots, t_n)$, an abstract state a_i for $1 \leq i \leq n$ is called* **locally infeasible**, *iff $\texttt{Post}(C_\Pi(\vec{b_{i-1}}) \cap \mathcal{X}, t_{i-1}, a_i) \cap \texttt{Pre}(a_i, t_i, a_{i+1}) = \emptyset$.*

The definition of locally infeasible abstract states is illustrated in figure 5. The figure shows a locally infeasible abstract state a_i. Although the `Pre` of a_{i+1} and the `Post` of a_{i-1} intersect, they do not intersect within a_i. Therefore, this abstract counter-example does not correspond to a concrete counter-example. The detection of locally infeasible abstract states can be implemented in a straight-forward fashion. In addition, new predicates can easily be computed that will disallow refined counter-examples. If a state a_i is locally infeasible, then the analysis can use the fact that the implemented optimization technique guarantees that either t_{i-1} or t_i is a discrete transition. If t_{i-1} is discrete, one reasonable choice is to use the predicates corresponding to the constraints of the polyhedral sets representing $\texttt{Post}(C_\Pi(\vec{b_{i-1}}) \cap \mathcal{X}, t_{i-1}, a_i)$ in the refined search. Otherwise, a possible approach is to use the predicates corresponding to $\texttt{Pre}(a_i, t_i, a_{i+1})$ in the refined search. This strategy of picking new predicates is denoted with `LocalStrategy` [2] from now on.

Consider the thermostat example of figure 1 for a vector of predicates $\Pi = (t \leq 1, t \leq 3, T \geq 5, T \leq 6)$. A possible path in this abstract state-space is to start in the abstract state a_0 with continuous state-space $0 \leq t \leq 1 \wedge 5 \leq T \leq 6$ while in location `Heat`, then enter the

[2] It is preferred to use predicates computed on the basis of discrete transitions, as these can be computed more easily and more exactly.

abstract state a_1 with $1 < t \leq 3 \wedge 5 \leq T \leq 6$ in location `Heat` following a transition t_c due to continuous flow, and end up in the abstract state a_2 with $0 \leq t \leq 1 \wedge 5 \leq T \leq 6$ in location `Check` following a transition t_d due to a discrete switch. The following shows that the abstract state a_1 is locally infeasible:

$$\begin{aligned}\texttt{Post}(a_0, t_c, a_1) &= \{(\texttt{Heat}, (t, T)) \in X \mid 1 < t \leq 1.5, 5 \leq T \leq 6, T \geq 2(t-1)+5\};\\ \texttt{Pre}(a_1, t_d, a_2) &= \{(\texttt{Heat}, (t, T)) \in X \mid 2 \leq t \leq 3, 5 \leq T \leq 6\};\end{aligned}$$

thus, $\texttt{Post}(a_0, t_c, a_1) \cap \texttt{Pre}(a_1, t_d, a_2) = \emptyset$ which implies that a_1 is indeed locally infeasible. Using `LocalStrategy` as described above, we add the predicate $t \geq 2$ to the vector of predicates, as this is the only new predicate in $\texttt{Pre}(a_1, t_d, a_2)$. This implies that the guard condition of this transition is important for the verification of this particular safety property.

The following theorem can now be proven about using the strategy `LocalStrategy` in case a locally infeasible abstract state is found. The theorem formalizes that this strategy guarantees that a refinement of the (unlabeled) counter-example will not be found in subsequent searches.

Theorem 3 *Assume a counter-example $p \in \mathcal{P}_n^{\Pi}$ for a vector of predicates Π, such that there is a locally infeasible abstract state a_i in p. A search in the refined abstract state-space given by the strategy* `LocalStrategy` *to find new predicates will not find a counter-example that is a refinement of p.*

It can also be shown that a search in a refined abstract state-space for a counter-example with a locally infeasible abstract state cannot discover a refined counter-example for a certain class of linear hybrid systems. For this result it is required that the linear hybrid system have at most one transition between any pair of locations. Additionally, it is required that the hybrid system have no self-transition, that is there is no transition with the same source and target location. A more formal description of this result can be found in [5].

4 Computing separating predicates

The previous sections describe two counter-example analysis algorithms. If the counter-example is found to be infeasible, then the analysis should 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. This section presents a greedy strategy for identifying the separating predicates. After discovering new predicates, these predicates can then be added to the set of predicates used before, and the search can then be rerun in the refined abstract state-space defined by the enriched predicate set.

4.1 Separating two disjoint convex polyhedra

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

Lemma 4 *The polyhedron Q is contained in $\mathcal{H}$ and the polyhedron P is contained in $\overline{\mathcal{H}}$.*

As a remark, lemma 4 also holds for any half-space which is normal to $s(p^*,q^*)$ and passes through an arbitrary point in $s(p^*,q^*)$. Hence, any such half-space can be used to define a separating predicate. To compute $d(P,Q)$ as well as p^* and q^*, there exist efficient algorithms [12] which take time $\mathcal{O}(K_P + K_Q)$ where K_P and K_Q are the number of vertices of P and Q.[3]

4.2 Separating two disjoint sets of convex polyhedra

This section proceeds with the problem of finding a set Π of separating predicates for two sets of convex polyhedra $\mathcal{P}_1$ and $\mathcal{P}_2$. In order to keep the size of the abstract state space as small as possible, the analysis tries to find Π with the smallest number of predicates. Many related polyhedral separation problems have been considered in the literature (see [22,23,34] and references therein). However, the solutions proposed in these works are only for two and three dimensional polyhedra. On the other hand, even in low dimensions most separation problems were shown to be intractably hard. In three dimensions the problem of finding a minimum facet-separator for two polyhedral solids is NP-complete [20]. Therefore, in this work the objective is not to find an optimal solution but to develop some heuristics which are effective on the problem of separating reachable sets of hybrid systems for abstraction refinement purposes.

The solution is based on the following observation. Given two set of polyhedra $\mathcal{P}_1$ and $\mathcal{P}_2$, if the convex hulls of $\mathcal{P}_1$ and $\mathcal{P}_2$ are disjoint, then one can apply the method presented in the previous section to find a separating predicate. If the convex hulls intersect, it is clear that $\mathcal{P}_1$ and $\mathcal{P}_2$ cannot be separated by a single hyperplane. The main idea is to divide $\mathcal{P}_1$ and $\mathcal{P}_2$ into subsets of polyhedra such that their convex hulls do not intersect allowing to find a separating predicate. The procedure of subdivision can be performed in a hierarchical way. Initially, all polyhedra in $\mathcal{P}_1$ and $\mathcal{P}_2$ are recursively subdivided until the convex hulls are pairwise disjoint. Moreover, for efficiency purposes, instead of convex hulls, approximations by non-axis-aligned

[3] The problem of finding a separating predicate for two disjoint polyhedra can also be formulated as a linear programming problem and thus solved in polynomial time.

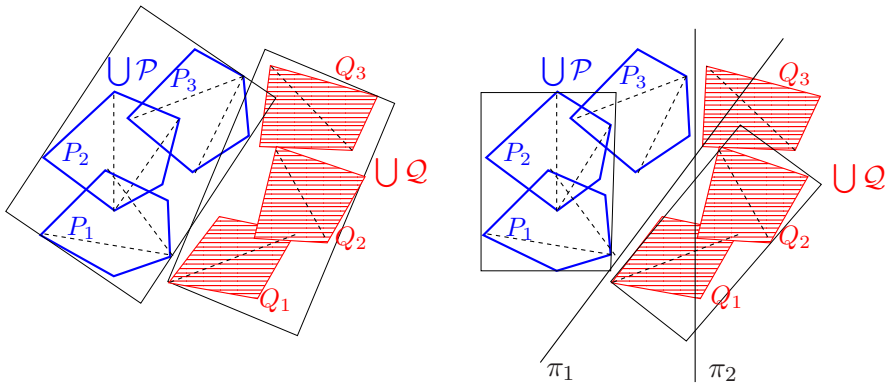

Fig. 6. Illustrating the greedy algorithm that separates sets of polyhedra

bounding boxes are used which are easier to compute and test for overlaps (see figure 6). The figure shows a case where subdividing based on non-axis-aligned bounding boxes the sets $\mathcal{P} = \{P_1, P_2, P_3\}$ and $\mathcal{Q} = \{Q_1, Q_2, Q_3\}$ respectively into $\{P_1, P_2\}$, $\{P_3\}$ and $\{Q_1, Q_2\}$, $\{Q_3\}$ allows to find two separating predicates π_1 and π_2. One way of computing tight fitting bounding boxes is to align the axes of the box in the directions along which the vertices of the polyhedra tend to lie. From the vertices of the polyhedra the matrix of covariance can be determined and its largest eigenvectors can be taken to define the orientation of the box.

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

As one can see from line 4, a greedy strategy is used to choose separating predicates, that is the predicate that can separate the largest number of polyhedra is selected. An alternative selection criterion is to maximize the volume of separable polyhedra. The goal of line 5 is to exclude the subsets of $\bigcup\mathcal{P}_1$ and $\bigcup\mathcal{P}_2$ that the selected predicate π_m can separate. Indeed, if one of the sets $Int(\mathcal{P}_1, \pi_m)$ and $Int(\mathcal{P}_2, \pi_m)$ is empty, then either $\mathcal{P}_1$ or $\mathcal{P}_2$ lies entirely outside the half-space $\mathcal{H}(\pi_m)$. This means that the predicate π_m can separate a part of one set from the other, and the algorithm only needs to continue with the remaining part.

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

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

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

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

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

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

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

line passing through the two most distant points. [4] It is not easy to know which option is better (in terms of number of resulting predicates), and often the first one is preferred since the splitting hyperplane is easier to compute. Finally, the following lemma can be used to achieve better efficiency.

Lemma 5 *If a set of predicates* Π *separates the boundaries of* $\mathcal{P}_1$ *and* $\mathcal{P}_2$ *then it separates* $\mathcal{P}_1$ *and* $\mathcal{P}_2$*.*

To prove the lemma, it is remarked that Π separates $\mathcal{P}_1$ and $\mathcal{P}_2$ iff any line segment between a point in $\mathcal{P}_1$ and another point in $\mathcal{P}_2$ intersects with the hyperplane of at least one predicate in Π. Hence, if Π separates the boundaries of $\mathcal{P}_1$ and $\mathcal{P}_2$, then it separates $\mathcal{P}_1$ and $\mathcal{P}_2$ since any line segment connecting points in the interior of two disjoint sets must cross the boundaries of both sets.

Using lemma 5, only some boundary layers of $\mathcal{P}_1$ and $\mathcal{P}_2$ can be considered instead of the whole sets, which allows to obtain tighter convex approximations and thus requires less splitting. To extract a boundary layer for $\mathcal{P}_1$ and $\mathcal{P}_2$, begin by triangulating the two sets. Let B be the bounding box of $\mathcal{P}_1 \cup \mathcal{P}_2$. The subset G of $B \setminus (\bigcup \mathcal{P}_1 \cup \bigcup \mathcal{P}_2)$ that has a common boundary with both $\mathcal{P}_1$ and $\mathcal{P}_2$ is called the separation space. The boundary layers $\mathcal{P}'_1$ and $\mathcal{P}'_2$ are chosen as the sets of simplices in the triangulations of $\mathcal{P}_1$ and $\mathcal{P}_2$ which are adjacent to the separation space G. Intuitively, the hyperplanes of Π form a separating surface inside G; therefore, it suffices to use algorithm 2 to separate $\mathcal{P}'_1$ and $\mathcal{P}'_2$.

[4] These ideas are inspired by collision detection techniques in robotics and computer graphics [29].

4.3 The Thermostat Example

This section demonstrates the global counter-example analysis algorithm as well as the procedure to separate two disjoint sets of polyhedra using the thermostat example as described in figure 1. For purposes of illustration, the verification is started with the predicates mentioned in the model $\hat{\Pi}$; that means, we are considering all predicates mentioned in equation (1) except $t \leq 0$, which is not sufficient to prove safety.

The first iteration of the algorithm produces a spurious counter-example of length 7 after 11 abstract states have been discovered by the search of the abstract state-space. The separation routine suggests the following four linear predicate to refine the abstract state-space:

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

Please notice the last suggested predicate and its similarity to the predicate $t \leq 0$ considering the normal range $5 \leq T \leq 10$. The model designer may have been able to use this suggested set of predicates to refine the abstract state space by adding the predicate $t \leq 0$. Following the example, after refining the predicates with the help of these four predicates, the system still finds a spurious counter-example, and suggests four more predicates. In a third round, the system generates eleven more predicates after discovering another spurious counter-example, one of which is `0.0139043*T + 0.999903*t <= 0.152558`. The total set of 28 predicates is then in the following iteration enough to prove the thermostat example safe. The search in the abstract state-space finds 358 reachable abstract states. This compares to the ten predicates of equation (1) that are sufficient to prove safety while discovering 35 reachable abstract states.

5 Implementation Issues

This section presents algorithms for the validation analysis of counter-examples encountered during the search of the abstract state-space as presented in the preceding sections and a greedy polyhedral separation routine to discover new predicates. The algorithms as presented earlier in this paper are inspired by similar techniques used for the analysis of counter-examples encountered during program abstraction of discrete computer models.

However, computationally, it is often not possible to compute the `Post`-sets corresponding to the statement $R_i = \texttt{Post}(R_{i-1}, t_{i-1}, a_i)$ in algorithm 1 exactly, and approximations are needed. Similarly, the `Pre`- and `Post`-sets used in the definition of locally infeasible abstract states cannot be computed precisely, but rather need to be approximated. This section explores the implementation issues that arise due to this constraint which is a significant

additional constraint on the computation method.

Since the discrete program abstraction methods can compute these sets of reachable states exactly, they have a significant advantage in picking new predicates based on such a counter-example analysis. This is one reason why the idea of a second analysis algorithm (the local feasibility checker) has not surfaced in previous program abstraction methodologies. It is worth mentioning here that the previously mentioned CEGAR algorithm implementation for hybrid systems also provides a similar local feasibility check [15]. The authors describe certain advantages of considering *fragments* of a counter-example compared to the whole counter-example. Fragments are sub-paths of unspecified length of the original counter-example. Thus, the here presented local feasibility check can be seen as a particular fragment. However, the authors only provide an experimental description why fragments can be interesting to consider in this approximation-based counter-example analysis methodology without providing a formal comparison of the expected relative strength of predicates as described earlier in this paper.

First, consider the order of the analysis algorithms. Since the local feasibility checker provides a fast and reliable way to eliminate a common spurious counter-example pattern, it is clearly advantageous to perform this algorithm first. The algorithm should only raise a flag, if the counter-example to be analyzed is clearly spurious. Therefore, the implementation actually computes *over-approximations* of the reachable `Pre`- and `Post`-sets that are due to continuous flow.[5] Using an over-approximated polyhedral set computation, one can guarantee that a problem will only be detected if the counter-example is indeed spurious.

In order to keep the approximation tight in this analysis check, the implementation actually uses by default a 5-times smaller time-step than was used during the search of the abstract state-space. However, this finer precision factor can be changed by the user as a parameter to the verification tool. Again, a tradeoff between precision on the one hand and space and time considerations on the other has to be made.

Now, consider the forward analysis algorithm as presented in section 3.1. In contrast to the local feasibility checking algorithm, this analysis algorithm actually has two main contributions: Firstly, it should flag a problem if a spurious counter-example is found. In this case, it should also provide the greedy separation routine of section 4 with sets of polyhedra to be separated. Secondly, however, if the abstract counter-example corresponds to a concrete counter-example, it should be able to provide a witness trace in the concrete hybrid system model proving the validity of the counter-example and a witness trace of the violation of the property at hand. Since the polyhedral sets need to be approximated, it cannot be guaranteed that an abstract counter-example is identified as spurious if and only if it really is spurious. Given these requirements, it is thus clear that the forward search analysis algorithm needs to compute *under-approximations* of the reachable sets of states (for

[5] It should be noted, that it is possible to compute exact polyhedral sets for transitions due to discrete jumps. As mentioned earlier, it is guaranteed that at least one of the two considered transitions for the local feasibility check is due to a discrete jump.

transitions due to continuous flow). One way of implementing such an under-approximated analysis algorithm is to keep track only of particular slices of the reachable sets of states. The algorithm 3 presents such an approach and should be understood as one possible implementation of algorithm 1. An alternative would be to implement a procedure that computes under-approximated flow-pipes between the slices.

Algorithm 3 Under-approximation analysis of a counter-example

$\mathcal{R}_0 = \{P \subseteq \mathcal{X} | P = C_\Pi(\vec{b_0}) \cap \{x \in \mathcal{I}_{l_0} | (l_0, x) \in X_0\}\}$
for all $1 \leq i \leq n$ **do**
  if t_{i-1} is discrete **then**
    $\mathcal{R}_i = \bigcup_{P \in \mathcal{R}_{i-1}} \{\texttt{Post}(P, t_{i-1}, a_i)\} \setminus \{\emptyset\}$
  else
    $\mathcal{R}_i = \bigcup_{P \in \mathcal{R}_{i-1}} \{\texttt{Post}(P, k \cdot r, a_i) | k \geq 1\} \setminus \{\emptyset\}$
  end if
  if $\mathcal{R}_i = \emptyset$ **then**
    return "Counter-example is probably spurious!"
  end if
end for
return "Counter-example corresponds to a concrete trace!"

The algorithm computes for each abstract state a set of polyhedral slices $\mathcal{R}_i$ representing an under-approximation of the forward analysis as described in algorithm 1. The initial set $\mathcal{R}_0$ of polyhedra is initialized by exactly one polyhedron representing the initial states of the initial abstract state mentioned in the counter-example. For the following abstract states on the counter-example, the set of under-approximated polyhedra is computed as follows. If the transition is discrete, the algorithm computes the image of the enabled states for each polyhedron in the set $\mathcal{R}_{i-1}$. However, if the transition is continuous, the algorithm computes for each polyhedron in the set $\mathcal{R}_{i-1}$ the slices at time $r, 2r, 3r, \ldots$. These slices can be computed exactly. Therefore, the set of these polyhedra constitutes an under-approximation of the reachable set of states. This analysis algorithm then checks whether any $\mathcal{R}_i$ is empty, which signals that the performed under-approximation could not produce a witness trace. This means that the counter-example *may be* spurious without guaranteeing this result. However, if the analysis algorithm finds that all sets including $\mathcal{R}_n$ are non-empty, it has thus found proof that this counter-example corresponds to a counter-example in the concrete hybrid system.

There are certain optimizations that can be helpful in this analysis algorithm. First of all, the search in the abstract state-space finds the first possible time instance τ that an abstract state can be reached by continuous flow from another one. This information can now be used to reduce the amount of computations necessary in the analysis algorithm. The algorithm needs to only compute the slices at time $k \cdot r$ for $k \geq \frac{\tau}{r}$ since it is known that previous slices will definitely not intersect the next abstract state. Secondly, for the computation to be feasible, one can prescribe a limit on the number of slices that one wants to consider for continuous time flow at each instance of continuous flow. This parameter is also under user control in the implementation of the tool.

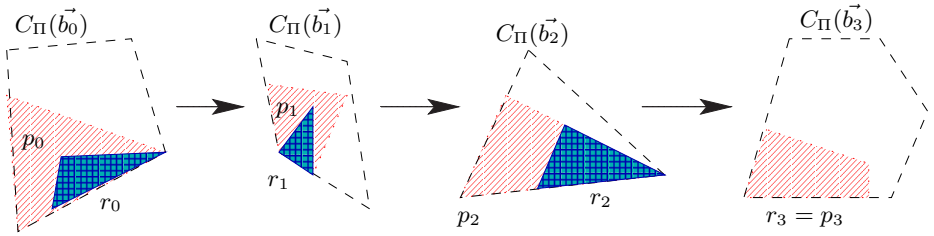

Fig. 7. Computing a concrete witness trace for a non-spurious abstract counter-example

The following two sections discuss the implemented procedures that govern the two possible outcomes of algorithm 3. First, the outcome is considered that the counter-example is found to correspond to a concrete one. Following that the computation of sets of polyhedra to be separated is discussed in case the counter-example is found to probably be spurious.

5.1 Computing Witness Traces

If algorithm 3 finds that the abstract counter-example corresponds to a concrete trace, a follow-up algorithm should be able to find a set of initial states and a sequence of concrete transitions that constitute a witness trace. This witness trace can then be used as a simulation guide in the original system to verify the existence of a counter-example. It should be noted that the approach described here is currently the only verification algorithm for hybrid systems that computes a concrete witness trace in the original system. The methodology presented here uses a slightly modified version of algorithm 3. Instead of simply saving the set of polyhedral slices at each step of the algorithm, the algorithm actually saves full traces up to that point. Each trace consists of a sequence of polyhedral slices and an appropriate transition information between successive polyhedra. This information is either the appropriate discrete switch or the exact time that the next slice has been computed.

Assuming that algorithm 3 finds that the counter-example actually does correspond to a concrete trace, it is then clear that the algorithm has found a sequence of locations and polyhedra and corresponding transition information of the form $(l_0, p_0) \rightarrow_{t_0} (l_1, p_1) \rightarrow_{t_1} \ldots \rightarrow_{t_{n-1}} (l_n, p_n)$ with $l_i \in L$ and $p_i \subseteq C_\Pi(\vec{b_i}) \cap \mathcal{X}$ for all $0 \leq i \leq n$. It is thus clear that there is a state in p_0 while in location l_0 that can end up in the set of unsafe states by taking the transitions $\rightarrow_{t_0}, \rightarrow_{t_1}, \ldots, \rightarrow_{t_{n-1}}$. However, not all states in p_0 will end in an unsafe state following these transitions. To compute a witness trace, the algorithm takes a complete trace, and starting from the last polyhedron p_n computes successively the `Pre`-sets given the applicable transition that leads to the previous polyhedron. It thus computes subsets r_i of the polyhedra p_i, and it is guaranteed that all states in r_0 following the transitions will end up in an unsafe state inside $r_n = p_n$. This procedure is illustrated in figure 7.

The figure shows an abstract counter-example with four abstract states $(l_0, \vec{b_0})$, $(l_1, \vec{b_1})$, $(l_2, \vec{b_2})$ and $(l_3, \vec{b_3})$ with $(l_i, \vec{b_i}) \rightarrow_{t_i} (l_{i+1}, \vec{b_{i+1}})$ for $i \in \{0, 1, 2\}$. Only the continuous state-space is shown in figure 7, and a particular trace as computed by algorithm 3 is given as $p_i \subseteq C_\Pi(\vec{b_i})$ for $i \in \{0, 1, 2, 3\}$. The computation of a witness trace first sets $r_3 = p_3$, and then computes

$r_2 \subseteq p_2$ as the `Pre`-set of r_3 given the transition t_2. It is guaranteed that r_2 is not empty. Analogously, the algorithm then computes $r_1 \subseteq p_1$ and $r_0 \subseteq p_0$. Any state in r_0 can then be used to prove the validity of the discovered abstract counter-example in the concrete system.

It is worth emphasizing that while most hybrid systems verification tools can produce a counter-example trace if the property to be verified is violated, it is however also the case that these traces represent over-approximations and need not necessarily reflect a counter-example in the concrete system. The approach outlined here is the only hybrid systems verification tool that computes a guaranteed concrete counter-example. This can be useful in applications such as automated test vector generation as proposed in [35].

5.2 Separating Sets of Polyhedra

The previous section described how to compute a witness trace if algorithm 3 determines that the abstract counter-example corresponds to a concrete trace. This section elaborates on the case that the algorithm determines that the abstract counter-example is spurious. As discussed earlier, the algorithm actually computes an under-approximation of the reachable sets of states following the counter-example. As discussed in section 4, the separation of polyhedra uses the last set of reachable polyhedra R_k that is not empty, and separates this set from the set of states that correspond to the `Pre` of the following abstract state s_{k+1}.

The implementation uses various heuristics to generate good separation predicates. Initially, the algorithm computes an *over-approximation* of the `Pre`-set with an user-specified finer precision than used in the abstract search. It should be noted that since an over-approximation of the `Pre`-set is computed, it is not guaranteed that it will be disjoint from the set of reachable states R_k. If R_k, however, does intersect with this over-approximation, the separation routine will not be able to compute any separation predicates. In this case, the algorithm re-computes the `Pre`-set; however, this time it computes an *under-approximation*. This guarantees that the two sets will be disjoint and can thus be separated using the greedy algorithm as presented in section 4. However, it should be understood that the predicates are based on under-approximated reachable sets of states that will be used in an abstract state-space search based on over-approximations.

6 Case Studies

We presented foundations for automated verification of safety properties of hybrid systems by combining the ideas of counter-example guided predicate abstraction and polyhedral approximation of reachable sets of linear continuous dynamics. The presented counter-example analysis tool extends previous work on predicate abstraction of hybrid systems [3]. Our current prototype implementation of the predicate abstraction model checking and the counter-example analysis tool are both implemented in C++ using library functions of the hybrid

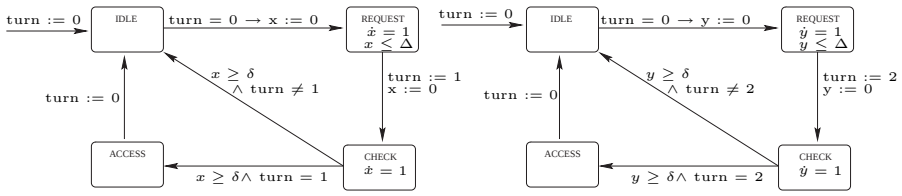

Fig. 8. The two processes for the mutual exclusion example

systems reachability tool `d/dt` [9]. We implemented a translation procedure from CHARON [2] source code to the predicate abstraction input language which is based on the `d/dt` input language. Our tool uses the polyhedral libraries CDD [24] and QHull [11]. We have implemented the global analysis algorithm, the local feasibility check, as well as the computation of separating predicates as part of the counter-example analysis tool.

6.1 Fischer's Mutual Exclusion

We first look at an example of mutual exclusion which uses time-based synchronization in a multi-process system. We want to implement a protocol that allows a shared resource to be used exclusively by at most one of two processes at any given time. The state machines for the two processes are shown in figure 8. The example is small enough to be used effectively for an illustration of our approach.

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 takes 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. The constraints $\dot{x} = 0$ and $\dot{y} = 0$ are omitted in the respective `Idle` and `Check` locations in the figure in order not to clutter the presentation. Similar case studies have been studied in various contexts and with slightly different models, as, for example, in [31].

The possible execution traces depend on the two positive parameters Δ and δ. If the parameters are such that $\Delta \geq \delta$ is true, we can find a counter-example that proves the two processes may access the shared resource at the same time. On the other hand, if $\delta > \Delta$, then the system preserves mutual exclusive use of the shared resource.

We use this example to illustrate the use of the local feasibility check of counter-examples for the case that $\delta > \Delta$. Consider the abstract system defined by the predicates used in the description of the 2-process Fischer's mutual exclusion protocol. These are: $x \geq \delta, y \geq \delta, x \leq \Delta, y \leq \Delta, \delta > \Delta, \Delta > 0, \delta > 0, x \geq 0$ and $y \geq 0$. The search in the abstract state-space finds a counter-example of length nine. The third abstract state a_3 in the counter-example has both processes in their respective `Request` locations, `turn` $= 0$, and $0 \leq x \leq \Delta, 0 \leq y \leq \Delta$.

The following state a_4 can be reached by a discrete transition t_d, and the first process is now in its `Check` location, while `turn` $= 1$ and $0 \leq x \leq \Delta, 0 \leq y \leq \Delta$. The fifth abstract state a_5 can then be reached by a continuous transition t_c, so that the locations and the `turn` variable are unchanged, but now we have $x > \delta \wedge 0 \leq y \leq \Delta$. It can be shown that a_4 is locally infeasible:

$$\texttt{Pre}(a_4, t_c, a_5) = \left\{ \begin{array}{c} ((\texttt{Check, Request}), (\texttt{turn}, x, y, \Delta, \delta)) \in X \mid \\ \texttt{turn} = 1 \wedge 0 \leq y < x \leq \Delta < \delta \end{array} \right\} \text{ and}$$

$$\texttt{Post}(a_3, t_d, a_4) = \left\{ \begin{array}{c} ((\texttt{Check, Request}), (\texttt{turn}, x, y, \Delta, \delta)) \in X \mid \\ \texttt{turn} = 1 \wedge 0 \leq x \leq y \leq \Delta < \delta \wedge 0 < \Delta \end{array} \right\};$$

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

6.2 Coordinated Adaptive Cruise Control

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

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

with a large enough initial separation, the combined verification approach enabled us to prove safety of the abstract system which implies safety of the concrete system.

7 Conclusions

This paper described algorithms for the validation analysis of abstract counter-examples in the concrete hybrid system. If this validation analysis finds that the abstract counter-example does in fact represent a valid counter-example in the concrete system, a concrete witness trace is computed that can be used by the user as an input to a simulation engine. This approach is currently the only tool that computes a guaranteed witness trace, and it can be used in various applications such as automated test vector generation.

If the validation analysis determines that the abstract counter-example is spurious, this paper also describes methods to identify appropriate new predicates to be used in subsequent abstract state-space explorations. The success of the abstract search crucially depends on the choice of the predicates and the methods described in this paper produce such predicates. We also defined a notion of refinement of abstract states and abstract paths that provides valuable insight into the quality of such computed predicates.

References

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[2] R. Alur, T. Dang, J. Esposito, Y. Hur, F. Ivančić, V. Kumar, I. Lee, P. Mishra, G. Pappas, and O. Sokolsky. Hierarchical modeling and analysis of embedded systems. *Proceedings of the IEEE*, 91(1):11–28, January 2003.

[3] R. Alur, T. Dang, and F. Ivančić. Reachability analysis of hybrid systems via predicate abstraction. In *Hybrid Systems: Computation and Control, Fifth International Workshop*, LNCS 2289, pages 35–48. Springer-Verlag, 2002.

[4] R. Alur, T. Dang, and F. Ivančić. Reachability analysis of hybrid systems using counter-Example guided predicate abstraction. Technical Report MS-CIS-02-34, University of Pennsylvania, Philadelphia, PA, November 2002.

[5] R. Alur, T. Dang, and F. Ivančić. Counter-example guided predicate abstraction of hybrid systems. In *Ninth International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 2619, pages 208–223. Springer-Verlag, April 2003.

[6] R. Alur, T. Dang, and F. Ivančić. Progress on reachability analysis of hybrid systems via predicate abstraction. In *Hybrid Systems: Computation and Control, Sixth International Workshop*, LNCS 2623, pages 4–19. Springer-Verlag, April 2003.

[7] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[8] R. Alur, A. Itai, R.P. Kurshan, and M. Yannakakis. Timing verification by successive approximation. *Information and Computation*, 118(1):142–157, 1995.

[9] E. Asarin, O. Bournez, T. Dang, and O. Maler. Approximate reachability analysis of piecewise-linear dynamical systems. In *Hybrid Systems: Computation and Control, Third International Workshop*, LNCS 1790, pages 21–31. 2000.

[10] T. Ball and S. Rajamani. Bebop: A symbolic model checker for boolean programs. In *SPIN 2000 Workshop on Model Checking of Software*, LNCS 1885, pages 113–130. 2000.

[11] C. Barber, D. Dobkin, and H. Huhdanpaa. The Quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4):469–483, December 1996.

[12] S. Cameron. A comparison of two fast algorithms for computing the distance between convex polyhedra. *IEEE Transactions on Robotics and Automation*, 13(6):915–920, 1997.

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

[14] E. Clarke, A. Fehnker, Z. Han, B. Krogh, O. Stursberg, and M. Theobald. Verification of hybrid systems based on counterexample-guided abstraction refinement. In *Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 2619, pages 192–207. Springer-Verlag, 2003.

[15] E.M. Clarke, A. Fehnker, Z. Han, B. Krogh, J. Ouaknine, O. Stursberg, and M. Theobald. Abstraction and counterexample-guided refinement of hybrid systems. *International Journal of Foundations of Computer Science*, 2003. (submitted).

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

[17] E.M. Clarke and R.P. Kurshan. Computer-aided verification. *IEEE Spectrum*, 33(6):61–67, 1996.

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

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

[20] G. Das and D. Joseph. The complexity of minimum convex nested polyhedra. In *Canadian Conference on Computational Geometry*, 1990.

[21] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In *Hybrid Systems III: Verification and Control*, LNCS 1066. Springer-Verlag, 1996.

[22] D. Dobkin and D. Kirkpatrick. Determining the separation of preprocessed polyhedra - a unified approach. In *Proc. of ICALP'90*, pages 400–413, 1990.

[23] H. Edelsbrunner and F.P. Preparata. Minimum polygon separation. *Information and Computation*, 77:218–232, 1987.

[24] K. Fukuda. cddlib reference manual, cddlib version 092a. Technical report, McGill University, 2001.

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

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

[27] G.J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.

[28] G.J. Holzmann and M.H. Smith. Automating software feature verification. *Bell Labs Technical Journal*, 5(2):72–87, 2000.

[29] T. Hudson, M. Lin, J. Cohen, S. Gottschalk, and D. Manocha. V-collide: Accelerate collision detection for vrml. In *Proc. of VRML'97*, 1997.

[30] F. Ivančić. Modeling and analysis of hybrid systems. *Ph.D. dissertation, University of Pennsylvania*, Philadelphia, PA, 2003.

[31] K. J. Kristoffersen, F. Laroussinie, K. G. Larsen, P. Pettersson, and Wang Yi. A compositional proof of a real-time mutual exclusion protocol. In *Proc. 7th Int. Joint Conf. Theory and Practice of Software Development (TAPSOFT'97), Lille, France, Apr. 1997*, volume 1214, pages 565–579. Springer, 1997.

[32] K. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *Springer International Journal of Software Tools for Technology Transfer*, 1, 1997.

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

[34] N. Medgiddo. On the complexity of polyhedral separability. *Discrete and Computational Geometry*, pages 325–337, 1988.

[35] L. Tan, O. Sokolsky, and I. Lee. Property-coverage testing. Technical Report MS-CIS-03-02, University of Pennsylvania, January 2003.