Automatic Symbolic Verification of Embedded Systems*

Rajeev Alur† Thomas A. Henzinger‡ Pei-Hsin Ho‡

Abstract. We present a model-checking procedure and its implementation for the automatic verification of embedded systems. The system components are described as *Hybrid Automata*—communicating machines with finite control and real-valued variables that represent continuous environment parameters such as time, pressure, and temperature. The system requirements are specified in a temporal logic with stop watches, and verified by symbolic fixpoint computation. The verification procedure—implemented in the Cornell Hybrid Technology Tool, HyTech—applies to hybrid automata whose continuous dynamics is governed by linear constraints on the variables and their derivatives. We illustrate the method and the tool by checking safety, liveness, time-bounded, and duration requirements of digital controllers, schedulers, and distributed algorithms.

1 Introduction

Hybrid systems are digital real-time systems that are embedded in analog environments. Due to the rapid development of digital-processor technology, hybrid systems directly control much of what we depend on in our daily lives. Many hybrid systems, ranging from automobiles to aircraft, operate in safety-critical situations and therefore call for rigorous analysis techniques. Yet traditional program verification methods allow us, at best, to approximate continuously changing environments by discrete sampling. Only recently have there been some attempts at developing a verification methodology for hybrid systems [GNRR93].

In this paper, we pursue the approach suggested in [ACHH93, ACH$^+$94] for solving reachability problems of constant-slope hybrid systems, whose variables follow piecewise-linear trajectories. We present progress in three directions. First, we generalize the system model to accommodate, in addition to variables with piecewise-linear trajectories, also variables whose trajectories are characterized by linear constraints on the slopes. This extension allows the approximation of nonlinear behavior by piecewise-linear envelopes. Second, we generalize the method to verify, in addition to reachability properties, also temporal formulas with clocks and stop watches. This extension allows the analysis of liveness, time-bounded, and duration requirements of hybrid systems. Third, we report on an implementation of the verification procedure.

Hybrid automata. We model the components of a hybrid system as *Hybrid Automata* [ACHH93]. A hybrid automaton is a generalized finite-state machine that is equipped with continuous variables. The discrete actions of a program are modeled by a program counter whose value changes

*A preliminary version of this paper appeared in the *Proceedings of the 14th Annual IEEE Real-time Systems Symposium* (RTSS 1993), pp. 2–11.

†AT&T Bell Laboratories, Murray Hill, NJ 07974 (alur@research.att.com).

‡Department of Computer Science, Cornell University, Ithaca, NY 14853 ({tah,ho}@cs.cornell.edu). Supported in part by the National Science Foundation under grant CCR-9200794, by the United States Air Force Office of Scientific Research under contract F49620-93-1-0056, and by the Defense Advanced Research Projects Agency under grant NAG2-892.

instantaneously at various points in time by moving through a finite set of control locations. The continuous activities of the environment are modeled by real-valued variables whose values change continuously over time according to the laws of physics: for each control location, the continuous activities of the environment are governed by differential equations. This model for hybrid systems is inspired by the Phase Transition Systems of [MMP92] and [NSY93], and can be viewed as a generalization of Timed Automata [AD94]; a similar model has been proposed and studied independently in [NOSY93].

For verification purposes, we restrict ourselves to *Linear Hybrid Automata*, which are introduced in Section 2. In each location of a linear hybrid automaton, the behavior of all variables are governed by linear constraints on the first derivatives. Common examples of linear constraints are constant differential equations, rectangular differential inclusions, and rate comparisons. For instance, the constant differential equation $\dot{x} = 2$ restricts the variable x to a linear trajectory with slope 2. The rectangular differential inclusion $1 \leq \dot{x} \leq 3$ restricts x to any trajectory whose slope stays within the interval $[1, 3]$. The rate comparison $\dot{x} \geq \dot{y}$ ensures that x always grows faster than y. Using rectangular differential inclusions we can model distance x assuming bounded speed $\dot{x} \in [l, u]$, or time y assuming a clock with bounded drift rate $\dot{y} \in [l, u]$. Rectangular differential inclusions are also useful to approximate nonlinear trajectories by piecewise-linear envelopes.

Integrator logic. Real-time requirements of systems can be specified in TCTL [ACD93], a branching-time logic that extends CTL [CES86] with clock variables. In Section 3, we introduce *Integrator Computation Tree Logic*, ICTL, which strengthens TCTL in the style of [BES93] by admitting integrator variables. An integrator (or stop watch) is a clock that can be stopped and restarted. While clocks suffice for the specification of time-bounded requirements—such as "A response is obtained if a ringer has been pressed continuously for at least d seconds"—integrators are necessary to accumulate delays, and are useful for specifying duration requirements—such as "A response is obtained if a ringer has been pressed, possibly intermittently, for at least d seconds." We use ICTL to specify safety, liveness, time-bounded, and duration requirements of hybrid automata.

Model checking. Model checking is a powerful technique for the automatic verification of systems: a model-checking algorithm determines whether a mathematical model of a system meets a requirement specification that is given as a temporal-logic formula. For discrete finite-state systems, model checking has a long history spanning more than a decade, and has been successful in validating communication protocols and hardware circuits [CES81, QS81, LP85, McM93]. In recent years, model-checking algorithms have been developed also for real-time systems that are described by discrete programs with real-valued clocks [AFH91, ACD93].

As the variables of a hybrid system range over the real numbers, the state space is infinite, and state sets—so-called *regions*—must be represented *symbolically* rather than enumeratively. A symbolic model-checking algorithm for verifying TCTL-requirements of real-time systems is presented in [HNSY94]. It has been observed in [ACHH93] and in [NOSY93] that the primitives of that algorithm can be redefined for the reachability analysis of constant-slope hybrid automata, whose variables are governed by constant differential equations. We extend this result and present a symbolic model-checking procedure for verifying ICTL-requirements of linear hybrid automata.

Given an ICTL-formula and a hybrid automaton, we compute the target region of states that satisfy the formula by successive approximation, as the limit of an infinite sequence of regions. The approximation sequence is generated by iterating boolean operators and weakest-precondition operators on regions. For linear hybrid automata, all regions of the approximation sequence are *linear* in the sense that they can be defined in the theory $(\mathbb{R}, \leq, +)$ of the reals with addition, whose formulas are boolean combinations of linear inequalities. In Section 4, we compute the weakest precondition of $(\mathbb{R}, \leq, +)$-formulas with respect to linear hybrid automata. The precondition computation, then, is iterated in Section 5 to construct a $(\mathbb{R}, \leq, +)$-formula that defines the target region.

Symbolic computation. The model-checking procedure has been implemented as part of the *Cornell HYbrid TECHnology Tool*, HyTech, using the symbolic computation system Mathematica [Wol88] for manipulating and simplifying $(\mathbb{R}, \leq, +)$-formulas. In particular, the computation of weakest preconditions of linear regions requires quantifier elimination in the theory $(\mathbb{R}, \leq, +)$. In Section 6, we present some details of the implementation and, in Section 7, we illustrate the method and the tool by specifying and verifying four examples. We check (1) a gate controller for a railroad crossing, (2) a timing-based mutual-exclusion protocol with distributed, asynchronously drifting local clocks, (3) a preemptive scheduler with prioritized tasks, and (4) the temperature control of a nuclear reactor, approximating exponential temperature changes using rectangular differential inclusions.

These examples demonstrate an important advantage of the symbolic approach to verification. Although, in theory, the computational complexity of the verification problem is proportional to the magnitudes of the system delays, in practice the performance of the symbolic procedure is quite insensitive to the size of delays. Indeed, in place of concrete values for system delays, we can use symbolic parameters and our procedure will output necessary and sufficient constraints on these parameters for the system to satisfy the desired property [AHV93]. As is illustrated in examples (2) and (4), such a symbolic delay analysis can help the system designer to choose critical system parameters.

The convergence of all approximation sequences, and thus the termination of the model-checking procedure, is guaranteed only for restricted classes of linear hybrid systems, as already the reachability problem for constant-slope hybrid systems is undecidable [ACHH93, KPSY93]. Notwithstanding, we have found that the method is of practical interest. First, the successive-approximation process converges within a few iterations for many examples we have attempted to verify, including those of Section 7. Second, there is little difference for the practitioner between a diverging (i.e., nonterminating) process and a process that runs out of time or space resources. Thus we submit that it is important not only to identify decidable verification problems, but also to improve the applicability of semidecision procedures for verification.

2 System Description Language: Hybrid Automata

Informally, a hybrid automaton [ACHH93] consists of a finite vector $\vec{x}$ of real-valued variables and a labeled multigraph (V, E). By $\dot{\vec{x}}$ we denote the vector of first derivatives of the variables in $\vec{x}$. The edges E represent discrete system actions and are labeled with nondeterministic guarded assignments to $\vec{x}$. The vertices V represent continuous environment activities and are labeled with constraints on $\vec{x}$ and $\dot{\vec{x}}$. The state of the automaton changes either through instantaneous system actions or, while time elapses, through continuous activities. In this paper, we restrict ourselves to guards, assignments, and vertex constraints that are linear expressions.

2.1 Syntax

Let $\vec{y}$ be a vector of real-valued variables. A *linear term* over $\vec{y}$ is a linear combination of variables from $\vec{y}$ with integer coefficients. A *linear inequality* over $\vec{y}$ is an inequality between linear terms over $\vec{y}$. A (*closed*) *convex linear formula* over $\vec{y}$ is a finite conjunction of (nonstrict) linear inequalities over $\vec{y}$. A *linear formula* over $\vec{y}$ is a finite boolean combination of linear inequalities over $\vec{y}$. Every linear formula can be transformed into disjunctive normal form, that is, into a finite disjunction of convex linear formulas.

A *linear hybrid automaton* A consists of the following components:

Data variables A finite vector $\vec{x} = (x_1, \ldots, x_n)$ of real-valued *data variables*. The size n of $\vec{x}$ is called the *dimension* of A.

A *data state* is a point $\vec{s} = (s_1, \ldots, s_n)$ in n-dimensional real space $\mathbb{R}^n$ or, equivalently, a function that assigns to each data variable x_i a real value $s_i \in \mathbb{R}$. A *convex data region* is a polyhedron in $\mathbb{R}^n$. A *data region* is a finite union of convex data regions. A (*convex*) *data predicate* is a (convex) linear formula over $\vec{x}$. The (convex) data predicate p defines the (convex) data region $[\![p]\!] \subseteq \mathbb{R}^n$, where $\vec{s} \in [\![p]\!]$ iff $p[\vec{x} := \vec{s}]$ is true. The data region S is (*convex*) *linear* if there is a (convex) data predicate that defines S.

For each data variable x_i, we use the dotted variable $\dot{x}_i$ to denote the first derivative of x_i. A *differential inclusion* is a polyhedron in $\mathbb{R}^n$. A *rate predicate* is a convex linear formula over the set $\vec{\dot{x}} = (\dot{x}_1, \ldots, \dot{x}_n)$ of dotted variables. The rate predicate r defines the differential inclusion $[\![r]\!] \subseteq \mathbb{R}^n$, where $\vec{\dot{s}} \in [\![r]\!]$ iff $r[\vec{\dot{x}} := \vec{\dot{s}}]$ is true.

For each data variable x_i, we use the primed variable x_i' to denote the new value of x_i after a transition. An *action predicate* is a convex linear formula over the set $\vec{x} \uplus \vec{x}'$ of data variables $\vec{x}$ and primed variables $\vec{x}' = (x_1', \ldots, x_n')$. The action predicate q maps each data state $\vec{s} \in \mathbb{R}^n$ to the convex data region $[\![q]\!](\vec{s}) \subseteq \mathbb{R}^n$, where $\vec{s}' \in [\![q]\!](\vec{s})$ iff $q[\vec{x} := \vec{s}, \vec{x}' := \vec{s}']$ is true.

Control locations A finite set V of vertices called *control locations*.

A *state* $(v, \vec{s})$ of the automaton A consists of a control location $v \in V$ and a data state $\vec{s} \in \mathbb{R}^n$. A *region* $R = \bigcup_{v \in V}(v, S_v)$ is a collection of data regions $S_v \subseteq \mathbb{R}^n$, one for each control location $v \in V$. A *state predicate* $\phi = \bigcup_{v \in V}(v, p_v)$ is a collection of data predicates p_v, one for each control location $v \in V$. The state predicate ϕ defines the region $[\![\phi]\!] = \bigcup_{v \in V}(v, [\![p_v]\!])$. The region R is *linear* if there is a state predicate that defines R.

We write (v, S) for the region $(v, S) \cup \bigcup_{v' \neq v}(v', \emptyset)$, and (v, p) for the state predicate $(v, p) \cup \bigcup_{v' \neq v}(v', \mathit{false})$. When writing state predicates, we use the *location counter* ℓ, which ranges over the set V of control locations. The location constraint $\ell = v$ denotes the state predicate (v, true). The data predicate p, when used as a state predicate, denotes the collection $\bigcup_{v \in V}(v, p)$. For two state predicates $\phi = \bigcup_{v \in V}(v, p_v)$ and $\phi' = \bigcup_{v \in V}(v, p_v')$, we define $\neg\phi = \bigcup_{v \in V}(v, \neg p_v)$, $\phi \vee \phi' = \bigcup_{v \in V}(v, p_v \vee p_v')$, and $\phi \wedge \phi' = \bigcup_{v \in V}(v, p_v \wedge p_v')$.

Location invariants A labeling function *inv* that assigns to each control location $v \in V$ a convex data predicate $\mathit{inv}(v)$, the *invariant* of v. The invariants are used to enforce the progress of a system from one control location to another, because the control of the automaton A may reside in the location v only as long as the invariant $\mathit{inv}(v)$ is true. The state $(v, \vec{s})$ is *admissible* if $\vec{s} \in [\![\mathit{inv}(v)]\!]$. We write Σ_A for the set of admissible states of A, and ϕ_A for the state predicate $\bigcup_{v \in V}(v, \mathit{inv}(v))$ that defines the set of admissible states.

Continuous activities A labeling function *dif* that assigns to each control location $v \in V$ a rate predicate $\mathit{dif}(v)$, the *activity* of v. The activities constrain the rates at which the values of data variables change: while the automaton control resides in the location v, the first derivatives of all data variables stay within the differential inclusion $[\![\mathit{dif}(v)]\!]$.

Transitions A finite multiset E of edges called *transitions*. Each transition (v, v') identifies a source location $v \in V$ and a target location $v' \in V$. For each location $v \in V$, there is a *stutter transition* $e_v = (v, v)$.

Discrete actions A labeling function *act* that assigns to each transition $e \in E$ an action predicate $act(e)$, the *action* of e. If the automaton control proceeds from the location v to the location v' via the transition $e = (v, v')$, then the values of all data variables change nondeterministically from $\vec{s}$ to a point in the data region $[\![act(e)]\!](\vec{s})$. For example, a transition with the action label

$$x_1 \leq 3 \ \wedge \ 3 \leq x_1' \leq 5 \ \wedge \ x_2' = x_2 \ \wedge \ x_3' = x_1 + 1$$

can be traversed only when the value of x_1 is at most 3. The transition updates the value of x_1 to a real number in the interval $[3, 5]$, the value of x_2 remains unchanged, and the new value of x_3 is 1 greater than the old value of x_1. All stutter transitions are labeled with the action predicate $\vec{x}' = \vec{x}$.

Synchronization labels A finite set L of *synchronization labels* and a labeling function *syn* that assigns to each transition $e \in E$ a set of synchronization labels from L. The set L is called the *alphabet* of A. The synchronization labels are used to define the parallel composition of two automata: if both automata share a synchronization label a, then each a-transition of one automaton must be accompanied by an a-transition of the other automaton. All stutter transitions are labeled with the empty set of synchronization labels.

The restriction to convex invariants, activities, and actions does not limit the expressiveness of linear hybrid automata, because nonconvex invariants and activities can be modeled by splitting locations (see Section 4), and nonconvex actions can be modeled by splitting transitions.

A special case of a linear hybrid automaton is a timed automaton [AD94]. The data variable x_i of A is a *clock* if for all control locations $v \in V$, $dif(v)$ implies $\dot{x}_i = 1$, and for all transitions $e \in E$, $act(e)$ implies $x_i' = 0$ or $x_i' = x_i$; that is, each clock always increases with the rate at which time advances, and all transitions either reset a clock to 0 or leave its value unchanged. A linear inequality is *simple* if it has the form $x = y$, $x \leq c$, or $x \geq c$, for some integer constant c. The linear hybrid automaton A is a *timed automaton* if all data variables in $\vec{x}$ are clocks, and all invariants and actions are boolean combinations of simple linear inequalities.

2.2 Semantics

At any time instant, the state of a hybrid automaton specifies a control location and values for all data variables. The state can change in two ways: (1) by an instantaneous transition that changes both the control location and the values of data variables, or (2) by a time delay that changes only the values of data variables in a continuous manner according to the rate predicate of the current control location.

A *data trajectory* (δ, ρ) of the linear hybrid automaton A consists of a nonnegative *duration* $\delta \in \mathbb{R}_{\geq 0}$ and a differentiable function $\rho\colon [0, \delta] \to \mathbb{R}^n$ with the derivative $\frac{d\rho(t)}{dt}$ for all $t \in (0, \delta)$; The data trajectory (δ, ρ) maps every real $t \in [0, \delta]$ to the data state $\rho(t)$. Consider two reals t_1 and t_2 with $0 \leq t_1 \leq t_2 \leq \delta$. We write $\rho[t_1, t_2]$ for the data trajectory (δ', ρ') with $\delta' = t_2 - t_1$, and $\rho'(t) = \rho(t + t_1)$ for all $t \in [0, \delta']$. The data trajectory (δ, ρ) is *linear* if there is a constant rate vector $\dot{\vec{s}} \in \mathbb{R}^n$ such that $\frac{d\rho(t)}{dt} = \dot{\vec{s}}$ for all $t \in (0, \delta)$. The data trajectory (δ, ρ) is *piecewise linear* if there are *finitely* many reals $t_1, \ldots, t_k \in [0, \delta]$ such that the data trajectories $\rho[0, t_1], \rho[t_1, t_2], \ldots, \rho[t_k, \delta]$ are linear.

A *trajectory* τ of A is an infinite sequence

$$(v_0, \delta_0, \rho_0) \;\rightarrow\; (v_1, \delta_1, \rho_1) \;\rightarrow\; (v_2, \delta_2, \rho_2) \;\rightarrow\; (v_3, \delta_3, \rho_3) \;\rightarrow\; \cdots$$

of control locations $v_i \in V$ and data trajectories (δ_i, ρ_i) such that for all $i \geq 0$:

Invariants for all reals $t \in [0, \delta_i]$, $\rho_i(t) \in [\![inv(v_i)]\!]$;

Activities for all reals $t \in (0, \delta_i)$, $\frac{d\rho(t)}{dt} \in [\![dif(v_i)]\!]$;

Actions there is a transition $e_i = (v_i, v_{i+1}) \in E$ with $\rho_{i+1}(0) \in [\![act(e_i)]\!](\rho_i(\delta_i))$.

A *position* of the trajectory τ is a pair (i, ϵ) that consists of a nonnegative integer i and a nonnegative real $\epsilon \leq \delta_i$. The positions of τ are ordered lexicographically: the position (i, δ) precedes the position (j, ϵ), denoted $(i, \delta) < (j, \epsilon)$, iff either $i < j$, or $i = j$ and $\delta < \epsilon$. The *state at position* (i, ϵ) of τ is $\tau(i, \epsilon) = (v_i, \rho_i(\epsilon))$ (notice that all states of τ are admissible). The *time at position* (i, ϵ) of τ is the finite sum $t_\tau(i, \epsilon) = (\Sigma_{0 \leq j < i}\, \delta_j) + \epsilon$. The *duration* of the trajectory τ is the infinite sum $\delta_\tau = \Sigma_{j \geq 0}\, \delta_j$.

The trajectory τ *diverges* if $\delta_\tau = \infty$. The trajectory τ is *linear* if all data trajectories (δ_i, ρ_i) of τ are linear. By $[\![A]\!]$ we denote the set of trajectories of the automaton A. If $\mathcal{T}$ is a set of trajectories, $\mathcal{T}^{div}$ is the set of divergent trajectories in $\mathcal{T}$, and $\mathcal{T}_{lin}$ is the set of linear trajectories in $\mathcal{T}$.

Consider two positions $\pi_1 = (v_i, \epsilon_1)$ and $\pi_2 = (v_j, \epsilon_2)$ of the trajectory τ. We write $\tau[\pi_1, \pi_2]$ for the trajectory fragment

$$(v_i, \rho[\epsilon_1, \delta_i]) \;\rightarrow\; (v_{i+1}, \delta_{i+1}, \rho_{i+1}) \;\rightarrow\; \cdots \;\rightarrow\; (v_j, \rho[0, \epsilon_2]),$$

and $\tau[\pi_1, \infty]$ for the trajectory

$$(v_i, \rho[\epsilon_1, \delta_i]) \;\rightarrow\; (v_{i+1}, \delta_{i+1}, \rho_{i+1}) \;\rightarrow\; (v_{i+2}, \delta_{i+2}, \rho_{i+2}) \;\rightarrow\; \cdots$$

The trajectory set $[\![A]\!]$ is closed under suffixes (if $\tau \in [\![A]\!]$ and π is a position of τ, then $\tau[\pi, \infty] \in [\![A]\!]$); stuttering (if $\tau \in [\![A]\!]$ and π is a position of τ, then $(\tau[(0,0), \pi]\tau[\pi, \infty]) \in [\![A]\!]$); fusion (if $\tau, \tau' \in [\![A]\!]$, π is a position of τ, π' is a position of τ', and $\tau(\pi) = \tau'(\pi')$, then $(\tau[(0,0), \pi]\tau[\pi', \infty]) \in [\![A]\!]$); and limits (if for all positions π of τ there is a trajectory $\tau' \in [\![A]\!]$ and a position π' of τ' such that $\tau[(0,0), \pi] = \tau'[(0,0), \pi']$, then $\tau \in [\![A]\!]$). Notice that fusion closure asserts that the future evolution of a hybrid automaton is completely determined by the present state of the automaton. Also notice that the suffix, stutter, and fusion closures of a trajectory set $\mathcal{T}$ are inherited by the subsets $\mathcal{T}^{div}$ and $\mathcal{T}_{lin}$. If $\mathcal{T}$ is closed under limits, then so is $\mathcal{T}_{lin}$, and $\mathcal{T}^{div}$ is closed under divergent limits ("divergence-safe") [HNSY94].

The linear hybrid automaton A is *nonzeno* if for every admissible state σ of A there is a divergent trajectory τ of A such that $\tau(0,0) = \sigma$. In other words, A is nonzeno iff every finite prefix of a trajectory is a prefix of a divergent trajectory. Notice that if A is nonzeno, then the states that occur on divergent trajectories of A are precisely the admissible states Σ_A. We restrict our attention to nonzeno hybrid automata. In [HNSY94] it is shown how a timed automaton may be turned into a nonzeno automaton with the same divergent trajectories; this is done by strengthening the location invariants, and applies to many hybrid automata also.

2.3 Composition

A hybrid system typically consists of several components that operate concurrently and communicate with each other. We describe each component as a linear hybrid automaton. The component automata coordinate through shared data variables, and to facilitate message-type coordination, we also use synchronization labels on the automaton transitions. The linear hybrid automaton that models the entire system is then constructed from the component automata using a product operation.

Let $A_1 = (\vec{x}_1, V_1, inv_1, dif_1, E_1, act_1, L_1, syn_1)$ and $A_2 = (\vec{x}_2, V_2, inv_2, dif_2, E_2, act_2, L_2, syn_2)$ be two linear hybrid automata of dimensions n_1 and n_2, respectively. The *product* $A_1 \times A_2$ of A_1 and A_2 is the linear hybrid automaton $A = (\vec{x}_1 \cup \vec{x}_2, V_1 \times V_2, inv, dif, E, act, L_1 \cup L_2, syn)$:

- Each location (v, v') in $V_1 \times V_2$ has the invariant $inv(v, v') = inv_1(v) \wedge inv_2(v')$ and the activity $dif(v, v') = dif_1(v) \wedge dif_2(v')$; that is, an admissible state of A consists of an admissible state of A_1 and an admissible state of A_2, whose shared parts coincide, and whose rate vector obeys the differential inclusions that are associated with both components locations.
- E contains the transition $e = ((v_1, v_1'), (v_2, v_2'))$ iff

  (1) $v_1 = v_2$ and there is a transition $e_2 = (v_1', v_2') \in E_2$ with $L_1 \cap syn_2(e_2) = \emptyset$; or
  (2) there is a transition $e_1 = (v_1, v_2) \in E_1$ with $syn_1(e_1) \cap L_2 = \emptyset$, and $v_1' = v_2'$; or
  (3) there is a transition $e_1 = (v_1, v_2) \in E_1$ and a transition $e_2 = (v_1', v_2') \in E_2$ such that $syn_1(e_1) \cap L_2 = syn_2(e_2) \cap L_1 \neq \emptyset$.

  In case (1), $act(e) = (\bigwedge_{x \in \vec{x}_1 \setminus \vec{x}_2} x' = x) \wedge act_2(e_2)$ and $syn(e) = syn_2(e_2)$. In case (2), $act(e) = act_1(e_1) \wedge (\bigwedge_{x \in \vec{x}_2 \setminus \vec{x}_1} x' = x)$ and $syn(e) = syn_1(e_1)$. In case (3), $act(e) = act_1(e_1) \wedge act_2(e_2)$ and $syn(e) = syn_1(e_1) \cup syn_2(e_2)$.

Since the two component automata A_1 and A_2 may share data variables, the dimension of A lies between $max(n_1, n_2)$ and $n_1 + n_2$. According to the definition of E, the transitions of the two component automata are interleaved, provided they have no labels in $L_1 \cap L_2$. Labels in $L_1 \cap L_2$ must be synchronized, and cause the simultaneous traversal of component transitions. Notice that, in cases (1) and (2), the stutter transitions of the component automata result in stutter transitions of the product automaton.

2.4 Example: Railroad gate controller

We model a control system for a railroad crossing using linear hybrid automata. The system consists of three processes—a train, a gate, and a gate controller.

The variable x represents the distance of the train from the gate. Initially, the train is far from the gate and always moves at a speed that varies between 48 and 52 meters per second. When the train approaches the gate, a sensor that is placed at a distance of 1000 meters from the crossing detects the train and sends the signal *app* to the controller. The train then may slow down to a speed between 40 and 52 meters per second. If the controller is idle upon receipt of the approach signal *app*, it requires up to 5 seconds to send the command *lower* to the gate; the delay of the controller is modeled by the clock z. If the gate is open, it is lowered from 90 radius degrees to 0 degrees at the constant rate of 20 degrees per second; the position of the gate in degrees is represented by the variable y. A second sensor placed at 100 meters past the crossing detects the leaving train and signals *exit* to the controller, which, after another delay of up to 5 seconds, sends the command *raise* to the gate. We assume that the distance between consecutive trains is at least

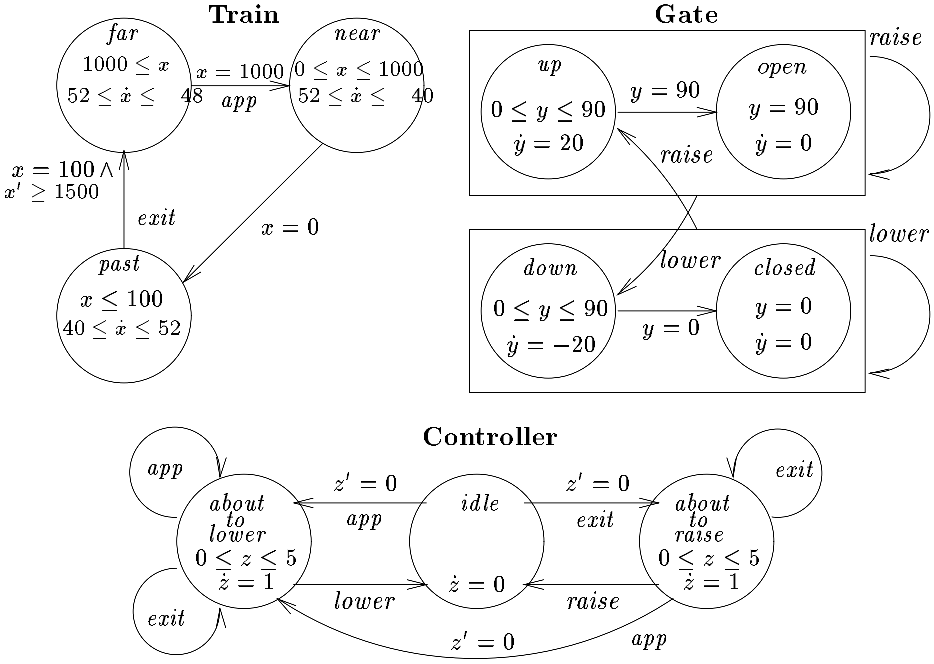

Figure 1: Railroad gate controller

1500 meters, so when the sensor detects a leaving train, the next (or returning) train is at least 1500 meters from the crossing.

The controller must accept arriving *app* and *exit* signals at any time, and the gate must always accept controller commands. For fault tolerance considerations, we design the controller so that an *exit* signal is ignored if the gate is about to be lowered, while an *app* signal always causes the gate to be lowered.

The three hybrid automata that model the train, the gate, and the controller are shown in Figure 1. In the graphical representation of the automata we use "superlocations" to save on edges. In particular, the gate automaton has four locations—*up* ("being raised"), *open*, *down* ("being lowered"), and *closed*. We suppress invariants of the form *true*, conjuncts of the form $\dot{x} = 0$ for activities, conjuncts of the form $x' = x$ for actions, and we suppress stutter transitions. The location *up* of the gate automaton has the invariant $0 \leq y \leq 90$, the activity $\dot{y} = 20$, and two outgoing transitions; the transition to the location *open* has the activity $y = 90 \wedge y' = y$, and no synchronization labels; the transition to *down* has the activity $y' = y$ and the synchronization label *lower*. The synchronization labels model signals (from the train to the controller) and commands (from the controller to the gate). For instance, when the train automaton changes location using an edge labeled with *app*, the controller automaton is required to traverse an edge with the same label.

3 Property Description Language: Integrator Logic

The formulas of the *Integrator Computation Tree Logic* ICTL for a given linear hybrid automaton A contain two kinds of variables—data and control variables of A, and integrators. An integrator (or stop watch) is a clock that can be stopped and restarted. We adopt the notation of [BES93] to generalize the clock reset ("freeze") quantifier of TPTL [AH94] to a reset quantifier for integrators.

While the clock reset quantifier $z.\,\varphi$ introduces (binds) the clock z and sets its value at 0, the integrator reset quantifier $(z\colon U).\,\varphi$ introduces (binds) the integrator z, declares its type to be U, and sets its value to 0. The *type* $U \subseteq V$ of z is a set of locations from A. The value of an integrator of type U increases with the rate at which time advances whenever the automaton control is in a location in U, and its value stays unchanged whenever the automaton control is in a location in $V \backslash U$. In particular, a clock is an integrator of type V.

3.1 Syntax

The formulas of ICTL are built from state predicates using boolean operators, the two temporal operators $\exists\mathcal{U}$ ("possibly") and $\forall\mathcal{U}$ ("inevitably"), and the reset quantifier for integrators. Intuitively, the formula $\varphi_1\exists\mathcal{U}\varphi_2$ holds in the automaton state σ if along *some* automaton trajectory that starts from σ, the second argument φ_2 holds in some state of the trajectory, and the first argument φ_1 holds in all intermediate states. The formula $\varphi_1\forall\mathcal{U}\varphi_2$ asserts that along *every* trajectory that starts from σ, the first argument φ_1 is true until the second argument φ_2 becomes true.

Let A be linear hybrid automaton with the data variables $\vec{x}$ and the control locations V, and let $\vec{z}$ be a vector of real-valued variables called *integrators.* A *$\vec{z}$-extended data predicate* of A is a linear formula over $\vec{x} \uplus \vec{z}$. A *$\vec{z}$-extended state predicate* of A is a collection of $\vec{z}$-extended data predicates, one for each location in V. The *A-formulas* of ICTL are defined inductively by the grammar

$$\varphi \ ::= \ \phi \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1\exists\mathcal{U}\varphi_2 \mid \varphi_1\forall\mathcal{U}\varphi_2 \mid (z\colon U).\,\varphi$$

where ϕ is a $\vec{z}$-extended state predicate, $U \subseteq V$ is a set of locations, and z is an integrator from $\vec{z}$. The ICTL-formula φ is *closed* if every occurrence of an integrator in φ is bound by a reset quantifier. We restrict ourselves to closed formulas of ICTL. We also assume that different reset quantifiers in φ bind different integrators, which can be achieved by renaming bound variables.

If all integrators in φ have the type *true*, then φ is a formula of TCTL [ACD93]. When writing ICTL-formulas, we suppress the integrator type *true*, and we use boolean combinations of location constraints for defining integrator types. Typical abbreviations for ICTL-formulas include the standard temporal operators $\forall\Diamond\varphi$, $\exists\Box\varphi$, and $\varphi_1\exists\mathcal{W}\varphi_2$, for $\mathit{true}\forall\mathcal{U}\varphi$, $\neg\forall\Diamond\neg\varphi$, and $\varphi_1\exists\mathcal{U}\varphi_2 \vee \exists\Box\varphi_1$, respectively. We also use time-bounded temporal operators [ACD93] such as $\forall\Diamond_{\leq 5}\,\varphi$, which stands for the ICTL-formula $z.\,\forall\Diamond(z \leq 5 \wedge \varphi)$, that is, $(z : \mathit{true}).\,(\mathit{true}\forall\mathcal{U}(z \leq 5 \wedge \varphi))$.

3.2 Semantics

Every closed A-formula ψ of ICTL defines a region $[\![\psi]\!]_A$ of the automaton A. The region $[\![\psi]\!]_A$ is called the *characteristic A-region* of ψ, and is defined in three steps. First, we extend A to an automaton $A_{\vec{z}}$ whose data variables include the integrators from $\vec{z}$. Second, we interpret ψ over the states of the extended automaton $A_{\vec{z}}$. Third, we relate the states of $A_{\vec{z}}$ to the states of A.

Let $\vec{z} = (z_1, \ldots, z_m)$ be the vector of integrators that occur in the formula ψ, and let $U_1, \ldots, U_n$ be the corresponding types (as specified by ψ). From the n-dimensional hybrid automaton $A = (\vec{x}, V, \mathit{inv}, \mathit{dif}, E, \mathit{act}, L, \mathit{syn})$ we construct the $(n+m)$-dimensional *$\vec{z}$-extension* of A as the linear hybrid automaton $A_{\vec{z}} = (\vec{x} \uplus \vec{z}, V, \mathit{inv}, \mathit{dif}', E, \mathit{act}', L, \mathit{syn})$:

- For each integrator z_i in $\vec{z}$, and each location $v \in V$, if $v \in U_i$ then $\mathit{dif}'(v) = \mathit{dif}(v) \wedge \dot{z}_i = 1$; if $v \notin U_i$ then $\mathit{dif}'(v) = \mathit{dif}(v) \wedge \dot{z}_i = 0$.

- For each integrator z_i in $\vec{z}$, and each transition $e \in E$, $\mathit{act}'(e) = \mathit{act}(e) \wedge z_i' = z_i$.

Each data state $\vec{s} \in \mathbb{R}^{n+m}$ of $A_{\vec{z}}$ consists of an *$\vec{x}$-projection* $\vec{s}|_{\vec{x}} \in \mathbb{R}^n$, which is a data state of A, and a *$\vec{z}$-projection* $\vec{s}|_{\vec{z}} \in \mathbb{R}^m$, which assigns to each integrator in $\vec{z}$ a real value. Each state $\sigma = (v, \vec{s})$ of $A_{\vec{z}}$, then, consists of a state $\sigma|_{\vec{x}} = (v, \vec{s}|_{\vec{x}})$ of A, and an integrator valuation $\sigma|_{\vec{z}} = \vec{s}|_{\vec{z}}$. In particular, $\Sigma_{A_{\vec{z}}} = \Sigma_A \times \mathbb{R}^m$.

The projection operation $|_{\vec{x}}$ is extended to regions and trajectories in the natural way: the $\vec{x}$-projection of the $A_{\vec{z}}$-region R is the A-region that contains the $\vec{x}$-projections of all states in R; the $\vec{x}$-projection of the data trajectory (δ, ρ) of $A_{\vec{z}}$ is the data trajectory of A that maps every real $t \in [0, \delta]$ to the data state $\rho(t)|_{\vec{x}}$; the $\vec{x}$-projection of the $A_{\vec{z}}$-trajectory τ is the A-trajectory that results from τ by replacing all data trajectories with their $\vec{x}$-projections. Notice that each trajectory τ of $A_{\vec{z}}$ is completely determined by the trajectory $\tau|_{\vec{x}}$ of A, and the initial integrator valuation $\tau(0,0)|_{\vec{z}} \in \mathbb{R}^m$.

Given a set $\mathcal{T}$ of A-trajectories, the *$\vec{z}$-extension* $\mathcal{T}_{\vec{z}}$ consists of all $A_{\vec{z}}$-trajectories whose $\vec{x}$-projections are in $\mathcal{T}$. For a state σ of $A_{\vec{z}}$, the satisfaction relation $\sigma \models_{\mathcal{T}} \psi$ is defined inductively on the subformulas of ψ:

$\sigma \models_{\mathcal{T}} \phi$ iff $\sigma \in [\![\phi]\!]$;

$\sigma \models_{\mathcal{T}} \neg\varphi$ iff $\sigma \not\models_{\mathcal{T}} \varphi$;

$\sigma \models_{\mathcal{T}} \varphi_1 \vee \varphi_2$ iff $\sigma \models_{\mathcal{T}} \varphi_1$ or $\sigma \models_{\mathcal{T}} \varphi_2$;

$\sigma \models_{\mathcal{T}} \varphi_1 \exists\mathcal{U} \varphi_2$ iff for some trajectory $\tau \in \mathcal{T}_{\vec{z}}$ with $\tau(0,0) = \sigma$, there is a position π of τ such that $\tau(\pi) \models_{\mathcal{T}} \varphi_2$, and for all positions π' of τ, if $\pi' \leq \pi$ then $\tau(\pi') \models_{\mathcal{T}} \varphi_1 \vee \varphi_2$;

$\sigma \models_{\mathcal{T}} \varphi_1 \forall\mathcal{U} \varphi_2$ iff for all trajectories $\tau \in \mathcal{T}_{\vec{z}}$ with $\tau(0,0) = \sigma$, there is a position π of τ such that $\tau(\pi) \models_{\mathcal{T}} \varphi_2$, and for all positions π' of τ, if $\pi' \leq \pi$ then $\tau(\pi') \models_{\mathcal{T}} \varphi_1 \vee \varphi_2$;

$\sigma \models_{\mathcal{T}} (z : p).\, \varphi$ iff $\sigma[z := 0] \models_{\mathcal{T}} \varphi$, where $\sigma[z := 0]$ is the state that differs from σ at most in the value of z, which is 0.

The disjunctions in the definitions of the temporal operators $\exists\mathcal{U}$ and $\forall\mathcal{U}$ account for the possibility that the second argument φ_2 may hold throughout a left-open interval of a trajectory [HNSY94].

We write $[\psi]_{\mathcal{T}}$ for the $A_{\vec{z}}$-region of all states σ such that $\sigma \models_{\mathcal{T}} \psi$. Since ψ is closed, if $\sigma \models_{\mathcal{T}} \psi$ and $\sigma|_{\vec{x}} = \sigma'|_{\vec{x}}$, then $\sigma' \models_{\mathcal{T}} \psi$; that is, $[\psi]_{\mathcal{T}} = ([\psi]_{\mathcal{T}})|_{\vec{x}} \times \mathbb{R}^m$. The *characteristic A-region* $[\![\psi]\!]_A$ is defined to be the $\vec{x}$-projection of the $A_{\vec{z}}$-region $[\psi]_{[\![A]\!]^{div}}$. The state σ of the automaton A *satisfies* the formula ψ if $\sigma \in [\![\psi]\!]_A$.

3.3 Example: Railroad gate controller

The linear hybrid automaton A *meets* the requirement specified by the A-formula ψ of ICTL iff all admissible states of A satisfy ψ; that is, $[\![\psi]\!]_A = \Sigma_A$.

To illustrate the use of ICTL as a specification language, recall the railroad gate controller from Section 2. The initial condition of the system is given by the state predicate

$$\phi_0 : \quad \ell = (\mathit{far}, \mathit{open}, \mathit{idle}),$$

which asserts that the train is far from the gate, which is open, and the controller is idle. We require the following properties of the controller. The *safety property*

$$\phi_0 \;\rightarrow\; \forall\Box(x \leq 10 \;\rightarrow\; \ell[2] = \mathit{closed})$$

asserts that whenever a train is within 10 meters of the gate, the gate must be closed (we write $\ell[i]$ for the i-th component of the program counter ℓ, so $\ell[1]$ ranges over the locations of the train automaton, etc.). Since the safety requirement is met by a controller that keeps the gate closed forever, we add the *liveness (response) property*

$$\phi_0 \;\rightarrow\; \forall\Box\forall\Diamond(\ell[2] = open)$$

that the gate will always open again. Indeed, this eventual liveness requirement may not be satisfactory (imagine you are in a car waiting to cross at a closed gate!), so we may wish to require instead the stronger *time-bounded response property*

$$\phi_0 \;\rightarrow\; \forall\Box\forall\Diamond_{\leq 33}\,(\ell[2] = open)$$

that the gate will always open within 33 seconds.

To demonstrate the use of integrators, we consider the additional requirement that within any time interval longer than an hour, the gate must be open at least 80% of the time. This *duration (utility) property* can be expressed in ICTL by the formula

$$\phi_0 \;\rightarrow\; \forall\Box(z_1.\,(z_2\colon \ell[2] = open).\,\forall\Box(z_1 \geq 3600 \;\rightarrow\; 10z_2 \leq 8z_1).$$

Here z_1 is a clock that measures the length of a time interval, and z_2 is an integrator that measures the accumulated time that the gate is open during the interval measured by z_1.

4 Hybrid Automata as Infinite-state Transition Systems

We analyze hybrid automata by building on techniques for the analysis of discrete transition systems, which move in discrete steps through a state space. Suppose we are given a linear hybrid automaton A. Since the trajectories of A move continuously through the infinite state space Σ_A, we decompose each trajectory into a countable number of steps. Each step records a time delay (of arbitrary finite duration), or a transition of A (instantaneous).

4.1 Step relations

Since a time delay, on its way from an initial state to a target state, passes through an infinite number of intermediate states, we need to record the region that is visited during the delay. This leads to the following definitions of the time-step and transition-step relations. Let $Q = \bigcup_v (v, Q_v)$ be a region of A.

Time step For all states $\sigma_1 = (v_1, \vec{s}_1)$ and $\sigma_2 = (v_2, \vec{s}_2)$ of A, $\sigma_1 \xrightarrow{Q} \sigma_2$ if $v_1 = v_2$, and there exists a data trajectory (δ, ρ) such that

(1) $\rho(0) = \vec{s}_1$ and $\rho(\delta) = \vec{s}_2$;
(2) for all reals $t \in [0, \delta]$, $\rho_i(t) \in [\![inv(v)]\!] \cap Q_v$;
(3) for all reals $t \in (0, \delta)$, $\frac{d\rho(t)}{dt} \in [\![dif(v)]\!]$.

In other words, $(v, \vec{s}_1) \xrightarrow{Q} (v, \vec{s}_2)$ iff in location v, starting from the data state $\vec{s}_1$, it is possible to reach the data state $\vec{s}_2$ by letting time pass, without leaving the region Q. In this case, we call (δ, ρ) the *witness* for $(v, \vec{s}_1) \xrightarrow{Q} (v, \vec{s}_2)$. We define $\sigma_1 \xrightarrow{Q}_{lin} \sigma_2$ if there is a piecewise-linear witness for $\sigma_1 \xrightarrow{Q} \sigma_2$; and $\sigma_1 \xrightarrow{Q}_1 \sigma_2$, if there is a linear witness for $\sigma_1 \xrightarrow{Q} \sigma_2$. Clearly, $\xrightarrow{Q}_1 \;\subseteq\; \xrightarrow{Q}_{lin} \;\subseteq\; \xrightarrow{Q}$. For simplicity, we write $\rightarrow$ for $\xrightarrow{\Sigma_A}$.

Transition step For all states σ_1 and σ_2 of A, $\sigma_1 \stackrel{Q}{\mapsto} \sigma_2$ if $\sigma_1, \sigma_2 \in \Sigma_A \cap Q$, and there exists a transition $e \in E$ such that $\sigma_2 \in [\![act(e)]\!](\sigma_1)$.

The binary relation $\stackrel{Q}{\Rightarrow}$ on the states of A is *Q-reflexive* if (1) $\sigma_1 \stackrel{Q}{\Rightarrow} \sigma_2$ implies $\sigma_1, \sigma_2 \in \Sigma_A \cap Q$, and (2) for all $\sigma \in \Sigma_A \cap Q$, we have $\sigma \stackrel{Q}{\Rightarrow} \sigma$. The time-step relation $\stackrel{Q}{\rightarrow}$ is Q-reflexive because of witness trajectories with duration 0, and the transition-step relation $\stackrel{Q}{\mapsto}$ is Q-reflexive because of stutter transitions.

We now show that for linear regions Q the time-step relation $\stackrel{Q}{\rightarrow}$ and the piecewise-linear time-step relation $\stackrel{Q}{\rightarrow}_{lin}$ coincide. For this purpose, we need a lemma and a few definitions.

Lemma 1 *Let A be a linear hybrid automaton, let v be a location of A, and let S be a convex data region contained in $[\![inv(v)]\!]$. If there is a data trajectory (ρ, δ) of A such that $\rho(0) = \vec{s}_1 \in S$ and $\rho(\delta) = \vec{s}_2 \in S$, then there is a linear data trajectory (ρ', δ) of A such that $\rho'(0) = \vec{s}_1$, $\rho'(\delta) = \vec{s}_2$, and $\rho'(t) \in S$ for all $t \in [0, \delta]$.*

Proof. Suppose that $\rho(0) = \vec{s}_1$ and $\rho(\delta) = \vec{s}_2$. We define a continuous function $\rho' : [0, \delta] \rightarrow \mathbb{R}^n$ such that $\rho'(t) = \vec{s}_1 + t \cdot (\frac{\vec{s}_2 - \vec{s}_1}{\delta})$. Then $\rho'(0) = \vec{s}_1$, $\rho'(\delta) = \vec{s}_2$, and $\frac{d\rho'(\vec{x})(t)}{dt} = \frac{\vec{s}_2 - \vec{s}_1}{\delta}$ for all $t \in (0, \delta)$.

Suppose that the rate predicate $dif(v)$ has the form $\bigwedge_{i=1,\ldots,k} r_i$. Let $r_i = (c_0 \sim \vec{c} \cdot \dot{\vec{x}})$ be a conjunct of $dif(v)$, where $\sim \in \{<, \leq\}$ and $\vec{c} \cdot \dot{\vec{x}}$ is the inner product of a constant vector $\vec{c}$ and vector $\dot{\vec{x}}$. Since (δ, ρ) is a data trajectory, for all time instants $t \in [t_i, t_{i+1}]$,

$$c_0 \sim \vec{c} \cdot \frac{d\rho(\vec{x})(t)}{dt}.$$

Integrating both sides of the above inequality from 0 to δ, we get $c_0(\delta) \sim \vec{c} \cdot (\vec{s}_2 - \vec{s}_1)$; that is,

$$c_0 \sim \vec{c} \cdot \frac{\vec{s}_2 - \vec{s}_1}{\delta}.$$

Since $\frac{d\rho'(\vec{x})(t)}{dt} = \frac{\vec{s}_2 - \vec{s}_1}{\delta}$ for all $t \in (0, \delta)$, we know that $\frac{d\rho'(\vec{x})(t)}{dt} \in [\![r_i]\!]$, for all $t \in (0, \delta)$. Moreover, since r_i is an arbitrary conjunct of $dif(v)$, we can conclude that $\frac{d\rho'(\vec{x})(t)}{dt} \in [\![dif(v)]\!]$ for all $t \in (0, \delta)$.

In addition, ρ' is linear, and both of its endpoints, $\vec{s}_1$ and $\vec{s}_2$, are in some convex data region S, so $\rho'(t) \in S$ for all $t \in [0, \delta]$. This proves our claim that (δ, ρ') is a linear data trajectory such that $\rho'(0) = \rho(0)$, $\rho'(\delta) = \rho(\delta)$ and $\rho'(t) \in S$ for all $t \in [0, \delta]$. ■

Let p be a data predicate. A *closed convex covering* of p is a set $\{p_1, \ldots, p_k\}$ of closed convex data predicates p_i such that $[\![p]\!] \subseteq \bigcup_{1 \leq i \leq k} [\![p_i]\!]$. A closed convex covering of p can be easily constructed from the disjunctive normal form of p. Assume that $p = p_1 \vee \ldots \vee p_k$ is in disjunctive normal form, with each disjunct p_i of the form $\bigwedge_j (e_j \sim c_j)$, where e_j is a linear term and c_j is an integer constant. We define the data predicate transformer *close* such that the data predicate $close(p)$ results from the data predicate p by replacing each strict inequality $e_j > c_j$ or $e_j < c_j$ by the corresponding nonstrict inequality $e_j \geq c_j$ or $e_j \leq c_j$, respectively. Then $\{close(p_1), \ldots, close(p_k)\}$ is a closed convex covering of p.

If $\{p_1, \ldots, p_k\}$ is a closed convex covering of the data predicate p, then the set $C = \{p_1 \wedge p, \ldots, p_k \wedge p\}$ of convex data predicates is an *exact convex covering* of p; that is, $[\![p]\!] = \bigcup_{1 \leq i \leq k} [\![p_i \wedge p]\!]$. We call each convex data region $[\![p_i \wedge p]\!]$ a *patch* of C.

Theorem 1 *Let A be a linear hybrid automaton, let $Q = \bigcup_v (v, Q_v)$ be a region of A, let v be a location of A, and let $\vec{s}_1$ and $\vec{s}_2$ be two data states of A. If Q_v is linear, then $(v, \vec{s}_1) \stackrel{Q}{\rightarrow} (v, \vec{s}_2)$ iff $(v, \vec{s}_1) \stackrel{Q}{\rightarrow}_{lin} (v, \vec{s}_2)$. If Q_v is convex linear, then $(v, \vec{s}_1) \stackrel{Q}{\rightarrow} (v, \vec{s}_2)$ iff $(v, \vec{s}_1) \stackrel{Q}{\rightarrow}_1 (v, \vec{s}_2)$.*

Proof. It is clear that $(v,\vec{s}_1) \xrightarrow{Q}_{lin} (v,\vec{s}_2)$ implies $(v,\vec{s}_1) \xrightarrow{Q} (v,\vec{s}_2)$. We show that $(v,\vec{s}_1) \xrightarrow{Q} (v,\vec{s}_2)$ implies $(v,\vec{s}_1) \xrightarrow{Q}_{lin} (v,\vec{s}_2)$. We say a data trajectory is *trivial* if its duration is 0. If $\vec{s}_1 = \vec{s}_2$, then a witness of $(v,\vec{s}_1) \xrightarrow{Q} (v,\vec{s}_2)$ is trivial, and thus $(v,\vec{s}_1) \xrightarrow{Q}_{lin} (v,\vec{s}_2)$. So assume that $\vec{s}_1 \neq \vec{s}_2$, and therefore, every witness for $(v,\vec{s}_1) \xrightarrow{Q} (v,\vec{s}_2)$ is not trivial. Since Q_v is linear, there is a data predicate p_v such that $[\![p_v]\!] = Q_v$. Let $C = \{p_1, \ldots, p_k\}$ be an exact convex covering of $p_v \wedge inv(v)$. A *finite crossing sequence* of a nontrivial data trajectory (δ, ρ) on C is a time sequence $(0 = t_0, t_1, t_2, \ldots, t_m = \delta)$ with $t_0 < t_1 < t_2 < \cdots < t_m$ such that (1) for each $1 \leq i \leq m-1$, $\rho(t_i)$ is in two distinct patches of C, and (2) for each $0 \leq i \leq m-1$, both $\rho(t_i)$ and $\rho(t_{i+1})$ are in the same patch of C. Notice that although the two endpoints $\rho(t_i)$ and $\rho(t_{i+1})$ of the data trajectory $\rho[t_i, t_{i+1}]$ are in the same patch, say $[\![p_j]\!]$, of C, the interior points of $\rho[t_i, t_{i+1}]$ may not all be in $[\![p_j]\!]$. Since C is finite, every nontrivial data trajectory (δ, ρ) must have a finite crossing sequence.

Consider a finite crossing sequence $(t_0 = 0, t_1, \ldots, t_m = \delta)$ of a witness (δ, ρ) for $(v,\vec{s}_1) \xrightarrow{Q} (v,\vec{s}_2)$ on C. For each $i \in \{0, \ldots, m-1\}$, $\rho[t_i, t_{i+1}]$ is a data trajectory such that both data state $\rho(t_i)$ and $\rho(t_{i+1})$ are in the convex data region $[\![p_j \wedge inv(v)]\!]$, where $[\![p_j]\!]$ is a patch of C. By Lemma 1, there is a linear data trajectory $(t_{i+1} - t_i, \rho_i)$ such that $\rho_i(0) = \rho(t_i)$, $\rho_i(t_{i+1} - t_i) = \rho(t_{i+1})$ and $\rho_i(t) \in [\![p_j \wedge inv(v)]\!]$ for all $t \in [0, t_{i+1} - t_i]$. So the concatenation of these linear data trajectories $(t_1, \rho_0), (t_2 - t_1, \rho_1), \ldots, (\delta - t_{m-1}, \rho_{m-1})$ is a piecewise-linear witness for $(v,\vec{s}_1) \xrightarrow{Q} (v,\vec{s}_2)$, and thus $(v,\vec{s}_1) \xrightarrow{Q}_{lin} (v,\vec{s}_2)$.

On the other hand, if Q_v is convex, then $C = \{p_v \wedge inv(v)\}$ is an exact convex covering of $p_v \wedge inv(v)$. Then it follows immediately from Lemma 1 that $(v,\vec{s}_1) \xrightarrow{Q} (v,\vec{s}_2)$ if and only if $(v,\vec{s}_1) \xrightarrow{Q}_1 (v,\vec{s}_2)$. ■

4.2 Precondition operators

Let Q and R be two regions of the hybrid automaton A, and let $\overset{Q}{\Rightarrow}$ be a Q-reflexive binary relation on the states of A. The $\overset{Q}{\Rightarrow}$*-precondition* $pre^{Q}_{\Rightarrow}(R)$ of R is the region of A from which a state in R can be reached in a single $\overset{Q}{\Rightarrow}$-step; that is, $\sigma \in pre^{Q}_{\Rightarrow}(R)$ if there is a state $\sigma' \in R$ such that $\sigma \overset{Q}{\Rightarrow} \sigma'$. Since $\overset{Q}{\Rightarrow}$ is Q-reflexive, the precondition operator $pre_{\Rightarrow}$ is monotonic on the subregions of $\Sigma_A \cap Q$; that is, for all regions $R \subseteq \Sigma_A \cap Q$, we have $R \subseteq pre^{Q}_{\Rightarrow}(R) \subseteq \Sigma_A \cap Q$.

We consider the *time-precondition* operator $pre_{\rightarrow}$ and the *transition-precondition* operator $pre_{\mapsto}$, and show that if both regions Q and R are linear, then so are the preconditions $pre^{Q}_{\rightarrow}(R)$ and $pre^{Q}_{\mapsto}(R)$. This is done by constructing from the state predicates ϕ and χ that define Q and R, respectively, two state predicates $pre^{\phi}_{\rightarrow}(\chi)$ and $pre^{\phi}_{\mapsto}(\chi)$ that define $pre^{Q}_{\rightarrow}(R)$ and $pre^{Q}_{\mapsto}(R)$, respectively. We then define the A*-precondition* $pre^{Q}_{A}(R)$ to be the union $pre^{Q}_{\rightarrow}(R) \cup pre^{Q}_{\mapsto}(R)$. If ϕ defines Q, and χ defines R, then $pre^{Q}_{A}(R)$ is defined by the state predicate

$$pre^{\phi}_{A}(\chi) \;=\; pre^{\phi}_{\rightarrow}(\chi) \vee pre^{\phi}_{\mapsto}(\chi).$$

In the following, suppose that $Q = \bigcup_v (v, Q_v)$ and $R = \bigcup_v (v, R_v)$. Let $\phi = \bigcup_v (v, q_v)$ be a state predicate such that for each location v of A, $[\![q_v]\!] = Q_v$, and let $\chi = \bigcup_v (v, r_v)$ be a state predicate such that for each location v of A, $[\![r_v]\!] = R_v$.

Time precondition

We write $pre^{Q_v}_{\rightarrow}(v; R_v)$ for the data region such that from any state in the region $(v, pre^{Q_v}_{\rightarrow}(v; R_v))$ a state in the region (v, R_v) can be reached in a single $\xrightarrow{Q}$-step. We show that the data region $pre^{Q_v}_{\rightarrow}(v; R_v)$ is linear by constructing from q_v and r_v a data predicate $pre^{q_v}_{\rightarrow}(v; r_v)$ that defines the data region $pre^{Q_v}_{\rightarrow}(v;R_v)$. Then

$$pre^{\phi}_{\rightarrow}(\chi) \;=\; \bigcup_{v\in V}(v, pre^{q_v}_{\rightarrow}(v;r_v)).$$

The construction of $pre^{q_v}_{\rightarrow}(v;r_v)$ proceeds in two steps. First we construct a data predicate $pre^{true}_{\rightarrow}(v;r_v)$ such that $[\![pre^{true}_{\rightarrow}(v;r_v)]\!] = pre^{[\![inv(v)]\!]}_{\rightarrow}(v;R_v)$. Then we apply the precondition operator $pre^{true}_{\rightarrow}$ repeatedly to construct the data predicate $pre^{q_v}_{\rightarrow}(v;r_v)$.

Lemma 2 *Let A be a linear hybrid automaton, let v be a location of A, and let r_v be a data predicate of A. If*

$$pre^{true}_{\rightarrow}(v;r_v) \;=\; inv(v) \wedge (\exists\delta\geq 0.\, \exists\vec{d}.\, dif(v)[\vec{\dot{x}} := \vec{d}] \wedge (r_v \wedge inv(v))[\vec{x} := \vec{x}+\delta\cdot\vec{d}]),$$

then $[\![pre^{true}_{\rightarrow}(v;r_v)]\!] = pre^{[\![inv(v)]\!]}_{\rightarrow}(v;[\![r_v]\!])$.

Proof. The quantified formula in $pre^{true}_{\rightarrow}(v;r_v)$ specifies that a data state $\vec{x}$ satisfies $pre^{true}_{\rightarrow}(v;r_v)$ iff the following four conditions hold:

1. $\vec{x}$ is in $[\![inv(v)]\!]$ (the conjunct $inv(v)$);
2. there exist a duration δ and a slope vector $\vec{d}$ that define a linear witness $(\delta, \rho(t) = \vec{x}+t\cdot\vec{d})$ for a single $\rightarrow$ step (the quantified variables δ and $\vec{d}$);
3. the slope vector $\vec{d}$ satisfies the rate predicate $dif(v)$ of location v (the conjunct $dif(v)[\vec{\dot{x}} := \vec{d}]$); and
4. $\rho(\delta)$ is $[\![r]\!] \cap [\![inv(v)]\!]$ (the conjuncts $(r \wedge inv(v))[\vec{x} := \vec{x}+\delta\cdot\vec{d}]$).

Since $[\![inv(v)]\!]$ is convex, by Theorem 1, we know that the time-step relation $\overset{[\![(v,inv(v))]\!]}{\rightarrow} = \overset{[\![(v,inv(v))]\!]}{\rightarrow}_1$. So it suffices to consider only linear witnesses in Condition 2. In addition, since the witness is linear, and since both $\rho(0)$ and $\rho(\delta)$ are in $[\![inv(v)]\!]$, Condition 1 together with Condition 4 implies that $\rho(t) \in [\![inv(v)]\!]$ for all $t\in[0,\delta]$.

According to the definition of the time-step relation $\overset{[\![(v,inv(v))]\!]}{\rightarrow}_1$, it is clear that $[\![pre^{true}_{\rightarrow}(v;r_v)]\!] = pre_{\rightarrow}(v;R_v)$. ■

The formula $pre^{true}_{\rightarrow}(v;r_v)$ of Lemma 2 contains the vector $\delta\cdot\vec{d}$ of variable products, which gives rise to nonlinear terms. We therefore replace the vector $\delta\cdot\vec{d}$ of variable products by a vector $\vec{c}$ of variables. Let $\delta\cdot dif(v)$ be the rate predicate that results from multiplying each constant of the rate predicate $dif(v)$ by the variable δ. For example, if r is the rate predicate $\dot{x}_1 \leq 3\dot{x}_2+6 \wedge \dot{x}_3 = 1$, then $\delta\cdot r$ is the rate predicate $\dot{x}_1 \leq 3\dot{x}_2 + 6\delta \wedge \dot{x}_3 = \delta$. Then the formula

$$inv(v) \wedge (\exists\delta\geq 0.\, \exists\vec{d}.\, dif(v)[\vec{\dot{x}} := \vec{d}] \wedge (r_v \wedge inv(v))[\vec{x} := \vec{x}+\delta\cdot\vec{d}])$$

is equivalent to the formula

$$inv(v) \wedge (\exists\delta\geq 0.\, \exists\vec{c}.\, (\delta\cdot dif(v))[\vec{\dot{x}} := \vec{c}] \wedge (r_v \wedge inv(v))[\vec{x} := \vec{x}+\vec{c}]).$$

The next proposition follows.

Proposition 1 *Let A be a linear hybrid automaton, let v be a location of A, and let r_v be a data predicate of A. If*

$$pre^{true}_{\rightarrow}(v;r_v) \;=\; inv(v) \wedge (\exists\delta\geq 0.\, \exists\vec{c}.\, (\delta\cdot dif(v))[\vec{\dot{x}} := \vec{c}] \wedge (r_v \wedge inv(v))[\vec{x} := \vec{x}+\vec{c}]),$$

then $[\![pre^{true}_{\rightarrow}(v;r_v)]\!] = pre^{[\![inv(v)]\!]}_{\rightarrow}(v;[\![r_v]\!])$.

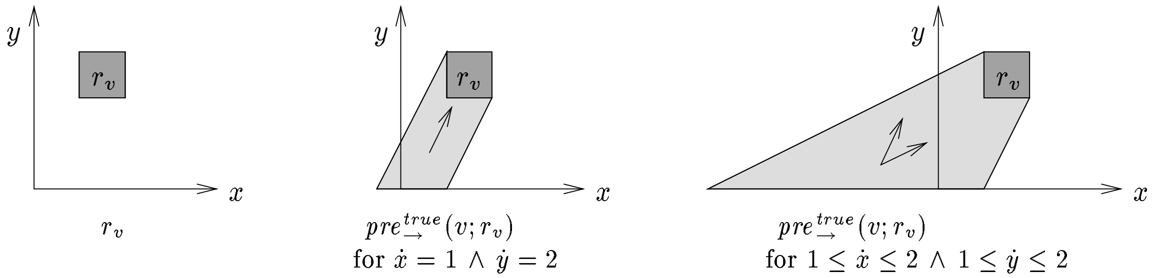

Figure 2: The time-precondition operator $pre^{true}_{\rightarrow}$

The formula $pre^{true}_{\rightarrow}(v; r_v)$ is a formula of the first-order theory $(\mathbb{R}, \leq, +)$ of the reals with addition. Since this theory admits quantifier elimination, the formula $pre^{true}_{\rightarrow}(v; r_v)$ is equivalent to a data predicate. The quantifier-elimination procedure used by HyTech is discussed in Section 6.

Example. Let us consider two examples of computing the data predicate $pre^{true}_{\rightarrow}(v; r_v)$ using Proposition 1. First, suppose that the linear hybrid automaton A has two data variables, x and y, the invariant $inv(v) = (y \geq 0)$, and the activity $dif(v) = (\dot{x} = 1 \wedge \dot{y} = 2)$ for the location v. Consider the data predicate $r_v = (1 \leq x \leq 2 \wedge 2 \leq y \leq 3)$ (see the left of Figure 2). Then, according to Proposition 1,

$$pre^{true}_{\rightarrow}(v; r_v) \;=\; (y \geq 0 \wedge (\exists \delta \geq 0.\, \exists c_1, c_2.\, c_1 = \delta \wedge c_2 = 2\delta \wedge 1 \leq x + c_1 \leq 2 \wedge 2 \leq y + c_2 \leq 3)).$$

Eliminating the two existential quantifiers inside out, we obtain

$$pre^{true}_{\rightarrow}(v; r_v) \;=\; (y \geq 0 \wedge (\exists \delta \geq 0.\, 1 \leq x + \delta \leq 2 \wedge 2 \leq y + 2\delta \leq 3))$$

and, finally, the data predicate

$$pre^{true}_{\rightarrow}(v; r_v) \;=\; (x \leq 2 \wedge 0 \leq y \leq 3 \wedge -2 \leq y - 2x \leq 1)$$

(see the center of Figure 2). Second, suppose that the activity $dif(v)$ of the location v is $1 \leq \dot{x} \leq 2 \wedge 1 \leq \dot{y} \leq 2$. Then, according to Proposition 1,

$$pre^{true}_{\rightarrow}(v; r_v) \;=\; (y \geq 0 \wedge (\exists \delta \geq 0.\, \exists c_1, c_2.\, \delta \leq c_1 \leq 2\delta \wedge \delta \leq c_2 \leq 2\delta \wedge 1 \leq x + c_1 \leq 2 \wedge 2 \leq y + c_2 \leq 3)).$$

Eliminating the two existential quantifiers, we obtain

$$pre^{true}_{\rightarrow}(v; r_v) \;=\; (y \geq 0 \wedge (\exists \delta \geq 0.\, 1 - x \leq 2\delta \wedge 2 - y \leq 2\delta \wedge \delta \leq 2 - x \wedge \delta \leq 3 - y))$$

and the equivalent data predicate

$$pre^{true}_{\rightarrow}(v; r_v) \;=\; (x \leq 2 \wedge 0 \leq y \leq 3 \wedge 2x - y \leq 2 \wedge 2y - x \leq 5)$$

(see the right of Figure 2). ■

We now reduce the construction of the data predicate $pre^{q_v}_{\rightarrow}(v, r_v)$ to a sequence of applications of the precondition operator $pre^{true}_{\rightarrow}$ defined in Proposition 1. We proceed in three steps. First, let v' be a new, fictitious location with the invariant $inv(v') = (inv(v) \wedge q_v)$ and the activity $dif(v)$. Then

$$pre^{Q_v}_{\rightarrow}(v; R_v) \;=\; pre^{[\![inv(v')]\!]}_{\rightarrow}(v'; R_v).$$

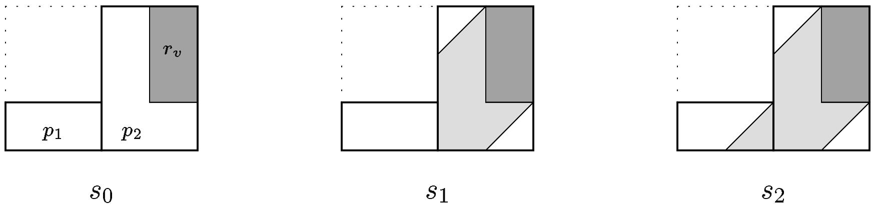

Figure 3: The time-precondition operator $pre_{\rightarrow}^{q_v}$

The invariant $inv(v')$, however, may not be convex, in which case we cannot compute a data predicate $pre_{\rightarrow}^{true}(v'; r_v)$ using Proposition 1.

So, second, we split the location v' into several locations with convex invariants. Let $C = \{p_1, \ldots, p_k\}$ be an exact convex covering of the data predicate $inv(v')$. We split v' into the set $V' = \{v'_1, \ldots, v'_k\}$ of new, fictitious locations v'_i such that for all $1 \leq i \leq k$, the invariant $inv(v'_i)$ is p_i, and the activity $dif(v'_i)$ is $dif(v)$. Let

$$pre_{\rightarrow}(V'; R_v) \;=\; \bigcup_{v'_i \in V'} pre_{\rightarrow}^{[\![inv(v'_i)]\!]}(v'_i; R_v).$$

Then the data predicate $pre_{\rightarrow}(V'; r_v)$ that defines the data region $pre_{\rightarrow}(V'; R_v)$ can be constructed using Proposition 1 and disjunction.

A single $\xrightarrow{Q}$-step, however, may proceed from the data region $pre_{\rightarrow}^{Q_v}(v; R_v)$ to the data region R_v through more than one of the patches of C. Since there are k different patches of C, it will suffice to iterate the precondition $pre_{\rightarrow}(V'; R_v)$ k times. So, third, we define a sequence of k data regions, S_0 to S_k, such that $S_0 = S_v$ and for all $1 \leq j \leq k$, $S_j = pre_{\rightarrow}(V'; S_{j-1})$. Let $s_0, \ldots, s_k$ be the sequence of corresponding data predicates; that is, for all $0 \leq j \leq k$, $[\![s_j]\!] = S_j$. The next proposition shows that $S_k = pre_{\rightarrow}^{[\![inv(v')]\!]}(v'; R_v)$. Thus the data predicate $pre_{\rightarrow}^{q_v}(v; r_v) = s_k$ defines the data region $pre_{\rightarrow}^{Q_v}(v; R_v)$.

Proposition 2 *Let A be a hybrid automaton, let v be a location of A, and let q_v and r_v be two data predicates of A. Let $\{p_1, \ldots, p_k\}$ be an exact convex covering of the data predicate $inv(v) \wedge q_v$. Let $V' = \{v'_1, \ldots, v'_k\}$ be a set of locations such that for all $1 \leq i \leq k$, $inv(v'_i) = p_i$ and $dif(v'_i) = dif(v')$. If $S_0 = [\![r_v]\!]$ and for all $1 \leq j \leq k$, $S_j = pre_{\rightarrow}(V'; S_{j-1})$, then $S_k = pre_{\rightarrow}^{[\![q_v]\!]}(v; [\![r_v]\!])$.*

Proof. Since $dif(v'_i) = dif(v')$ and $\bigvee_{v'_i} inv(v'_i) = inv(v')$, by definition, S_k defines the set of data states that can reach a data state in R_v by k $\rightarrow_1$ steps. According to the definition of $pre_{\rightarrow}(v'; R_v)$, $S_k \subseteq pre_{\rightarrow}(v'; R_v)$. Since $inv(v')$ has an exact convex covering $\{p_1, \ldots, p_k\}$, the proof of Theorem 1 implies that the time-step relation $(v', \vec{s}) \rightarrow (v', \vec{s'})$ must have a piecewise-linear witness that consists of k linear data trajectories. In other words, $(v', \vec{s}) \rightarrow (v', \vec{s'})$ implies that $(v', \vec{s}) \rightarrow_1{}^j (v', \vec{s'})$ for some $0 \leq j \leq k$. Consequently, $pre_{\rightarrow}(v'; R_v) \subseteq S_k$. Thus $pre_{\rightarrow}(v'; R_v) = S_k$. ■

Example. Let us consider an example of computing the data predicate $pre_{\rightarrow}^{q_v}(v; r_v)$ using Proposition 2. Suppose that the linear hybrid automaton A has two data variables, x and y, the invariant $inv(v) = (0 \leq x \leq 4 \wedge 0 \leq y \leq 3)$ and the activity $dif(v) = (\dot{x} = 1 \wedge \dot{y} = 1)$ for the location v. Consider the data predicates

$$q_v \;=\; ((0 \leq x \leq 2 \wedge 0 \leq y \leq 1) \vee (2 \leq x \leq 4 \wedge 0 \leq y \leq 3))$$

and $r_v = (3 \leq x \leq 4 \wedge 2 \leq y \leq 3)$. The new location v' has the invariant

$$inv(v') \;=\; (0 \leq x \leq 2 \wedge 0 \leq y \leq 1) \vee (2 \leq x \leq 4 \wedge 0 \leq y \leq 3).$$

Since $inv(v')$ is not convex, and

$$\{p_1, p_2\} \;=\; \{0 \leq x \leq 2 \wedge 0 \leq y \leq 1,\; 2 \leq x \leq 4 \wedge 0 \leq y \leq 3\}$$

is an exact convex covering of $inv(v')$, we split the location v' into two locations $V' = \{v_1, v_2\}$ such that $inv(v_1) = p_1$ and $inv(v_2) = p_2$. Then

$$\begin{array}{lllll}
s_0 & = & r_v & = & (3 \leq x \leq 4 \wedge 2 \leq y \leq 3), \\
s_1 & = & pre_{\rightarrow}(V'; s_0) & = & (2 \leq x \leq 4 \wedge 0 \leq y \leq 3 \wedge 1 \leq x - y \leq 3), \\
s_2 & = & pre_{\rightarrow}(V'; s_1) & = & ((2 \leq x \leq 4 \wedge 0 \leq y \leq 3 \wedge 1 \leq x - y \leq 3) \vee \\
 & & & & (1 \leq x \leq 2 \wedge 0 \leq y \leq 1 \wedge 1 \leq x - y))
\end{array}$$

(see Figure 3). By Proposition 2, $pre^{q_v}_{\rightarrow}(v; r_v) = s_2$. ■

Transition precondition

We write $pre^{Q}_{\mapsto}(v; R_v)$ for the region of states from which a state in (v, R_v) can be reached in a single $\overset{Q}{\mapsto}$-step. We show that the region $pre^{Q}_{\mapsto}(v; R_v)$ is linear by constructing from ϕ and r_v a state predicate $pre^{\phi}_{\mapsto}(v; r_v)$ that defines the region $pre^{Q}_{\mapsto}(v; R_v)$. Then

$$pre^{\phi}_{\mapsto}(\chi) \;=\; \bigcup_{v \in V} pre^{\phi}_{\mapsto}(v; r_v).$$

The next proposition constructs $pre^{\phi}_{\mapsto}(v; r_v)$ as a formula of the theory $(\mathbb{R}, \leq, +)$, from which a data predicate can be obtained by quantifier elimination.

Proposition 3 *Let A be a linear hybrid automaton with the transition set E, let v be a location of A, let $\phi = \bigcup_v (v, q_v)$ be a state predicate of A, and let r_v be a data predicate of A. If*

$$pre^{\phi}_{\mapsto}(v; r_v) \;=\; \bigcup_{(v',v) \in E} (v',\; q_{v'} \wedge inv(v') \wedge (\exists \vec{x}'.\, act(v', v) \wedge (r_v \wedge q_v \wedge inv(v))[\vec{x} := \vec{x}'])),$$

then $[\![pre^{\phi}_{\mapsto}(v; r_v)]\!] = pre^{[\![\phi]\!]}_{\mapsto}(v; [\![r_v]\!])$.

Proof. For each location v', the quantified formula in $pre^{\phi}_{\mapsto}(v; r_v)$ specifies that a state $(v', \vec{x})$ satisfies $pre^{\phi}_{\mapsto}(v; r_v)$ iff the following three conditions hold:

1. the data state $\vec{x}$ is in $[\![q_{v'} \wedge inv(v')]\!]$ (the conjuncts $q_{v'} \wedge inv(v')$);
2. there is a state $(v, \vec{x}')$ that is reachable from state $(v', \vec{x})$ by taking a single $\mapsto$ step through transition (v', v) (the conjunct $act(v', v)$); and
3. $\vec{x}'$ is in $[\![r_v \wedge q_v \wedge inv(v)]\!]$ (the conjuncts $(r_v \wedge q_v \wedge inv(v))[\vec{x} := \vec{x}']$).

According to the definition of the transition-step relation $\overset{Q}{\mapsto}$, the proposition holds. ■

Example. Let us consider an example of computing the state predicate $pre^{\phi}_{\mapsto}(v; r_v)$ using Proposition 3. Suppose that the linear hybrid automaton A has two data variables, x and y, and only one non-stutter transition, $e = (v', v)$, with the target location v. Suppose that $act(e) = (x \leq$

$3 \wedge x' \geq 5 \wedge y' = x)$, and that $q_v = q_{v'} = inv(v) = inv(v') = true$. Consider the data predicate $r_v = (x \geq 6 \wedge y \leq 2)$. Then, according to Proposition 3,

$$\begin{array}{rcl} pre^{\phi}_{\mapsto}(v; r_v) & = & (v',\ (\exists x'.\, \exists y'.\, x \leq 3 \wedge x' \geq 5 \wedge y' = x \wedge x' \geq 6 \wedge y' \leq 2)) \wedge \\ & & (v,\ (\exists x'.\, \exists y'.\, x' = x \wedge y' = y \wedge x' \geq 6 \wedge y' \leq 2)) \end{array}$$

(the first conjunct corresponds to the transition e, and the second conjunct corresponds to the stutter transition of v). Eliminating the two existential quantifiers inside out, we obtain

$$pre^{\phi}_{\mapsto}(v; r_v) = (v',\ (\exists x'.\, x \leq 3 \wedge x' \geq 5 \wedge x' \geq 6 \wedge x \leq 2)) \wedge (v,\ (\exists x'.\, x' = x \wedge x' \geq 6 \wedge y \leq 2))$$

and, finally, the state predicate

$$pre^{\phi}_{\mapsto}(v; r_v) = (v',\ x \leq 2) \wedge (v,\ x \geq 6 \wedge y \leq 2). \blacksquare$$

The concluding theorem follows from Propositions 1, 2, and 3.

Theorem 2 *Let A be a linear hybrid automaton, and let ϕ and χ be two state predicates of A. Then $[\![pre^{\phi}_{A}(\chi)]\!] = pre^{[\![\phi]\!]}_{A}([\![\chi]\!])$.*

5 Symbolic Model Checking

Given a nonzeno linear hybrid automaton A, and a closed A-formula ψ of ICTL, the *model-checking problem* asks to compute the characteristic A-region $[\![\psi]\!]_A$, by providing a state predicate ϕ that defines the answer $[\![\psi]\!]_A$; that is, $[\![\phi]\!] = [\![\psi]\!]_A$. The state predicate ϕ is called a *characteristic predicate* of (A, ψ). In general, a characteristic predicate may not exist, and it is undecidable if a given state predicate is a characteristic predicate of (A, ψ) [ACHH93, KPSY93]. In [HNSY94], a symbolic model-checking algorithm, SMC, is presented for computing a characteristic predicate of (A, ψ) in the case of a timed automaton A and a TCTL-formula ψ. We extend the SMC-algorithm and obtain a semi-decision procedure that, provided it terminates, returns a characteristic predicate of (A, ψ) in the general case.

5.1 The SMC-procedure

The SMC-procedure approximates a characteristic predicate of (A, ψ) by a sequence of state predicates. If the successive-approximation sequence converges in a finite number of steps, then the SMC-procedure terminates and returns a characteristic predicate of (A, ψ). The successive-approximation sequence, however, may diverge, in which case the SMC-procedure does not terminate.

Let $\vec{z}$ be the vector of integrators that occur in the formula ψ, together with a new clock $z_{\varphi_1 \forall \mathcal{U} \varphi_2}$ for each subformula $\varphi_1 \forall \mathcal{U} \varphi_2$ of ψ (i.e., z is different from the data variables of A and the integrators of ψ). The SMC-procedure uses the hybrid automaton $A_{\vec{z}}$, which extends A with the integrators from $\vec{z}$ (see Section 3), and the precondition operator $pre_{A_{\vec{z}}}$ on state predicates, which was introduced and computed in Section 4.

Procedure SMC:

Input: a nonzeno linear hybrid automaton A;
a closed A-formula ψ of ICTL.
Output: a characteristic predicate $|\psi|$ of (A, ψ).

Recall that $\phi_A = \bigcup_{v \in V}(v, inv(v))$. The characteristic predicate $|\psi|$ is computed inductively on the subformulas of ψ:

$|\phi| \ := \ \phi \wedge \phi_A$;

$|\neg\varphi| \ := \ \neg|\varphi|$;

$|\varphi_1 \vee \varphi_2| \ := \ |\varphi_1| \vee |\varphi_2|$;

$|\varphi_1 \exists\mathcal{U} \varphi_2| \ := \ \bigvee_{i\geq 0} \chi_i$, where
$\chi_0 \ := \ |\varphi_2|$ and
$\chi_{i+1} \ := \ \chi_i \vee pre_{A_{\vec{z}}}^{|\varphi_1 \vee \varphi_2|}(\chi_i)$
(i.e., compute the sequence $\chi_0, \chi_1, \chi_2, \ldots$ of state predicates until the state predicate $\chi_i \equiv \chi_{i+1}$ is valid[1]);

$|\varphi_1 \forall\mathcal{U} \varphi_2| \ := \ \bigvee_{i\geq 0} \chi_i$, where
$\chi_0 \ := \ |\varphi_2|$ and
$\chi_{i+1} \ := \ |\chi_i \vee \neg z_{\varphi_1 \forall\mathcal{U}\varphi_2}. (\neg\chi_i \exists\mathcal{U}(\neg(\varphi_1 \vee \chi_i) \vee z_{\varphi_1 \forall\mathcal{U}\varphi_2} > 1^2))|$;

$|(z\colon U).\,\varphi| \ := \ |\varphi|[z := 0]$ (i.e., replace all occurrences of z in $|\varphi|$ by 0). ■

The SMC-procedure operates on the $\vec{z}$-extended automaton $A_{\vec{z}}$, and all intermediate results are $\vec{z}$-extended state predicates. First, observe that if the given automaton A is nonzeno, then so is the $\vec{z}$-extension $A_{\vec{z}}$, and that $\phi_{A_{\vec{z}}} = \phi_A$. Second, observe that a characteristic predicate of $(A_{\vec{z}}, \psi)$ can always be simplified to a characteristic predicate of (A, ψ), because $|\psi|$ does not constrain the integrators from $\vec{z}$.

5.2 Possibility

Let ϕ_1 and ϕ_2 be two $\vec{z}$-extended state predicates, and consider the region $R = \Sigma_{A_{\vec{z}}} \cap [\![\phi_1 \vee \phi_2]\!]$ of $A_{\vec{z}}$. The SMC-procedure computes the characteristic region $[\![\phi_1 \exists\mathcal{U} \phi_2]\!]_{A_{\vec{z}}}$ as the least solution X of the equation $f(X) = X$, where

$$f(X) \ = \ [\![\phi_2]\!] \cup pre_{A_{\vec{z}}}^{[\![\phi_1 \vee \phi_2]\!]}(X)$$

is a monotonic function on the subregions of R. This is justified by the following proposition.

Proposition 4 *Let B be a linear hybrid automaton, and let ϕ_1 and ϕ_2 be two state predicates of B. Then $[\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B$ is the least solution of the equation $X \ = \ [\![\phi_2]\!] \cup pre_B^{[\![\phi_1 \vee \phi_2]\!]}(X)$.*

Proof. We define the function $f\colon \Sigma_B \to \Sigma_B$ such that $f(X) \ = \ [\![\phi_2]\!] \cup pre_B^{[\![\phi_1 \vee \phi_2]\!]}(X)$. We first show that $[\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B$ is a fixpoint of f; that is, $[\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B \ = \ f([\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B)$. Since the precondition operator $pre_B^{[\![\phi_1 \exists\mathcal{U} \phi_2]\!]}$ is monotonic on the subregions of Σ_B, $[\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B \subseteq pre_B^{[\![\phi_1 \vee \phi_2]\!]}([\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B)$. Thus $[\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B \subseteq f([\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B)$.

On the other hand, let σ be a state in $pre_B^{[\![\phi_1 \vee \phi_2]\!]}([\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B)$. By definition, there is a state σ' in $[\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B$ such that either (1) $\sigma \stackrel{[\![\phi_1 \vee \phi_2]\!]}{\rightarrow} \sigma'$ or (2) $\sigma \stackrel{[\![\phi_1 \vee \phi_2]\!]}{\mapsto} \sigma'$. In either case, by the definition of trajectories of a linear hybrid automaton and the definition of $\phi_1 \exists\mathcal{U} \phi_2$, state σ is in $[\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B$. Thus $pre_B^{[\![\phi_1 \vee \phi_2]\!]}([\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B) \subseteq [\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B$. Moreover, by the definition of $\phi_1 \exists\mathcal{U} \phi_2$, $[\![\phi_2]\!] \subseteq [\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B$. Therefore, $f([\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B) \subseteq [\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B$.

Now we show that $[\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B$ is the least fixpoint of f. Let Q be a fixpoint of f; we claim that $[\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B \subseteq Q$. Let σ be a state in $[\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B$. Then there is a trajectory $\tau \in [\![B]\!]^{div}$ and

[1] As the state predicates are quantifier-free formulas of the theory $(\mathbb{R}, \leq, +)$, validity can be decided.
[2] Or any positive integer. This choice may effect the number of iterations.

a position (i_1, δ) of τ such that $\tau(0,0) = \sigma$, $\tau(i_1, \delta) \in [\![\phi_2]\!]$, and for all positions $(j, \epsilon) < (i_1, \delta)$, $\tau(j, \epsilon) \in [\![\phi_1 \vee \phi_2]\!]$. Since $[\![\phi_2]\!] \subseteq Q$, $\tau(i_1, \delta) \in Q$. By the definition of the precondition operator $pre_B^{[\![\phi_1 \vee \phi_2]\!]}$, we know that $\tau(i_1 - 1, 0) \in f(\tau(i_1, \delta))$. Since Q is a fixpoint of f and $\tau(i_1 - 1, 0) \in f(Q)$, we conclude that $\tau(i_1 - 1, 0) \in Q$. Similarly, $\tau(i_1 - k, 0) \in Q$ for all $1 \leq k \leq i_1$. In particular, $\sigma = \tau(0,0) \in Q$, which implies that $[\![\phi_1 \exists\mathcal{U} \phi_2]\!]_B \subseteq Q$. ■

The SMC-procedure computes the least solution of the equation $f(X) = X$ as the limit of the successive-approximation sequence $\emptyset$, $f(\emptyset)$, $f^2(\emptyset)$, $f^3(\emptyset)$, ... of regions, which, unlike in the case of timed automata, may not converge in any finite number of steps. The SMC-procedure, therefore, may not terminate.

5.3 Inevitability

The computation of the characteristic region for $\forall\mathcal{U}$-formulas is more involved, because the time-step and transition-step relations are reflexive [HNSY94]. We proceed in two steps. First we reduce inevitability $\forall\mathcal{U}$ (for nonzeno automata) to an iteration of time-bounded inevitability. Second, we reduce time-bounded inevitability (over divergent trajectories) to possibility $\exists\mathcal{U}$, which we know how to compute.

Reduction to time-bounded inevitability

Let B be a linear hybrid automaton, and let c be a nonnegative integer. We define the *time-bounded inevitability operator* $\forall\mathcal{U}_{\leq c}$ on regions such that for all state predicates ϕ_1 and ϕ_2, $[\![\phi_1]\!] \forall\mathcal{U}_{\leq c} [\![\phi_2]\!] = [\![\phi_1 \forall\mathcal{U}_{\leq c} \phi_2]\!]_B$. Given two regions Q and R of B, the region $Q \forall\mathcal{U}_{\leq c} R$ contains all states σ such that on all divergent trajectories of B that start in σ, the region Q is not left until, within c time units, a state in R is reached; that is, $\sigma \in Q \forall\mathcal{U}_{\leq c} R$ if for all divergent trajectories τ of B with $\tau(0,0) = \sigma$, there is a position π of τ such that $\tau(\pi) \in R$ and $t_\tau(\pi) \leq c$, and for all positions π' of τ, if $\pi' \leq \pi$ then $\tau(\pi') \in Q \cup R$. Notice that for nonzeno B, the $\forall\mathcal{U}_{\leq c}$-operator is monotonic in its second argument on the subregions of Σ_A; that is, for all regions $Q, R \subseteq \Sigma_A$, we have $R \subseteq Q \forall\mathcal{U}_{\leq c} R \subseteq \Sigma_A$.

Proposition 5 *Let B be a nonzeno linear hybrid automaton, let ϕ_1 and ϕ_2 be two state predicates of B, and and let $c \in \mathbb{N}_{>0}$ be a positive integer constant. Then $[\![\phi_1 \forall\mathcal{U} \phi_2]\!]_B$ is the least solution of the equation $X = [\![\phi_2]\!] \cup ([\![\phi_1 \vee \phi_2]\!] \forall\mathcal{U}_{\leq c} X)$.*

Proof. We define the function $f : \Sigma_B \to \Sigma_B$ such that $f(X) = [\![\phi_2]\!] \cup ([\![\phi_1 \vee \phi_2]\!] \forall\mathcal{U}_{\leq c} X)$. We first show that $[\![\phi_1 \forall\mathcal{U} \phi_2]\!]_B$ is a fixpoint of f; that is, $[\![\phi_1 \forall\mathcal{U} \phi_2]\!]_B = f([\![\phi_1 \forall\mathcal{U} \phi_2]\!]_B)$. Since the precondition operator $pre_B^{[\![\phi_1 \forall\mathcal{U} \phi_2]\!]}$ is monotonic on the subregions of Σ_B, $[\![\phi_1 \forall\mathcal{U} \phi_2]\!]_B \subseteq [\![\phi_1 \vee \phi_2]\!] \forall\mathcal{U}_{\leq c} [\![\phi_1 \forall\mathcal{U} \phi_2]\!]_B$ Thus $[\![\phi_1 \forall\mathcal{U} \phi_2]\!]_B \subseteq f([\![\phi_1 \forall\mathcal{U} \phi_2]\!]_B)$.

On the other hand, let σ be a state in $[\![\phi_1 \vee \phi_2]\!] \forall\mathcal{U}_{\leq c} [\![\phi_1 \forall\mathcal{U} \phi_2]\!]_B$. By the definitions of f and the $\forall\mathcal{U}_{\leq c}$-operator, for all divergent trajectories τ of B with $\tau(0,0) = \sigma$, there is a position π of τ such that $\tau(\pi) \in [\![\phi_1 \forall\mathcal{U} \phi_2]\!]_B$ and $t_\tau(\pi) \leq c$, and for all positions π' of τ, if $\pi' \leq \pi$ then $\tau(\pi') \in [\![\phi_1 \vee \phi_2]\!] \cup [\![\phi_1 \forall\mathcal{U} \phi_2]\!]_B$. By the definition of $\phi_1 \forall\mathcal{U} \phi_2$, we know that $\sigma \in [\![\phi_1 \forall\mathcal{U} \phi_2]\!]_B$, which implies that $[\![\phi_1 \vee \phi_2]\!] \forall\mathcal{U}_{\leq c} [\![\phi_1 \forall\mathcal{U} \phi_2]\!]_B \subseteq [\![\phi_1 \forall\mathcal{U} \phi_2]\!]_B$. Furthermore, by the definition of $\phi_1 \forall\mathcal{U} \phi_2$, $[\![\phi_2]\!] \subseteq [\![\phi_1 \forall\mathcal{U} \phi_2]\!]_B$. Therefore, $f([\![\phi_1 \forall\mathcal{U} \phi_2]\!]_B) \subseteq [\![\phi_1 \forall\mathcal{U} \phi_2]\!]_B$.

Now we show that $[\![\phi_1 \forall\mathcal{U} \phi_2]\!]_B$ is the least fixpoint of f. Let Q be a fixpoint of f; we claim that $[\![\phi_1 \forall\mathcal{U} \phi_2]\!]_B \subseteq Q$. Assume that there is a state $\sigma_0 \in [\![\phi_1 \forall\mathcal{U} \phi_2]\!]_B \backslash Q$; we will show a contradiction.

We construct inductively an infinite sequence of states $\sigma_i \in [\![\phi_1 \forall\mathcal{U} \phi_2]\!]_B \backslash Q$, for $i \in \mathbb{N}$. Suppose that we have already constructed σ_i. Since $\sigma_i \notin Q$ and Q is a fixpoint of f, we know that $\sigma_i \notin [\![\phi_1 \vee \phi_2]\!] \forall\mathcal{U}_{\leq c} Q$. Since B is nonzeno, there is a trajectory $\tau_i \in [\![B]\!]^{div}$ with $\tau(0,0) = \sigma_i$, and a position π_i of τ_i with $t_{\tau_i}(\pi_i) = c$, such that either

1. for some position $\pi \leq \pi_i$ of τ_i, $\tau_i(\pi) \notin [\![\phi_1 \vee \phi_2]\!]$ and for all positions $\pi' \leq \pi$ of τ_i, $\tau_i(\pi') \notin Q$; or

2. for all positions $\pi \leq \pi_i$ of τ_i, $\tau_i(\pi) \in [\![\phi_1 \vee \phi_2]\!] \backslash Q$.

But since $\sigma_i \in [\![\phi_1 \forall\mathcal{U} \phi_2]\!]_B$ and $[\![\phi_2]\!] \subseteq Q$, Condition 1 cannot happen. Condition 2 together with the assumption $\sigma_i \in [\![\phi_1 \forall\mathcal{U} \phi_2]\!]_B \backslash Q$ implies that

3. for all positions $\pi \leq \pi_i$ of τ_i, $\tau_i(\pi) \in [\![\phi_1 \forall\mathcal{U} \phi_2]\!] \backslash Q$.

So we can pick σ_{i+1} to be $\tau_i(\pi_i)$.

Now consider the trajectory τ that results from concatenating the trajectories $\tau_i[(0,0),\pi_i]$ for all $i \geq 0$:

$$\tau \;=\; \tau_0[(0,0),\pi_0] \to \tau_1[(0,0),\pi_1] \to \tau_2[(0,0),\pi_2] \to \cdots.$$

The trajectory τ diverges, because $c > 0$. Since $[\![B]\!]^{div}$ is fusion-closed and divergence-safe, $\tau \in [\![B]\!]^{div}$. Furthermore, $\tau(0,0) = \sigma_0$ and, by Condition 3, $\tau(\pi) \notin Q$ for all positions π of τ. Because $[\![\phi_2]\!] \subseteq Q$, we conclude that $\sigma_0 \notin [\![\phi_1 \forall\mathcal{U} \phi_2]\!]_B$, which is a contradiction. ■

Reduction to possibility

For linear arguments, the formula $\phi_1 \forall\mathcal{U}_{\leq c}\, \phi_2$ is definable by the $\exists\mathcal{U}$-operator. Roughly speaking, the condition ϕ_2 must inevitably become true within c time units iff it is not possible that ϕ_2 remains false for c time units.

Proposition 6 *Let B be a linear hybrid automaton, let ϕ_1 and ϕ_2 be two state predicates of B, let $c \in \mathbb{N}$ be a nonnegative integer constant, and let z be a new clock (i.e., z is different from the data variables of B). Then*

$$[\![\phi_1]\!] \forall\mathcal{U}_{\leq c} [\![\phi_2]\!] \;=\; [\![\neg z.\, ((\neg\phi_2)\exists\mathcal{U}(\neg(\phi_1 \vee \phi_2) \;\vee\; z > c))]\!]_B.$$

Proof. By the definition of the $\forall\mathcal{U}_{\leq c}$-operator and the semantics of the temporal operator $\forall\mathcal{U}$, and the reset quantifier z, it is clear that

$$[\![\phi_1]\!] \forall\mathcal{U}_{\leq c} [\![\phi_2]\!] = [\![z.\, ((\phi_1 \;\wedge\; z \leq c)\forall\mathcal{U}(\phi_2 \;\wedge\; z \leq c))]\!]_B,$$

which over all trajectories of $[\![B]\!]^{div}$ is equivalent to

$$[\![z.\, ((\phi_1 \;\wedge\; z \leq c)\forall\mathcal{W}(\phi_2 \;\wedge\; z \leq c))]\!]_B,$$

where the temporal operator $\forall\mathcal{W}$ is defined as follows: $\sigma \models_{[\![B]\!]^{div}} \varphi_1 \forall\mathcal{W} \varphi_2$ iff for all trajectories τ of $[\![B]\!]^{div}$ with $\tau(0,0) = \sigma$, either

1. there exists a position π of τ such that $\tau(\pi) \models_{[\![B]\!]^{div}} \varphi_2$ and for all positions $\pi' \leq \pi$, $\tau(\pi') \models_{[\![B]\!]^{div}} \varphi_1 \vee \varphi_2$; or

2. for all positions π of τ, $\tau(\pi) \models_{[\![B]\!]^{div}} \varphi_1$.

Second, we claim that for every hybrid automaton B, and all state predicates ϕ_1 and ϕ_2,

$$[\![\phi_1 \forall\mathcal{W} \phi_2]\!]_B \;=\; [\![\neg((\neg\phi_2)\exists\mathcal{U}(\neg\phi_1 \;\wedge\; \neg\phi_2))]\!]_B.$$

The above equality implies that

$$\llbracket z.\,((\phi_1 \,\wedge\, z \leq c)\forall\mathcal{W}(\phi_2 \,\wedge\, z \leq c))\rrbracket_B = \llbracket \neg z.\,((\neg\phi_2 \,\vee\, z > c)\exists\mathcal{U}(\neg(\phi_1 \,\vee\, \phi_2) \,\vee\, z > c))\rrbracket_B,$$

which would prove the proposition, since the right-hand side of the equality is equal to

$$\llbracket \neg z.\,((\neg\phi_2)\exists\mathcal{U}(\neg(\phi_1 \,\vee\, \phi_2) \,\vee\, z > c))\rrbracket_B.$$

Consider a state $\sigma \in \llbracket(\neg\phi_2)\exists\mathcal{U}(\neg\phi_1 \,\wedge\, \neg\phi_2)\rrbracket_B$. Then there exists a trajectory $\tau \in \llbracket B\rrbracket^{div}$ with $\tau(0,0) = \sigma$ and there is a position π_0 of τ such that $\tau(\pi_0) \notin (\llbracket\phi_1 \vee \phi_2\rrbracket)$ and for all positions $\pi < \pi_0$, $\tau(\pi) \notin \llbracket\phi_2\rrbracket$. By the definition of the operator $\forall\mathcal{W}$, it is clear that $\sigma \notin \llbracket\phi_1\forall\mathcal{W}\phi_2\rrbracket_B$. It remains to be shown that if $\sigma \notin \llbracket\phi_1\forall\mathcal{W}\phi_2\rrbracket_B$, then $\sigma \in \llbracket(\neg\phi_2)\exists\mathcal{U}(\neg\phi_1 \wedge \neg\phi_2)\rrbracket_B$. Assume that $\sigma \notin \llbracket\phi_1\forall\mathcal{W}\phi_2\rrbracket_B$. Then we know that $\sigma \notin \llbracket\phi_2\rrbracket$, and there is a trajectory $\tau \in \llbracket B\rrbracket^{div}$ with $\tau(0,0) = \sigma$ such that either

1. for all positions π of τ, $\tau(\pi) \notin \llbracket\phi_2\rrbracket$, and there is a position π of τ such that $\tau(\pi) \notin \llbracket\phi_1\rrbracket$; or

2. there are some positions π of τ with $\tau(\pi) \in \llbracket\phi_2\rrbracket$, and for each such position π, there is a position $\pi' < \pi$ such that $\tau(\pi') \notin \llbracket\phi_1 \,\vee\, \phi_2\rrbracket$.

Condition 1 implies that $\sigma \in \llbracket(\neg\phi_2)\exists\mathcal{U}(\neg\phi_1 \wedge \neg\phi_2)\rrbracket_B$. Now we assume that Condition 2 holds. Let π_0 be the infimum of all positions π with $\tau(\pi) \in \llbracket\phi_2\rrbracket$. Then we know that for all positions $\pi < \pi_0$ of τ, $\tau(\pi) \notin \llbracket\phi_2\rrbracket$, and either one of the following two cases must hold: (a) $\tau(\pi_0) \in \llbracket\phi_2\rrbracket$ or, (b) $\tau(\pi_0) \notin \llbracket\phi_2\rrbracket$ and there is a position π' that is arbitrarily close to position π_0 such that $\pi' > \pi_0$ and $\tau(\pi') \in \llbracket\phi_2\rrbracket$.

In case (a), Condition 2 implies that there is a position $\pi' < \pi_0$ such that $\tau(\pi') \notin \llbracket\phi_1 \,\vee\, \phi_2\rrbracket$. Moreover, we know that for all positions $\pi < \pi_0$ of τ, $\tau(\pi) \notin \llbracket\phi_2\rrbracket$, so $\sigma \in \llbracket(\neg\phi_2)\exists\mathcal{U}(\neg\phi_1 \,\wedge\, \neg\phi_2)\rrbracket_B$.

In case (b), assume that position $\pi_0 = (i, \delta_0)$ is the last position of trajectory τ in location v. Then no matter whether state $\tau(i+1, 0)$ is in $\llbracket\phi_2\rrbracket$ or not, position $(i+1, 0)$ should have been the infimum of all positions π with $\tau(\pi) \in \llbracket\phi_2\rrbracket$, which is a contradiction. So there must be some position $\pi_2 = (i, \delta_2)$ in the same location v such that $\delta_2 > \delta_0$ and $\tau(\pi_2) \in \llbracket\phi_2\rrbracket$. If there is such a position π_2 such that for all position π_1 with $\pi_0 < \pi_1 < \pi_2$, $\pi_1 \in \llbracket\phi_1 \,\vee\, \phi_2\rrbracket$, then according to Condition 2, there must be a position $\pi' < \pi_0$ such that $\tau(\pi') \notin \llbracket\phi_1 \,\vee\, \phi_2\rrbracket$. Thus we can conclude $\sigma \in \llbracket(\neg\phi_2)\exists\mathcal{U}(\neg\phi_1 \,\wedge\, \neg\phi_2)\rrbracket_B$.

Now it remains to consider the cases that satisfy the following condition (Condition 3): for each position π_2 such that $\pi_2 > \pi_0$ and $\tau(\pi_2) \in \llbracket\phi_2\rrbracket$, there exists a position π_1 such that $\pi_2 > \pi_1 > \pi_0$, and $\tau(\pi_1) \in \llbracket\neg\phi_1 \,\wedge\, \neg\phi_2\rrbracket$. Suppose that $\phi_2 = \bigcup_v(v, p_v)$, and $\neg\phi_1 \,\wedge\, \neg\phi_2 = \bigcup_v(v, q_v)$. Let (v, δ, ρ) be the trajectory fragment of the trajectory τ containing the position π_0 in location v.

Since $\pi_0 = (v, \delta_0)$ is the infimum of all positions π with $\tau(\pi) \in \llbracket\phi_2\rrbracket$, for each $\epsilon > 0$, there is a positive real number ϵ' such that $0 < \epsilon' < \epsilon$ and $\rho(\delta_0 + \epsilon') \in \llbracket p_v\rrbracket$. Moreover, by Condition 3, we know that there is a positive real number ϵ'', such that $0 < \epsilon'' < \epsilon'$ and $\rho(\delta_0 + \epsilon'') \in \llbracket q_v\rrbracket$. In other words, for any $\epsilon > 0$, the open ball $B(\rho(\delta_0), \epsilon)$ intersects $\llbracket q_v\rrbracket$. Therefore $\rho(\delta_0)$ is an *accumulation point* of $\llbracket q_v\rrbracket$. By point-set topology, $\rho(\delta_0)$ is in $\llbracket close(q_v)\rrbracket$.

In addition, since $\rho(\delta_0)$ is an accumulation point of $\llbracket q_v\rrbracket$, again, by point-set topology, for any $\epsilon > 0$, the open ball $B(\rho(\delta_0), \epsilon)$ contains infinitely many points (data states) in $\llbracket q_v\rrbracket$. Note that $\llbracket close(q_v)\rrbracket$ is defined by the data predicate $close(q_v)$, which cannot define infinitely many isolated data states (this would require infinitely many disjuncts). Therefore, there must be a convex data region $S \subseteq \llbracket close(q_v)\rrbracket$ such that for some $\hat{\epsilon} > 0$, $\rho(\delta_0)$ and $\rho(\delta_0 + \hat{\epsilon}) \in \llbracket q_v\rrbracket$ are both in S. By the proof of Lemma 1, we know that there is also a linear data trajectory $(\rho', \hat{\epsilon})$ such that $\rho'(0) = \rho(\delta_0)$, $\rho'(\hat{\epsilon}) = \rho(\delta_0 + \hat{\epsilon})$, and $\rho'(t) \in \llbracket close(q_v)\rrbracket$ for all $t \in [0, \hat{\epsilon}]$.

Since B is nonzeno and $[\![B]\!]$ is fusion-closed, the trajectory fragment $\tau[(0,0),\pi_0] \to (v,\hat{\epsilon},\rho')$ can be extended to a divergent B-trajectory that is a witness trajectory for $\sigma \in [\![(\neg\phi_2)\exists\mathcal{U}(\neg\phi_1 \wedge \neg\phi_2)]\!]_B$. ■

The partial correctness of the SMC-procedure follows from Propositions 4, 5, and 6 (let $B = A_{\vec{z}}$).

Theorem 3 *If the procedure* SMC *halts on the input* (A,ψ)*, then* $|\psi|$ *is a characteristic predicate of* (A,ψ)*; that is,* $[\![|\psi|]\!] = [\![\psi]\!]_A$.

6 The Cornell Hybrid Technology Tool

The SMC-procedure has been implemented in HYTECH, which runs on Sun Sparc stations under MATHEMATICA. Throughout the verification process, state predicates are represented as quantifier-free formulas of the theory $(\mathbb{R},\leq,+)$ of the reals with addition, over data variables, integrators, and the program counter ℓ. HYTECH performs the following operations on state predicates:

Boolean operations Trivial.

Quantifier elimination This is necessary to compute the time-precondition operation $pre_{\to}$ and the transition-precondition operation $pre_{\mapsto}$ on state predicates. See below.

Validity checking This is necessary (1) throughout the SMC-procedure, to see if a successive-approximation sequence of state predicates has converged, and (2) after termination of the SMC-procedure, to see if the input automaton A meets the requirement specified by the input formula ψ. Let $|\psi|$ be a characteristic predicate of (A,ψ). Then A meets ψ iff the state predicate $|\psi| \equiv \phi_A$ is valid.

The state predicate ϕ is valid iff $\neg\phi$ is unsatisfiable. HYTECH determines the satisfiability of state predicates using linear programming. First each linear formula is converted into disjunctive normal form. Then for each disjunct, which is a conjunction, the linear-programming algorithm of MATHEMATICA decides if that conjunction of linear inequalities has a solution. This is an exponential decision procedure for deciding the satisfiability of linear formulas, whose satisfiability problem is NP-complete [HNSY94].

Simplification For quantifier elimination and validity checking, HYTECH converts state predicates into disjunctive normal form, which may cause an exponential blow-up of the size of the formulas. To alleviate this problem, we simplify all formulas after each step of the verification process by applying rewrite rules. We use a polynomial-time set of rewrite rules, whose iterated application to a linear formula in disjunctive normal form brings each disjunct into normal form, but may not eliminate redundant or overlapping disjuncts. It is worth noting that the repeated simplification of state predicates is, overall, by far the most time-consuming activity of HYTECH.

6.1 Quantifier elimination

The theory $(\mathbb{R},\leq,+)$ admits quantifier elimination, and we first implemented the theoretically optimal decision procedure of Ferrante and Rackoff [FR75]. We found, however, that the following "naive" quantifier-elimination procedure performs better for our purposes, perhaps because we need to deal only with quantified formulas in a particular form.

We eliminate quantifiers one by one, starting with innermost quantifiers. Consider the formula $\exists\delta.\, p$, for a linear (quantifier-free) formula p over the set $\vec{x}$ of data variables and the quantified variable δ. We first convert $\exists\delta.\, p$ into an equivalent formula of the form $\bigvee \exists\delta.\, p_i$, where each p_i is a convex data predicate over $\vec{x} \uplus \{\delta\}$. This is always possible, because the existential quantifier distributes over disjunction. We then convert each linear inequality in p_i into the form $0 \sim e$, where e is a linear term, and $\sim \in \{<, \leq\}$. So suppose that p_i is of the form

$$0 \sim_1 e_1 + c_1 \cdot \delta \;\wedge\; \ldots \;\wedge\; 0 \sim_k e_k + c_k \cdot \delta \;\wedge\; 0 \sim_{k+1} e_{k+1} + c_{k+1} \cdot \delta \;\wedge\; \ldots \;\wedge\; 0 \sim_m e_m + c_m \cdot \delta,$$

where $c_1, \ldots, c_k > 0$ and $c_{k+1}, \ldots, c_m < 0$. We now "solve" each conjunct for δ; that is, we convert each conjunct into the form $\frac{e_j}{c_j} \sim_j \delta$, for $1 \leq j \leq k$; and $\delta \sim_{j'} \frac{e_{j'}}{c_{j'}}$, for $k+1 \leq j' \leq m$. Then we eliminate δ by combining conjuncts. For example, the conjuncts $x+3 < \delta$ and $\delta \leq 4y$ are combined to the new conjunct $x+3 < 4y$. We so obtain the following formula p'_i:

$$\bigwedge_{\substack{1 \leq j \leq k \\ k+1 \leq j' \leq m}} \left(\frac{e_j}{c_j} \sim_{jj'} \frac{e_{j'}}{c_{j'}}\right),$$

where $\sim_{jj'}$ is $<$ if $\sim_j$ or $\sim_{j'}$ is $<$; and $\sim_{jj'}$ is $\leq$, otherwise. It is not difficult to check that the resulting quantifier-free formula $\bigvee p'_i$ equivalent to the original formula $\exists\delta.\, p$.

6.2 The input language

The HyTech syntax for describing hybrid automata in a textual language is chosen so that the input file can be read directly by Mathematica. As a self-explanatory example, consider the following input file, which specifies the railroad gate controller from Section 2.

```
AutomataNo = 3
Variables = {x,y,z}

LocatioNo = {3,4,3}
inv[l[1]==1] = 1000<=x              (* far            *)
inv[l[1]==2] = 0<=x && x<=1000      (* near           *)
inv[l[1]==3] = 0<=x && x<=100       (* past           *)
inv[l[2]==1] = 0<=y && y<=90        (* up             *)
inv[l[2]==2] = 90==y                (* open           *)
inv[l[2]==3] = 0<=y && y<=90        (* down           *)
inv[l[2]==4] = 0==y                 (* closed         *)
inv[l[3]==1] = 0<=z && z<=5         (* about to lower *)
inv[l[3]==2] = True                 (* idle           *)
inv[l[3]==3] = 0<=z && z<=5         (* about to raise *)

dif[1,1,x] = {-52,-48}              (* far            *)
dif[1,2,x] = {-52,-40}              (* near           *)
dif[1,3,x] = {40,52}                (* past           *)
dif[2,1,y] = {20,20}                (* up             *)
dif[2,2,y] = {0,0}                  (* open           *)
dif[2,3,y] = {-20,-20}              (* down           *)
dif[2,4,y] = {0,0}                  (* closed         *)
dif[3,1,z] = {1,1}                  (* about to lower *)
dif[3,2,z] = {0,0}                  (* idle           *)
dif[3,3,z] = {1,1}                  (* about to raise *)

Labels = {app,exit,lower,raise}
syn[app] = {1,3}
syn[exit] = {1,3}
syn[lower] = {2,3}
syn[raise] = {2,3}
```

```
TransitioNo = {3,6,8}
act[1,1] = { l[1]==1 && 1000==x, app, {l[1] -> 2}}
act[1,2] = { l[1]==2 && 0==x, {l[1] -> 3}}
act[1,3] = { l[1]==3 && 100==x, exit, {l[1] -> 1, x -> 1500}}
act[2,1] = { l[2]==1 && 90==y, {l[2] -> 2}}
act[2,2] = { l[2]==1, lower, {l[2] -> 3}}
act[2,3] = { l[2]==2, lower, {l[2] -> 3}}
act[2,4] = { l[2]==3 && 0==y, {l[2] -> 4}}
act[2,5] = { l[2]==3, raise, {l[2] -> 1}}
act[2,6] = { l[2]==4, raise, {l[2] -> 1}}
act[3,1] = { l[3]==1, app, {l[3] -> 1}}
act[3,2] = { l[3]==1, exit, {l[3] -> 1}}
act[3,3] = { l[3]==1, lower, {l[3] -> 2}}
act[3,4] = { l[3]==2, app, {l[3] -> 1, z -> 0}}
act[3,5] = { l[3]==2, exit, {l[3] -> 3, z -> 0}}
act[3,6] = { l[3]==3, exit, {l[3] -> 3}}
act[3,7] = { l[3]==3, raise, {l[3] -> 2}}
act[3,8] = { l[3]==3, app, {l[3] -> 1, z -> 0}}
```

As a complete user's manual for HyTech can be requested from the authors, only a few points are elaborated here:

- `inv[l[1]==3] = 0<=x && x<=100` specifies that the invariant of the third (*past*) location of the first (train) component automaton is $0 \leq x \leq 100$.
- `dif[1,3,x]={40,52}` specifies that the activity of the third location of the first component automaton contains the conjunct $40 \leq \dot{x} \leq 52$. (More general activities are being implemented.)
- `syn[app]={1,3}` specifies that the synchronization label *app* belongs to the alphabets of the first and third component automata (but not to the alphabet of the second).
- `t[1,3] = {l[1]==3 && 100==x, exit, {l[1] -> 1, x -> 1500}}` specifies that the third transition of the first automaton proceeds from the third location to the first (*far*) location, its action is $x = 100 \wedge x' = 1500$, and it is labeled with the synchronization label *exit*. (More general actions are being implemented.)

When computing the product of the input automata, we first enumerate all possible composite locations and transitions, and then remove composite locations with unsatisfiable invariants or activities, and composite transitions with unsatisfiable actions or inconsistent synchronization labels. The transitions are indexed by target location, as to facilitate a quick access when computing preconditions.

Now suppose we wish to check if the railroad gate controller meets the time-bounded response requirement

$$(\ell = (far, open, idle)\ \rightarrow\ \forall\Box\forall\Diamond_{\leq 33}\,(\ell[2] = open)$$

from Section 3. We then write:

```
InitialRegion = l[1]==1 && l[2]==2 && l[3]==2
Integrators = {u}
dif[_,_,u] = {1,1}
Compute[Impl[InitialRegion,AA[AET[u,33,l[3]==2]]]]
```

Here `AA` stands for the invariance operator $\forall\Box$, and `AET` denotes time-bounded inevitability. The first argument of the operator `AET` is an integrator, u, for measuring the time bound. The type declaration `dif[_,_,u]` of u, which augments the description of the input system, asserts that u is a clock; that is, $\dot{u} = 1$ in all locations. (The automatic extension of the input system with integrators is being implemented.)

7 Examples

We now illustrate the application of HyTech with four examples. First, we use HyTech to verify the safety and time-bounded response properties of the railroad gate controller from Section 2. Second, we use HyTech to automatically derive sufficient and necessary timing conditions for a distributed mutual-exclusion protocol with drifting clocks. Third, we use HyTech to check a safety property of a preemptive scheduler with prioritized tasks. Fourth, we use HyTech to automatically derive sufficient timing conditions for the temperature control of a nuclear reactor.

7.1 Railroad gate control

Recall the railroad gate controller from Section 2, and the initial condition ϕ_0 from Section 3. The safety requirement of Section 3 can be rewritten as

$$\phi_0 \;\rightarrow\; \neg\exists\Diamond(x \leq 10 \;\wedge\; \ell[2] \neq \mathit{closed}).$$

To avoid negation, we ask HyTech to first compute the characteristic predicate

$$\phi\colon \quad |\exists\Diamond(x \leq 10 \;\wedge\; \ell[2] \neq \mathit{closed})|$$

and then check that the state predicate $\phi_0 \wedge \phi$ is unsatisfiable. It follows that the railroad gate controller meets the safety requirement. HyTech takes 74.3 seconds of CPU time to verify this task.[3] The characteristic predicate ϕ computed by HyTech is the state predicate

$$(\ell[1] = 2 \wedge \ell[2] = 1 \wedge \ell[3] = 1 \wedge 0 \leq z \wedge -1220 \leq -5x - 13y \wedge -90 \leq -y \wedge 0 \leq x \wedge -5 \leq
-z \wedge -270 \leq -x - 52z) \vee (\ell[1] = 2 \wedge \ell[2] = 1 \wedge \ell[3] = 2 \wedge 90 = y \wedge 0 \leq z \wedge 0 \leq x \wedge -5 \leq
-z \wedge -270 \leq -x - 52z) \vee (\ell[1] = 2 \wedge \ell[2] = 1 \wedge \ell[3] = 3 \wedge 0 \leq z \wedge 0 \leq x \wedge 0 \leq y \wedge -5 \leq -z \wedge -50 \leq
-5x + 13y \wedge -270 \leq -x - 52z) \vee (\ell[1] = 2 \wedge \ell[2] = 2 \wedge \ell[3] = 1 \wedge 0 \leq x \wedge -90 \leq -y \wedge -1220 \leq
-5x - 13y) \vee (\ell[1] = 2 \wedge \ell[2] = 2 \wedge \ell[3] = 2 \wedge 90 = y \wedge 0 \leq x) \vee (\ell[1] = 2 \wedge \ell[2] = 2 \wedge \ell[3] = 3 \wedge 0 \leq
y \wedge 0 \leq x \wedge -50 \leq -5x + 13y) \vee (\ell[1] = 2 \wedge \ell[2] = 3 \wedge \ell[3] = 1 \wedge 0 \leq z \wedge -1220 \leq -5x - 13y \wedge -90 \leq
-y \wedge 0 \leq x \wedge -5 \leq -z \wedge -270 \leq -x - 52z) \vee (\ell[1] = 2 \wedge \ell[2] = 3 \wedge \ell[3] = 2 \wedge 90 = y \wedge 0 \leq z \wedge 0 \leq
x \wedge -5 \leq -z \wedge -270 \leq -x - 52z) \vee (\ell[1] = 2 \wedge \ell[2] = 3 \wedge \ell[3] = 3 \wedge 0 \leq z \wedge 0 \leq x \wedge 0 \leq y \wedge -5 \leq
-z \wedge -50 \leq -5x + 13y \wedge -270 \leq -x - 52z) \vee (\ell[1] = 3 \wedge \ell[2] = 1 \wedge \ell[3] = 1 \wedge -10 \leq -x \wedge -90 \leq
-y \wedge -5 \leq -z \wedge 0 \leq z) \vee (\ell[1] = 3 \wedge \ell[2] = 1 \wedge \ell[3] = 2 \wedge 90 = y \wedge -10 \leq -x \wedge -5 \leq -z \wedge 0 \leq
z) \vee (\ell[1] = 3 \wedge \ell[2] = 1 \wedge \ell[3] = 3 \wedge -10 \leq -x \wedge 0 \leq y \wedge -5 \leq -z \wedge 0 \leq z) \vee (\ell[1] = 3 \wedge \ell[2] =
2 \wedge \ell[3] = 1 \wedge -90 \leq -y \wedge -10 \leq -x) \vee (\ell[1] = 3 \wedge \ell[2] = 2 \wedge \ell[3] = 2 \wedge 90 = y \wedge -10 \leq -x) \vee (\ell[1] =
3 \wedge \ell[2] = 2 \wedge \ell[3] = 3 \wedge 0 \leq y \wedge -10 \leq -x) \vee (\ell[1] = 3 \wedge \ell[2] = 3 \wedge \ell[3] = 1 \wedge -10 \leq -x \wedge -90 \leq
-y \wedge -5 \leq -z \wedge 0 \leq z) \vee (\ell[1] = 3 \wedge \ell[2] = 3 \wedge \ell[3] = 2 \wedge 90 = y \wedge -10 \leq -x \wedge -5 \leq -z \wedge 0 \leq
z) \vee (\ell[1] = 3 \wedge \ell[2] = 3 \wedge \ell[3] = 3 \wedge -10 \leq -x \wedge 0 \leq y \wedge -5 \leq -z \wedge 0 \leq z) \vee (\ell[1] = 2 \wedge \ell[2] =
1 \wedge \ell[3] = 1 \wedge 0 \leq z \wedge -90 \leq -y \wedge -270 \leq -x - 52z \wedge -10 \leq y - 20z \wedge 180 \leq x + 2y) \vee (\ell[1] =
2 \wedge \ell[2] = 1 \wedge \ell[3] = 1 \wedge 0 \leq z \wedge 0 \leq x \wedge 0 \leq x + 2y \wedge -2650 \leq -5x + 13y - 520z \wedge -90 \leq -y \wedge -100 \leq
y - 20z \wedge -5 \leq -z \wedge -2390 \leq -5x - 13y) \vee (\ell[1] = 2 \wedge \ell[2] = 1 \wedge \ell[3] = 2 \wedge 90 = y \wedge 0 \leq z \wedge 0 \leq
x \wedge -5 \leq -z \wedge -504 \leq -x - 52z) \vee (\ell[1] = 2 \wedge \ell[2] = 2 \wedge \ell[3] = 1 \wedge 180 \leq x + 2y \wedge -90 \leq -y) \vee (\ell[1] =
2 \wedge \ell[2] = 3 \wedge \ell[3] = 1 \wedge 0 \leq z \wedge -90 \leq -y \wedge -270 \leq -x - 52z \wedge -10 \leq y - 20z \wedge 180 \leq x + 2y) \vee (\ell[1] =
2 \wedge \ell[2] = 3 \wedge \ell[3] = 3 \wedge 0 \leq z \wedge 0 \leq y \wedge 0 \leq x \wedge -180 \leq x - 2y \wedge -190 \leq -y - 20z \wedge -1220 \leq
-5x + 13y \wedge -3820 \leq -5x - 13y - 520z \wedge -5 \leq -z) \vee (\ell[1] = 2 \wedge \ell[2] = 3 \wedge \ell[3] = 4 \wedge 0 = y \wedge 0 \leq z \wedge 0 \leq
x \wedge -5 \leq -z \wedge -504 \leq -x - 52z) \vee (\ell[1] = 3 \wedge \ell[2] = 3 \wedge \ell[3] = 4 \wedge 0 = y \wedge -10 \leq -x \wedge -5 \leq -z \wedge 0 \leq
z) \vee (\ell[1] = 2 \wedge \ell[2] = 1 \wedge \ell[3] = 1 \wedge 0 \leq z \wedge -90 \leq -y \wedge -504 \leq -x - 52z \wedge -10 \leq y - 20z \wedge 180 \leq$$

[3] All performance data are measured on a Sun SPARC-670MP server running Mathematica.

$x + 2y) \vee (\ell[1] = 2 \wedge \ell[2] = 3 \wedge \ell[3] = 3 \wedge 0 \leq z \wedge 0 \leq y \wedge 180 \leq x + 2y \wedge -5 \leq -z \wedge -190 \leq -y - 20z \wedge -180 \leq x - 2y) \vee (\ell[1] = 2 \wedge \ell[2] = 3 \wedge \ell[3] = 3 \wedge 0 \leq z \wedge 0 \leq x - 2y \wedge 0 \leq y \wedge -100 \leq -y - 20z \wedge -504 \leq -x - 52z) \vee (\ell[1] = 2 \wedge \ell[2] = 3 \wedge \ell[3] = 4 \wedge 0 = y \wedge 180 \leq x \wedge -5 \leq -z \wedge 0 \leq z).$

The time-bounded response requirement of Section 3 is verified by HyTech using 215.1 seconds of CPU time.

7.2 Timing-based mutual exclusion

One of the advantages of a symbolic model-checking procedure is that we can attempt to verify system descriptions with unknown constants (parameters) [AHV93]. A *parameter* of the hybrid automaton A is a data variable x whose value is modified neither by continuous activities nor by discrete actions; that is, for all control locations v of A, $dif(v)$ implies $\dot{x} = 1$, and for all transitions e of A, $act(e)$ implies $x' = x_i$.

For example, HyTech may be used to design the delay parameters of a system. Consider the mutual-exclusion problem for an asynchronous distributed system with local clocks. The system consists of two processes, P_1 and P_2, with atomic read and write operations on a shared memory. Each process has a critical section, and at every time instant at most one of the two processes is allowed to be in its critical section. Mutual exclusion can be ensured by a version of Fischer's protocol [Lam87], which we first describe in pseudocode. Each process P_i, for $i = 1, 2$, executes the following algorithm:

repeat
    repeat
        await $k = 0$
        $k := i$
        delay b
    until $k = i$
    Critical section
    $k := 0$
forever

The two processes P_1 and P_2 share the variable k, and process P_i is allowed into its critical section iff $k = i$. Each process has a private clock. The instruction **delay** b delays a process for at least b time units as measured by its local clock. Furthermore, each process takes at most a time units, as measured by its local clock, for a single write access to the shared memory (i.e., for the assignment $k := i$).

To make matters more interesting, we assume that the two local clocks of the processes P_1 and P_2 are inaccurate and proceed at different rates. Indeed, the clock rates may vary within certain bounds: the local clock of P_1 is slow—its rate varies between 4/5 and 1—and the local clock of P_2 is fast—its rate varies between 1 and 11/10. The resulting system can be modeled by the product of the two hybrid automata shown in Figure 4. Each of the two automata models one process, with the two critical sections being represented by the locations 4 and D, respectively. The parameters a and b are shared data variables with unknown, constant values (i.e., their rate of change is 0 in all locations, and they are not altered by any transitions).

The initial condition ϕ_0 of the system is given by the state predicate

$$\phi_0: \quad \ell = (1, A) \wedge k = 0.$$

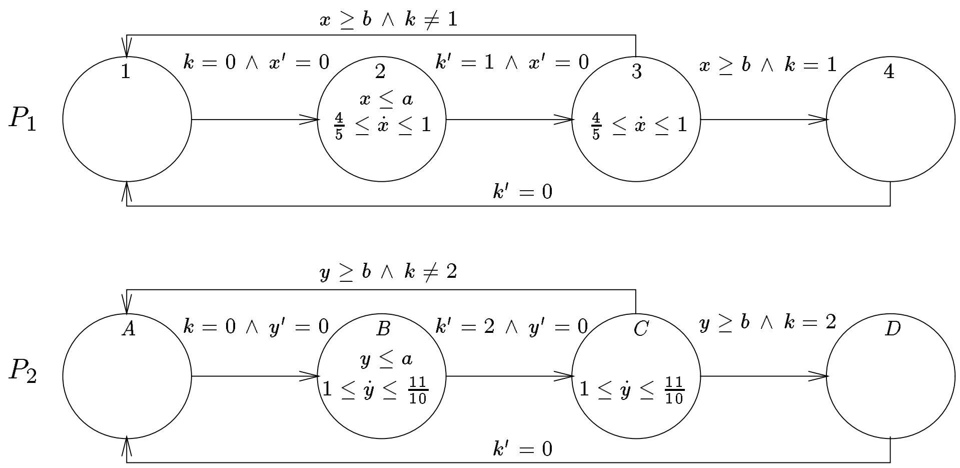

Figure 4: Timing-based mutual-exclusion protocol

The mutual-exclusion requirement is specified in ICTL-formula

$$\phi_0 \;\rightarrow\; \neg\exists\Diamond(\ell = (4, D)).$$

The characteristic predicate

$$\phi: \quad |\phi_0 \wedge \exists\Diamond(\ell = (4, D))|,$$

as computed by HYTECH in 50.1 seconds of CPU time, is the state predicate

$$\ell = (1, A) \wedge k = 0 \wedge (a \geq b \vee 11a \geq 8b).$$

It follows that the system meets the mutual-exclusion requirement precisely in those states in which ϕ is false. Therefore, the protocol guarantees mutual exclusion iff the delay parameters a and b are chosen such that $8b > 11a$.

A further advantage of the symbolic approach to system analysis is its insensitivity to the magnitude of constants in the system description. To demonstrate this, we performed the following experiment. We verified the mutual-exclusion property for hybrid automata that are identical to those shown in Figure 4, except that the parameters a and b are instantiated, first with $a = 2$ and $b = 4$, then with $a = 2 \cdot 10^3$ and $b = 4 \cdot 10^3$, and finally with $a = 2 \cdot 10^6$ and $b = 4 \cdot 10^6$. We found that, for this example, the verification time taken by HYTECH is independent of the magnitude of the parameter values. The results are shown in Figure 7.2.

Magnitude of the coefficients	CPU time (in seconds)
10^0	6.11
10^3	6.55
10^6	6.36

Figure 5: Coefficient size versus performance

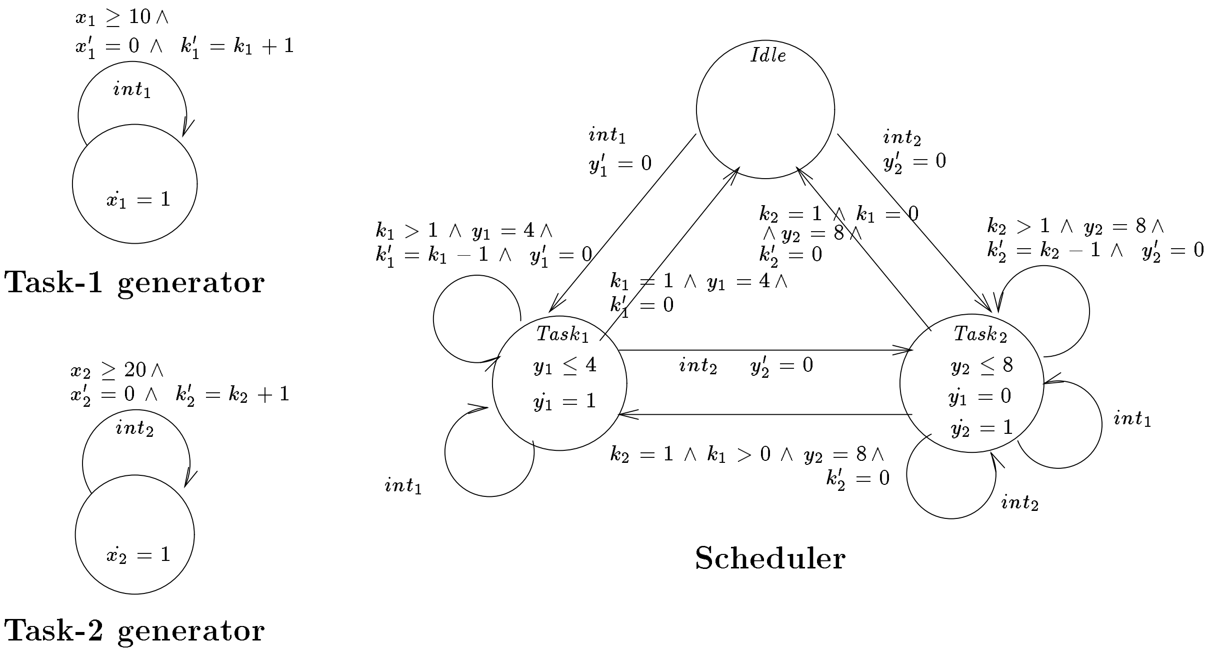

Figure 6: Preemptive two-task scheduler

7.3 Preemptive scheduling

The execution of multiple tasks on a shared processor can be modeled with integrators. Suppose that the integrator y_T runs at slope 1 while the task T is being executed, and y_T runs at slope 0 while the task T is waiting for processor time. Then, the integrator y_T always indicates the cumulative amount of processor time that has been dedicated to task T. This information is needed to model the completion of task T.

In our particular setup, all tasks fall into two priority classes. The two small hybrid automata on the left of Figure 6 model an environment that generates two types of interrupts: an interrupt of type int_1 arrives at most once every 10 seconds; an interrupt of type int_2, at most once every 20 seconds. For every int_i interrupt, a task of type i needs to be executed: each type-1 task requires 4 seconds; each type-2 task, 8 seconds. Only one processor is available for the execution of all tasks, and type-2 tasks have priority over type-1 tasks; that is, if a type-2 interrupt is generated while a type-1 task is being executed, then the type-2 task is interrupted and resumed only after the completion of the newly arrived type-1 task. The resulting priority scheduler is modeled by the hybrid automaton on the right of Figure 6. In location $Task_1$, a type-1 task is being executed; in location $Task_2$, a type-2 task is being executed. The variable k_i represents the number of incomplete (i.e., running and pending) tasks of type i; the variable y_i represents the execution time of the current task of type i. The three automata synchronize using the labels int_1 and int_2; that is, whenever an int_i interrupt arrives, the scheduler takes a transition that is labeled with int_i. The entire scheduling system, then, is the product of the three component automata.

The initial condition ϕ_0 of the system is given by the state predicate

$$\phi_0\colon \quad \ell = (\cdot, \cdot, Idle) \wedge k_1 = 0 \wedge k_2 = 0.$$

We wish to check that the number of incomplete type-1 tasks never exceeds 2 and the number of

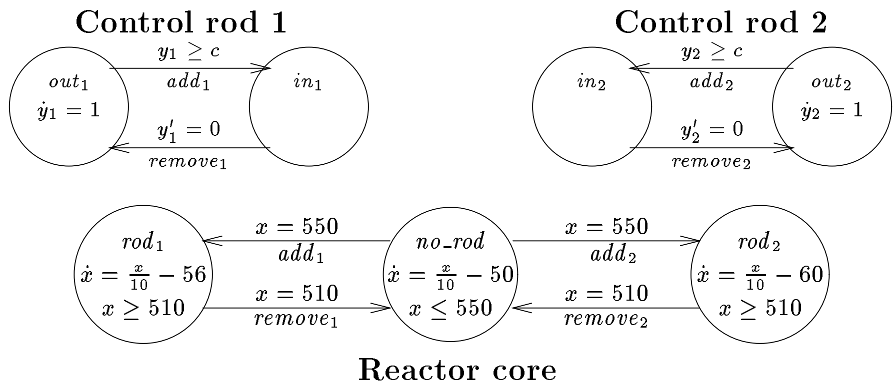

Figure 7: Nonlinear reactor model

incomplete type-2 tasks never exceeds 1; that is,

$$\phi_0 \ \rightarrow \ \neg\exists\Diamond(k_1 > 2 \ \vee \ k_2 > 1).$$

HyTech successfully verifies this safety property using 174.3 seconds of CPU time. Notice that both k_1 and k_2 are data variables that range over the finite domain $\{0, 1, 2, \geq 3\}$ and, therefore, their values can be encoded by splitting control locations. Using such an encoding, the number of tasks types that can be dealt with in less than one hour of CPU time increases from 2 to 4.

7.4 Temperature control

In practical control applications nonlinear variables, such as temperature, abound. The behavior of a nonlinear variable x in location v can be approximated conservatively by a differential inclusion $dif(v)$ of the form $a \leq \dot{x} \leq b$, where the integer constants a and b specify the minimal and maximal rate of change of the variable x in location v. Thus, every trajectory of the approximated nonlinear system is a trajectory of the approximating linear hybrid automaton. It follows that if the approximating linear hybrid automaton meets a safety requirement, then so does the original nonlinear system. Approximations can be refined arbitrarily by splitting control locations and strengthening the corresponding differential inclusions [HH95].

Consider, for example, a toy nuclear reactor with two control rods [NOSY93]. The temperature of the reactor core is represented by the nonlinear variable x. Initially the core temperature is 510 degrees and both control rods are outside the reactor core. In this case, the core temperature rises according to the differential equation $\dot{x} = \frac{x}{10} - 50$. The reactor is shut down once the core temperature increases beyond 550 degrees. In order to prevent a shutdown, one of two control rods can be put into the reactor core. Control rod 1 dampens the core temperature according to the differential equation $\dot{x} = \frac{x}{10} - 56$; control rod 2 has a stronger effect and dampens the core temperature according to the differential equation $\dot{x} = \frac{x}{10} - 60$. Either control rod is removed once the core temperature falls back to 510 degrees. The reactor core is modeled by the hybrid automaton at the bottom of Figure 7.

An additional design requirement asserts that when a control rod is removed from the reactor core, it cannot be put back into the core for c seconds, for a parameter c. This requirement is enforced by the clock y_1, which measures the elapsed time since control rod 1 has been removed from the reactor core, and the clock y_2, which measures the elapsed time since control rod 2 has been

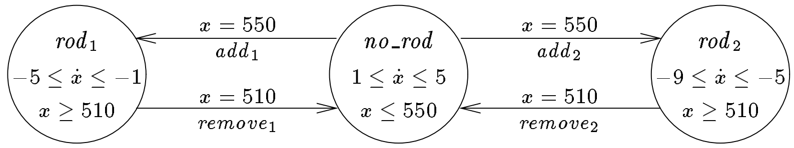

Figure 8: The approximating linear core automaton

removed. The control rods are modeled by the two hybrid automata at the top of Figure 7. The rod automata synchronize with the core automaton through shared edge labels such as $remove_1$, which indicates the removal of control rod 1. The entire reactor system, then, is obtained by constructing product of the core automaton and the two rod automata.

In order to approximate the behavior of the nonlinear variable x by differential inclusions, we compute, for each location of the core automaton, the minimum and the maximum of the derivative $\dot{x}$ from the differential equations and the location invariants. For example, in location *no_rod*, the derivative $\dot{x}$ is bounded below by 1 and is bounded above by 5. The resulting linear core automaton is shown in Figure 8.

The initial condition ϕ_0 of the system is given by the state predicate

$$\phi_0: \quad \ell = (no\_rod, out_1, out_2) \wedge x = 510 \wedge y_1 = c \wedge y_2 = c.$$

The safety requirement that the reactor never needs to be shut down is specified by the ICTL-formula

$$\phi_0 \;\rightarrow\; \neg\exists\Diamond(x = 550 \wedge y_1 < c \wedge y_2 < c).$$

This formula asserts that whenever the core temperature reaches 550 degrees, then one of the two clocks shows at least c seconds, thus allowing the corresponding control rod to be put into the reactor core. The condition on the parameter c thats prevents a shutdown of the reactor is computed by HyTech as $9c < 184$, using 20.5 seconds of CPU time. Notice that while the condition $9c < 184$ is both necessary and sufficient for the approximating linear system, it is only sufficient for the original nonlinear system (more refined approximations will give sharper conditions).

8 Future Work

Our current implementation of HyTech represents state sets as logical formulas (state predicates). We need to search for more efficient representations of state sets to verify complex systems. In the absence of continuous variables, considerable success has been reported in verifying complex systems using state-set representations that are based on binary-decision diagrams [McM93]. In the case of timed automata, sets of clock values can be represented efficiently using integer matrices [Dil89]. For linear hybrid automata, a geometric representation of linear state sets as polyhedra—perhaps combined with abstract-interpretation techniques for convergence acceleration [Hal93]—may be the most promising direction [HH94]. Another line of recent research pursues the approximate analysis of nonlinear hybrid systems within the linear framework of this paper [HH95].

Acknowledgments. We thank Costas Courcoubetis and Joesph Sifakis for helpful discussions.

References

[ACD93] R. Alur, C. Courcoubetis, and D.L. Dill. Model checking in dense real time. *Information and Computation*, 104(1):2–34, 1993.

[ACH+94] R. Alur, C. Courcoubetis, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. In G. Cohen and J.-P. Quadrat, editors, *Proceedings of the 11th International Conference on Analysis and Optimization of Systems: Discrete-event Systems*, Lecture Notes in Control and Information Sciences 199, pages 331–351. Springer-Verlag, 1994.

[ACHH93] R. Alur, C. Courcoubetis, T.A. Henzinger, and P.-H. Ho. Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems*, Lecture Notes in Computer Science 736, pages 209–229. Springer-Verlag, 1993.

[AD94] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[AFH91] R. Alur, T. Feder, and T.A. Henzinger. The benefits of relaxing punctuality. In *Proceedings of the Tenth Annual Symposium on Principles of Distributed Computing*, pages 139–152. ACM Press, 1991.

[AH94] R. Alur and T.A. Henzinger. A really temporal logic. *Journal of the ACM*, 41(1):181–204, 1994.

[AHV93] R. Alur, T.A. Henzinger, and M.Y. Vardi. Parametric real-time reasoning. In *Proceedings of the 25th Annual Symposium on Theory of Computing*, pages 592–601. ACM Press, 1993.

[BES93] A. Bouajjani, R. Echahed, and J. Sifakis. On model checking for real-time properties with durations. In *Proceedings of the Eighth Annual Symposium on Logic in Computer Science*, pages 147–159. IEEE Computer Society Press, 1993.

[CES81] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Workshop on Logic of Programs*, Lecture Notes in Computer Science 131. Springer-Verlag, 1981.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal-logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[Dil89] D.L. Dill. Timing assumptions and verification of finite-state concurrent systems. In J. Sifakis, editor, *CAV 89: Automatic Verification Methods for Finite-state Systems*, Lecture Notes in Computer Science 407, pages 197–212. Springer-Verlag, 1989.

[FR75] J. Ferrante and C. Rackoff. A decision procedure for the first-order theory of real addition with order. *SIAM Journal on Computing*, 4(1):69–76, 1975.

[GNRR93] R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors. *Hybrid Systems*. Lecture Notes in Computer Science 736. Springer-Verlag, 1993.

[Hal93] N. Halbwachs. Delay analysis in synchronous programs. In C. Courcoubetis, editor, *CAV 93: Computer-aided Verification*, Lecture Notes in Computer Science 697, pages 333–346. Springer-Verlag, 1993.

[HH94] T.A. Henzinger and P.-H. Ho. Model-checking strategies for linear hybrid systems. Technical Report CSD-TR-94-1437, Cornell University, 1994. Presented at the Seventh International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, and at the Workshop on Hybrid Systems and Autonomous Control (Ithaca, NY).

[HH95] T.A. Henzinger and P.-H. Ho. Algorithmic analysis of nonlinear hybrid systems. To appear at CAV, 1995.

[HNSY94] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[KPSY93] Y. Kesten, A. Pnueli, J. Sifakis, and S. Yovine. Integration graphs: a class of decidable hybrid systems. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems*, Lecture Notes in Computer Science 736, pages 179–208. Springer-Verlag, 1993.

[Lam87] L. Lamport. A fast mutual exclusion algorithm. *ACM Transactions on Computer Systems*, 5(1):1–11, 1987.

[LP85] O. Lichtenstein and A. Pnueli. Checking that finite-state concurrent programs satisfy their linear specification. In *Proceedings of the 12th Annual Symposium on Principles of Programming Languages*, pages 97–107. ACM Press, 1985.

[McM93] K.L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem.* Kluwer Academic Publishers, 1993.

[MMP92] O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In J.W. de Bakker, K. Huizing, W.-P. de Roever, and G. Rozenberg, editors, *Real Time: Theory in Practice*, Lecture Notes in Computer Science 600, pages 447–484. Springer-Verlag, 1992.

[NOSY93] X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. An approach to the description and analysis of hybrid systems. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems*, Lecture Notes in Computer Science 736, pages 149–178. Springer-Verlag, 1993.

[NSY93] X. Nicollin, J. Sifakis, and S. Yovine. From ATP to timed graphs and hybrid systems. *Acta Informatica*, 30:181–202, 1993.

[QS81] J. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In M. Dezani-Ciancaglini and U. Montanari, editors, *Fifth International Symposium on Programming*, Lecture Notes in Computer Science 137, pages 337–351. Springer-Verlag, 1981.

[Wol88] S. Wolfram. *Mathematica: A System for Doing Mathematics by Computer.* Addison-Wesley Publishing Company, 1988.