Automatic Symbolic Verification of Embedded Systems

Rajeev Alur, Thomas A. Henzinger, and 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.

Index Terms—Specification and verification, real-time and hybrid systems, model checking, symbolic analysis, temporal logic.

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 [18], [9], [8].

In this paper, we pursue the approach suggested in [3], [2] 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. The procedure can also be used to synthesize requirements on the parameter values for a hybrid system to satisfy its specification. Third, we report on an implementation of the verification procedure.

Hybrid automata. We model the components of a hybrid system as *Hybrid Automata* [3]. 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 instantaneously at various points in time by moving through a finite set of control locations. The continuous activities 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 are governed by differential equations. This model for hybrid systems is inspired by the Phase Transition Systems of [37], [39], and can be viewed as a generalization of Timed Automata [4]; a similar model has been proposed and studied independently in [38].

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 [21].

Integrator logic. Real-time requirements of systems can be specified in TCTL [1], a branching-time logic that extends CTL [12] with clock variables. In Section 3, we introduce

- *R. Alur is with the Computing Sciences Research Center, Bell Laboratories, Murray Hill, NJ 07974. E-mail: alur@research.att.com.*
- *T.A. Henzinger is with the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, CA 94720-1770. E-mail: tah@eecs.berkeley.edu.*
- *P.-H. Ho is with Intel Development Labs, Intel Corporation, Hillsboro, OR 97124. E-mail: pho@ichips.intel.com.*

Manuscript received Mar. 20, 1995; accepted Jan. 13, 1996.
A preliminary version of this paper appeared in the Proceedings of the 14th Annual IEEE Real-Time Systems Symposium (RTSS 1993), pp. 2-11.
For information on obtaining reprints of this article, please send e-mail to: transactions@computer.org, and reference IEEECS Log Number S95738.

0098-5589/96$05.00 ©1996 IEEE

Integrator Computation Tree Logic, ICTL, which strengthens TCTL in the style of [10] 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 [11], [40], [34], [36]. In recent years, model-checking algorithms have been developed also for real-time systems that are described by discrete programs with real-valued clocks [5], [1].

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 [28]. It has been observed in [3], [38] 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 quantifier-free formulas are Boolean combinations of linear inequalities (the theory admits quantifier elimination). 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 [41] 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 five 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,
4) the temperature control of a nuclear reactor, approximating exponential temperature changes using rectangular differential inclusions, and
5) a distributed control system consisting of two sensors and a controller that commands a robot based on the sensor readings.

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 [7]. 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 [3], [32]. Notwithstanding, we have found that the method is of practical interest. First, the successive-approximation process converges on several nontrivial applications we have been able to verify using HYTECH [22], [25], [31]. 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. The boundary between decidable and undecidable questions about hybrid automata is identified in [27], [20].

2 System Description Language: Hybrid Automata

Informally, a hybrid automaton [3] 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 constraints on the values of $\vec{x}$ before and after actions. The vertices V represent continuous environment activities and are labeled with constraints on the values of $\vec{x}$ and $\dot{\vec{x}}$ during activities. The state of the automaton changes either through instantaneous system actions or, while time elapses, through differentiable environment activities. In this paper, we restrict ourselves to edge and vertex constraints that are linear expressions. A hybrid system is described as a collection of hybrid automata, one per component, that operate concurrently and coordinate with each other. The communication is achieved via shared variables as well as synchronization labels.

2.1 Syntax

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

A *linear hybrid automaton* $A = (\vec{x}, V, inv, dif, E, act, L, syn)$ 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 convex polyhedron in $\mathbb{R}^n$. A *data region* is a finite union of convex data regions. A data region is *almost convex* if it is connected and has a convex closure. 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.

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 convex polyhedron in $\mathbb{R}^n$. A *rate predicate* is a convex linear formula over the vector $\dot{\vec{x}} = (\dot{x}_1, \ldots, \dot{x}_n)$ of dotted variables. The rate predicate r defines the differential inclusion $[\![r]\!] \subseteq \mathbb{R}^n$, where $\dot{\vec{s}} \in [\![r]\!]$ iff $r[\dot{\vec{x}} := \dot{\vec{s}}]$ 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* $q = (\vec{y}, q')$ consists of a set $\vec{y} \subseteq \vec{x}$ of *updated variables* and a convex linear formula q' over the set $\vec{x} \uplus \vec{y}'$ of data variables and primed updated variables. The *closure* $\{q\}$ of the action predicate q is the convex linear formula $q' \wedge \bigwedge_{x \in \vec{x} \setminus \vec{y}} (x' = x)$; that is, all data variables that are not updated remain unchanged. The action predicate q defines a function $[\![q]\!]$ from data states to convex data regions: For all data states $\vec{s}, \vec{s}' \in \mathbb{R}^n$, let $\vec{s}' \in [\![q]\!](\vec{s})$ iff $\{q\}[\vec{x}, \vec{x}' := \vec{s}, \vec{s}']$ is true. The action predicate q is *enabled* in the data state $\vec{s}$ if the data region $[\![q]\!](\vec{s})$ is nonempty.

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 $inv(v)$, 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 $inv(v)$ is true. The state $(v, \vec{s})$ is *admissible* if $\vec{s} \in [\![inv(v)]\!]$. We write Σ_A for the set of admissible states of A, and ϕ_A for the state predicate $\bigcup_{v \in V}(v, inv(v))$ 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 $dif(v)$, 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 $[\![dif(v)]\!]$.

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. The automaton control can proceed from the location v to the location v' via the transition $e = (v, v')$ only when the action $act(e)$ is enabled. If $act(e)$ is enabled in the data state $\vec{s}$, then the values of all data variables change nondeterministically from $\vec{s}$ to some point in the data region $[\![act(e)]\!](\vec{s})$. For example, if $n = 4$, a transition with the action label

$$\left(\{x_1, x_2, x_3\},\ x_1 \leq 3 \wedge 3 \leq x_1' \leq 5 \wedge x_2' = x_1 + 1\right)$$

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 new value of x_2 is 1 greater than the old value of x_1, the new value of x_3 is any real number, and the value of x_4 remains unchanged (because x_4 is not updated). The action predicates can be used for synchronizing hybrid automata via shared variables, which is illustrated in the mutual-exclusion protocol of Section 7.2. All stutter transitions are labeled with the action $(\vec{x}, \vec{x}' = \vec{x})$, and thus, leave all data variables unchanged.

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 (that is, a transition whose synchronization label contains a) of one automaton must be accompanied by an a-transition of the other automaton. The use of synchronization labels in composition is illustrated in the railroad gate controller of Section 2.4. 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 [4]. 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 $e \in E$, the closure $\{act(e)\}$ 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 $x + c \leq y$, $x \leq c$, 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 $\rho: [0, \delta] \to \mathbb{R}^n$ 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 a 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 $\bar{s} \in \mathbb{R}^n$ such that $\frac{d\rho(t)}{dt} = \bar{s}$ for all $t \in (0, \delta)$. The data trajectory (δ, ρ) is *piecewise linear* if there are *finitely* many reals $t_1, \ldots, t_k$ with $0 < t_1 < \cdots < t_k < \delta$ such that the data trajectories $\rho[0, t_1], \rho[t_1, t_2], \ldots, \rho[t_k, \delta]$ are linear. If $k = 1$, then (δ, ρ) is called *two-piece linear*.

The data trajectory (δ, ρ) is a v-trajectory, for a location $v \in V$, if

1) for all reals $t \in [0, \delta]$, $\rho(t) \in [\![inv(v)]\!]$, and
2) for all reals $t \in (0, \delta)$, $\frac{d\rho(t)}{dt} \in [\![dif(v)]\!]$.

A *trajectory* τ of A is an infinite sequence

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

of control locations $v_i \in V$ and corresponding v_i-trajectories (δ_i, ρ_i) such that for all $i \geq 0$, 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 $\pi_1 = (i, \epsilon_1)$ and $\pi_2 = (j, \epsilon_2)$ of the trajectory τ. We write $\tau[\pi_1, \pi_2]$ for the trajectory fragment

$$(v_i, \delta_i - \epsilon_1, \rho_i[\epsilon_1, \delta_i]) \to (v_{i+1}, \delta_{i+1}, \rho_{i+1}) \to \cdots \to (v_j, \epsilon_2, \rho_j[0, \epsilon_2]),$$

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

$$(v_i, \delta_i - \epsilon_1, \rho_i[\epsilon_1, \delta_i]) \to (v_{i+1}, \delta_{i+1}, \rho_{i+1}) \to (v_{i+1}, \delta_{i+2}, \rho_{i+2}) \to \cdots$$

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") [28].

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 [28], it is shown how a timed automaton may be turned into a nonzeno timed automaton with the same divergent trajectories; this transformation 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 synchronization labels. 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 $dif(v, v') = dif_1(v) \wedge dif_2(v')$. Thus, 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 vectors obey 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) $v_1 = v_1'$ and there is a transition $e_2 = (v_2, v_2') \in E_2$ with $L_1 \cap syn_2(e_2) = \emptyset$; or
  2) there is a transition $e_1 = (v_1, v_1') \in E_1$ with $syn_1(e_1) \cap L_2 = \emptyset$, and $v_2 = v_2'$; or
  3) there is a transition $e_1 = (v_1, v_1') \in E_1$ and a transition $e_2 = (v_2, v_2') \in E_2$ such that $syn_1(e_1) \cap L_2 = syn_2(e_2) \cap L_1$.

  In case 1), $act(e) = act_2(e_2)$ and $syn(e) = syn_2(e_2)$. In case 2), $act(e) = act_1(e_1)$ and $syn(e) = syn_1(e_1)$. In case 3), if

  $$act_1(e_1) = (\vec{y}_1, q_1') \text{ and } act_2(e_2) = (\vec{y}_2, q_2'),$$

  then

  $$act(e) = (\vec{y}_1 \cup \vec{y}_2, q_1' \wedge q_2'),$$

  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. When two transitions are synchronized, the set of updated variables is the union of the updated variables of the component transitions, and the constraints on the updated values are obtained by taking the conjunction of constraints imposed by the component transitions. This explains the role of the updated variables in the action predicates. For instance, the action predicate $q_1 = (\{x\}, x' = x)$ is different from the action predicate $q_2 = (\emptyset, true)$ even though their closures $\{q_1\}$ and $\{q_2\}$ are identical: When composed with a transition with the action predicate $(\{x\}, x' = 2)$, the first one gives $(\{x\}, x = 2 \wedge x' = 2)$, while the second one gives $(\{x\}, x' = 2)$.

Notice that, in case 3), 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. Since the dotted variable $\dot{x}$ is the first derivative of x with respect to time, $\dot{x}$ represents the velocity of the train. 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 1,000 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 1,500 meters, so when the sensor detects a leaving train, the next (or returning) train is at least 1,500 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 Fig. 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 and activities of the form *true*, actions of the form $(\emptyset, true)$, and stutter transitions. Actions are written as nondeterministic guarded commands. For example, the nondeterministic guarded command $x = 100 \rightarrow x := [1{,}500, \infty)$ denotes the action $(\{x\}, x = 100 \wedge x' \geq 1{,}500)$. 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 action $(\emptyset, y = 90)$, and no synchronization labels; the transition to *down* has the action $(\emptyset, true)$ 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.

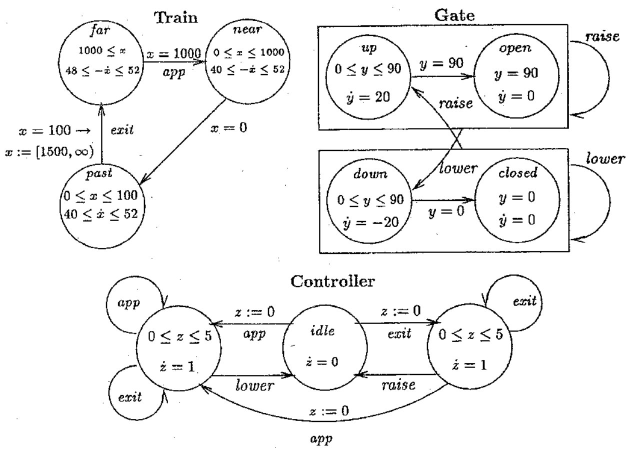


Fig 1. Railroad gate controller.

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 variables of A, and integrators. An integrator (or stop watch) is a clock that can be stopped and restarted, and is inspired by the concept of a *duration* in *Duration Calculus* [13]. We adopt the notation of [10] to generalize the clock reset ("freeze") quantifier of TPTL [6] 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 $(z : U).\ \varphi$ 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 of 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 a 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* (disjoint from $\vec{x}$). 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

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

where ϕ is a $\vec{z}$-extended state predicate of A, $U \subseteq V$ 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 V, then φ is a formula of TCTL [1]. When writing ICTL-formulas, we suppress the integrator type V, and we use Boolean combinations of location constraints for defining integrator types. Typical abbreviations for ICTL-formulas include the standard temporal operators $\forall\Diamond\varphi$, $\exists\Diamond\varphi$, and $\forall\Box\varphi$, for $true\forall\mathcal{U}\varphi$, $true\exists\mathcal{U}\varphi$, and $\neg\exists\Diamond\neg\varphi$, respectively. We also use time-bounded temporal operators such as $\forall\Diamond_{\leq 5}\varphi$, which stands for the ICTL-formula $z.\forall\Diamond(z \leq 5 \wedge \varphi)$; that is, $(z : true).(true\forall\mathcal{U}(z \leq 5 \wedge \varphi))$.

3.2 Semantics

Every closed A-formula ψ of ICTL defines a region $[\![\psi]\!]_A$ of the hybrid 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 $U_1, \ldots, U_m$ be the corresponding types (as specified by ψ). From the n-dimensional linear hybrid automaton $A = (\vec{x}, V, inv, dif, E, act, L, syn)$ we construct the $(n + m)$-dimensional $\vec{z}$*-extension* of A as the linear hybrid automaton $A_{\vec{z}} = (\vec{x} \uplus \vec{z}, V, inv, dif', E, act, L, syn)$ such that for each location $v \in V$,

$$dif'(v) = dif(v) \wedge \bigwedge_{\substack{1 \leq i \leq m \\ v \in U_i}} (\dot{z}_i = 1) \wedge \bigwedge_{\substack{1 \leq i \leq m \\ v \notin U_i}} (\dot{z}_i = 0).$$

Thus, in each location v, the integrator z_i increases with time if $v \in U_i$, and stays unchanged if $v \notin U_i$; the transitions of the automaton $A_{\vec{z}}$ leave all integrator variables unchanged. Each data state $\vec{s} \in \mathbb{R}^{n+m}$ of $A_{\vec{z}}$ consists of an $\vec{x}$-*projection* $\vec{s}|_{\vec{x}} \in \mathbb{R}^m$, 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 $\sigma \in [\![\phi]\!]$, for a state predicate ϕ;
- $\sigma \models_{\mathcal{T}} \neg\varphi$ 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 $\pi' \le \pi$, 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 $\pi' \le \pi$, then $\tau(\pi') \models_{\mathcal{T}} \varphi_1 \vee \varphi_2$;
- $\sigma \models_{\mathcal{T}} (z{:}U).\varphi$ 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 [28].

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 $[\psi]_{[A]^{div}}$ (recall that $[A]^{div}$ is the set of divergent trajectories of A). 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 if 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 2.4. The initial condition of the system is given by the state predicate

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

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*

$$\phi_0 \to \forall\Box(x \le 10 \to \ell[2] = \mathit{closed})$$

asserts that whenever a train is within 10 meters of the gate, the gate must be closed (we write $\ell[i]$ for the ith 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*

$$\phi_0 \to \forall\Box\forall\Diamond(\ell[2] = \mathit{open})$$

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*

$$\phi_0 \to \forall\Box\forall\Diamond_{\le 37}(\ell[2] = \mathit{open})$$

that the gate will always open within 37 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

$$\phi_0 \to \forall\Box z_1.(z_2 : \ell[2] = \mathit{open}).\forall\Box(z_1 \ge 3600 \to 10z_2 \ge 8z_1).$$

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, define $\sigma_1 \stackrel{Q}{\to} \sigma_2$ if $v_1 = v_2$, and there exists a v_1-trajectory (δ, ρ) such that

1) $\rho(0) = \vec{s}_1$,
2) $\rho(\delta) = \vec{s}_2$, and
3) for all reals $t \in [0, \delta]$, $\rho(t) \in Q_v$.

In other words, $(v, \vec{s}_1) \stackrel{Q}{\to} (v, \vec{s}_2)$ if 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) a *witness* for the time step $(v, \vec{s}_1) \stackrel{Q}{\to} (v, \vec{s}_2)$. We define $\sigma_1 \stackrel{Q}{\to}_{lin} \sigma_2$ if there is a piecewise-linear witness for $\sigma_1 \stackrel{Q}{\to} \sigma_2$; and $\sigma_1 \stackrel{Q}{\to}_1 \sigma_2$, if there is a linear witness for $\sigma_1 \stackrel{Q}{\to} \sigma_2$; and $\stackrel{Q}{\to}_2 = \stackrel{Q}{\to}_1 \circ \stackrel{Q}{\to}_1$. Clearly, $\stackrel{Q}{\to}_1 \subseteq \stackrel{Q}{\to}_2 \subseteq \stackrel{Q}{\to}_{lin} \subseteq \stackrel{Q}{\to}$. For simplicity, we write $\to$ for $\stackrel{\Sigma_A}{\to}$.

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

A binary relation $\overset{Q}{\Rightarrow}$ on the states of A is *Q-reflexive* if

1) $\sigma_1 \overset{Q}{\Rightarrow} \sigma_2$ implies $\sigma_1, \sigma_2 \in \Sigma_A \cap Q$, and
2) for all $\sigma \in \Sigma_A \cap Q$, we have $\sigma \overset{Q}{\Rightarrow} \sigma$.

The time-step relation $\overset{Q}{\rightarrow}$ is Q-reflexive because of witness trajectories with duration 0, and the transition-step relation $\overset{Q}{\mapsto}$ is Q-reflexive because of stutter transitions.

We show that for linear regions Q the time-step relation $\overset{Q}{\rightarrow}$ and the piecewise-linear time-step relation $\overset{Q}{\rightarrow}_{lin}$ coincide. We begin with the case of a convex data region Q.

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

PROOF. First, consider the case that S is convex. 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 $\rho'(t) = \vec{s}_1 + t \cdot (\frac{\vec{s}_2 - \vec{s}_1}{\delta})$. Since S is convex and ρ' is linear with both endpoints in S, we have $\rho'(t) \in S$ for all $t \in [0, \delta]$. Suppose that the rate predicate $dif(v)$ is a conjunction of linear inequalities over $\vec{x}$, and let $(c_0 \sim \vec{c} \cdot \vec{x})$ be one of its conjuncts, where $\sim \in \{<, \leq\}$ and $\vec{c} \cdot \vec{x}$ is the inner product of a constant vector $\vec{c}$ and vector $\vec{x}$. For all $t \in [t_i, t_{i+1}]$, $c_0 \sim \vec{c} \cdot \frac{d\rho(\vec{x})(t)}{dt}$. Integrating both sides of the above inequality from 0 to δ, we get $c_0 \cdot \delta \sim \vec{c} \cdot (\vec{s}_2 - \vec{s}_1)$. This gives $c_0 \sim \vec{c} \cdot \frac{d\rho'(\vec{x})(t)}{dt}$ for all $t \in (0, \delta)$, and hence, $\frac{d\rho'(\vec{x})(t)}{dt} \in [\![dif(v)]\!]$ for all $t \in (0, \delta)$. Thus, ρ' is the desired linear v-trajectory. If S is not convex, but almost convex, then there is a linear v-trajectory from $\vec{s}_1$ to some data state $\vec{s}_3$ in the interior of S, and a linear v-trajectory from $\vec{s}_3$ to $\vec{s}_2$. □

We now proceed to the general case. Let p be a data predicate in the disjunctive normal form $p = p_1 \vee \ldots \vee p_k$, with each disjunct p_i of the form $\bigwedge_j(a_j \sim 0)$ for linear terms a_j. For each i, let p_i' be the data predicate resulting from the data predicate p_i by replacing each strict inequality $a_j > 0$ or $a_j < 0$ by the corresponding nonstrict inequality $a_j \geq 0$ or $a_j \leq 0$, respectively. The almost convex data region $[\![p_i' \wedge p]\!]$ is called a *patch* of p.

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 $(v, \vec{s}_1) \overset{Q}{\rightarrow} (v, \vec{s}_2)$ iff $(v, \vec{s}_1) \overset{Q}{\rightarrow}_{lin} (v, \vec{s}_2)$. If Q_v is convex linear, then $(v, \vec{s}_1) \overset{Q}{\rightarrow} (v, \vec{s}_2)$ iff $(v, \vec{s}_1) \overset{Q}{\rightarrow}_1 (v, \vec{s}_2)$.*

PROOF. We show that

$$(v, \vec{s}_1) \overset{Q}{\rightarrow} (v, \vec{s}_2)$$

implies $(v, \vec{s}_1) \overset{Q}{\rightarrow}_{lin} (v, \vec{s}_2)$ assuming Q_v is linear. Consider a witness (δ, ρ) for $(v, \vec{s}_1) \overset{Q}{\rightarrow} (v, \vec{s}_2)$. If $\delta = 0$, the witness (δ, r) is linear, so consider $\delta > 0$. Without loss of generality assume that $Q_v \subseteq [\![inv(v)]\!]$, and Q_v is represented by a linear data predicate p in disjunctive normal form.

Let $0 = t_0 < t_1 < t_2 < \cdots < t_m = \delta$ be the (finite) sequence of time values and the sequence $Q_0, \ldots, Q_{m-1}$ of patches of p such that for each $0 \leq i < m$, the patch Q_i contains $\rho(t_i)$ and $\rho(t_{i+1})$ (the interior points of $\rho[t_i, t_{i+1}]$ may not all belong to the patch Q_i). Since the number of patches is finite, such a sequence exists (note that each patch may appear at most once, and thus, the length is bounded by the number of patches).

By Lemma 1, for each $0 \leq i < m$, there is a two-piece 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 $\rho_i(t) \in Q_i$ for all $t \in [0, t_{i+1} - t_i]$. So the concatenation of these piecewise-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 $(v, \vec{s}_1) \overset{Q}{\rightarrow} (v, \vec{s}_2)$, and thus $(v, \vec{s}_1) \overset{Q}{\rightarrow}_{lin} (v, \vec{s}_2)$.

If Q_v is convex, then there is a single convex patch, and $m = 1$. This gives $(v, \vec{s}_1) \overset{Q}{\rightarrow}_1 (v, \vec{s}_2)$ by Lemma 1. □

4.2 Precondition Operators

Let Q and R be two regions of the linear hybrid automaton A, and let $\overset{Q}{\Rightarrow}$ be a Q-reflexive binary relation on the states of A. The $\overset{Q}{\Rightarrow}$*-precondition* $pre_{\overset{Q}{\Rightarrow}}(R)$ 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, $\sigma \in pre_{\overset{Q}{\Rightarrow}}(R)$ 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 extensive on the subregions of $\Sigma_A \cap Q$; that is, for all regions $R \subseteq \Sigma_A \cap Q$, we have $R \subseteq pre_{\overset{Q}{\Rightarrow}}(R) \subseteq \Sigma_A \cap Q$.

We define 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 $pre_{\overset{Q}{\rightarrow}}(R)$ and $pre_{\overset{Q}{\mapsto}}(R)$. This is done by constructing from the state predicates ϕ and χ that define Q and R, respectively, two state predicates $pre_{\overset{\phi}{\rightarrow}}(\chi)$ and $pre_{\overset{\phi}{\mapsto}}(\chi)$ that define $pre_{\overset{Q}{\rightarrow}}(R)$ and $pre_{\overset{Q}{\mapsto}}(R)$, respectively. We then define the A*-precondition* $pre_A^Q(R)$ to be the union $pre_{\overset{Q}{\rightarrow}}(R) \cup pre_{\overset{Q}{\mapsto}}(R)$. If ϕ defines Q, and χ defines R, then $pre_A^Q(R)$ is defined by the state predicate

$$pre_A^{\phi}(\chi) = pre_{\overset{\phi}{\rightarrow}}(\chi) \vee pre_{\overset{\phi}{\mapsto}}(\chi).$$

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, $[\![r_v]\!] = R_v$.

4.2.1 Time Precondition

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

$$pre \overset{\phi}{\to}(\chi) = \bigcup_{v \in V}\left(v,\ pre \overset{q_v}{\to}(v; r_v)\right).$$

The construction of $pre \overset{q_v}{\to}(v; r_v)$ proceeds in two steps. First we construct a data predicate $pre \overset{true}{\to}(v; r_v)$, and then we apply the precondition operator $pre \overset{true}{\to}$ repeatedly to construct the data predicate $pre \overset{q_v}{\to}(v; r_v)$.

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. Define*

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

Then $[\![pre \overset{true}{\to}(v; r_v)]\!] = pre \overset{[\![inv(v)]\!]}{\to}(v; [\![r_v]\!])$. This holds even if the invariant of v defines an almost convex (rather than convex) linear region.

PROOF. A data state $\vec{s}$ satisfies $pre \overset{true}{\to}(v; r_v)$ iff $\vec{s}$ is in $[\![inv(v)]\!]$ and there is a v-trajectory (δ, ρ) such that $\rho(0) = \vec{s}$ and $\rho(\delta) \in R_v$. By Lemma 1, we can assume that (δ, ρ) is two-piece linear. Since $[inv(v)]$ is almost convex, we can assume that (δ, ρ) is linear. Hence, it suffices to check that there exist a duration δ and a slope vector $\vec{d}$ such that $\vec{d}$ satisfies the rate predicate $dif(v)$, the initial values $\vec{s}$ satisfy the invariant $inv(v)$, and the final values $\vec{s} + \delta \cdot \vec{d}$ satisfy the conjunction $inv(v) \wedge r_v$. □

The formula $pre \overset{true}{\to}(v; r_v)$ 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

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

is equivalent to the formula

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

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. Define*

$$pre \overset{true}{\to}(v; r_v) = inv(v) \wedge$$
$$\left(\exists \delta \geq 0.\exists \vec{c}.\left(\delta \cdot dif(v)\right)\left[\vec{\dot{x}} := \vec{c}\right] \wedge \left(r_v \wedge inv(v)\right)\left[\vec{x} := \vec{x} + \vec{c}\right]\right).$$

Then $[\![pre \overset{true}{\to}(v; r_v)]\!] = pre \overset{[\![inv(v)]\!]}{\to}(v; [\![r_v]\!])$.

The formula $pre \overset{true}{\to}(v; r_v)$ is a formula of the first-order theory $(\mathbb{R}, \leq, +)$ of the reals with addition. Since this theory admits quantifier elimination, the formula $pre \overset{true}{\to}(v; r_v)$ is equivalent to a data predicate. The quantifier-elimination procedure used by HYTECH is discussed in Section 6.1.

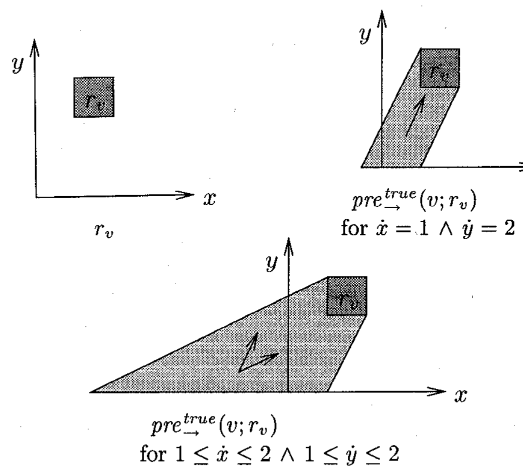


Fig. 2. The time-precondition operator $pre \overset{true}{\to}$.

EXAMPLE. Let us consider two examples of computing the data predicate $pre \overset{true}{\to}(v; r_v)$ 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 top left of Fig. 2). Then, according to Proposition 1,

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

Eliminating the existential quantifiers inside out, we obtain

$$pre \overset{true}{\to}(v; r_v) = \left(y \geq 0 \wedge \left(\exists \delta \geq 0.1 \leq x + \delta \leq 2 \wedge 2 \leq y + 2\delta \leq 3\right)\right)$$

and, finally, the data predicate

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

(see Fig. 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,

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

Eliminating the existential quantifiers, we obtain

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

and the equivalent data predicate

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

(see Fig. 2). □

We now reduce the construction of the data predicate $pre \overset{q_v}{\longrightarrow}(v, r_v)$ to a sequence of applications of the precondition operator $pre \overset{true}{\longrightarrow}$ 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

$$pre \overset{Q_v}{\longrightarrow}(v; R_v) = pre \overset{[inv(v')]}{\longrightarrow}(v'; R_v).$$

The invariant $inv(v')$, however, may not be (almost) convex, in which case we cannot compute the data predicate $pre \overset{true}{\longrightarrow}(v'; r_v)$ using Proposition 1.

So, second, we split the location v' into several locations with almost convex invariants. Let $p_1, \ldots, p_k$ be the predicates that define the patches of the data predicate $inv(v')$, that is, $inv(v') = p_1 \vee \ldots \vee p_k$. 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

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

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 $\overset{Q}{\rightarrow}$-step, however, may proceed from the data region $pre \overset{Q_v}{\longrightarrow}(v; R_v)$ to the data region R_v through more than one of the patches of $inv(v')$. Since there are k different patches, it will suffice to iterate the precondition $pre_{\rightarrow}(V'; R_v)$ k times. So, third, we define a sequence of k data regions, R_0 to R_k, such that $R_0 = R_v$ and for all $1 \leq j \leq k$, $R_j = pre_{\rightarrow}(V'; R_{j-1})$. Let $q_0, \ldots, q_k$ be the sequence of corresponding data predicates; that is, for all $0 \leq j \leq k$, $[q_j] = R_j$. The next proposition shows that $R_k = pre \overset{[inv(v')]}{\longrightarrow}(v'; R_v)$. Thus the data predicate $pre \overset{q_v}{\longrightarrow}(v; r_v) = q_k$ defines the data region $pre \overset{Q_v}{\longrightarrow}(v; R_v)$.

PROPOSITION 2. *Let A be a linear 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 the data predicates defining the patches of $inv(v) \wedge q_v$. 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 $R_0 = [r_v]$ and for all $1 \leq j \leq k$, $R_j = pre_{\rightarrow}(V'; R_{j-1})$, then $R_k = pre \overset{[q_v]}{\longrightarrow}(v; [r_v])$.*

PROOF. Let v' be as above. Since $dif(v'_i) = dif(v)$ and $\bigvee_{1 \leq i \leq k} inv(v'_i) = inv(v')$, by definition, R_k defines the set of data states that can reach a data state in R_v by k $\rightarrow_2$-steps. According to the definition of $pre_{\rightarrow}(v'; R_v)$, $R_k \subseteq pre_{\rightarrow}(v'; R_v)$. On the other hand, since $p_1, \ldots, p_k$ define the patches of $inv(v')$, 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 $2k$ linear data trajectories. In other words, $(v', \vec{s}) \rightarrow (v', \vec{s}')$ implies that for some $0 \leq j \leq k$, we have $(v', \vec{s}) \rightarrow_2^j (v', \vec{s}')$. Consequently, $pre_{\rightarrow}(v'; R_v) \subseteq R_k$. □

EXAMPLE. Let us consider an example of computing the data predicate $pre \overset{q_v}{\longrightarrow}(v; r_v)$ 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 $inv(v') = q_v$. Since $inv(v')$ is not convex, and

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

define the patches 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

$$\begin{aligned} q_0 &= r_v &&= (3 \leq x \leq 4 \wedge 2 \leq y \leq 3), \\ q_1 &= pre_{\rightarrow}(V'; q_0) &&= (2 \leq x \leq 4 \wedge 0 \leq y \leq 3 \wedge 1 \leq x - y \leq 3), \\ q_2 &= pre_{\rightarrow}(V'; q_1) &&= ((2 \leq x \leq 4 \wedge 0 \leq y \leq 3 \wedge 1 \leq x - y \leq 3) \vee \\ &&& \quad (1 \leq x \leq 2 \wedge 0 \leq y \leq 1 \wedge 1 \leq x - y)) \end{aligned}$$

(see Fig. 3). By Proposition 2, $pre \overset{q_v}{\longrightarrow}(v; r_v) = q_2$. □

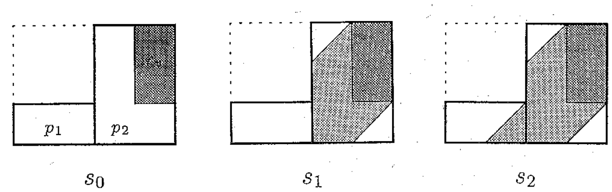


Fig. 3. The time-precondition operator $pre \overset{q_v}{\longrightarrow}$.

4.2.2 Transition Precondition

We write $pre \overset{Q}{\longmapsto}(v; R_v)$ 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 $pre \overset{Q}{\longmapsto}(v; R_v)$ is linear by constructing from

ϕ and r_v a state predicate $pre_{\mapsto}^{\phi}(v; r_v)$ that defines the region $pre_{\mapsto}^{Q}(v; R_v)$. Then

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

The next proposition constructs $pre_{\mapsto}^{\phi}(v; r_v)$ 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. Define*

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

Then $[\![pre_{\mapsto}^{\phi}(v; r_v)]\!] = pre_{\mapsto}^{[\![\phi]\!]}(v; [\![r_v]\!])$.

PROOF. For each location v', the state $(v', \vec{s})$ belongs to the region $pre_{\mapsto}^{[\![\phi]\!]}(v; [\![r_v]\!])$ if s satisfies the invariant $inv(v') \wedge q_{v'}$ and there exists a data state $\vec{s}'$ after the transition $e = (v', v)$ such that $\vec{s}'$ satisfies the target predicate r_v, the invariant $inv(v) \wedge q_v$, and the pair $(\vec{s}, \vec{s}')$ satisfies the action predicate $act(e)$. □

EXAMPLE. Let us consider an example of computing the state predicate $pre_{\mapsto}^{\phi}(v; r_v)$ using Proposition 3. Suppose that the linear hybrid automaton A has two data variables, x and y, and only one nonstutter transition, $e = (v', v)$, with the target location v. Suppose that $act(e) = (\{x, y\}, x \leq 3 \wedge x' \geq 5 \wedge y' = x)$, 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,

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

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

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

and, finally, the state predicate

$$pre_{\mapsto}^{\phi}(v; r_v) = (v', x \leq 2) \vee (v, x \geq 6 \wedge y \leq 2).$$ □

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 $[\![pre_A^{\phi}(\chi)]\!] = pre_A^{[\![\phi]\!]}([\![\chi]\!])$.*

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) [3], [32]. In [28], 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. The SMC-procedure uses the hybrid automaton $A_{\vec{z}}$, which extends A with the integrators from $\vec{z}$ (see Section 3.2), and the precondition operator $pre_{A_{\vec{z}}}$ on state predicates, which was introduced and computed in Section 4.2. Observe that if the given automaton A is nonzeno, then so is the $\vec{z}$-extension $A_{\vec{z}}$, and that $\phi_{A_{\vec{z}}} = \phi_A$. Also, a characteristic predicate of $(A_{\vec{z}}, \psi)$ can always be simplified to a characteristic predicate of (A, ψ), because the former does not constrain the integrators from $\vec{z}$.

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

1. As the state predicates are quantifier-free formulas of the theory $(\mathbb{R}, \leq, +)$, validity can be decided.

$\chi_0 := |\varphi_2|$ and

$\chi_{i+1} := \chi_i \vee \left| \neg z_{\varphi_1 \forall \mathcal{U} \varphi_2} . ((\neg \chi_i) \exists \mathcal{U} (\neg(\varphi_1 \vee \chi_i) \vee z_{\varphi_1 \forall \mathcal{U} \varphi_2} > 1)) \right|$ [2];

$|(z : U).\varphi| := |\varphi|[z := 0]$ (that is, replace all occurrences of z in $|\varphi|$ by 0). □

The computation corresponding to the formulas of the form $\varphi_1 \exists \mathcal{U} \varphi_2$ and $\varphi_1 \forall \mathcal{U} \varphi_2$ is explained in the subsequent subsections.

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. Let $\varphi = \phi_1 \exists \mathcal{U} \phi_2$ and $Q = [\varphi]_B$. We define the functions $g, f : \Sigma_B \to \Sigma_B$ such that $g(X) = pre_B^{[\phi_1 \vee \phi_2]}(X)$ and $f(X) = [\phi_2] \cup g(X)$.

We first prove that Q is a fixpoint of the function f. Since the precondition operator g is extensive on the subregions of Σ_B, $Q \subseteq g(Q) \subseteq f(Q)$. On the other hand, consider $\sigma \in g(Q)$. By definition, there is a state σ' in Q such that either $\sigma \overset{[\phi_1 \vee \phi_2]}{\rightarrow} \sigma'$ or $\sigma \overset{[\phi_1 \vee \phi_2]}{\mapsto} \sigma'$.

In either case, by the definition of trajectories of a linear hybrid automaton and the definition of $\exists \mathcal{U}$, state σ is in Q. Thus $g(Q) \subseteq Q$. Moreover, by definition, $[\phi_2] \subseteq Q$. Therefore, $f(Q) \subseteq Q$.

Now we show that Q is the least fixpoint of f. Let R be a fixpoint of f. Consider $\sigma \in Q$. Then there is a trajectory $\tau \in [B]^{div}$ and a position (k, δ) of τ such that $\tau(0, 0) = \sigma$, $\tau(k, \delta) \in [\phi_2]$, and for all positions $(j, \epsilon) < (k, \delta)$, $\tau(j, \epsilon) \in [\phi_1 \vee \phi_2]$. Since $[\phi_2] \subseteq R$, $\tau(k, \delta) \in R$. By the definition of the precondition operator g, we know that $\tau(k - 1, 0) \in f(R)$. Since R is a fixpoint of f and thus $f(R) = R$, we conclude that $\tau(k - 1, 0) \in R$. Inductively, $\tau(k - k', 0) \in R$ for all $1 \le k' \le k$. In particular, $\sigma = \tau(0, 0) \in R$, which implies that $Q \subseteq R$. □

The SMC-procedure computes the least solution of the equation $f(X) = X$ as the limit of the successive-approximation sequence $\emptyset, f(\emptyset), f^2(\emptyset), f^3(\emptyset), \ldots$ 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.

2. The constant 1 can be replaced by any positive integer. This choice may influence the number of iterations.

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 [28]. 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.

5.3.1 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* $\forall \mathcal{U}_{\le c}$ on regions such that for all state predicates ϕ_1 and ϕ_2, $[\phi_1] \forall \mathcal{U}_{\le c} [\phi_2] = [\phi_1 \forall \mathcal{U}_{\le c} \phi_2]_B$. Given two regions Q and R of B, the region $Q \forall \mathcal{U}_{\le c} R$ contains all states σ such that 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 $t_\tau(\pi) \le c$, and for all positions π' of τ, if $\pi' \le \pi$ then $\tau(\pi') \in Q \cup R$. Notice that for nonzeno B, the $\forall \mathcal{U}_{\le c}$-operator is extensive in its second argument on the subregions of Σ_B; that is, for all regions Q, $R \subseteq \Sigma_B$, we have $R \subseteq Q \forall \mathcal{U}_{\le c} R \subseteq \Sigma_B$.

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 be a positive integer constant. Then $[\phi_1 \forall \mathcal{U} \phi_2]_B$ is the least solution of the equation $X = [\phi_2] \cup ([\phi_1 \vee \phi_2] \forall \mathcal{U}_{\le c} X)$.*

PROOF. Let $Q = [\phi_1 \forall \mathcal{U} \phi_2]_B$, and define the functions $g, f : \Sigma_B \to \Sigma_B$ such that $g(X) = [\phi_1 \vee \phi_2] \forall \mathcal{U}_{\le c} X$ and $f(X) = [\phi_2] \cup g(X)$.

We first show that Q is a fixpoint of f. Since the $\forall \mathcal{U}_{\le c}$ –operator g is extensive on the subregions of Σ_B, $Q \subseteq g(Q) \subseteq f(Q)$. On the other hand, consider $\sigma \in g(Q)$. By the definitions of f and the $\forall \mathcal{U}_{\le c}$-operator, for all divergent trajectories τ of B with $\tau(0, 0) = s$, there is a position π of τ such that $\tau(\pi) \in Q$, $t_\tau(\pi) \le c$, and for all positions $\pi' \le \pi$, $\tau(\pi') \in [\phi_1 \vee \phi_2] \cup Q$. By the definition of $\forall \mathcal{U}$, we know that $\sigma \in Q$. Furthermore, by definition, $[\phi_2] \subseteq Q$, and hence, $f(Q) \subseteq Q$.

Now we show that Q is the least fixpoint of f. Let R be a fixpoint of f; we claim that $Q \subseteq R$. Assume that there is a state $\sigma_0 \in Q \backslash R$; we will show a contradiction.

We construct inductively an infinite sequence of states $\sigma_i \in Q \backslash R$, for $i \in \mathbb{N}$. We show how to construct σ_{i+1} from σ_i. Since $\sigma_i \notin R$ and $R = f(R)$, we know that $\sigma_i \notin g(R)$. Since B is nonzeno, there is a trajectory $\tau_i \in [B]^{div}$ with $\tau_i(0, 0) = \sigma_i$, and a position π_i of τ_i with $t_{\tau_i}(\pi_i) = c$, such that either

1) for some position $\pi \le \pi_i$ of τ_i, $\tau_i(\pi) \notin [\phi_1 \vee \phi_2]$ and for all positions $\pi' \le \pi$, $\tau_i(\pi') \notin R$; or
2) for all positions $\pi \le \pi_i$ of τ_i, $\tau_i(\pi) \in [\phi_1 \vee \phi_2] \backslash R$.

But since $\sigma_i \in Q$ and $[\![\phi_2]\!] \subseteq R$, Condition 1 cannot happen. Condition 2 together with the assumption $\sigma_i \in Q \backslash R$ implies that for all positions $\pi \le \pi_i$ of τ_i, $\tau_i(\pi) \in Q \backslash R$. So $\tau_i(\pi_i) \in Q \backslash R$ and thus we 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 $i \ge 0$:

$$\tau = (\tau_0[(0, 0), \pi_0], \tau_1[(0, 0), \pi_1], \tau_2[(0, 0), \pi_2], \ldots)$$

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 $\tau(\pi) \notin R$ for all positions π of τ. Because $[\![\phi_2]\!] \subseteq R$, we conclude that $\sigma_0 \notin Q$, which is a contradiction. □

5.3.2 Reduction to Possibility

For linear arguments, the formula $\phi_1 \forall \mathcal{U}_{\le c} \phi_2$ is definable by the $\exists\mathcal{U}$-operator. Intuitively, 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.

Let us define the *weak-until* operator $\forall\mathcal{W}$ as

$\sigma \models_{\mathcal{T}} \varphi_1 \forall\mathcal{W} \varphi_2$ iff for all trajectories $\tau \in \mathcal{T}$ with $\tau(0, 0) = \sigma$, either there exists a position π of τ such that $\tau(\pi) \models_{\mathcal{T}} \varphi_2$ and for all positions $\pi' \le \pi$, $\tau(\pi') \models_{\mathcal{T}} \varphi_1 \vee \varphi_2$; or for all positions π of τ, $\tau(\pi) \models_{\mathcal{T}} \varphi_1$.

The relationship between the until operator $\exists\mathcal{U}$ and the weak-until $\forall\mathcal{W}$ is given by the next proposition.

PROPOSITION 6. *For every linear hybrid automaton B, and all state predicates ϕ_1 and ϕ_2 of B,*

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

PROOF. From definitions, if $\sigma \in [\![(\neg\phi_2)\exists\mathcal{U}(\neg\phi_1 \wedge \neg\phi_2)]\!]_B$, then $\sigma \notin [\![\phi_1 \forall\mathcal{W}\phi_2]\!]_B$. It remains to be shown that if $\sigma \notin [\![\phi_1 \forall\mathcal{W}\phi_2]\!]_B$, then $\sigma \in [\![(\neg\phi_2)\exists\mathcal{U}(\neg\phi_1 \wedge \neg\phi_2)]\!]_B$. Assume that $\sigma \notin [\![\phi_1 \forall\mathcal{W}\phi_2]\!]_B$. Then we know that $\sigma \notin [\![\phi_2]\!]$, and there is a trajectory $\tau \in [\![B]\!]^{div}$ with $\tau(0, 0) = \sigma$ such that either

1) for all positions π of τ, $\tau(\pi) \notin [\![\phi_2]\!]$, and there is a position π of τ such that $\tau(\pi) \notin [\![\phi_1]\!]$; or
2) there are some positions π of τ with $\tau(\pi) \in [\![\phi_2]\!]$, and for each such position π, there is a position $\pi' < \pi$ such that $\tau(\pi') \notin [\![\phi_1 \vee \phi_2]\!]$.

Condition 1 implies that $\sigma \in [\![(\neg\phi_2)\exists\mathcal{U}(\neg\phi_1 \wedge \neg\phi_2)]\!]_B$. Now we assume that Condition 2 holds. Let π_0 be the infimum of all positions π with $\tau(\pi) \in [\![\phi_2]\!]$. If $\tau(\pi_0) \in [\![\phi_2]\!]$, then by Condition 2, there is some position $\pi_1 < \pi_0$ with $\tau(\pi_1) \notin [\![\phi_1 \vee \phi_2]\!]$, and since π_0 is the infimum, for all $\pi < \pi_0$, $\tau(\pi) \notin [\![\phi_2]\!]$, implying $\sigma \in [\![(\neg\phi_2)\exists\mathcal{U}(\neg\phi_1 \wedge \neg\phi_2)]\!]_B$. Suppose that $\tau(\pi_0) \notin [\![\phi_2]\!]$. Then, for all positions $\pi \le \pi_0$, $\tau(\pi) \notin [\![\phi_2]\!]$. If there is some position $\pi \le \pi_0$ with $\tau(\pi) \notin [\![\phi_1]\!]$, then we are done. If not, then for each position π_2 such that $\pi_2 > \pi_0$ and $\tau(\pi_2) \in [\![\phi_2]\!]$, there exists a position π_1 such that $\pi_2 > \pi_1 > \pi_0$, and $\tau(\pi_1) \in [\![\neg\phi_1 \wedge \neg\phi_2]\!]$. Let us consider this case further.

Let $\pi_0 = (i, \delta_0)$, and let the ith location on τ be v. Since π_0 is the infimum of all positions π with $\tau(\pi) \in [\![\phi_2]\!]$, there exists a position $\pi_2 = (i, \delta_2)$ in the same location v with $\delta_2 > \delta_0$ and $\tau(\pi_2) \in [\![\phi_2]\!]$. 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. For each $\epsilon > 0$, there exist positive real numbers ϵ', ϵ'' such that $0 < \epsilon'' < \epsilon' < \epsilon$ with $\rho(\delta_0 + \epsilon') \in [\![p_v]\!]$ and $\rho(\delta_0 + \epsilon'') \in [\![q_v]\!]$. Hence, for any $\epsilon > 0$, the open ball $B(\rho(\delta_0), \epsilon)$ contains infinitely many points (data states) in $[\![q_v]\!]$. But $[\![q_v]\!]$ is a finite union of convex data regions. Therefore, there must be some $\hat{\epsilon}$ such that for all $0 < \epsilon \le \hat{\epsilon}$, $\rho(\delta_0 + \epsilon) \in [\![q_v]\!]$. 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 $\rho'(t) \in [\![q_v]\!]$ for all $0 < t \le \hat{\epsilon}$. Since B is nonzeno and $[\![B]\!]$ is fusion-closed, the trajectory fragment $(\tau[(0,0), \pi_0], (v, \hat{\epsilon}, \rho'))$ can be extended to a divergent trajectory of B that is a witness trajectory for $\sigma \in [\![(\neg\phi_2)\exists\mathcal{U}(\neg\phi_1 \wedge \neg\phi_2)]\!]_B$. □

PROPOSITION 7. *Let B be a linear hybrid automaton, let ϕ_1 and ϕ_2 be two state predicates of B, let c be a nonnegative integer constant, and let z be a new clock (that is, different from the data variables of B). Then*

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

PROOF. By the definition of the $\forall\mathcal{U}_{\le c}$-operator and the semantics of the temporal operator $\forall\mathcal{U}$ and the reset quantifier z,

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

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

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

Using Proposition 6,

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

which proves the proposition, since the right-hand side of the equality is equivalent to

$$[\![\neg z.((\neg\phi_2)\exists\mathcal{U}(\neg(\phi_1 \vee \phi_2) \vee z > c))]\!]_B. \quad \square$$

The partial correctness of the SMC-procedure follows from Propositions 4, 5, and 7 (let B be $A_{\hat{z}}$).

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 $pre_{\rightarrow}$ 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 [28].

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 [17]. 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 the 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

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

where q_i does not contain δ, $c_1, \ldots, c_k > 0$ 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':

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

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 $\bigvee p_i'$ is 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 2.4.

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

TransitionNo = {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, lower, {l[2] -> 3}}
act[2,5] = {l[2]==4, lower, {l[2] -> 4}}
act[2,6] = {l[2]==3 && 0==y, {l[2] -> 4}}
act[2,7] = {l[2]==1, raise, {l[2] -> 1}}
act[2,8] = {l[2]==2, raise, {l[2] -> 2}}
act[2,9] = {l[2]==3, raise, {l[2] -> 1}}
act[2,10]= {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 $0 \le x \le 100$.
- `dif[1,3,x]={40,52}` specifies that the activity of the third location of the first component automaton contains the conjunct $40 \le \dot{x} \le 52$.
- `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).
- `act[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 $x = 100 \wedge x' = 1{,}500$, and it is labeled with the synchronization label *exit*. (The latest version of HYTECH permits also nondeterministic guarded assignments [26].)

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, in order 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

$$\ell = (\mathit{far}, \mathit{open}, \mathit{idle}) \to \forall\Box\forall\Diamond_{\le 37}(\ell[2] = \mathit{open})$$

from Section 3.3. 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,37,l[3]==2]]]]
```

Here AA stands for the invariance operator $\forall\Diamond$, 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.

7 EXAMPLES

We now illustrate the application of HYTECH with five examples.

7.1 Railroad Gate Control

Recall the railroad gate controller from Section 2.4, and the initial condition ϕ_0 from Section 3.3. The safety requirement of Section 3.3 can be rewritten as

$$\phi_0 \to \neg\exists\Diamond\big(x \le 10 \wedge \ell[2] \ne \mathit{closed}\big).$$

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

$$\phi\colon \big|\exists\Diamond\big(x \le 10 \wedge \ell[2] \ne \mathit{closed}\big)\big|$$

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

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

3. All performance data are measured on a Sun Sparc-670MP server running MATHEMATICA.

The time-bounded response requirement of Section 3.3 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) [7]. 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 $\dot{x} = 0$, and for all transitions e of A, the closure $\{act(e)\}$ implies $x' = x$.

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 [33], 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 local 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 Fig. 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; that is, their rate of change is 0 in all locations, and they are not altered by any transitions. Also the slope 0 for the discrete variable k is suppressed in all locations.

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

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

The mutual-exclusion requirement is specified by the ICTL-formula

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

The characteristic predicate

$$\phi\colon \left|\phi_0 \wedge \exists\Diamond(\ell = (4, D))\right|,$$

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 $8b > 11a$.

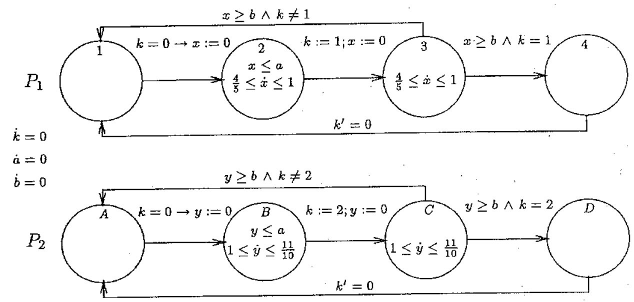


Fig. 4. Timing-based mutual-exclusion protocol.

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 Fig. 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 Fig. 5.

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

Fig. 5. Coefficient size vs. performance.

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 hybrid automata on the left of Fig. 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-1 task is interrupted and resumed only after the completion of the newly arrived type-2 task. The resulting priority scheduler is modeled by the hybrid automaton on the right of Fig. 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.

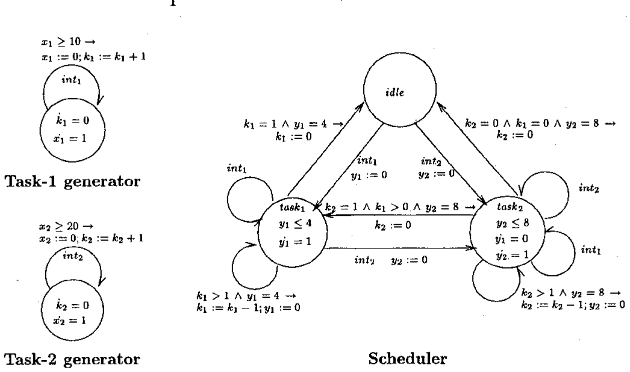


Fig. 6. Preemptive two-task scheduler.

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

$$\phi_0\colon \ell[3] = idle \wedge k_1 = 0 \wedge k_2 = 0.$$

We wish to check that the number of incomplete type-1 tasks never exceeds 2 and the number of 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 to achieve any desired precision by splitting control locations and strengthening the corresponding differential inclusions [21].

Consider, for example, a toy nuclear reactor with two control rods [38]. 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 the 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 Fig. 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 removed. The control rods are modeled by the two hybrid automata at the top of Fig. 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.

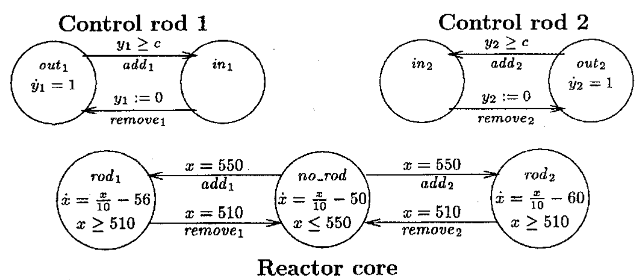


Fig. 7. Nonlinear reactor model.

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 Fig. 8.

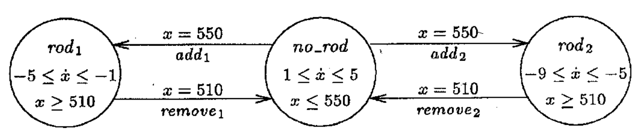


Fig. 8. The approximating linear core automaton.

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

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

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

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

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 that prevents a shutdown of the reactor is computed by HYTECH as $9c \le 184$, using 20.5 seconds of CPU time. Notice that while the condition $9c \le 184$ 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).

7.5 Distributed Robot Control

The distributed control system of [14] consists of two sensors and a controller that generates control commands to a robot according to the sensor readings. The two sensors share a single processor, and the priority of sensor 2 for using the processor is higher than the priority of sensor 1. Hence sensor 2 can preempt sensor 1, which has to wait until the processor is released by sensor 2.

The two sensors are modeled by the two linear hybrid automata in Fig. 9, and the priorities for using the shared processor are modeled by the scheduler automaton in Fig. 10. Each sensor can be constructing a reading (location *read*), waiting to send the reading to the controller (location *wait*), sending the reading (location *send*), or sleeping (location *done*). The processor can be idle (location *idle*), serving sensor 1 (location $sensor_1$), serving sensor 2 while sensor 1 is waiting (location *preempt*), or serving sensor 2 while sensor 1 is not waiting (location $sensor_2$). The shared processor for constructing sensor readings is requested via *request* transitions, the completion of a reading is signaled via *read* transitions, and the reading is delivered to the controller via *send* transitions. Sensor 1 takes 0.5 to 1.1 milliseconds and sensor 2 takes 1.5 to 2 milliseconds of processor time to construct a reading. These times are measured by the stop watches x_1 and x_2 of the scheduler automaton. In each location of the scheduler automaton, at most one of the two stop watches x_1 and x_2 is running, which reflects the fact that only one sensor can use the shared processor at a time.

Once constructed, the reading of sensor 1 expires if it is not delivered within 4 milliseconds, and the reading of sensor 2 expires if it is not delivered within 8 milliseconds. These times are measured by the clocks y_1 and y_2 of the sensor automata. If a reading expires, then a new reading must be constructed. After successfully delivering a reading, a sensor sleeps for 6 milliseconds (measured again by the clocks y_1 and y_2), and then constructs the next reading.

The controller is modeled by the automaton in Fig. 11. The controller is executed on a dedicated processor, so it does not compete with the sensors for processor time. The clock z measures the delays and time-outs of the controller. The controller accepts and acknowledges a reading from each sensor, in either order, and then computes and sends a command to the robot. The sensor readings are acknowledged via *ack* transitions, and the robot command is delivered via a *signal* transition. It takes 0.9 to 1 millisecond to receive and acknowledge a sensor reading. The two sensor readings that are used to construct a robot command must be received within 10 milliseconds. If the controller receives a reading from one sensor but does not receive the reading from the other sensor within 10 milliseconds, then the first sensor reading expires (via an *expire* transition). Once both reading are received, the controller takes 3.6 to 5.6 milliseconds to synthesize a robot command.

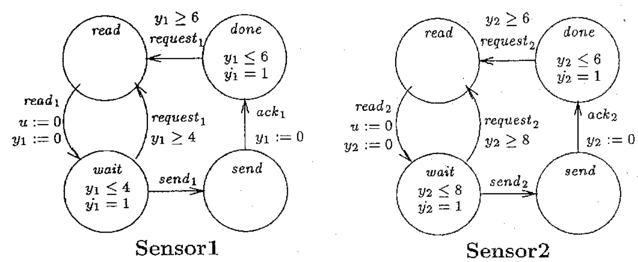


Fig. 9. The two sensors.

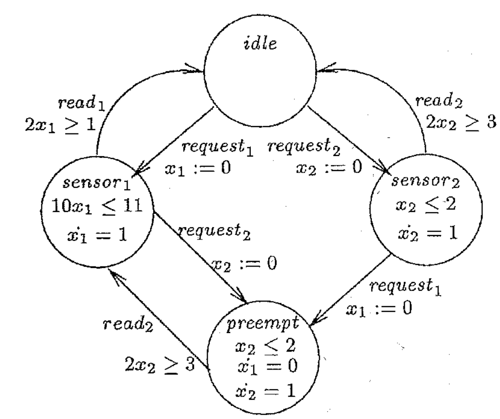


Fig. 10. The scheduler.

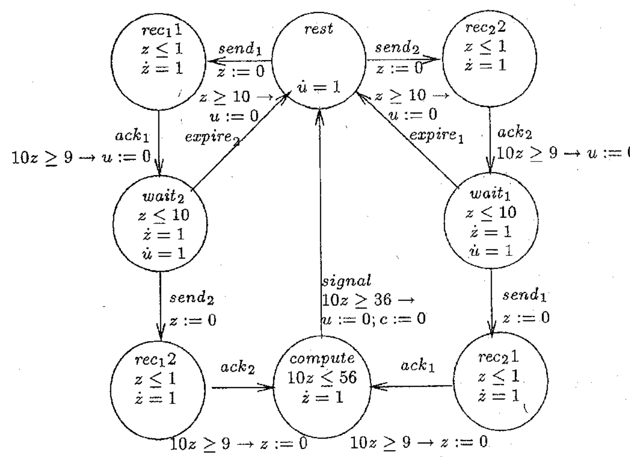


Fig. 11. The controller.

The product of the four automata does not exactly match the specification of [14]. This is because the *send* transitions should be urgent, that is, they should be taken as soon as they are enabled. The urgency of the *send* transitions can be enforced by adding an additional clock, u, and a global invariant as a conjunct to the invariants of all product locations. The clock u is reset whenever a sensor is ready to send a reading to the controller, and whenever the controller is ready to receive a sensor reading. The following global invariant asserts that $u = 0$ if both a sensor and the controller are ready for a transmission:

$$\begin{aligned}&(\ell[1] = wait \wedge \ell[4] = rest \rightarrow u = 0) \wedge \\ &(\ell[2] = wait \wedge \ell[4] = rest \rightarrow u = 0) \wedge \\ &(\ell[1] = wait \wedge \ell[4] = wait_1 \rightarrow u = 0) \wedge \\ &(\ell[2] = wait \wedge \ell[4] = wait_2 \rightarrow u = 0)\end{aligned}$$

This invariant ensures that whenever a transmission of a sensor reading is enabled, then the transmission happens immediately (in HYTECH, global invariants can be specified using the keyword `GlobalInvar`).

We want to know how often a robot command can be generated by the controller. For this purpose, we add to the system a clock c and a parameter a such that c measures the elapsed time since the last robot command was sent, and a represents the maximal value of c. The slope of the clock c is 1 in all product locations, and c is reset to 0 whenever a robot command is sent. The slope of the parameter a is 0 in all product locations. The initial condition ϕ_0 of the system is given by the state predicate

$$\phi_0\colon\ \ell = (done, done, idle, rest) \wedge y_1 = 6 \wedge y_2 = 6 \wedge c = 0 \wedge u = 0.$$

The characteristic predicate of the ICTL-formula

$$\phi_0 \rightarrow \forall\Box(c \leq a)$$

implies that $a \geq 11.2$; that is, a robot command is generated by the controller at least once every 11.2 milliseconds. If we switch the sensor priorities, and give sensor 1 higher priority than sensor 2, then a robot command is generated at least once every 11.0 milliseconds.

There is an alternative way of computing the upper bound on c. We can compute a sequence of predicates ϕ_0, ϕ_1, ... such that for each $i \geq 0$, $[\![\phi_{i+1}]\!]$ contains all states that can be reached from a state in $[\![\phi_i]\!]$ by a single time step or a transition step. The computation of ϕ_{i+1} from ϕ_i uses postcondition operators that are defined similar to the precondition operators of Section 4.2. If ϕ_i is equivalent to ϕ_{i+1}, then the fixpoint computation terminates, and $[\![\phi_i]\!]$ contains all states that appear on trajectories that start from an initial state. The projection of ϕ_i on the variable c gives the desired constraint $c \leq 11.2$. In this computation the parameter a is not required, and thus, the number of variables used is one fewer than necessary for checking the formula $\phi_0 \rightarrow \forall\Box(c \leq a)$. HYTECH supports such a symbolic forward computation using postconditions.

8 Recent Work

Since the submission of this paper, HYTECH has been redesigned and reimplemented [26], [25]. While the original HYTECH prototype described in Section 6 represents regions as logical formulas (state predicates) that are manipulated by MATHEMATICA, the new implementation represents regions as sets of convex polyhedra that are manipulated using a library of standard polyhedral operations [19]. The new verifier checks the safety requirement of the railroad gate controller in 0.72 rather than 74.3 seconds of CPU time (some other performance figures for the new implementation: time-bounded response requirement of the railroad gate controller, 2.7 s; parameter synthesis for the mutual-exclusion protocol, 1.87 s; preemptive two-task scheduler, 1.03 s; reactor temperature control, 1.2 s). The improved efficiency of HYTECH has made possible the analysis of nontrivial applications, such as a distributed audio control protocol used in Philips stereo components [31]. The new HYTECH also offers greatly enhanced functionality, including the possibility to specify urgent transitions, a flexible verification command language for iterating preconditions, postconditions, and Boolean operations on regions, and an error trace generator for the diagnostic analysis of failed verification attempts.

More efficient implementations of the SMC-procedure are possible for the restricted domain of timed automata. The convex data regions of timed automata have a special form, and can be represented by integer matrices and manipulated using standard matrix operations [15]. The SMC-procedure for timed automata has been implemented in this way in the tools KRONOS [16], COSPAN [43], VERITI [42], and UPPAAL [35]. Also, over timed automata, the SMC-procedure is guaranteed to terminate.

The exact analysis of linear hybrid automata using SMC can be prohibitively expensive, and even nonterminating. Furthermore, most control applications do not fall into the class of linear hybrid automata. Therefore much recent effort has gone into the abstract interpretation and approximate analysis of hybrid automata. Abstract-interpretation techniques for the convergence acceleration of the SMC-procedure for linear hybrid automata have been studied and implemented in [30], [23]. The tool POLKA, in particular, complements the exact methods of HYTECH, which may not terminate, with an emphasis on approximate methods that always terminate. The conservative approximation of nonlinear hybrid systems by linear hybrid automata is studied in [21], [29], [44].

The theory of hybrid automata as infinite-state transition systems is developed in [27], [20], [24]. There, verification questions about various classes of hybrid automata are classified according to their hardness, and verification algorithms are presented that, in some cases, are guaranteed to terminate even if the SMC-procedure does not.

Acknowledgments

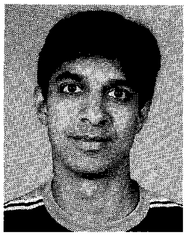
We thank Costas Courcoubetis, Nicolas Halbwachs, Peter Kopke, Joseph Sifakis, and Howard Wong-Toi for helpful discussions and valuable comments.

Thomas A. Henzinger's research was supported in part by the U.S. Office of Naval Research Young Investigator award N00014-95-1-0520, by the National Science Foundation CAREER award CCR-9501708, by National Science

Foundation grants CCR-9200794 and CCR-9504469, by U.S. Air Force Office of Scientific Research contract F49620-93-1-0056, and by Advanced Research Projects Agency grant NAG2-892.

REFERENCES

[1] R. Alur, C. Courcoubetis, and D.L. Dill, "Model Checking in Dense Real Time," *Information and Computation*, vol. 104, no. 1, pp. 2–34, 1993.
[2] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, "The Algorithmic Analysis of Hybrid Systems," *Theoretical Computer Science*, vol. 138, pp. 3–34, 1995.
[3] R. Alur, C. Courcoubetis, T.A. Henzinger, and P.-H. Ho, "Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems," *Hybrid Systems*, R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, eds., *Lecture Notes in Computer Science*, vol. 736, pp. 209–229. Springer-Verlag, 1993.
[4] R. Alur and D.L. Dill, "A Theory of Timed Automata," *Theoretical Computer Science*, vol. 126, pp. 183–235, 1994.
[5] R. Alur, T. Feder, and T.A. Henzinger, "The Benefits of Relaxing Punctuality," *Proc. 10th Ann. Symp. Principles of Distributed Computing*, pp. 139–152. ACM Press, 1991.
[6] R. Alur and T.A. Henzinger, "A Really Temporal Logic," *J. ACM*, vol. 41, no. 1, pp. 181–204, 1994.
[7] R. Alur, T.A. Henzinger, and M.Y. Vardi, "Parametric Real-Time Reasoning," *Proc. 25th Ann. Symp. Theory of Computing*, pp. 592–601. ACM Press, 1993.
[8] *Hybrid Systems III*, R. Alur, T.A. Henzinger, and E. Sontag, eds., *Lecture Notes in Computer Science*. Springer-Verlag, 1996.
[9] *Hybrid Systems II*, P. Antsaklis, A. Nerode, W. Kohn, and S. Sastry, eds., *Lecture Notes in Computer Science*, vol. 999. Springer-Verlag, 1995.
[10] A. Bouajjani, R. Echahed, and J. Sifakis, "On Model Checking for Real-Time Properties with Durations," *Proc. Eighth Ann. Symp. Logic in Computer Science*, pp. 147–159. IEEE CS Press, 1993.
[11] E.M. Clarke and E.A. Emerson, "Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic," *Workshop Logic of Programs, Lecture Notes in Computer Science*, vol. 131. Springer-Verlag, 1981.
[12] E.M. Clarke, E.A. Emerson, and A.P. Sistla, "Automatic Verification of Finite-State Concurrent Systems Using Temporal-Logic Specifications," *ACM Trans. Programming Languages and Systems*, vol. 8, no. 2, pp. 244–263, 1986.
[13] Z. Chaochen, C.A.R. Hoare, and A.P. Ravn, "A Calculus of Durations," *Information Processing Letters*, vol. 40, pp. 269–276, 1991.
[14] J.C. Corbett, "Modeling and Analysis of Real-Time Ada Tasking Programs," *Proc. 15th Ann. Real-time Systems Symp.* IEEE CS Press, 1994.
[15] D.L. Dill, "Timing Assumptions and Verification of Finite-State Concurrent Systems," *CAV 89: Automatic Verification Methods for Finite-state Systems*, J. Sifakis, ed., *Lecture Notes in Computer Science*, vol. 407, pp. 197–212. Springer-Verlag, 1989.
[16] C. Daws and S. Yovine, "Two Examples of Verification of Multirate Timed Automata Using Kronos," *Proc. 16th Ann. Real-time Systems Symp.*, pp. 66-75. IEEE CS Press, 1995.
[17] J. Ferrante and C. Rackoff, "A Decision Procedure for the First-Order Theory of Real Addition with Order," *SIAM J. Computing*, vol. 4, no. 1, pp. 69–76, 1975.
[18] *Hybrid Systems*, R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, eds., *Lecture Notes in Computer Science*, vol. 736. Springer-Verlag, 1993.
[19] N. Halbwachs, "Delay Analysis in Synchronous Programs," *CAV 93: Computer-aided Verification*, C. Courcoubetis, ed., *Lecture Notes in Computer Science*, vol. 697, pp. 333–346. Springer-Verlag, 1993.
[20] T.A. Henzinger, "Hybrid Automata with Finite Bisimulations," *ICALP 95: Automata, Languages, and Programming*, Z. Fülöp and F. Gécseg, eds., *Lecture Notes in Computer Science*, vol. 944, pp. 324–335. Springer-Verlag, 1995.
[21] T.A. Henzinger and P.-H. Ho, "Algorithmic Analysis of Nonlinear Hybrid Systems," *CAV 95: Computer-aided Verification*, P. Wolper, ed., *Lecture Notes in Computer Science*, vol. 939, pp. 225–238. Springer-Verlag, 1995.
[22] T.A. Henzinger and P.-H. Ho, "HYTECH: The Cornell Hybrid Technology Tool," *Hybrid Systems II*, P. Antsaklis, A. Nerode, W. Kohn, and S. Sastry, eds., *Lecture Notes in Computer Science*, vol. 999, pp. 265–293. Springer-Verlag, 1995.
[23] T.A. Henzinger and P.-H. Ho. "A Note on Abstract-Interpretation Strategies for Hybrid Automata," *Hybrid Systems II*, P. Antsaklis, A. Nerode, W. Kohn, and S. Sastry, eds., *Lecture Notes in Computer Science*, vol. 999, pp. 252–264. Springer-Verlag, 1995.
[24] M.R. Henzinger, T.A. Henzinger, and P.W. Kopke, "Computing Simulations on Finite and Infinite Graphs," *Proc. 36th Ann. Symp. Foundations of Computer Science*, pp. 453–462. IEEE CS Press, 1995.
[25] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi, "HYTECH: The Next Generation," *Proc. 16th Ann. Real-time Systems Symp.*, pp. 56–65. IEEE CS Press, 1995.
[26] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi, "A User Guide to HYTECH," *TACAS 95: Tools and Algorithms for the Construction and Analysis of Systems*, E. Brinksma, W.R. Cleaveland, K.G. Larsen, T. Margaria, and B. Steffen, eds., *Lecture Notes in Computer Science*, vol. 1,019, pp. 41–71. Springer-Verlag, 1995. Full version available as Technical Report CSD-TR-95-1532, Cornell Univ.
[27] T.A. Henzinger, P.W. Kopke, A. Puri, and P. Varaiya, "What's Decidable about Hybrid Automata?" *Proc. 27th Ann. Symp. Theory of Computing*, pp. 373–382. ACM Press, 1995.
[28] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, "Symbolic Model Checking for Real-Time Systems," *Information and Computation*, vol. 111, no. 2, pp. 193–244, 1994.
[29] P.-H. Ho, "Automatic Analysis of Hybrid Systems," PhD thesis, Cornell Univ., 1995.
[30] N. Halbwachs, P. Raymond, and Y.-E. Proy, "Verification of Linear Hybrid Systems by Means of Convex Approximation," *SAS 94: Static Analysis Symp.*, B. LeCharlier, ed., *Lecture Notes in Computer Science*, vol. 864, pp. 223–237. Springer-Verlag, 1994.
[31] P.-H. Ho and H. Wong-Toi, "Automated Analysis of an Audio Control Protocol," *CAV 95: Computer-aided Verification*, P. Wolper, ed., *Lecture Notes in Computer Science*, vol. 939, pp. 381–394. Springer-Verlag, 1995.
[32] Y. Kesten, A. Pnueli, J. Sifakis, and S. Yovine, "Integration Graphs: A Class of Decidable Hybrid Systems," *Hybrid Systems*, R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, eds., *Lecture Notes in Computer Science*, vol. 736, pp. 179–208. Springer-Verlag, 1993.
[33] L. Lamport, "A Fast Mutual Exclusion Algorithm," *ACM Trans. Computer Systems*, vol. 5, no. 1, pp. 1–11, 1987.
[34] O. Lichtenstein and A. Pnueli, "Checking that Finite-State Concurrent Programs Satisfy their Linear Specification," *Proc. 12th Ann. Symp. Principles of Programming Languages*, pp. 97–107. ACM Press, 1985.
[35] K.G. Larsen, P. Petterson, and W. Yi, "Compositional and Symbolic Model Checking for Real-Time Ssystems," *Proc. 16th Ann. Real-time Systems Symp.*, pp. 76-87. IEEE CS Press, 1995.
[36] K.L. McMillan, *Symbolic Model Checking: An Approach to the State Explosion Problem*. Kluwer Academic, 1993.
[37] O. Maler, Z. Manna, and A. Pnueli, "From Timed to Hybrid Systems," *Real Time: Theory in Practice*, J.W. de Bakker, K. Huizing, W.-P. de Roever, and G. Rozenberg, eds., *Lecture Notes in Computer Science*, vol. 600, pp. 447–484. Springer-Verlag, 1992.
[38] X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, "An Approach to the Description and Analysis of Hybrid Systems," *Hybrid Systems*, R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, eds., *Lecture Notes in Computer Science*, vol. 736, pp. 149–178. Springer-Verlag, 1993.
[39] X. Nicollin, J. Sifakis, and S. Yovine, "From ATP to Timed Graphs and Hybrid Systems," *Acta Informatica*, vol. 30, pp. 181–202, 1993.
[40] J. Queille and J. Sifakis, "Specification and Verification of Concurrent Systems in CESAR," *Fifth Int'l Symp. Programming*, M. Dezani-Ciancaglini and U. Montanari, eds., *Lecture Notes in Computer Science*, vol. 137, pp. 337–351. Springer-Verlag, 1981.
[41] S. Wolfram, *MATHEMATICA: A System for Doing Mathematics by Computer*. Addison-Wesley, 1988.
[42] H. Wong-Toi, "Symbolic Approximations for Verifying Real-Time Systems," PhD thesis, Stanford Univ., 1994.
[43] R. Alur and R.P. Kurshan, "Timing Analysis with COSPAN," in [8].
[44] T.A. Henzinger and H. Wong-Toi, "Linear Phase-Portrait Approximations for Nonlinear Hybrid Systems," in [8].



Rajeev Alur received the BTech degree in computer science from the Indian Institute of Technology, Kanpur, India, in 1987, and the PhD degree in computer science from Stanford University in 1991. He is a member of the technical staff in the Computing Sciences Research Center of Bell Laboratories at Murray Hill, New Jersey. His areas of research include specification and verification of reactive systems, real-time and hybrid systems, concurrency theory, and distributed algorithms.

Thomas A. Henzinger received the MS degree in computer science from the University of Delaware in 1986, the Dipl.-Ing. In computer science from Kepler University in Linz, Austria, in 1987, and the PhD degree in computer science from Stanford University in 1991. He spent a postdoctoral semester in Grenoble, France. From 1992 to 1995, he was an assistant professor of computer science at Cornell University. In 1996, he joined the Department of Electrical Engineering and Computer Sciences of the University of California at Berkeley.

Pei-Hsin Ho received the BS degree in mathematics from Chung-Yuan Christian University, Taiwan, in 1987, the MS degree in applied mathematics from National Chiao Tung University, Taiwan, in 1989, and the MS and PhD degrees in computer science from Cornell University in 1994 and 1995, respectively. He is currently with Intel Development Labs in Hillsboro, Oregon, working on the formal verification and specification of microprocessors.