Symbolic Analysis of Hybrid Systems*

Rajeev Alur[1] Thomas A. Henzinger[2] Howard Wong-Toi[3]

Abstract

A *hybrid system* is a dynamical system whose behavior exhibits both discrete and continuous change. A *hybrid automaton* is a mathematical model for hybrid systems, which combines, in a single formalism, automaton transitions for capturing discrete change with differential equations for capturing continuous change. In this survey, we demonstrate symbolic algorithms for the verification of and controller synthesis for *linear hybrid automata*, a subclass of hybrid automata that can be analyzed automatically.

1 Introduction

A hybrid system typically consists of a collection of digital programs that interact with each other and with an analog environment. Examples of hybrid systems include manufacturing controllers, automotive and flight controllers, medical equipment, microelectromechanical systems, and robots. When these systems occur in safety-critical applications, formal guarantees about the absence of logical and timing errors are desirable. The formal analysis of the mixed digital-analog nature of hybrid systems requires a mathematical model that incorporates the discrete behavior of computer programs with the continuous behavior of environment variables, such as time, position, and temperature. The model of our choice is the *hybrid automaton*—a finite automaton augmented with a finite number of real-valued variables that change continuously, as specified by differential equations and differential inequalities [2].

For analyzing hybrid systems, we build on the model-checking technology, in which a formal model of the system is checked, fully automatically, for correctness with respect to a requirement expressed in temporal logic [7, 12]. Model checking requires exploration of the entire state space of the system. For discrete finite-state systems, this can be done enumeratively, by considering each state individually, or symbolically, by computing with constraints over boolean variables that encode state sets. Because of its ability to deal with very large state spaces, symbolic model checking has been proven an effective technique for debugging of complex hardware [6]. For hybrid systems, the state space is infinite, so an enumerative approach is impossible, but the symbolic approach can be extended to a class of hybrid automata called *linear hybrid automata* by admitting linear constraints on continuous variables, namely, disjunctions of inequalities of the form $\mathbf{A}\mathbf{x} \sim \mathbf{c}$, where $\mathbf{A}$ is a constant matrix, $\mathbf{c}$ is a constant vector, and $\sim \in \{\leq, \geq\}$ is an inequality operator [4]. In a linear hybrid automaton, the dynamics of the continuous variables are defined by linear differential inequalities of the form $\mathbf{A}\dot{\mathbf{x}} \sim \mathbf{c}$, where $\dot{\mathbf{x}}$ is the vector of first derivatives of the variables $\mathbf{x}$.[1] Even though termination of the model checking procedure is not guaranteed—the model checking problem for linear hybrid automata is undecidable—the method is still of practical interest, because termination happens naturally in many examples and can be enforced in others, say, by considering the behavior of a system over a bounded interval of time. The procedure is implemented in the verifier HyTech [9].

Model checking can be used to provide more than a mere "yes" or "no" answer to the question of whether a system satisfies a correctness requirement. HyTech provides also diagnostic information that aids in design and debugging. If a system fails to satisfy a correctness requirement, then HyTech generates an error trajectory, which illustrates a time-stamped sequence of events that leads to a violation of the requirement. High-level system descriptions often use *design parameters*—symbolic constants with unknown, fixed values. The goal of parametric analysis is to determine necessary and sufficient constraints on the parameters under which safety violations cannot occur. HyTech can perform such analysis for linear hybrid automata.

Symbolic analysis methods can be applied to the problem of generating controller designs in addition to verifying existing designs [11, 13]. This approach provides a methodology to synthesize provably correct controllers, thereby avoiding the need for verification. The plant is modeled as a linear hybrid automaton whose

*This research was supported in part by the ONR YIP award N00014-95-1-0520, by the NSF CAREER award CCR-9501708, by the NSF grant CCR-9504469, by the AFOSR contract F49620-93-1-0056, by the ARO MURI grant DAAH-04-96-1-0341, by the ARPA grant NAG2-892, and by the SRC contract 95-DC-324.036.

[1] CIS Dept, Univ. of Pennsylvania, Philadelphia, PA 19104, alur@cis.upenn.edu.

[2] EECS Dept, Univ. of California, Berkeley, CA 94720, tah@eecs.berkeley.edu.

[3] Cadence Berkeley Labs, 2001 Addison Street, Third Fl., Berkeley, CA 94704, howard@cadence.com.

[1]It is important to realize that the definition of linearity for hybrid automata differs from the definition of linearity commonly used in systems theory. In particular, the differential inequalities of linear hybrid automata may not depend on the value of the variables, and thus dynamics of the form $\dot{x} = x$ are prohibited.

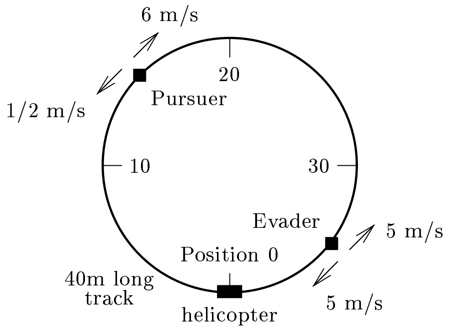

Figure 1: Pursuit game

mode-switches are partitioned into controllable and uncontrollable. Then, control for a given safety requirement can be formulated as a two-player game, which can be solved using a symbolic fixpoint computation procedure. This procedure, if it terminates, automatically generates a finite controller such that the closed loop system is correct with respect to the requirement.

The remaining paper consists of three sections. Section 2 presents the model of hybrid automata, Section 3 illustrates the analysis techniques, A simple pursuit game is used as a running example to demonstrate modeling, safety verification, parametric analysis, and controller synthesis. Section 4 gives pointers for further reading.

2 Linear Hybrid Automata

2.1 Example: a simple pursuit game

A linear hybrid automaton consists of a control graph with conditions on real-valued variables. Each node of the graph represents an operating mode of the system, and is annotated with differential inequalities that prescribe the possible evolutions (flows) of the real variables while the system remains in the given mode. Each node is also annotated with an invariant condition: while control of the linear hybrid automaton remains in a node, the variables must satisfy the node's invariant condition. Each edge of the graph represents a switch in operating mode, and is annotated with a condition that prescribes the possible changes (jumps) of the real variables when the system executes the given mode switch.

We use a simple two-person game of pursuit as a running example. There is a pursuer in a golf cart chasing an evader on a circular track 40 meters long. The cart can travel up to 6 meters per second in the clockwise direction, but only up to 1/2 meters per second going counterclockwise, since it must use its reverse gear to travel in this direction. The evader is on a bicycle, and travels at 5 meters per second in either direction. However, it makes a decision whether to change its direction only at fixed points in time, separated by exactly two

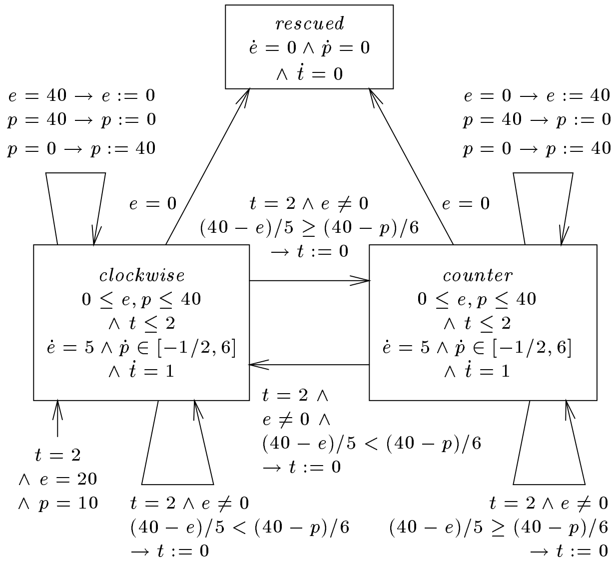

Figure 2: Automaton for pursuit game

seconds. The goal of the evader is to avoid the pursuer. The evader has the added advantage that there is a rescue helicopter at a fixed position on the track. The game scenario is depicted in Figure 1. The evader uses a very simple strategy. It determines whether it will win the race to the helicopter if both parties proceed clockwise at full speed. If so, it heads clockwise, otherwise it goes counterclockwise. The game, together with the evader's strategy, is modeled by the linear hybrid automaton in Figure 2. The variable p models the position of the pursuer on the track, measured in meters in a clockwise direction relative to the helicopter at position 0. The evader's position is modeled using the variable e. The clock t measures the delay between the choices made by the evader. There are three operating modes for the evader: going clockwise, modeled by the mode labeled *clockwise*, going counterclockwise, modeled by the mode labeled *counter*, and a rescued mode, labeled *rescued*. The variables evolve in mode *clockwise* according to the differential equations $\dot e = 5$ and $\dot t = 1$, and the differential inequality $\dot p \in [-1/2, 6]$. The invariant conditions $0 \leq e, p \leq 40$ and the mode switches appearing as selfloops above the modes reflect the fact that the track is circular and 40 meters long. All equalities and inequalities are over arithmetic modulo 40, and thus $e \neq 0$ represents $e \neq 0 \wedge e \neq 40$. The switches to the mode *rescued* model the evader's successful escape by helicopter. The other switches model the decisions of the evader. These switches are only enabled when the clock reaches the value 2. The time for the evader (*resp.* pursuer) to reach the helicopter is calculated as $(40 - e)/5$ (*resp.* $(40 - p)/6$). The evader is initially at position 20 directly opposite the track from the helicopter, and the pursuer between them at position 10.

2.2 Formal definition

An *atomic linear predicate* is an inequality between a rational constant and a linear combination of variables with rational coefficients, such as $3x_1 - x_2 + 7x_5 \leq 3/4$. A *convex linear predicate* is a finite conjunction of linear inequalities. A *linear predicate* is a finite disjunction of convex linear predicates. A *linear hybrid automaton* is a system $A = (X, V, \mathit{flow}, \mathit{inv}, \mathit{init}, E, \mathit{jump}, \Sigma, \mathit{syn})$ that consists of the following components [2]:

Variables A finite ordered set $X = \{x_1, x_2, \ldots, x_n\}$ of real-valued variables.

Control modes A finite set V of control modes.

Flow conditions A labeling function *flow* that assigns a flow condition to each control mode $v \in V$. The flow condition $\mathit{flow}(v)$ is a convex linear predicate over the variables $\dot{X}$, where $\dot{X} = \{\dot{x}_1, \ldots, \dot{x}_n\}$. The dotted variable $\dot{x}_i$, for $1 \leq i \leq n$, refers to the first derivative of x_i with respect to time. While the control of the hybrid automaton A is in mode v, the variables in X evolve along a differentiable curve such that at all points along the curve, the first derivatives of all variables satisfy the flow condition $\mathit{flow}(v)$.

Invariant conditions A labeling function *inv* that assigns an invariant condition to each control mode $v \in V$. The invariant condition $\mathit{inv}(v)$ is a convex linear predicate over the variables in X.

Initial conditions A labeling function *init* that assigns an initial condition to each control mode $v \in V$. The initial condition $\mathit{init}(v)$ is a convex linear predicate over the variables in X. The control of the hybrid automaton A may start in the control mode v when the initial condition $\mathit{init}(v)$ is true. In the graphical representation of automata, initial conditions appear as labels on incoming arrows without source modes, and initial conditions of the form *false* are not depicted.

Control switches A finite multiset E of control switches. Each control switch (v, v') is a directed edge between a source mode $v \in V$ and a target mode $v' \in V$.

Jump conditions A labeling function *jump* that assigns a jump condition to each control switch $e \in E$. The jump condition $\mathit{jump}(e)$ is a linear predicate over the variables in $X \cup X'$, where $X' = \{x_1', \ldots, x_n'\}$. The unprimed symbol x_i, for $1 \leq i \leq n$, refers to the value of the variable x_i before the control switch, and the primed symbol x_i' refers to the value of x_i after the control switch. Thus, a jump condition relates the values of the variables before a control switch to the possible values after the control switch. In the graphical representation of automata, we use guarded assignments to represent jump conditions; for example, assuming $n = 2$, the guarded assignment $x_1 = x_2 \rightarrow x_1 := 2x_2$ stands for the jump condition $x_1 = x_2 \wedge x_1' = 2x_2 \wedge x_2' = x_2$.

Events A finite set Σ of events, and a labeling function *syn* that assigns an event in Σ to each control switch $e \in E$. Though not used in our pursuit game, events permit the synchronization of jumps between concurrent hybrid automata.

2.3 States and trajectories

A *state* of the hybrid automaton A is a pair $(v, \mathbf{a})$ consisting of a control mode $v \in V$ and a vector $\mathbf{a} = (a_1, \ldots, a_n)$ that represents a value $a_i \in \mathbb{R}$ for each variable $x_i \in X$. The state $(v, \mathbf{a})$ of A is *admissible* if the predicate $\mathit{inv}(v)$ is true when each variable x_i is replaced by the value a_i. The state $(v, \mathbf{a})$ is *initial* if the predicate $\mathit{init}(v)$ is true when each x_i is replaced by a_i.

Consider a pair (q, q') of two admissible states $q = (v, \mathbf{a})$ and $q' = (v', \mathbf{a}')$. The pair (q, q') is a *jump* of A if there is a control switch $e \in E$ with source mode v and target mode v' such that the predicate $\mathit{jump}(e)$ is true when each variable x_i is replaced by the value a_i, and each primed variable x_i' is replaced by the value a_i'. We say that the switch e *witnesses* the jump. The pair (q, q') is a *flow* of A if $v = v'$ and either (a) $\mathbf{a} = \mathbf{a}'$, or (b) there exists a nonnegative real $\delta \in \mathbb{R}_{\geq 0}$ such that $\mathit{flow}(v)$ is true when each variable $\dot{x}_i$ is replaced by the value $(a_i' - a_i)/\delta$. If (q, q') is a jump, we say that q' is a jump successor of q; if (q, q') is a flow, then q' is called a flow successor of q (notice that every admissible state is a flow successor of itself, because there is always a flow of duration 0).

A *trajectory* of the hybrid automaton A is a finite sequence $q_0, q_1, \ldots, q_k$ of admissible states q_j such that (1) the first state q_0 of the sequence is an initial state of A, and (2) each pair $(q_j, q_j + 1)$ of consecutive states in the sequence is either a jump of A or a flow of A. A state of A is *reachable* if it is the last state of some trajectory of A.

2.4 Safety requirements

A safety requirement asserts that nothing bad will happen during the evolution of a system. Safety requirements can often be specified by describing the "unsafe" values and value combinations of the system variables. Then, the system satisfies the safety requirement iff all reachable states are safe. Safety verification, therefore, amounts to computing the set of reachable states.

For hybrid automata, we specify safety requirements using state assertions. A *state assertion* φ for the hybrid automaton A is a function that assigns to each control mode $v \in V$ a predicate $\varphi(v)$ over the variables in X. We say that the state assertion φ is true (or false) for a state $(v, \mathbf{a})$ of A if the predicate $\varphi(v)$ is true (false) when each variable x_i is replaced by the value a_i. The states for which φ is true are called the φ-states. If *unsafe* is a state assertion for the hybrid automaton A, then A *satisfies the safety requirement* specified by *unsafe* if the state assertion *unsafe* is false for all reachable states of A. For example, the unsafe states of the evader automaton (Figure 2) are those for which $e = p$.

3 Analysis of Linear Hybrid Automata

3.1 Computing the reachable states

To check if the hybrid automaton A satisfies the safety requirement specified by the state assertion *unsafe*, we attempt to compute another state assertion, *reach*, which is true exactly for the reachable states of A. If there is any state for which both *reach* and *unsafe* are true, then the safety requirement is violated, and we produce an error trajectory from an initial state to an unsafe state; if not, the safety requirement is satisfied. We attempt to compute the state assertion *reach* as follows. For a state assertion φ, let $Post(\varphi)$ be a state assertion that is true precisely for the jump and flow successors of the φ-states, *i.e.* $Post(\varphi)$ is true for a state q' iff there exists a φ-state q such that (q, q') is either a jump or a flow of A. The state assertion $\varphi_1 = Post(init)$ characterizes all states that are reachable by trajectories of length 1 (*i.e.* by a single jump or flow); the state assertion $\varphi_2 = Post(\varphi_1)$ characterizes all states that are reachable by trajectories of length 2; etc. Finally, if for some natural number k, we find that φ_k and $\varphi_{k+1} = Post(\varphi_k)$ are equivalent, then we can conclude that φ_k characterizes all states that are reachable by trajectories of *any* length, and therefore $reach = \varphi_k$.

A state assertion φ is *linear* if for every control mode $v \in X$, the predicate $\varphi(v)$ is linear. If A is a linear hybrid automaton, and φ a state assertion for A, then $Post(A)$ is computable, and is itself a linear state assertion [4]. This theorem enables the efficient automatic analysis of safety requirements and more general temporal-logic requirements for linear hybrid automata.

3.2 Safety verification of the pursuit game

We wish to verify that the evader's strategy is a winning strategy for the given initial position. The unsafe states are specified by the state assertion *unsafe* that assigns the predicate $e = p$ to the control modes *clockwise* and *counter*. The computation of the reachable states starts from the state assertion [2]

$$\begin{aligned}\varphi_0 \;=\; init \;=\; &\{(clockwise, e = 20 \wedge p = 10 \wedge t = 2),\\ &(counter, false), (rescued, false)\}.\end{aligned}$$

We compute the state assertion $\varphi_1 = Post(\varphi_0)$ in two steps. First, we find that all jump successors of φ_0-states are those states for which the state assertion

$$\begin{aligned}&\{(clockwise, e = 20 \wedge p = 10 \wedge t = 0),\\ &(counter, false), (rescued, false)\}\end{aligned}$$

is true. This is because the only mode switch that is enabled is the selfloop at the bottom of mode *clockwise*, since $(40 - 20)/5 < (40 - 10)/6$ and $t = 2$. The values of e and p are unaltered by the switch, while t is reset to 0. Second, we find that the only flow successors of the φ_0-states are the φ_0-states themselves since the invariant $t \leq 2$ together with the differential equation $\dot{t} = 1$ and the fact that t has value 2 implies that no further time can evolve in this mode. Thus we obtain the state assertion

$$\begin{aligned}\varphi_1 &= Post(\varphi_0)\\ &= \{(clockwise, e = 20 \wedge p = 10 \wedge (t = 0 \vee t = 2)),\\ &\quad (counter, false), (rescued, false)\}.\end{aligned}$$

For computing φ_2, there are no new jump successors since there is no control switch for which the guard is enabled when $t = 0$. The addition of all flow successors for mode *clockwise* results in

$$\begin{aligned}\varphi_2 &= Post(\varphi_1)\\ &= \{(clockwise, \quad e = 20 \wedge p = 10 \wedge t = 2\\ &\qquad\qquad \vee 0 \leq t \leq 2 \wedge e = 20 + 5t\\ &\qquad\qquad \wedge 10 - t/2 \leq p \leq 10 + 6t),\\ &\quad (counter, false), (rescued, false)\}.\end{aligned}$$

After two more iterations, we obtain

$$\begin{aligned}\varphi_5 &= Post(\varphi_4)\\ &= \{(clockwise, \quad e = 20 \wedge p = 10 \wedge t = 2\\ &\qquad\qquad \vee 0 \leq t \leq 2 \wedge e = 20 + 5t\\ &\qquad\qquad \wedge 10 - t/2 \leq p \leq 10 + 6t,\\ &\qquad\qquad \vee 0 \leq t \leq 2 \wedge e = 30 + 5t\\ &\qquad\qquad \wedge 9 - t/2 \leq p \leq 22 + 6t),\\ &\quad (counter, false),\\ &\quad (rescued, t = 2 \wedge e = 0 \wedge 8 \leq p \leq 34)\}.\end{aligned}$$

Since $Post(\varphi_5) = \varphi_5$, we conclude that $\varphi_5 = reach$. HyTech performs these computations for us, fully automatically, and determines that no reachable state is unsafe.

3.3 Parametric analysis

In a linear hybrid automaton A, a design parameter α can be represented as a variable whose value never changes, *i.e.* all flow conditions must imply $\dot{\alpha} = 0$ and all jump conditions must imply $\alpha' = \alpha$. Then, in all states of a trajectory of A, the parameter α has the same value (but the value of α may differ from trajectory to trajectory). The value $a \in \mathbb{R}$ is called *safe for* α if whenever we add the conjunct $\alpha = a$ to all initial conditions of A, then no unsafe state is reachable. This is the case precisely when there is no trajectory of A such that (1) the last state of the trajectory is unsafe, and (2) the parameter α has

[2] We write $\{(clockwise, p_1), (counter, p_2), (rescued, p_3)\}$ for the state assertion that assigns p_1 to the control mode *clockwise*, p_2 to *counter*, and p_3 to *rescued*.

the value a in the last state. Thus, the predicate $\exists X \setminus \{\alpha\}. \bigvee_{v \in V}(\mathit{reach}(v) \wedge \mathit{unsafe}(v))$ is a predicate over the variable α which is true precisely for the unsafe values for α. If *reach* and *unsafe* are linear state assertions, then the existential quantifier can be eliminated effectively, and we obtain, by negation, a linear predicate that characterizes exactly the safe values for the parameter α.

For the given evader strategy, we can use parametric analysis to determine the exact set of initial positions for the pursuer for which the evader can win the game. For this purpose, we introduce a parameter α to represent the initial position of the pursuer. The unsafe states are as the same as before. We then let HyTech compute the values of α for which an unsafe state is reachable. HyTech generates the condition $0 \le \alpha \le 2 \wedge 16 \le \alpha \le 40$, implying that the evader wins precisely when the pursuer starts at a position strictly between 2 and 16.

3.4 Controller synthesis

To formulate the controller synthesis problem, we must first embellish our definition of linear hybrid automata to enable a means of control. We let the event set Σ of the automaton be partitioned into a set Σ_c of controllable events and a set Σ_u of uncontrollable events. Mode switches labeled with controllable events are called controllable; they can occur only when the controller designates. Mode switches labeled with uncontrollable events are called uncontrollable; they may occur whenever they are enabled. Intuitively a controller continually observes the state of the plant and chooses at any time to force a controllable mode switch to occur, provided it is enabled, or to let time pass. A controller for a linear hybrid automaton A is a function f from the states of A to $\Sigma_c \cup \{\perp\}$, where the symbol $\perp$ represents a null control action. Let $q = (v, \mathbf{a})$ and $q' = (v', \mathbf{a}')$ be two admissible states. The pair (q, q') is a *controlled jump* if $f(q) = \sigma$ and there exists a control switch $e \in E$ with event label $\sigma \in \Sigma_c$ that witnesses the jump (q, q'). The pair (q, q') is an *uncontrollable jump* if there is a control switch with event label in Σ_u that witnesses the jump. The pair (q, q') is a *controlled flow* if it is a flow and $f(v, \hat{\mathbf{a}}) = \perp$ for all $\hat{\mathbf{a}}$ on the straight line between the points $\mathbf{a}$ and $\mathbf{a}'$, excluding possibly $\mathbf{a}'$. A *controlled trajectory* of a linear hybrid automaton A and the controller f is a finite sequence $q_0, q_1, \ldots, q_k$ of admissible states such that q_0 is initial and each pair of consecutive states is either a controlled jump, an uncontrollable jump, or a controlled flow.

Given a linear hybrid automaton A and a safety requirement *unsafe*, the controller synthesis problem is to find a controller such that there are no controlled trajectories for which the last state is an *unsafe*-state. Starting from the state assertion $\varphi_0 = \mathit{unsafe}$, we compute a state assertion $\varphi_1 = \mathit{Inevitable}(\varphi_0)$ for the states from which it is impossible for any controller to avoid

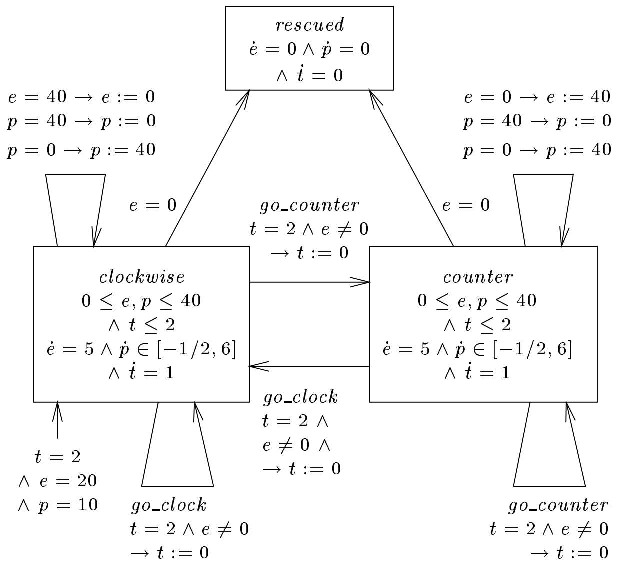

Figure 3: Evader automaton for control

entering an *unsafe*-state within at most a single uncontrollable jump or a controlled flow, *i.e.* for every φ_1-state q, there exists a φ_0-state q' such that either (1) (q, q') is an uncontrollable jump, or (2) (q, q') is a flow of A such that for all q'' along the witness for (q, q'), if there exists a jump successor of q'' for a mode switch labeled with $\sigma_c \in \Sigma_c$, then there exists a φ_0-state that is a jump successor of q'' for a mode switch labeled with σ_c. From φ_1, we compute the state assertion φ_2 for the states from which it is impossible to avoid entering a φ_1 state within at most a single uncontrolled jump, controlled jump, or a controlled flow. The algorithm continues to compute the state assertions φ_k, similarly defined. If, for any k, $\varphi_k = \varphi_{k+1}$, then the algorithm terminates, and the φ_k-states are precisely those states for which there exists no controller that meets the safety requirement.

The control strategy given for the evader in the pursuit example is non-optimal. Recall that from a starting position of 20, the evader wins iff the pursuer's initial position lies in the range $(2, 16)$. The linear hybrid automaton of Figure 3 is a model of the plant representing the evader's potential moves. The controllable events are *go_clock* for moving counterclockwise and *go_counter* for moving counterclockwise. The synthesis procedure generates a winning strategy for all starting positions of the pursuer other than those in the range $[16, 20]$. Intuitively, the strategy is to head for the helicopter in a clockwise direction unless either the pursuer can intercept the evader immediately or going continually counterclockwise will guarantee reaching the helicopter successfully. The evader however is not assured of reaching the helicopter. It may move back and forth along the track continually. For example, if the pursuer remains at position 30, then the evader must shuttle back and forth between positions 10 and 20; it may never progress to the helicopter itself, for fear that the pursuer will meet it just before, and it may never move to position 30, since the pursuer might be waiting there.

We illustrate the beginning of the synthesis procedure. The procedure starts with φ_0 :

$$\{(\mathit{clockwise}, e = p \bmod 40), (\mathit{counter}, e = p \bmod 40), (\mathit{rescued}, \mathit{false})\}.$$

To compute $\mathit{Inevitable}(\varphi_0)$, we must consider the uncontrollable jumps and the controlled flows leading into φ_0-states. Since the only mode switches with uncontrollable events either (a) have target mode *rescued* to which φ_0 assigns the predicate *false*, or (b) are self-loops which do not affect the truth of the predicate $e = p \bmod 40$, we need only consider flows. The desired φ_1 is

$$\begin{aligned}\{(&\mathit{clockwise}, 0 \le t \le 2 \wedge 0 \le e, p \le 40 \\ &\qquad \wedge ((p + 40 - e) \bmod 40 \le \tfrac{11}{2}(2 - t) \\ &\qquad\quad \vee (e + 40 - p) \bmod 40 \le 2 - t)), \\ \{(&\mathit{counter}, 0 \le t \le 2 \wedge 0 \le e, p \le 40 \\ &\qquad \wedge (e + 40 - p) \bmod 40 \le 11(2 - t)), \\ (&\mathit{rescued}, \mathit{false})\}.\end{aligned}$$

4 Discussion

We refer the reader to [9] for more information on HyTech and its applications, and to [8] for a survey of theoretical results. Here, we conclude with some noteworthy aspects of symbolic analysis.

Timed Automata. A timed automaton [3] is a linear hybrid automaton in which all continuous variables change at the constant rate 1. Model checking of temporal requirements of a timed automaton, unlike a linear hybrid automaton, is decidable [1]. Furthermore, symbolic analysis of a timed automaton requires the manipulation of a special class of linear constraints, namely, disjunctions of inequalities of the form $\mathbf{x} \sim \mathbf{c}$ and $\mathbf{x} - \mathbf{y} \sim \mathbf{c}$, for clock vectors $\mathbf{x}$ and $\mathbf{y}$ [10]. Consequently, the analysis can be performed using more efficient representations.

Linear Approximations. While linear hybrid automata are expressive compared to other formalisms for which model checking is possible, such as finite automata and timed automata, many embedded applications do not meet the linearity constraints. In such cases, we have to conservatively approximate the system using linear hybrid automata so that if the approximate automaton satisfies a correctness requirement, then the original system satisfies the requirement as well. If, on the other hand, the approximate system violates the requirement, and the generated error trajectory is not a possible trajectory of the original system, then the approximation must be refined.

Divergence. To establish a liveness requirement, e.g, the evader will eventually reach the helicopter, we need to consider *infinite* trajectories of the system. However, we wish to exclude unrealistic trajectories along which time does not diverge. For instance, for the evader automaton, we do not want to consider the trajectory in which the control stays at the node *clockwise*, with no jumps and infinitely many flows whose durations form a converging sequence. In controller synthesis, this issue arises even for safety requirements: we do not want a controller which would avoid the unsafe states by forcing infinitely many controlled switches within a finite duration of time. The interested reader should consult [13, 5] for treatment of the divergence problem.

References

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

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

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

[4] R. Alur, T. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering*, 22(3):181–201, 1996.

[5] R. Alur and T. Henzinger. Modularity for timed and hybrid systems. In *Proc. Ninth CONCUR*, LNCS 1243, pp. 74–88, 1997.

[6] J. Burch, E. Clarke, K. McMillan, D. Dill, and L. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–70, 1992.

[7] E. Clarke and E. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Workshop on Logic of Programs*, LNCS 131, 1981.

[8] T. Henzinger. The theory of hybrid automata. In *Proc. 11th LICS*, pp. 278–292, 1996.

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

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

[11] O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In *STACS 95*, LNCS 900, pp. 229–242, 1995.

[12] J. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Fifth Intl. Symp. on Programming*, LNCS 137, pp. 337–351, 1981.

[13] H. Wong-Toi. The synthesis of controllers for linear hybrid automata. In *Proc. 36th CDC*, 1997.