# An Introduction to Hybrid Automata

Jean-François Raskin, email: jraskin@ulb.ac.be

Computer Science Department
University of Brussels
Belgium

## 1 Introduction

Hybrid systems are digital real-time systems embedded in analog environments. A paradigmatic example of a hybrid system is a digital embedded control program for an analog plant environment, like a furnace or an airplane: the controller state moves discretely between control modes, and in each control mode, the plant state evolves continuously according to physical laws. Those systems combine discrete and continuous dynamics. Those aspects have been studied in computer science and in control theory. Computer scientists have introduced *hybrid automata* [Hen00], a formal model that combines discrete control graphs, usually called *finite state automata*, with continuously evolving variables. A hybrid automaton exhibits two kinds of state changes: *discrete jump transitions* occur instantaneously, and *continuous flow transitions* occur when time elapses.

Hybrid systems are often systems that are safety critical. As a consequence, their reliability is a central issue. For example, the correctness of a digital controller that monitors the temperature of a nuclear reactor is crucial. We present hybrid automata as formal models that define *trajectories* (behaviors) of hybrid systems. Properties of a hybrid system assign values to its trajectories: for example, they can classify trajectories as good or bad. The behaviors of a hybrid automaton are often complex, and it may thus be difficult to reason about them. This is why, since the early works on hybrid automata, the emphasis has been on their *computer aided analysis*. Model-checking methods [CGP99] have been studied extensively and tools able to analyze complex hybrid systems have been developed.

This chapter is organized as follows. First, we introduce the *syntax* and *semantics* of hybrid automata, and show how complex hybrid systems can be modeled *compositionally* as *products of* hybrid automata. Then, we define *safety properties* of hybrid automata and show how to model them using *monitors*. We show that the *verification* of those properties reduces naturally to reachability problems, that is, to decide if there exists a trajectory of the hybrid system that reaches a given set of states. As hybrid automata can be very complex mathematical objects, restricted subclasses for which we have *automatic analysis methods* have been introduced. In this introduction, we focus on *rectangular hybrid automata* and show how they can be used to *over-approximate* the behavior of more complex hybrid automata. We close the chapter by referencing the literature to allow the reader into go deeper in this flourishing research subject.

## 2 Hybrid Automata: A Model for Hybrid Systems

To illustrate the main notions about hybrid automata, we use a running example throughout the chapter. The components of the running example are depicted in Fig. 1. It shows a system composed of three devices: (*i*) a tank that contains water and that can be heated using a gas burner, (*ii*) a gas burner that can be *turned on* or *turned off*, and (*iii*) a thermometer that monitors the temperature of the water inside the tank and periodically issues signals when the temperature of the water in the tank is above or below certain thresholds. Later, we will add to this system a *controller* that will observe the signals issued by the thermometer and will issue orders to the gas burner in order to maintain the temperature of the water within a given range.

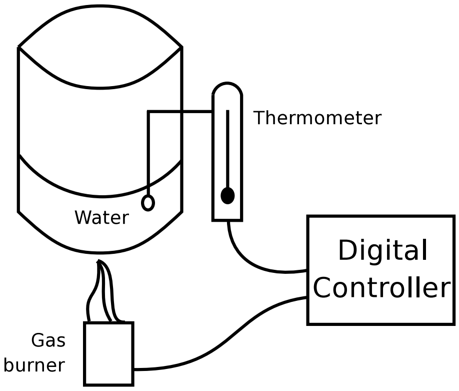


**Fig. 1.** Our running example

We first describe in detail the behavior of the temperature of the water in the tank. When the gas burner is OFF, the temperature of the water, denoted by the variable $x$, decreases according to the following exponential function: $x(t) = Ie^{-Kt}$ where $I$ is the initial temperature of the water, $K$ is a constant that depends on the nature of the tank (how much it conducts heat for example), and $t$ denotes time. However, this law is only true when the temperature of the water is greater than 20 degrees, the temperature of the room where the tank is located. When the heater is OFF and the temperature of the water is 20 degrees, then the temperature stays constant. On the other hand, when the gas burner is ON, the temperature of the water increases according to the following exponential function $x(t) = Ie^{-Kt} + h(1 - e^{-Kt})$ where $I$, $K$, and $t$ are as before and $h$ is a constant that depends on the power of the gas burner. Again, this rule is only true if the water in the tank has a temperature that is less than or equal to 100 degrees. When the temperature of the water reaches 100 degrees, it stays constant (the pressure increases but we omit that in our model). Fig. 2 shows a fragment of a possible evolution of the temperature of the water within the tank.

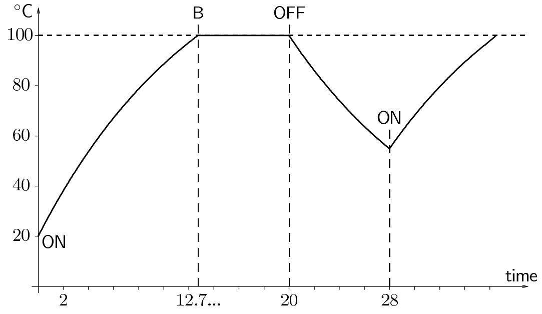


**Fig. 2.** One possible behavior of the tank

As we can see from the description of the evolution of the temperature in the tank, the system is not purely continuous. The evolution of the temperature depends on the mode of the system (the burner is ON or OFF, the temperature is below or above 100 degrees), and the system can switch discretely from one mode to another (if the burner is turned off, for example). Therefore, a natural model for such a system should mix continuous evolutions with discrete switches. Hybrid automata are well suited to describe such complex mixed discrete-continuous behaviors. Their syntax is defined in the next subsection.

### 2.1 Syntax

A hybrid automaton is a *generalized finite-state automaton* that is equipped with continuous variables. The *discrete changes* of the hybrid system are modeled by *edges* of the automaton, and the *continous evolutions* of the hybrid system are modeled by *differential equations* that label locations of the automaton. The syntax of hybrid automata is defined as follows.

**Definition 1** [Hybrid Automaton] A *hybrid automaton* $H$ is a tuple $\langle \mathsf{Loc}, \mathsf{Edge}, \Sigma, X, \mathsf{Init}, \mathsf{Inv}, \mathsf{Flow}, \mathsf{Jump} \rangle$ where:

- $\mathsf{Loc}$ is a finite set $\{l_1, l_2, \dots l_n\}$ of (control) locations that represent *control modes* of the hybrid system.
- $\Sigma$ is a finite set of *event names*.
- $\mathsf{Edge} \subseteq \mathsf{Loc} \times \Sigma \times \mathsf{Loc}$ is a finite set of *labelled edges* that represent *discrete changes* of control mode in the hybrid system. Those changes are labelled by *event names* taken from the finite set of labels $\Sigma$.
- $X$ is a finite set $\{x_1, x_2, \dots, x_m\}$ of real-valued variables. We write $\dot{X}$ for the set of dotted variables $\{\dot{x}_1, \dot{x}_2, \dots, \dot{x}_m\}$ which are used to represent first derivatives of the variables during continuous evolutions (inside a mode), and we write $X'$ for the primed variables $\{x'_1, x'_2, \dots, x'_m\}$ that are used to represent updates at the conclusion of discrete changes (from one control mode to another).
- $\mathsf{Init}, \mathsf{Inv}, \mathsf{Flow}$ are functions that assign three predicates to each location. $\mathsf{Init}(l)$ is a predicate whose free variables are from $X$ and which states the possible valuations for those variables when the hybrid system starts from location $l$. $\mathsf{Inv}(l)$ is a predicate whose free variables are from $X$ and which constrains the possible valuations for those variables when the control of the hybrid system is in location $l$. $\mathsf{Flow}(l)$ is a predicate whose free variables are from $X \cup \dot{X}$ and which states the possible continuous evolutions when the control of the hybrid system is in location $l$.
- $\mathsf{Jump}$ is a function that assigns to each labelled edge a predicate whose free variables are from $X \cup X'$. $\mathsf{Jump}(e)$ states when the discrete change modeled by $e$ is possible and what the possible updates of the variables are when the hybrid system makes the discrete change.

The evolution of the temperature of the water in the tank is modeled using the hybrid automaton of Fig. 3. Locations are drawn as boxes with rounded corners and edges as arrows. Locations are named $t_1$ to $t_4$. A predicate next to a location denotes an invariant predicate. Invariant predicates equivalent to *true* are omitted. A predicate next to a location within a box denotes an initial predicate. Initial predicates equivalent to *false* are omitted. Predicates inside locations denote flow predicates. Edges are labelled by event names and update predicates. The hybrid automaton is composed of four locations that model the four different modes of evolution of the temperature within the tank as described above. Variable $x$ is used to model the temperature of the water in the tank.

Location $t_1$ models the behavior of the system when the temperature of the water is between 20 and 100 degrees as indicated by the invariant $20 \leq x \leq 100$ and the gas burner is $\mathsf{ON}$. In that case, the dynamics that govern the variable $x$ is given by the flow predicate $\dot{x} = K(h - x)$. Location $t_2$ models the behavior of the system when the temperature of the water has reached 100 degrees. In that location, the flow predicate $\dot{x} = 0$ models the fact that the water temperature stays constant. Location $t_3$ models the tank system when the heater is $\mathsf{OFF}$ and the temperature of the water is above 20 degrees. In that case, the flow predicate that governs the evolution of $x$ is $\dot{x} = -Kx$. Finally,

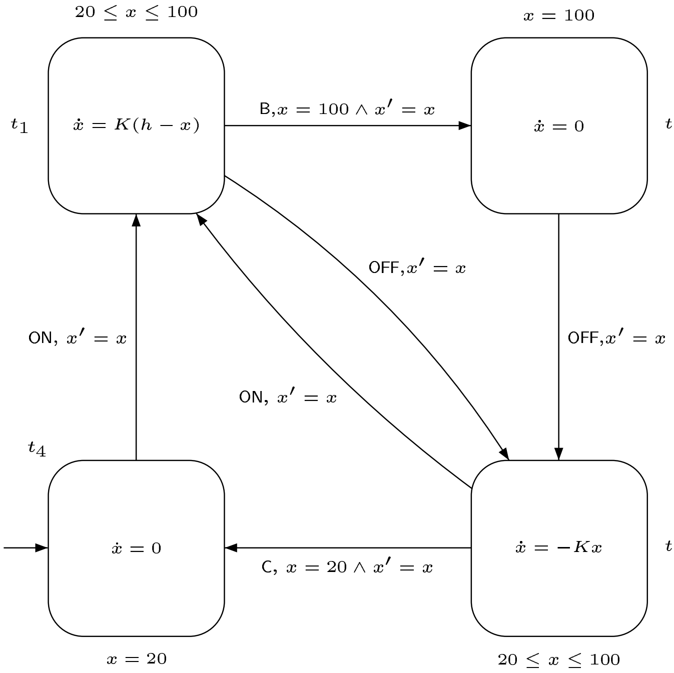


**Fig. 3.** A hybrid automaton for the tank

location $t_4$ models the behavior of the tank when the gas burner is OFF and the temperature of the water is equal to 20 degrees.

The edge from location $t_1$ to location $t_2$ is crossed when the temperature of the water reaches 100 degrees. In that situation, the control of the hybrid automaton cannot stay in location $l_1$ without violating the invariant $20 \le x \le 100$. The predicate $x = 100$ ensures that this edge can only be crossed when the temperature has reached 100 degrees. Edge from location $t_1$ to location $t_3$ is crossed when the burner is *turned off*. In that case, the dynamics of the system changes instantaneously from $\dot{x} = K(h-x)$ to $\dot{x} = -Kx$. This edge can be crossed at any time when the control is in location $t_2$. In the sequel, we fix the value of $K$ to be 0.075 and $h$ to be 150.

### 2.2 Semantics

At any instant, the *state* of the hybrid system specifies a *control location* and values for all *real-valued variables.* The state can change in two ways: ($i$) by an instantaneous transition jump that changes possibly both the control location and the values of some real-valued variables, or ($ii$) by a time delay that changes only the values of the real-valued variables in a smooth manner according to the flow and invariant of the current control location. To capture those behaviors in a formal way, we use *timed transition systems*, which are defined as follows.

**Definition 2** [Timed Transition System] A *timed transition system* $\mathsf{TTS}$ is a tuple $\langle S, S_0, \Sigma, \rightarrow \rangle$ where $S$ is a (possibly infinite) set of states, $S_0 \subseteq S$ is the subset of initial states, $\Sigma$ is a finite set of labels, and $\rightarrow \subseteq S \times \Sigma \cup \mathbb{R}^{\geq 0} \times S$ is the transition relation. We write $s \rightarrow_d s'$ for $(s, d, s') \in \rightarrow$.

We denote $[X \rightarrow \mathbb{R}]$ the set of valuations that map variables from $X$ to real numbers. Let $p$ be a predicate over the set of variables $X$, then $[\![p]\!]$ is the set of valuations $v \in [X \rightarrow \mathbb{R}]$ satisfying $p$, noted $v \models p$. Let $q$ be a predicate over the set of variables $X \cup X'$, then $[\![q]\!]$ is the set of pairs of valuations $(v, v') \in [X \rightarrow \mathbb{R}] \times [X' \rightarrow \mathbb{R}]$ such that $(v, v') \models q$. Let $r$ be a predicate over the set of variables $X \cup \dot{X}$, then $[\![r]\!]$ is the set of pairs of valuations $(v, \dot{v}) \in [X \rightarrow \mathbb{R}] \times [\dot{X} \rightarrow \mathbb{R}]$ such that $(v, \dot{v}) \models r$. The $\mathsf{TTS}$ associated to a hybrid automaton is defined as follows.

**Definition 3** The timed transition system $\langle S, S_0, \Sigma, \rightarrow \rangle$ of the hybrid automaton $H = \langle \mathsf{Loc}, \mathsf{Edge}, \Sigma, X, \mathsf{Init}, \mathsf{Inv}, \mathsf{Flow}, \mathsf{Jump} \rangle$, written as $[\![H]\!]$, is defined as follows:

- $S$ is the set of pairs $(l, v)$ where $l \in \mathsf{Loc}$ and $v \in [X \rightarrow \mathbb{R}]$ such that $v \in [\![\mathsf{Inv}(l)]\!]$, this set is called the *state space* of $H$;
- $S_0$ is the subset of pairs $(l, v) \in S$ such that $v \in [\![\mathsf{Init}(l)]\!]$, this set is called the *initial state space* of $H$;
- the transitions are either:
  - **discrete**: for each edge $e = (l, \sigma, l') \in \mathsf{Edge}$, we have $(l, v) \rightarrow_\sigma (l', v')$ if $(l, v) \in S$, $(l', v') \in S$, and we have that $(v, v') \in [\![\mathsf{Jump}(e)]\!]$;
  - **continuous**: for each nonnegative real $\delta \in \mathbb{R}^{\geq 0}$, we have $(l, v) \rightarrow_\delta (l', v')$ if $l = l'$, $(l, v) \in S$, $(l', v') \in S$, and there is a differentiable function $f : [0, \delta] \rightarrow \mathbb{R}^m$, with first derivative $\dot{f} : (0, \delta) \rightarrow \mathbb{R}^m$, such that the following conditions hold:
    - $f(0) = v$,
    - $f(\delta) = v'$,
    - for all reals $\epsilon \in (0, \delta)$, both $f(\epsilon) \in [\![\mathsf{Inv}(l)]\!]$ and $(f(\epsilon), \dot{f}(\epsilon)) \in [\![\mathsf{Flow}(l)]\!]$.

    The function $f$ is called a *witness* for the transition $(l, v) \rightarrow_\delta (l', v')$.

In this transition system, we abstract continuous flows by transitions retaining only the information about the source, target and duration of each flow.

The paths contained in the timed transition system of a hybrid automaton $H$ are formal representations of the possible trajectories of the hybrid system modeled by $H$, i.e., the evolutions of the state of the hybrid system along time. Formally, a *finite path*, noted $\lambda$, in the timed transition system $T = \langle S, S_0, \Sigma, \rightarrow \rangle$ is a finite sequence alternating between states and transition labels $s_0 \tau_0 s_1 \tau_1 \ldots \tau_{n-1} s_n$ such that at any $i$, $0 \leq i \leq n$, $s_i \in S$ and for any $i$, $0 \leq i < n$, $(s_i, \tau_i, s_{i+1}) \in \rightarrow$. We call $n + 1$ the *length of the path* and it is denoted by $|\lambda|$. This definition is extended to infinite paths as follows: an *infinite path* $\lambda$ in the timed transition system $T$ is an infinite sequence alternating between states and transition labels $s_0 \tau_0 s_1 \tau_1 \dots \tau_{n-1} s_n \dots$ such that for any $i \geq 0$: $s_i \in S$ and $(s_i, \tau_i, s_{i+1}) \in \rightarrow$. The length of an infinite path is $+\infty$. The *duration* of a (finite or infinite) path $\lambda$ is the sum of time labels that appear along $\lambda$. That is, given $\lambda = s_0 \tau_0 s_1 \tau_1 \dots s_n \tau_n \dots$, let $J$ be a subset of indices $j$ in $\{0, 1, \dots |\lambda|\}$ such that $\tau_j \in \mathbb{R}^{\geq 0}$, then the duration of $\lambda$ is defined by $\mathsf{Duration}(\lambda) = \sum_{j \in J} \tau_j$. We say that a (finite or infinite) path $\lambda$ is *initial* if its first state $s_0$ is an initial state of the $\mathsf{TTS}$, i.e. $s_0 \in S_0$. We write $\mathsf{Path}_F(T)$ for the set of *finite initial paths* of $S$ and $\mathsf{Path}_\infty(T)$ for the set of *infinite initial paths* of $S$.

*Example 1.* The following path belongs to $\mathsf{Path}_F([\![\mathsf{Tank}]\!])$:

$$(t_4, x \mapsto 20) \overbrace{\rightarrow_{\mathsf{ON}}}^{(1)} (t_1, x \mapsto 20) \overbrace{\rightarrow_{10}}^{(2)} (t_1, x \mapsto 88.59\dots) \overbrace{\rightarrow_{2.74\dots}}^{(3)} (t_1, x \mapsto 100) \overbrace{\rightarrow_{\mathsf{B}}}^{(4)}$$
$$(t_2, x \mapsto 100) \overbrace{\rightarrow_{5}}^{(5)} (t_2, x \mapsto 100) \overbrace{\rightarrow_{\mathsf{OFF}}}^{(6)} (t_3, x \mapsto 100) \overbrace{\rightarrow_{8}}^{(7)} (t_3, x \mapsto 54.88\dots)$$

Transition (1) is discrete: the control of the tank instantaneously changes from control location $t_4$ to control location $t_1$. The value of $x$ remains equal to 20 due to the jump predicate $x' = x$ expressing that the value of $x$ is left unchanged by the discrete jump. The witness function for time step (2) is $f(t) = 20e^{-0.075t} + 150(1 - e^{-0.075t})$ on the interval $[0, 10]$. For time step (3) the witness function is $f(t) = 88.59\dots e^{-0.075t} + 150(1 - e^{-0.075t})$ on the interval $[0, 2.75]$. Transition (4) is a discrete change that is forced by the invariant $20 \leq x \leq 100$ that labels location $t_1$. The witness function for time step (5) is $f(t) = 100$ on the interval $[0, 5]$. Transition (6) is a discrete change that can occur at any time when in location $t_2$. The witness function for time step (7) is $f(t) = 100e^{-0.075t}$ on the interval $[0, 8]$.

*Remark 1.* If we are interested in the infinite behaviors of a hybrid automaton, then we are usually interested in infinite sequences of transitions that do not converge in time. In fact, trajectories during which an infinite number of discrete changes occur in a finite amount of time are not realistic. It is clear that if a controller takes discrete switches say at times $\frac{1}{2}, \frac{3}{4}, \frac{7}{8}, \frac{15}{16}, \dots$, then it is not implementable. In this case, we say that the controller is Zeno. The *nonZenoness property* of an infinite path can be expressed as follows. Let $T\langle S, S_0, \Sigma, \rightarrow\rangle$ be a $\mathsf{TTS}$ and $\lambda$ be an infinite path of $T$. The path $\lambda$ is *nonZeno* if and only if $\mathsf{Duration}(\lambda) = +\infty$. The divergence of time is a *liveness* assumption [AS85], and it is the only liveness assumption we need to consider [Hen92]. Algorithmic methods for checking nonzenoness properties of timed and hybrid automata are given in [HNSY94].

## 2.3 Composition

Nontrivial systems consist of several *interacting components* (three in our running example). We model each component as a hybrid automaton, and the components coordinate with each other by *shared variables* and *shared events*. The automaton for the thermometer and for the gas burner are given in Fig. 4. The thermometer uses the shared variable $x$ to synchronize with the tank: the behavior of the thermometer depends on the evolution of the variable $x$ whose evolution is governed by the hybrid automaton that models the tank. The flow of this variable is not constrained in the thermometer automaton. In our formalization, the thermometer samples the variable $x$ once every $\frac{1}{10}$ time units and issues the event $\mathsf{DW93}$ if the temperature is below 93 degrees, issues the event $\mathsf{UP95}$ if the temperature is above 95 degrees, and issues an *internal* event $\epsilon$ in other cases (this event is not shared with other components). The sampling rate is enforced using the analog variable $z$ that evolves with a derivative equal to 1. Such a variable counts time and is called a *clock*. The gas burner uses events to synchronize with the tank. The gaz burner communicates with the tank by synchronizing control switches through the two shared events $\mathsf{ON}$ and $\mathsf{OFF}$. The time needed for the gas burner to turn off or turn on is fixed at $\frac{1}{10}$ time units.

To formalize those intuitions, we use the notion of the product of two hybrid automata which is defined as follows.

**Definition 4** [Automata-Product] Let $H^1 = \langle \mathsf{Loc}^1, \mathsf{Edge}^1, \Sigma^1, X^1, \mathsf{Init}^1, \mathsf{Inv}^1, \mathsf{Flow}^1, \mathsf{Jump}^1 \rangle$ and $H^2 = \langle \mathsf{Loc}^2, \mathsf{Edge}^2, \Sigma^2, X^2, \mathsf{Init}^2, \mathsf{Inv}^2, \mathsf{Flow}^2, \mathsf{Jump}^2 \rangle$ be two hybrid automata such that $\mathsf{Loc}^1 \cap \mathsf{Loc}^2 = \emptyset$. Their synchronized product, noted as $H^1 \otimes H^2$, is the hybrid automaton $H = \langle \mathsf{Loc}, \mathsf{Edge}, \Sigma, X, \mathsf{Init}, \mathsf{Inv}, \mathsf{Flow}, \mathsf{Jump} \rangle$ defined as follows:

- $\mathsf{Loc} = \big\{ \{l^1, l^2\} \mid l^1 \in \mathsf{Loc}^1 \wedge l^2 \in \mathsf{Loc}^2 \big\}$.
- $\mathsf{Edge}$ is defined as follows: $(\{l_1^1, l_1^2\}, \sigma, \{l_2^1, l_2^2\}) \in \mathsf{Edge}$ iff either
  1. $\sigma \in \Sigma^1 \setminus \Sigma^2$, $(l_1^1, \sigma, l_2^1) \in \mathsf{Edge}^1$, and $l_1^2 = l_2^2$;
  2. $\sigma \in \Sigma^2 \setminus \Sigma^1$, $(l_1^2, \sigma, l_2^2) \in \mathsf{Edge}^2$, and $l_1^1 = l_2^1$;
  3. $\sigma \in \Sigma^1 \cap \Sigma^2$, $(l_1^1, \sigma, l_2^1) \in \mathsf{Edge}^1$ and $(l_1^2, \sigma, l_2^2) \in \mathsf{Edge}^2$.

  Conditions (1) and (2) express that unshared events (also called *internal events*) are interleaved while condition (3) expresses that shared events must occur simultaneously in the two automata.
- $\Sigma = \Sigma^1 \cup \Sigma^2$.
- $X = X^1 \cup X^2$.
- for any location $\{l^1, l^2\} \in \mathsf{Loc}$, we have that $\mathsf{Init}(\{l^1, l^2\}) = \mathsf{Init}^1(l^1) \wedge \mathsf{Init}^2(l^2)$.
- for any location $\{l^1, l^2\} \in \mathsf{Loc}$, we have that $\mathsf{Inv}(\{l^1, l^2\}) = \mathsf{Inv}^1(l^1) \wedge \mathsf{Inv}^2(l^2)$.
- for any location $\{l^1, l^2\} \in \mathsf{Loc}$, we have that $\mathsf{Flow}(\{l^1, l^2\}) = \mathsf{Flow}^1(l^1) \wedge \mathsf{Flow}^2(l^2)$.
- for any edge $(\{l_1^1, l_1^2\}, \sigma, \{l_2^1, l_2^2\}) \in \mathsf{Edge}$, we have that:
  1. $\mathsf{Jump}(\{l_1^1, l_1^2\}, \sigma, \{l_2^1, l_2^2\}) = \mathsf{Jump}((l_1^1, \sigma, l_2^1)) \wedge \bigwedge_{x \in X^2 \setminus X^1} x' = x$ if $\sigma \in \Sigma^1 \setminus \Sigma^2$;
  2. $\mathsf{Jump}(\{l_1^1, l_1^2\}, \sigma, \{l_2^1, l_2^2\}) = \mathsf{Jump}((l_1^2, \sigma, l_2^2)) \wedge \bigwedge_{x \in X^1 \setminus X^2} x' = x$ if $\sigma \in \Sigma^2 \setminus \Sigma^1$;

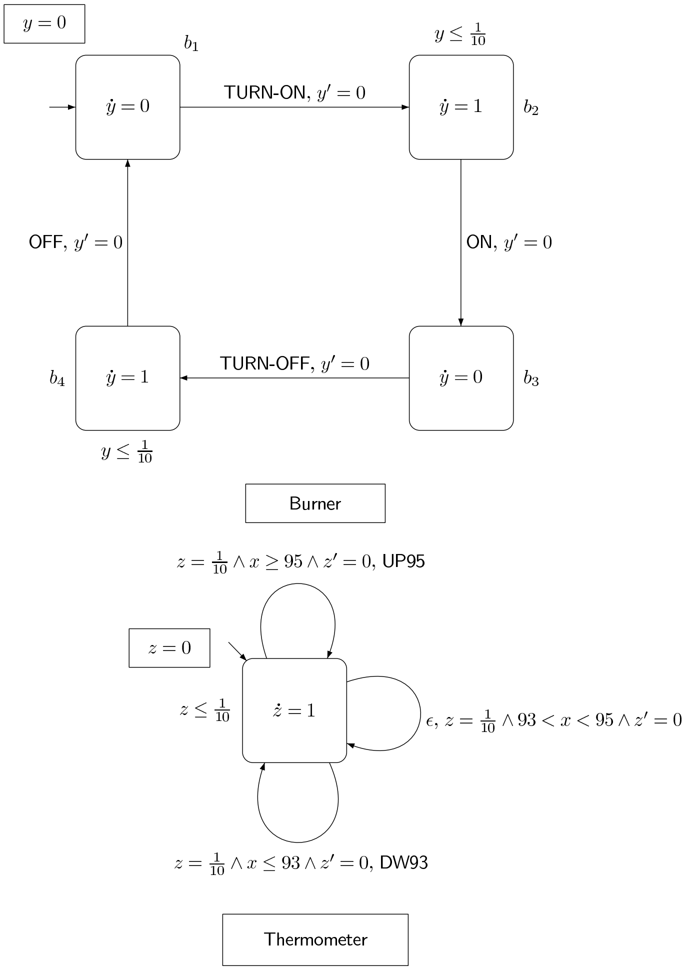
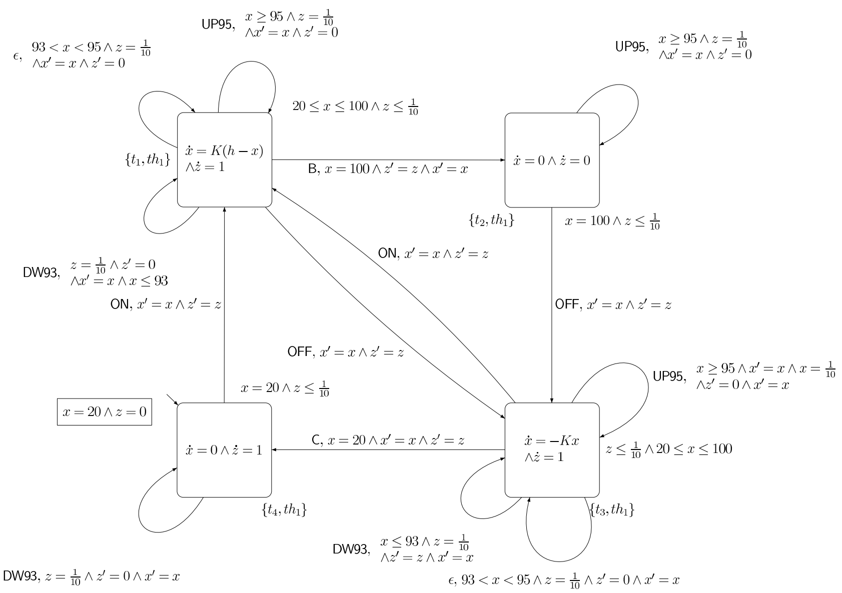


**Fig. 4.** Hybrid automata for the burner and the thermometer

3. $\mathsf{Jump}(\{l_1^1, l_1^2\}, \sigma, \{l_2^1, l_2^2\}) = \mathsf{Jump}((l_1^1, \sigma, l_2^1)) \wedge \mathsf{Jump}((l_1^2, \sigma, l_2^2))$ if $\sigma \in \Sigma^1 \cap \Sigma^2$;

Conditions 1 and 2 express that discrete changes that are local to one automaton have the enabling condition and the effect described by the jump predicate of that automaton and the variables which are not shared remain unchanged. Condition 3 expresses that discrete changes shared by the two automata have as enabling condition the conjunction of the enabling conditions of each discrete change. Their effect is the conjunction of the effects of each discrete change.

In our example, we obtain the complete system by composing the three automata. It is easy to show that the product operation that we have defined is commutative and associative, so we can write $\mathsf{Sys} = \mathsf{Tank} \otimes \mathsf{Burner} \otimes \mathsf{Thermo}$. Fig. 5 shows the hybrid automaton obtained by composing the automaton for the tank and the automaton for the thermometer. We have omitted transitions that are incompatible with the invariant of their starting location. That is, edges $e = (l, \sigma, l')$ such that $[\![\mathsf{Jump}(e) \wedge \mathsf{Inv}(l)]\!] = \emptyset$ are not depicted.



UP95, $x \geq 95 \wedge z = \frac{1}{10}$ $\wedge x' = x \wedge z' = 0$
$\epsilon$, $93 < x < 95 \wedge z = \frac{1}{10}$ $\wedge x' = x \wedge z' = 0$
UP95, $x \geq 95 \wedge z = \frac{1}{10}$ $\wedge x' = x \wedge z' = 0$
$20 \leq x \leq 100 \wedge z \leq \frac{1}{10}$
$\{t_1, th_1\}$
$\dot{x} = K(h - x)$ $\wedge \dot{z} = 1$
B, $x = 100 \wedge z' = z \wedge x' = x$
$\dot{x} = 0 \wedge \dot{z} = 0$
$\{t_2, th_1\}$
$x = 100 \wedge z \leq \frac{1}{10}$
ON, $x' = x \wedge z' = z$
DW93, $z = \frac{1}{10} \wedge z' = 0$ $\wedge x' = x \wedge x \leq 93$
ON, $x' = x \wedge z' = z$
OFF, $x' = x \wedge z' = z$
OFF, $x' = x \wedge z' = z$
UP95, $x \geq 95 \wedge x' = x \wedge x = \frac{1}{10}$ $\wedge z' = 0 \wedge x' = x$
$x = 20 \wedge z \leq \frac{1}{10}$
$x = 20 \wedge z = 0$
$\dot{x} = 0 \wedge \dot{z} = 1$
C, $x = 20 \wedge x' = x \wedge z' = z$
$\dot{x} = -Kx$ $\wedge \dot{z} = 1$
$z \leq \frac{1}{10} \wedge 20 \leq x \leq 100$
$\{t_4, th_1\}$
$\{t_3, th_1\}$
DW93, $x \leq 93 \wedge z = \frac{1}{10}$ $\wedge z' = z \wedge x' = x$
DW93, $z = \frac{1}{10} \wedge z' = 0 \wedge x' = x$
$\epsilon$, $93 < x < 95 \wedge z = \frac{1}{10} \wedge z' = 0 \wedge x' = x$

**Fig. 5.** Product of tank and thermometer

## 3 Properties of Hybrid Systems

Properties assign values to trajectories of hybrid systems. In this introduction, we restrict ourselves to properties that classify trajectories as *good* or *bad* according to whether or not they stay or not in a given set of (good) states. Those properties are called *safety properties* [AS85], and, are the most important class of properties when considering *safety critical systems*.

Let us go back to our running example. Now that we have a complete model of our system, we would like to design a controller that enforces some desired behaviors. The controller will be an additional hybrid automaton that, when composed with the automata modeling our system, must enforce the following properties on the trajectories of the entire system:

- ($R1$) the temperature in the tank must never reach 100 degrees;
- ($R2$) after 15 seconds of operation, the system must be in *stable regime*, which means that the temperature of the water in the tank must always stay between 91 and 97 degrees;
- ($R3$) during this stable regime, the burner is never continuously ON for more than two seconds.

The three properties above are *safety properties.* They impose that the system should stay within a set of *safe states*, or equivalently, that the system should never enter a set of *bad states* (states where the safety property is violated). This is clear for property ($R1$) where the bad states are the states where the value of $x$ is greater than or equal to 100. We will see later that this is also the case for the other two properties. In this chapter, we only focus on safety properties. Pointers to the literature are given in the last section for other classes of properties.

We propose in Fig. 6, a possible controller for our system. The behavior of this controller is as follows. The controller observes two events coming from the thermometer (UP95, DW93) and issues two events toward the gas burner (TURN-ON, TURN-OFF). Initially, the controller waits in location $c_1$ until it sees the event DW93. When this event occurs, the control switches instantaneously to location $c_2$. There, it immediately switches to $c_3$ by emitting the event TURN-ON toward the gas burner. In location $c_3$, the controller ignores the event DW93 and waits for the event UP95. When this event takes place, the control moves to location $c_4$ where it instantaneously emits the event TURN-OFF toward the gas burner.

In the next section, we show how to formalize the requirements expressed informally above and how to prove, using algorithmic methods, that the controller we propose fulfills those requirements.

### 3.1 Safety properties and monitors

*Safety properties*

To formalize safety properties, we need some more notation. Let $T = \langle S, S_0, \Sigma, \rightarrow \rangle$ be a TTS. Let $\lambda = s_0 \tau_0 s_1 \tau_1 \dots s_n$ be a finite path in $T$. We denote $\mathsf{State}(\lambda)$ for the set of states that *appear* along the path $\lambda$. We say that a path $\lambda$ *reaches* a state $s$ if $s \in \mathsf{State}(\lambda)$. We say that a state $s$ is *reachable* in $T$ if $s \in \bigcup_{\lambda \in \mathsf{Path}_F(T)} \mathsf{State}(\lambda)$. The set of states that are reachable in $T$ is noted $\mathsf{Reach}(T)$. A set of states $R \subseteq S$ is called a *region.* We note $\overline{R}$ for the complement of $R$ in the state space of $T$, that is, $\overline{R} = S \setminus R$. We say that $T$ is *safe for* $R$ iff $\mathsf{Reach}(T) \subseteq R$. A region $R$ is *reachable* in $T$ iff $R \cap \mathsf{Reach}(T) \neq \emptyset$.

**Definition 5** [Verification Problems] Let $H$ be a hybrid automaton with TTS $[\![H]\!]$ whose state space is $S$, and let $R \subseteq S$ be a region. The *safety problem* associated to $R$ asks whether $\mathsf{Reach}([\![H]\!]) \subseteq R$.The *reachability problem* associated to $R$ asks whether $\mathsf{Reach}([\![H]\!]) \cap R \neq \emptyset$.

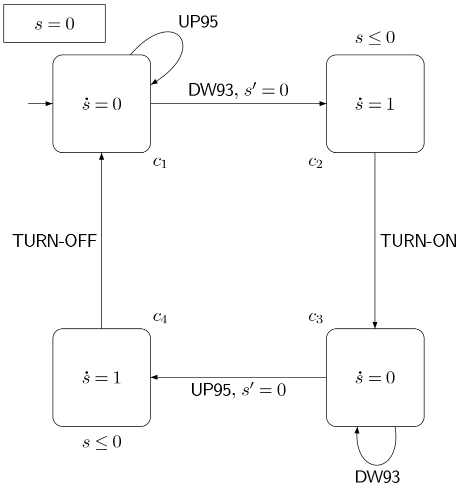


**Fig. 6.** A controller for the system

Those two problems are *dual* in the following formal sense.

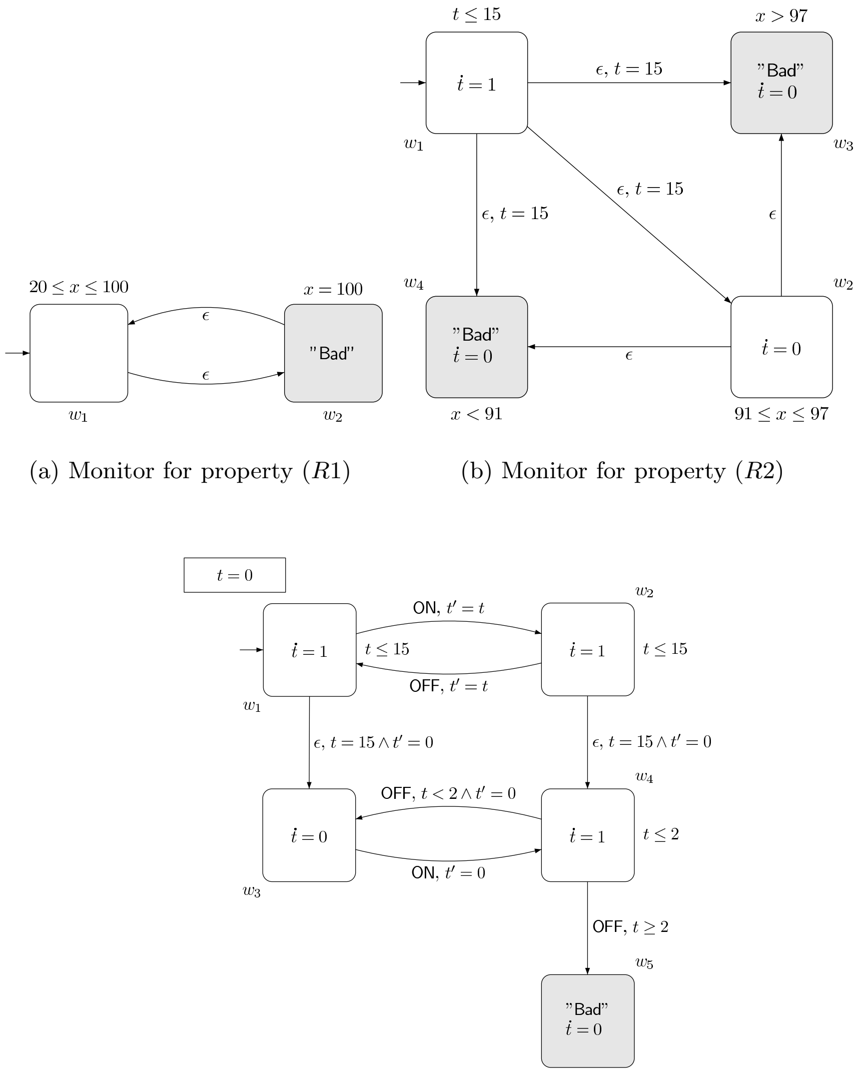
**Theorem 1.** *For any* TTS $T$*, for any region* $R$ *of* $T$*,* $\mathsf{Reach}(T) \subseteq R$ *iff* $\mathsf{Reach}(T) \cap \overline{R} = \emptyset$*.*

Hence, solving a safety problem boils down to solving its dual reachability problem. In that reachability problem, the region $\overline{R}$ is often called the set of *bad states.*

*Monitors*

In order to formalize safety requirements, it is often very convenient to use a *monitor automaton*, also often called an *observer*, that "watches" the trajectories of the system and enters "Bad" locations whenever one trajectory violates a given safety property. Safety verification is then reduced to deciding the reachability of a set of "Bad" locations.

In Fig. 7(a), 7(b), and 7(c), we give the monitors for the safety requirements ($R1$), ($R2$), and ($R3$) respectively. The automaton $\mathsf{Moni}_1$ monitors the value of variable $x$ whose dynamics is defined in the tank automaton. As soon as $x$ reaches the value 100, the control of the monitor can move to location $w_2$ which is a Bad location. Thus to verify property ($R1$), we have to establish that no state in which the control of $\mathsf{Moni}_1$ is in location $w_2$ is reachable in $[\![\mathsf{Tank} \otimes \mathsf{Burner} \otimes \mathsf{Thermo} \otimes \mathsf{Controller} \otimes \mathsf{Moni}_1]\!]$. In that case, we know that the controller ensures requirement ($R1$). The automaton $\mathsf{Moni}_2$ initially maintains a variable $t$ that counts the time elapsed since the initialization of the system. When this variable reaches value 15 (the system was started 15 seconds ago), the control has to leave location $w_1$. If the value of variable $x$ (the



(a) Monitor for property ($R1$)

(b) Monitor for property ($R2$)

(c) Monitor for property ($R3$)

**Fig. 7.** Monitors for the safety properties ($R1$), ($R2$), and ($R3$)

temperature of the water inside the tank) at that time is between 91 and 97, the control moves to location $w_2$ and the control can stay there only if the temperature stays within this interval of values. On the other hand, if the value is, or becomes, less than 91, it moves to location $w_3$, and if the value is, or becomes, greater than 97, it moves to location $w_4$. Locations $w_3$ and $w_4$ are the $\mathsf{Bad}$ locations. It is clear that if no state in which the control of $\mathsf{Moni}_2$ is either $w_3$ or $w_4$ is reachable in $[\![\mathsf{Tank} \otimes \mathsf{Burner} \otimes \mathsf{Thermo} \otimes \mathsf{Controller} \otimes \mathsf{Moni}_2]\!]$, then we know that the controller ensures requirement $(R2)$. Finally, the automaton $\mathsf{Moni}_3$ works as follows. For the first 15 seconds, the control stays in location $w_1$, if the burner is $\mathsf{OFF}$, or in location $w_2$, if the burner is $\mathsf{ON}$. After 15 seconds, the control moves to location $w_3$ or $w_4$. In $w_3$, each time the event $\mathsf{ON}$ occurs, the variable $t$ is reset and the control moves to location $w_4$ where $t$ counts time. There, the monitor waits for the next $\mathsf{OFF}$ event. If this next $\mathsf{OFF}$ event occurs within 2 time units ($t < 2$), then the control moves back to location $w_3$ where the monitor waits for the next $\mathsf{ON}$ event. On the other hand, if the event $\mathsf{OFF}$ occurs after 2 time units ($t \geq 2$), then the control of the monitor moves to location $w_5$, a "$\mathsf{Bad}$" location. Again, it is clear that our system satisfies requirement $(R3)$ if no state where the control of $\mathsf{Moni}_3$ is in location $w_5$ can be reached in $[\![\mathsf{Tank} \otimes \mathsf{Burner} \otimes \mathsf{Thermo} \otimes \mathsf{Controller} \otimes \mathsf{Moni}_3]\!]$.

### 3.2 How do we solve reachability problems?

We have seen that safety verification problems can be reduced to reachability problems. We introduce here some basic notions useful for reachability problems. Given a $\mathsf{TTS}$ $T = \langle S, S_0, \Sigma, \rightarrow\rangle$, we define the following two operators:

- the *direct successor operator* $\mathsf{Post}_T : 2^S \rightarrow 2^S$, is an operator that, given a set of states, returns the set of direct successors of those states in $T$. Formally, for any $S' \subseteq S$, we have that

  $$\mathsf{Post}_T(S') = \big\{s \in S \mid \exists s' \in S' : (\exists \sigma \in \Sigma : s' \rightarrow_\sigma s \vee \exists \delta \in \mathbb{R}^{\geq 0} : s' \rightarrow_\delta s)\big\}.$$

- the *direct predecessor operator* $\mathsf{Pre}_T : 2^S \rightarrow 2^S$, is an operator that, given a set of states, returns the set of direct predecessors of those states in $T$. Formally, for any $S' \subseteq S$, we have that

  $$\mathsf{Pre}_T(S') = \big\{s \in S \mid \exists s' \in S' : (\exists \sigma \in \Sigma : s \rightarrow_\sigma s' \vee \exists \delta \in \mathbb{R}^{\geq 0} : s \rightarrow_\delta s')\big\}.$$

The set of reachable states of a hybrid automaton $H$ can be described as the least solution (for the subset order over sets of states) of equations constructed using the direct successor or predecessor operators:

- The set of reachable states of a hybrid automaton $H$ with $\mathsf{TTS}$ $[\![H]\!]=$ $\langle S, S_0, \Sigma, \rightarrow\rangle$ can be described as the least solution of the following equation:

  $$X = (S_0 \cup \mathsf{Post}_{[\![H]\!]}(X)), \tag{1}$$

  where $X$ ranges over sets of states.

- Symmetrically, the set of states of this automaton that can reach a given region $R$ can be described as the least solution of the following equation:

  $$X = (R \cup \mathsf{Pre}_{[\![H]\!]}(X)), \tag{2}$$

  where $X$ ranges over sets of states.

As the direct successor and the direct predecessor operators are monotone for the subset order, we know by the Tarsky theorems that the least solutions of those equations can be obtained by successive approximations. Unfortunately, this does not mean that we can effectively solve those equations. In fact, the fixpoint is not necessarily reached within a finite number of steps. In general, reachability problems are undecidable for even the simplest class of hybrid automata (we give detailed references to the literature later). Even applying the direct successor or predecessor operator to a region one time may be very difficult as it amounts to solving general differential equations. We do not know how to do that in general. This is why subclasses of hybrid systems for which we know how to compute direct successors or predecessors of regions have been studied in the literature [ACH$^+$95]. In the next section, we study a particularly interesting one, the *rectangular hybrid automata* [PV94, HKPV98].

# 4 Rectangular Hybrid Automata

## 4.1 Syntactic restrictions

An interval is a convex non-empty subset of the positive real numbers with greatest lower bound in $\mathbb{Q} \cup \{-\infty\}$ and least upper bound in $\mathbb{Q} \cup \{+\infty\}$. As usual, intervals can be denoted by $(a, b), [a, b), (a, b]$ or $[a, b]$ where $a \in \mathbb{Q} \cup \{-\infty\}$ and $b \in \mathbb{Q} \cup \{+\infty\}$, and $a \leq b$. Let $I$ be an interval, we note $glb(I)$ for the *greatest lower bound* of $I$ and $lub(I)$ for the *least upper bound* of $I$. Let $X$ be a set of variables, we note $\mathsf{Rect}(X)$ for the following set of formulas:

$$\mathsf{Rect}(X) \ni \phi_1, \phi_2 := \bot \mid \top \mid x \in I \mid \phi_1 \wedge \phi_2$$

, where $x$ belongs to the set of variables $X$, and $I$ is an interval. Those formulas are called *rectangular predicates*. The set of formula $\mathsf{Rect}(\dot{X})$ is defined in the same way, replacing $X$ by $\dot{X}$. Those formulas are called *rectangular flow predicates*. We need a last set of formulas. We denote by $\mathsf{UpdateRect}(X)$, the following set of formulas:

$$\mathsf{UpdateRect}(X) \ni \phi_1, \phi_2 := \bot \mid \top \mid x \in I \mid x' \in I \mid x' = x \mid \phi_1 \wedge \phi_2$$

where $x$ belongs to the set of variables $X$, $x'$ belongs to $X'$ the set of primed copies of variables in $X$, and $I$ is an interval. Formulas from this set are called *rectangular update predicates*.

**Definition 6** [Rectangular Automaton] A rectangular automaton is a hybrid automaton $H = \langle \mathsf{Loc}, \mathsf{Edge}, \Sigma, X, \mathsf{Init}, \mathsf{Inv}, \mathsf{Flow}, \mathsf{Jump}\rangle$ where for any $l \in \mathsf{Loc}$, $\mathsf{Init}(l)$ and $\mathsf{Inv}(l)$ are rectangular predicates over $X$, that is, formulas taken in $\mathsf{Rect}(X)$, for any edge $e \in \mathsf{Edge}$, $\mathsf{Jump}(e)$ is a rectangular update predicate over $X$, that is a formula taken in $\mathsf{UpdateRect}(X)$, and finally, for any location $l \in \mathsf{Loc}$, $\mathsf{Flow}(l)$ is a rectangular flow predicate over $\dot{X}$, that is, a formula taken in $\mathsf{Rect}(\dot{X})$.

It is easy to show that the composition of two rectangular automata is again a rectangular automaton. The hybrid automata for the gaz burner, the thermometer, the controller, and the three monitors are all rectangular hybrid automata.

### 4.2 Reachability analysis of rectangular hybrid automata

The computation of the $\mathsf{Pre}$ and $\mathsf{Post}$ operators is easier in the case of rectangular hybrid automata. For that class of hybrid automata, we are able to define a semi-algorithm (no guarantee of termination) for reachability. This semi-algorithm manipulates regions that are infinite sets of states. Therefore, we need a way to represent regions in a symbolic way.

A *linear term* over the set of variables $X$ is a linear combination of the variables in $X$ with integer coefficients. A *linear formula* over $X$ is a boolean combination of inequalities between linear terms over $X$. Given a linear formula $\Psi$, we write $\llbracket \Psi \rrbracket$ for the set of valuations $v$ of the variables in $X$ such that $v \models \Psi$. If we allow quantifiers with linear formulas, we obtain the *theory of reals with addition*, noted $\mathsf{T}(\mathbb{R}, 0, 1, +, \leq)$. Note that rectangular predicates, rectangular flow predicates, and rectangular update predicates are linear formulas over $X$, $\dot{X}$, and $X \cup X'$ respectively.

Let $H = \langle \mathsf{Loc}, \mathsf{Edge}, \Sigma, X, \mathsf{Init}, \mathsf{Inv}, \mathsf{Flow}, \mathsf{Jump}\rangle$ be a rectangular automaton. A *symbolic region* $\mathcal{R}$ of $H$ is a finite set $\{(l, \Psi_l) \mid l \in \mathsf{Loc}\}$ of pairs, where $l \in \mathsf{Loc}$ is a location of the automaton and $\Psi_l$ is a linear formula such that $\llbracket \Psi_l \rrbracket \subseteq \llbracket \mathsf{Inv}(l) \rrbracket$. Let $l \in \mathsf{Loc}$ and let $\mathsf{Flow}(l)$ be the rectangular flow predicate that labels $l$. We denote by $\llbracket \mathsf{Flow}(l) \rrbracket\ (x)$ the set of values $\{\dot{v}(x) \mid \dot{v} \in \llbracket \mathsf{Flow}(l) \rrbracket\}$, that is the set of possible values of the first derivative of variable $x$ when the control is in location $l$. It is easy to show that this set is an interval of the real numbers with rational lower and upper bounds.

Given a location $l \in \mathsf{Loc}$ and a set of valuations $V \subseteq [X \to \mathbb{R}]$, such that $V \subseteq \llbracket \mathsf{Inv}(l) \rrbracket$: the *forward time closure*, noted $\langle V \rangle_l^{\nearrow}$ of $V$ at $l$ is the set of valuations of variables in $X$ that are reachable from some valuation $v \in V$ by letting time pass:

$$\langle V \rangle_l^{\nearrow} = \left\{ v' \mid \exists v \in V, t \in \mathbb{R}^{\geq 0} : \forall x \in X : \begin{array}{l} \wedge v(x) + t \times glb(\llbracket \mathsf{Flow}(l) \rrbracket\ (x)) \prec_x^1 v'(x) \\ \wedge v'(x) \prec_x^2 v(x) + t \times lub(\llbracket \mathsf{Flow}(l) \rrbracket\ (x)) \\ \wedge v' \in \llbracket \mathsf{Inv}(l) \rrbracket \end{array} \right\}$$

where

$$\prec_x^1 = \begin{cases} \leq \text{ if } glb(\llbracket \mathsf{Flow}(l) \rrbracket (x)) \in \llbracket \mathsf{Flow}(l) \rrbracket (x), \text{ i.e., the interval is left closed} \\ < \text{ if } glb(\llbracket \mathsf{Flow}(l) \rrbracket (x)) \notin \llbracket \mathsf{Flow}(l) \rrbracket (x), \text{ i.e., the interval is left open} \end{cases}$$

where

$$\prec_x^2 = \begin{cases} \leq \text{ if } lub(\llbracket \mathsf{Flow}(l) \rrbracket (x)) \in \llbracket \mathsf{Flow}(l) \rrbracket (x), \text{ i.e., the interval is right closed} \\ < \text{ if } lub(\llbracket \mathsf{Flow}(l) \rrbracket (x)) \notin \llbracket \mathsf{Flow}(l) \rrbracket (x), \text{ i.e., the interval is right open} \end{cases}$$

The set above can be defined inside $\mathsf{T}(\mathbb{R}, 0, 1, +, \leq)$. As $\mathsf{T}(\mathbb{R}, 0, 1, +, \leq)$ admits quantifier elimination, it is clear that given any linear formula $\Psi$, we can construct a linear formula $\Phi$ such that $\llbracket \Phi \rrbracket = \langle \llbracket \Psi \rrbracket \rangle_l^{\nearrow}$.

Given an edge $e \in \mathsf{Edge}$ and a set of valuations $V \subseteq [X \to \mathbb{R}]$, the *postcondition* $\mathsf{post}_e(V)$ of $V$ with respect to $e$ is the set of valuations that are reachable from some valuation $v \in V$ by taking the discrete transition $e$:

$$\mathsf{post}_e(V) = \{v' \mid \exists v \in V : (v, v') \in \llbracket \mathsf{Jump}(e) \rrbracket\}.$$

Again, as $\mathsf{T}(\mathbb{R}, 0, 1, +, \leq)$ admits quantifier elimination, and for any edge $e$, $\mathsf{Jump}(e)$ is a rectangular update predicate over $X$, and so a linear formula over $X \cup X'$, it is clear that if we are given a linear formula $\Psi$, then we can construct a linear formula $\Phi$ such that $\llbracket \Phi \rrbracket = \mathsf{post}_e(\llbracket \Psi \rrbracket)$.

We can now define the forward time closure and the edge postcondition operators of $H$ over symbolic regions. Let $\mathcal{R} = \{(l, \Psi_l \wedge \mathsf{Inv}(l)) \mid l \in \mathsf{Loc}\}$ be a symbolic region of $H$:

- $\langle \mathcal{R} \rangle^{\nearrow} = \bigcup_{l \in \mathsf{Loc}} \{(l, \langle \llbracket \Psi_l \rrbracket \rangle_l^{\nearrow})\}$
- $\mathsf{post}(\mathcal{R}) = \bigcup_{e=(l,\sigma,l') \in \mathsf{Edge}} \{(l', \mathsf{post}_e(\Psi_l))\}$

From those two operators, we can define our symbolic post operator for rectangular automata as follows. Let $\mathcal{R} = \{(l, \Psi_l) \mid l \in \mathsf{Loc}\}$ be a symbolic region of $H$:

$$\mathsf{Post}(\mathcal{R}) = \mathsf{post}(\langle \mathcal{R} \rangle^{\nearrow}).$$

Now, we can use the Tarsky fixpoint theorem to find the least solution of equation (1) by successive approximations defined as follows:

- $\mathcal{R}_0 = \{(l, \mathsf{Init}(l)) \mid l \in \mathsf{Loc}\}$
- for any integer $i > 0$, $\mathcal{R}_i = \mathcal{R}_{i-1} \cup \mathsf{Post}(\mathcal{R}_{i-1})$

This approximation schema defines naturally a semi-algorithm for reachability. This algorithm is given in Fig. 8.

### 4.3 Rectangular hybrid automata as abstractions

Let us go back to our running example. Remember that the automata for the burner, the thermometer, the controller, and the three monitors that we have defined above are all in the class of rectangular hybrid automata. The only automaton of our example which is outside the class of rectangular hybrid automata is the automaton for the tank.

```
A symbolic algorithm for reachability
begin
   $R := \{(l, \mathsf{Init}(l) \wedge \mathsf{Inv}(l)) \mid l \in \mathsf{Loc}\}$;
   $Prec := \emptyset$;
   while $[\![R]\!] \not\subseteq [\![Prec]\!]$ do
         $Prec := Prec \cup R$;
         $R := Post(R)$;
   od
   if $Bad \cap Prec = \emptyset$ then $return(OK)$ else $return(KO)$; fi
where $[\![R]\!] \not\subseteq [\![Prec]\!]$ holds if there exist $(l, \Psi) \in R$ and $(l, \Psi') \in Prec$ such that
$\forall x_1, \ldots, x_m : \Psi(x_1, \ldots, x_n) \rightarrow \Psi'(x_1, \ldots, x_n)$ is not a valid formula.
```

**Fig. 8.** Semi-algorithm for the reachability analysis of rectangular hybrid automata

In this subsection, we show how to *approximate* complex dynamics with rectangular dynamics in a systematic way. Those systematic approximations allow us to use automatic tools, like HyTech [HHWT97], to analyze approximated systems and, in a lot of practical cases, to infer the important properties of the original (complex) systems. This methodology is closely related to the theory of abstract interpretation studied by computer scientists [Cou96] and the approximation techniques used in analysis of dynamical systems [HS74].

We introduced here an approximation schema known as the *rectangular phase-portrait approximation* scheme; see [HHWT98] for more details. The idea of this approximation scheme can be stated as follows. For each control mode of the hybrid automaton that we want to approximate, the state space is partitioned into rectangular regions, and within each region, the flow field is overapproximated using rectangular flows. Those approximations may be obtained manually, using techniques from dynamical system theory, or in some cases automatically, when lower and upper bounds on derivatives can be obtained from bounds on the value of variables within rectangular regions. The approximations can be arbitrarily precise by approximating over suitably small regions of the state space.

Let us illustrate that approximation schema on our running example. Let us consider the location $t_1$ of the tank. In this location, we know that the possible values for $x$, the temperature of the water within the tank, are such that $20 \leq x \leq 100$ (this is given by the invariant that labels the location) and the flow of $x$ is given by the flow predicate $\dot{x} = K(h - x)$. As the second derivative of $x$ in the interval $[20, 100]$ is never zero, we know that the minimal value of the first derivative of $x$ in this interval occurs when $x = 100$ and the maximal derivative occurs when $x = 20$. Remember that we have fixed the value of the constant $K$ to 0.075 and $h$ to 150. With those constants, we know that the values of the first derivative of $x$ within $[20, 100]$ are bounded from below by 3.75 and from above by 9.75. It means that if we replace the flow predicate of location $t_2$ by $\dot{x} \in [\frac{375}{100}, \frac{975}{100}]$, or by $\dot{x} \in [3, 10]$ to keep things simple, then we are sure that the resulting automaton will define at least the trajectories defined by the original automaton. We can repeat this schema for each location of the original automaton. In this way we obtain a rectangular hybrid automaton that overapproximates the behavior of our original model in the sense that any trajectory of the original automaton can be mimicked by the approximating automaton (and so is a trajectory of the approximating automaton). In this introduction the notion of approximations is left informal; it can be formalized using notions like simulations [Mil71], and we refer the interested reader to [HHWT98] for a correctness proof. The automaton obtained by this schema is given in Fig. 9 and is noted RectTank.

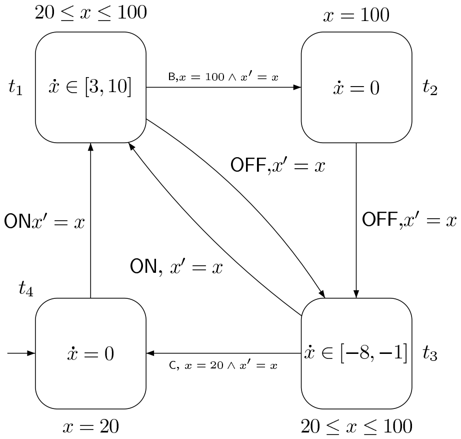


**Fig. 9.** Rectangular automaton RectTank for the tank

Let us now analyze the behaviors of our system approximated as a product of rectangular hybrid automata. This model can be analyzed using the tool HyTech [HHWT97]. HyTech is a model-checking tool for the reachability analysis of linear hybrid automata, a class of hybrid automata that subsumes the class of rectangular hybrid automata. HyTech allows us to describe each component of the system directly as a rectangular automaton in a textual syntax and to formalize reachability questions using a simple (and yet powerful) script language.

For our analysis of the tank system, we consider the product of each of the three monitors $\mathsf{Moni}_i$, $1 \leq i \leq 3$, of Fig. 7(c), with the system $\mathsf{RectTank} \otimes \mathsf{Burner} \otimes \mathsf{Thermo} \otimes \mathsf{Controller}$. Again, it is easy to show that since RectTank overapproximates the behaviors of Tank, and if "Bad" locations are not reachable in $\mathsf{RectTank} \otimes \mathsf{Burner} \otimes \mathsf{Thermo} \otimes \mathsf{Controller} \otimes \mathsf{Moni}_i$ then "Bad" is also not reachable in $\mathsf{Tank} \otimes \mathsf{Burner} \otimes \mathsf{Thermo} \otimes \mathsf{Controller} \otimes \mathsf{Moni}_i$. This means that if we can prove that a safety requirement is verified in the approximated system, then it is also verified for the original system.

When running the three verification tasks in HyTech, only the verification task of property $(R1)$ is positive in the approximated system; the two other properties turn out to be false in this approximation. HyTech provides witness trajectories that lead to bad states, that is, trajectories where the control of monitors $\mathsf{Moni}_2$ and $\mathsf{Moni}_3$ enter bad locations. If we look carefully at those trajectories, we can see that they are not possible in the original system. In particular, there are continuous transitions that cannot be mimicked by the concrete system. Those paths are present because of the overapproximation. To rule out those *spurious* paths, we have to refine our initial approximation and get closer to the real dynamics of the temperature of the water in the tank. For that purpose, we proceed as follows. As suggested above, we must partition the state space in smaller rectangular regions to capture more precisely the first derivative of $x$. To do that, we need to split some control modes of our original automaton. Consider the control mode modeled by location $t_1$, that is, when the burner is ON and the temperature is rising following the flow predicate $\dot{x} = K(h - x)$. Instead of considering only the rectangular region $20 \leq x \leq 100$, we will consider the four regions $20 \leq x \leq 50$, $50 \leq x \leq 91$, $91 \leq x \leq 95$, and finally $95 \leq x \leq 100$. For those regions, we can approximate, using the same reasoning as above, the first derivative of $x$ by the following rectangular flow predicates: $\dot{x} \in [7, 10]$ for the first region, $\dot{x} \in [4, 8]$ for the second region, $\dot{x} \in [4, 5]$ for the third region, and $\dot{x} \in [3, 5]$ for the last region. This splitting is depicted in Fig. 10. Internal actions are taken to move the control from one region to the next when the boundaries of the region are reached. We can also apply this process to location $t_3$ and split this control mode into 3 locations as follows. Instead of the region $20 \leq x \leq 100$, we use the regions $20 \leq x \leq 91$, $91 \leq x \leq 97$, and finally $97 \leq x \leq 100$. The flow predicates that we obtain are, respectively, $\dot{x} \in [-7, -1]$, $\dot{x} \in [-8, -6]$, and $\dot{x} \in [-8, -7]$. Finally, we obtain a new overapproximating automaton that we denote $\mathsf{RectTank}_2$. Fig. 11 shows how the dynamics of the temperature of the water is approximated within the refined rectangular automaton for the tank within location $t_1$.

Now if we test the reachability of the Bad location of the monitors $\mathsf{Moni}_i$, $1 \leq i \leq 3$, in $\mathsf{Tank} \otimes \mathsf{Burner} \otimes \mathsf{Thermo} \otimes \mathsf{Controller} \otimes \mathsf{Moni}_i$, with HyTech, we obtain that the “Bad” locations are not reachable in the three cases. This allows us to conclude that our controller is correct for the original (complex) system.

## 5 Beyond this Introduction

We close this chapter by referencing the literature. The interested reader will find in this section references to articles that will allow her/him to go beyond this introduction. We have organized the section into subsections devoted to active areas of research in the field of hybrid automata. Some references below have already been given above. Those references are not intended to be

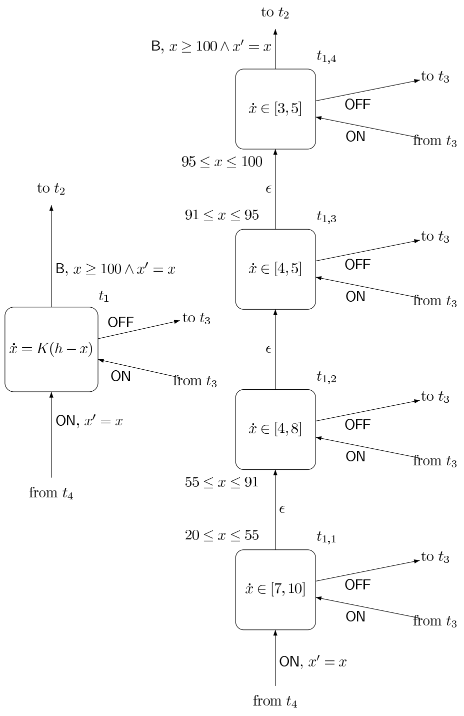


**Fig. 10.** Refinement by location splitting

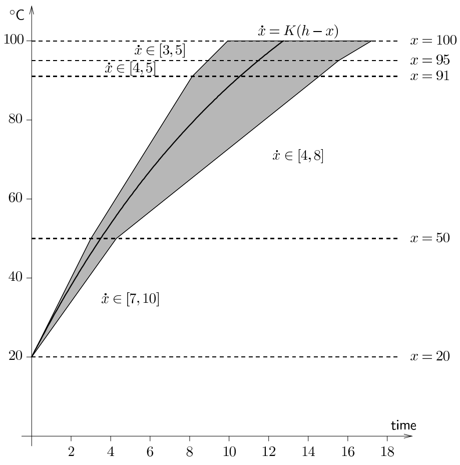


**Fig. 11.** Approximation of the dynamics by rectangles with rectangular regions

exhaustive (some important works may have been forgotten), but they are, from the point of view of the author, natural papers to look at in order to delve deeper into notions only sketched in this introductory chapter.

### 5.1 Analysis: Subclasses, decidability and complexity results

In [AD90, AD94], Alur and Dill have introduced timed automata. This was the first proposal to extend finite state automata with continuous variables. Timed automata are a subclass of hybrid automata where continuous variables are *clocks*, that is, continuous variables that have constant slopes equal to 1 (they count time), values of clocks are compared to constants, and the only updates allowed are resets to 0. The reachability problem for timed automata is decidable (it is PSpace-complete). Symbolic procedures to analyze timed automata are given in [HNSY94]. The first proposition to extend timed models to more general hybrid models can be found in [MMP92]. Rectangular hybrid automata have been proposed in [PV94]. The reachability problem of rectangular hybrid automata is undecidable in the general case, but it is decidable for the subclass of initialized rectangular hybrid automata [HKPV98]. Other interesting subclasses of hybrid automata that can be analyzed algorithmically are integration graphs [KPSY93] and dynamical systems with piecewise constant derivatives [AMP95]. More details and pointers about analysis and decidability results related to subclasses of hybrid automata can be found in [ACH$^+$95, Hen00].

### 5.2 Beyond monitors: Temporal logics and real-time logics

Temporal logics have proven useful for specification and verification of reactive systems [Pnu77, CES86]. In this introduction, we have focused only on the verification of the important class of safety properties: many more involved properties reduce to safety properties if progress of time is ensured [Hen92]. Nevertheless, there has been a lot of research on suitable formalisms to express properties of hybrid systems. In particular, temporal logics have been extended for real-time. The reader interested in real-time logics is refered to [ACD93, AFH96, RS97, Hen98, HRS98, Ras99] for definitions and verification methods related to those logics. As an illustration of the expressive power of real-time logics, we give here the formalization of the three requirements of our running example in the logic MITL [AFH96]. The following formulas are requirements that any infinite trajectory of the tank system must verify. The $\Box$ operator is read as "Always" (in the future), $\Box_{\geq 15}$ is read as "always after 15 time units", $\Diamond_{<2}$ is read as "there exists a state distant of less than 2 time units". The three requirements are then formalized as follows:

- $\Box(x < 100)$, meaning that in any trajectory, in any state, the temperature of the water is strictly less than 100 degrees;
- $\Box_{\geq 15}(91 \leq x \leq 97)$, meaning that in any trajectory, after 15 time units, the temperature of the water is always between 91 and 97 degrees;

- $\Box_{\geq 15}(\mathsf{ON} \rightarrow \Diamond_{<2}\mathsf{OFF})$, meaning that, in any trajectory, after 15 time units, any state where the burner is ON is followed within 2 time units by a state where the burner is OFF.

### 5.3 Equivalence relations and abstraction

Abstraction methods are used to simplify models and make their analysis more tractable. Several equivalence relations have been studied for subclasses of hybrid systems. For example, it can be shown that transition systems of timed automata admit finite state abstractions, called region graphs, that are time-abstract bisimilar, see [AD94] for details. Those equivalence results are used to prove decidability of verification problems on subclasses of hybrid automata [Hen95, HK96] and allow the use of well-known model-checking procedures that are guaranteed to terminate in the presence of finite quotients [HM00]. Other techniques that are not exact but use overapproximations have been proposed and have proven useful in practice: the approximation schema proposed in Section 4.3 is detailed and proven correct in [HHWT98]. Other interesting works about overapproximations can be found, among others, in [HRP94, AIKY95, ADI03].

### 5.4 Control synthesis

In this introduction, we have shown how we can model and verify controllers using hybrid automata. In our example, we have proposed a controller for the system and proven that the controller was correct for a list of requirements. A more ambitious goal than algorithmic (controller) verification is algorithmic (controller) synthesis. Here are some references about control synthesis [Won97, AMPS98, CHR02, HK99, HHM99].

### 5.5 Semantics and robustness

The semantics of hybrid automata that we defined in this chapter can be described as *perfect*. For example, it is possible to model, with this semantics, a controller that takes a given transition when a variable of the environment has exactly a given value. This can be considered as unrealistic because any implementation of such a controller will measure its environment through sensors that have *finite precision*. Alternative semantics that can be considered as *robust* are proposed in [GHJ97, HR00, AB01, Frä99].

### 5.6 Tool support and case studies

Several tools for the automatic analysis of hybrid automata have been implemented. The tools KRONOS [DOTY96] and UPPAAL [BLL$^{+}$96] can be used to analyze the subclass of timed automata. The tool HYTECH [HHWT97] allows the analysis of linear hybrid automata. The tool CHARON [AGH$^+$00] and the tool **d/dt** [ADM02] allow the analysis of a more general class of hybrid automata.

Those tools have been applied successfully to a large set of case studies in a variety of application domains. Interesting case studies can be found in [HWT96, Tom98, SMF97, BGK$^+$96, ABI$^+$01].

*Acknowledgements*

I would like to thank Alessandro Cimatti for reading a first version of this chapter and for giving invaluable advice to improve it. I would also like to thank Bram De Wachter, Laurent Doyen, Pierre Ganty, and Gilles Geeraerts for carefully reading a draft of this chapter and for helping me with several of the figures.

## References

[AB01] E. Asarin and A. Bouajjani. Perturbed turing machines and hybrid systems. In *Proc. of the IEEE Symposium on Logic in Computer Science*, pages 269–278, 2001.

[ABI$^+$01] R. Alur, C. Belta, F. Ivančić, V. Kumar, M. Mintz, G. J. Pappas, H. Rubin, and J. Schug. Hybrid modeling and simulation of biomolecular networks. *Lecture Notes in Computer Science*, 2034:19–31, 2001.

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

[ACH$^+$95] 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*, 138:3–34, 1995.

[AD90] R. Alur and D.L. Dill. Automata for modeling real-time systems. In M.S. Paterson, editor, *ICALP 90: Automata, Languages, and Programming*, Lecture Notes in Computer Science 443, pages 322–335. Springer-Verlag, Berlin, 1990.

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

[ADI03] R. Alur, T. Dang, and F. Ivancic. Counter-example guided predicate abstraction of hybrid systems. In *TACAS: International Workshop on Tools and Algorithms for the Construction and Analysis of Systems, LNCS*, 2003.

[ADM02] E. Asarin, T. Dang, and O. Maler. The **d/dt** tool for verification of hybrid systems. *Lecture Notes in Computer Science*, 2404:365–377, 2002.

[AFH96] R. Alur, T. Feder, and T.A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43:116–146, 1996.

[AGH$^+$00] Rajeev Alur, Radu Grosu, Yerang Hur, Vijay Kumar, and Insup Lee. Modular specification of hybrid systems in CHARON. In *HSCC*, pages 6–19, 2000.

[AIKY95] R. Alur, A. Itai, R.P. Kurshan, and M. Yannakakis. Timing verification by successive approximation. *Information and Computation*, 118(1):142–157, 1995.

[AMP95] E. Asarin, O. Maler, and A. Pnueli. On the analysis of dynamical systems having piecewise-constant derivatives. *Theoretical Computer Science*, 238:35–65, 1995.

[AMPS98] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller synthesis for timed automata. In *Proc. IFAC Symposium on System Structure and Control*, pages 469–474. Elsevier, Amsterdam, 1998.

[AS85] B. Alpern and F. B. Schneider. Defining liveness. *Information Processing Letters*, 21:181–185, 1985.

[BGK$^+$96] J. Bengtsson, W.O.D. Griffioen, K.J. Kristoffersen, K.G. Larsen, F. Larsson, P. Pettersson, and W. Yi. Verification of an audio protocol with bus collision using UPPAAL. In R. Alur and T.A. Henzinger, editors, *CAV 96: Computer-aided Verification*, Lecture Notes in Computer Science 1102, pages 244–256. Springer-Verlag, Berlin, 1996.

[BLL$^+$96] J. Bengtsson, K.G. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL: a tool-suite for automatic verification of real-time systems. In R. Alur, T.A. Henzinger, and E.D. Sontag, editors, *Hybrid Systems III*, Lecture Notes in Computer Science 1066, pages 232–243. Springer-Verlag, Berlin, 1996.

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

[CGP99] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

[CHR02] F. Cassez, T.A. Henzinger, and J.-F. Raskin. A comparison of control problems for timed and hybrid systems. In *HSCC 02: Hybrid Systems—Computation and Control*, Lecture Notes in Computer Science 2289, pages 134–148. Springer-Verlag, Berlin, 2002.

[Cou96] P. Cousot. Abstract interpretation. *ACM Computing Surveys*, 28(2):324–328, June 1996.

[DOTY96] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In R. Alur, T.A. Henzinger, and E.D. Sontag, editors, *Hybrid Systems III*, Lecture Notes in Computer Science 1066, pages 208–219. Springer-Verlag, Berlin, 1996.

[Frä99] M. Fränzle. Analysis of hybrid systems: An ounce of realism can save an infinity of states. In *Proc. of CSL'99: Computer Science Logic, LNCS 1683*, pages 126–140. Springer-Verlag, Berlin, 1999.

[GHJ97] V. Gupta, T.A. Henzinger, and R. Jagadeesan. Robust timed automata. In *HART 97: Hybrid and Real-Time Systems*, Lecture Notes in Computer Science 1201, pages 331–345. Springer-Verlag, Berlin, 1997.

[Hen92] T.A. Henzinger. Sooner is safer than later. *Information Processing Letters*, 43:135–141, 1992.

[Hen95] T.A. Henzinger. Hybrid automata with finite bisimulations. In *ICALP 95: Automata, Languages, and Programming*, Lecture Notes in Computer Science 944, pages 324–335. Springer-Verlag, Berlin, 1995.

[Hen98] T.A. Henzinger. It's about time: Real-time logics reviewed. In *CONCUR 98: Concurrency Theory*, Lecture Notes in Computer Science 1466, pages 439–454. Springer-Verlag, Berlin, 1998.

[Hen00] T.A. Henzinger. The theory of hybrid automata. In M.K. Inan and R.P. Kurshan, editors, *Verification of Digital and Hybrid Systems*, NATO ASI

Series F: Computer and Systems Sciences 170, pages 265–292. Springer-Verlag, Berlin, 2000.

[HHM99] T.A. Henzinger, B. Horowitz, and R. Majumdar. Rectangular hybrid games. In *CONCUR 99: Concurrency Theory*, Lecture Notes in Computer Science 1664, pages 320–335. Springer-Verlag, Berlin, 1999.

[HHWT97] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, pages 110–122, 1997.

[HHWT98] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. Algorithmic analysis of nonlinear hybrid systems. *IEEE Transactions on Automatic Control*, 43:540–554, 1998.

[HK96] T.A. Henzinger and P.W. Kopke. State equivalences for rectangular hybrid automata. In *CONCUR 96: Concurrency Theory*, Lecture Notes in Computer Science 1119, pages 530–545. Springer-Verlag, Berlin, 1996.

[HK99] T.A. Henzinger and P.W. Kopke. Discrete-time control for rectangular hybrid automata. *Theoretical Computer Science*, 221:369–392, 1999.

[HKPV98] T.A. Henzinger, P.W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57:94–124, 1998.

[HM00] T.A. Henzinger and R. Majumdar. A classification of symbolic transition systems. In *STACS 00: Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science 1770, pages 13–34. Springer-Verlag, Berlin, 2000.

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

[HR00] T.A. Henzinger and J.-F. Raskin. Robust undecidability of timed and hybrid systems. In *HSCC 00: Hybrid Systems—Computation and Control*, Lecture Notes in Computer Science 1790, pages 145–159. Springer-Verlag, Berlin, 2000.

[HRP94] N. Halbwachs, P. Raymond, and Y.-E. Proy. Verification of linear hybrid systems by means of convex approximation. In B. LeCharlier, editor, *SAS 94: Static Analysis Symposium*, Lecture Notes in Computer Science 864, pages 223–237. Springer-Verlag, Berlin, 1994.

[HRS98] T.A. Henzinger, J.-F. Raskin, and P.-Y. Schobbens. The regular real-time languages. In *ICALP 98: Automata, Languages, and Programming*, Lecture Notes in Computer Science 1443, pages 580–591. Springer-Verlag, Berlin, 1998.

[HS74] M.W. Hirsh and S. Smale. *Differential Equations, Dynamical Systems and Linear Algebra.* Academic Press, 1974.

[HWT96] T.A. Henzinger and H. Wong-Toi. Using HyTech to synthesize control parameters for a steam boiler. In *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*, Lecture Notes in Computer Science 1165, pages 265–282. Springer-Verlag, Berlin, 1996.

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

[Mil71] R. Milner. An algebraic definition of simulation between programs. In *Second International Joint Conference on Artificial Intelligence*, pages 481–489. The British Computer Society, 1971.

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

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE Computer Society Press, Washington D.C., 1977.

[PV94] A. Puri and P. Varaiya. Decidability of hybrid systems with rectangular differential inclusions. In D.L. Dill, editor, *CAV 94: Computer-aided Verification*, Lecture Notes in Computer Science 818, pages 95–104. Springer-Verlag, Berlin, 1994.

[Ras99] J.-F. Raskin. *Logics, Automata, and Classical Theories for Deciding Real Time.* PhD thesis, Facultés Universitaires Notre-Dame de la Paix, Namur, Belgium, 1999.

[RS97] J.-F. Raskin and P.-Y. Schobbens. State-clock logic: a decidable real-time logic. In O. Maler, editor, *HART 97: Hybrid and Real-time Systems*, Lecture Notes in Computer Science 1201, pages 33–47. Springer-Verlag, Berlin, 1997.

[SMF97] T. Stauner, O. Mueller, and M. Fuchs. Using HYTECH to verify an automotive control system. In O. Maler, editor, *Hybrid and Real-Time Systems*, LNCS 1201, pages 139–153, Grenoble, France, 1997. Springer Verlag, Berlin.

[Tom98] C. Tomlin. *Hybrid Control of Air Traffic Management Systems.* PhD thesis, University of California at Berkeley, 1998.

[Won97] H. Wong-Toi. The synthesis of controllers for linear hybrid automata. In *Proc. 36th Conference on Decision and Control*, pages 4607–4612. IEEE Press, New-York, 1997.