HyLTL: a temporal logic for model checking hybrid systems

Davide Bresolin
Università degli Studi di Verona
Dipartimento di Informatica
Verona, Italy
davide.bresolin@univr.it

The model-checking problem for hybrid systems is a well known challenge in the scientific community. Most of the existing approaches and tools are limited to safety properties only, or operates by transforming the hybrid system to be verified into a discrete one, thus loosing information on the continuous dynamics of the system. In this paper we present a logic for specifying complex properties of hybrid systems called HyLTL, and we show how it is possible to solve the model checking problem by translating the formula into an equivalent hybrid automaton. In this way the problem is reduced to a reachability problem on hybrid automata that can be solved by using existing tools.

1 Introduction

Hybrid systems exhibit both a discrete and a continuous behaviour with a tight interaction between the two. Typical examples include discrete controllers that operate in a continuous environment, like automotive power-train systems, where a four stroke engine is modeled by a switching continuous system and is controlled by a digital controller. In order to model and specify hybrid systems in a formal way, the notion of *hybrid automata* has been introduced [1, 9, 14]. Intuitively, a hybrid automaton is a "finite-state automaton" with continuous variables that evolve according to dynamics characterizing each discrete state (called a *location* or *mode*). Of particular importance in the analysis of hybrid automata is the *model checking problem*, that is, the problem of verifying whether a given hybrid automaton respects some property of interest. The state of a hybrid automaton consists of the pairing of a discrete location with a vector of continuous variables, therefore it has the cardinality of continuum. This makes the model checking problem computationally difficult. Indeed, even for very simple properties and systems, this problem is not decidable [10].

Many different approaches have been used in the literature to tackle this problems. For simple classes of hybrid systems, like timed automata, the model checking problem can be solved exactly [2], and tools like Kronos [18] and UPPAAL [12] can be used to verify CTL properties of timed automata. For more complex classes of systems, the problem became undecidable. Nevertheless, many different approximation techniques may be used to obtain an answer, at least in some cases. Tools like PhaVer [6] and SpaceEx [7] can compute approximations of the reachable set of hybrid automata with linear dynamics, and thus can be used to verify safety properties. Other tools, like HSOLVER [17], and Ariadne [3], can manage also systems with nonlinear dynamics, but are still limited to safety properties only.

We are aware of only very few approaches that can specify and verify complex properties of hybrid systems in a systematic way. A first attempt was made in [11], where an extension of the Temporal Logic of Actions called TLA+ is used to specify and implement the well-known gas burner example. Later on, Signal Temporal Logic (STL), an extension of the well-known Metric Interval Logic to hybrid traces, has been introduced to monitor hybrid and continuous systems [15]. More recent approaches include the tool KeYmaera [16], that uses automated theorem proving techniques to verify nonlinear hybrid systems

Bortolussi, Bujorianu, Pola (Eds.): HAS 2013
EPTCS 124, 2013, pp. 73–84, doi:10.4204/EPTCS.124.8

symbolically, and the logic HRELTL [4], that is supported by an extension of the discrete model checker NuSMV, but it is limited to systems with linear dynamics.

In this paper we present an alternative approach for model checking hybrid systems. We define a logic for specifying properties called HyLTL, that extends the well-known LTL logic to hybrid traces. Then, we show how it is possible to translate a formula of this logic into an equivalent hybrid automaton. In this way the model checking problem is reduced to a reachability problem on the composition of the automaton representing the system and the one representing the (negation of the) formula, and can be solved by using existing tools. An example of verification of a simple property using PhaVer is given to show the feasibility of the approach.

1.1 Preliminaries

Before formally defining hybrid automata and the syntax and semantics of HyLTL we need to introduce some basic terminology. Throughout the paper we fix the *time axis* to be the set of non-negative real numbers $\mathbb{R}^+$. An *interval* I is any convex subset of $\mathbb{R}^+$, usually denoted as $[t_1,t_2] = \{t \in \mathbb{R}^+ : t_1 \leq t \leq t_2\}$.

We also fix a countable universal set $\mathcal{V}$ of *variables*, ranging over the reals. Given a set of variables $X \subseteq \mathcal{V}$, a *valuation* over X is a function $\mathbf{x} : X \mapsto \mathbb{R}$ that associates a value to every variable in X. The set $\mathrm{Val}(X)$ is the set of all valuations over X. Given a valuation $\mathbf{x}$ and a subset of variables $Y \subseteq X$, we denote the *restriction* of $\mathbf{x}$ to Y as $\mathbf{x}{\downarrow}Y$. The restriction operator is extended to sets of valuations in the usual way. A valuation $\mathbf{x}$ over X and a valuation $\mathbf{y}$ over Y *agree* when they assign the same value to common variables, i.e., $\mathbf{x}{\downarrow}X \cap Y = \mathbf{y}{\downarrow}X \cap Y$. When valuations $\mathbf{x}$ over X and $\mathbf{y}$ over Y agree, we denote by $\mathbf{x} \sqcup \mathbf{y}$ the *union* of $\mathbf{x}$ and $\mathbf{y}$, defined as the valuation $\mathbf{z}$ such that $\mathbf{z}{\downarrow}X = \mathbf{x}$ and $\mathbf{z}{\downarrow}Y = \mathbf{y}$. Notice that valuations over disjoint sets of variables always agree, and thus their union is always defined.

A notion that will play an important role in the paper is the one of *trajectory*. A trajectory over a set of variables X is a *differentiable* function $\tau : I \mapsto \mathrm{Val}(X)$, where I is a left-closed interval with left endpoint equal to 0. Since τ is differentiable, its derivative is defined in every point of the domain but the endpoints: we denote with $\dot{\tau}$ the corresponding function giving the value of the derivative of τ for every point in the interior of I (note that $\dot{\tau}$ might not be differentiable neither continuous). With $\mathrm{dom}(\tau)$ we denote the domain of τ, while with $\tau.ltime$ (the *limit time* of τ) we define the supremum of $\mathrm{dom}(\tau)$. The *first state* of a trajectory is $\tau.fstate = \tau(0)$, while, when $\mathrm{dom}(\tau)$ is right-closed, the *last state* of a trajectory is defined as $\tau.lstate = \tau(\tau.ltime)$. We denote with $Trajs(X)$ the set of all trajectories over X. Given a subset $Y \subseteq X$, the *restriction* of τ to Y is denoted as $\tau{\downarrow}Y$ and it is defined as the trajectory $\tau' : \mathrm{dom}(\tau) \mapsto \mathrm{Val}(Y)$ such that $\tau'(t) = \tau(t){\downarrow}Y$ for every $t \in \mathrm{dom}(\tau)$.

Variables will be used in the paper also to build *constraints*: conditions on the value of variables and on their derivative that can define sets of valuations, sets of trajectories, and jump relations. Formally, given a set of variables X, and a set of mathematical operators OP (e.g. $+$, $-$, $\cdot$, exponentiation, sin, cos, ...), we define the corresponded set of *dotted variables* $\dot{X}$ as $\{\dot{x}|x \in X\}$ and the set of *primed variables* X' as $\{x'|x \in X\}$. We use OP, X, $\dot{X}$ and X' to define the following two classes of constraints.

- *Jump constraints*: expressions built up from variables in $X \cup X'$, constants from $\mathbb{R}$, mathematical operators from OP and the usual equality and inequality relations ($\leq$, $=$, $>$, ...). Examples of jump constraints are $x' = 4y + z$, $x^2 \leq y'$, $y' > \cos(y)$.
- *Flow constraints*: expressions built up from variables in $X \cup \dot{X}$, constants from $\mathbb{R}$, mathematical operators from OP and the usual equality and inequality relations ($\leq$, $=$, $>$, ...). Examples of flow constraints are $\dot{x} = 4y + z$, $x' + y \geq 0$, $\sin(x) > \cos(y')$.

We use jump constraints to give conditions on pairs of valuations $(\mathbf{x}, \mathbf{x}')$. Given a jump constraint c, we say that $(\mathbf{x}, \mathbf{x}')$ respects c, and we denote it with $(\mathbf{x}, \mathbf{x}') \vdash c$, when, by replacing every variable x with its value in $\mathbf{x}$ and every primed variable x' with the value of the unprimed variable in $\mathbf{x}'$ we obtain a solution for c.

Flow constraints will be used to give conditions on trajectories. Given a flow constraint c, we say that a trajectory τ respects c, and we denote it with $\tau \vdash c$, if and only if for every time instant $t \in \mathrm{dom}(\tau)$, both the value of the trajectory $\tau(t)$ and the value of its derivative $\dot{\tau}(t)$ respect c (we assume that $\dot{\tau}(t)$ always respects c for $t = 0$ and $t = \tau.ltime$).

To conclude the preliminary section, we recall from the introduction that hybrid systems interleaves continuous and discrete evolution. For this reason their behaviors are usually defined in therms of *hybrid traces*, mixing continuous trajectories with discrete events. Formally, given a (possibly finite) set of *actions* (or discrete events) A and a (possibly finite) set of variables X, an *hybrid trace over A and X* is any infinite sequence $\alpha = \tau_1 a_1 \tau_2 a_2 \tau_3 a_3 \ldots$ such that τ_i is a trajectory over X and a_i is an action in A for every $i \geq 1$.

Notice that this definition of hybrid traces allows an infinite sequence of discrete events to occur in a finite amount of time (Zeno behaviors). This will not be a problem for the semantics of the logic nor for the correctness of the model checking algorithm. Zeno behaviors are usually not desirable in real applications, but are very difficult to exclude completely from the language and the formal model of the system. For this reason we choose to include Zeno behaviors in the semantics of our logic at a first stage, leaving a more comprehensive study of this aspect as future work.

2 Hybrid Automata

An hybrid automaton is a finite state machine enriched with continuous dynamics labelling each discrete state (or *location*), that alternates continuous and discrete evolution. In continuous evolution, the discrete state does not change, while time passes and the evolution of the continuous state variables follows the dynamic law associated to the current location. A discrete evolution step consists of the activation of a *discrete transition* that can change both the current location and the value of the state variables, in accordance with the reset function associated to the transition.

The definition of hybrid automata given in this paper extends the one given in [1] to support the parallel composition of automata. Usually, compositional formalisms introduce either a distinction between internal variables and actions (that are hidden to the other automata) and external ones, that are shared between automata (possibly with a further partition between inputs and outputs) [13]. This is usually justified by the need to describe in a precise way how a component interacts with the environment and the other components of the system. However, in this paper we are more interested in the analysis of a single component on its own, and this distinction between "local" and "global" variables and actions is not strictly necessary. Hence, to simplify the definitions and the proofs we choose to consider all variables and actions as shared between the different automata.

Definition 1. *A* hybrid automaton *is a tuple* $\mathcal{H} = \langle \mathrm{Loc}, X, A, \mathrm{Edg}, \mathrm{Dyn}, \mathrm{Res}, \mathrm{Init} \rangle$ *such that:*

1. Loc *is a finite set of* locations*;*
2. X *is a finite set of* variables*;*
3. A *is a finite set of* actions*;*
4. $\mathrm{Edg} \subseteq \mathrm{Loc} \times A \times \mathrm{Loc}$ *is a set of* discrete transitions*;*
5. Dyn *is a mapping that associates to every location* $\ell \in \mathrm{Loc}$ *a set of flow constraints* $\mathrm{Dyn}(\ell)$ *over* $X \cup \dot{X}$ *describing the* dynamics *of* ℓ*;*

6. Res *is a mapping that associates every discrete transition* $(\ell, e, \ell') \in \text{Edg}$ *with a set of jump constraints* $\text{Res}(\ell, e, \ell')$ *over* $X \cup X'$ *describing the guard and reset function of the transition;*
7. $\text{Init} \subseteq \text{Loc}$ *is a set of* initial locations.

Composition is defined as a binary operation on hybrid automata.

Definition 2. *Given two hybrid automata* $\mathcal{H}_1 = \langle \text{Loc}_2, X_1, A_1, \text{Edg}_1, \text{Dyn}_1, \text{Res}_1, \text{Init}_1 \rangle$ *and* $\mathcal{H}_2 = \langle \text{Loc}_2, X_2, A_2, \text{Edg}_2, \text{Dyn}_2, \text{Res}_2, \text{Init}_2 \rangle$*, we define their composition* $\mathcal{H}_1 \| \mathcal{H}_2$ *as the hybrid automaton* $\mathcal{H} = \langle \text{Loc}, X, A, \text{Edg}, \text{Dyn}, \text{Res}, \text{Init} \rangle$ *such that:*

1. $\text{Loc} = \text{Loc}_1 \times \text{Loc}_2$*;*
2. $X = X_1 \cup X_2$*;*
3. $A = A_1 \cup A_2$*;*
4. $((\ell_1, \ell_2), a, (\ell_1', \ell_2')) \in \text{Edg}$ *iff (i)* $a \in A_i$ *and* $(\ell_i, a, \ell_i') \in \text{Edg}_i$*, or (ii)* $a \notin A_i$ *and* $\ell_i = \ell_i'$*, for* $i = 1, 2$*;*
5. *for every* $(\ell_1, \ell_2) \in \text{Loc}$ *we have that* $\text{Dyn}((\ell_1, \ell_2)) = \text{Dyn}_1(\ell_1) \cup \text{Dyn}_2(\ell_2)$*;*
6. *for every* $((\ell_1, \ell_2), a, (\ell_1', \ell_2')) \in \text{Edg}$*, we have that* $\text{Res}((\ell_1, \ell_2), a, (\ell_1', \ell_2'))$ *is the minimal set of jump constraints such that, for* $i = 1, 2$*:*
   (i) if $a \in A_i$ *then* $\text{Res}_i(\ell_i, a, \ell_i') \subseteq \text{Res}((\ell_1, \ell_2), a, (\ell_1', \ell_2'))$*, and*
   (ii) if $a \notin A_i$ *then* $\{x = x' | x \in X_i \setminus X_{3-i}\} \subseteq \text{Res}((\ell_1, \ell_2), a, (\ell_1', \ell_2'))$*;*
7. $\text{Init} = \text{Init}_1 \times \text{Init}_2$.

The *state* of an hybrid automaton $\mathcal{H}$ is a pair $(\ell, \mathbf{x})$, where $\ell \in \text{Loc}$ is a location and $\mathbf{x} \in \text{Val}(X)$ is a valuation for the continuous variables. A state $(\ell, \mathbf{x})$ is said to be *admissible* if $(\ell, \mathbf{x}) \vdash \text{Dyn}(\ell)$. Transitions can be either *continuous*, capturing the continuous evolution of the state, or *discrete*, capturing instantaneous and discontinuous changes of the state. Formally, they are defined as follows.

Definition 3. *Let* $\mathcal{H}$ *be a hybrid automaton. The* continuous transition relation $\xrightarrow{\tau}$ *between admissible states, where* τ *is a bounded trajectory over X, is defined as follows:*

$$(\ell, \mathbf{x}) \xrightarrow{\tau} (\ell, \mathbf{x}') \iff \tau.fstate = \mathbf{x},\ \tau.lstate = \mathbf{x}',\ \text{and}\ \tau \vdash \text{Dyn}(\ell). \tag{1}$$

The discrete transition relation $\xrightarrow{e}$ *between admissible states, where* $a \in A$*, is defined as follows:*

$$(\ell, \mathbf{x}) \xrightarrow{a} (\ell', \mathbf{x}') \iff (\ell, a, \ell') \in \text{Edg} \wedge \mathbf{x} \vdash \text{Dyn}(\ell) \wedge \mathbf{x}' \vdash \text{Dyn}(\ell') \wedge (\mathbf{x}, \mathbf{x}') \vdash \text{Res}(\ell, a, \ell'). \tag{2}$$

This semantics allows us to define the notion of hybrid traces *generated* by the hybrid automaton $\mathcal{H}$.

Definition 4. *Let* $\mathcal{H}$ *be a hybrid automaton, and let* $\alpha = \tau_1 a_1 \tau_2 a_2 \ldots$ *be a (finite or infinite) hybrid trace over X and A. We say that* α *is* generated *by* $\mathcal{H}$ *if there exists a corresponding sequence of locations* $\ell_1 \ell_2 \ldots$ *such that* $\ell_1 \in \text{Init}$ *and, for every* $i \geq 1$*: (i)* $(\ell_i, \tau_i.fstate) \xrightarrow{\tau_i} (\ell_i, \tau_i.lstate)$*, and (ii)* $(\ell_i, \tau_i.lstate) \xrightarrow{a_i} (\ell_{i+1}, \tau_{i+1}.fstate)$.

Figure 1 depicts an example of a simple hybrid automaton that models a thermostat. The variable x represents the temperature of the room. In the location *idle* the heater is off and temperature decreases according to the flow condition $\dot{x} = -0.2x$. The automaton is allowed to stay in *idle* while the temperature is equal or greater to 17, and can jump to *heat* as soon as the temperature decrease under 19 degrees. In location *heat* the heater is turned on and the temperature increases according to the flow condition $\dot{x} = 30 - 0.2x$. The automaton is allowed to stay in *heat* until the temperature is less or equal to 23, and can jump to *idle* when it is greater than 21. Initially, the heater is off.

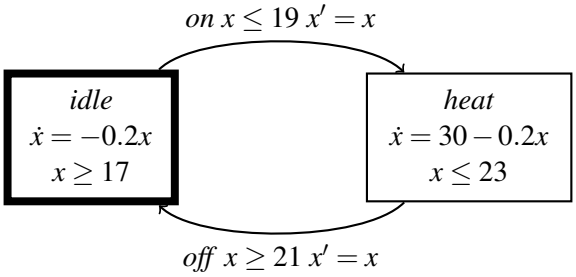

Figure 1: The thermostat automaton $\mathcal{H}_T$.

3 HyLTL: syntax and semantics

The logic HyLTL is an extension of the well-known temporal logic LTL to hybrid traces. Given a *finite* set of actions A and a *finite* set of variables X, the language of HyLTL is defined from a set of *flow constraints* CS over X by the following grammar:

$$\varphi ::= c \in CS \mid a \in A \mid \top \mid \bot \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \varphi\,\mathbf{U}\,\varphi \mid \varphi\,\mathbf{R}\,\varphi \tag{3}$$

In HyLTL flow constraints from CS and actions from A take the role of propositional letters in standard temporal logics, $\top$ and $\bot$ are the logical constants *true* and *false*, $\neg$, $\wedge$ and $\vee$ are the usual boolean connectives, **X**, **U** and **R** are hybrid counterpart of the standard *next*, *until* and *release* temporal operators.

Let $\alpha = \tau_1 a_1 \tau_2 a_2 \ldots$ be an infinite hybrid trace. For every $i > 0$, the truth value of a HyLTL formula φ over α at position i is given by the truth relation $\Vdash$, formally defined as follows:

- for every $c \in CS$, $\alpha, i \Vdash c$ iff $\tau_i \vdash c$;
- for every $a \in A$, $\alpha, i \Vdash a$ iff $i > 1$ and $a_{i-1} = a$;
- $\alpha, i \Vdash \top$ and $\alpha, i \nVdash \bot$;
- $\alpha, i \Vdash \neg\varphi$ iff $\alpha, i \nVdash \varphi$;
- $\alpha, i \Vdash \varphi \wedge \psi$ iff $\alpha, i \Vdash \varphi$ and $\alpha, i \Vdash \psi$;
- $\alpha, i \Vdash \varphi \vee \psi$ iff $\alpha, i \Vdash \varphi$ or $\alpha, i \Vdash \psi$;
- $\alpha, i \Vdash \mathbf{X}\varphi$ iff $\alpha, i+1 \Vdash \varphi$;
- $\alpha, i \Vdash \varphi\,\mathbf{U}\,\psi$ iff there exists $j \geq i$ such that $\alpha, i \Vdash \psi$, and for every $i \leq k < j$, $\alpha, k \Vdash \varphi$;
- $\alpha, i \Vdash \varphi\,\mathbf{R}\,\psi$ iff for all $j \geq i$, if for every $i \leq k < j$, $\alpha, k \nVdash \varphi$ then $\alpha, j \Vdash \psi$.

Intuitively, the operator **X** ("next discrete action") requires that some formula hold after the next discrete action. The operator **U** forces a property to hold until some other property holds.

The next and until operator are sufficient to express other useful temporal operators, namely the "always" operator **G** and the "eventually" operator **F**. They can be defined as usual:

$$\mathbf{F}\varphi = \top\,\mathbf{U}\,\varphi \qquad\qquad \mathbf{G}\varphi = \neg\,\mathbf{F}\,\neg\varphi$$

Using the base and derived operators of HyLTL it is possible to express a number of different properties, including safety properties, liveness properties, reactivity properties and so on. For instance, in the case

of the Thermostat example described in Figure 1, the usual property of *"keeping the temperature of the room between 15 and 25 degrees"* is defined by the following formula:

$$\varphi_{safe} = \mathbf{G}\,(x \geq 15 \wedge x \leq 25) \tag{4}$$

Safety properties are the most common type of requirements on hybrid systems, but HyLTL can easily go beyond them. For example, it can express reactivity properties like *"whenever the heater is turned ON it is eventually turned OFF"*:

$$\varphi_{react} = \mathbf{G}\,(on \rightarrow \mathbf{F}off) \tag{5}$$

Finally, the presence of discrete actions and of flow constraints in the syntax of HyLTL allow this logic to express properties involving both the discrete and the continuous behaviors of the system, thus capturing their hybrid nature. An example on the thermostat can be *"it is not possible that the heater turns on when the temperature is above 21 degrees"*:

$$\varphi_{hyb} = \neg\,\mathbf{F}\,(x \geq 21 \wedge \mathbf{X}on) \tag{6}$$

4 From HyLTL to Hybrid Automata with Büchi conditions

In this section we show how to translate a HyLTL formula into an equivalent hybrid automaton, that is, an hybrid automaton representing all traces that satisfy the formula. In this paper we use a declarative construction, similar to the tableau construction presented in [5] for model checking of LTL formulas.

In analogy with the classical construction, we have to enrich the notion of hybrid automata with a suitable *acceptance condition* to identify the traces generated by the automaton that fulfills the semantics of the until operator. To this end, we mark some of the locations of the hybrid automaton as *final location* and we extend Definition 1 with a *Generalized Büchi accepting condition* for final locations.

Definition 5. *A* Hybrid Automaton with Generalized Büchi condition (GBHA) *is a tuple* $\mathcal{H} = \langle \mathrm{Loc}, X, A, \mathrm{Edg}, \mathrm{Dyn}, \mathrm{Res}, \mathrm{Init}, \mathcal{F} \rangle$ *such that* $\langle \mathrm{Loc}, X, A, \mathrm{Edg}, \mathrm{Dyn}, \mathrm{Res}, \mathrm{Init} \rangle$ *is a Hybrid Automaton, and* $\mathcal{F} = \{F_1, \ldots, F_n\} \subseteq 2^{\mathrm{Loc}}$ *is a finite set of sets of final locations.*

We say that an hybrid trace $\alpha = \tau_1 a_1 \tau_2 a_2 \ldots$ is *accepted* by a GBHA $\mathcal{H}$ if there exists an *infinite* sequence of locations $\ell_1 \ell_2 \ldots$ such that:

(i) $\ell_1 \in \mathrm{Init}$;
(ii) for every $i \geq 1$, $(\ell_i, \tau_i.fstate) \xrightarrow{\tau_i} (\ell_i, \tau_i.lstate)$;
(iii) for every $i \geq 1$, $(\ell_i, \tau_i.lstate) \xrightarrow{a_i} (\ell_{i+1}, \tau_{i+1}.fstate)$;
(iv) for every $F_j \in \mathcal{F}$ there exists $f_j \in F_j$ that occurs infinitely often in the sequence.

Notice that α must be a sequence generated by $\mathcal{H}$. However, not all sequences generated by the automaton are accepting: only those that respect the additional accepting condition are considered.

Given a HyLTL formula φ, we show now how to build a GBHA that accepts exactly all hybrid traces that satisfies φ. As in the case of LTL, we assume that the formula is in negated normal form, and we define the notion of *closure* of a HyLTL formula as follows.

Definition 6. *Give an HyLTL formula* φ *and a set of actions A we define the* closure of φ *as the smallest set* $\mathrm{cl}(\varphi)$ *of formulas satisfying the following conditions:*

- $\varphi \in \mathrm{cl}(\varphi)$;
- *for each* $a \in A$, $a \in \mathrm{cl}(\varphi)$;
- *if* $\psi_1 \in \mathrm{cl}(\varphi)$ *then* $\neg\psi_1 \in \mathrm{cl}(\varphi)$;
- *if* $\psi_1 \wedge \psi_2 \in \mathrm{cl}(\varphi)$ *then* $\psi_1, \psi_2 \in \mathrm{cl}(\varphi)$;
- *if* $\psi_1 \vee \psi_2 \in \mathrm{cl}(\varphi)$ *then* $\psi_1, \psi_2 \in \mathrm{cl}(\varphi)$;
- *if* $\mathbf{X}\psi_1 \in \mathrm{cl}(\varphi)$ *then* $\psi_1 \in \mathrm{cl}(\varphi)$;
- *if* $\psi_1 \,\mathbf{U}\, \psi_2 \in \mathrm{cl}(\varphi)$ *then* $\psi_1, \psi_2 \in \mathrm{cl}(\varphi)$;
- *if* $\psi_1 \,\mathbf{R}\, \psi_2 \in \mathrm{cl}(\varphi)$ *then* $\psi_1, \psi_2 \in \mathrm{cl}(\varphi)$;

The subsets of $\mathrm{cl}(\varphi)$ will label the locations of the hybrid automaton. We aim to construct an automaton such that if a location is labelled by a subset $M \subseteq \mathrm{cl}(\varphi)$ then every accepting run starting from that location satisfies all formulas in M. For this reason, we can ignore every subset of the closure that is clearly inconsistent or subsumed by a consistent superset of formulas, and concentrate our attention to *maximally consistent* subsets of $\mathrm{cl}(\varphi)$ defined as follows.

Definition 7. *A subset* $M \subseteq \mathrm{cl}(\varphi)$ *is* maximally consistent *if it respects the following conditions:*

1. $\top \in M$;
2. $\psi_1 \in M$ *iff* $\neg\psi_1 \notin M$;
3. $\psi_1 \wedge \psi_2 \in M$ *iff* $\psi_1 \in M$ *and* $\psi_2 \in M$;
4. $\psi_1 \vee \psi_2 \in M$ *iff* $\psi_1 \in M$ *or* $\psi_2 \in M$;
5. *if* $a \in M$ *then for each* $b \neq a$, $b \notin M$.

The first four conditions are the usual boolean consistency conditions, while conditions 5 guarantees that actions in A are mutually exclusive. Let $\mathrm{mc}(\varphi)$ be the set of maximally consistent subsets of $\mathrm{cl}(\varphi)$. We are going to use $\mathrm{mc}(\varphi)$ as the location of the GBHA representing φ.

Definition 8. *Let* φ *be a HyLTL formula over a set of variables* X *and actions* A. *The corresponding GBHA* $\mathcal{H}_\varphi = \langle \mathrm{Loc}, X, A, \mathrm{Edg}, \mathrm{Dyn}, \mathrm{Res}, \mathrm{Init}, \mathcal{F} \rangle$ *is defined as follows:*

- $\mathrm{Loc} = \mathrm{mc}(\varphi)$;
- *for each* $M, M' \in \mathrm{mc}(\varphi)$ *and* $a \in A$, $(M, a, M') \in \mathrm{Edg}$ *if and only if:*
  - $a \in M'$;
  - $\mathbf{X}\psi_1 \in M$ *iff* $\psi_1 \in M'$;
  - $\psi_1 \,\mathbf{U}\, \psi_2 \in M$ *iff* $\psi_2 \in M$ *or* ($\psi_1 \in M$ *and* $\psi_1 \,\mathbf{U}\, \psi_2 \in M'$);
  - $\psi_1 \,\mathbf{R}\, \psi_2 \in M$ *iff* $\psi_1, \psi_2 \in M$ *or* ($\psi_2 \in M$ *and* $\psi_1 \,\mathbf{R}\, \psi_2 \in M'$);
- *for each* $M \in \mathrm{mc}(\varphi)$, $\mathrm{Dyn}(M) = \{c \in CS | c \in M\}$ *(i.e., the set of flow constraints of* M*)*;
- *for each* $(M, a, M') \in \mathrm{Edg}$, $\mathrm{Res}(M, a, M') = \top$;
- $\mathrm{Init} = \{M \in \mathrm{mc}(\varphi) | \varphi \in M \text{ and } M \cap A = \emptyset\}$;
- *for each* $\psi_1 \,\mathbf{U}\, \psi_2 \in \mathrm{cl}(\varphi)$, $\{M \in \mathrm{mc}(\varphi) | \psi_2 \in M \text{ or } \neg(\psi_1 \,\mathbf{U}\, \psi_2) \in M\} \in \mathcal{F}$.

The conditions on the set of discrete transitions Edg guarantee that any run of the automaton does not violate the semantics of the temporal operators. The accepting condition $\mathcal{F}$ is a set of sets of locations, where every set in $\mathcal{F}$ corresponds to an until-formula in $\mathrm{cl}(\varphi)$. They guarantee that whenever a formula $\psi_1 \,\mathbf{U}\, \psi_2$ become true in some location, then the promise ψ_2 is eventually fulfilled later on in the run. The following theorem proves that $\mathcal{H}_\varphi$ accepts exactly all the hybrid traces satisfying φ.

Theorem 1. *Let* φ *be HyLTL-formula, and* $\alpha = \tau_1 a_1 \tau_2 a_2 \ldots$ *be an infinite hybrid trace over* X *and* A. *Then* $\alpha, 1 \Vdash \varphi$ *if and only if* $\mathcal{H}_\varphi$ *accepts* α.

Proof. Suppose that $\alpha, 1 \Vdash \varphi$. For every $i \geq 1$, let $M_i = \{\psi \in \mathrm{cl}(\varphi) | \alpha, i \Vdash \psi\}$. It is easy to see that every M_i is a maximal consistent set of formulas. Moreover, by the definition of $\mathcal{H}_\varphi$, we have that $M_1 \in \mathrm{Init}$ and that, for every $i \geq 1$, $\tau_i \vdash \mathrm{Dyn}(M_i)$, $(M_i, a_i, M_{i+1}) \in \mathrm{Edg}$, and $(\tau_i.lstate, \tau_{i+1}.fstate) \vdash \mathrm{Res}(M_i, a_i, M_{i+1})$.

Hence, α is generated by $\mathcal{H}_\varphi$. To show that α is also accepted by the automaton, let $\psi_1 \,\mathbf{U}\, \psi_2$ be an until formula in $\mathrm{cl}(\varphi)$. Two cases may arise: either there exists some index j such that for all $k \geq j$, $\alpha, k \not\Vdash \psi_1 \,\mathbf{U}\, \psi_2$; or there exists an infinite set of indexes $j_1, j_2, \ldots$ such that $\alpha, j_k \Vdash \psi_1 \,\mathbf{U}\, \psi_2$. In the former case, all maximal consistent sets M_k are such that $\neg(\psi_1 \,\mathbf{U}\, \psi_2) \in M_k$. In the latter case, by the semantics of HyLTL, for each j_k there must exists an index h_k such that $\alpha, h_k \Vdash \psi_2$: this implies that $\psi_2 \in M_{h_k}$. In both

cases there exists a final location $f \in \{M \in \mathrm{mc}(\varphi) | \psi_2 \in M \text{ or } \neg(\psi_1 \mathbf{U} \psi_2) \in M\}$ that occurs infinitely often in the run of the automaton. This proves that α is accepted by $\mathcal{H}$.

Conversely, suppose that α is accepted by $\mathcal{H}_\varphi$, and let $M_1 M_2 \ldots$ be an accepting sequence of locations for α. We prove the claim by showing that the following stronger property holds:

$$\textit{for every } i \geq 1 \textit{ and for every } \psi \in \mathrm{cl}(\varphi),\ \psi \in M_i \textit{ if and only if } \alpha, i \Vdash \psi. \tag{7}$$

We proceed by induction on the structure of ψ.

- $\varphi = c$, with $c = \top$ or $c \in CS$. By the definition of $\mathcal{H}_\varphi$, $c \in \mathrm{Dyn}(M_i)$ and thus, by the definition of the continuous transition relation, $\tau_i \vdash c$.
- $\varphi = a$, for some $a \in A$. By the definition of $\mathcal{H}_\varphi$, $M_i \notin \mathrm{Init}$: this implies that $i > 1$. Consider now the action a_{i-1} in α: by the definition of $\mathcal{H}_\varphi$, we have that $a_{i-1} = a$ and thus $\alpha, i \Vdash a$.
- $\varphi = \psi_1 \wedge \psi_2$. By the definition of consistent set, $\psi_1 \wedge \psi_2 \in M_i$ if and only if $\psi_1 \in M_i$ and $\psi_2 \in M_i$. By the induction hypothesis, we have that $\alpha, i \Vdash \psi_1$ and $\alpha, i \Vdash \psi_2$. By the semantics of HyLTL, we have that $\alpha, i \Vdash \psi_1 \wedge \psi_2$.
- $\varphi = \psi_1 \vee \psi_2$ or $\varphi = \neg\psi_1$. This case is similar to the previous one and thus skipped.
- $\varphi = \mathbf{X}\psi_1$. By the definition of $\mathcal{H}_\varphi$, since $(M_i, a_i, M_{i+1}) \in \mathrm{Edg}$ we have that $\psi_1 \in M_{i+1}$. By inductive hypothesis, we have that $\alpha, i+1 \Vdash \psi_1$ and thus that $\alpha, i \Vdash \mathbf{X}\psi_1$.
- $\varphi = \psi_1 \mathbf{U} \psi_2$. Suppose that $\psi_1 \mathbf{U} \psi_2 \in M_i$. Then, by the definition of $\mathcal{H}_\varphi$, either $\psi_1 \in M_i$ or $\psi_2 \in M_i$. By inductive hypothesis, this implies that $\alpha, i \Vdash \psi_1$ or $\alpha, i \Vdash \psi_2$. In the latter case we can conclude that $\alpha, i \Vdash \psi_1 \mathbf{U} \psi_2$.

  In the former case, by the definition of the discrete transitions Edg, we have $\psi_1 \mathbf{U} \psi_2 \in M_{i+1}$. By the definition of the acceptance condition of $\mathcal{H}_\varphi$, we have that there must exists at least one index $j \geq i+1$ such that $\psi_2 \in M_j$. Let k be the smallest of those indexes: by induction hypothesis we have that $\alpha, k \Vdash \psi_2$. We can also prove by induction on m that, for every $i+1 \leq m < k$, $\psi_1, \psi_1 \mathbf{U} \psi_2 \in M_m$. When $m = i+1$, the proof is trivial. When $i+1 < m < k$ we have, by induction hypothesis, that $\psi_1 \mathbf{U} \psi_2 \in M_{m-1}$. Since $\psi_2 \notin M_{m-1}$, by the definition of Edg we have that $\psi_1 \mathbf{U} \psi_2 \in M_m$ and thus that $\psi_1 \in M_m$. This implies that, by inductive hypothesis on φ, that $\alpha, m \Vdash \psi_1$, and thus that $\alpha, i \Vdash \psi_1 \mathbf{U} \psi_2$.

  Now, suppose that $\alpha, i \Vdash \psi_1 \mathbf{U} \psi_2$. By the semantics of HyLTL, two cases may arise:

  - $\alpha, i \Vdash \psi_2$. By inductive hypothesis we have that $\psi_2 \in M_i$. By the definition of $\mathcal{H}_\varphi$, this implies that $\psi_1 \mathbf{U} \psi_2 \in M_i$.
  - $\alpha, i \Vdash \psi_1$ and $\alpha, i+1 \Vdash \psi_1 \mathbf{U} \psi_2$. By the semantics of HyLTL, there must exists at least one index $j \geq i+1$ such that $\psi_2 \in M_j$. Let k be the smallest of those indexes: by proceeding as above we can conclude that $\psi_1 \mathbf{U} \psi_2 \in M_i$.

- Finally, the case when $\varphi = \psi_1 \mathbf{R} \psi_2$ is similar to the previous one and can be skipped.

This concludes the proof of property (7). By definition of Init, $\varphi \in M_1$, and thus, by (7), we have proved that $\alpha \Vdash \varphi$, as required. □

5 Model checking HyLTL

In the previous section we have shown how to build an GBHA that is equivalent to a HyLTL formula. In this section we show how this can be exploited to solve the model checking problem for HyLTL.

Let $\mathcal{H}_S$ be a hybrid automaton representing the system under verification, and let φ be the HyLTL formula representing a property that the system should respect. Consider the GBHA $\mathcal{H}_{\neg\varphi}$ that is equivalent to *the negation of the property*: by Theorem 1, it accepts all the hybrid traces that *violates* the property we want to verify. Now, if we compose the automaton for the system with the automaton for $\neg\varphi$ we obtain a GBHA $\mathcal{H}_S \| \mathcal{H}_{\neg\varphi}$ such that:

- it generates only hybrid traces that are generated by $\mathcal{H}_S$;
- it accepts only hybrid traces that are accepted by $\mathcal{H}_{\neg\varphi}$.

Hence, $\mathcal{H}_S \| \mathcal{H}_{\neg\varphi}$ accepts only those hybrid traces that are generated by the system and violates the property. This means that $\mathcal{H}_S$ respects the property φ if and only $\mathcal{H}_S \| \mathcal{H}_{\neg\varphi}$ *does not accept any hybrid trace*.

This last property of $\mathcal{H}_S \| \mathcal{H}_{\neg\varphi}$ is a reachability property that can be tested by existing tools for the reachability analysis of hybrid automata, and thus allows to use existing technology to verify properties of HyLTL. The only thing that one need to do is to write a procedure implementing the construction of Definition 8 to build the automaton for the negation of the formula, and then send the results to the reachability analysis tool.

Feasibility of this approach has been tested by verifying the Thermostat example of Section 2 against the example formula $\varphi_{hyb} = \neg\,\mathbf{F}\,(x \geq 21 \wedge \mathbf{X}on)$ of Section 3, using the well-known software package PhaVer [6]. As a first step, it is necessary to build the automaton for $\neg\varphi_{hyb} = \mathbf{F}\,(x \geq 21 \wedge \mathbf{X}on) = \top\,\mathbf{U}\,(x \geq 21 \wedge \mathbf{X}on)$. The application of Definition 8 leads to the GBHA depicted in Figure 2, where initial locations have a thick border and final locations are grayed out. In this particular case there is only one until formula in the closure, so $\mathcal{F}$ is a singleton set. To simplify the picture, some of the formulas labeling the locations have been left out.

Since PhaVer allows the composition of hybrid automata, it was not necessary to compute the parallel composition of $\mathcal{H}_T$ and $\mathcal{H}_{\neg\varphi_{hyb}}$. However, PhaVer does not allow to specify acceptance conditions on hybrid automata. Hence, to search for an accepting run we have to modify the automaton for the formula as follows:

- we assigned to every final state a unique numerical value different from 0;
- we added two auxiliary variables, f and y, with initial value 0;
- the derivative of f and y are equal to 0 in all locations;
- when a transition exiting a final state is taken, and the current value of f is 0, f can be non-deterministically reset to the value associated to the final state. At the same time, the current value of x is stored in y;
- all other transitions leave the values of f and x unchanged.

In this way we exploit the non-determinism to guess the occurrence of a final location that will occur infinitely often in the accepting run of the automaton, and we can check its existence within PhaVer: it is sufficient to compute the set of reachable states of $\mathcal{H}_T \| \mathcal{H}_{\neg\varphi_{hyb}}$ and check whether there exists a state where the location of $\mathcal{H}_{\neg\varphi_{hyb}}$ is a final one, the value of f is equal to the numerical value of that location, and $y = x$. If this is not the case, then there are no accepting runs and the property is verified. If, on the other hand, such a state can be found, it may be the case that a loop from a final location to a final

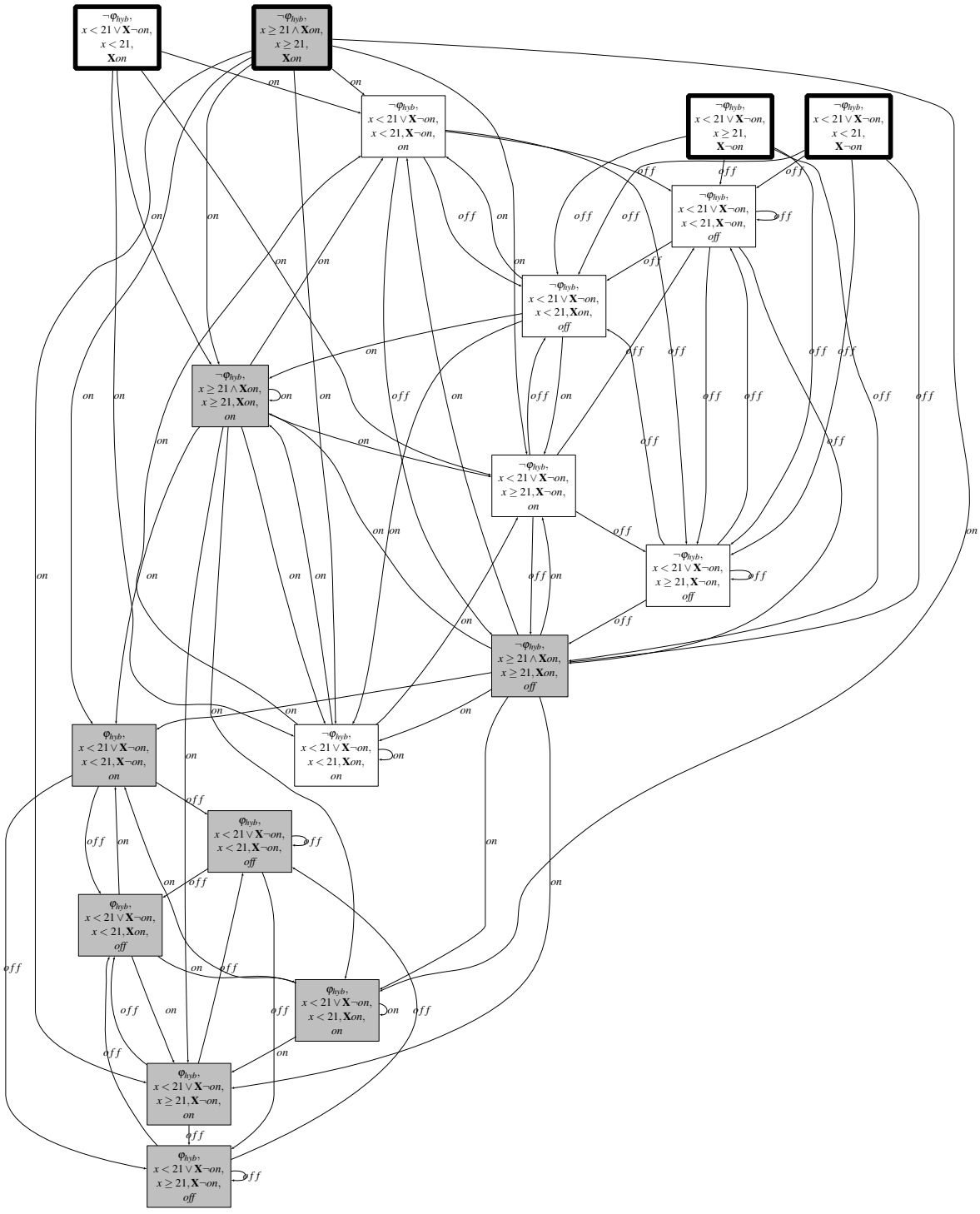

Figure 2: The GBHA for $\neg\varphi_{hyb}$.

location can be built. Notice that in the latter case we cannot conclude that the system falsify the required property: PhaVer computes an *over-approximation* of the exact reachable set of the system. Hence, it may be the case that no accepting run exists, but the test found a loop from a final location to a final due to over-approximation errors.

In the example given in this paper, PhaVer was able to correctly verify that the thermostat respect the property φ_{hyb}. In the case of this very simple example, the computation time was almost instantaneous (0.2 seconds on an Intel Core Duo 2.0 GHz notebook).

6 Conclusion

In this paper we presented HyLTL, a logic that is able to express properties of hybrid traces, and we have shown how to translate a formula of HyLTL to an equivalent hybrid automaton with Büchi acceptance conditions. In this way it is possible to solve the model checking for HyLTL by reducing it to a reachability problem on the composition of the hybrid automata representing the system with the one representing the (negation of the) formula. Feasibility of the approach has been tested by verifying a very simple example using the well-known tool PhaVer.

This is still a preliminary work, that can be extended in many directions. The expressivity of the logic can be extended by adding jump predicates to the language, to express properties on the reset functions of the system. The algorithm for computing the equivalent automaton can be improved by using an on-the-fly approach like the one used in [8] for LTL. Finally, tool support for the logic can be extended by customizing existing reachability analysis tools to manage final locations directly, without the need to introduce additional variables to the model.

References

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P. h. Ho, X. Nicollin, A. Olivero, J. Sifakis & S. Yovine (1995): *The Algorithmic Analysis of Hybrid Systems*. Theoretical Computer Science 138, pp. 3–34, doi:10.1016/0304-3975(94)00202-T.

[2] R. Alur & D. L. Dill (1994): *A Theory of Timed Automata*. Journal of Theoretical Computer Science 126(2), pp. 183–235, doi:10.1016/0304-3975(94)90010-8.

[3] L. Benvenuti, D. Bresolin, P. Collins, A. Ferrari, L. Geretti & T. Villa (2012): *Assume-guarantee verification of nonlinear hybrid systems with ARIADNE*. Int. J. Robust. Nonlinear Control, doi:10.1002/rnc.2914.

[4] A. Cimatti, M. Roveri & S. Tonetta (2009): *Requirements Validation for Hybrid Systems*. In: *CAV*, *LNCS* 5643, pp. 188–203, doi:10.1007/978-3-642-02658-4_17.

[5] E. M. Clarke, O. Grumberg & D. A. Peled (2000): *Model Checking*. MIT Press.

[6] G. Frehse (2008): *PHAVer: algorithmic verification of hybrid systems past HyTech*. International Journal on Software Tools for Technology Transfer (STTT) 10, pp. 263–279, doi:10.1007/s10009-007-0062-x.

[7] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang & O. Maler (2011): *SpaceEx: Scalable Verification of Hybrid Systems*. In: Proc. 23rd International Conference on Computer Aided Verification (CAV 2011), LNCS 6806, Springer Berlin / Heidelberg, pp. 379–395, doi:10.1007/978-3-642-22110-1_30.

[8] R. Gerth, D. Peled, M. Vardi & P. Wolper (1995): *Simple on-the-fly automatic verification of linear temporal logic*. In: Protocol Specification, Testing and Verification, IFIP Conference Proceedings 38, Chapman & Hall, pp. 3–18.

[9] T. A. Henzinger (2000): *The theory of hybrid automata*. In M. K. Inan & R. P. Kurshan, editors: *Verification of Digital and Hybrid Systems*, *NATO ASI Series F: Computer and Systems Sciences* 170, Springer, pp. 265–292, doi:10.1007/978-3-642-59615-5_13.

[10] T. A. Henzinger, P. W. Kopke, A. Puri & P. Varaiya (1998): *What's Decidable about Hybrid Automata?* *Journal of Computer and System Sciences* 57(1), pp. 94 – 124, doi:10.1006/jcss.1998.1581.

[11] L. Lamport (1993): *Hybrid systems in TLA+s*. In RobertL. Grossman, Anil Nerode, AndersP. Ravn & Hans Rischel, editors: *Hybrid Systems*, *Lecture Notes in Computer Science* 736, Springer Berlin Heidelberg, pp. 77–102, doi:10.1007/3-540-57318-6_25.

[12] K. G. Larsen, P. Pettersson & W. Yi (1997): *UPPAAL in a nutshell*. *Int. J. on Software Tools for Technology Transfer* 1(1–2), pp. 134–152, doi:10.1007/s100090050010.

[13] N. Lynch, R. Segala & F. Vaandrager (2003): *Hybrid I/O automata*. *Information and Computation* 185(1), pp. 105 – 157, doi:10.1016/S0890-5401(03)00067-1.

[14] O. Maler, Z. Manna & A. Pnueli (1991): *From Timed to Hybrid Systems*. In: *Real-Time: Theory in Practice*, *LNCS* 600, Springer-Verlag, pp. 447–484, doi:10.1007/BFb0032003.

[15] O. Maler & D. Nickovic (2004): *Monitoring Temporal Properties of Continuous Signals*. In Yassine Lakhnech & Sergio Yovine, editors: *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, *Lecture Notes in Computer Science* 3253, Springer Berlin Heidelberg, pp. 152–166, doi:10.1007/978-3-540-30206-3_12.

[16] A. Platzer & J.-D. Quesel (2008): *KeYmaera: A Hybrid Theorem Prover for Hybrid Systems*. In: *Proc. of the Third International Joint Conference on Automated Reasoning (IJCAR 2008)*, *LNCS* 5195, Springer Berlin / Heidelberg, pp. 171–178, doi:10.1007/978-3-540-71070-7_15.

[17] S. Ratschan & Z. She (2007): *Safety Verification of Hybrid Systems by Constraint Propagation Based Abstraction Refinement*. *ACM Transactions in Embedded Computing Systems* 6(1), doi:10.1145/1210268.1210276.

[18] S. Yovine (1997): *Kronos: a verification tool for real-time systems*. *Int. J. on Software Tools for Technology Transfer* 1(1–2), pp. 123–133, doi:10.1007/s100090050009.