Improving HyLTL model checking of hybrid systems

Davide Bresolin
University of Verona (Italy)
davide.bresolin@univr.it

The problem of model-checking hybrid systems is a long-time challenge in the scientific community. Most of the existing approaches and tools are either limited on the properties that they can verify, or restricted to simplified classes of systems. To overcome those limitations, a temporal logic called HyLTL has been recently proposed. The model checking problem for this logic has been solved by translating the formula into an equivalent hybrid automaton, that can be analized using existing tools. The original construction employs a declarative procedure that generates exponentially many states upfront, and can be very inefficient when complex formulas are involved. In this paper we solve a technical issue in the construction that was not considered in previous works, and propose a new algorithm to translate HyLTL into hybrid automata, that exploits optimized techniques coming from the discrete LTL community to build smaller automata.

1 Introduction

Hybrid systems are heterogeneous systems characterized by a tight interaction between discrete and continuous components. Typical examples include discrete controllers that operate in a continuous environment, as in the case of manufacturing plants, robotic systems, and cyberphysical embedded systems. Because of their heterogeneous nature, hybrid systems cannot be faithfully modeled by discrete only nor by continuous only formalisms. In order to model and specify them in a formal way, the notion of *hybrid automata* has been introduced [1, 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. Unfortunately, the model checking problem is computationally very difficult. Indeed, even for simple properties and systems, this problem is not decidable [11].

For very simple classes of hybrid systems, like timed automata, the model checking problem can be solved exactly [2]. Tools like Kronos [20] and UPPAAL [13] can be used to verify properties of timed automata. For more complex classes of systems, the problem became undecidable, and many different approximation techniques may be used to obtain an answer, at least in some cases. Tools like PhaVer [8] and SpaceEx [9] 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 [4], can manage systems with nonlinear dynamics, but are still limited to safety properties.

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 [12], 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 symbolically, and the logic HRELTL [6], that is supported by an extension of the discrete model checker NuSMV, but it is limited to systems with linear dynamics.

Gabriele Puppis, Tiziano Villa (Eds.): Fourth International
Symposium on Games, Automata, Logics and Formal Verification
EPTCS 119, 2013, pp. 79–92, doi:10.4204/EPTCS.119.9

To overcome the limitations of the current technologies, an automata-theoretic approach for model checking hybrid systems has been recently proposed [5]. The work is based on an extension of the well-known temporal logic LTL to hybrid traces called HyLTL. The model checking problem for this logic has been solved by translating the formula into an equivalent hybrid automaton, reducing the model checking problem to a reachability problem that can be solved by existing tools. The original construction employs a declarative procedure that generates exponentially many states upfront, and can be very inefficient when complex formulas are involved.

In this paper we solve a technical issue in the construction that was not considered in previous works by identifying the precise fragment of HyLTL that can be translated into hybrid automata, and we propose a new algorithm to translate formulas into hybrid automata, that exploits optimized techniques coming from the discrete LTL community to be more efficient than the original declarative approach.

2 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 finite set of variables $X \subseteq \mathcal{V}$, a *valuation* over X is a function $\mathbf{x} : X \mapsto \mathbb{R}^n$ that associates a value to every variable in X. The set $\mathrm{Val}(X)$ is the set of all valuations over X.

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 function $\tau : I \mapsto \mathrm{Val}(X)$, where I is a left-closed interval with left endpoint equal to 0. We assume trajectories to be differentiable almost everywhere on the domain, and we denote with $\dot{\tau}$ the corresponding (partial) function giving the value of the derivative of τ for every point in the interior of I where τ is differentiable (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. If $[t,t']$ is a subinterval of $\mathrm{dom}(\tau)$, we denote whith $\tau{\downarrow_{[t,t']}}$ the trajectory τ' such that $\mathrm{dom}(\tau') = [0,t'-t]$ and $\tau'(t'') = \tau(t''+t)$ for every $t'' \in \mathrm{dom}(\tau')$. Given two trajectories τ_1 and τ_2 such that $\tau_1.ltime < +\infty$, their concatenation $\tau_1 \cdot \tau_2$ is the trajectory with domain $[0, \tau_1.ltime + \tau_2.ltime]$ such that $\tau_1 \cdot \tau_2(t) = \tau_1(t)$ if $t \in \mathrm{dom}(\tau_1)$, $\tau_1 \cdot \tau_2(t) = \tau_2(t - \tau_1.ltime)$ otherwise.

Variables will be used in the paper 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 *tilde variables* $\tilde{X}$ as $\{\tilde{x} | x \in X\}$. We use OP, X, $\dot{X}$ and $\tilde{X}$ to define the following two classes of constraints.

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

We use jump constraints to give conditions on pairs of valuations $(\tilde{\mathbf{x}}, \mathbf{x})$. Given a jump constraint c, we say that $(\tilde{\mathbf{x}}, \mathbf{x})$ respects c, and we denote it with $(\tilde{\mathbf{x}}, \mathbf{x}) \vdash c$, when, by replacing every variable x with its value in $\mathbf{x}$ and every tilde variable $\tilde{x}$ with the value of the corresponding normal variable in $\tilde{\mathbf{x}}$ 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 $\dot{\tau}(t)$ respects c when $\dot{\tau}$ is not defined on t).

3 HyLTL: syntax and semantics

The logic HyLTL is an extension of the well-known temporal logic LTL to hybrid systems. 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* FC over X by the following grammar:

$$\varphi ::= f \in FC \mid a \in A \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \varphi\,\mathbf{U}\,\varphi \mid \varphi\,\mathbf{R}\,\varphi \tag{1}$$

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

The semantics of HyLTL is given in terms of *hybrid traces* mixing continuous trajectories with discrete events. Formally, given a set of actions A and a 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 $i \geq 1$. 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 $f \in FC$, $\alpha, i \Vdash f$ if and only if $\tau_i \vdash f$;
- for every $a \in A$, $\alpha, i \Vdash a$ iff $i > 1$ and $a_{i-1} = a$;
- $\alpha, i \Vdash \neg\varphi$ if and only if $\alpha, i \nVdash \varphi$;
- $\alpha, i \Vdash \varphi \wedge \psi$ if and only if $\alpha, i \Vdash \varphi$ and $\alpha, i \Vdash \psi$;
- $\alpha, i \Vdash \varphi \vee \psi$ if and only if $\alpha, i \Vdash \varphi$ or $\alpha, i \Vdash \psi$;
- $\alpha, i \Vdash \mathbf{X}\varphi$ if and only if $\alpha, i+1 \Vdash \varphi$;
- $\alpha, i \Vdash \varphi\,\mathbf{U}\,\psi$ if and only if there exists $j \geq i$ such that $\alpha, j \Vdash \psi$, and for every $i \leq k < j$, $\alpha, k \Vdash \varphi$;
- $\alpha, i \Vdash \varphi\,\mathbf{R}\,\psi$ if and only if for all $j \geq i$, if for every $i \leq k < j$, $\alpha, k \nVdash \varphi$ then $\alpha, j \Vdash \psi$.

Other temporal operators, such as the "always" operator **G** and the "eventually" operator **F** can be defined as usual:

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

3.1 HyLTL with positive constraints

In this paper we will pay a special attention on formulas of HyLTL where flow constraints from FC appears only in positive form, because it will turn out that they constitue the class of formulas that can be

when $a \in A$:	$\pi(a) = a$	$\pi(\neg a) = \neg a$
when $f \in FC$:	$\pi(f) = f \wedge \mathbf{X}((T \wedge f)\,\mathbf{U}\,\neg T)$	$\pi(\neg f) = \bar{f} \vee \mathbf{X}(T\,\mathbf{U}\,(T \wedge \bar{f}))$
	$\pi(\varphi \wedge \psi) = \pi(\varphi) \wedge \pi(\psi)$	$\pi(\varphi \vee \psi) = \pi(\varphi) \vee \pi(\psi)$
	$\pi(\varphi\,\mathbf{U}\,\psi) = (T \vee \pi(\varphi))\,\mathbf{U}\,(\neg T \wedge \pi(\psi))$	$\pi(\varphi\,\mathbf{R}\,\psi) = (\neg T \wedge \pi(\varphi))\,\mathbf{R}\,(T \vee \pi(\psi))$
	$\pi(\mathbf{X}\varphi) = \mathbf{X}(T\,\mathbf{U}\,(\neg T \wedge \pi(\varphi)))$	

Table 1: The translation function π from HyLTL to HyLTL$^+$

translated into hybrid automata. This particular fragment is called *HyLTL with positive flow constraints*, denoted by HyLTL$^+$, and formally defined by the following grammar:

$$\psi ::= f \in FC \mid a \in A \mid \neg a \in A \mid \psi \wedge \psi \mid \psi \vee \psi \mid \mathbf{X}\psi \mid \psi\,\mathbf{U}\,\psi \mid \psi\,\mathbf{R}\,\psi \tag{2}$$

Despite being a syntactical fragment, HyLTL$^+$ turns out to be equally expressive as the full language, at the price of adding an auxiliary action symbol. In the following, given a constraint c we denote with $\bar{c}$ the corresponding "dual" constraint obtained by replacing $<$ with $\geq$, $>$ with $\leq$, $=$ with $\neq$, and so on. Notice that a trajectory τ that satisfies the negation of a flow constraint $\neg c$ does not necessarily satisfy $\bar{c}$. Indeed, by the semantics of HyLTL we have that $\tau \vdash \neg c$ if *there exists* a time instant t such that $\tau(t) \nvdash c$, while $\tau \vdash \bar{c}$ if *for all* time instants t we have that $\tau(t) \nvdash c$.

Hence, given a trajectory τ with domain $\mathrm{dom}(\tau) = [0, t_{max}]$ such that $\tau \vdash \neg c$, it is possible to find a point $t \in [0, t_{max}]$ such that $\tau(t) \nvdash c$ and we can split τ into three sub-trajectories τ_b, $\tau_{\bar{c}}$, τ_e such that $\tau_b = \tau{\downarrow_{[0,t]}}$, $\tau_{\bar{c}} = \tau{\downarrow_{[t,t]}}$ and $\tau_e = \tau{\downarrow_{[t,t_{max}]}}$: it is easy to see that $\tau_{\bar{c}} \vdash \bar{c}$. In the following, the auxiliary action symbol T will be used to represent the splitting points of trajectories when translating formulas with negated flow constraints to formulas with positive flow constraints only.

Given a formula of HyLTL in *in negated normal form* φ, consider the translation function π defined in Table 1. To compare hybrid traces satisfying the original formula φ with the ones satisfying $\pi(\varphi)$ we have to remove the occurrences of T from the latter. To this end, we define a suitable restriction operator over hybrid traces.

Definition 1. *Let A a set of action, and $B \subset A$. Given a hybrid trace $\alpha = \tau_1 a_2 \tau_2 a_2 \ldots$ over A we define its* restriction *to B as the hybrid trace $\alpha{\downarrow_B}$ obtained from α by first removing the actions not in B and then concatenating adjacent trajectories.*

The following lemma states that $\pi(\varphi)$ is a formula of HyLTL$^+$ equivalent to φ.

Lemma 1. *For every hybrid trace α over A and X and every HyLTL-formula φ we have that $\alpha, 1 \Vdash \varphi$ if and only if there exists a hybrid trace β over $A \cup \{T\}$ and X such that $\beta{\downarrow_A} = \alpha$ and $\beta, 1 \Vdash \pi(\varphi)$.*

Proof. Let $\alpha = \tau_1 a_1 \tau_2 a_2 \ldots$ be an hybrid trace over A such that $\alpha, 1 \Vdash \varphi$, and let FC be the set of flow constraints that appears in φ. We will build a sequence of hybrid traces $\beta_0, \beta_1, \beta_2, \ldots$ over $A \cup \{T\}$ as follows.

1. β_0 is the empty sequence.
2. For every $i \geq 1$, consider the i-th trajectory τ_i in α, and let $C_i = \{f \in FC \mid \tau_i \nvdash f\}$. Given an enumeration $f_1, \ldots, f_n$ of C_i, we have that it is possible to find a set of time instants $t_1, \ldots, t_n$ such that $\tau_i(t_j) \vdash \bar{f}$ for every $1 \leq j \leq n$. W.l.o.g., we can assume that $\tau_i.ftime = t_0 \leq t_1 \leq t_2 \leq \ldots \leq t_n \leq t_{n+1} = \tau_i.ltime$ and we can define the sequence of trajectories $\mu_1, \mu_2, \ldots, \mu_{2n+1}$ such that

$$\mu_1 = \tau_i{\downarrow_{[t_0,t_1]}}, \qquad \mu_{2j} = \tau_i{\downarrow_{[t_j,t_j]}}, \qquad \mu_{2j+1} = \tau_i{\downarrow_{[t_j,t_{j+1}]}} \qquad \text{for every } 1 \leq j \leq n \tag{3}$$

   We define $\beta_i = \beta_{i-1}\mu_1 T \mu_2 T \ldots T \mu_{2n+1} a_i$.

The hybrid trajectory we are looking for is the limit trajectory $\beta = \lim_{i\to\infty} \beta_i$.

Given an index i, we will denote by α^i and β^i the suffix of α and of β starting at position i. We show that β respects the following property: *"for every subformula ψ of φ and $i \geq 1$, $\alpha, i \Vdash \psi$ iff $\beta, j \Vdash \pi(\psi)$, where j is the unique index such that $\beta^j\downarrow_A = \alpha^i$"*. The proof is by induction on ψ.

- If $\psi = a$ or $\psi = \neg a$ for some $a \in A$ the property holds trivially.
- Suppose $\psi = f$ for some $f \in FC$. By the semantics, we have that $\tau_i \vdash f$. Consider now the sequence $\mu_1 T \mu_2 T \ldots T \mu_{2n+1} a_i$ built in the construction of β_i, and let j be the index of μ_1 in β. By (3) we have that $\mu_h \vdash f$ for every $1 \leq h \leq 2n+1$. This implies that $\beta, j \Vdash f \wedge \mathbf{X}((T \wedge f)\,\mathbf{U}\,\neg T)$.
- If $\psi = \neg f$ for some $f \in FC$ then we have that $\tau_i \nvdash f$. Let $\mu_1 T \mu_2 T \ldots T \mu_{2n+1} a_i$ be the sequence built in the construction of β_i. Since $f \in C_i$, we have that there exists $t_0 \leq t_k \leq t_{n+1}$ such that $\tau_i(t_k) \vdash \bar{f}$. By (3), this implies that $\mu_k \vdash \bar{f}$. Let j be the index of μ_1 in β. Two case may arise: either $\mu_k = \mu_1$ and thus $\beta, j \Vdash \bar{f}$, or $\mu_k \neq \mu_1$ and then $\beta, j \Vdash \mathbf{X}(T\,\mathbf{U}\,(T \wedge \bar{f}))$. In both cases the property is satisfied.
- The cases of the boolean operators $\vee$ and $\wedge$ are trivial and can be skipped.
- Suppose $\psi = \psi_1\,\mathbf{U}\,\psi_2$, and let i be such that $\alpha, i \Vdash \psi_1\,\mathbf{U}\,\psi_2$. By the semantics, we have that there exists $k \geq i$ such that $\alpha, k \Vdash \psi_2$ and, for every $i \leq h < k$, $\alpha, h \Vdash \psi_1$. Now, let j and l be the two indexes such that $\beta^j\downarrow_A = \alpha^i$ and $\beta^l\downarrow_A = \alpha^k$. By inductive hypothesis we can assume that $\beta, l \Vdash \pi(\psi_2)$, while by the definition of the $\downarrow_A$ operator we have that $\beta, l \Vdash a_i \neq T$. Hence, $\beta, l \Vdash \neg T \wedge \pi(\psi_2)$. Consider now any index m such that $j \leq m < l$. Two cases may arise: either $\beta, m \Vdash T$, or not. In the latter case, we have that it is possible to find an index $i \leq h < k$ such that $\beta^m\downarrow_A = \alpha^h$. Since $\alpha, h \Vdash \psi_1$, by inductive hypothesis we have that $\beta, m \Vdash \pi(\psi_1)$. Hence, in both cases $\beta, m \Vdash T \vee \pi(\psi_1)$. This proves that $\beta, j \Vdash (T \vee \pi(\psi_1))\,\mathbf{U}\,(\neg T \wedge \pi(\psi_2)) = \pi(\psi)$.
  To prove the converse implication, suppose that $\beta, j \Vdash (T \vee \pi(\psi_1))\,\mathbf{U}\,(\neg T \wedge \pi(\psi_2))$. By the semantics, we have that there exists $l \geq j$ such that $\beta, l \Vdash \neg T \wedge \pi(\psi_2)$ and, for every $j \leq m < l$, $\beta, m \Vdash T \vee \pi(\psi_1)$. Since $\beta, l \Vdash \neg T$ it is possible to find an index k such that $\beta^l\downarrow_A = \alpha^k$. Hence, by inductive hypothesis we have that $\alpha, k \Vdash \psi_2$. Now, let h be such that $i \leq h < k$, and consider the index m such that $\beta^m\downarrow_A = \alpha^h$. By the semantics we have that $\beta, m \Vdash T \vee \pi(\psi_1)$. Since, by definition of the restriction operator, $\beta, m \nVdash T$, we have that $\beta, m \Vdash \pi(\psi_1)$ and thus, by inductive hypothesis, that $\alpha, h \Vdash \psi_1$. This proves that $\alpha, i \Vdash \psi_1\,\mathbf{U}\,\psi_2$.
- The cases of the temporal operators $\mathbf{X}$ and $\mathbf{R}$ can be proved by a similar argument.

By the property it is immediate to conclude that, since $\alpha, 1 \Vdash \varphi$ then $\beta, 1 \Vdash \pi(\varphi)$.

To conclude the proof, suppose that there exists a hybrid trace β such that $\beta, 1 \Vdash \pi(\varphi)$, and let $\alpha = \beta\downarrow_A$. By an induction on the structure of φ similar to the one above, we can prove that $\alpha, 1 \Vdash \varphi$. □

4 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.

In this section we recap the definition of Hybrid Automata introduced in [5] to solve the model checking problem for HyLTL.

Definition 2. *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. $\text{Edg} \subseteq \text{Loc} \times A \times \text{Loc}$ *is a set of* discrete transitions*;*
5. Dyn *is a mapping that associates to every location* $\ell \in \text{Loc}$ *a set of flow constraints* $\text{Dyn}(\ell)$ *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* $\tilde{X} \cup X$ *describing the guard and reset function of the transition;*
7. $\text{Init} \subseteq \text{Loc}$ *is a set of* initial locations.

The *state* of a 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 changes of the state.

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:*

$$(\ell, \mathbf{x}) \xrightarrow{\tau} (\ell, \mathbf{x}') \iff \tau.fstate = \mathbf{x} \wedge \tau.lstate = \mathbf{x}' \wedge \tau \vdash \text{Dyn}(\ell). \tag{4}$$

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

$$(\ell, \mathbf{x}) \xrightarrow{a} (\ell', \mathbf{x}') \iff \mathbf{x} \vdash \text{Dyn}(\ell) \wedge \mathbf{x}' \vdash \text{Dyn}(\ell') \wedge (\mathbf{x}, \mathbf{x}') \vdash \text{Res}(\ell, a, \ell'). \tag{5}$$

The above definitions allows an infinite sequence of discrete events to occur in a finite amount of time (Zeno behaviors). Such behaviors are physically meaningless, but very difficult to exclude completely from the semantics. In this paper we assume that all hybrid automata under consideration do not generate Zeno runs. This can be achieved, for instance, by adding an extra clock variable that guarantees that the delay between any two discrete actions is bounded from below by some constant. Moreover, we assume that all hybrid automata are *progressive*, that is, that all runs can be extended to an infinite one: it is not possible to stay forever in a location and never activate a new discrete action.

We can view progressive, non-Zeno hybrid automata as *generators* of hybrid traces, as formally expressed by the following definition.

Definition 4. *Let* $\mathcal{H}$ *be a progressive, non-Zeno hybrid automaton, and let* $\alpha = \tau_1 a_1 \tau_2 a_2 \ldots$ *be a 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)* $(\ell_i, \tau_i.lstate) \xrightarrow{a_i} (\ell_{i+1}, \tau_{i+1}.fstate)$.

Our definition of hybrid automata admits composition, under the assumpion that all variables and actions are shared between the different automata. The formal definition of the parallel composition operator $\|$ can be found in [5]. In this paper it is sufficient to recall that it respects the usual "compositionality property", that is, that the set of hybrid traces generated by a composition of hybrid automata corresponds to the intersection of the hybrid traces generated by the components (up to projection to the correct set of actions and variables).

5 Model checking HyLTL

In analogy with the classical automata-theoretic approach, in [5] the model checking problem for HyLTL has been solved by translating the HyLTL formula into an equivalent hybrid automaton, enriched with

a suitable *Büchi acceptance condition* to identify the traces generated by the automaton that fulfills the semantics of HyLTL.

Definition 5. *A* Hybrid Automaton with Büchi condition (BHA) *is a tuple* $\mathcal{H} = \langle \text{Loc}, X, A, \text{Edg}, \text{Dyn}, \text{Res}, \text{Init}, \mathcal{F} \rangle$ *such that* $\langle \text{Loc}, X, A, \text{Edg}, \text{Dyn}, \text{Res}, \text{Init} \rangle$ *is a Hybrid Automaton, and* $\mathcal{F} \subseteq \text{Loc}$ *is a finite set final locations.*

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

(i) $\ell_1 \in \text{Init}$;
(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) there exists $\ell_f \in \mathcal{F}$ that occurs infinitely often in the sequence.

By the above definition, not all sequences generated by the automaton are accepting: only those that respect the additional accepting condition are considered.

By the definition of the dynamics, hybrid automata can enforce only *positive constraints* on the continuous flow of the system. Hence, they can only recognize formulas of the positive flow fragment of HyLTL, as summarized by the following theorem.

Theorem 1 ([5])**.** *Given a HyLTL*$^+$ *formula* φ*, it is possible to build a BHA* $\mathcal{H}_\varphi$ *that accepts all and only those hybrid traces that satisfies* φ*.*

Theorem 1 and Lemma 1 can be exploited to solve the model checking problem for full HyLTL as follows. 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 formula $\neg\varphi$ and its translation $\overline{\varphi} = \pi(\neg\varphi)$. By Lemma 1 we have that $\overline{\varphi}$ is a formula of HyLTL$^+$ that is equivalent to $\neg\varphi$, and thus we can build a BHA $\mathcal{H}_{\overline{\varphi}}$ that is equivalent to *the negation of the property*: 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 $\overline{\varphi}$ we obtain a BHA $\mathcal{H}_S \| \mathcal{H}_{\overline{\varphi}}$ that 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 $\mathcal{H}_S \| \mathcal{H}_{\overline{\varphi}}$ *does not accept any hybrid trace*.

It is worth pointing out that the reachability problem of hybrid automata is undecidable. This means that the model checking of HyLTL is an undecidable problem as well (reachability can be expressed by an eventuality formula). However, this does not mean that out logic is completely intractable. A number of different approximation techniques have been developed in the past years to obtain an answer to the reachability problem (at least in some cases), and they can be exploited to solve the model checking problem of HyLTL as well. Indeed, $\mathcal{H}_S \| \mathcal{H}_{\overline{\varphi}}$ accepts an hybrid trace if and only if there exists a loop that includes a final location and that is reachable from the initial states. As shown in [5], this property can be reduced to a reachability property that can be tested by existing tools for the analysis of hybrid automata. The only thing that one needs to do is to write a procedure implementing the construction of $\mathcal{H}_{\overline{\varphi}}$, and then send the results to the reachability analysis tool.

6 An improved construction algorithm

The algorithm presented in [5] to build a BHA equivalent to a HyLTL$^+$-formula φ is based on a declarative construction. While being simple to understand, it suffers of a major drawback from the efficiency point of view: it generates exponentially many locations upfront, even though many of them may be inconsistent, redundant or unreachable. This implies that the resulting BHA can be very big, even for very

$\gamma(\varphi) = \bigwedge_{i=0}^{n-1} \neg b_i \wedge \gamma_0(\varphi)$		
$\gamma_0(a) = \mathbf{b}(a)$	$\gamma_0(f) = f$	when $a \in A$ or $f \in FC$
$\gamma_0(\neg\varphi) = \neg\gamma_0(\varphi)$	$\gamma_0(\varphi \wedge \psi) = \gamma_0(\varphi) \wedge \gamma_0(\psi)$	$\gamma_0(\varphi \vee \psi) = \gamma_0(\varphi) \vee \gamma_0(\psi)$
$\gamma_0(\mathbf{X}\varphi) = \mathfrak{X}(\gamma_0(\varphi))$	$\gamma_0(\varphi \mathbf{U} \psi) = \gamma_0(\varphi) \mathfrak{U} \gamma_0(\psi)$	$\gamma_0(\varphi \mathbf{R} \psi) = \gamma_0(\varphi) \mathfrak{R} \gamma_0(\psi)$

Table 2: The translation function γ from HyLTL to LTL

simple formulas. In this section we describe an improved construction algorithm, based on the following steps:

A. the HyLTL$^+$-formula φ is first translated into a suitable formula of discrete LTL $\gamma(\varphi)$;

B. a discrete Büchi automaton $\mathcal{A}_{\gamma(\varphi)}$, equivalent to $\gamma(\varphi)$, is built using one of the many optimized tools available in the literature;

C. a BHA $\mathcal{H}_\varphi$, equivalent to φ, is built from $\mathcal{A}_{\gamma(\varphi)}$.

The new algorithm improves the original one by building a smaller BHA, thanks to the use of optimized tools for LTL in step B.

6.1 From HyLTL to discrete LTL

Let FC and A be respectively the set of all flow constraints and discrete actions appearing in φ. For the sake of simplicity, we will assume that $\|A \cup \{T\}\| = 2^n - 1$ for some $n \in \mathbb{N}$ (if this is not the case, we can always add some fresh action symbols to A that will not appear in the formula). Under this assumption we can represent action symbols from $A \cup \{T\}$ by means of a set of n propositional letters $B = \{b_0, \ldots, b_{n-1}\}$, where every possible combination of the truth values, but the one where all letters are false, uniquely identify one action symbol. For every $a \in A \cup \{T\}$ let $\mathbf{b}(a)$ be the corresponding encoding. By definition, we put $\mathbf{b}(T) = \bigwedge_{i=0}^{n-1} b_i$.

If we consider $AP = FC \cup \{b_0, \ldots, b_{n-1}\}$ as a set of propositional letters for discrete LTL, we have that we can transform any hybrid trace $\alpha = \tau_1 a_1 \tau_2 a_2 \ldots$ into a discrete sequence $\Sigma(\alpha) = \sigma_1 \sigma_2 \sigma_3 \ldots$ where every element is a subset of AP defined as follows: $\sigma_1 = \{f \in FC \mid \tau_1 \vdash f\}$; for every $i > 1$, $\sigma_i = \{f \in FC \mid \tau_i \vdash f\} \cup \{b_j \in B \mid b_j \text{ holds true in } \mathbf{b}(a_{i-1})\}$.

Now, let $\gamma(\varphi)$ be the discrete LTL formula obtained from φ by means of the translation function γ defined in Table 2. It is easy to see that $\Sigma(\alpha)$ is a model for $\gamma(\varphi)$, as proved by the following lemma.

Lemma 2. *For every hybrid trace α, $\alpha, 1 \Vdash \varphi$ if and only if $\Sigma(\alpha), 1 \Vdash \gamma(\varphi)$.*

Proof. Let φ be a HyLTL formula, and α a hybrid trace. We prove the lemma by showing that the following stronger claim holds:

$$\text{for every } i \geq 1,\ \alpha, i \Vdash \varphi \text{ if and only if } \Sigma(\alpha), i \Vdash \gamma_0(\varphi).$$

We reason by induction on the structure of φ:

- if $\varphi = a$, with $a \in A$, then $\gamma_0(a) = \mathbf{b}(a)$ and the claim follows easily by the definition of $\Sigma(\alpha)$;
- if $\varphi = f$, with $f \in FC$, then $\gamma_0(f) = f$ and the claim follows easily by the definition of $\Sigma(\alpha)$;
- the boolean cases are trivial and thus skipped;
- when $\varphi = \mathbf{X}\psi$, we have that $\alpha, i \Vdash \mathbf{X}\psi$ iff $\alpha, i+1 \Vdash \psi$. By inductive hypothesis we have that $\Sigma(\alpha), i+1 \Vdash \gamma_0(\psi)$, from which we can conclude that $\Sigma(\alpha), i \Vdash \mathfrak{X}\gamma_0(\psi)$;

- suppose $\varphi = \psi_1 \mathbf{U} \psi_2$. By the semantic of HyLTL, we have that $\alpha, i \Vdash \psi_1 \mathbf{U} \psi_2$ iff there exists $j \geq i$ such that $\alpha, j \Vdash \psi_2$, and for every $i \leq k < j$, $\alpha, k \Vdash \psi_1$. By inductive hypothesis we have that $\Sigma(\alpha), j \Vdash \gamma_0(\psi_2)$ and that $\Sigma(\alpha), k \Vdash \gamma_0(\psi_1)$ for every $i \leq k < j$. Hence, $\Sigma(\alpha), i \Vdash \gamma_0(\psi_1) \mathfrak{U} \gamma_0(\psi_2)$ and the claim is proved.
- the case when $\varphi = \psi_1 \mathbf{R} \psi_2$ is analogous.

To conclude the proof it is sufficient to consider that, by definition, $\Sigma(\alpha), 1 \Vdash \bigwedge_{i=0}^{n-1} \neg b_i$. Hence, from the claim it is immediate to conclude that $\Sigma(\alpha), 1 \Vdash \bigwedge_{i=0}^{n-1} \neg b_i \wedge \gamma_0(\varphi)$ if and only if $\alpha, 1 \Vdash \varphi$. □

When φ is a formula of HyLTL$^+$ we have that also $\gamma(\varphi)$ is a formula where flow constraints appear only in positive form. Hence, $\gamma(\varphi)$ cannot force the negation of a flow constraint to hold in any of the elements σ_i of a discrete sequence, as formally stated by the following lemma.

Lemma 3. *Let $\Sigma = \sigma_1 \sigma_2 \ldots$ and $\mathrm{P} = \rho_1 \rho_2 \ldots$ be two discrete sequences such that for every $i \geq 1$, $\sigma_i \cap B = \rho_i \cap B$ (the sequences agrees on the propositional letters in B) and $\sigma_i \subseteq \rho_i$ (every flow constraint that is true in Σ is true also in P). Then, for every LTL formula γ where flow constraints appear only in positive form and index $j \geq 1$, if $\Sigma, j \Vdash \gamma$ then $\mathrm{P}, j \Vdash \gamma$.*

Proof. Suppose $\Sigma, j \Vdash \gamma$. We prove the claim by induction on the structure of γ.

- If $\gamma = b_k$ or $\gamma = \neg b_k$, for some $b_k \in B$, we have that the claim follows immediately by the fact that $\sigma_j \cap B = \rho_j \cap B$;
- If $\gamma = f$ for some $f \in FC$, by the semantics of LTL we have that $f \in \sigma_j$. By hypothesis $\sigma_j \subseteq \rho_j$ and this implies that $\mathrm{P}, j \Vdash f$;
- The remaining cases can be easily proved from the inductive hypothesis and the semantics of LTL. □

6.2 Building the Büchi automaton $\mathcal{A}_{\gamma(\varphi)}$

Since the seminal work of Vardi and Wolper [19], translation of LTL formulas into equivalent Büchi automata plays an important role in many model checking and satisfiability checking algorithms. This led to the development of many translation algorithms exploiting several heuristics and optimization techniques. According to the experiments in [18], two leading tools are LTL2BA [10] and SPOT [7]. A new version of the former, called LTL3BA, has been recently introduced [3]. According to the authors, it is faster and it produces smaller automata than LTL2BA, while it produces automata of similar quality with respect to SPOT, being usually faster.

We choose to use LTL3BA as the tool for translating the formula $\gamma(\varphi)$ into the Büchi automaton $\mathcal{A}_{\gamma(\varphi)}$, since it is a state-of-the-art tool that is freely available under an open source license. Nevertheless, the high level HyLTL$^+$ translation algorithm is independent from the specific tool used to build $\mathcal{A}_{\gamma(\varphi)}$, and can be easily adapted to use other tools.

The output of LTL3BA is a Büchi automaton $\mathcal{A}_{\gamma(\varphi)}$ of the form $\langle Q, q_0, \delta, F \rangle$, where Q is the set of states, q_0 is the unique initial state, δ is the transition relation and F is the set of final states. To merge many transitions into a single one, the transitions are labelled with conjunctions of atomic propositions from AP: the automaton can fire a transition (q, β, q') whenever it reads a symbol σ_j of the discrete sequence that satisfies the boolean formula β. Since $\gamma(\varphi)$ is a formula where flow constraints appear only positively, Lemma 3 guarantees that we can assume, without loss of generality, that in the boolean formulas labeling the transitions of $\mathcal{A}_{\gamma(\varphi)}$ flow constraints appear only positively. The following lemma connects the language of $\mathcal{A}_{\gamma(\varphi)}$ with the set of hybrid traces satisfying φ.

Algorithm 1: how to build the BHA equivalent to φ

Input: $\mathcal{A}_{\gamma(\varphi)} = \langle Q, q_0, \delta, F \rangle$
Output: $\mathcal{H}_\varphi = \langle \text{Loc}, X, A \cup \{T\}, \text{Edg}, \text{Dyn}, \text{Res}, \text{Init}, \mathcal{F} \rangle$

```
Loc = ∅, Edg = ∅;
𝓛 = ∅;
foreach transition (q0, β, q) ∈ δ do
    if β → ⋀_{i=0}^{n-1} ¬b_i then
        C = {f ∈ FC | β → f};
        add (q, C) to Loc;
        add (q, C) to Init;
        set Dyn(q, C) = C;
        add (q, C) to 𝓛;
    end
end
while the queue 𝓛 is not empty do
    extract an element (q, C) from 𝓛;
    foreach transition (q, β, q') ∈ δ do
        C' = {f ∈ FC | β → f};
        if (q', C') ∉ Loc then
            add (q', C') to Loc;
            set Dyn(q', C') = C';
            add (q', C') to 𝓛;
        end
        foreach a ∈ A ∪ {T} do
            if β → b(a) then
                add transition (q, C, a, q', C') to Edg;
                set Res(q, C, a, q', C') = ⊤;
            end
        end
    end
end
ℱ = {(q, C) ∈ Loc | q ∈ F};
```

Lemma 4. *Let φ be a HyLTL$^+$ formula, and α a hybrid trace. Then $\alpha, 1 \Vdash \varphi$ if and only if $\Sigma(\alpha)$ is accepted by $\mathcal{A}_{\gamma(\varphi)}$.*

6.3 From $\mathcal{A}_{\gamma(\varphi)}$ to $\mathcal{H}_\varphi$

By Lemma 4, we have that the language of $\mathcal{A}_{\gamma(\varphi)}$ *contains* all the discrete sequences $\Sigma(\alpha)$ such that α satisfies φ. However, $\mathcal{A}_{\gamma(\varphi)}$ may accepts also "spurious" discrete sequences that do not represent a hybrid trace (for instance, sequences where flow constraints are contradictory). Algorithm 1 accepts as input the discrete automaton $\mathcal{A}_{\gamma(\varphi)}$ and build a BHA $\mathcal{H}_\varphi$ that accepts only the hybrid traces satisfying φ.

The following theorem proves that the algorithm is correct.

Theorem 2. *Let φ be a formula of HyLTL$^+$, and let $\mathcal{H}_\varphi$ be the BHA built by Algorithm 1. For every hybrid trace α, we have that $\mathcal{H}_\varphi$ accepts α if and only if $\alpha, 1 \Vdash \varphi$.*

Proof. Let $\alpha = \tau_1 a_1 \tau_2 a_2 \ldots$ be a hybrid trace such that $\alpha, 1 \Vdash \varphi$. By Lemma 4, we have that $\mathcal{A}_{\gamma(\varphi)}$ accepts the discrete sequence $\Sigma(\alpha)$. Let $q_0 \xrightarrow{\beta_1} q_1 \xrightarrow{\beta_2} q_2 \xrightarrow{\beta_2} \ldots$ be an accepting run of $\mathcal{A}_{\gamma(\varphi)}$ over $\Sigma(\alpha)$. For every $i \geq 1$, let $C_i = \{f \in FC \mid \beta_i \rightarrow f\}$, and consider the sequence $(q_1, C_1), (q_2, C_2), (q_3, C_3) \ldots$. By Algorithm 1 we have that:

1. every pair (q_i, C_i) of the sequence is a location of $\mathcal{H}_\varphi$;
2. $(q_1, C_1) \in$ Init;
3. every set of flow constraints C_i is such that $\mathrm{Dyn}(q_i, C_i) = C_i$;
4. the transition $(q_i, C_i, a_i, q_{i+1}, C_{i+1}) \in$ Edg with reset condition $\top$.

By definition of $\Sigma(\alpha)$ we have that $\tau_i \vdash C_i$, and thus we can conclude that for every $i \geq 1$, both $(q_i, C_i, \tau_i.fstate) \xrightarrow{\tau_i} (q_i, C_i, \tau_i.lstate)$ and $(q_i, C_i, \tau_i.lstate) \xrightarrow{a_i} (q_{i+1}, C_{i+1}, \tau_{i+1}.fstate)$ are valid transitions of $\mathcal{H}_\varphi$. This means that α is generated by $\mathcal{H}_\varphi$. Since $\Sigma(\alpha)$ is accepted by the discrete automaton $\mathcal{A}_{\gamma(\varphi)}$ is possible to find a location $(q_f, C_f) \in \mathcal{F}$ that occurs infinitely often in the sequence. This proves that α is accepted by $\mathcal{H}_\varphi$.

To conclude the proof, consider a hybrid trace $\alpha = \tau_1 a_1 \tau_2 a_2 \ldots$ that is accepted by $\mathcal{H}_\varphi$, and let $\Sigma(\alpha) = \sigma_1 \sigma_2 \ldots$ be the corresponding discrete sequence. By the semantics of BHA, it is possible to find an accepting sequence of locations $(q_1, C_1), (q_2, C_2), (q_3, C_3) \ldots$ such that $(q_i, C_i, \tau_i.fstate) \xrightarrow{\tau_i} (q_i, C_i, \tau_i.lstate)$ and $(q_i, C_i, \tau_i.lstate) \xrightarrow{a_i} (q_{i+1}, C_{i+1}, \tau_{i+1}.fstate)$ for every $i \geq 1$. By Algorithm 1 we have that there exists an accepting run $q_0 \xrightarrow{\rho_1} q_1 \xrightarrow{\rho_2} q_2 \xrightarrow{\rho_3} \ldots$ of the discrete automaton $\mathcal{A}_{\gamma(\varphi)}$ over the discrete sequence $\mathrm{P} = \rho_1 \rho_2 \ldots$ where $\rho_i = C_i \cup \{b_j \in B \mid b_j \text{ holds true in } \mathbf{b}(a_{i-1})\}$ for every $i \geq 1$. Since every location (q_i, C_i) is such that $\mathrm{Dyn}(q_i, C_i) = C_i$ we have that for every $f \in C_i$, $\tau_i \vdash f$ and thus that $\rho_i \subseteq \sigma_i$. From Lemma 3 we can conclude that, since $\mathcal{A}_{\gamma(\varphi)}$ accepts P then $\mathcal{A}_{\gamma(\varphi)}$ accepts also $\Sigma(\alpha)$. By Lemma 4 we can conclude that $\alpha, 1 \Vdash \varphi$. □

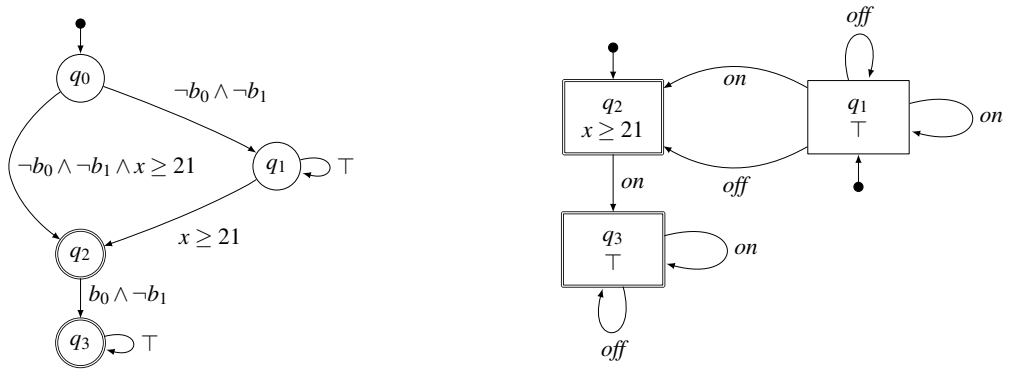
7 The improved algorithm at work

In [5] feasibility of the automaton-based model checking approach has been tested by verifying the well-known Thermostat example against the HyLTL formula $\varphi_{hyb} = \neg \mathbf{F}(x \geq 21 \wedge \mathbf{X} on)$ corresponding to the property that *"it is not possible that the heater turns on when the temperature is above 21 degrees"*.

To verify the example 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 original declarative construction builds a BHA with 18 locations. In this section we will apply the new algorithm to the formula and we will show that the resulting BHA is much smaller that the previous one. Notice that the formula $\neg\varphi_{hyb}$ is a formula where flow constraints appears only in positive form. Hence, it is not necessary to apply the translation π of Table 1 to obtain a formula of HyLTL$^+$. The first step of the translation algorithm is thus the application of function γ (Table 2) to obtain the following formula of discrete LTL:

$$\gamma(\neg\varphi_{hyb}) = \neg b_0 \wedge \neg b_1 \wedge \top\, \mathfrak{U}\, (x \geq 21 \wedge \mathfrak{X}(b_0 \wedge \neg b_1)),$$

where we assume that $\mathbf{b}(on) = b_0 \wedge \neg b_1$. By using the tool LTL3BA we obtain the Büchi automaton $\mathcal{A}_{\gamma(\neg\varphi_{hyb})}$ depicted in Figure 1a. Then, by applying Algorithm 1 we can build the BHA depicted in Figure 1b. In both pictures initial states/locations are identified by a bullet-arrow while the final states/locations have a double border. The final BHA obtained by the new construction algorithm is made of only 3 location, with a great improvement over the original declarative construction.



(a) Büchi automaton $\mathcal{A}_{\gamma(\neg\varphi_{hyb})}$.

(b) Hybrid automaton $\mathcal{H}_{\neg\varphi_{hyb}}$.

Figure 1: The discrete and hybrid automata for $\neg\varphi_{hyb}$.

As a second example, consider the globally-eventually formula $\varphi_{liv} = \mathbf{G}(\neg x \geq 18 \rightarrow \mathbf{X}\mathbf{F}\,on)$ expressing the liveness property to *"eventually switch the heater on if the temperature falls below* 18 *degrees"*. In this case the negation of the property is the formula $\neg\varphi_{liv} = \mathbf{F}(\neg x \geq 18 \wedge \mathbf{X}\mathbf{G}\,\neg on) = \top\,\mathbf{U}\,(\neg x \geq 18 \wedge \mathbf{X}(\bot\,\mathbf{R}\,\neg on))$, that do not belongs to the language of HyLTL$^+$. Hence, it is necessary to apply the translation function π to obtain the following equivalent formula:

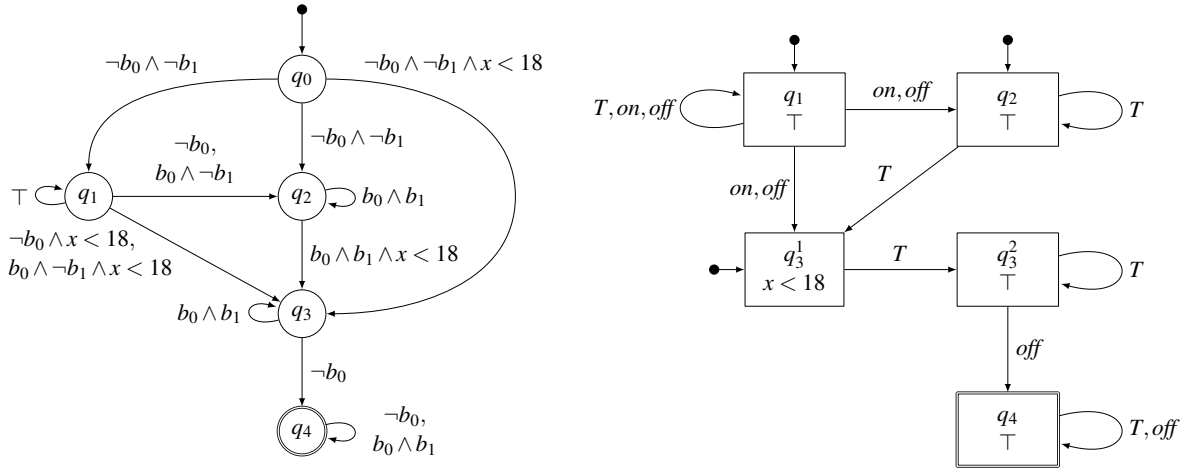
$$\begin{aligned}
\overline{\varphi}_{liv} &= \pi\Big(\top\,\mathbf{U}\,\big(\neg x \geq 18 \wedge \mathbf{X}(\bot\,\mathbf{R}\,\neg on)\big)\Big) = (T \vee \top)\,\mathbf{U}\,\Big(\neg T \wedge \pi\big(\neg x \geq 18 \wedge \mathbf{X}(\bot\,\mathbf{R}\,\neg on)\big)\Big) \\
&= \top\,\mathbf{U}\,\Big(\neg T \wedge \pi(\neg x \geq 18) \wedge \pi\big(\mathbf{X}\,(\bot\,\mathbf{R}\,\neg on)\big)\Big) \\
&= \top\,\mathbf{U}\,\Big(\neg T \wedge \Big(x < 18 \vee \mathbf{X}\big(T\,\mathbf{U}\,(T \wedge x < 18)\big)\Big) \wedge \mathbf{X}\Big(T\,\mathbf{U}\,\big(\neg T \wedge \pi(\bot\,\mathbf{R}\,\neg on)\big)\Big)\Big) \\
&= \top\,\mathbf{U}\,\Big(\neg T \wedge \Big(x < 18 \vee \mathbf{X}\big(T\,\mathbf{U}\,(T \wedge x < 18)\big)\Big) \wedge \mathbf{X}\Big(T\,\mathbf{U}\,\big(\neg T \wedge \bot\,\mathbf{R}\,(T \vee \neg on)\big)\Big)\Big)
\end{aligned}$$

The input formula for LTL3BA is thus

$$\begin{aligned}
\gamma(\overline{\varphi}_{safe}) = \neg b_0 \wedge \neg b_1 \wedge \top\,\mathfrak{U}\Bigg(&\neg(b_0 \wedge b_1) \wedge \Big(x < 18 \vee \mathfrak{X}\big((b_0 \wedge b_1)\,\mathfrak{U}\,(b_0 \wedge b_1 \wedge x < 18)\big)\Big) \\
&\wedge \mathfrak{X}\Big((b_0 \wedge b_1)\,\mathfrak{U}\,\big(\neg(b_0 \wedge b_1) \wedge \bot\,\mathfrak{R}\,\big((b_0 \wedge b_1) \vee \neg(b_0 \wedge \neg b_1)\big)\big)\Big)\Bigg)
\end{aligned}$$

while the resulting discrete Büchi automaton is depicted in Figure 2a. Algorithm 1 transforms it into the BHA with 5 locations shown in Figure 2b. Notice that, despite the increased complexity of the formula due to the translation into HyLTL$^+$ the final result is still of very small size.

We have verified that the thermostat example given in [5] respects the two example properties φ_{hyb} and φ_{liv} using the software package PhaVer [8]. Since the system and the automata for the properties are very simple, the computation time was almost instantaneous: less than $0.1s$ for both formulas on an Intel Core 2 Duo 2.4 GHz iMac with 4 Gb of RAM.



(a) Büchi automaton $\mathcal{A}_{\gamma(\overline{\varphi}_{liv})}$.

(b) Hybrid automaton $\mathcal{H}_{\overline{\varphi}_{liv}}$.

Figure 2: The discrete and hybrid automata for $\neg\varphi_{liv}$.

8 Conclusion

In this paper we extended the current research on HyLTL, a logic that is able to express properties of hybrid traces, and that can be used to verify hybrid systems. We identified the fragment of HyLTL that can be transformed into hybrid automata, that is, the positive flow constraints fragment HyLTL$^+$. Then, we have shown that every property definable in the full language is also definable by HyLTL$^+$. Finally, we developed a new algorithm to translate formulas into hybrid automata, that turned out to be much more efficient than the original declarative algorithm.

This work 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. A comprehensive tool support for the logic is currently missing: an implementation of the complete model checking algorithm into the software package Ariadne [4] is under development.

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*. *J. of Theor. Computer Science* 126(2), pp. 183–235, doi:10.1016/0304-3975(94)90010-8.

[3] T. Babiak, M. Kretínský, V. Rehák & J. Strejcek (2012): *LTL to Büchi Automata Translation: Fast and More Deterministic*. In: *Proc. of the 18th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2012)*, *LNCS* 7214, pp. 95–109, doi:10.1007/978-3-642-28756-5_8.

[4] 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.

[5] D. Bresolin (2013): *HyLTL: a temporal logic for model checking hybrid systems*. In: *Proc. of the 3rd International Workshop on Hybrid Autonomous Systems (HAS 2013)*, *EPTCS* 118, pp. 64–75. To appear.

[6] 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.

[7] A. Duret-Lutz (2011): *LTL translation improvements in SPOT*. In: *Proc. of the 5th Int. Conf. on Verification and Evaluation of Computer and Communication Systems (VECoS'11)*, British Computer Society, pp. 72–83.

[8] 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.

[9] 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.

[10] P. Gastin & D. Oddoux (2001): *Fast LTL to Büchi Automata Translation*. In: *Proc. of the 13th Int. Conf. on Computer Aided Verification (CAV 2001)*, *LNCS* 2102, Springer, pp. 53–65, doi:10.1007/3-540-44585-4_6.

[11] 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.

[12] L. Lamport (1993): *Hybrid systems in TLA+*. In: *Hybrid Systems*, *LNCS* 736, Springer, pp. 77–102, doi:10.1007/3-540-57318-6_25.

[13] 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.

[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: *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, *LNCS* 3253, Springer, 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 3rd International Joint Conference on Automated Reasoning (IJCAR 2008)*, *LNCS* 5195, Springer, 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 Trans. in Embedded Computing Systems* 6(1), doi:10.1145/1210268.1210276.

[18] K. Y. Rozier & M. Y. Vardi (2010): *LTL satisfiability checking*. *Int. J. on Software Tools for Technology Transfer* 12(2), pp. 123–137, doi:10.1007/s10009-010-0140-3.

[19] M. Y. Vardi & P. Wolper (1986): *An Automata-Theoretic Approach to Automatic Program Verification*. In: *Proc. of the 1st Symposium on Logic in Computer Science (LICS'86)*, IEEE Computer Society, pp. 332–344.

[20] 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.