Verification of Hybrid Automata by Synthesis and Refinement

Ruggero Lanotte[1], Andrea Maggiolo-Schettini[2], Simone Tini[1], and Angelo Troina[2]

[1] Dipartimento di Scienze della Cultura, Politiche e dell'Informazione, Università dell'Insubria, via Valleggio 11, 22100, Como, Italy
[2] Dipartimento di Informatica, Università di Pisa, via Buonarroti 2, 56127, Pisa, Italy

Abstract. We present a verification technique to check properties of systems in the model of Linear Hybrid Automata. Given an automaton synthesized from a logic formula, we stepwise specialize transitions and states by operations which preserve the language accepted by the original automaton. We define a proof system based on refinement technique that preserves the properties of the original automaton, and we prove its correctness.

1 Introduction

Hybrid Automata (*HA*) [1, 6] are among the most studied models for *hybrid systems* (see, e.g., [10]), i.e. dynamical systems combining discrete and continuous state changes. Hybrid automata combine finite state machines with continuously evolving *variables*, and exhibit two kinds of state changes: discrete jump transitions, which occur instantaneously, and continuous flow transitions, which occur while time elapses.

Most of hybrid system applications are safety critical and require guarantees of safe operation. For this reason, several formalisms have been introduced to express behavioral properties of hybrid systems (for a survey see [4]). Among them, we consider the formalism PTL [5], which permits us to express both qualitative and quantitative aspects of the temporal behavior of systems.

In this paper we present a strategy to check whether Hybrid Automata satisfy properties expressed by PTL formulae. First of all, we introduce a set of *refinement rules*, i.e. a set of transformations schemata that permit us to map Hybrid Automata to other Hybrid Automata. We prove that, by applying any of our rules to any Hybrid Automaton H, we obtain a Hybrid Automaton H' such that the language recognized by H' is a subset of the language recognized by H. Since PTL is a linear time temporal logic, this implies that all PTL formulae satisfied by H are satisfied also by H'.

Now, to check whether a given Hybrid Automaton H satisfies a given PTL formula ϕ, we firstly synthesize from ϕ an Hybrid Automaton H^ϕ satisfying ϕ. To this purpose, we employ the same synthesis algorithm of [11, 8, 5, 3]. Then, by several applications of our refinement rules, we transform H^ϕ to a Hybrid

Automaton satisfying ϕ. If, by this refinement process, we obtain H, then we infer that H satisfies ϕ. Otherwise, we cannot infer anything.

A well known technique to check whether T satisfies ϕ is to build an automaton $T^{\neg\phi}$ that can perform all the runs that do not satisfy ϕ, and to check whether the cartesian product $T \times T^{\neg\phi}$ has an empty set of runs (see [11, 8, 5, 3]). Now, also this strategy is sound, but emptiness is only semi-decidable. The strategy of [11, 8, 5, 3] is automatic (i.e. model checking), our strategy requires an expert user able to choose the "right" refinement operations, and recalls theorem proving.

We propose our technique as an alternative to the one mentioned based on model checking. It is our purpose to continue our investigation to see in which cases, if any, either of the two techniques may be more convenient.

The paper, after recalling some basic notions of *HA* and Temporal Logics (Section 2 and 3), expounds by an example the proposed technique (Section 4). Section 5 gives the refinement rules and proves their correctness. Section 6 contains some conclusive remarks.

2 Hybrid Automata

Let us assume a temporal domain $Time$ and positive real numbers as its values, namely $Time = \mathbb{R}^+$.

Let us assume a set of real variables Var. An *evaluation* over Var is a function $v : Var \to \mathbb{R}$. The set of all possible evaluations is denoted with V.

Transitions of *HA* are labeled by a symbol, representing the action performed, and a set of *guards*, namely a subset of $V \times V$, where the first part represents the evaluation enabling the transition and the second part represents the new evaluation after the firing of the transition. Each state has an *invariant* and a set of *activities*. An invariant is a set of evaluations in V such that at least one of these evaluations must hold whenever the state is active. An activity is a function from $Time$ to V representing a possible evolution of the variables when the state is active. We shall denote with $\mathcal{F}$ the set of activities, i.e. $\mathcal{F} = \{f \mid f : Time \to V\}$.

Definition 1. *A* Hybrid Automaton *is a tuple* $(Q, q_0, Var, v_0, \Sigma, E, Act, Inv, F)$, *where:*

- Q *is a finite set of* states.
- $q_0 \in Q$ *is the* initial state.
- Var *is a finite set of* variables.
- v_0 *is the* initial evaluation *over* Var.
- Σ *is a finite set of* action symbols.
- $E \subseteq Q \times \Sigma \times 2^{V \times V} \times Q$ *is a finite set of* transitions, *where* $2^{V \times V}$ *is the set of* transition guards.
- $Act : Q \to 2^{\mathcal{F}}$ *is a function which assigns to each state a set of* activities. *We require that the activities of each state are time-invariant, namely, that for all state* q *and* $f \in Act(q)$, *if* $t \in Time$, *then* $f + t \in Act(q)$, *where* $f + t$ *is the function such that* $(f + t)(t') = f(t + t')$.

– $Inv : Q \to 2^V$ *is a function which assigns to each state* q *an* invariant.
– $F \subseteq Q$ *is the set of repeated states (Büchi acceptance condition).*

Let us describe now the semantics of *HA*. An automaton evolves from configurations to configurations by performing steps.

A *configuration* is a pair (q, v), where q is a state in Q and v is an evaluation in V. Given two configurations (q, v) and (q', v'), a symbol $a \in \Sigma$ and a time $t \in Time$, there is a *step* from (q, v) to (q', v') at time t with action a, written $(q, v) \to_t^a (q', v')$, iff:

– there is a function $f \in Act(q)$ such that $f(0) = v$
– there is a transition $(q, a, (f(t), v'), q') \in E$
– for all $t' \in [0, t]$ it holds that $f(t') \in Inv(q)$
– $v' \in Inv(q')$.

A *run* r is an infinite sequence of steps $r = (q_0, v_0) \to_{t_0}^{a_0} (q_1, v_1) \to_{t_1}^{a_1} \ldots$. With $inf(r)$ we denote the set of states occurring infinitely many times in the run r. The *language* recognized by H is the set $\{r \,|\, r \text{ is a run of } H \text{ and } inf(r) \cap F \neq \emptyset\}$.

The class of *Linear Hybrid Automata* is the subclass of *HA* where transition guards, activities and invariants are expressed by linear terms (linear combination of variables with integer coefficients) and formulas (boolean combination of inequalities between terms). More formally, Linear Hybrid Automata are *HA* satisfying the following requirements:

– For all states $q \in Q$, $Act(q)$ can be expressed by a set of differential equations of the form $\dot{x} = k_x$, where k_x is an integer constant called *rate*.
– For all states $q \in Q$, $Inv(q)$ can be expressed as a linear formula over Var.
– For all transitions $e \in E$, the set of relations $Y \subseteq V \times V$ can be expressed by a guarded set of non-deterministic assignments $\psi \Rightarrow \{x := [\alpha_x, \beta_x] \,|\, x \in Var\}$, where ψ is a liner formula and α_x, β_x are linear terms.

In this paper we focus on Linear Hybrid Automata, and we represent each transition $e \in E$ with a tuple of the form (q, a, ψ, q'). In the following of the paper we continue to use the notions of Hybrid Automata and *HA*, but we refer to the restricted model of Linear Hybrid Automata.

3 Temporal Logics

Let P denote a set of *atomic propositions* that can be satisfied or not by states. From now on we extend all *HA* with a labeling function $\mu : Q \to 2^P$, which labels each state q with the set of atomic propositions that are satisfied in q.

Definition 2. *A PTL (see [5]) formula* ϕ *is defined as follows*

$$\phi ::= p \mid \psi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \bigcirc \phi \mid \phi_1 \mathcal{U} \phi_2$$

where $p \in P$, ψ *is a linear formula on* Var, *and* I *is an interval contained in* $\mathbb{R}$.

If $r = (q_0, v_0) \rightarrow_{t_0}^{a_0} (q_1, v_1) \rightarrow_{t_1}^{a_1} \ldots$ is a run, then with r^i we denote the run $(q_i, v_i) \rightarrow_{t_i}^{a_i} (q_{i+1}, v_{i+1}) \rightarrow_{t_{i+1}}^{a_{i+1}} \ldots$.

Let us now give the semantics of PTL. As all linear temporal logics, PTL is interpreted over runs. We write $r \models \phi$ when *the run* $r = (q_0, v_0) \rightarrow_{t_0}^{a_0} (q_1, v_1) \rightarrow_{t_1}^{a_1} \ldots$ *satisfies the formula* ϕ. The relation $\models$ is defined inductively as follows:

$r \models true$
$r \models p$ iff $p \in \mu(q_0)$
$r \models \psi$ iff $v_0 \models \psi$
$r \models \neg\phi$ iff $r \not\models \phi$
$r \models \phi_1 \vee \phi_2$ iff either $r \models \phi_1$ or $r \models \phi_2$
$r \models \phi_1 \wedge \phi_2$ iff both $r \models \phi_1$ and $r \models \phi_2$
$r \models \bigcirc\phi$ iff $r^1 \models \phi$
$r \models \phi_1 \mathcal{U} \phi_2$ iff there is some $j \geq 0$ such that: $r^i \models \phi_1$ for all $i \in [0, j-1]$, $r^j \models \phi_2$.

An automaton H satisfies a formula ϕ, written $H \models \phi$, if and only if $r \models \phi$ holds for each run r of H.

4 Verification by refinement: An example

Let ϕ be a formula and T be a Timed Automaton. A known technique to check whether T satisfies ϕ is to build an automaton $T^{\neg\phi}$ that can perform all the runs that do not satisfy ϕ, and to check whether the Cartesian product $T \times T^{\neg\phi}$ has an empty set of runs (see [11, 8, 5, 3]).

This strategy is sound for *HA* but emptiness is only semi-decidable.

Let ϕ be a PTL formula and H be an *HA*. Our strategy to check whether H satisfies ϕ is the following: First of all we synthesize an automaton H^ϕ from ϕ by employing the synthesis algorithm of [11, 8, 5, 3] with linear formulae as atomic propositions. Hence, $H^\phi \models \phi$. Then, we apply a set of refinement operations to H^ϕ ensuring that all the resulting *HA* have a subset of the runs of H^ϕ. These refinement operations are chosen from a set of refinement rules. Now, if H can be derived from these transformations, then $H \models \phi$. Otherwise, we cannot infer anything.

Hence, both our strategy and that of [11, 8, 5, 3] are sound. The strategy of [11, 8, 5, 3] is automatic (i.e. model checking), our strategy requires an expert user able to choose the "right" refinement operations, and recalls theorem proving.

We explain our strategy by means of an example.

The automaton depicted in Figure 1 is the automaton used in [1] to describe a system that controls the temperature of the coolant in a reactor tank by moving independent rods. The goal is to maintain the coolant between the temperatures m and M. When the temperature reaches its maximum value M, the tank must be refrigerated with one of the rods. The temperature rises at rate v_r and decreases at rates v_1 and v_2 depending on which rod is being used. A rod can be moved again only if τ time units have elapsed since the end of its previous movement. If the temperature of the coolant cannot decrease because

there is no available rod, a complete shutdown is required. The clocks (i.e. real variables with rate 1) x and y represent the times elapsed since the last use of rod 1 and rod 2 respectively, while the variable t measures the temperature.

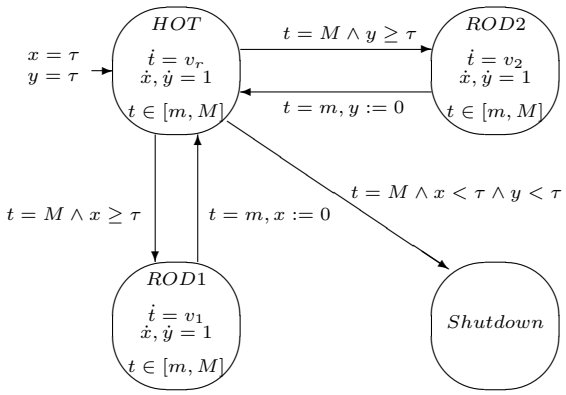

Fig. 1. Temperature control system

We want to prove that the automaton satisfies the formula

$$\Box((t = M \wedge x < \tau \wedge HOT) \Rightarrow \bigcirc(Shutdown \vee ROD2)) \tag{1}$$

Here, as usual, $\Box\phi$ stands for $false\,\mathcal{U}\,\phi$. Formula (1) states that, at any stage of the computation, if state HOT is active, the value of x is less than τ and the value of t is M, then the next step can take either to the state $ROD2$ (it is the case of $y \geq \tau$) or to the state $Shutdown$. Informally, if the first rod is not available and the temperature has reached M, then, either the second rod is activated, or the system shutdowns. From formula (1) we obtain the automaton in Figure 2 by means of the synthesis algorithm mentioned. This process may take into some consideration the structure of the automaton in Figure 1. We may suppose that the initial states coincide, and that the activity functions and the reset of the transitions are similar to those of the automaton of Figure 1.

We use a notation where transitions without label have guard $true/\{x := (-\infty, +\infty) \,|\, x \in Var\}$, and states with only proposition (as $\neg HOT$) have a function assigning to each variable in Var any possible continuous function and the invariant $true$. The algorithm gives to each state as activity the set of all possible functions and as invariant the set of all possible valuations. For each state, we have refined these sets with the corresponding sets defined in the temperature control systems. This transformation does not introduce new runs (i.e. either removes runs or maintains the same runs).

We can do some other transformations on the automaton of Figure 2, without introducing new runs.

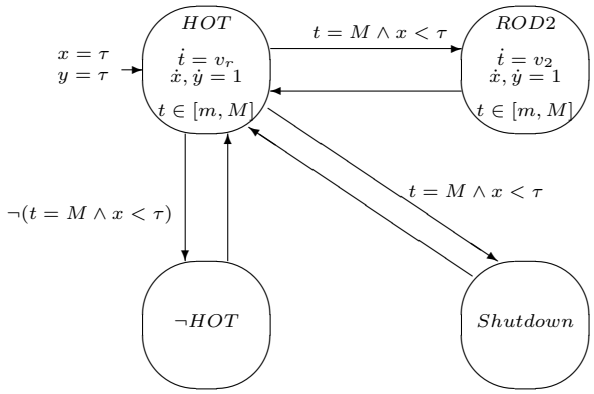

Fig. 2. Automaton for $\Box((t = M \wedge x < T \wedge HOT) \Rightarrow \bigcirc(Shutdown \vee ROD2))$

In Figure 3 we duplicate the state $\neg HOT$. This operation preserves the language.

Several operations on *HA*, such as deleting a transition, replacing a guard with a more demanding one, or specializing an activity with one enclosed, permit to obtain another automaton with a language contained in the original one. In Figure 4 we underline the refinement of this step. In Figure 5 we show the result. We note that $(t = M \wedge x < \tau \wedge y < \tau) \Rightarrow (t = M \wedge x < \tau)$ and $(t = M \wedge x \geq \tau) \Rightarrow (x \geq \tau) \Rightarrow \neg(t = M \wedge x < \tau)$.

Now, we join the two states named $ROD2$, we specialize the transition label $\neg(t = M \wedge x < \tau)$ with $x \geq \tau$, and we specialize the state $\neg HOT$ to obtain the state $ROD1$. In Figure 5 we underline the refinement of this step. In Figure 6 we show the result.

The last step consists in refining the transitions emphasized in Figure 6. It is obvious that for each ψ, $\psi \Rightarrow true$. The propositions $(t = M \wedge y \geq \tau) \Rightarrow (t = M)$ and $(t = M \wedge x \geq \tau) \Rightarrow (x \geq \tau)$ hold. Finally, if we specialize $x := I$ with the assignment $x := I'$ such that $I' \subset I$, we have a language contained in the original one. If we do this refinement on the automaton in Figure 6, then we get the automaton in Figure 1. We infer that the automaton in Figure 1 satisfies the property expressed by (1).

5 Refinement rules

In this section we formalize our system of refinement rules.

Given a hybrid automaton $H = (Q, q_0, Var, v_0, \Sigma, E, Act, Inv, F)$, we have the following rules.

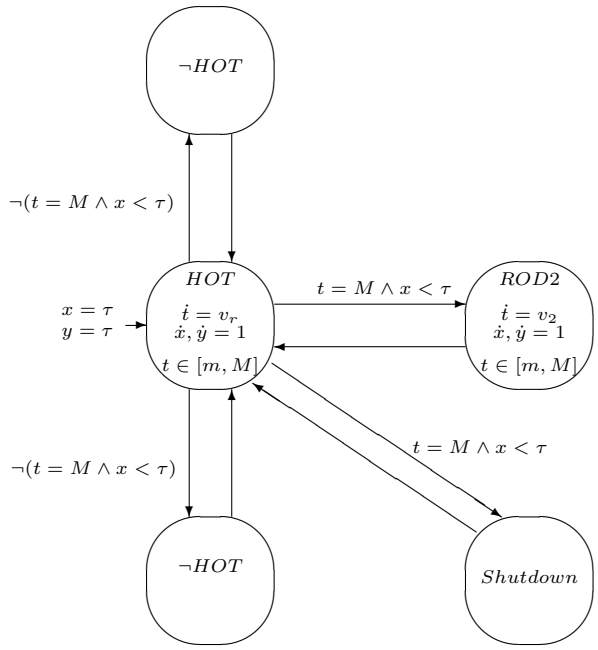

Fig. 3. Duplicating $\neg HOT$

r1) Transition Deletion
Given a transition $e \in E$ to be deleted, H is refined into

$$(Q, q_0, Var, v_0, \Sigma, E \setminus \{e\}, Act, Inv, F)$$

r2) Guard Refinement
Given a transition $e = (q, a, \varphi, q') \in E$ to be refined, and a formula $\varphi' \Rightarrow \varphi$, H is refined into

$$(Q, q_0, Var, v_0, \Sigma, (E \setminus \{e\}) \cup (q, a, \varphi', q'), Act, Inv, F)$$

r3) Activity Refinement
Given a state $q \in Q$ whose activities are to be refined, and a set of activities A_q such that $A_q \subseteq Act(q)$, H is refined into

$$(Q, q_0, Var, v_0, \Sigma, E, Act', Inv, F)$$

where $Act'(\bar{q}) = \begin{cases} Act(\bar{q}) & \bar{q} \neq q \\ A_q & \bar{q} = q \end{cases}$

r4) Invariant Refinement
Given a state $q \in Q$ whose invariant is to be refined, and a set of evaluations V_q such that $V_q \subseteq Inv(q)$, H is refined into

$$(Q, q_0, Var, v_0, \Sigma, E, Act, Inv', F)$$

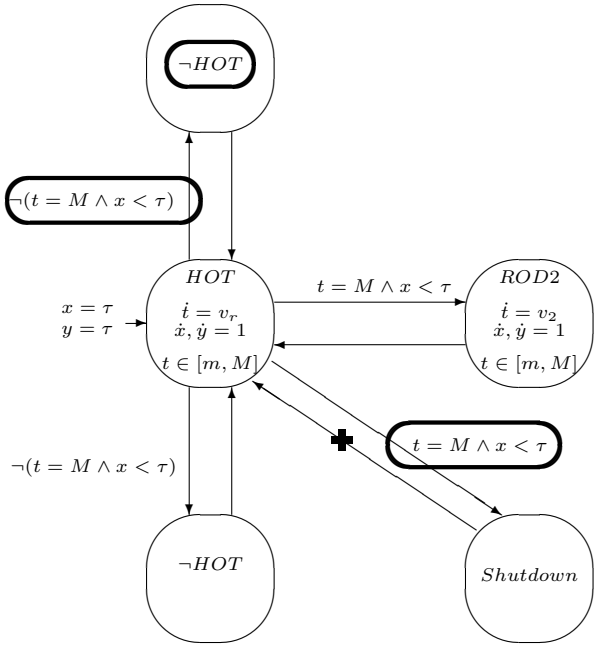

Fig. 4. Transition guards and activities to be strengthened

where $Inv'(\bar{q}) = \begin{cases} Inv(\bar{q}) & \bar{q} \neq q \\ V_q & \bar{q} = q \end{cases}$

r5) State Duplication
Given a state $q \in Q$ to be duplicated, H is refined into

$$(Q', q_0, Var, v_0, \Sigma, E', Act', Inv', F)$$

where

$$\begin{aligned}
&Q' = Q \cup \{q_d\} \qquad q_d \notin Q \\
&E' = E \cup E_q \\
&E_q = \{(q_d, a, \psi, \bar{q}) \mid (q, a, \psi, \bar{q}) \in E\} \cup \{(\bar{q}, a, \psi, q_d) \mid (\bar{q}, a, \psi, q) \in E\} \\
&Act'(\bar{q}) = \begin{cases} Act(\bar{q}) & \bar{q} \neq q_d \\ Act(q) & \bar{q} = q_d \end{cases} \\
&Inv'(\bar{q}) = \begin{cases} Inv(\bar{q}) & \bar{q} \neq q_d \\ Inv(q) & \bar{q} = q_d \end{cases}
\end{aligned}$$

r6) Joining of States
Given two states $q_1, q_2 \in Q$ to be joined such that $Inv(q_1) = Inv(q_2)$ and $Act(q_1) = Act(q_2)$, if one of the following conditions holds:

$$\begin{aligned}
&i)\ \ \forall(q, a, \psi, q_1) \in E\ \exists(q, a, \psi, q_2) \in E \wedge \forall(q, a, \psi, q_2) \in E\ \exists(q, a, \psi, q_1) \in E \\
&ii)\ \forall(q_1, a, \psi, q) \in E\ \exists(q_2, a, \psi, q) \in E \wedge \forall(q_2, a, \psi, q) \in E\ \exists(q_1, a, \psi, q) \in E
\end{aligned}$$

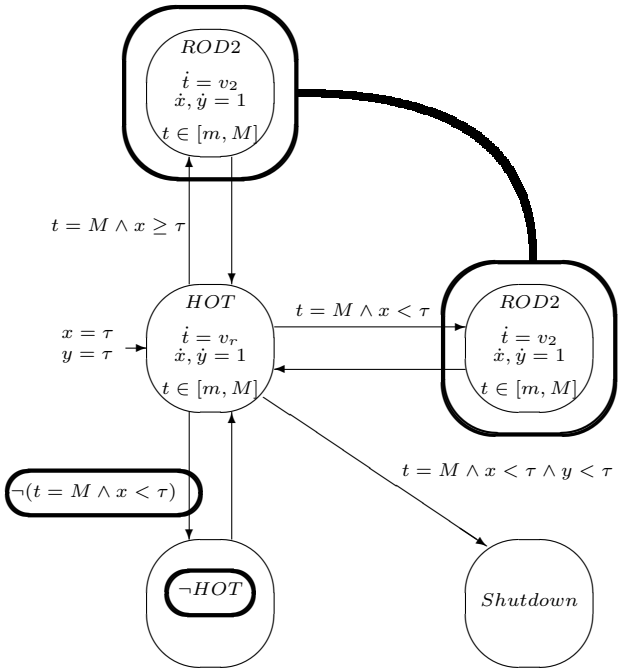

Fig. 5. New transition guards and activities to be strengthened; states to be joined

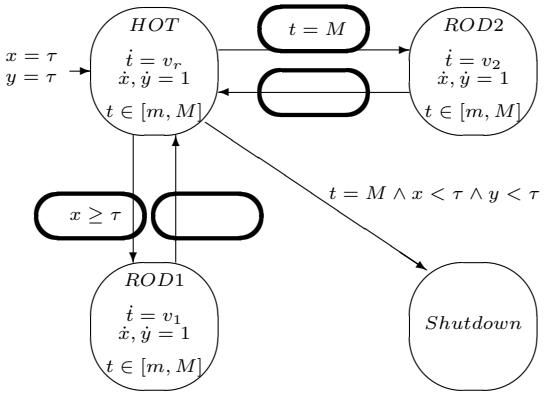

Fig. 6. Last step: further transition guards to be strengthened

H is refined into

$$(Q', q_0, Var, v_0, \Sigma, E', Act', Inv', F)$$

where:

$$Q' = (Q \setminus \{q_1, q_2\}) \cup \{q_f\} \qquad q_f \notin Q$$
$$E' = (E \setminus E_{q_1 q_2}) \cup E_f$$
$$E_{q_1 q_2} = \{(q, a, \psi, q') \in E \mid q \in \{q_1, q_2\} \vee q' \in \{q_1, q_2\}\}$$
$$E_f = \{(q, a, \psi, q_f) \mid (q, a, \psi, q') \in E,\ q \notin \{q_1, q_2\},\ q' \in \{q_1, q_2\}\} \cup$$
$$\{(q_f, a, \psi, q) \mid (q', a, \psi, q) \in E,\ q \notin \{q_1, q_2\},\ q' \in \{q_1, q_2\}\} \cup$$
$$\{(q_f, a, \psi, q_f) \mid (q', a, \psi, q) \in E,\ q, q' \in \{q_1, q_2\}\}$$
$$Act'(\bar{q}) = \begin{cases} Act(\bar{q}) & \bar{q} \neq q_f \\ Act(q_1) & \bar{q} = q_f \end{cases}$$
$$Inv'(\bar{q}) = \begin{cases} Inv(\bar{q}) & \bar{q} \neq q_f \\ Inv(q_1) & \bar{q} = q_f \end{cases}$$

The transformations applied in the previous section to the automaton in Figure 2 are instantiations of the rules r1–r6.

5.1 Correctness

In this subsection we prove that, by applying any of the previous refinement rules to any automaton, we get an automaton accepting a sublanguage of the language accepted by the original one.

Given a hybrid automaton $H = (Q, q_0, Var, v_0, \Sigma, E, Act, Inv, F)$, the automaton H' obtained by applying to H one of the rules r1, r2, r3 or r4, trivially accepts a language contained in the language accepted by H. In fact, these rules put only stronger conditions on the transitions and on the states of H. As a consequence, each run of H' can be performed also by H. Denoting with $\mathcal{L}(H)$ the language accepted by H, we have the following proposition.

Theorem 1. *Given a hybrid automaton $H = (Q, Var, \Sigma, E, Act, Inv, F)$ and the automaton H' obtained from H by applying any of the rules r1, r2, r3, r4, it holds that $\mathcal{L}(H') \subseteq \mathcal{L}(H)$.*

When duplicating a state of the automaton H according to rule r5, we get an automaton H' that can execute runs passing through the new state q_d not belonging to H. Hence, these runs could not be executed by H. But, if we consider the language accepted by H', since all transitions incoming (outgoing) to (from) state q_d have a counterpart transition incoming (outgoing) to (from) state q, we have that words obtained by runs passing through the state q_d are equal to words obtained by replacing in each run the state q_d with the state q. Runs obtained in this way are obviously executable by H. As a consequence, the language accepted by H' is equal to the language accepted by H.

Theorem 2. *Given a hybrid automaton $H = (Q, Var, \Sigma, E, Act, Inv, F)$ and the automaton H' obtained from H by applying rule r5, it holds that $\mathcal{L}(H') = \mathcal{L}(H)$.*

Proof. Trivially $\mathcal{L}(H) \subseteq \mathcal{L}(H')$ (see definition of r5). Given a run $r = (q_0, v_0) \rightarrow^{a_0}_{t_0} (q_1, v_1) \rightarrow^{a_1}_{t_1} (q_2, v_2) \rightarrow \ldots$ executable by T' we have the following cases:

$$\begin{array}{ll} a) & q_i \neq q_d \forall i \\ b) & \exists i : q_i = q_d \end{array}$$

If a) holds, then r is an executable run either for automaton H and the word $a_0 a_1 \ldots \in \mathcal{L}(T)$. If b) holds, we may transform $r = \ldots \rightarrow (q_{i-1}, v_{i-1}) \rightarrow^{a_{i-1}}_{t_{i-1}} (q_d, v_i) \rightarrow^{a_i}_{t_i} (q_{i+1}, v_{i+1}) \rightarrow \ldots$ into $r' = \ldots \rightarrow (q_{i-1}, v_{i-1}) \rightarrow^{a_{i-1}}_{t_{i-1}} (q, v_i) \rightarrow^{a_i}_{t_i} (q_{i+1}, v_{i+1}) \rightarrow \ldots$ related to the same word $a_0 a_1 \ldots$. We repeat this procedure by induction on the steps of r until case a) is reached. So, for each run r executable by H' there exists a run r' executable by H accepting the same word, thus implying $\mathcal{L}(H') \subseteq \mathcal{L}(H)$. □

When we join two states according to rule r6, we have to check that one of the conditions *i*) or *ii*) holds. Condition *i*) implies that the two states q_1 and q_2 to fuse have the same incoming transitions, while condition *ii*) implies that they have the same outgoing transitions. We can apply rule r6 only if one of the above conditions holds. If this is the case, we get an automaton H' where states q_1 and q_2 are substituted by the state q_f, and all transitions incoming (outgoing) to (from) states q_1 and q_2 are transformed in transitions incoming (outgoing) to (from) state q_f. These considerations make easy to see that each run executable by automaton H' passing through the state q_f can be simulated by a run of H where each occurrence of q_f is replaced by q_1 or q_2. As a consequence, the language accepted by H' is equal to the language accepted by H.

Theorem 3. *Given a hybrid automaton $H = (Q, Var, \Sigma, E, Act, Inv, F)$ and the automaton H' obtained from H by applying rule r6, it holds that $\mathcal{L}(H') = \mathcal{L}(H)$.*

Proof. Claim 1: $\mathcal{L}(H') \subseteq \mathcal{L}(H)$.
Given a run $r = (q_0, v_0) \rightarrow^{a_0}_{t_0} (q_1, v_1) \rightarrow^{a_1}_{t_1} (q_2, v_2) \rightarrow \ldots$ executable by H' we have the following cases:

$$\begin{array}{ll} a) & q_i \neq q_f \forall i \\ b) & \exists i : q_i = q_f \end{array}$$

If a) holds, then r is an executable run either for automaton H and the word $a_0 a_1 \ldots \in \mathcal{L}(H)$. If b) holds, we may transform $r = \ldots \rightarrow (q_{i-1}, v_{i-1}) \rightarrow^{a_{i-1}}_{t_{i-1}} (q_f, v_i) \rightarrow^{a_i}_{t_i} (q_{i+1}, v_{i+1}) \rightarrow \ldots$ into $r' = \ldots \rightarrow (q_{i-1}, v_{i-1}) \rightarrow^{a_{i-1}}_{t_{i-1}} (q', v_i) \rightarrow^{a_i}_{t_i} (q_{i+1}, v_{i+1}) \rightarrow \ldots$ related to the same word $a_0 a_1 \ldots$, and where $q' \in \{q_1, q_2\}$, $(q_{i-1}, a, \psi, q') \in E$, and $(q', a, \psi, q_{i+1}) \in E$. We repeat this procedure by induction on the steps of r until case a) is reached. Each step of the run r obtained in this way is executable by the automaton H. In fact, by definition of rule r6, for each transition (q, a, ψ, q_f) of H', there exists a transition (q, a, ψ, q') of H such that $q' \in \{q_1, q_2\}$ and for each transition (q_f, a, ψ, q), there exists a transition (q', a, ψ, q) of H such that $q' \in \{q_1, q_2\}$.

So, for each run r executable by H' there exists a run r executable by H accepting the same word, implying $\mathcal{L}(H') \subseteq \mathcal{L}(H)$.

Claim 2: $\mathcal{L}(H) \subseteq \mathcal{L}(H')$.
Following the intuition of the previous proofs, each run executable by H could be transformed into a run related to the same word executable by H' where each occurrence of q_1 and q_2 is substituted with q_f. □

6 Conclusions

We have outlined a technique for verification of Hybrid Automata based on synthesis and refinement. The synthesis of an automaton from a formula in Temporal Logics is a well established technique. The refinement is done by means of rules whose correctness is proved. The method is proposed as an alternative to verification based on the Cartesian product with the automaton synthesized from the negative of the formula expressing the property we want to verify, and on the proof that the product automaton has an empty set of runs.

It is our purpose to continue our investigation to find cases, if any, in which our technique is more convenient.

References

1. R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P. H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine: The Algorithmic Analysis of Hybrid Systems. Theor. Comp. Sci. 138(1), 1995, 3–34.
2. R. Alur and D. L. Dill: A Theory of Timed Automata. Theor. Comp. Sci. 126(2), 1994, 183–235.
3. R. Alur, T. Feder, and T. A. Henzinger: The Benefits of Relaxing Punctuality. J. ACM 43(1), 1996, 116–146.
4. R. Alur and T. A. Henzinger: Logic and Models of Real Time: A Survey. In Proc. Real Time: Theory in Practice, REX Workshop, Lecture Notes in Computer Science 600, Springer, Berlin, 1992, 74–106.
5. R. Alur and T. A. Henzinger: A Really Temporal Logic. J. ACM 41(1), 1994, 181–204.
6. R. Alur, T. A. Henzinger, and P. H. Ho: Automatic Symbolic Verification of Embedded Systems. IEEE Trans. Software Eng. 22(6), 1996, 181–201.
7. R. Lanotte, A. Maggiolo-Schettini, A. Peron, and S. Tini: Transformations of Timed Cooperating Automata. Fundam. Inform. 47(3–4), 2001, 271–282.
8. F. Laroussinie, K. G. Larsen, and C. Weise: ¿From Timed Automata to Logic - And Back. In Proc. Int. Symp. on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science 969, Springer, Berlin, 1995, 27–41.
9. A. Maggiolo-Schettini and A. Peron: Retiming Techniques for Statecharts. In Proc. Int. Conf. on Formal Technique in Real Time and Fault Tolerant Systems, Lecture Notes in Computer Science 1135, Springer, Berlin, 1996, 55–71.
10. A. Pnueli and J. Sifakis: Special issue on Hybrid Systems. Theor. Comput. Sci. 138(1), 1995.
11. P. Wolper: Constructing Automata from Temporal Logic Formulas: A tutorial. Lectures on formal methods and performance analysis: First EEF/Euro Summer School on Trends in Computer Science, Springer, Berlin, 261–277, 2002.