Model-checking for hybrid systems by quotienting and constraints solving

Franck Cassez[1] and François Laroussinie[2]

[1] IRCCyN, EC Nantes & CNRS UMR 6597, France, email:
Franck.Cassez@irccyn.ec-nantes.fr
[2] LSV, ENS de Cachan & CNRS UMR 8643, France, email:
Francois.Laroussinie@lsv.ens-cachan.fr

Abstract. In this paper we present a semi-algorithm to do compositional model-checking for hybrid systems. We first define a modal logic L_ν^h which is expressively complete for linear hybrid automata. We then show that it is possible to extend the result on compositional model-checking for parallel compositions of finite automata and networks of timed automata to linear hybrid automata. Finally we present some results obtained with an extension of the tool CMC to handle a subclass of hybrid automata (the stopwatch automata).

1 Introduction

Model checking for timed and hybrid systems. Model-checking algorithms for finite-state automata have been extended to *timed automata* [ACD93] and tools like KRONOS [Yov97] or UPPAAL [LPY97] have been used successfully also for verifying many industrial applications [BGK+96,MY96]. Hybrid systems [ACH+95] are a strong extension of timed automata and model-checking (or reachability) is undecidable [HKPV98] for these models. Nevertheless semi-algorithms have been implemented in tools like HyTech [HHWT97].

Heuristics for model-checking. In the timed verification area, model-checking is decidable but its complexity is high (PSPACE-complete or EXPTIME-complete) [ACD93,AL99]. A lot of works deal with heuristics to overcome this complexity blow-up (symbolic approaches [HNSY94], efficient and compact data structures [BLP+99], on-the-fly algorithms [BTY97] etc). From a practical point of view it is interesting to have different methods to verify a system: an approach can be very efficient over some classes of systems while another one works well for other classes. This last point is crucial for hybrid systems as model-checking is undecidable: for instance the so-called *forward* and *backward* reachability analysis algorithms [ACH+95,AHH96] are complementary.

Compositional Model-Checking. An alternative method to standard model-checking is *compositional model-checking* [And95,LL95,LPY95]. Given a system $S = (H_1|\cdots|H_n)$ and a property φ, the method consists in building a *quotient* formula φ/H_n s.t. $(H_1|\cdots|H_n) \models \varphi$ iff $(H_1|\cdots|H_{n-1}) \models \varphi/H_n$. Some simplification techniques can be applied in order to keep a small size for φ/H_n. By quotienting every component of the system one after the other, the remaining problem (if the last formula is not reduced to tt nor ff by simplifications) is to check a quotient property φ' on the *nil* process which is an automaton that cannot do any action. Figure 1 describes the two steps of the method.

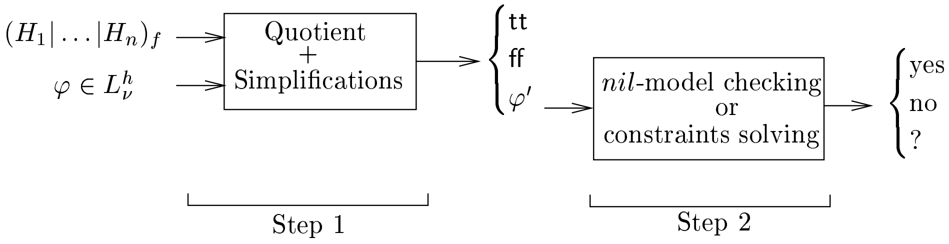

Fig. 1. Compositional Model Checking overview.

Our contribution. We propose here a compositional algorithm for linear hybrid automata. First we introduce a new modal logics L_ν^h (a kind of hybrid μ-calculus) which uses variables. L_ν^h allows us to express many kinds of properties, in particular it can be used to do compositional model-checking. Then we present reduction strategies to simplify L_ν^h formulas. When H_i's are hybrid automata, the two steps of the compositional method may not terminate (they require to compute fixed points over polyhedras), but it is possible to use abstractions for the first step in such a way that (1) termination is ensured and (2) $S \models \varphi \Leftrightarrow nil \models \varphi/S$ still holds. Then any problem $(H_1|\ldots|H_n) \models \varphi$ can be reduced to a nil-model-checking problem. Moreover we will see that this last step can be seen as a kind of constraints solving problem. Finally we present results obtained with a prototype HCMC which deals with hybrid automata where the slopes of variables belong to $\{0, 1\}$.

2 Hybrid automata

Notations. Let V and V' be two finite sets of variables with $V \cap V' = \emptyset$. A *valuation* is a mapping from a set V of variables into $\mathbb{R}$. For two valuations $v \in \mathbb{R}^V$, $v' \in \mathbb{R}^{V'}$, we define the valuation $v.v' \in \mathbb{R}^{V \cup V'}$ by $(v.v')(x) = v(x)$ if $x \in V$ and $(v.v')(x) = v'(x)$ if $x \in V'$. If $|V| = n$, a valuation v can be interpreted as a vector $\overline{v}$ of $\mathbb{R}^n$. We also recall the following useful definitions:

- A linear expression over V is of the form $a + \sum_i a_i v_i$ with $a, a_i \in \mathbb{Z}, v_i \in V$. The set of *linear constraints* $\mathcal{C}(V)$ over V is the set of formulas built using boolean connectives over expressions of the form $e_1 \bowtie e_2$ where e_1 and e_2 are linear expressions and $\bowtie$ belongs to $\{=, <, >, \leq, \geq\}$. Given a valuation v and a linear constraint γ, the boolean value $\gamma(v)$ describes whether γ is satisfied by v or not.

- A *linear assignment* over V is of the form $\overline{v} := A.\overline{v} + b$ where A is a $n \times n$ matrix with coefficients in $\mathbb{Z}$ and b is a vector of $\mathbb{Z}^n$. We denote an assignment α by a pair (A, b) and write $\overline{v} := \alpha(\overline{v})$ for $\overline{v} := A.\overline{v} + b$. $\mathcal{L}(V)$ is the set of linear assignments.
- A *continuous change* of variables is defined w.r.t. an element d of $\mathbb{Z}^V$ (hereafter referred to as the activity vector or direction) corresponding to the first derivative of each variable: given $t \in \mathbb{R}_{\geq 0}$, the valuation $v + d.t$ is defined by $(v + d.t)(x) = v(x) + d(x).t$.

We will need a particular property on subsets of $\mathbb{R}^n$ we refer to as d-strongly connection for some $d \in \mathbb{Z}^n$: A region $r \subseteq \mathbb{R}^n$ is *d-strongly connected* iff $\forall v, v' \in r$ $v' = v + d.t$ (for some $t \in \mathbb{R}_{\geq 0}$) implies $\forall\, 0 \leq t' \leq t, v + d.t' \in r$. This means that if one can go from one point in r to another also in r following the direction d then all the intermediate points along the path are in r. If $\mathsf{Future}(r, d)$ (resp. $\mathsf{Past}(r, d)$) denotes the future (resp. past) extension of r in direction d, then r is d-strongly connected is equivalent to $r = \mathsf{Future}(r, d) \cap \mathsf{Past}(r, d)$ (then this condition can be effectively checked for).

The model. Hybrid automata [ACH+95,AHH96,Hen96] are used to model systems which combine *discrete* and *continuous* evolutions.

Definition 1 (Hybrid automaton) *A* hybrid automaton *H is a 7-tuple $(N, l_0, V, A, E, Act, Inv)$ where:*

- *N is a finite set of* locations*,*
- *$l_0 \in N$ is the* initial location*,*
- *V is a finite set of real-valued* variables*,*
- *A is a finite set of* actions*,*
- *$E \subseteq N \times \mathcal{C}(V) \times A \times \mathcal{L}(V) \times N$ is a finite set of* edges*; $e = \langle l, \gamma, a, \alpha, l' \rangle \in E$ represents an edge from the location l to the location l' with the guard γ, the label a and the linear assignment α.*
- *$Act \in (\mathbb{Z}^V)^N$ assigns an* activity *vector for V to any location. $Act(l)(x)$ represents the first derivative of x in location l.*
- *$Inv \in \mathcal{C}(V)^N$ assigns an* invariant *to any location. We require that for any l, $Inv(l)$ is $Act(l)$-strongly connected.* □

Example 1. As a running example we will use the scheduler given in [AHH96] and depicted[1] on Figures 2 and 3. This scheduler *Sched* (Figure 3) can handle two types of tasks: type T_1 and type T_2. Priority is given to tasks of type T_2 and in case a task of type T_1 is currently running it is preempted: then measuring the execution time of tasks of type T_1 requires the use of a stopwatch y_1. We also use a stopwatch y_2 to measure the execution time of tasks of type T_2 (although it is not necessary as tasks of type T_2 cannot be preempted.) Obviously, y_i will be running at rate 1 in location T_i and 0 otherwise. Tasks of type T_1 take 4 time units and 8 time units for T_2. The initial state of *Sched* is *Idle*.

[1] The automata are designed using the GasTeX package available at http://www.liafa.jussieu.fr/~gastin/gastex.

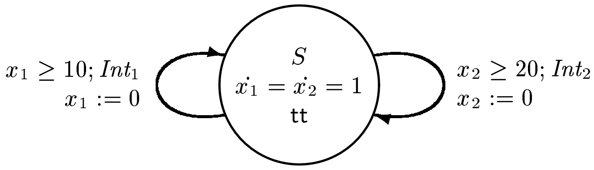

Fig. 2. Automaton of the environment: *Env*

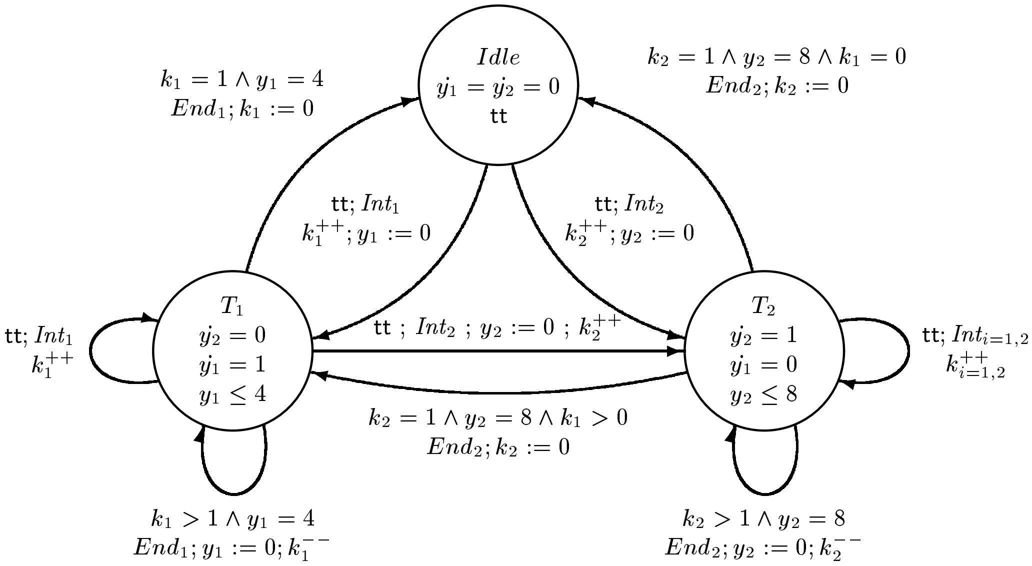

Fig. 3. Automaton of the scheduler: *Sched*

The number of pending and running tasks of type i is given by the integers[2] k_i (integers are implemented as stopwatch with slope 0 in every location). The arrivals (events Int_1 and Int_2) of the tasks are described by the (timed) automaton *Env* (Figure 2). This automaton specifies that the interval between two consecutive arrivals of tasks of type T_1 (resp. T_2) is more than 10 (resp. 20) time units. □

Semantics of hybrid automata. The semantics of a hybrid automaton is given by an (infinite) transition system. At any time, the *configuration* of the system is a pair (l, v) where l is a location and v a valuation. The configuration can change in two different ways: **a discrete change** can occur when a transition in E is enabled in the configuration (l, v), and **a continuous change** can occur

[2] On the figures, k_i^{++} stands for $k_i := k_i + 1$ and k_i^{--} for $k_i := k_i - 1$.

according to the evolution law of the variables (given by $Act(l)$) and as long as the invariant $Inv(l)$ remains true. The initial configuration of the hybrid automaton is (l_0, v_0) with $v_0 = \{0\}^V$ (i.e. $v_0(x) = 0\ \forall x \in V$).

Definition 2 (Semantics of a hybrid automaton) *The semantics of a hybrid automaton $H = (N, l_0, V, A, E, Act, Inv)$ is a labeled transition system $S_H = (Q, q_0, \to)$ with $Q = N \times \mathbb{R}^V$, $q_0 = (l_0, v_0)$ is the initial state ($v_0(x) = 0$, $\forall x \in V$) and $\to$ is defined by:*

$$\langle l, v\rangle \xrightarrow{\ a\ } \langle l', v'\rangle \quad \text{iff } \exists\, (l, \gamma, a, \alpha, l') \in E \ s.t. \begin{cases} \gamma(v) = \textit{tt},\ v' = \alpha(v) \ \textit{and} \\ Inv(l')(v') = \textit{tt} \end{cases}$$

$$\langle l, v\rangle \xrightarrow{\ \epsilon(t)\ } \langle l', v'\rangle \text{ iff } \begin{cases} l = l' \qquad v' = v + Act(l).t \quad \textit{and} \\ \forall\, 0 \le t' \le t,\ Inv(l)(v + Act(l).t') = \textit{tt} \end{cases}$$

A run *of a hybrid automata H is a path in S_H.* □

Remark 1. Due to the $Act(l)$-connectedness, the condition $\forall 0 \le t' \le t,\ Inv(l)(v + Act(l).t') = \mathtt{tt}$ is equivalent to $Inv(l)(v + Act(l).t) = \mathtt{tt}$ whenever $Inv(l)(v) = \mathtt{tt}$.

Parallel composition of hybrid automata. It is convenient to describe a system as a parallel composition of hybrid automata. To this end, we use the classical composition notion based on a *synchronization function* à la Arnold-Nivat [Arn94]. Let $H_1, \ldots, H_n$ be n hybrid automata with $H_i = (N_i, l_{i,0}, V_i, A, E_i, Act_i, Inv_i)$. A *synchronization function* f is a partial function from $(A \cup \{\bullet\})^n \hookrightarrow A$ where $\bullet$ is a special symbol used when an automaton is not involved in a step of the global system. Note that f is a synchronization function with renaming. We denote by $(H_1|\ldots|H_n)_f$ the parallel composition of the H_i's w.r.t. f. The configurations of $(H_1|\ldots|H_n)_f$ are pairs $(\bar{l}, v)$ with $\bar{l} = (l_1, \ldots, l_n) \in N_1 \times \ldots \times N_n$ and $v = v_1 \cdots v_n$ with[3] $v_i \in \mathbb{R}^{V_i}$ (we assume that all sets V_i of variables are disjoint.) Then the semantics of a synchronized product is also a transition system: the synchronized product can do a discrete transition if all the components agree to and time can progress in the synchronized product also if all the components agree to. This is formalized by the following definition:

Definition 3 (Parallel composition of hybrid automata) *Let $H_1, H_2, \ldots, H_n$ be n hybrid automata with $H_i = (N_i, l_{i,0}, V_i, A, E_i, Act_i, Inv_i)$, and f a (partial) synchronization function $(A \cup \{\bullet\})^n \hookrightarrow A$. The semantics of $(H_1|\ldots|H_n)_f$ is a labeled transition system $S = (Q, q_0, \to)$ with $Q = N_1 \times \ldots \times N_n \times \mathbb{R}^V$, q_0 is the initial state $((l_{1,0}, \ldots, l_{n,0}), v_0)$ and $\to$ is defined by:*

- $(\bar{l}, v) \xrightarrow{\ b\ } (\bar{l'}, v')$ *iff there exists $(a_1, \ldots, a_n) \in (A \cup \{\bullet\})^n$ s.t. $f(a_1, \ldots, a_n) = b$ and for any i we have:*
  - . *If $a_i = \bullet$, then $l'_i = l_i$ and $v'_i = v_i$,*
  - . *If $a_i \in A$, then $\langle l_i, v_i\rangle \xrightarrow{\ a_i\ } \langle l'_i, v'_i\rangle$.*

[3] v_i is the restriction of v to V_i.

– $(\bar{l}, v) \xrightarrow{\epsilon(t)} (\bar{l}, v')$ *iff* $\forall i \in [1..n]$, *we have* $\langle l_i, v_i\rangle \xrightarrow{\epsilon(t)} \langle l_i, v'_i\rangle$ □

Example 2. The synchronization function f for the parallel composition $(Sched \mid Env)$ of the scheduler and the environment of example 1 is given by: $f(Int_i, Int_i) = Int_i$ and $f(End_i, \bullet) = End_i$ for any i in $\{1, 2\}$.

3 L_ν^h: A modal logic for hybrid automata

3.1 Syntax of L_ν^h

We use a fixed point logic L_ν^h to specify the properties of hybrid automata. This logic is an extension (with variables) of the logic L_ν presented in [LL95]. It allows only maximal fixed points and consequently the specification of safety properties (this includes time-bounded liveness properties).

Definition 4 (L_ν^h: L_ν for hybrid systems) *Let K be a finite set of variables, Id a set of identifiers, and A an alphabet of actions. The set of L_ν^h formulas over K, A and Id is defined inductively by:*

$$L_\nu^h \ni \varphi, \psi \quad ::= \quad \varphi \vee \psi \mid \varphi \wedge \psi \mid \langle a\rangle\varphi \mid [a]\varphi \mid \langle\delta_d\rangle\varphi \mid [\delta_d]\varphi \mid \ \mathsf{Z} \mid \alpha \ \mathsf{in}\ \varphi \mid \gamma$$

where $a \in A$, $\alpha \in \mathcal{L}(K)$, $\gamma \in \mathcal{C}(K)$, $d \in \mathbb{Z}^K$ and $Z \in \mathsf{Id}$. □

3.2 Semantics of L_ν^h

It is straightforward to define the $\mathtt{tt}$ and $\mathtt{ff}$ operators and moreover implication $\varphi \Rightarrow \psi$ whenever no identifier occurs in φ (for example, $c \Rightarrow \varphi$ with $c \in \mathcal{C}(K)$) since this fragment is closed under negation. The meaning of identifiers is given by a declaration $\mathcal{D} : \mathsf{Id} \rightarrow L_\nu^h$.

Given a parallel composition S of hybrid automata, we interpret L_ν^h formula w.r.t. extended states $(\bar{l}, v, u)$ where $(\bar{l}, v)$ is a state of S and u is valuation for K variables (namely formula variables). Intuitively $\langle a\rangle$ (resp. $[a]$) denotes the existential (resp. universal) quantification over a-transitions, $\langle\delta_d\rangle$ (resp. $[\delta_d]$) denotes the existential (resp. universal) quantification over continuous transitions of S w.r.t. the direction d for the variables of K. The α in φ formula means that after the change of formula variables according to α ($\alpha \in \mathcal{L}(K)$), the new extended state verifies φ. The linear constraint γ over K variables holds for $(\bar{l}, v, u)$ whenever $\gamma(u) = \mathtt{tt}$. Finally Z holds for an an extended state, if it belongs to the largest solution of the equation $\mathsf{Z} = \mathcal{D}(\mathsf{Z})$. For formula of the type $x := 0$ in φ we simply write x in φ. Moreover when no formula variable occur in a formula, we write $\langle\delta\rangle$ (resp. $[\delta]$) instead of $\langle\delta_\emptyset\rangle$ (resp. $[\delta_\emptyset]$). Formally the semantics of L_ν^h is given in Table 1.

$$\begin{array}{lll}
\langle \bar{l}, v, u\rangle \models_{\mathcal{D}} \varphi \wedge \phi & \text{iff } \langle \bar{l}, v, u\rangle \models_{\mathcal{D}} \varphi \ \text{ and } \ \langle \bar{l}, v, u\rangle \models_{\mathcal{D}} \phi \\
\langle \bar{l}, v, u\rangle \models_{\mathcal{D}} \varphi \vee \phi & \text{iff } \langle \bar{l}, v, u\rangle \models_{\mathcal{D}} \varphi \ \text{ or } \ \langle \bar{l}, v, u\rangle \models_{\mathcal{D}} \phi \\
\langle \bar{l}, v, u\rangle \models_{\mathcal{D}} \langle a\rangle\varphi & \text{iff } \exists \langle \bar{l'}, v', u\rangle \ s.t. \ \langle \bar{l}, v\rangle \xrightarrow{a} \langle \bar{l'}, v'\rangle \text{ and } \langle \bar{l'}, v', u\rangle \models_{\mathcal{D}} \varphi \\
\langle \bar{l}, v, u\rangle \models_{\mathcal{D}} [a]\varphi & \text{iff } \forall \langle \bar{l'}, v', u\rangle, \langle \bar{l}, v\rangle \xrightarrow{a} \langle \bar{l'}, v'\rangle \text{ implies } \langle \bar{l'}, v', u\rangle \models_{\mathcal{D}} \varphi \\
\langle \bar{l}, v, u\rangle \models_{\mathcal{D}} \langle \delta_d\rangle\varphi & \text{iff } \exists t \in \mathbb{R}^{\geq 0} \ s.t. \ \langle \bar{l}, v\rangle \xrightarrow{\epsilon(t)} \langle \bar{l}, v + Act(\bar{l}).t\rangle \text{ and} \\
& \quad \langle \bar{l}, v + Act(\bar{l}).t, u + d.t\rangle \models_{\mathcal{D}} \varphi \\
\langle \bar{l}, v, u\rangle \models_{\mathcal{D}} [\delta_d]\varphi & \text{iff } \forall t \in \mathbb{R}^{\geq 0}, \langle \bar{l}, v\rangle \xrightarrow{\epsilon(t)} \langle \bar{l}, v + Act(\bar{l}).t\rangle \text{ implies} \\
& \quad \langle \bar{l}, v + Act(\bar{l}).t, u + d.t\rangle \models_{\mathcal{D}} \varphi \\
\langle \bar{l}, v, u\rangle \models_{\mathcal{D}} \gamma & \text{iff } \gamma(u) = \mathsf{tt} \\
\langle \bar{l}, v, u\rangle \models_{\mathcal{D}} \alpha \ \mathsf{in} \ \varphi & \text{iff } \langle \bar{l}, v, \alpha(u)\rangle \models_{\mathcal{D}} \varphi \\
\langle \bar{l}, v, u\rangle \models_{\mathcal{D}} \ \mathsf{Z} & \text{iff } \langle \bar{l}, v, u\rangle \text{ belongs to the maximal solution of } \mathsf{Z} = \mathcal{D}(\mathsf{Z})
\end{array}$$

Table 1. Semantics of the modal logic L_ν^h

3.3 Examples of L_ν^h formulas

As L_ν^h is a conservative extension of L_ν, we can derive classical temporal operators as in the examples below:

– To express that the action `error` is never performed, we can use the following equation:

$$X \stackrel{def}{=} \bigwedge_{a \in A} [a]X \wedge [\mathtt{error}]\mathsf{ff} \wedge [\delta]X$$

We denote this formula by $\mathsf{ALWAYS}_{\mathcal{A}}([\mathtt{error}]\mathsf{ff})$. In this example there is no formula variable and the $[\delta]$ deals only with continuous transitions [4] of the system S which is being specified.

– To express that the number of a-transitions is less than 100 along any run, we can use the formula: $x := 0 \ \mathsf{in} \ X$ with X defined by:

$$X \stackrel{def}{=} (x \leq 100) \wedge \bigwedge_{b \in A \setminus \{a\}} [b]X \wedge [a](x := x{+}1 \ \mathsf{in} \ X) \wedge [\delta_{\{\dot{x}=0\}}]X$$

In this case, x is just a discrete formula variable.

– More generally $\mathsf{ALWAYS}_{\mathcal{A},E}(\varphi)$ with $\mathcal{A} \subseteq A$ and $E \subseteq \mathbb{Z}^K$ a finite set of directions for K variables, states that φ holds from any reachable state using actions transition in $\mathcal{A}$ and delay transitions with derivatives in E for K variables. $\mathsf{ALWAYS}_{\mathcal{A},E}(\varphi)$ can be defined with the following equation:

$$X \stackrel{def}{=} \varphi \wedge \bigwedge_{b \in \mathcal{A}} [b]X \wedge \bigwedge_{e \in E} [\delta_e]X$$

[4] the operator does not contain the evolution law of automata variables since they only depend on the system.

– In the same manner, the weak until operator $\varphi_1 \, \mathsf{Until}_{\mathcal{A},E} \, \varphi_2$ is defined as the largest fixed point of:

$$X \stackrel{def}{=} \varphi_2 \vee \big(\varphi_1 \wedge \bigwedge_{b \in \mathcal{A}} [b]X \wedge \bigwedge_{e \in E} [\delta_e]X\big)$$

– As for timed automata, the following fixed point equation interpreted over a parallel composition of two hybrid automata (without invariant [5] H_1 and H_2 with the synchronization function $f(a, \bullet) = a_1$ and $f(\bullet, a) = a_2$ for any $a \in A$, expresses the (strong) bisimilarity of H_1 and H_2:

$$X \stackrel{def}{=} \bigwedge_{a \in A} [a_1]\langle a_2 \rangle X \wedge \bigwedge_{a \in A} [a_2]\langle a_1 \rangle X \wedge [\delta]X$$

Example 3. We want to specify on our scheduling system that a task of type T_2 never waits: as a new arrival of a task T_2 will preempt a running T_1 task, this amounts to check that $k_2 \leq 1$ as k_2 gives the number of pending and running T_2 tasks; we would also like to prove that at most one task of type T_1 may be pending i.e. $k_1 \leq 2$. We express the previous property as a reachability property of an `error` transition. Let Sched' be the hybrid automaton obtained from Sched by adding `error`-transitions $\langle l, (k_1 > 2 \vee k_2 > 1), \mathtt{error}, \emptyset, l \rangle$ for $l \in \{\mathit{Idle}, T_1, T_2\}$. It remains to check that `error`-transitions can not be fired i.e. $(\mathit{Env}|\mathit{Sched}')_{f'} \models \mathsf{ALWAYS}_{\mathcal{A}}([\mathtt{error}]\mathsf{ff})$ that is:

$$X \stackrel{def}{=} [\mathit{Int}_1]X \wedge [\mathit{Int}_2]X \wedge [\mathit{End}_1]X \wedge [\mathit{End}_2]X \wedge [\mathtt{error}]\mathsf{ff} \wedge [\delta]X$$

□

4 Compositional verification of hybrid automata

4.1 Quotient construction.

Given a hybrid system $(H_1 \mid \cdots \mid H_n)_f$ and a L_ν^h formula φ, we want to build a formula $\varphi/_f H_n$ s.t. $(H_1 \mid \cdots \mid H_n)_f \models \varphi$ iff $(H_1 \mid \cdots \mid H_{n-1}) \models \varphi/_f H_n$. The definition of the quotient construction is given in Table 2. Note that it is easier to define it w.r.t. a binary synchronization function between $(H_1 \mid \cdots \mid H_{n-1})$ and H_n but it is straightforward to decompose $(H_1 \mid \cdots \mid H_n)_f$ in such a manner.

For the no-action label, the conventions are the following: $\langle \bullet \rangle \varphi \equiv [\bullet]\varphi \equiv \varphi$. Note that the variables in V_n become formula variables in the quotient formula; this entails that the operators $\langle \delta_d \rangle$ and $[\delta_d]$ occurring in $\varphi/_f H_n$ deal with $K \cup V_n$. Moreover given $d \in \mathbb{Z}^K$ and $d' \in \mathbb{Z}^{V_n}$, $d.d'$ denotes the corresponding integer activity vector over $K \cup V_n$. Finally note that the quotienting of identifiers may increase [6] the number of fixed point equations in $\mathcal{D}'$; in the worst case, the size of $\mathcal{D}'$ is $|\mathcal{D}|.|N|$ where $|N|$ is the number of locations of the quotiented automaton. This will motivate the use of reduction methods. Now we have:

[5] The case of automata with invariant can be handled by a more complex formula based on the same idea.

[6] A new Z^l identifier can be added to $\mathcal{D}'$ for any location l and any $\mathsf{Z} \in \mathsf{Id}$.

$(\varphi_1 \wedge \varphi_2)/_{f}\, l = (\varphi_1/_{f}\, l) \wedge (\varphi_2/_{f}\, l)$	$(\varphi_1 \vee \varphi_2)/_{f}\, l = (\varphi_1/_{f}\, l) \vee (\varphi_2/_{f}\, l)$
$(\langle a \rangle \varphi)/_{f}\, l = \bigvee_{(l,\gamma,c,\alpha,l') \in E_n \;/\; f(b,c)=a} \gamma \wedge (\alpha \text{ in } Inv(l')) \wedge \langle b \rangle (\alpha \text{ in } (\varphi/_{f}\, l'))$	
$([a]\varphi)/_{f}\, l = \bigwedge_{(l,\gamma,c,\alpha,l') \in E_n \;/\; f(b,c)=a} \Big(\gamma \wedge (\alpha \text{ in } Inv(l'))\Big) \Rightarrow [b](\alpha \text{ in } (\varphi/_{f}\, l'))$	
$(\langle \delta_d \rangle \varphi)/_{f}\, l = \langle \delta_{d.Act(l)} \rangle (Inv(l) \wedge \varphi/_{f}\, l)$	
$([\delta_d]\varphi)/_{f}\, l = [\delta_{d.Act(l)}](Inv(l) \Longrightarrow \varphi/_{f}\, l)$	$\mathsf{Z}/_{f}\, l = \mathsf{Z}^l$
$(\alpha \text{ in } \varphi)/_{f}\, l = \alpha \text{ in } (\varphi/_{f}\, l)$	$\gamma/_{f}\, l = \gamma$

Table 2. Quotienting rules to obtain $\varphi/_{f}\, l$

Theorem 1. *Let $((H_1 \mid \cdots \mid H_{n-1})_{f'} \mid H_n)_f$ be a system of n hybrid automata $H_i = (N_i, l_{0,i}, V_i, A_i, E_i, Act_i, Inv_i)$ and φ an L_ν^h formula over K. If $\langle(\overline{\rho}, l), v.w\rangle$ is a configuration of $((H_1 \mid \cdots \mid H_{n-1})_{f'} \mid H_n)_f$ with $v \in \cup_{i<n} \mathbb{R}^{V_i}$, $w \in \mathbb{R}^{V_n}$ and $u \in \mathbb{R}^K$ s.t. $Inv(l)(w) = \mathbf{tt}$, then we have:*

$$\langle(\overline{\rho}, l), v.w, u\rangle \models_{\mathcal{D}} \varphi \qquad \textit{if and only if} \qquad \langle \overline{\rho}, v, w.u\rangle \models_{\mathcal{D}'} \varphi/_{f}\, l \qquad \square$$

A sketch of the proof of Theorem 1 is given in appendix A. The following definition shifts the meaning of the quotient to hybrid automata:

Definition 5 (Quotient of φ by a hybrid automaton) *Let $H = (N, l_0, V, A, E, Act, Inv)$ and $H' = (N', l_0', V', A', E', Act', Inv')$ be hybrid automata with v_0, v_0' their initial valuations and f a synchronization function for H and H'. Let φ be an L_ν^h formula with clocks over K, $\mathcal{D}$ a declaration and u_0 the valuation $\{0\}^K$. Then $(H \mid H')_f \models_{\mathcal{D}} \varphi$ iff $\langle(l_0, l_0'), (v_0.v_0'), u_0)\rangle \models_{\mathcal{D}} \varphi$. We also define the quotient $\varphi/_{f}\, H$ to be $\varphi/_{f}\, l_0$. Consequently $(H \mid H')_f \models_{\mathcal{D}} \varphi$ iff $H' \models_{\mathcal{D}'} (\varphi/_{f}\, H)$ where $\mathcal{D}'$ deals with the set of identifiers which are introduced during quotienting.* $\square$

Then verifying a system can be reduced to a nil-model checking problem [7]:

Corollary 1. *Let $(\cdots(H_1|H_2)_{f_1} \cdots |H_n)_{f_{n-1}}$ be an hybrid system, $\langle \bar{l_0},\, v_0\rangle$ its initial configuration, φ an L_ν^h formula, $\mathcal{D}$ a declaration and u_0 the valuation $\{0\}^K$. If $Inv(\bar{l_0})(v_0) = \mathbf{tt}$, we have: $\langle(l_{0,1}, \ldots, l_{0,n}), v_0, u_0\rangle \models_{\mathcal{D}} \varphi \;\Leftrightarrow\; nil \models_{\mathcal{D}'} (\varphi/_{f_{n-1}} H_n \cdots /_{f_0} H_1)$* $\square$

[7] nil can only let time elapse without performing any action, this is an automaton with no variable and no edge.

4.2 Simplification strategies.

The size of φ/H is in $O(|\varphi|.|H|)$ if we consider the formula as a dag (viz. represented by a data structure with sharing of sub-formula). Therefore the final quotient formula $\varphi/H_n/\ldots/H_1$ may be in size exponential in $|H_1|+\ldots+|H_n|+|\varphi|$. This blow-up of the quotient formula corresponds to the state explosion problem which occurs in classical model-checking approach (for example, complexity of model-checking for alternation free μ-calculus over product of classical automata is EXPTIME-complete [KVW98] while it is only P-complete for one automaton). We are going to apply syntactical and semantical reductions over the quotient formula after each quotienting in order (to try) to keep in the quotient formula φ/H only the part of the behavior of H which is relevant w.r.t. the property φ we want to check.
Many simplifications used for timed automata also apply here: *Boolean Simplifications* ($\mathsf{tt} \wedge \varphi \equiv \varphi$, $\langle a \rangle \mathsf{ff} \equiv \mathsf{ff}$, etc), *Trivial Equation Elimination* (equations of the form $X \stackrel{def}{=} [a]X \wedge [\delta_d]X$ have $X = \mathsf{tt}$ as solution etc.), *Equivalence Reduction* (if two identifiers X and Y are equivalent we may collapse them into a single identifier).
Hitzone Reduction consists in computing, for any atomic constraint $\xi \in \mathcal{C}(K)$ occurring in φ, an upper approximation of the set of valuations for K (i.e. a linear constraint S_ξ over the formula variables) where the truth value of ξ is needed to decide the truth value of φ for the initial configuration. These S_ξ sets are obtained by a (forward) fixed point computation. Afterwards, ξ can be replaced by tt (resp. ff) whenever [8] $S_\xi \subseteq [\![\xi]\!]$ (resp. $S_\xi \cap [\![\xi]\!] = \emptyset$).
We illustrate the hitzone reduction with the following example. Consider the formula $\varphi \stackrel{def}{=} X_0$ with:

$$\begin{aligned}
X_0 &\stackrel{def}{=} \Big(x_2 \le 3 \Rightarrow [a](x_2 := 0 \text{ in } X_1)\Big) \wedge \langle b \rangle (x_1 := 0 \text{ in } X_0) \wedge [\delta_{(1,1)}]X_0 \\
X_1 &\stackrel{def}{=} (x_2 \le 1 \ \Rightarrow\ [b]X_2) \wedge [\delta_{(0,1)}](x_2 \le 4 \ \Rightarrow\ X_1) \\
X_2 &\stackrel{def}{=} [c]X_2 \wedge (x_2 \ge x_1 - 3)
\end{aligned}$$

First we compute S_{X_0} (resp. S_{X_1}, S_{X_2}) corresponding to the set of K valuations where the truth value of X_0 (resp. X_1, X_2) is needed to decide whether φ holds for the initial configuration. This is done by a forward fixed point computation: we start with $S^0_{X_0} = [\![x_1 = x_2 = 0]\!]$ and $S^0_{X_1} = S^0_{X_2} = [\![\mathsf{ff}]\!]$, 4 iterations are necessary to obtain the result:

- $S^1_{X_0} = [\![x_1 = x_2]\!]$, $S^1_{X_1} = [\![x_1 = x_2 = 0]\!]$ and $S^1_{X_2} = [\![\mathsf{ff}]\!]$,
- $S^2_{X_0} = [\![x_1 = x_2 \vee x_1 = 0 \le x_2]\!]$,
  $S^2_{X_1} = [\![x_2 = 0 \le x_1 \le 3 \vee x_1 = 0 \le x_2 \le 4]\!]$ and $S^2_{X_2} = [\![x_1 = x_2 = 0]\!]$,
- $S^3_{X_0} = [\![0 \le x_1 \le x_2]\!]$, $S^3_{X_1} = [\![0 \le x_1 \le 3 \wedge 0 \le x_2 \le 4]\!]$ and $S^3_{X_2} = [\![x_2 = 0 \le x_1 \le x_3 \vee x_1 = 0 \le x_2 \le 1]\!]$,
- $S^4_{X_0} = [\![0 \le x_1 \le x_2]\!]$, $S^4_{X_1} = [\![0 \le x_1 \le 3 \wedge 0 \le x_2 \le 4]\!]$ and $S^4_{X_2} = [\![0 \le x_1 \le 3 \wedge 0 \le x_2 \le 1]\!]$,

[8] $[\![\xi]\!]$ denotes the set of valuations satisfying the linear constraint ξ.

This computation of sets $S^j_{X_i}$ is done by a forward propagation of linear constraint, the sub-formulas (α in φ), ($[\delta_d]\varphi$) and ($\gamma \Rightarrow \varphi$) are seen as *predicate transformers*.

Figure 4 shows the final value of S_{X_0}, S_{X_1} and S_{X_2}. Therefore the constraint $(x_2 \geq x_1 - 3)$ in X_2 can be reduced to tt since $S_{X_2} \subseteq [\![\, x_2 \geq x_1 - 3 \,]\!]$. Then trivial equation elimination allows us to reduce X_2 (resp. X_1) to tt. Finally we can simplify X_0 to $\langle b \rangle (x_1 := 0 \text{ in } X_0) \wedge [\delta_{(1,1)}] X_0$.

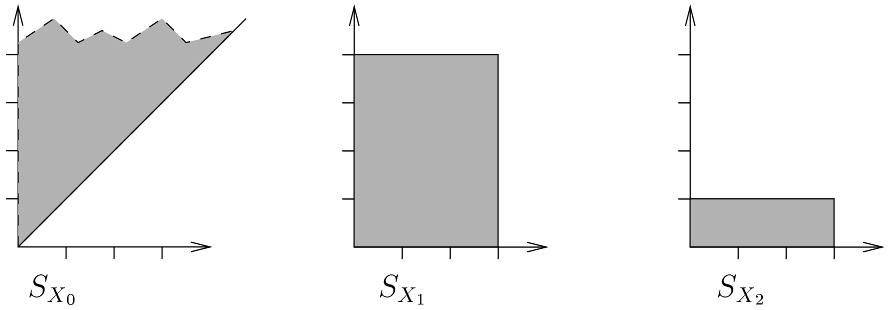

Fig. 4. Example of hitzones computation.

Of course, the computation of S_ξ is based on operations over polyhedra and may not terminate (contrary to other simplifications). Nevertheless it is possible to use coarser over-approximation of the S_ξ's to ensure termination, this clearly leads to less efficient simplifications (i.e. allowing less atomic constraints reductions).

Sharing variables. In the definitions of L^h_ν and H_i, we assume that the variables sets V_i and K are disjoint. This hypothesis is important only if hitzone simplification is applied because this reduction assumes that transitions of automata which have not yet been quotiented do not modify the value of formula variables (for ex. the sub-formula $(x < 10) \wedge \langle a \rangle (x > 10)$ is reduced to ff because it is assumed that x is not updated by performing the a-transition). If we do not apply the hitzone simplification, automata variables can be used inside the formula and variables can even be shared between several automata. In these cases, the hitzone reduction can only be applied after the quotienting of the last automaton that uses the shared variables (i.e. when all the control part they are concerned with is present in the quotient formula).

Nil-model-checking and constraints solving. Let φ' be $\varphi/H_n/\ldots/H_1$. Note that φ' expresses a property over *nil* and then we can assume [9] that no $\langle a \rangle$ and no $[a]$ occur in φ'. In fact, deciding whether φ' holds for *nil* requires to compute the fixed point of equations $X_i = \mathcal{D}'(X_i)$ where $\mathcal{D}'$ is the definition of identifiers occurring in φ'. But an extended configuration of *nil* is just an $|K'|$-tuple of real

[9] For *nil* process, we have $\langle a \rangle \varphi \equiv \mathsf{ff}$ and $[a]\varphi \equiv \mathsf{tt}$.

numbers and then $nil \models \varphi'$ can be seen as a constraints problem over sets of $|K'|$-tuple of real numbers. For example, $nil \models X_0$ with the following declaration:

$$X_0 \stackrel{def}{=} \Big[(x_2 = 1) \Rightarrow (x_2 := 0 \text{ in } X_1)\Big] \wedge (x_1 < 1 \vee x_1 > 1) \wedge [\delta_{(0,1)}](x_2 \leq 1 \Rightarrow X_0)$$
$$X_1 \stackrel{def}{=} \Big[(x_2 = 1) \Rightarrow (x_2 := 0 \text{ in } X_0)\Big] \wedge [\delta_{(1,1)}](x_2 \leq 1 \Rightarrow X_1)$$

is equivalent to the problem of deciding [10] whether $(0,0) \in S_0$ where S_0 and S_1 $(\subseteq \mathbb{R} \times \mathbb{R})$ are defined as the maximal sets verifying:

If $(x_1, x_2) \in S_0$ Then $(x_2 = 1 \Rightarrow (x_1, 0) \in S_1)$ and $(x_1 < 1 \vee x_1 > 1)$ and $(\forall t \geq 0, x_2 + t \leq 1 \Rightarrow (x_1, x_2 + t) \in S_0)$
If $(x_1, x_2) \in S_1$ Then $(x_2 = 1 \Rightarrow (x_1, 0) \in S_0)$ and $(\forall t \geq 0, x_2 + t \leq 1 \Rightarrow (x_1 + t, x_2 + t) \in S_1)$

Therefore in the compositional verification, the first step deals with an high level description of the problem (a parallel composition of hybrid automata and a specification written with L_ν^h) while the second step treats a more basic description and could be analyzed by a constraints solver over real numbers.

In [ACH+95] reachability problems of the form "is it possible to reach a configuration verifying φ_0 ?" (where φ_0 is an atomic constraint) are reduced to a fixed point computation of an equation system which encodes the behavior of the hybrid system. In fact if we apply our quotient technique over the reachability formula $\mathsf{ALWAYS}(\neg\varphi_0)$, we obtain the same kind of equations (it can be smaller thanks to the simplification step), and then the compositional method can be seen as an extension of the previous results over hybrid system since it deals with any kind of property which can be expressed with alternation-free modal μ-calculus.

5 Hybrid CMC and examples

Hybrid CMC . We extended CMC (a tool implementing the compositional method for timed automata [LL98]) in order to handle a subclass of hybrid systems where variables have slopes in $\{0, 1\}$. Moreover we use classical constraints of TA (viz. $x \bowtie m$ or $x - y \bowtie m$) and assignments are of the form $x := y + m$ or $x := m$. The same restrictions apply to the modal logic handled by HCMC. This subclass of hybrid system remains very expressive (model checking is clearly undecidable) and allows to model a large variety of systems [BF99a].
Given an hybrid system and a modal specification, HCMC allows us to build the simplified quotient formula. Moreover there is a procedure to (try to) solve *nil* model-checking problems. We use a DBM-like data structure to represent constraints over variables in the algorithms [Dil89]. This choice motivates the restriction to slopes in $\{0, 1\}$ (other slopes would require extended forms of constraints) but even in this framework some operations (like $\mathsf{Future}(d, z) = \{v + d.t \mid v \in$

[10] Here the answer is " no".

$z, t \in \mathbb{R}\}$) can only be approximated. These abstractions ensures termination of the two steps (simplifications and *nil* model-checking algorithm) of the compositional approach. However the problem being undecidable, an answer not in $\{\text{yes}, \text{no}\}$ may occur. This third result corresponds to cases where the abstractions used for the second step are too large to be able to conclude. The prototype HCMC is available at the web address: `http://www.lsv.ens-cachan.fr/~fl`.

Examples. The current version of HCMC has been applied over several examples. Here are the results:

- The scheduler described in Example 1 has been successfully verified: the quotienting of the formula in Example 3 by the two components is directly reduced to tt by the simplifications.
- We applied the method over a verification problem for two Petri Nets considered in the MARS project [11] and verified by HyTech (following the method of [BF99b]). Here the problem consists in verifying that a place always contains less than 2 tokens. The verification succeeds in every case (either directly after the first step, or after the *nil* model-checking computation).
- We have tried to verify the ABR protocol. We adapted the model used in [BF99a] for HyTech to HCMC. Here the first step gave a specification with 25 equations but the second step couldn't conclude due to approximations.

6 Conclusion and future work

In this paper we have extended the compositional model-checking method defined in [LL95] for timed automata to hybrid automata. Our work is two fold:

1. on the theoretical aspects we have proven that it is possible to do compositional model-checking for hybrid automata; we have defined the logic L_ν^h, i.e. an extension of modal μ-calculus, so that this logic can express many properties for specification of reactive systems and is expressively complete for linear hybrid automata.
2. we have implemented a prototype tool HCMC to handle verification of hybrid automata where variables have slopes in $\{0, 1\}$.

As described in section 5, our method consists in two distinct steps: quotienting and then constraints solving. Our current implementation uses DBMs as a data structure to represent the regions of $\mathbb{R}^n$. Obviously we are only able to manipulate over-approximation of the actual regions of our hybrid automata.

If this is no harm for step 1 as we only miss some simplifications, it is a real impediment for step 2: if the approximation is too coarse we are not able to prove some properties.

Our future work will consist in extending HCMC in order to deal with integer slopes without approximations (this could be done by using a tool like HyTech

[11] `http://www.loria.fr/~xie/Mars.html`

and a polyhedra library to do the final *nil*-model-checking and simplifications or a constraints solver over the reals). We can also easily extend our logic and algorithm to cope with slopes within integers intervals i.e. $\dot{x} \in [l, u]$ so that we will be able to use HCMC on real-life examples and compare the performances with other related tools.

Acknowledgment

The authors would like to thank Béatrice Bérard, Kim Guldstrand Larsen and the anonymous referees for their useful comments on this work.

References

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

[ACH+95] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science B*, 137, January 1995.

[AHH96] R. Alur, T. A. Henziger, and P. H. Ho. Automatic symbolic verification of embedded systems. *IEEE Transactions On Software Engineering*, 22(3):181–201, March 1996.

[AL99] L. Aceto and F. Laroussinie. Is your model checker on time ? In *Proc. 24th Int. Symp. Math. Found. Comp. Sci. (MFCS'99), Szklarska Poreba, Poland, Sep. 1999*, volume 1672 of *Lecture Notes in Computer Science*, pages 125–136. Springer, 1999.

[And95] H. R. Andersen. Partial Model Checking. In *Proc. of LICS'95*, 1995.

[Arn94] André Arnold. *Finite Transition System*. Prentice Hall, 1994.

[BF99a] B. Bérard and L. Fribourg. Automated verification of a parametric real-time program: the ABR conformance protocol. In *Proc. 11th Int. Conf. Computer Aided Verification (CAV'99), Trento, Italy, July 1999*, volume 1633 of *Lecture Notes in Computer Science*, pages 96–107. Springer, 1999.

[BF99b] B. Bérard and L. Fribourg. Reachability analysis of (timed) Petri nets using real arithmetic. In *Proc. 10th Int. Conf. Concurrency Theory (CONCUR'99), Eindhoven, The Netherlands, Aug. 1999*, volume 1664 of *Lecture Notes in Computer Science*, pages 178–193. Springer, 1999.

[BGK+96] J. Bengtsson, W. O. David Griffioen, K. J. Kristoffersen, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. Verification of an audio protocol with bus collision using uppaal. In *Proc. of the 8th International Conference on Computer-Aided Verification, LNCS 1102*, pages 244–256, 1996.

[BLP+99] G. Behrmann, K. G. Larsen, J. Pearson, C. Weise, and W. Yi. Efficient timed reachability analysis using clock difference diagrams. In *11th Computer-Aided Verification*, Trento, Italy, July 1999.

[BTY97] A. Bouajjani, S. Tripakis, and S. Yovine. On-the-Fly Symbolic Model Checking for Real-Time Systems. In *Proc. of the 18th IEEE Real-Time Systems Symposium, RTSS'97*. IEEE Computer Society Press, December 1997.

[Dil89] D.L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proc. Workshop Automatic Verification Methods for Finite State Systems, Grenoble, LNCS 407*, 1989.

[Hen96] T. A. Henzinger. The theory of hybrid automata. In *Proceedings, 11*$^{\text{th}}$ *Annual IEEE Symposium on Logic in Computer Science*, pages 278–292, New Brunswick, New Jersey, 27–30 July 1996. IEEE Computer Society Press.

[HHWT97] T. A. Henzinger, P. H. Ho, and H. Wong-Toi. HYTECH: A model checker for hybrid systems. *Lecture Notes in Computer Science*, 1254:460–??, 1997.

[HKPV98] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57(1):94–124, August 1998.

[HNSY94] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[KVW98] O. Kupferman, M. Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking, 1998. Full version of the CAV'94 paper, accepted for publication in J. ACM.

[LL95] F. Laroussinie and K.G. Larsen. Compositional Model-Checking of Real Time Systems. In *Proc. CONCUR'95, Philadelphia, USA, LNCS 962*, pages 27–41. Springer-Verlag, August 1995.

[LL98] F. Laroussinie and K. G. Larsen. CMC: A tool for compositional model-checking of real-time systems. In *Proc. IFIP Joint Int. Conf. Formal Description Techniques & Protocol Specification, Testing, and Verification (FORTE-PSTV'98)*, pages 439–456. Kluwer Academic Publishers, 1998.

[LPY95] K. G. Larsen, P. Pettersson, and W. Yi. Compositional and symbolic model-checking of real-time systems. In *Proc. of the 16th IEEE Real-Time Systems Symposium, RTSS'95*, 1995.

[LPY97] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *Journal of Software Tools for Technology Transfer*, 1(1/2):134–152, October 1997.

[MY96] O. Maler and S. Yovine. Hardware timing verification using KRONOS. In *Proc. 7th Israeli Conference on Computer Systems and Software Engineering*, Herzliya, Israel, June 1996.

[Yov97] S. Yovine. Kronos: A Verification Tool for real-Time Systems. *Journal of Software Tools for Technology Transfer*, 1(1/2):123–133, October 1997.

A Proof of theorem 1

For simplicity we restrict the proof to the case of two hybrid automata H_1 and H_2 and a synchronization function f. The proof is carried out by induction on the formula of L^h_ν. For all inductive definitions except $[\delta_d]\varphi$ and its dual $\langle\delta_d\rangle\varphi$ the induction steps are exactly the same as for timed automata (see [LL95]). We here focus on the case of the rule $[\delta_d]\varphi$ (the case $\langle\delta_d\rangle\varphi$ works in the same manner) involving continuous evolution as this case involves the main difference between hybrid and timed automata: the rates of the variables are values of $\mathbb{Z}$ and not always 1.

Proof (Rule $[\delta_d]\varphi$.). Let $\langle(l_1,l_2),v_1.v_2\rangle$ be a configuration of $(H_1|H_2)_f$. Given a $t\geq 0$, v_1' (resp. v_2') will denote $v_1 + Act(l_1).t$ (resp. $v_2 + Act(l_2).t$) and given an activity vector d, u' will denote $u + d.t$.

$\Rightarrow$. Assume $\langle (l_1,l_2), v_1.v_2, u\rangle \models_{\mathcal{D}} [\delta_d]\varphi$ and $Inv(l_2)(v_2) = \mathsf{tt}$.
Then by definition of L_ν^h semantics (Table 1), we have:

$$\forall\, t \geq 0, \langle (l_1,l_2), v_1.v_2\rangle \xrightarrow{\epsilon(t)} \langle (l_1,l_2), v_1'.v_2'\rangle \textit{ implies } \langle (l_1,l_2), v_1'.v_2', u'\rangle \models_{\mathcal{D}} \varphi$$

We have to show: $\langle l_1, v_1, v_2.u\rangle \models_{\mathcal{D}'} [\delta_{d.Act(l_2)}](Inv(l_2) \Rightarrow \varphi/_{f}\, l_2)$.

Let $t \geq 0$ s.t. $\langle l_1, v_1\rangle \xrightarrow{\epsilon(t)} \langle l_1, v_1'\rangle$, there are two cases:

- $Inv(l_2)(v_2') = \mathsf{ff}$: therefore $(Inv(l_2) \Rightarrow \varphi/_{f}\, l_2)$ holds for $\langle l_1, v_1', v_2'.u'\rangle$,
- $Inv(l_2)(v_2') = \mathsf{tt}$: therefore there exists $\langle (l_1,l_2), v_1.v_2\rangle \xrightarrow{\epsilon(t)} \langle (l_1,l_2), v_1'.v_2'\rangle$ because $Inv(l_2)(v_2) = Inv(l_2)(v_2') = \mathsf{tt}$ and $Inv(l_2)$ is $Act(l_2)$-strongly connected. Moreover $\langle (l_1,l_2), v_1'.v_2', u'\rangle \models_{\mathcal{D}} \varphi$ entails that $(\varphi/_{f}\, l_2)$ holds for $\langle l_1, v_1', v_2'.u'\rangle$ because we have the induction hypothesis:

  $$\langle (l_1,l_2), v_1'.v_2', u'\rangle \models_{\mathcal{D}} \varphi \;\Leftrightarrow\; \langle l_1, v_1', v_2'.u'\rangle \models_{\mathcal{D}'} \varphi/_{f}\, l_2$$

  Then we have: $\langle l_1, v_1', v_2'.u'\rangle \models_{\mathcal{D}'} (Inv(l_2) \Rightarrow \varphi/_{f}\, l_2)$.

Therefore we have: $\langle l_1, v_1, v_2.u\rangle \models_{\mathcal{D}'} [\delta_{d.Act(l_2)}](Inv(l_2) \Rightarrow \varphi/_{f}\, l_2)$.

$\Leftarrow$. Assume $\langle l_1, v_1, v_2.u\rangle \models_{\mathcal{D}'} [\delta_{d.Act(l_2)}](Inv(l_2) \Rightarrow \varphi/_{f}\, l_2)$.
We want to show $\langle (l_1,l_2), v_1.v_2, u\rangle \models_{\mathcal{D}} [\delta_d]\varphi$.
Let $t \geq 0$ s.t. $\langle (l_1,l_2), v_1.v_2\rangle \xrightarrow{\epsilon(t)} \langle (l_1,l_2), v_1'.v_2'\rangle$. This entails that there exists $\langle l_1, v_1\rangle \xrightarrow{\epsilon(t)} \langle l_1, v_1'\rangle$ and then $\langle l_1, v_1', v_2'.u'\rangle \models_{\mathcal{D}'} (Inv(l_2) \Rightarrow \varphi/_{f}\, l_2)$. Moreover we know $Inv(l_2)(v_2') = Inv(l_2)(v_2) = \mathsf{tt}$ and then we have $\langle l_1, v_1', v_2'.u'\rangle \models_{\mathcal{D}'} (\varphi/_{f}\, l_2)$. Therefore by i.h. we obtain: $\langle (l_1,l_2), v_1'.v_2', u'\rangle \models_{\mathcal{D}} \varphi$.
Then we have: $\langle (l_1,l_2), v_1.v_2, u\rangle \models_{\mathcal{D}} [\delta_d]\varphi$.

$\square$