Robust Undecidability of Timed and Hybrid Systems *

Thomas A. Henzinger[1] Jean-François Raskin[1,2]

[1] EECS Department
University of California at Berkeley
Berkeley, CA 94720-1770, USA

[2] Computer Science Department
Université Libre de Bruxelles
Bld du Triomphe CP212, 1050 Brussels, Belgium

October 1999

Abstract

The algorithmic approach to the analysis of timed and hybrid systems is fundamentally limited by undecidability, of universality in the timed case (where all continuous variables are clocks), and of emptiness in the rectangular case (which includes drifting clocks). Traditional proofs of undecidability encode a single Turing computation by a single timed trajectory. These proofs have nurtured the hope that the introduction of "fuzziness" into timed and hybrid models (in the sense that a system cannot distinguish between trajectories that are sufficiently similar) may lead to decidability. We show conclusively that this is not the case, by sharpening both fundamental undecidability results. Besides the obvious blow our results deal to the algorithmic method, they also prove that the standard model of timed and hybrid systems, while not "robust" in its definition of trajectory acceptance (which is affected by tiny perturbations in the timing of events), is quite robust in its mathematical properties: the undecidability barriers are not affected by reasonable perturbations of the model.

*NSF CAREER award CCR-9501708,DARPA (NASA) grant NAG2-1214, DARPA (Wright-Patterson AFB),grant F33615-C-98-3614, ARO MURI grant DAAH-04-96-1-0341

1 Introduction

The main limitations of the algorithmic method for analyzing timed and hybrid systems find their precise expression in two well-publicized undecidability results. First, the universality problem for timed automata (does a timed automaton accept all timed words?) is undecidable [AD94]. This implies that timing requirements which are expressible as timed automata cannot be model checked. Consequently, more restrictive subclasses of timing requirements have been studied (e.g., Event-Clock Automata [AFH94], Metric Interval Temporal Logic [AFH96]). Second, the emptiness/reachability problem for rectangular automata (does a rectangular automaton accept any timed word, or equivalently, can a rectangular automaton reach a given location?) is undecidable [HKPV95]. While several orthogonal undecidability results are known for hybrid systems, it is the rectangular reachability problem which best highlights the essential limitations of the algorithmic approach to systems with continuous dynamics. This is because the rectangular automaton model is the minimal generalization of the timed automaton model capable of approximating continuous dynamics (using piecewise linear envelopes). It follows that rectangularity as an abstraction is insufficient for checking invariants of hybrid systems, and further loss of information is necessary (e.g., initialization [HKPV95], discretization [HK97]).

Both central undecidability results have been proved by encoding each computation of some Turing-complete machine model as a trajectory of a timed or hybrid system. The encodings are quite fragile: given a deterministic Turing machine M with empty input, one constructs either a timed automaton that rejects the single trajectory which encodes the halting computation of M (rendering universality undecidable), or a rectangular automaton that accepts that single trajectory (rendering emptiness/reachability undecidable). However, if the specified trajectory is perturbed in the slightest way, it no longer properly encodes the desired Turing computation. This has led several researchers to conjecture that undecidability is due to the ability of timed and hybrid automata to differentiate real points in time with infinite precision, and the same researchers have expressed the hope that a more realistic, slightly "fuzzy" model of timed and hybrid systems might not suffer from undecidability.

This has led researchers to conjecture [Fra99] that undecidability is due to the ability of timed and hybrid automata to differentiate real points in time with infinite precision. Consequently, one might hope that a more realistic, slightly "fuzzy" model of timed and hybrid systems might not suffer from undecidability.[1] In a similar vein, in [GHJ97] it is conjectured that unlike timed automata, robust timed automata, which do not accept or reject individual trajectories but bundles ("tubes") of closely related trajectories, can be complemented.

In this paper, we refute these conjectures. In doing so, we show that the sources of undecidability for timed and hybrid systems are structural, robust, and intrinsic to mixed discrete-continuous dynamics, rather than an artifact of a particular syntax or of the ability to measure time with arbitrary precision. We redo both undecidability proofs by encoding each Turing computation not as a single trajectory but as a trajectory tube of positive diameter. This requires considerable care and constitutes the bulk of this paper. As corollaries we obtain the following results:

Robust timed and rectangular automata Robust automata introduce "fuzziness" semantically, by accepting tubes rather than trajectories [GHJ97]. We prove that universality is undecidable for robust timed automata (since emptiness is decidable, it follows that they are not complementable), and that emptiness/reachability is undecidable for robust rectangular automata.

Open rectangular automata Open automata introduce "fuzziness" syntactically, by restricting all guard and differential-inclusion intervals to open sets. We prove that emptiness/reachability is undecidable for open rectangular automata.

A main impact of these results is, of course, negative: they deal a serious blow to our ability for analyzing timed and hybrid systems automatically, much more so than the previously known results, which rely on questionable, "fragile" modeling assumptions (one trajectory may be accepted even if all slightly perturbed

[1] Note that "fuzziness," as meant here, is fundamentally distinct from "discretization," which is known to lead to decidability in many cases. Intuitively, fuzziness preserves the density of the time domain, while discretization does not. Mathematically, discretization is performed with respect to a fixed real $\epsilon > 0$ representing finite precision, while fuzziness quantifies over $\epsilon > 0$ existentially.

trajectories are rejected, and vice versa). There is, however, also a positive interpretation of our results: they show that the "standard" model for timed and hybrid systems, with its fragile definition of trajectory acceptance, does not give rise to a fragile theory but, on the contrary, is very robust with respect to its mathematical properties (such as decidability versus undecidability).

2 Trajectories and Tubes

In this paper, we consider finite trajectories only. A *trajectory* over an alphabet Σ is an element of the language $(\Sigma \times \mathbb{R}^+)^*$, where $\mathbb{R}^+$ stands for the set of positive reals excluding 0. Thus, a trajectory is a finite sequence of pairs from $\Sigma \times \mathbb{R}^+$. We call the first element of each pair an *event*, and the second element the *time-gap* of the event. The time-gap of an event represents the amount of time that has elapsed since the previous event of the trajectory (the first time-gap can be thought of representing the amount of time that has elapsed since the "beginning of time"). For a trajectory τ, we denote its length (i.e., the number of pairs in τ) by $\text{len}(\tau)$, and its projection onto Σ^* (i.e., the sequence of events that results from removing the time-gaps) by $\text{untime}(\tau)$. We assign time-stamps to the events of a trajectory: for the i-th event of τ, the *time-stamp* is defined to be $t_\tau(i) = \sum_{1 \leq j \leq i} \delta_j$, where δ_j is the time gap associated with the j-th event of τ.

Metrics on trajectories

Let the set of all trajectories be denoted **Traj**. Assuming that trajectories cannot be generated and recorded with infinite precision, in order to get an estimate of the amount of error in the data that represents a trajectory, we need a metric on **Traj**. Here we define, as an example, one particular metric d; in [GHJ97], it is shown that all reasonable metrics define the same topology on trajectories. Given two trajectories τ and τ', we define:

- $d(\tau, \tau') = \infty$ if $\text{untime}(\tau) \neq \text{untime}(\tau')$.
- $d(\tau, \tau') = \max\{|t_\tau(i) - t_{\tau'}(i)| : 1 \leq i \leq \text{len}(\tau)\}$.

Thus, only two trajectories with the same length and the same sequence of events have a finite distance, and finite errors may occur only in measuring time. The metric measures the maximal difference in the time-stamps of any two corresponding events: two timed words are close to each other if they have the same events in the same order, and the times at which these events occur are not very different. For instance, for $\tau_1 = (a, 1)(a, 1)(a, 1)$ and $\tau_2 = (a, 0.9)(a, 1.2)(a, 1.2)$, we have $d(\tau_1, \tau_2) = 0.3$.

Given a metric, we use the standard definition of open sets. Formally, for the metric d, a trajectory τ, and a positive real $\epsilon \in \mathbb{R}^+$, define the *d-tube around τ of diameter ϵ* to be the set $T(\tau, \epsilon) = \{\tau' : d(\tau, \tau') < \epsilon\}$ of all trajectories at a d-distance less than ϵ from τ. A d-open set O, called a *d-tube*, is any subset of **Traj** such that for all trajectories $\tau \in O$, there is a positive real $\epsilon \in \mathbb{R}^+$ with $T(\tau, \epsilon) \subseteq O$. Thus, if a d-tube contains a trajectory τ, then it also contains all trajectories in some neighborhood of τ. Let the set of all d-tubes be denoted **Tube**.

From trajectory languages to tube languages

A *trajectory language* is any subset of **Traj**; a *tube language* [GHJ97] is any subset of **Tube**. Every trajectory language L induces a tube language $[L]$, which represents a "fuzzy" rendering of L. In $[L]$ we wish to include a tube iff sufficiently many of its trajectories are contained in L. We define "sufficiently many" as any dense subset, in the topological sense.

For this purpose we review some simple definitions from topology. A set S of trajectories is closed if its complement $S^c = \textbf{Traj} - S$ is open. The closure $\overline{S}$ of a set S of trajectories is the least closed set containing S, and the interior S^{int} is the greatest open set contained in S. The set S' of trajectories is dense in S iff $S \subseteq \overline{S'}$.

Formally, given a trajectory language L, the corresponding tube language is defined as

$$[L] = \{O \in \textbf{Tube} : O \subseteq \overline{L}\}.$$

Thus, a tube O is in $[L]$ if for each trajectory $\tau \in O$ there is a sequence of trajectories with limit τ such that all elements of this sequence are in L. Equivalently, L must be dense in O; that is, for every trajectory $\tau \in O$ and for every positive real $\epsilon \in \mathbb{R}^+$, there is a trajectory $\tau' \in L$ such that $d(\tau, \tau') < \epsilon$. Since the tubes in $[L]$ are closed under subsets and union, the tube language $[L]$ can be identified with the maximal tube in $[L]$, which is the interior $\overline{L}^{int}$ of the closure of L.

We will define the semantics of a robust rectangular automaton with trajectory set L to be the tube set $[L]$. This has the effect that a robust rectangular automaton cannot generate (or accept) a particular trajectory when it refuses to generate (rejects) sufficiently many surrounding trajectories. Neither can the automaton refuse to generate a particular trajectory when it may generate sufficiently many surrounding trajectories. Our definition of "sufficiently many" as "dense subset" does not seem all that strong, because every tube O, while uncountable, has dense subsets that are countable (such as the set of trajectories in O all of whose time-gaps are rationals). However, when we define rectangular automata below, we will see that the syntax of rectangular automata will not allow us to specify very strange trajectory languages L. In particular, we will not be able to specify a trajectory language L such that both L and L^c are dense in some tube O. Thus, for rectangular automata, a tube will be accepted iff all but finitely many of its trajectories are accepted, and it will be rejected iff all but finitely many of its trajectories are rejected.

3 Robust Timed and Rectangular Automata

A *interval* has the form (a, b), $[a, b]$, $(a, b]$, or $[a, b)$, where $a \in \mathbb{Q} \cup \{-\infty\}$, $b \in \mathbb{Q} \cup \{\infty\}$, and $a \leq b$ if I is of the form $[a, b]$, and $a < b$ otherwise. We say that the interval I is *open* if it is of the form (a, b) and *closed* if it is of the form $[a, b]$. We write Rect for the set of intervals. A *rectangular constraint for the variable x* is an constraint of the form $x \in I$ where I is an interval.

A *rectangular automaton* [HKPV95] is a tuple $A = \langle \Sigma, Q, Q_0, Q_f, C, E, \mathsf{Lab}, \mathsf{Init}, \mathsf{Pre}, \mathsf{Reset}, \mathsf{Post}, \mathsf{Flow} \rangle$ (i) Σ is a finite alphabet of events; (ii) Q is a finite set of locations; (iii) $Q_0 \subseteq Q$ is a set of start locations; (iv) $Q_f \subseteq Q$ is a set of accepting locations; (v) C is a finite set of real-valued variables; (vi) $E \subseteq Q \times Q$ is a finite set of transitions; (vii) $\mathsf{Lab} : E \to \Sigma$ is a function that labels each edge e with a letter of the alphabet Σ; (viii) $\mathsf{Init} : Q_0 \to C \to \mathsf{Rect}$ is a function that indicates for each initial location $q_0 \in Q_0$ and each variable $x \in C$ the possible initial values of this variable when the control of the automaton starts in location q_0; for convenience, in figures we write $x \in I$ inside q_0 instead of $\mathsf{Init}(q_0, x) = I$; (ix) $\mathsf{Pre} : E \to C \to \mathsf{Rect}$ is a function that associates for each edge e and each continuous variable x a rectangular constraint that must hold to cross the edge; $\mathsf{Pre}(e, x) = I$ means that the value of the continuous variable must lie in the interval I before crossing the edge e; (x) $\mathsf{Post} : E \to C \to \mathsf{Rect}$ is a function that associates with each edge e and each continuous variable x a rectangular constraint that must hold after crossing the edge; $\mathsf{Post}(e, x) = I$ means that the value of the continuous variable x must lie in the interval I after crossing the edge e; (xi) $\mathsf{Reset} : E \to 2^C$ is the function that associates with each edge e the subset of variables that are reset when crossing e; if a variable x belongs to the set $\mathsf{Reset}(e)$ then the value, after crossing the edge e, of this variable is taken nondeterministically from the interval $\mathsf{Post}(e, x)$; (xii) $\mathsf{Flow} : Q \to C \to \mathsf{Rect}$ is a function that associates with each location q and variable x a rectangular flow constraint; $\mathsf{Flow}(q, x) = I$ implies that the flow (the first derivative) of the variable x when the control is in location q lies within the rectangle I.

We now define timed automata as a syntactical subset of rectangular automata. A rectangular automaton A is a *timed automaton* [AD94] if the function Flow of A is such that for all locations $q \in Q$, and for all variables $x \in C$: $\mathsf{Flow}(q, x) = [1, 1]$, that is, every continuous variable is a clock. We say that a timed automaton A is *open* if all intervals used in constraints with the functions Init, Pre, and Post are open. Similarly, a rectangular automaton A is *open* if all intervals used in constraints with the functions Init, Pre, Post, and Flow are open.

A rectangular automaton A defines a labelled transition system with an infinite state space S, the infinite set of labels $\mathbb{R}^+ \cup \Sigma$, and the binary relation R. Each transition with label σ correspond to an edge step whose observation is $\sigma \in \Sigma$. Each transition with label $\delta \in \mathbb{R}^+$ corresponds to a time step of duration δ. The states and transitions of A are defined as follows:

State. A *state* $(q, \mathbf{x})$ of A consists of a discrete part $q \in Q$ and a constinuous part $\mathbf{x} \in \mathbb{R}^n$. The *state space* $S \subset Q \times \mathbb{R}^n$ is the set of all states of A. The state $(q, \mathbf{x})$ is an *initial state* of A if $q \in Q_0$ and $\mathbf{x} \in \mathsf{Init}(q)$.

Jump transitions. For each edge $e = (q_1, q_2)$ of A, we define the binary relation $\rightarrow^e \subset S^2$ by $(q_1, \mathbf{x}) \rightarrow^e (q_2, \mathbf{y})$ iff $\mathbf{x} \in \mathsf{Pre}(e)$, $\mathbf{y} \in \mathsf{Post}(e)$ and for every coordinate $i \in \{1, \ldots, n\}$ with $i \notin \mathsf{Reset}(e)$, we have $\mathbf{x}_i = \mathbf{y}_i$. Hence $\mathbf{x}$ and $\mathbf{y}$ differ only at coordinates that are in the reset set of e. For each observation $\sigma \in \Sigma$, we define the *edge-step relation* $\rightarrow^\sigma \subset S^2$ by $s_1 \rightarrow^\sigma s_2$ iff $s_1 \rightarrow^e s_2$ for some edge $e \in E$ with $\mathsf{Lab}(e) = \sigma$.

Flow transitions. For each strictly positive real number $\delta \in \mathbb{R}^+$, we define the binary *time-step relation* $\rightarrow^\delta \subset S^2$ by $(q_1, \mathbf{x}) \rightarrow^\delta (q_2, \mathbf{y})$ iff $q_1 = q_2$ and $\frac{\mathbf{y}-\mathbf{x}}{\delta} \in \mathsf{Flow}(q_1)$.

So the transition relation $R \subseteq S \times S$ is defined as follows: $R = \{\rightarrow^e \mid e \in E\} \cup \{\rightarrow^\delta \mid \delta \in \mathbb{R}^+\}$.

Trajectory acceptance and reachable locations

We now define the trajectory language and the reachable locations of a rectangular automaton A.

Trajectory language. A *run* of the automaton A is a path $(q_0, \mathbf{x}_0) \rightarrow^{\delta_0} (q_0, \mathbf{y}_0) \rightarrow^{\sigma_0} (q_1, \mathbf{x}_1) \rightarrow^{\delta_1} \ldots \rightarrow^{\sigma_n} (q_{n+1}, \mathbf{x}_{n+1})$ in the transition system of A that alternates between time steps and edge steps. The run is said *initial* if $q_0 \in Q_0$ and $\mathbf{x}_0 \in \mathsf{Init}(q_0)$, and *accepting* if $q_n \in Q_f$. We say that the trajectory $\tau = (\sigma_0, \delta_0)(\sigma_1, \delta_1) \ldots (\sigma_n, \delta_n)$ is accepted by the rectangular automaton A if A has an initial and accepting run $(q_0, \mathbf{x}_0) \rightarrow^{\delta_0} (q_0, \mathbf{y}) \rightarrow^{\sigma_0} (q_1, \mathbf{x}_1) \rightarrow^{\delta_1} \ldots \rightarrow^{\sigma_n} (q_{n+1}, \mathbf{x}_{n+1})$. We say that the trajectory τ *leads to* location q_{n+1}. We note $L(A)$ the set of trajectories accepted by A.

Reachable locations. We say that a location q of A is *reachable* if there exists an trajectory τ accepted by A that leads to q. We say that a set of locations $\{q_1, \ldots, q_n\}$ is reachable if there exists $q \in \{q_1, \ldots, q_n\}$ such that q is reachable.

The *trajectory emptiness problem* for a rectangular automaton A is to decide wether or not $L(A)$ is empty. The *trajectory universality problem* for a rectangular automaton A is to decide wether or not $L(A)$ contains all trajectories over the alphabet Σ. The *reachability problem* for a rectangular automaton A is to decide if a given set of locations of A is reachable. Note that the language emptiness problem for a class of rectangular automaton is decidable iff the location reachability problem is decidable. The previously known results about these problems are summarized in the following table:

Class of Automata	Emptiness/Reachability	Universality
Timed Automata [AD94]	Decidable	Undecidable
Rectangular Automata [HKPV95]	Undecidable	Undecidable

Figure 1: Known decidability and undecidability results about timed and rectangular automata.

Tube acceptance and robustly reachable locations

The rectangular automaton A accepts the set $[L(A)]$ of tubes [GHJ97]. The following examples illustrate tube acceptance. First, consider the timed automaton A_1 of Figure 2(a). This automaton accepts all trajectories over the unary alphabet $\{a\}$ which contain two consecutive a events with a time-gap in the

(a) (b)

Figure 2: The timed automata A_1 and A_2.

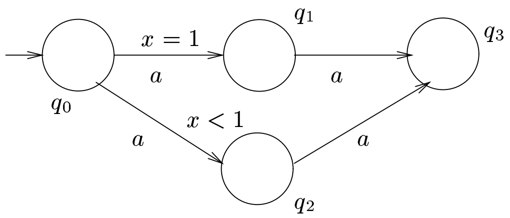

Figure 3: Robustly and non-robustly reachable locations.

open interval $(1, 2)$. This property is invariant under sufficiently small perturbations of the time-stamps. Hence the automaton A_1 accepts precisely those tubes that consist of trajectories in $L(A_1)$, and the maximal accepted tube is $L(A_1)$ itself. In the timed automaton A_2 of Figure 2(b), the open interval $(1, 2)$ is replaced by the closed interval $[1, 2]$. This changes the set of accepted trajectories but not the set of accepted tubes: $L(A_1) \subset L(A_2)$ but $[L(A_1)] = [L(A_2)]$. Notice that the "boundary trajectories" accepted by A_2, with two consecutive a's at a time-gap of 1 or 2 but no consecutive a's at a time-gap strictly between 1 and 2, are not accepted robustly, because there are arbitrarily small perturbations that are not acceptable.

Let us now define the notion of robust reachability. We say that a location q of a rectangular automaton A is *robustly reachable* if there exists a tube O accepted by A such that each trajectory in O leads to q. The automaton of figure 3 illustrates this notion: the locations q_0, q_2, and q_3 are robustly reachable, while the location q_1 is not robustly reachable.

The *robust emptiness problem* for a rectangular automaton A is to decide if wether or not $[L(A)]$ is empty. The *robust universality problem* for a rectangular automaton A is to decide wether of not $[L(A)]$ contains all tubes over Σ. The *robust reachability problem* for a rectangular automaton A is to decide, given a set of locations of A, if some location in the set is robustly reachable. In the next sections of this paper, we sharpen the known undecidability results about timed and hybrid systems. We show that the introduction of fuzziness into timed and hybrid models via the notion of tubes (this fuzziness can be intuitively seen as the semantic removal of equality) does not change the undecidability results. Our results are summarized in the following table; only the positive result was previously known [GHJ97]:

Class of Automata	Robust Emptiness/Robust Reachability	Robust Universality
Timed Automata	Decidable	Undecidable
Rectangular Automata	Undecidable	Undecidable

Figure 4: Decidability results about robust timed and rectangular automata.

4 Some Properties of Robust Timed Automata

We recall in this section some results presented in [GHJ97]. We will need those notions to establish our results. The first proposition tells us that when we consider tube acceptance, we can restrict our attention either to closed or open timed automata.

Proposition 1 *For every timed automaton A, we can construct a timed automaton $\overline{A}$, called the closure of A, that uses only closed rectangles in* Pre, Post, Init *and such that $L(\overline{A}) = \overline{L(A)}$. Furthermore, we can construct an open timed automaton A^{int}, called the interior of A, such that: $[L(A)] = [L(A^{int})] = [L(\overline{A})]$.*

The following proposition shows that for open timed automata, tube emptiness coincides with trajectory emptiness.

Proposition 2 *For every open timed automaton A and every trajectory τ, if τ is accepted by A along some path, then there is a positive real $\epsilon \in \mathbb{R}^+$ such that all trajectories in the tube $T(\tau, \epsilon)$ are accepted by A along the same path.*

By Proposition 1 we can reduce the problem of checking if a timed automaton A accepts any tube to the problem of checking if the interior automaton A^{int} accepts any tube. Moreover, by Proposition 2, the open automaton A^{int} accepts any tube iff it accepts any trajectory. The latter problem can be solved using the region construction of [AD94]. In fact, for checking the emptiness of open timed automata such as A^{int}, only open regions need be considered.

Theorem 1 *The problem of deciding whether a timed automaton accepts any tube is complete for PSPACE.*

The previous result shows that timed automata yield a decidable theory of tubes. The tube languages defined by timed automata are closed under union intersection [GHJ97]. The closure under negation was left open in [GHJ97]. We show here that unfortunately, the tube languages definable by timed automata are not closed under negation. We first recall some notions about the complement of a tube language. The timed automaton B is a *trajectory complement* of the timed automaton A iff B accepts precisely the trajectories that are not accepted by A; that is, $L(B) = L(A)^c$. Before defining the tube complements of a timed automaton, we observe an important property of the trajectory languages that can be defined by timed automata.

Proposition 3 *For every timed automaton A, there is no tube O such that both $L(A)$ and $L(A)^c$ are both dense in O.*

It follows that a tube cannot be accepted by both a timed automaton A and a trajectory complement of A. This observation will allow us to relate the tube complements of a timed automaton to its trajectory complements.

For defining the tube complements of a timed automaton A, it is not useful to consider the boolean complement $\mathbf{Tube} - [L(A)]$ of the tube language $[L(A)]$. For $[L(A)]$ is closed under subsets and union. Therefore, unless $[L(A)] = \emptyset$ or $[L(A)] = \mathbf{Tube}$, the boolean complement $\mathbf{Tube} - [L(A)]$ cannot be induced by any trajectory language and, hence, cannot be accepted by any timed automaton. Thus, for every tube language $\mathcal{L} \subseteq \mathbf{Tube}$, we define the *tube complement* of $\mathcal{L}$ to be the set

$$\mathcal{L}^c = \{O \in \mathbf{Tube} : O \cap \bigcup \mathcal{L} = \emptyset\}$$

of tubes that are disjoint from the tubes in $\mathcal{L}$. The following proposition shows that for every timed automaton A, the tube complement $[L(A)]^c$ is induced by the trajectory complement $L(A)^c$; that is, $[L(A)^c] = [L(A)]^c$.

Proposition 4 *If L is a trajectory language and there is no tube O such that both L and L^c are dense in O, then $[L]^c = [L^c]$.*

For two timed automata A and B, we say that B is a *tube complement* of A iff B accepts precisely the tubes that do not intersect any tube accepted by A; that is, $[L(B)] = [L(A)]^c$. From Propositions 3 and 4, it follows that every trajectory complement of a timed automaton is also a tube complement (the converse is generally not true). Since $[L(A)]^c = [L(A^{int})]^c = [L(A^{int})^c]$, in order to construct tube complements, it would suffice to construct trajectory complements of open timed automata.[2] This, however, is not possible as we show in the next section.

5 The Universality Problem for Robust Timed Automata

In this section, we show that the halting problem for two-counter machines can be reduced to the robust universality problem for robust timed automata.

A *two-counter machine* M is a triple $\langle\{b_1, \ldots, b_n\}, C, D\rangle$ where $\{b_1, \ldots, b_n\}$ are n instructions, C and D are two counters ranging over the natural numbers. Each instruction b_i, $0 \leq i \leq n$ is of three possible forms: (i) Conditionnal jump instruction: those instructions test a counter being zero and then jump conditionally

[2]Similarly, since $[L(A)]^c = [L(\overline{A})]^c = [L(\overline{A})^c]$, it would suffice to construct trajectory complements of closed timed automata. This, however, is known to be impossible [AD94].

to the next instruction; (ii) Increment/Decrement instruction: those instructions increment or decrement the value of one of the two counters and then jump nondeterministically to one of the two possible next instructions; (iii) Stop instruction: this instruction puts an end to the machine execution.

A *configuration of a two-counter machine* M is a triple $\gamma = \langle i, c, d\rangle$ where i is the program counter indicating the current instruction, c and d are the values of the counters C and D. An *computation of a two-counter machine* M is a sequence $\overline{\gamma} = \gamma_0\gamma_1 \dots \gamma_n \dots$ of configurations such that $\gamma_0 = \langle 0, 0, 0\rangle$ that is the first instruction is b_0 and the initial value of the two counters C and D is zero, for every position i, $0 \leq i < |\overline{\gamma}|$, γ_{i+1} is a M-successor configuration of γ_i. The *halting problem* for a two-counter machine M is to decide whether or not the execution of M has at least one computation that ends in a *stop instruction*. The problem of deciding if a 2-counter machine M has a halting computation is undecidable.

An Undecidable Tube Language $\mathsf{L}^{\mathsf{Undec}}_{\mathsf{Tube}}$

We define in this section a tube language that we call $\mathsf{L}^{\mathsf{Undec}}_{\mathsf{Tube}}(M)$ that is parametrized by a two-counter machine M and is non-empty iff M has a halting computation.

We first review how the undecidability of the universality problem for timed automata was established by Alur and Dill [AD94] and explain why their proof does not translate directly to our robust timed automata. The language $\mathsf{L}_{\mathsf{Undec}}(M)$ is defined as follows: $\tau = (\sigma, \delta) \in \mathsf{L}_{\mathsf{Undec}}(M)$ iff (i) $\sigma = b_{i_0}c^{c_0}d^{d_0}b_{i_1}c^{c_1}d^{d_1}\dots b_{i_m}c^{c_m}d^{d_m}$ such that $\langle i_0, c_0, d_0\rangle, \langle i_1, c_1, d_1\rangle \dots \langle i_m, c_m, d_m\rangle$ is a halting computation of M; (ii) for all positions $j \geq 0$, the time-stamp of b_{i_j} is j; (iii) for all $j \geq 1$, (a) if $c_{j+1} = c_j$ then for every c at time t in the interval $(j, j+1)$ there is a c at time $t+1$; (b) $c_{j+1} = c_j + 1$ then for every c at time t in the interval $(j+1, j+2)$, except the last one, there is a c at time $t-1$; (c)if $c_{j+1} = c_j - 1$ then for every c at time t in the interval $(j, j+1)$, except the last one, there is a c at time $t+1$; (iv) the same requirements hold for d's.

So the i-th configuration is encoded on the interval $[i, i+1)$ and to enforce a requirement such as the number of c events in two successive configurations is the same, it is required that every c in the first interval is matching a c at distance 1 and vice versa. Note that the use of the punctuality constraint has the following consequence.

Proposition 5 *There is no tube $O \in$ **Tube** such that O is dense in $\mathsf{L}_{\mathsf{Undec}}$, i.e. $[\mathsf{L}_{\mathsf{Undec}}] = \emptyset$.*

This has nurtured some hope that, by removing the possibility to specify punctuallity constraints, robust timed automata could have a decidable robust universality problem. Unfortunately this is not the case. We next show that we can define a set of trajectories that is a tube language and encode halting computations of a two-counter machine M. Furthermore the tube complement of this tube language can be defined by a robust (open) timed automata. The undecidability of the tube universality problem and the non closure under complement of robust timed automata will follow.

To facilitate the definition of $\mathsf{L}^{\mathsf{Undec}}_{\mathsf{Tube}}$, the undecidable tube language, we first introduce some new notions. We call an *open (closed) slot* an open (closed) interval of the real numbers. We define the open (closed) slot between t_1 and t_2 as the set $\{t \mid t_1 < t < t_2\}$ ($\{t \mid t_1 \leq t \leq t_2\}$). Given two real numbers t_1 and t_2, with $t_1 < t_2$, we say that (t_3, t_4), respectively $[t_3, t_4]$, is the open, respectively, the closed, slot *generated* by t_1 and t_2 if we have $t_1 + 1 = t_3$ and $t_2 + 1 = t_4$.

The main idea of $\mathsf{L}^{\mathsf{Undec}}_{\mathsf{Tube}}$ is that we will encode the configuration i within open intervals $(i, i+1)$ and the next configuration $i+1$ will be encoded in the open slot generated by the time of occurence of the begin marker and the end marker of configuration i. For the encoding of the elements of a configuration and their relation with the next configuration we also use open slots. For instance, we use the sequence $\mathsf{B}^{\mathsf{Inst}} \cdot b_{j_i} \cdot \mathsf{E}^{\mathsf{Inst}}$ to encode that b_{j_i} is the instruction executed in the i-th configuration, $\mathsf{B}^{\mathsf{Inst}}$ and $\mathsf{E}^{\mathsf{Inst}}$ are used as delimiters of the instruction encoding and to generate the slot for the next instruction. Let us assume that t_1 and t_2 are the time-stamp of $\mathsf{B}^{\mathsf{Inst}}$ and $\mathsf{E}^{\mathsf{Inst}}$ respectively. Then the encoding of the next instruction has to take place in the open slot $(t_1 + 1, t_2 + 1)$ generated by the slot for the current instruction. As we use a dense time domain, this constraint can always be satisfied. We will proceed in the same way for the encoding of the values of the two counters. The value of the counters C and D are encoded as follows: if the value of the counter C is u in configuration i then the sequence $\mathsf{b}^{\mathsf{c}} \cdot \mathsf{e}^{\mathsf{c}}$ is repeated u times in the encoding of the configuration i. If the counter C is unchanged from configuration i to configuration $i+1$, we verify that the

$\mathsf{b^c} \cdot \mathsf{e^c}$ sequences in configuration $i+1$ appear exactly in the open slots defined by the $\mathsf{b^c} \cdot \mathsf{e^c}$ sequences in configuration i.

Having the intuition underlying the language $\mathsf{L}^{\mathsf{Undec}}_{\mathsf{Tube}}(\mathrm{M})$, we now define it more precisely and establish that the set of trajectories that forms $\mathsf{L}^{\mathsf{Undec}}_{\mathsf{Tube}}(\mathrm{M})$ define indeed a non empty set of tubes iff the machine M has a halting computation.

Alphabet of the language The set of events that we will use in the encoding is the following: (i) $\mathsf{B^{Conf}}$ and $\mathsf{E^{Conf}}$ are the delimiters for the begin and end of the encoding of a configuration; (ii) $\mathsf{B^{Inst}}$ and $\mathsf{E^{Inst}}$ are the delimiters for the begin and end of the encoding of the instruction executed in a configuration; (iii) $b_1, b_2, \ldots, b_n$ are used to represent the n instructions; (iv) $\mathsf{B^C}$ and $\mathsf{E^C}$ are the delimiters for the encoding of the value of the counter C in a configuration; (v) $\mathsf{B^D}$ and $\mathsf{E^D}$, idem for the counter D; (vi) $\mathsf{b^c}$ and $\mathsf{e^c}$ are used to encode the value of the counter C; (vii) $\mathsf{b^d}$ and $\mathsf{e^d}$, idem for D.

Qualitative requirements The trajectories of $\mathsf{L}^{\mathsf{Undec}}_{\mathsf{Tube}}$ agree with the following regular expression: $(\mathsf{B^{Conf}} \cdot \mathsf{B^{Inst}} \cdot (b_1 \mid b_2 \mid \ldots \mid b_n) \cdot \mathsf{E^{Inst}} \cdot \mathsf{B^C} \cdot (\mathsf{b^c} \cdot c \cdot \mathsf{e^c})^* \cdot \mathsf{E^C} \cdot \mathsf{B^D} \cdot (\mathsf{b^d} \cdot d \cdot \mathsf{e^d})^* \cdot \mathsf{E^D} \cdot \mathsf{E^{Conf}})^*$. Furthermore, if the configuration i contains the sequence $\mathsf{B^{Inst}} \cdot b_{j_i} \cdot \mathsf{E^{Inst}}$ then the configuration $i+1$ contains the sequence $\mathsf{B^{Inst}} \cdot b_{j_{i+1}} \cdot \mathsf{E^{Inst}}$ where $b_{j_{i+1}}$ is a valid next instruction of b_{j_i}. We refer to those requirements as QUAL. Note that QUAL can be imposed by an untimed finite automaton, and thus, also its negation.

Encoding of configurations (i) The first configuration is encoded on the open interval $(0, 1)$. That is, if the event $\mathsf{B^{Conf}}$ occurs at time t_1 and the event $\mathsf{E^{Conf}}$ occurs at time t_2 then we have that $0 < t_1 < t_2 < 1$. This requirement is noted RT^a_1; (ii) the configuration $i+1$ is always encoded in the open slot defined by the configuration i. That is, if the event $\mathsf{B^{Conf}}$ of configuration i occurs at time t_1 and the event $\mathsf{E^{Conf}}$ occurs at time t_2 then the encoding of the configuration $i+1$ takes place in the open slot (t_1+1, t_2+1). This requirement is noted RT^b_1.

Encoding of instructions The encoding of the instruction executed by the two-counter machine M during the configuration $i+1$ takes place in the slot defined by the encoding of the instruction executed in the configuration i. That is, if $\mathsf{B^{Inst}}$ and $\mathsf{E^{Inst}}$ append at time t_1 and t_2 in encoding of configuration i then $\mathsf{B^{Inst}}$ and $\mathsf{E^{Inst}}$ appear at time t_3 and t_4 in the encoding of configuration $i+1$ with the following (open) real-time constraint: $t_1 + 1 < t_3 < t_4 < t_2 + 1$. This requirement is noted RT_2.

Encoding of the two counters We only explain in details the case when the counter C is incremented from configuration i to configuration $i+1$. The other operations are left to the reader. The encoding of the values of the counter C for two adjacent configurations is represented graphically in figure 5, there the counter C is incremented from u to $u+1$. (i) If in configuration i the event $\mathsf{B^C}$ and $\mathsf{E^C}$ occur at time t_1 and t_2 respectively, then the events $\mathsf{B^C}$ and $\mathsf{E^C}$ appear for in configuration $i+1$ within the open slot (t_1+1, t_2+1). This requirement is noted RT^a_3; (ii) for each $\mathsf{b^c} \cdot \mathsf{e^c}$ sequence, such that $\mathsf{b^c}$ occurs at time t_1 and $\mathsf{e^c}$ occurs at time t_2, in the encoding of configuration i, there is exactly one sequence $\mathsf{b^c} \cdot \mathsf{e^c}$ sequence in the encoding of configuration $i+1$ that takes place in the open slot (t_1+1, t_2+1). This requirement is noted RT^b_3; (iii) conversely, each $\mathsf{b^c} \cdot \mathsf{e^c}$ that appears in the encoding of the configuration $i+1$, with the exception of the last, must lie in the open slot defined by $\mathsf{b^c} \cdot \mathsf{e^c}$ sequence of configuration i. This requirement is noted RT^c_3; (iv) finally, the last $\mathsf{b^c} \cdot \mathsf{e^c}$ sequence in the encoding of configuration $i+1$ appears in the slot generated by the two events $\mathsf{B^C}$ and $\mathsf{E^C}$ if $C = 0$ in configuration i, and appears in the slot generated by the last $\mathsf{e^c}$ event and $\mathsf{E^C}$ event of configuration i if $C > 0$ in that configuration. This requirement is noted RT^d_3.

The following proposition is a direct consequence of the use of strict inequalities in the definition of the language $\mathsf{L}^{\mathsf{Undec}}_{\mathsf{Tube}}(M)$.

Proposition 6 *For every trajectory τ_1 that belongs to $\mathsf{L}^{\mathsf{Undec}}_{\mathsf{Tube}}(M)$, there exists an $\epsilon > 0$ such that for every trajectory τ_2, if $d(\tau_1, \tau_2) < \epsilon$ then $\tau_2 \in \mathsf{L}^{\mathsf{Undec}}_{\mathsf{Tube}}(M)$.*

Corollary 1 *For every two-counter machine M with a halting computation, $[\mathsf{L}^{\mathsf{Undec}}_{\mathsf{Tube}}(M)]$ is a nonempty tube language.*

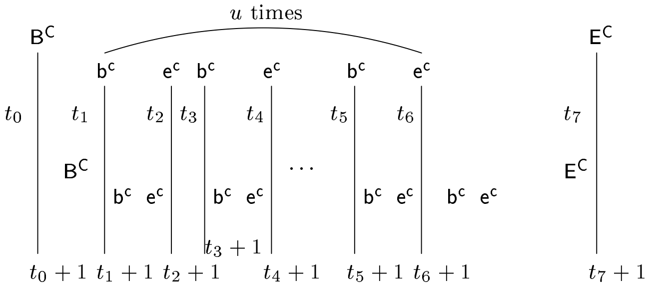

Figure 5: Incrementation of the value of C in two consecutive configurations.

Another direct consequence of the proposition 6 is the following corollary.

Corollary 2 *There is no tube O that is dense both in $\mathsf{L}^{\mathsf{Undec}}_{\mathsf{Tube}}(M)$ and in $(\mathsf{L}^{\mathsf{Undec}}_{\mathsf{Tube}}(M))^c$.*

Note also that by proposition 6 and by the corollary 2, we know that the tube semantics of a timed automaton that accepts the complement of the trajectories of $\mathsf{L}^{\mathsf{Undec}}_{\mathsf{Tube}}(M)$, is exactly the complement of the tube language $[\mathsf{L}^{\mathsf{Undec}}_{\mathsf{Tube}}(M)]$. The following lemma shows that it is possible to construct such a timed automaton. Its proof is given in the appendix.

Lemma 1 *There exists a timed automaton A_M that accepts exactly the trajectories that are not in $\mathsf{L}^{\mathsf{Undec}}_{\mathsf{Tube}}(M)$.*

Combining lemma 1 and proposition 4, we obtain the following theorem.

Theorem 2 *For every two-counter machine M, there exists a timed automaton A_M that accepts every tube iff the two-counter machine M has no halting computation.*

As a direct consequence, we have the following.

Theorem 3 *The robust universality problem for timed automata is undecidable.*

As the robust emptiness problem for timed automata is decidable, we have the following.

Corollary 3 *There are tube languages definable by robust timed automata whose complements are not definable by any robust timed automata.*

The Trajectory Complementation Problem for Open Timed Automata

We now give an answer to the question whether the trajectory language of every open timed automaton is complementable. We answer this question negatively in the following theorem which is derived from the results of previous section. The proof of this theorem is given in the appendix.

Theorem 4 *There are trajectory languages definable by open timed automata whose trajectory complements are not definable by any timed automata (open or not).*

6 The Robust Reachability Problem for Rectangular Automata

In this section we investigate undecidable reachability problems and show that they remain undecidable even when we remove equality from the specification formalism. In [HKPV95], it is shown that the formalism of rectangular automata lies at the boundary between decidable hybrid formalisms and undecidable ones. We show here that this boundary stays valid if we do not use equality.

Another Encoding of Two-Counter Machines Computations

We first expose a way to encode computations of two-counter machine using trajectories that are definable by rectangular automata.

To each halting computation $\langle i_0, c_0, d_0\rangle, \langle i_1, c_1, d_1\rangle, \ldots, \langle i_n, c_n, d_n\rangle$, we associate the following trajectory $(b_{i_j}, t_{(j,0)}), (\mathsf{B}^\mathsf{C}, t_{(j,1)}), (\mathsf{c}, t_{(j,2)})(\mathsf{B}^\mathsf{D}, t_{(j,3)})(\mathsf{d}, t_{(j,4)})$ with $0 \leq j \leq n$, and with the following additional timing constraints: (i) if the value of the counter C is u in configuration i then the time difference between the events B^C and c in the encoding of configuration i is equal to $\frac{1}{2^u}$, that is $t_{(i,2)} - t_{(i,1)} = \frac{1}{2^u}$. So incrementing the value of C is encoded as dividing by two the distance between the events B^C and c in two adjacent configurations. Testing that the value of the counter C is equal to 0 amounts in verifying that the delay between the event B^C and the following c is equal to 1. (ii) idem for the counter D.

The language defined as above associated to the two-counter machine M is noted $\mathsf{L}^{\mathsf{Undec}}_{\mathsf{Rect}}(M)$ and it is definable by a rectangular automata [HKPV95].

A Relaxation of this Encoding

Again, let us show that we do not need equality to encode the two-counter machine computations. We define the following relaxation of the previous encoding. To each halting computation $\langle i_0, c_0, d_0\rangle, \langle i_1, c_1, d_1\rangle, \ldots, \langle i_n, c_n, d_n\rangle$, we associate the following trajectories:

$$(b_{i_j}, t_{(j,0)}), (\mathsf{B}^\mathsf{C}, t_{(j,1)}), (\mathsf{b}^\mathsf{c}, t_{(j,2)}), (\mathsf{e}^\mathsf{c}, t_{(j,3)})(\mathsf{B}^\mathsf{D}, t_{(j,4)})(\mathsf{b}^\mathsf{d}, t_{(j,5)})(\mathsf{e}^\mathsf{d}, t_{(j,6)}) \text{ with } 0 \leq j \leq n.$$

with the following additional timing constraints. We just give the constraints for the encoding of the value of counter C, the same requirements stand for the counter D.

Encoding of the initial value of C. Initially the value of the counter C is zero. To encode that $C = 0$, we require that if the events B^C,b^c, and e^c are issued at time t_1, t_2, and t_3 then the following constraint is satisfied: $t_1 + \frac{1}{2} < t_2 < t_3 < t_1 + 1$.

Encoding of the value of C in the configuration $i+1$. Let d_1 denote the distance that separates the events B^C and b^c, and let d_2 denote the distance that separates the events B^C and e^c in the encoding of the value of C in the configuration i. And in the same way, let d_3 and d_4 be those two distances in the encoding of the value of C in configuration $i+1$. Then we have the following requirements: (i) if C is incremented between i and $i+1$: $\frac{d_1}{2} < d_3 < d_4 < \frac{d_2}{2}$; (ii) if C is decremented between i and $i+1$: $2 \times d_1 < d_3 < d_4 < 2 \times d_2$; (iii) if C is unchanged between i and $i+1$: $d_1 < d_3 < d_4 < d_2$.

We note this trajectory language $\mathsf{L}^{\mathsf{Undec}}_{\mathsf{OpenRect}}(M)$. We can establish the following invariant from the real-time constraints expressed above.

Proposition 7 *If the value of the counter C is u in configuration i then the following invariant is true: $\frac{1}{2^{u+1}} < d_1 < d_2 < \frac{1}{2^u}$, where d_1 is the time gap between event B^C and event b^c, and d_2 is the time gap between event B^C and event e^c.*

This last proposition gives us a way to check that the value of counter C is 0. To verify that C is equal to 0, we only have to check that the sequence $\mathsf{b}^\mathsf{c} \cdot \mathsf{e}^\mathsf{c}$ appends in the open slot $(t + \frac{1}{2}, t+1)$ where t is the time at which B^C is issued. Now let us show that given a two-counter machine M, we are able to construct an *open* rectangular automaton A_M that accepts exaclty the language $\mathsf{L}^{\mathsf{Undec}}_{\mathsf{OpenRect}}(M)$. This is established in the following proposition and its proof is given in the appendix.

Lemma 2 *The trajectory language $\mathsf{L}^{\mathsf{Undec}}_{\mathsf{OpenRect}}(M)$ is definable by an open rectangular automaton A_M.*

As a direct consequence of the last lemma, we have the following.

Theorem 5 *The trajectory emptiness and reachability problems for open rectangular automata are undecidable.*

The following proposition is a generalization to open rectangular automata of proposition 2.

Proposition 8 *For every open rectangular automaton A and every trajectory τ, if τ is accepted by A along some path, then there is a positive real $\epsilon \in \mathbb{R}^+$ such that all trajectories in the tube $T(\tau, \epsilon)$ are accepted by A along the same path.*

This proposition implies that tube and trajectory emptiness coincide for open rectangular automata, so we have the following theorem.

Theorem 6 *The robust emptiness and robust reachability problems for rectangular automata are undecidable.*

7 Conclusion

We refuted the conjecture that the undecidability results in timed and hybrid systems are due to the ability of these systems to differentiate real points in time with infinite precision. We showed that the robust universality problem for timed automata and the robust reachability problem for rectangular automata are undecidable. Furthermore, we showed that the reachability problem for open rectangular automata is also undecidable. Finally, we established that there exist trajectory languages definable by open timed automata whose trajectory complements are not definable by any (open or closed) timed automata. We suspect that the undecidability proof for the robust universality problem of timed automata can be modified to prove that the trajectory universality problem for open timed automata is also undecidable, but we have not yet succeeded in doing this.

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.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[ACHH93] R. Alur, C. Courcoubetis, T.A. Henzinger, and P.-H. Ho. Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems I*, Lecture Notes in Computer Science 736, pages 209–229. Springer-Verlag, 1993.

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

[AFH94] R. Alur, L. Fix, and T.A. Henzinger. A determinizable class of timed automata. In D.L. Dill, editor, *CAV 94: Computer-aided Verification*, Lecture Notes in Computer Science 818, pages 1–13. Springer-Verlag, 1994.

[AFH96] R. Alur, T. Feder, and T.A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43(1):116–146, 1996.

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

[Fra99] M. Franzle. Analysis of Hybrid Systems: An ounce of realism can save an infinity of states. Published in: J. Flum and M. Rodriguez-Artalejo (eds.), *Computer Science Logic (CSL'99).* Lecture Notes in Computer Science 1683, Springer Verlag, 1999.

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

[Hen96] T.A. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual Symposium on Logic in Computer Science*, pages 278–292. IEEE Computer Society Press, 1996. Invited tutorial.

[HHWT95] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: the next generation. In *Proceedings of the 16th Annual Real-time Systems Symposium*, pages 56–65. IEEE Computer Society Press, 1995.

[HK97] T.A. Henzinger and P.W. Kopke Discrete-time control for rectangular hybrid automata. *ICALP 97: Automata, Languages, and Programming.* P. Degano and R. Gorrieri and A. Marchetti-Spaccamela, editors. Lecture Notes in Computer Science 1256, Springer-Verlag, 582–593, 1997.

[HKPV95] T.A. Henzinger, P.W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? Proceedings of *the 27th Annual Symposium on Theory of Computing* ACM Press, 373–382, 1995.

[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. Special issue for LICS 92.

Appendix: Proofs

Lemma 1 *There exists a timed automaton A_M that accepts exactly the trajectories that are not in $\mathsf{L}^{\mathsf{Undec}}_{\mathsf{Tube}}(M)$.*

Proof. It is sufficent to show that for each of the requirements, we can construct a timed automaton that accepts exactly the trajectories that are violating the requirement. The union of those automata is exactly what we are looking for: the timed automaton that accepts the trajectory complement of $\mathsf{L}^{\mathsf{Undec}}_{\mathsf{Tube}}(M)$.

First note that this is trivial for the requirements QUAL. In fact, regular languages are closed under negation and expressible as finite state automata. We give the two automata for the negation of the real-time requirements RT_1^b and RT_3^c. The timed automata for the other requirements are obtained in a similar way and left to the reader.

- The timed automata for requirement RT_1^b is given in figure 7 and this automaton accepts exactly the trajectories where there exists a configuration $i+1$ which is not encoded in the open slot defined by the encoding of configuration i.
- The timed automata for requirement RT_3^c is given in figure 7 and this automaton accepts exactly the trajectories where there is two adjacent configurations i and $i+1$ such that:
  - the instruction executed in configuration i increments the counter C, i.e. $b \in I^C$, where I^C is the subset of instructions that increment the counter C;
  - there is a sequence $\mathsf{b^c} \cdot \mathsf{e^c}$ in configuration i that defines an open slot in configuration $i+1$ that does not contain the sequence $\mathsf{b^c} \cdot \mathsf{e^c}$.

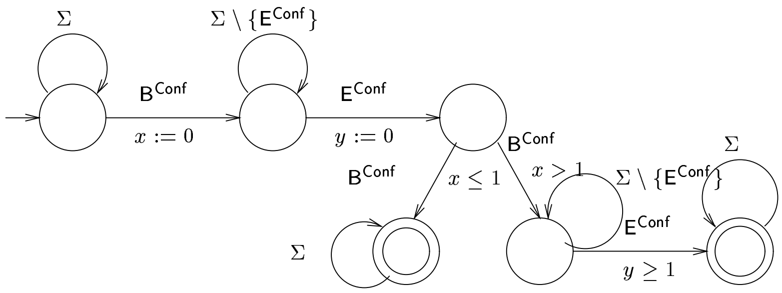

Figure 6: A timed automaton for the negation of requirement RT_1^b

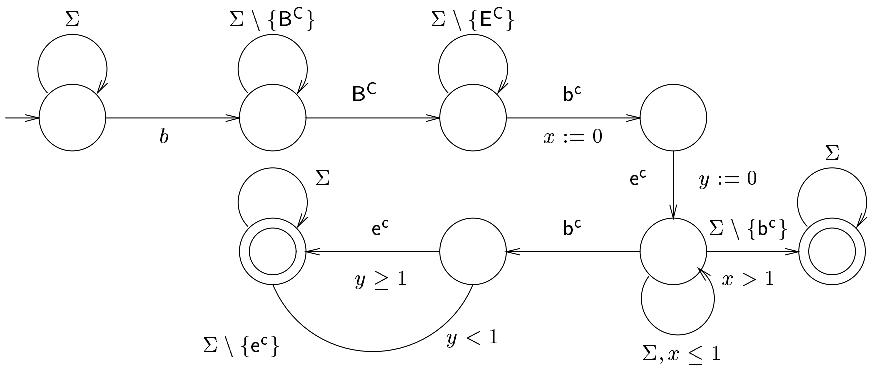

Figure 7: A timed automaton for the negation of requirement RT_3^c

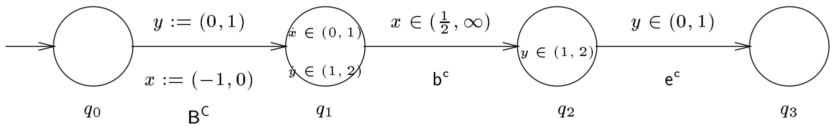

Figure 8: Open rectangular automaton for checking $C = 0$.

Theorem 4 *There are trajectory languages definable by open timed automata whose complements are not definable by any timed automata (open or closed).*

Proof. By reduction ad absurdum. We have shown that we were able to construct a timed automaton A_M that was accepting the complement of the trajectories contained in $\mathsf{L}^{\mathsf{Undec}}_{\mathsf{Tube}}(M)$. This automaton A_M defines a set of trajectories $L(A_M)$ such that $[L(A_M)]$ is exaclty the complement of the tube language $[\mathsf{L}^{\mathsf{Undec}}_{\mathsf{Tube}}(M)]$. By proposition 1, we know that there exists an open timed automaton, namely the interior of A_M, noted A^{int}_M such that $[L(A^{int}_M)] = [L(A_M)] = [\mathsf{L}^{\mathsf{Undec}}_{\mathsf{Tube}}(M)]^c$. By lemma 4, if we were able to complement the open automaton A^{int}_M, then we should obtain an automaton whose tube semantics would be precesely $[\mathsf{L}^{\mathsf{Undec}}_{\mathsf{Tube}}(M)]$ and this is impossible as emptiness of robust timed automata is decidable and this would allow us to decide the halting problem of two-counter machines.

Lemma 2 *The language* $\mathsf{L}^{\mathsf{Undec}}_{\mathsf{OpenRect}}(M)$ *is expressible by an open rectangular automata* A_M.

Proof We sketch the proof by giving an open rectangular automaton that checks if the value of C equals 0 or not. After we give an open rectangular automaton that checks that C is actually incremented after that an instruction that increments C is executed.

Let us show that the control of the automaton of figure 7 reaches the location q_3 only if the events B^{C}, b^c and e^c occurs at time t_1, t_2, and t_3, respectively, and $d_1 = t_2 - t_1$ and $d_2 = t_3 - t_1$, then $\frac{1}{2} < d_1 < d_2 < 1$. When b^c is issued, we have $d_1 \in (\frac{1}{2}, +\infty)$, in fact, the inferior value $\frac{1}{2}$ is obtained by observing that when B^{C} occurs $\mathsf{Inf}(x) = 0$, its maximal flow is bounded by 1 when the control resides in location q_1, so it takes at least $\frac{1}{2}$ time units for the variable x to overtake the value $\frac{1}{2}$. As the flow of x can be arbitrarilly small in q_1, it can take arbitrarilly long for x to overtake the value $\frac{1}{2}$, so the upper bound is $+\infty$. Let us now see what are the constraints on the value of d_2. First, note that y can be initialized arbitrarily close to 0 when the event B^{C} is issued. Also the flow of y can be arbitrarilly large when the control is in location q_1 or q_2, so we can only deduce that $d_2 \in (0, \infty)$. But as e^c has to occur before e^c, we have $d_1 < d_2$ and thus $d_2 > \frac{1}{2}$. Let us now give an upper bound on the value of d_2. The value of y when initialized must be at least strictly greater than 0. In q_1 and q_2, the flow of y is at least greater than 1 so $d_2 < 1$. We obtain as desired $\frac{1}{2} < d_1 < d_2 < 1$ if the control reaches q_3.

Let us now check that the automaton of figure 7 accepts only trajectory where an instruction that increments the counter C is encountered then time of occurence of the events B^{C}, b^c and e^c reflect this incrementation. To show that the automaton checks exactly the desired constraints, we first establish bounds on the value of the variable x and y at time t_0, t_1, t_2, and t_3 represented in figure 7. The bounds are given in table 7. So at time t_3, we have that $x \in (d_1, +\infty)$ and $y \in (-\infty, d_2)$. Now let us see the constraints that we obtain on d_3 and d_4. First, by taking into account that $x \in (d_1, +\infty)$ at t_3 and the flow of x in q_5 in included in the interval $(-2, 0)$, we can deduce that $d_3 \in (\frac{d_1}{2}, +\infty)$. Second, by taking into account that $y \in (-\infty, d_2)$ at t_3 and that the flow of y in q_5 is included in the interval $(-\infty, -2)$, we obtain $d_3 \in (-\infty, \frac{d_2}{2})$. As b^c is issued before e^c, we have that $d_3 < d_4$ and thus $\frac{d_1}{2} < d_3 < d_4 < \frac{d_2}{2}$, as desired.

	t_0	t_1	t_2	t_3
$\mathsf{Inf}(x)$	0	d_1	d_1	d_1
$\mathsf{Sup}(x)$	1	$2 \times d_1 + 1$	$d_1 + d_2 + 1$	$+\infty$
$\mathsf{Inf}(y)$	-1	-1	-1	$-\infty$
$\mathsf{Sup}(y)$	0	d_1	d_2	d_2

Figure 9: Inferior and superior bounds on the values of variables x and y.

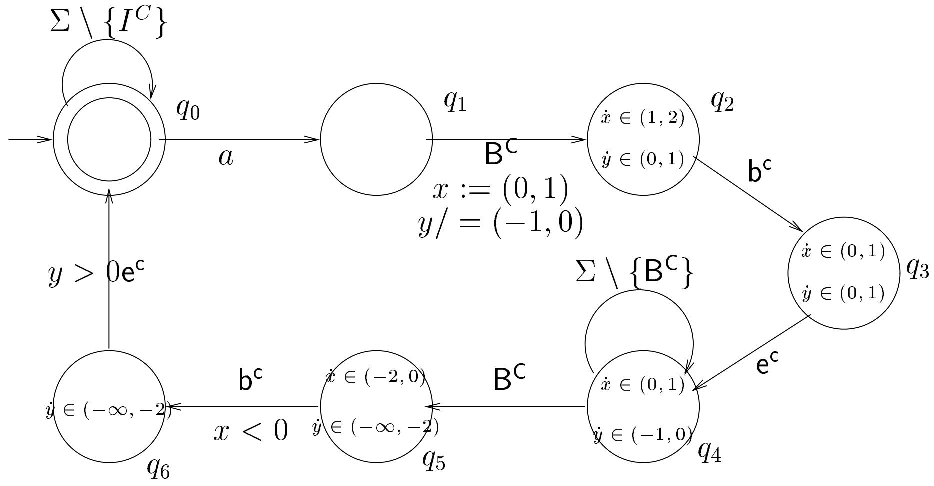

Figure 10: Open rectangular automaton to check incrementation of counter C.

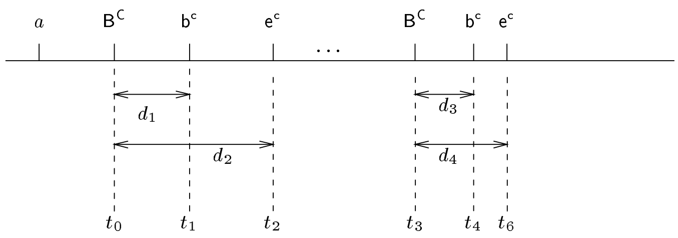

Figure 11: Two successive encodings of the value of counter C.