Interrupt Timed Automata: verification and expressiveness

Béatrice Bérard, Serge Haddad, Mathieu Sassolas

FMSD manuscript No.
(will be inserted by the editor)

Interrupt Timed Automata: Verification and Expressiveness

Béatrice Bérard · Serge Haddad · Mathieu Sassolas

Wednesday 28th March, 2012

Abstract We introduce the class of Interrupt Timed Automata (ITA), a subclass of hybrid automata well suited to the description of timed multi-task systems with interruptions in a single processor environment.

While the reachability problem is undecidable for hybrid automata we show that it is decidable for ITA. More precisely we prove that the untimed language of an ITA is regular, by building a finite automaton as a generalized class graph. We then establish that the reachability problem for ITA is in NEXPTIME and in PTIME when the number of clocks is fixed. To prove the first result, we define a subclass ITA$_-$ of ITA, and show that (1) any ITA can be reduced to a language-equivalent automaton in ITA$_-$ and (2) the reachability problem in this subclass is in NEXPTIME (without any class graph).

In the next step, we investigate the verification of real time properties over ITA. We prove that model checking SCL, a fragment of a timed linear time logic, is undecidable. On the other hand, we give model checking procedures for two fragments of timed branching time logic.

We also compare the expressive power of classical timed automata and ITA and prove that the corresponding families of accepted languages are incomparable. The result also holds for languages accepted by controlled real-time automata (CRTA), that extend timed automata. We finally combine ITA with CRTA, in a model which encompasses both classes and show that the reachability problem is still decidable. Additionally we show that the languages of ITA are neither closed under complementation nor under intersection.

Keywords Hybrid automata, timed automata, multi-task systems, interrupts, decidability of reachability, model checking, real-time properties.

Parts of this paper have been published in the proceedings of FoSSaCS'09 [9] and Time'10 [10]

Béatrice Bérard · Mathieu Sassolas
Université Pierre & Marie Curie, LIP6/MoVe, CNRS UMR 7606, Paris, France
E-mail: {beatrice.berard | mathieu.sassolas}@lip6.fr

Serge Haddad
École Normale Supérieure de Cachan, LSV, CNRS UMR 8643, INRIA, Cachan, France
E-mail: haddad@lsv.ens-cachan.fr

1 Introduction

1.1 Context

The model of timed automata (TA), introduced in [4], has proved very successful due to the decidability of several important verification problems including reachability and model checking. A timed automaton consists of a finite automaton equipped with real valued variables, called clocks, which evolve synchronously with time, during the sojourn in states. When a discrete transition occurs, clocks can be tested by guards, which compare their values with constants, and reset. The decidability results were obtained through the construction of a finite partition of the state space into regions, leading to a finite graph which is time-abstract bisimilar to the original transition system, thus preserving reachability.

Consider several tasks executing on a single processor (possibly scheduled beforehand, although this step is beyond the scope of this paper). As a result, tasks are intertwined and may interrupt one another [37]. Since the behaviour of such systems may depend on the current execution times of the tasks, a timed model should measure these execution times, which involves clock suspension in case of interruptions. Unfortunately, timed automata lack this feature of clock suspension, hence more expressive models should be considered.

Hybrid automata (HA) have subsequently been proposed as an extension of timed automata [30], with the aim to increase the expressive power of the model. In this model, clocks are replaced by variables which evolve according to a differential equation. Furthermore, guards consist of more general constraints on the variables and resets are extended into (possibly non deterministic) updates. This model is very expressive, but reachability is undecidable in HA. The simpler model obtained by allowing clocks to be stopped and resumed, stopwatch automata (SWA), would be sufficient to model task interruptions in a processor. However, reachability is also undecidable for SWA [18]. Many classes have been defined, between timed and hybrid automata, to obtain the decidability of this problem.

Task automata [23] and suspension automata [31] model explicitly the scheduling of processes. Some classes restrict the use of variation of clock rate in hybrid automata to achieve decidability. Examples of such classes are systems with piece-wise constant derivatives [6], controlled real-time automata [21]. Guards may also be restricted, as in multi-rate or rectangular automata [3], some integration graphs [26], or polygonal hybrid systems [7]. Restricting reset may also lead to decidability as in the hybrid automata with strong resets [13] or initialized stopwatch automata [24]. O-minimal hybrid systems [28,29] provide algebraic constraints on hybrid systems to yield decidability. Extensions of timed automata to release some constraints were also considered, as in some updatable timed automata [12].

While untimed properties like reachability and LTL [33,38] or CTL model checking [22,34,19], are useful for such models, real time verification consider more precise requirements, for instance quantitative response time properties. Therefore, timed extensions of these logics have been defined. In the case of linear time logics, verification of the most natural extension MTL [27] is undecidable on TA. However, several decidable fragments such as MITL [5] and SCL [35] have subsequently been defined. In the case of timed variants of branching time logics, different versions of Timed CTL (TCTL) [2,25] have been defined. Model checking procedures on TA for both versions of TCTL have been developed and implemented in several tools [8,15].

1.2 Contributions

In this paper, we define a subclass of hybrid automata, called Interrupt Timed Automata (ITA), well suited to the description of multi-task systems with interruptions in a single processor environment.

The ITA model. In an ITA, the finite set of control states is organized according to *interrupt levels*, ranging from 1 to n, with exactly one active clock for a given level. The clocks from lower levels are suspended and those from higher levels are not yet defined (thus have arbitrary value 0). On the transitions, guards are linear constraints using only clocks from the current level or the levels below and the relevant clocks can be updated by linear expressions, using clocks from lower levels. Finally, each state has a policy (lazy, urgent or delayed) that rules the sojourn time. This model is rather expressive since it combines variables with rate 1 or 0 (usually called stopwatches) and linear expressions for guards or updates. The ITA model is formally defined in Section 2.

Reachability problem. As said before, the reachability problem is undecidable for automata with stopwatches [24,18,16]. However, we prove that it is decidable for ITA.

More precisely, we first show that the untimed language of an ITA is effectively regular (Section 3). The corresponding procedure significantly extends the classical region construction of [4] by associating with each state a family of orderings over linear expressions. This construction yields a decision algorithm for reachability in 2-EXPTIME, and PTIME when the number of clocks is fixed. This should be compared to TA with 3 clocks for which reachability is PSPACE complete [20].

We define a slight restriction of the model, namely ITA$_-$, which forbids updates of clocks other than the one of the current level. We prove that for any ITA one can build an equivalent ITA$_-$ w.r.t. language equivalence, whose size is at most exponential w.r.t. the size of the ITA and polynomial when the number of clocks is fixed. Based on the existence of a bound for the length of the minimal reachability path, we then show that reachability on ITA$_-$ can be decided in NEXPTIME without any class graph construction. This yields a NEXPTIME procedure for reachability in ITA (Section 4).

Model checking over ITA. We then focus on the verification of real time properties for ITA (Section 5), expressed in timed extensions of LTL and CTL.

First we show that the model checking of timed (linear time) logic MITL [5] is undecidable. Actually, even the fragment SCL [35] cannot be verified on ITA, while the corresponding verification problem over TA is PSPACE-complete.

We then consider two fragments of the timed (branching time) logic TCTL, introduced in [25] and also studied later from the expressiveness point of view [14]. The first one, $\mathsf{TCTL}_c^{\text{int}}$, contains formulas involving comparisons of model clocks as atomic propositions. In this logic, it is possible to express properties like: *(P1) a safe state is reached before spending 3 t.u. in handling some interruption.* Decidability is obtained by a generalized class graph construction in 2-EXPTIME (PTIME if the number of clocks is fixed). Since the corresponding fragment cannot refer to global time, we consider a second fragment, TCTL_p, in which we can reason on minimal or maximal delays. Properties like *(P2) the system is error free for at least 50 t.u.* or *(P3) the system will reach a safe state within 7 t.u.* can be expressed. In this case, the decidability procedure

has a complexity in NEXPTIME for the existential fragment and 2-EXPTIME for the universal fragment (respectively NP and co-NP if the number of clocks is fixed).

Expressiveness. We also study the expressive power of the class ITA (Section 6), in comparison with the original model of timed automata and the more general controlled real-time automata (CRTA) proposed in [21]. In CRTA, clocks and states are colored and a time rate is associated with every state. During the visit of a state, all clocks colored by the color of the state evolve with the state rate while the others do not evolve. We prove that the corresponding families of languages ITL and TL, as well as ITL and CRTL, are incomparable. Additionally we show that ITL is neither closed under complementation nor under intersection.

Extensions. We finally investigate compositions of ITA and other timed models (Section 7). In the first composition, a synchronous product of an ITA and a TA, we prove that the reachability problem becomes undecidable. We then define a more appropriate product of ITA and CRTA. The CRTA part describes a basic task at an implicit additional level 0. For this extended model denoted by ITA$^+$, we show that reachability is still decidable with the same complexity and in PSPACE when the number of clocks is fixed.

2 Interrupt Timed Automata

2.1 Notations

The sets of natural, rational and real numbers are denoted respectively by $\mathbb{N}$, $\mathbb{Q}$ and $\mathbb{R}$. A *timed word* over an alphabet Σ is a finite sequence $w = (a_1, \tau_1) \dots (a_n, \tau_n)$ where a_i is in Σ and $(\tau_i)_{1 \leq i \leq n}$ is a non-decreasing sequence of real numbers. The *length* of w is n and the *duration* of w is τ_n.

For a finite set X of clocks, a linear expression over X is a term of the form $\sum_{x \in X} a_x \cdot x + b$ where b and $(a_x)_{x \in X}$ are in $\mathbb{Q}$. We denote by $\mathcal{C}(X)$ the set of constraints obtained by conjunctions of atomic propositions of the form $C \bowtie 0$, where C is a linear expression over X and $\bowtie \in \{>, \geq, =, \leq, <\}$. The subset $\mathcal{C}_0(X)$ of $\mathcal{C}(X)$ contains constraints of the form $x + b \bowtie 0$. An *update* over X is a conjunction (over X) of assignments of the form $x := C_x$, where x is a clock and C_x is a linear expression over X. The set of all updates over X is written $\mathcal{U}(X)$, with $\mathcal{U}_0(X)$ for the subset containing only assignments of the form $x := 0$ (reset) or of the form $x := x$ (no update). For a linear expression C and an update u, the expression $C[u]$ is obtained by "applying" u to C, *i.e.* substituting each x by C_x in C, if $x := C_x$ is the update for x in u. For instance, for the set of two clocks $X = \{x_1, x_2\}$, expression $C = x_2 - 2x_1 + 3$ and update u defined by $x_1 := 1 \wedge x_2 := 2x_1 + 1$, applying u to C yields the expression $C[u] = 2x_1 + 2$.

A clock valuation is a mapping $v : X \mapsto \mathbb{R}$, with $\mathbf{0}$ the valuation where all clocks have value 0. The set of all clock valuations is $\mathbb{R}^X$ and we write $v \models \varphi$ when valuation v satisfies the clock constraint $\varphi \in \mathcal{C}(X)$. For a valuation v, a linear expression C and an update u, the value $v(C)$ is obtained by replacing each x in C by $v(x)$ and the valuation $v[u]$ is defined by $v[u](x) = v(C_x)$ for x in X if $x := C_x$ is the update for x in u. Observe that an update is performed simultaneously on all clocks. For instance, let $X = \{x_1, x_2, x_3\}$ be a set of three clocks. For valuation $v = (2, 1.5, 3)$ and update

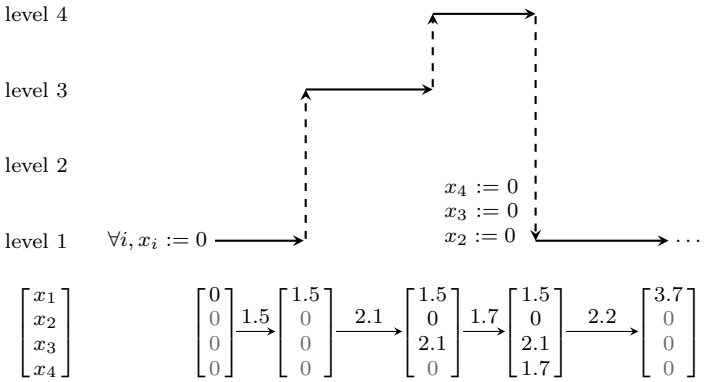


Fig. 1 Interrupt levels and clocks in an ITA.

u defined by $x_1 := 1 \wedge x_2 := x_2 \wedge x_3 := 3x_2 - x_1$, applying u to v yields the valuation $v[u] = (1, 1.5, 2.5)$.

2.2 Models of timed systems

The model of ITA is based on the principle of multi-task systems with interruptions, in a single processor environment. We consider a set of tasks with different priority levels, where a higher level task represents an interruption for a lower level task. At a given level, exactly one clock is active (rate 1), while the clocks for tasks of lower levels are suspended (rate 0), and the clocks for tasks of higher levels are not yet activated and thus contain value 0. The mechanism is illustrated in Fig. 1, where irrelevant clock values are greyed. An example of such behavior can be produced by the ITA depicted in Fig. 2, which describes a system that answer requests according to their priority. It starts by receiving a request for a *main* task of priority 1. The treatment of this task can be interrupted by tasks of priority 2 or 3, depending on how far the system is in the execution of the main task. Tasks of priority 2 and 3 may generate errors (modeled by an interruption of higher level), after which the system recovers. On this system, deciding if it is possible – or always the case – that the main task is executed in less than a certain amount of time would give an insight on the quality of service of the system.

Enabling of a transition depends on the clocks valuation. The enabling conditions, called *guards*, are linear constraints on the clock values of levels lower than or equal to the current level: the ones that are relevant before the firing of the transition. Additionally, a transition can update the values of the clocks. If the transition decreases (resp. increases) the level, then each clock which is relevant after (resp. before) the transition can either be left unchanged or take a linear expression of clocks of strictly lower level.

Along with its level, each state has a timing policy which indicates whether time may (Lazy, default), may not (Urgent) or must (Delayed) elapse in a state. Note that in TA, this kind of policy can be enforced by an additional clock while this is not possible here because there is a single clock per level. This additional feature is needed for the definition and further use of the model of ITA$_-$ (see Section 4). Note that the class graph construction of Section 3 is still valid without them.

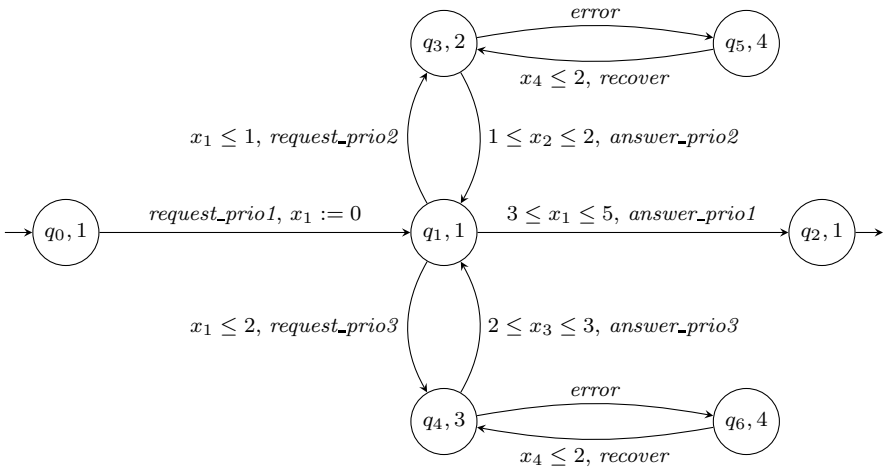


Fig. 2 An ITA that produces – among others – the behavior represented in Fig. 1.

We also add a labeling of states with atomic propositions, in view of interpreting logic formulas on these automata. In the sequel, the level of a transition is the level of its source state. We also say that a transition is lazy (resp. urgent, delayed) if the policy of its source state is lazy (resp. urgent, delayed).

Definition 1 An *interrupt timed automaton* is a tuple $\mathcal{A} = \langle \Sigma, AP, Q, q_0, F, pol, X, \lambda, lab, \Delta \rangle$, where:

- Σ is a finite alphabet, AP is a set of atomic propositions
- Q is a finite set of states, q_0 is the initial state, $F \subseteq Q$ is the set of final states,
- $pol : Q \to \{Lazy, Urgent, Delayed\}$ is the timing policy of states,
- $X = \{x_1, \ldots, x_n\}$ consists of n interrupt clocks,
- the mapping $\lambda : Q \to \{1, \ldots, n\}$ associates with each state its level and we call $x_{\lambda(q)}$ the *active clock* in state q. The mapping $lab : Q \to 2^{AP}$ labels each state with a subset of AP of atomic propositions,
- $\Delta \subseteq Q \times \mathcal{C}(X) \times (\Sigma \cup \{\varepsilon\}) \times \mathcal{U}(X) \times Q$ is the set of transitions. Let $q \xrightarrow{\varphi,a,u} q'$ in Δ be a transition with $k = \lambda(q)$ and $k' = \lambda(q')$. The guard φ is a conjunction of constraints $\sum_{j=1}^{k} a_j x_j + b \bowtie 0$ (involving only clocks from levels less than or equal to k). The update u is of the form $\wedge_{i=1}^{n} x_i := C_i$ with:
  - if $k > k'$, *i.e.* the transition decreases the level, then for $1 \leq i \leq k'$, C_i is either of the form $\sum_{j=1}^{i-1} a_j x_j + b$ or $C_i = x_i$ (unchanged clock value) and for $i > k'$, $C_i = 0$;
  - if $k \leq k'$ then for $1 \leq i \leq k$, C_i is of the form $\sum_{j=1}^{i-1} a_j x_j + b$ or $C_i = x_i$, and for $i > k$, $C_i = 0$.

A configuration (q, v, β) of the associated transition system consists of a state q of the ITA, a clock valuation v and a boolean value β expressing whether time has elapsed since the last discrete transition. This third component is needed to define the semantics according to the policies.

Definition 2 The semantics of an ITA $\mathcal{A}$ is defined by the (timed) transition system $\mathcal{T}_{\mathcal{A}} = (S, s_0, \to)$. The set S of configurations is $\Big\{(q, v, \beta) \mid q \in Q,\ v \in \mathbb{R}^X,\ \beta \in \{\top, \bot\}\Big\}$,

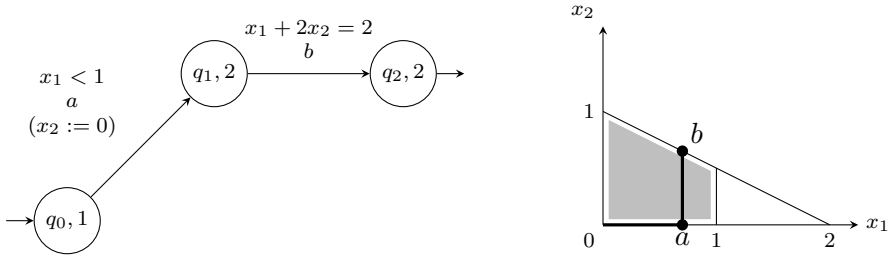
with initial configuration $s_0 = (q_0, \mathbf{0}, \bot)$. The relation $\rightarrow$ on S consists of two types of steps:

Time steps: Only the active clock in a state can evolve, all other clocks are suspended. For a state q with active clock $x_{\lambda(q)}$, a time step of duration $d > 0$ is defined by $(q, v, \beta) \xrightarrow{d} (q, v', \top)$ with $v'(x_{\lambda(q)}) = v(x_{\lambda(q)}) + d$ and $v'(x) = v(x)$ for any other clock x. A time step of duration 0 leaves the system $\mathcal{T}_{\mathcal{A}}$ in the same configuration. When $pol(q) = Urgent$, only time steps of duration 0 are allowed from q.

Discrete steps: A discrete step $(q, v, \beta) \xrightarrow{a} (q', v', \bot)$ can occur if there exists a transition $q \xrightarrow{\varphi, a, u} q'$ in Δ such that $v \models \varphi$ and $v' = v[u]$. When $pol(q) = Delayed$ and $\beta = \bot$, discrete steps are forbidden.

The labeling function lab is naturally extended to configurations by $lab(q, v, \beta) = lab(q)$.

An ITA $\mathcal{A}_1$ is depicted in Fig. 3(a), with two interrupt levels (and two interrupt clocks). A geometric view is given in figure 3(b), with a possible trajectory: first the value of x_1 increases from 0 in state q_0 (horizontal line) and, after transition a occurs, its value is frozen in state q_1 while x_2 increases (vertical line) until reaching the line $x_2 = -\frac{1}{2}x_1 + \frac{1}{2}$. The light grey zone defined by $\left(0 < x_1 < 1,\ 0 < x_2 < -\frac{1}{2}x_1 + \frac{1}{2}\right)$ corresponds to the set of valuations reachable in state q_1 and from which state q_2 is reachable.



(a) An ITA $\mathcal{A}_1$ with two interrupt levels

(b) A possible trajectory in $\mathcal{A}_1$

Fig. 3 An example of ITA and a possible execution.

We now briefly recall the classical model of Timed Automata (TA) [4] as well as the model of Controlled Real-Time Automata (CRTA) [21]. Note that in both models, timing policies can be enforced by clock constraints.

Definition 3 A *timed automaton* is a tuple $\mathcal{A} = \langle \Sigma, Q, q_0, F, X, \Delta \rangle$, where Σ, Q, q_0, F are defined as in an ITA, X is a set of clocks and the set of transitions is $\Delta \subseteq Q \times \mathcal{C}_0(X) \times (\Sigma \cup \{\varepsilon\}) \times \mathcal{U}_0(X) \times Q$, with guards in $\mathcal{C}_0(X)$ and updates in $\mathcal{U}_0(X)$.

The semantics of a timed automaton is also defined as a timed transition system, with the set $Q \times \mathbb{R}^X$ of configurations (no additional boolean value). Discrete steps are similar to those of ITA but in time steps, all clocks evolve with same rate 1: $(q, v) \xrightarrow{d} (q, v')$ iff for each clock x in X, $v'(x) = v(x) + d$.

Controlled Real-Time Automata extend TA with the following features: the clocks and the states are partitioned according to colors belonging to a set Ω and with every state is associated a rational velocity. When time elapses in a state, the set of active clocks (i.e. with the color of the state) evolve with rate equal to the velocity of the state while other clocks remain unchanged. For sake of clarity, we now propose a slightly simplified version of CRTA.

Definition 4 A CRTA $\mathcal{A} = (\Sigma, Q, q_0, F, X, up, low, vel, \lambda, \Delta)$ on a finite set Ω of colors is defined by:

- Σ, the alphabet of actions,
- Q, the set of states, with $q_0 \in Q$ the initial state and $F \subseteq Q$ the set of final states,
- X the set of clocks,
- mappings up and low associate with each clock respectively an upper and a lower bound,
- $vel : Q \mapsto \mathbb{Q}$ the velocity mapping,
- $\lambda : X \uplus Q \mapsto \Omega$ the coloring mapping and
- $\Delta \subseteq Q \times \mathcal{C}_0(X) \times (\Sigma \cup \{\varepsilon\}) \times \mathcal{U}_0(X) \times Q$ the set of transitions, with guards in $\mathcal{C}_0(X)$ and updates in $\mathcal{U}_0(X)$.

Moreover, the lower and upper bound mappings satisfy $low(x) \leq 0 \leq up(x)$ for each clock $x \in X$, and $low(x) \leq b \leq up(x)$ for each constant b such that $x \bowtie b$ is a constraint in $\mathcal{A}$.

The original semantics of CRTA is rather involved in order to obtain decidability of the reachability problem. It ensures that entering a state q in which clock x is active, the following conditions on the clock bounds hold : if $vel(q) > 0$ then $x \geq low(x)$ and if $vel(q) < 0$ then $x \leq up(x)$. Instead (and equivalently) we add a syntactical restriction which ensures this behavior. For instance, if a transition with guard φ and reset u enters state q with $vel(q) < 0$ and if x is the only clock such that $\lambda(x) = \lambda(q)$, then we replace this transition by two other transitions: the first one has guard $\varphi \wedge x > up(x)$ and adds $x := 0$ to the reset condition u, the other has guard $\varphi \wedge x \leq up(x)$ and reset u. In the general case where k clocks have color $\lambda(q)$, this leads to 2^k transitions. With this syntactical condition, again the only difference from ITA concerns a time step of duration d, defined by $(q, v) \xrightarrow{d} (q, v')$, with $v'(x) = v(x) + vel(q)d$ if $\lambda(x) = \lambda(q)$ and $v'(x) = v(x)$ otherwise.

A run of an automaton $\mathcal{A}$ in ITA, TA or CRTA is a finite or infinite path in the associated timed transition system $\mathcal{T}_\mathcal{A}$, where (possibly null) time steps and discrete steps alternate. An *accepting run* is a finite run starting in s_0 and ending in a configuration associated with a state of F. For such a run with label $d_1 a_1 d_2 \dots d_n a_n$, we say that the word $(a_1, d_1)(a_2, d_1 + d_2) \dots (a_n, d_1 + \dots + d_n)$ (where ε actions are removed) is accepted by $\mathcal{A}$. The set $\mathcal{L}(\mathcal{A})$ contains the timed words accepted by $\mathcal{A}$ and $Untimed(\mathcal{L}(\mathcal{A}))$, the untimed language of $\mathcal{A}$, contains the projections onto Σ^* of the timed words in $\mathcal{L}(\mathcal{A})$. Interrupt Timed Languages or ITL (resp. Timed Languages or TL and Controlled Real-Time Languages or CRTL) denote the family of timed languages accepted by an ITA (resp. a TA and a CRTA).

For instance, the language L_1 accepted by the ITA $\mathcal{A}_1$ in Fig. 3(a) is

$$L_1 = \mathcal{L}(\mathcal{A}_1) = \{(a, \tau)(b, 1 + \frac{\tau}{2}) \mid 0 \leq \tau < 1\}.$$

Languages of infinite timed words accepted by Büchi or Muller conditions could be studied but this analysis should address technical issues such as Zeno runs and infinite sequences of ε-transitions.

In the context of model-checking, we also consider *maximal runs* which are either infinite or such that no discrete step is possible from the last configuration. The set of maximal runs starting from configuration s is denoted by $Exec(s)$. Since maximal runs can be finite or infinite, we do not exclude Zeno behaviors. We use the notion of (totally ordered) positions (which allow to consider several discrete actions simultaneously) along a maximal run [25]: for a run ρ, we denote by $<_\rho$ the strict order over positions. For position π along ρ, the corresponding configuration is denoted by s_π, the prefix of ρ up to π is written $\rho^{\leq\pi}$ and its duration, $Dur\left(\rho^{\leq\pi}\right)$, is the sum of all delays along the finite run $\rho^{\leq\pi}$. Similarly, the suffix of ρ starting from π is denoted by $\rho^{\geq\pi}$. For two positions $\pi \leq_\rho \pi'$, the subrun of ρ between these positions is written $\rho_{[\pi,\pi']}$, its duration is $Dur\left(\rho^{\leq\pi'}\right) - Dur\left(\rho^{\leq\pi}\right)$. The length of ρ, denoted by $|\rho|$, is the number of discrete transitions occurring in ρ.

3 Regularity of untimed ITL

We prove in this section that the untimed language of an ITA is regular. Similarly to TA (and to CRTA), the proof is based on the construction of a (finite) class graph which is time abstract bisimilar to the transition system $\mathcal{T}_\mathcal{A}$. This result also holds for infinite words with standard Büchi conditions. As a consequence, we obtain decidability of the reachability problem, as well as decidability for plain CTL^* model-checking.

The construction of classes is much more involved than in the case of TA. More precisely, it depends on the expressions occurring in the guards and updates of the automaton (while in TA it depends only on the maximal constant occurring in the guards). We associate with each state q a set of expressions $Exp(q)$ with the following meaning. The values of clocks giving the same ordering of these expressions correspond to a class. In order to define $Exp(q)$, we first build a family of sets $\{E_k\}_{1\leq k\leq n}$. Then $Exp(q) = \bigcup_{k\leq\lambda(q)} E_k$ (recall that $\lambda(q)$ is the index of the active clock in state q). Finally in Theorem 1 we show how to build the class graph which proves the regularity of the untimed language. This immediately yields a reachability procedure given in Proposition 1.

3.1 Construction of $\{E_k\}_{k\leq n}$

We first introduce an operation, called *normalization*, on expressions relative to some level. As explained in the construction below, this operation will be used to order expression values at a given level.

Definition 5 (Normalization) Let $C = \sum_{i\leq k} a_i x_i + b$ be an expression over $X_k = \{x_i \mid i \leq k\}$, the *$k$-normalization* of C, $\mathtt{norm}(C,k)$, is defined by:

- if $a_k \neq 0$ then $\mathtt{norm}(C,k) = x_k + (1/a_k)(\sum_{i<k} a_i x_i + b)$;
- else $\mathtt{norm}(C,k) = C$.

Since guards are linear expressions with rational constants, we can assume that in a guard $C \bowtie 0$ occurring in a transition outgoing from a state q with level k, the expression C is either $x_k + \sum_{i<k} a_i x_i + b$ (by k-normalizing the expression and if necessary changing the comparison operator) or $\sum_{i<k} a_i x_i + b$. It is thus written as $\alpha x_k + \sum_{i<k} a_i x_i + b$, with $\alpha \in \{0, 1\}$.

The construction of $\{E_k\}_{k \leq n}$ proceeds top down from level n to level 1 after initializing $E_k = \{x_k, 0\}$ for all k. As we shall see below, when handling the level k, we add new terms to E_i for $1 \leq i \leq k$. These expressions are the ones needed to compute a (pre)order on the expressions in E_k.

- At level k, first for every expression $\alpha x_k + \sum_{i<k} a_i x_i + b$ (with $\alpha \in \{0, 1\}$) occurring in a guard of an edge leaving a state of level k, we add $-\sum_{i<k} a_i x_i - b$ to E_k.
- Then we iterate the following procedure until no new term is added to any E_i for $1 \leq i \leq k$.
  1. Let $q \xrightarrow{\varphi,a,u} q'$ with $\lambda(q) \geq k$ and $\lambda(q') \geq k$. Let $C \in E_k$, then we add $C[u]$ to E_k (recall that $C[u]$ is the expression obtained by applying update u to C).
  2. Let $q \xrightarrow{\varphi,a,u} q'$ with $\lambda(q) < k$ and $\lambda(q') \geq k$. Let C and C' be two different expressions in E_k. We compute $C'' = \mathtt{norm}(C[u] - C'[u], \lambda(q))$, choosing an arbitrary order between C and C' in order to avoid redundancy. Let us write C'' as $\alpha x_{\lambda(q)} + \sum_{i<\lambda(q)} a_i x_i + b$ with $\alpha \in \{0, 1\}$. Then we add $-\sum_{i<\lambda(q)} a_i x_i - b$ to $E_{\lambda(q)}$.

We illustrate this construction of expressions for the automaton $\mathcal{A}_1$ of Fig. 3(a). Initially, we have $E_1 = \{0, x_1\}$ and $E_2 = \{0, x_2\}$. When treating level 2, first, expression $-\frac{1}{2}x_1 + 1$ is added to E_2 as normalization of the guard $x_1 + 2x_2 = 2$. Then transition labeled by a updates x_2 (by reseting it to 0). As a result, we have to add to E_1 all differences of expressions of E_2 updated by $x_2 := 0$. This only produces expression $-\frac{1}{2}x_1 + 1 - 0$ which is normalized into $x_1 - 2$; thus expression 2 is added to E_1. When treating level 1, expression 1 from the guard of transition a is added to E_1. As a result, we obtain $E_1 = \{x_1, 0, 1, 2\}$ and $E_2 = \{x_2, 0, -\frac{1}{2}x_1 + 1\}$.

Lemma 1 *The construction procedure of $\{E_k\}_{k \leq n}$ terminates and the size of every E_k is bounded by $(E+2)^{2^{n(n-k+1)}+1}$ where E is the size of the edges of the ITA.*

Proof Given some k, we prove the termination of the stage relative to k. Observe that the second step only adds new expressions to $E_{k'}$ for $k' < k$. Thus the two steps can be ordered. Let us prove the termination of the first step of the saturation procedure. We define E_k^0 as the set E_k at the beginning of this stage and E_k^i as this set after insertion of the i^{th} item in it. With each added item $C[u]$ can be associated its *father* C. Thus we can view E_k as an increasing forest with finite degree (due to the finiteness of the edges) and finitely many roots. Assume that this step does not terminate. Then we have an infinite forest and by König lemma, it has an infinite branch $C_0, C_1, \ldots$ where $C_{i+1} = C_i[u_i]$ for some update u_i such that $C_{i+1} \neq C_i$. Observe that the number of updates that change the variable x_k is either 0 or 1 since once x_k disappears it cannot appear again. We split the branch into two parts before and after this update or we still consider the whole branch if there is no such update. In these (sub)branches, we conclude with the same reasoning that there is at most one update that change the variable x_{k-1}. Iterating this process, we conclude that the number of updates is at most $2^k - 1$ and the length of the branch is at most 2^k.

For the sake of readability, we set $B = E + 2$. The final size of E_k is thus at most $E_k^0 \times B^{2^k}$ since the width of the forest is bounded by B.

In the second step, we add at most $B \times (|E_k| \times (|E_k| - 1))/2$ to E_i for every $i < k$. This concludes the proof of termination.

We now prove by a painful backward induction that as soon as $n \geq 2$, $|E_k| \leq B^{2^{n(n-k+1)}+1}$. The doubly exponential size of E_n (proved above) is propagated downwards by the saturation procedure. We define $p_k = |E_k|$.

Basis case $k = n$. We have $p_n \leq p_n^0 \times B^{2^n}$ where p_n^0 is the number of guards of the outgoing edges from states of level n. Thus $p_n \leq B \times B^{2^n} = B^{2^n+1} = B^{2^{n(n-n+1)}+1}$ which is the claimed bound.

Inductive case. Assume that the bound holds for $k < j \leq n$. Due to all executions of the second step of the procedure at strictly higher levels, p_k^0 expressions were added to E_k, with:

$$\begin{aligned}
p_k^0 &\leq B + B \times ((p_{k+1} \times (p_{k+1} - 1))/2 + \cdots + (p_n \times (p_n - 1))/2) \\
p_k^0 &\leq B + B \times (B^{2^{n(n-k)+1}+2} + \cdots + B^{2^{n+1}+2}) \\
p_k^0 &\leq B \times (n - k + 1) \times B^{2^{n(n-k)+1}+2} \\
p_k^0 &\leq B \times B^n \times B^{2^{n(n-k)+1}+2} \quad \text{(here we use } B \geq 2\text{)} \\
p_k^0 &\leq B^{2^{n(n-k)+1}+n+3}
\end{aligned}$$

Taking into account the first step of the procedure for level k, we have:

$$p_k \leq B^{2^{n(n-k)+1}+2^k+n+3}.$$

Let us consider the term $\delta = 2^{n(n-k+1)} + 1 - (2^{n(n-k)+1} + 2^k + n + 3)$. Since $k < n$,

$$\begin{aligned}
\delta &\geq (2^{n-1} - 1)2^{n(n-k)+1} - (2^k + n + 2) \\
\delta &\geq (2^{n-1} - 1)2^{n(n-k)+1} - (2^{n-1} + 2^n) \\
\delta &\geq (2^{n-1} - 1)2^{n(n-k)+1} - 2^{n+1} \geq 0
\end{aligned}$$

Thus $p_k \leq B^{2^{n(n-k)+1}+2^k+n+3} \leq B^{2^{n(n-k+1)}+1} = (E+2)^{2^{n(n-k+1)}+1}$ which is the claimed bound. □

3.2 Construction of the class automaton

In order to analyze the size of the class automaton defined below, we recall and adapt a classical result about partitions of n-dimensional Euclidian spaces.

Definition 6 Let $\{H_k\}_{1 \leq k \leq m}$ be a family of hyperplanes of $\mathbb{R}^n$. A *region* defined by this family is a connected component of $\mathbb{R}^n \setminus \bigcup_{1 \leq k \leq m} H_k$. An *extended region* defined by this family is a connected component of $\bigcap_{k \in I} H_k \setminus \bigcup_{k \notin I} H_k$ where $I \subseteq \{1, \ldots, m\}$.

Proposition 1 ([39]) *The number of regions defined by the family* $\{H_k\}_{1\leq k\leq m}$ *is at most* $\sum_{i=0}^{n}\binom{m}{i}$.

We derive from this proposition:

Corollary 1 *The number of extended regions defined by the family* $\{H_k\}_{1\leq k\leq m}$ *is at most* $\sum_{p=0}^{n}\binom{m}{p}\sum_{i=0}^{n-p}\binom{m-p}{i}\leq e^2m^n$.

Proof Observe that an extended region is a region belonging to an intersection of at most n hyperplanes (by removing redundant hyperplanes). Thus counting the number of such intersections and applying the previous proposition yields the following formula:

$$\sum_{p=0}^{n}\binom{m}{p}\sum_{i=0}^{n-p}\binom{m-p}{i}\leq\sum_{p=0}^{n}\frac{m^p}{p!}\sum_{i=0}^{n-p}\frac{m^{n-p}}{i!}=m^n\sum_{p=0}^{n}\frac{1}{p!}\sum_{i=0}^{n-p}\frac{1}{i!}\leq e^2m^n$$

□

Theorem 1 *The untimed language of an ITA is regular.*

Proof First, we assume that the policy of every state is lazy. At the end of the proof, we explain how to adapt the construction for states with urgent or delayed policies.

Class definition. Let $\mathcal{A}$ be an ITA with E transitions and n clocks, the decision algorithm is based on the construction of a (finite) class graph which is time abstract bisimilar to the transition system $\mathcal{T}_{\mathcal{A}}$. A class is a syntactical representation of a subset of reachable configurations. More precisely, it is defined as a pair $R=(q,\{\preceq_k\}_{1\leq k\leq\lambda(q)})$ where q is a state and $\preceq_k$ is a total preorder over E_k, for $1\leq k\leq\lambda(q)$.

The class R describes the set of configurations:

$$[\![R]\!]=\{(q,v,\beta)\mid\beta\in\{\top,\bot\},\ \forall k\leq\lambda(q)\ \forall(g,h)\in E_k,\ g[v]\leq h[v]\text{ iff }g\preceq_k h\}$$

The initial state of this graph is defined by the class R_0 with $[\![R_0]\!]$ containing $(q_0,\mathbf{0},\bot)$ which can be straightforwardly determined. For example, for ITA $\mathcal{A}_1$ of Fig. 3(a), the initial class is $R_0=(q_0,Z_0)$ with $Z_0: x_1=0<1<2$. The final states are all $R=\big(q,\{\preceq_k\}_{1\leq k\leq\lambda(q)}\big)$ with $q\in F$.

Observe that fixing a state, the set of configurations $[\![R]\!]$ of a non empty class R is exactly an extended region associated with the hyperplanes defined by the comparison of two expressions of some E_k. Since $(E+2)^{2^{n^2}+1}$ is an upper bound of the number of expressions of any level, $m=(E+2)^{2^{n^2+1}+2}$ is an upper bound of the number of hyperplanes. So using Corollary 1, the number of semantically different classes for a given state is bounded by:

$$e^2m^n=e^2(E+2)^{2^{n^2+1}n+2n}$$

Since one can test semantical equality between classes in polynomial time w.r.t. their size [36], we implicitely consider in the sequel of the proof classes modulo the semantical equivalence.

As usual, there are two kinds of transitions in the graph, corresponding to discrete steps and time steps.

Discrete step. Let $R = (q, \{\preceq_k\}_{1\leq k\leq \lambda(q)})$ and $R' = (q', \{\preceq'_k\}_{1\leq k\leq \lambda(q')})$ be two classes. There is a transition $R \xrightarrow{e} R'$ for a transition $e : q \xrightarrow{\varphi,a,u} q'$ if there is some $(q, v) \in [\![R]\!]$ and $(q', v') \in [\![R']\!]$ such that $(q, v) \xrightarrow{e} (q', v')$. In this case, for all $(q, v) \in [\![R]\!]$ there is a $(q', v') \in [\![R']\!]$ such that $(q, v) \xrightarrow{e} (q', v')$. This can be decided as follows.

Firability condition. Write $\varphi = \bigwedge_{j\in J} C_j \bowtie_j 0$. Since we assumed normalized guards, for every j, $C_j = \alpha x_k + \sum_{i<k} a_i x_i + b$ (with $\alpha \in \{0, 1\}$ and $k = \lambda(q)$). By construction $C'_j = -\sum_{i<\lambda(q)} a_i x_i - b \in E_k$. For each $j \in J$, we define a condition depending on $\bowtie_j$. For instance, if $C_j \leq 0$, we require that $\alpha x_k \preceq_k C'_j$, or if $C_j > 0$ we require that $\alpha x_k \not\preceq_k C'_j \wedge C'_j \preceq \alpha x_k$.

Successor definition. R' is defined as follows. Let $k \leq \lambda(q')$ and $g', h' \in E_k$.

1. Either $k \leq \lambda(q)$, by construction, $g'[u], h'[u] \in E_k$ then $g' \preceq'_k h'$ iff $g'[u] \preceq_k h'[u]$.
2. Or $k > \lambda(q)$, let $D = g'[u] - h'[u] = \sum_{i\leq\lambda(q)} c_i x_i + d$, and $C = \mathtt{norm}(D, \lambda(q))$, and write $C = \alpha x_{\lambda(q)} + \sum_{i<\lambda(q)} a_i x_i + b$ (with $\alpha \in \{0, 1\}$). By construction $C' = -\sum_{i<\lambda(q)} a_i x_i - b \in E_{\lambda(q)}$.
   When $c_{\lambda(q)} \geq 0$ then $g' \preceq'_k h'$ iff $\alpha x_{\lambda(q)} \preceq_{\lambda(q)} C'$.
   When $c_{\lambda(q)} < 0$ then $g' \preceq'_k h'$ iff $C' \preceq_{\lambda(q)} \alpha x_{\lambda(q)}$.

By definition of $[\![\cdot]\!]$,

- For any $(q, v) \in [\![R]\!]$, if there exists $(q, v) \xrightarrow{e} (q', v')$ then the firability condition is fulfilled and (q', v') belongs to $[\![R']\!]$.
- If the firability condition is fulfilled then for each $(q, v) \in [\![R]\!]$ there exists $(q', v') \in [\![R']\!]$ such that $(q, v) \xrightarrow{e} (q', v')$.

Time step. Let $R = (q, \{\preceq_k\}_{1\leq k\leq \lambda(q)})$. There is a transition $R \xrightarrow{succ} Post(R)$ for $Post(R) = (q, \{\preceq'_k\}_{1\leq k\leq \lambda(q)})$, the time successor of R, which is defined as follows.

For every $i < \lambda(q)$ $\preceq'_i = \preceq_i$. Let $\sim$ be the equivalence relation $\preceq_{\lambda(q)} \cap \preceq^{-1}_{\lambda(q)}$ induced by the preorder. On equivalence classes, this (total) preorder becomes a (total) order. Let V be the equivalence class containing $x_{\lambda(q)}$.

1. Either $V = \{x_{\lambda(q)}\}$ and it is the greatest equivalence class. Then $\preceq'_{\lambda(q)} = \preceq_{\lambda(q)}$ (thus $Post(R) = R$).
2. Either $V = \{x_{\lambda(q)}\}$ and it is not the greatest equivalence class. Let V' be the next equivalence class. Then $\preceq'_{\lambda(q)}$ is obtained by merging V and V', and preserving $\preceq_{\lambda(q)}$ elsewhere.
3. Either V is not a singleton. Then we split V into $V \setminus \{x_{\lambda(q)}\}$ and $\{x_{\lambda(q)}\}$ and "extend" $\preceq_{\lambda(q)}$ by $V \setminus \{x_{\lambda(q)}\} \preceq'_{\lambda(q)} \{x_{\lambda(q)}\}$.

By definition of $[\![\cdot]\!]$, for each $(q, v) \in [\![R]\!]$, there exists $d > 0$ such that $(q, v + d) \in [\![Post(R)]\!]$ and for each d with $0 \leq d' \leq d$, then $(q, v + d') \in [\![R]\!] \cup [\![Post(R)]\!]$.

We now explain how the policy is handled. Given a state q such that $pol(q) = U$, for every class $R = (q, \{\preceq_k\}_{1\leq k\leq \lambda(q)})$ we delete the time steps outgoing from R. The case of a state q such that $pol(q) = D$, is a little bit more involved. First we partition classes between *time open* classes, where for every every configuration of the class there exists a small amount of time elapse that let the new configuration in the same class, and

time closed classes. The partition is performed w.r.t. the equivalence class V of $x_{\lambda(q)}$ for the relation $\sim$ (see above in the proof). The class R is time open iff $V = \{x_{\lambda(q)}\}$. Then we successively replace every time closed class R by two copies R^- and R^+, which capture wether time has elapsed since the last last discrete step. Thus, a time edge entering R is redirected towards R^+ while a discrete edge entering R is redirected towards R^-. A time step $R \xrightarrow{succ} R'$ is replaced by two transitions $R^- \xrightarrow{succ} R'$ and $R^+ \xrightarrow{succ} R'$, while a discrete step $R \xrightarrow{e} R'$ is replaced by the transition $R^+ \xrightarrow{e} R'$. Time open classes allow time elapsing, hence no splitting is required for these classes.

Since there is at most one time edge outgoing from a class, the number of edges of the new graph is at most twice the number of edges in the original graph. □

Proposition 2 *The reachability problem for Interrupt Timed Automata is decidable and belongs to* 2-EXPTIME *and* PTIME *when the number of clocks is fixed.*

Proof The reachability problem is solved by building the class graph and applying standard reachability algorithm. Since the number of semantically different classes is at most doubly exponential in the size of the model and the semantical equivalence can be checked in polynomial time w.r.t. the size of the class (also doubly exponential) this leads to a 2-EXPTIME complexity. When the number of clocks is fixed the size of the graph is at most polynomial w.r.t. the size of the problem leading to a PTIME procedure. No complexity gain can be obtained by a non deterministic search without building the graph since the size of the graph is only polynomial w.r.t. the size of a class. □

Remarks. This result should be contrasted with the similar one for TA. The reachability problem for TA is PSPACE-complete and thus less costly to solve than for ITA. However, fixing the number of clocks does not reduce the complexity for TA (when this number is greater than or equal to 3) while this problem belongs now to PTIME for ITA. Summarizing, the main source of complexity for ITA is the number of clocks, while in TA it is the binary encoding of the constants [20].

Since the construction of the graph depends on a set of expressions, there is no notion of *granularity* as in Timed Automata. When the only guards are comparisons to constants and the only updates resets of clocks (as in Timed Automata), the abstraction obtained is coarser than the region abstraction of [4]: it consists only in products of intervals.

3.3 Example

We illustrate this construction of a class automaton for the automaton $\mathcal{A}_1$ of Fig. 3(a). The resulting class automaton is depicted on Fig. 4, where dashed lines indicate time steps.

Recall that we obtained $E_1 = \{x_1, 0, 1, 2\}$ and $E_2 = \left\{x_2, 0, -\frac{1}{2}x_1 + 1\right\}$. In state q_0, the only relevant clock is x_1 and the initial class is $R_0 = (q_0, Z_0)$ with $Z_0 : x_1 = 0 < 1 < 2$. Its time successor is $R_0^1 = (q_0, Z_0^1)$ with $Z_0^1 : 0 < x_1 < 1 < 2$. Transition a leading to q_1 can be taken from both classes, but not from the next time successors $R_0^2 = (q_0, 0 < x_1 = 1 < 2)$, $R_0^3 = (q_0, 0 < 1 < x_1 < 2)$, $R_0^4 = (q_0, 0 < 1 < x_1 = 2)$, or $R_0^5 = (q_0, 0 < 1 < 2 < x_1)$.

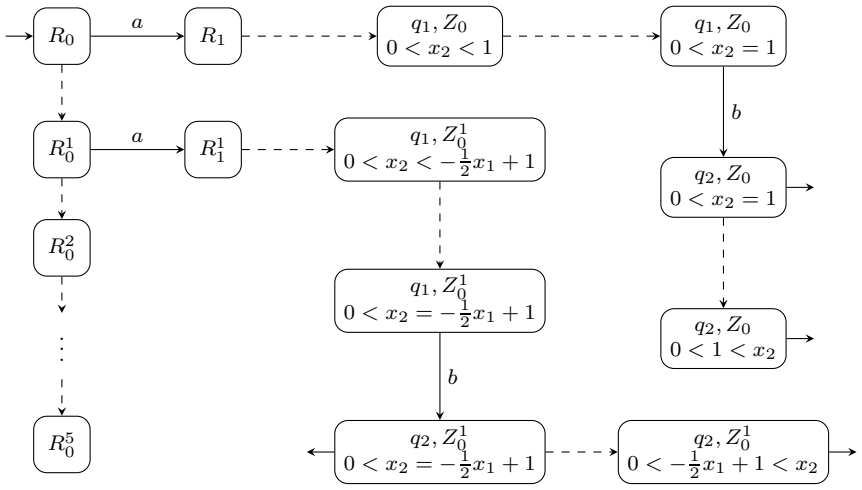


Fig. 4 The class automaton for $\mathcal{A}_1$

Transition a switches from R_0 to $R_1 = (q_1, Z_0, x_2 = 0 < 1)$, because $x_1 = 0$, and from R_0^1 to $R_1^1 = \left(q_1, Z_0^1, x_2 = 0 < -\frac{1}{2}x_1 + 1\right)$. Transition b is fired from those time successors for which $x_2 = -\frac{1}{2}x_1 + 1$.

On the geometric view of figure 3(b), the displayed trajectory corresponds to the following path in the class automaton:

$$R_0 \to R_0^1 \xrightarrow{a} R_1^1 \to \left(q_1, Z_0^1, 0 < x_2 < -\frac{1}{2}x_1 + 1\right) \to \left(q_1, Z_0^1, 0 < x_2 = -\frac{1}{2}x_1 + 1\right)$$
$$\xrightarrow{b} \left(q_2, Z_0^1, 0 < x_2 = -\frac{1}{2}x_1 + 1\right)$$

4 A simpler model

4.1 Definition of ITA$_-$

We introduce a restricted version of ITA, called ITA$_-$, which is interesting both from a theoretical and a practical point of view. When modeling interruptions in real-time systems, the clock associated with some level measures the time spent in this level or more generally the time spent by some tasks at this level. Thus when going to a higher level, this clock is not updated until returning to this level. The ITA$_-$ model takes this feature into account. Moreover, it turns out that the reachability problem for ITA$_-$ can be solved more efficiently. This also provides a better complexity upper-bound for the reachability problem on ITA (in the general case).

Definition 7 The subclass ITA$_-$ of ITA is defined by the following restriction on updates. For a transition $q \xrightarrow{\varphi,a,u} q'$ of an automaton $\mathcal{A}$ in ITA$_-$ (with $k = \lambda(q)$ and $k' = \lambda(q')$), the update u is of the form $\wedge_{i=1}^{n} x_i := C_i$ with:

- if $k > k'$, then for $1 \leq i \leq k'$, $C_i := x_i$ and for $k'+1 \leq i \leq n$, $C_i = 0$, *i.e.* the only updates are the resets of now irrelevant clocks;

– if $k \leq k'$ then C_k is of the form $\sum_{j=1}^{k-1} a_j x_j + b$ or $C_k = x_k$. For $k < i \leq k'$, $C_i = 0$ and $C_i = x_i$ otherwise.

Thus, complex updates appear only in transitions increasing the level, and only for the active clock of the transition level.

The proof of the following result is based on Propositions 3 and 5 proved in the next two sections.

Theorem 2 *The reachability problem for ITA belongs to* NEXPTIME.

Proof Given an ITA $\mathcal{A}$ with transitions of size E and constants coded over b bits, we build the ITA$_-$ $\mathcal{A}'$ of Proposition 3. Then we apply on $\mathcal{A}'$ the reachability procedure of Proposition 5. In this procedure, we consider paths of length bounded by $(E'+n)^{3n}$, where E' is the number of transitions of $\mathcal{A}'$. Since $E' \leq 2^{4b \cdot E \cdot n^2}$ (as shown in the proof of Proposition 3), the length of the paths considered is bounded by

$$(E'+n)^{3n} \leq \left(2^{4b \cdot E \cdot n^2} + n\right)^{3n} \leq (n+2)^{12b \cdot E \cdot n^3}$$

which establishes the claimed upper bounds. □

4.2 From ITA to ITA$_-$

In this subsection we prove that ITA and ITA$_-$ are equivalent w.r.t. the associated (timed) languages.

Proposition 3 *Given an ITA $\mathcal{A}$, we build an automaton $\mathcal{A}'$ in ITA$_-$ accepting the same timed language and with the same clocks such that its number of edges (resp. states) is exponential w.r.t. the number of edges (resp. states) in $\mathcal{A}$ and polynomial when the number of clocks is fixed.*

Proof Starting from ITA $\mathcal{A} = \langle \Sigma, AP, Q, q_0, F, pol, X, \lambda, lab, \Delta \rangle$, the construction of automaton $\mathcal{A}'$ relies on memorizing at a given level i, for every clock x_j at a lower level, an expression depending on $x_1, \ldots, x_{j-1}$, corresponding to the delayed update of x_j. This expression is used later to replace the value of x_j in guards and to restore its correct value by update after decreasing to level j.

To this aim we associate with every pair of levels $i \geq j$, a set of expressions $F_{i,j}$ inductively defined by:

– $F_{i,i} = \{x_i\}$
– $\forall i > j$ $F_{i,j} = F_{i-1,j} \cup \{e[\{x_k \leftarrow e_k\}_{k<j}] \mid e$ is the expression of an update of x_j by an edge of level i and $\forall k,\ e_k \in F_{i,k}\}$

We write $F_j = F_{n,j} = \bigcup_{i=j}^{n} F_{i,j}$. The set F_j thus contains all expressions of updates of x_j that appear at higher levels.

Although the number of expressions is syntactically doubly exponential w.r.t. the number of clocks, one can show that the number of *distinct* expressions is only singly exponential.

First we assume that ITA $\mathcal{A}$ has only integral constants, the case of rational constants is handled at the end of the proof. It can be shown that every expression e_k of F_k can be written

$$e_k = \sum_{i_0, \ldots, i_p \in \mathrm{sub}(k)} \alpha_{k,i_p} \cdot \alpha_{i_p,i_{p-1}} \cdots \alpha_{i_1,i_0} \cdot x_{i_0}$$

with the convention that x_0 is the constant 1, and where $\mathrm{sub}(k)$ is the set of all (ordered) subsequences of $0, \ldots, k-1$ and $\alpha_{j,i}$ is the coefficient of x_i in some update of x_j.

For the family α of all integers $\alpha_{j,i}$, assume that these constants are coded over b_α bits each (including the sign of the coefficient). The expression x_{i_0} can also be coded into an integer of $\log_2(n)$ bits (with a special symbol to indicate that it is the expression of a clock rather than a constant). Let $b = \max(b_\alpha, \log_2(n) + 1)$ be the (maximal) number of bits used to code a coefficient. Then each term of the sum is a product of at most k such coefficients, therefore can be coded with kb bits. Summing at most 2^k such products yields an integer that can be coded over $kb + k$ bits. Thus there can be at most $2^{k(b+1)}$ different expressions in E_k.

Automaton $\mathcal{A}'$ is then defined as follows.

- The set of states is

$$Q' = \{(q^+, e_1, \ldots, e_{i-1}) \mid q \in Q,\ \lambda(q) = i \text{ and } \forall j,\ e_j \in F_j\} \\ \cup \{(q^-, e_1, \ldots, e_i) \mid q \in Q,\ \lambda(q) = i \text{ and } \forall j,\ e_j \in F_j\},$$

  with $pol(q^+, e_1, \ldots, e_{i-1}) = pol(q)$ and $pol(q^-, e_1, \ldots, e_i) = U$.
  Note that the sequence is empty if $i = 1$. Moreover:

$$\lambda(q^+, e_1, \ldots, e_{i-1}) = \lambda(q^-, e_1, \ldots, e_i) = \lambda(q).$$

- The initial state of $\mathcal{A}'$ is $(q_0^+, x_1, \ldots, x_{i-1})$ if $\lambda(q_0) = i$. The final states of $\mathcal{A}'$ are the states with first component q^+ for $q \in F$.
- Let $q \xrightarrow{\varphi,a,u} q'$ be a transition in $\mathcal{A}$ such that $\lambda(q) = i$, $\lambda(q') = i'$ and u is defined by $\bigwedge_{j=1}^{i} x_j := C_j$.
  - If $i \leq i'$, then for every $(q^+, e_1, \ldots, e_{i-1})$ there is a transition

$$(q^+, e_1, \ldots, e_{i-1}) \xrightarrow{\varphi',a,u'} (q'^+, e'_1, \ldots, e'_{i'-1})$$

    in $\mathcal{A}'$ with $\varphi' = \varphi(\{x_j \leftarrow e_j\}_{j<i})$, update u' is defined by $x_i := C_i[\{x_j \leftarrow e_j\}_{j<i}]$; for all $j < i$, $e'_j = C_j[\{x_k \leftarrow e_k\}_{k<i}]$ and for all j such that $i \leq j < i'$, $e'_j = x_j$.
  - If $i > i'$ then for every $(q^+, e_1, \ldots, e_{i-1})$ there is a transition

$$(q^+, e_1, \ldots, e_{i-1}) \xrightarrow{\varphi',a,u'} (q'^-, e'_1, \ldots, e'_{i'})$$

    in $\mathcal{A}'$ with $\varphi' = \varphi(\{x_j \leftarrow e_j\}_{j<i})$, update u' contains only the trivial updates $x_j := x_j$ for all clocks and for all $j \leq i'$, $e'_j = C_j[\{x_k \leftarrow e_k\}_{k<i}]$.
  - For every $(q^-, e_1, \ldots, e_i)$ there is in $\mathcal{A}'$ a transition

$$(q^-, e_1, \ldots, e_i) \xrightarrow{true,\varepsilon,x_i:=e_i} (q^+, e_1, \ldots, e_{i-1}).$$

In words, given a transition, the guard is modified according to these expressions. The modification of the update consists only in applying the update at the current level and taking into account the other updates in the expressions labeling the destination state. When the transition increases the level, the expression associated with a new "frozen" clock (x_j for $i \leq j < i'$) is the clock itself. The urgent states $(q^-, -)$ are introduced for handling the case of a transition that decreases the level. In this case, one reaches such a state that memorizes also the expression of the clock at the current level. Note that the memorized expressions can correspond to an update proceeded

at any (higher) level. From this state a single transition must be (immediately) taken whose effect is to perform the update corresponding to the memorized expression.

It is routine to check that the languages of the two automata are identical. Each transition in $\mathcal{A}$ is replaced by several transitions in $\mathcal{A}'$, which number is bounded by the number of expressions that can be attached to the source of the original transition. In addition, transitions decreasing level are further "split" through states $(q^-, -)$. Thus the number E' of transitions in $\mathcal{A}'$ is bounded by

$$\begin{aligned} E' &\leq 2 \cdot E \cdot |F_n|^n \\ &\leq 2 \cdot E \cdot \left(2^{n(b(E+1)+1)}\right)^n \\ &\leq 2 \cdot E \cdot 2^{n^2(b(E+1)+1)} \\ &\leq 2^{n^2(b(E+1)+1)+1+\log_2(E)} \\ &\leq 2^{n^2((b+1)(E+1)+1)} \\ E' &\leq 2^{4b \cdot E \cdot n^2} \end{aligned}$$

(provided $E \geq 2$). This yields the exponential complexity for the number of transitions. The case of the number of states is similar.

In the case when there are rational constants, assume each constant is coded with a pair (r, d) of numerator and denominator. Assume each r and d can be coded over b bits. We compute the *lcm* δ of all denominators: since there are at most E constants (E, the size of Δ contains the number of guards and updates), δ can be coded over Eb bits. We consider ITA $\mathcal{A}_\delta$ which is $\mathcal{A}$ where all constants are multiplied by δ. Thus a constant of $\mathcal{A}_\delta$ is an integer that can be coded over $b' = Eb + b = b(E+1)$ bits. The above bound on the number of expressions applies on $\mathcal{A}_\delta$. Note that after the construction of $\mathcal{A}'_\delta$, $\mathcal{A}'$ can be obtained by dividing each constant in $\mathcal{A}'_\delta$ by δ. □

Example. We illustrate this construction on ITA $\mathcal{A}_2$ of Fig. 5. The sets of expressions are computed as on Table 1 and the resulting ITA$_-$ $\mathcal{A}'_2$ is depicted on Fig. 6.

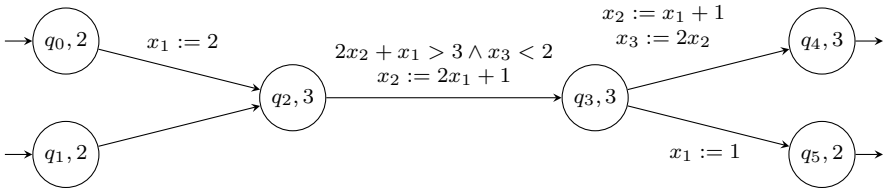


Fig. 5 ITA $\mathcal{A}_2$ containing updates of frozen clocks

The translation above of an ITA into an equivalent ITA$_-$ induces an exponential blowup. The proposition below shows that the bound is reached.

Proposition 4 *There exist a family $\{\mathcal{A}_n\}_{n\in\mathbb{N}}$ of ITA with two states, n clocks and constants coded over b bits, where b is polynomial in n, such that the equivalent ITA$_-$ built by the procedure above has a number of states greater than or equal to 2^n.*

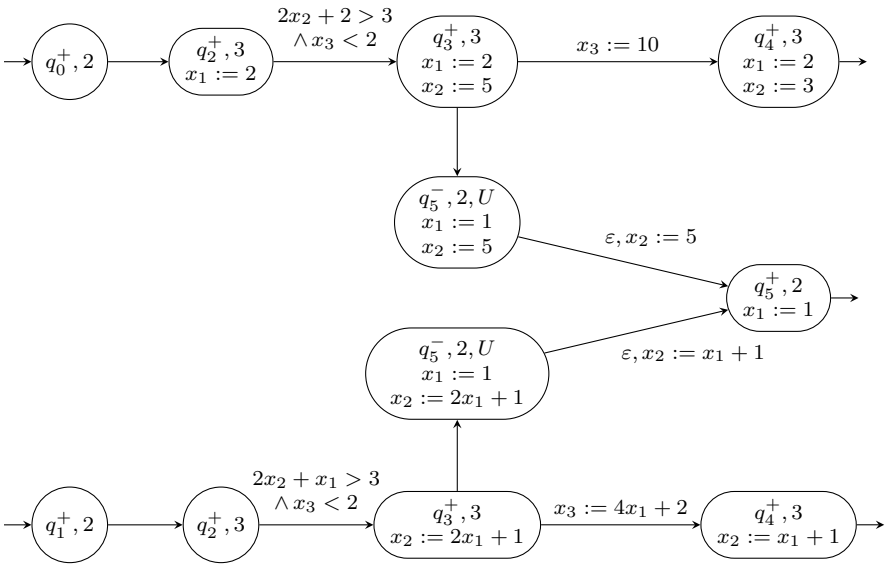


Fig. 6 ITA$_-$ $\mathcal{A}'_2$ equivalent to $\mathcal{A}_2$

$i \setminus j$	1	2	3
1	$\{x_1\}$		
2	$\{x_1, 2\}$	$\{x_2\}$	
3	$\{x_1, 2, 1\}$	$\{x_2,\ \underbrace{2x_1+1,\ 5,\ 3}_{q_2 \xrightarrow{x_2:=2x_1+1} q_3}\ ,\ \underbrace{x_1+1,\ 3,\ 2}_{q_3 \xrightarrow{x_2:=x_1+1} q_4}\}$	$\{x_3\}$

Table 1 Sets of expressions $F_{i,j}$ for $\mathcal{A}_2$.

Proof For $n \in \mathbb{N}$, let $\mathcal{A}_n$ be the ITA with n clocks and two states q_{init} (initial) and q (final) both of level n (and lazy policy) built as follows. There is a transition from q_{init} to q with update $\bigwedge_{k=1}^{n} x_k := 1$ that sets all clocks to 1. For $1 \leq k \leq n$ there are two loops on q with updates $x_k := x_{k-1}$ and $x_k := \alpha_k x_{k-1}$ respectively, where α_k is the kth prime number (and with the convention that x_0 is the constant 1).

When building the sets of expressions, no expressions are added until level n, since all updates occur at this level. At level k, $F_{n,k}$ contains (at least) 2^k expressions: all possible products of the first k prime numbers, namely

$$F_{n,k} \supset \left\{ \prod_{i \in I} \alpha_i \;\middle|\; I \subseteq \{1, \ldots, k\} \right\}.$$

Indeed, at level 1, $F_{n,1} = \{x_1, 1, 2\}$. Now assume that $F_{n,k-1}$ contains all products $\prod_{i \in I} \alpha_i$ where $I \subseteq \{1, \ldots, k-1\}$. By update $x_k := x_{k-1}$, $F_{n,k} \supset F_{n,k-1}$. By update $x_k := \alpha_k x_{k-1}$, $F_{n,k}$ contains all products $\alpha_k \prod_{i \in I} \alpha_i = \prod_{i \in I \uplus \{k\}} \alpha_i$. Therefore

$$F_{n,k} \supset \left\{ \prod_{i \in I} \alpha_i \;\middle|\; I \subseteq \{1, \ldots, k-1\} \right\} \cup \left\{ \prod_{i \in I \uplus \{k\}} \alpha_i \;\middle|\; I \subseteq \{1, \ldots, k-1\} \right\}$$

$$F_{n,k} \supset \left\{ \prod_{i \in I} \alpha_i \;\middle|\; I \subseteq \{1, \ldots, k\} \right\}.$$

The expressions thus built are distinct, since they are products of distinct prime numbers. Remark that the set of expression for level k is in bijection with a sequence of updates $x_1 := \ldots, x_2 := \ldots, \ldots, x_k := \ldots$, the choice of the update depending on the choice of the set I.

Therefore all expressions of $F_{n,n}$ are reached (in association with state q) and the set of states in $\mathcal{A}'_n$ is at least of size 2^n. In addition, it should be noted that the nth prime number is in $O(n \log_2(n))$, therefore can be coded over $O(\log_2(n)^2)$ bits. So the size of the constants appearing in the updates (and the size of the representation of $\mathcal{A}_n$) is polynomial in n while the representation of $\mathcal{A}'_n$ is exponential in n.

4.3 Reachability on ITA$_-$

In this section we use counting arguments to obtain an upper bound for the reachability problem on ITA$_-$.

The following counting lemma does not depend on the effect of the updates but only on the timing constraints induced by the policies.

Lemma 2 (Counting Lemma) *Let $\mathcal{A}$ be an ITA$_-$ with E transitions and n clocks, then in a sequence $(e_1, \ldots, e_l)$ of transitions of $\mathcal{A}$ where $l > (E+n)^{3n}$, there exist $i < j$ with $e_i = e_j$ such that the level of any transition e_k with $i \leq k \leq j$ is greater than or equal to the level of e_i, say p, and:*

- *either e_i updates x_p,*
- *either no e_k with $i \leq k \leq j$ updates x_p and e_i is delayed or lazy.*
- *or no e_k with $i \leq k \leq j$ updates x_p and no time elapses for clock x_p between e_i and e_j.*

Proof Assume that the conclusions of the lemma are not satisfied, we claim that $l \leq (E + 2n)^{3n}$.

First we prove that the number of transitions of level m that occur between two occurrences of transitions of strictly lower level is less than or equal to $(E+2)^3$. Indeed there can be no more than E occurrences of transitions that update x_m. Then between two such transitions (or before the first or after the last) there can be no more than E lazy or delayed transitions of level m that do not update x_m. Finally between any kind of previous transitions (or before the first or after the last), there can be no more than E urgent transitions that do not update x_m, since they prevent time from elapsing at level m.

Summing up, there can be no more than $E+E(E+1)+E(E(E+1)+1) \leq (E+1)^3$ transitions of level m that occur between two occurrence of transitions of strictly lower level.

Now we prove by induction that the number of transitions at level less than or equal to m is at most $(E + m)^{3m}$. This is true for $m = 1$ by the previous proof. Assume the formula valid for m, then grouping the transitions of level $m + 1$ between the occurrences of transition of lower level (or before the first or after the last), we obtain that the number of transitions at levels less than or equal to $m + 1$ is at most:

$$\begin{aligned}(E+m)^{3m} + ((E+m)^{3m} + 1)(E+1)^3 &\leq (E+m)^{3m+3} + 2(E+m)^{3m} \\ &\leq (E+m+1)^{3(m+1)} \qquad \square\end{aligned}$$

Proposition 5 *The reachability problem for ITA$_-$ belongs to* NEXPTIME*. More precisely, reachability can be checked over paths with length less than or equal to* $(E+n)^{3n}$*, where* E *is the number of transitions and* n *is the number of clocks.*

Proof Let $\mathcal{A} = (\Sigma, Q, q_0, F, pol, X, \lambda, \Delta)$ be an ITA$_-$ with n clocks. Let $E = |\Delta|$ be the number of transitions of $\mathcal{A}$. Assume that there is a run of minimal length ρ from (q_0, v_0) to some configuration (q_f, v_f). Suppose now that $|\rho| > B = (E+n)^{3n}$. We will build a run ρ' from (q_0, v_0) to (q_f, v_f) that is strictly smaller, hence contradicting the minimality hypothesis.

Since $|\rho| > B$, then one of the three cases of Lemma 2 applies. Therefore there is a transition e at level k repeated twice, from positions π and π' and separated by a subrun σ containing only transitions of level higher than or equal to k. Moreover:

- Either e updates x_k. In this case, all clocks have the same value after the first and the second occurrence of e. Hence removing $e\sigma = \rho_{[\pi,\pi'[}$ from ρ yields a valid run ρ' of $\mathcal{A}$ reaching (q_f, v_f). Run ρ' is strictly smaller than ρ, since $e\sigma$ which is of length at least 1 was removed.
- Either no update occurred for x_k and e is delayed or lazy. In this case, upon reaching π', the clocks of level $i < k$ have retained the same value, while x_k has increased by $Dur\left(\rho_{[\pi,\pi']}\right)$. Hence when replacing $e\sigma = \rho_{[\pi,\pi'[}$ by a time step of duration $Dur\left(\rho_{[\pi,\pi']}\right)$, the configuration in π' is unchanged. In addition, since e was delayed or lazy, this time step is allowed in $\mathcal{A}$, and this yields a shorter run of $\mathcal{A}$.
- Or no update occurred and π and π' are at the same instant (separated by instantaneous actions). In this case, all clocks of level smaller than or equal to k again have the same value after the first and the second occurrence of e. Again removing $\rho_{[\pi,\pi'[}$ yields a smaller run.

The decision procedure is as follows. It non deterministically guesses a path in the ITA$_-$ whose length is less than or equal to the bound. In order to check that this path yields a run, it builds a linear program whose variables are $\left\{x_i^j\right\}$, where x_i^j is the value of clock x_i after the jth step, and $\{d_j\}$ where d_j is the amount of time elapsed during the jth step, when j corresponds to a time step. The equations and inequations are deduced from the guards and updates of discrete transitions in the path and the delay of the time steps. The size of this linear program is exponential w.r.t. the size of the ITA$_-$. As a linear program can be solved in polynomial time [36], we obtain a procedure in NEXPTIME. □

One could wonder whether the class graph construction would lead to a better complexity when applied on ITA$_-$. Unfortunately, the number of expressions occurring in the class graph while being smaller than for ITA is still doubly exponential w.r.t. the size of the model.

5 Timed model-checking

First observe that model-checking CTL* formulas on ITA can be done with classical procedures on the class graph previously built. We now consider verification of real time formulas.

In the case of linear time, the logic LTL has been extended into the Metric Temporal Logic (MTL) [27], by adding time intervals as constraints to the U modality. However, MTL suffers from undecidability of the model-checking problem on TA. Hence decidable fragments have been proposed, such as Metric Interval Temporal Logic (MITL) [5], which prohibits the use of point intervals (of the form $[a, a]$). Later, MITL was restricted into State Clock Logic (SCL) [35], in order to obtain more efficient verification procedures. Model-checking MITL (thus SCL) on TA is decidable. Unfortunately, we show here that model-checking SCL (thus MITL) on ITA is undecidable. For this, we reduce the halting problem on a two counter machine into model-checking an SCL formula on an ITA.

Concerning branching time logics, at least two different timed extensions of CTL have been proposed. The first one [2] also adds time intervals to the U modality while the (more expressive) second one considers formula clocks [25]. Model-checking timed automata was proved decidable in both cases and compared expressiveness was revisited later on [14].

We conjecture that model-checking of TCTL is undecidable when using two (or more) formula clocks. Indeed, as shown in Section 7.1, the reachability problem in a product of an ITA and a TA with two clocks is undecidable, thus prohibiting model-checking techniques through automaton product and reachability testing as in [1]. However, contrary to what is claimed in [10], this is not enough to yield an undecidability proof.

Two fragments for which model-checking is decidable on ITA have nonetheless been identified. The first one, $\mathsf{TCTL}_c^{\text{int}}$, accepts only internal clocks (from the automaton on which the formulas will be evaluated) as formula clocks. The second one, TCTL_p, restricts the nesting of U modalities. We provide verification procedures in both cases.

5.1 Undecidability of State Clock Logic

We first consider the timed extension of linear temporal logic, and more particularly the SCL fragment [35].

Definition 8 Formulas of the timed logic SCL are defined by the following grammar:

$$\psi = p \mid \psi \wedge \psi \mid \neg\psi \mid \psi \,\mathsf{U}\, \psi \mid \psi \,\mathsf{S}\, \psi \mid 4_{\bowtie a}\psi \mid 2_{\bowtie a}\psi$$

where $p \in AP$ is an atomic proposition, $\bowtie \in \{>, \geq, =, \leq, <\}$, and a is a rational number.

We use the usual shorthands $\mathbf{t}$ for $\neg(p \wedge \neg p)$, $\mathsf{F}\psi$ for $\mathbf{t}\,\mathsf{U}\,\psi$, $\mathsf{G}\psi$ for $\neg(\mathsf{F}\neg\psi)$ and $\varphi \Rightarrow \psi$ for $\neg(\varphi \wedge \neg\psi)$.

The semantics are defined in the usual manner for boolean operators and U. The S modality is the past version of U. Modality $4_{\bowtie a}\psi$ is true if the *next* time ψ is true will occur in a delay that respects the condition $\bowtie a$. Similarly, $2_{\bowtie a}\psi$ is true if the *last* time ψ was true occurred in a (past) delay that respects the condition $\bowtie a$. More

formally, for an execution ρ, we inductively define $(\rho,\pi) \models \varphi$ by:

$(\rho,\pi) \models p$ iff $p \in lab(s_\pi)$
$(\rho,\pi) \models \varphi \wedge \psi$ iff $(\rho,\pi) \models \varphi$ and $(\rho,\pi) \models \psi$
$(\rho,\pi) \models \neg\varphi$ iff $(\rho,\pi) \not\models \varphi$
$(\rho,\pi) \models \varphi \,\mathsf{U}\, \psi$ iff there is a position $\pi' \geq_\rho \pi$ such that $(\rho,\pi') \models \psi$ and forall π'' s.t. $\pi \leq_\rho \pi'' <_\rho \pi'$, $(\rho,\pi'') \models \varphi \vee \psi$
$(\rho,\pi) \models \varphi \,\mathsf{S}\, \psi$ iff there is a position $\pi' \leq_\rho \pi$ such that $(\rho,\pi') \models \psi$ and forall π'' s.t. $\pi \geq_\rho \pi'' >_\rho \pi'$, $(\rho,\pi'') \models \varphi \vee \psi$
$(\rho,\pi) \models 4_{\bowtie a}\varphi$ iff either $(\rho,\pi) \models \varphi$ and $0 \bowtie a$ or, there is a position $\pi' >_\rho \pi$ such that $(\rho,\pi') \models \varphi$, $Dur\big(\rho_{[\pi,\pi']}\big) \bowtie a$ and forall π'' s.t. $\pi \leq_\rho \pi'' <_\rho \pi', (\rho,\pi'') \not\models \varphi$
$(\rho,\pi) \models 2_{\bowtie a}\varphi$ iff either $(\rho,\pi) \models \varphi$ and $0 \bowtie a$ or, there is a position $\pi' <_\rho \pi$ such that $(\rho,\pi') \models \varphi$, $Dur\big(\rho_{[\pi',\pi]}\big) \bowtie a$ and forall π'' s.t. $\pi \geq_\rho \pi'' >_\rho \pi', (\rho,\pi'') \not\models \varphi$

Given an ITA $\mathcal{A}$ and an SCL formula φ, $\mathcal{A} \models \varphi$ if for all executions ρ of $\mathcal{A}$, $(\rho,\pi_0) \models \varphi$, where $\pi_0 = 0$ is the initial position of ρ.

Theorem 3 *Model checking SCL over ITA is undecidable. Specifically, there exists a fixed formula using only modalities* U *and* $2_{=a}$ *such that checking its truth over ITA with* 3 *levels is undecidable.*

Proof We build an ITA and an SCL formula that together simulate a deterministic two counter machine. More specifically, we define a formula φ_{2cm} such that given a two counter machine $\mathcal{M}$, we can build an ITA $\mathcal{A}_\mathcal{M}$ with three clocks such that $\mathcal{A}_\mathcal{M} \models \varphi_{2cm}$ if and only if $\mathcal{M}$ does not halt.

Recall that such a machine $\mathcal{M}$ consists of a finite sequence of labeled instructions, which handle two counters c and d, and ends at a special instruction with label *Halt*. The other instructions have one of the two forms below, where $e \in \{c,d\}$ represents one of the two counters:

- $e := e+1$; goto ℓ'
- if $e > 0$ then ($e := e-1$; goto ℓ') else goto ℓ''

Without loss of generality, we may assume that the counters have initial value zero. The behavior of the machine is described by a (possibly infinite) sequence of configurations: $\langle \ell_0,0,0\rangle\langle \ell_1,n_1,p_1\rangle \ldots \langle \ell_i,n_i,p_i\rangle \ldots$, where n_i and p_i are the respective counter values and ℓ_i is the label, after the i^{th} instruction. The problem of termination for such a machine ("is the *Halt* label reached?") is known to be undecidable [32].

The idea of the encoding is that, provided the execution satisfies the formula, clocks of level 1 and 2 keep the values of c and d indifferently, by $x_i = \frac{1}{2^n}$ if n is the value of a counter e. Level 3 will be used as the working level. Transmitting the value of clocks to lower levels, prohibited in the ITA model, will be enforced by SCL formulas. In the sequel, we will define:

- a module $\mathcal{A}_\leftrightarrow$ and a formula $\varphi_\leftrightarrow$ such that the values contained in clocks x_1 and x_2 at the beginning of an execution ρ are swapped if and only if $(\rho,0) \models \varphi_\leftrightarrow$,
- a module $\mathcal{A}_+$ and a formula φ_+ such that if the value of x_2 is $\frac{1}{2^n}$ at the beginning of an execution ρ, then x_2 has value $\frac{1}{2^{n+1}}$ if and only if $(\rho,0) \models \varphi_+$,
- a module $\mathcal{A}_-$ and a formula φ_- such that if the value of x_2 is $\frac{1}{2^n}$ with $n > 0$ at the beginning of an execution ρ, then x_2 has value $\frac{1}{2^{n-1}}$ if and only if $(\rho,0) \models \varphi_-$.

Joining these modules according to $\mathcal{M}$ yields an ITA. Combining the formulas (independently of $\mathcal{M}$), we obtain an SCL formula that is satisfied if some execution, while complying to the formulas of the modules, reaches the final state. Both constructions are explained in details after the definitions below.

Let us define formulas $Span_1 = \mathsf{q}' \Rightarrow 2\ {}_{=1}\mathsf{q}$ and $Span_2 = \mathsf{p}' \Rightarrow 2\ {}_{=2}\mathsf{p}$ where p, p', q, q' are propositional variables. Let x_1^0 and x_2^0 denote the respective values of x_1 and x_2 upon entering a given module.

Swapping module. The module $\mathcal{A}_{\leftrightarrow}$ that swaps the values of x_1 and x_2 is depicted in Fig. 7. Note that this module does not actually swap the values of x_1 and x_2 for every execution. However, by imposing that state q_{end} is reached exactly 2 time units after q_0 (or q_0') was left, and that q_4 (resp. q_4') is reached exactly 1 t.u. after q_1 (resp. q_1') was left, the values of x_1 and x_2 will be swapped. This requirement can be expressed in SCL by $\varphi_{\leftrightarrow} = \mathsf{G}\ (Span_1 \wedge Span_2)$. Let w_i be the time elapsed in state q_i, for an execution ρ of $\mathcal{A}_{\leftrightarrow}$ that satisfies $\varphi_{\leftrightarrow}$. Note that q_{start} and $q_{end}^{\neq}$ are all urgent, hence no time can elapse in these states. We shall therefore consider only what happens in the swapping submodules. We detail only the case when $x_2 > x_1$, the case when $x_2 < x_1$ is analogous. The ITA constraints provide:

$$\begin{array}{ll} w_0 = 0 & (q_0 \text{ is urgent}) \\ w_1 = x_2^0 - x_1^0 & (\text{update } x_3 := x_1 \text{ and guard } x_3 = x_2) \\ w_2 = 1 - x_2^0 & (\text{guard } x_3 = 1) \\ w_4 = 0 & (q_4 \text{ is urgent}) \end{array}$$

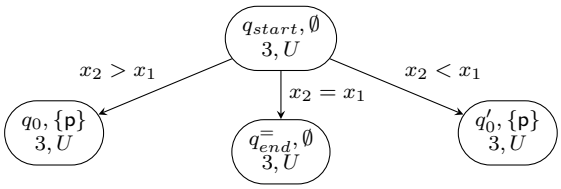
The time spent between the last instant q was satisfied (upon leaving q_1) and the only instant when q' is true (upon entering q_4) is exactly the time spent in states q_2 and q_3. Similarly, the time between the last instant p was satisfied (leaving q_0) and the instant p' is true (when reaching $q_{end}^{\neq}$) is the total amount of time spent in q_1, q_2, q_3, q_4, and q_5. Hence, if $\varphi_{\leftrightarrow}$ is satisfied then:

$$\begin{array}{rl} w_2 + w_3 = 1 & \left(\mathsf{q}' \Rightarrow 2\ {}_{=1}\mathsf{q}\right) \\ w_1 + w_2 + w_3 + w_4 + w_5 = 2 & \left(\mathsf{p}' \Rightarrow 2\ {}_{=2}\mathsf{p}\right) \end{array}$$

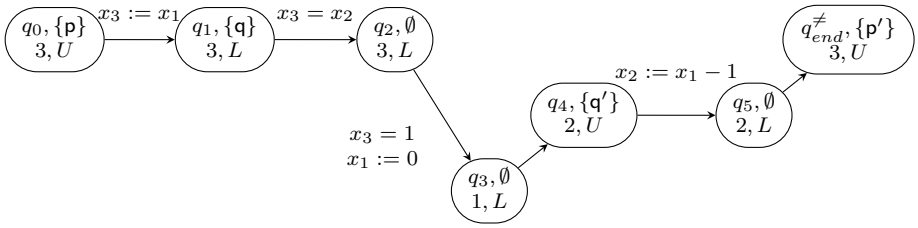
Hence $w_3 = x_2^0$ and $w_5 = 1 - w_1 = x_1^0 - \left(x_2^0 - 1\right)$. Since upon entering q_3, clock x_1 has value 0, when leaving, x_1 has value x_2^0. Similarly, when entering q_5, x_2 has value $x_1 - 1 = x_2^0 - 1$, therefore x_2 has value x_1^0 when reaching $q_{end}^{\neq}$. Note that this module swaps x_1 and x_2 regardless of their coding a counter value.

Incrementation module. The same idea applies for the incrementation module $\mathcal{A}_+$ of Fig. 8. We force the time spent in total in r_1 and r_2 is one, expressed in SCL by $\varphi_+ = \mathsf{G}\, Span_1$. The guards and updates in $\mathcal{A}_+$ ensure that, with the same notation as above, time spent in r_1 will be $1 - \frac{1}{2}x_2^0$. Hence, when reaching r_3, clock x_2 will have value $\frac{1}{2}x_2^0$. Therefore, if $x_2^0 = \frac{1}{2^n}$, coding a counter of value n, at the end of $\mathcal{A}_+$, x_2 has value $\frac{1}{2^{n+1}}$, thus coding a value $n+1$ for the same counter.

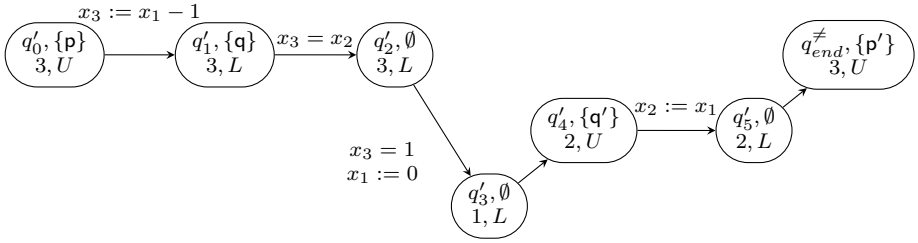
Decrementation module. Decrementation, for which the corresponding module is depicted on Fig. 9, is handled in a similar manner (with $\varphi_- = \varphi_+ = \mathsf{G}\, Span_1$). The only difference is that x_2 has to be compared to 1 in order to test if the value of the counter encoded by x_2 is 0.



(a) Choice submodule.



(b) Swapping submodule ($x_2 > x_1$).



(c) Swapping submodule ($x_2 < x_1$).

Fig. 7 Swapping module $\mathcal{A}_{\leftrightarrow}$. Submodules are connected through identical states (q_0, q'_0, $q^{\neq}_{end}$).

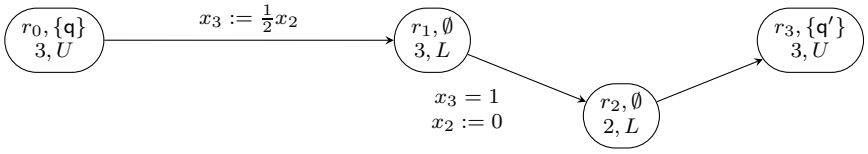


Fig. 8 Incrementation module.

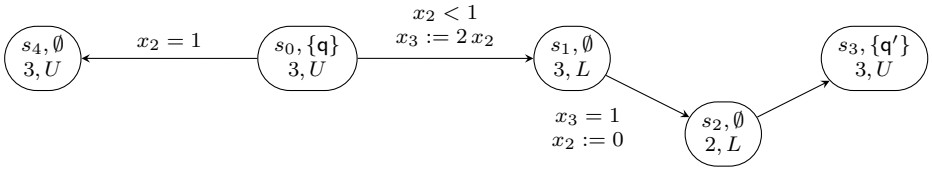


Fig. 9 Decrementation module.

Since the constraints in $Span_1$ (and $Span_2$) are equalities, they can be satisfied only if q' (and p') are true at a single point in time.

Automaton $\mathcal{A}_{\mathcal{M}}$ is then defined as the concatenation of modules according to $\mathcal{M}$. For clarity, a state (q, ℓ) denotes state q in a module corresponding to instruction ℓ.

Namely, an instruction ℓ incrementing c and going to ℓ' is an incrementation module with a transition from (r_3, ℓ) to the first state of the module corresponding to ℓ' (either (q_{start}, ℓ'), (r_0, ℓ') or (s_0, ℓ')). In the case of an incrementation of d, the corresponding module will be the concatenation of $\mathcal{A}^{in}_{\leftrightarrow}$, $\mathcal{A}_{+}$, and $\mathcal{A}^{out}_{\leftrightarrow}$. Modules $\mathcal{A}^{in}_{\leftrightarrow}$ and $\mathcal{A}^{out}_{\leftrightarrow}$ are two copies of a swapping module $\mathcal{A}_{\leftrightarrow}$. The states of $\mathcal{A}^{in}_{\leftrightarrow}$ and $\mathcal{A}^{out}_{\leftrightarrow}$ will be respectively denoted (q, ℓ, in) and (q, ℓ, out)) to avoid confusion. The last swap is performed in order to restore that x_2 contains the value of c and x_1 the value of d. The concatenation is done by transitions from $(q^{\neq}_{end}, \ell, in)$ and $(q^{=}_{end}, \ell, in)$ to (r_0, ℓ), from (r_3, ℓ) to (q_{start}, ℓ, out). States $(q^{\neq}_{end}, \ell, out)$ and $(q^{=}_{end}, \ell, out)$ are then linked to the first state of the module for ℓ'.

Decrementation is handled in a similar way. The main difference resides in the fact that (s_4, ℓ) is linked to the first state of ℓ''. In the decrementation of d, (s_4, ℓ) is linked to a swapping module $\mathcal{A}^{out'}_{\leftrightarrow}$ (disjoint from $\mathcal{A}^{in}_{\leftrightarrow}$ and $\mathcal{A}^{out}_{\leftrightarrow}$), in turn linked to the first state of ℓ''.

The *Halt* instruction is encoded in a single state h labeled with $\{\mathsf{h}\}$. The initial state of the automaton is a new state *Init* of level 3. It has *urgent* policy and satisfies no atomic proposition. State *Init* is linked to the first state of the module corresponding to ℓ_0, the initial instruction of $\mathcal{M}$, by a transition that updates both x_1 and x_2 to 1, simulating the initialization of both counters to 0.

Let us define formula $\varphi_{2cm} = \mathsf{F}(\neg Span_1 \vee \neg Span_2) \vee \mathsf{G}\neg\mathsf{h}$. An execution ρ of $\mathcal{A}_{\mathcal{M}}$ satisfies φ_{2cm} if either it violates at some point a constraint $Span_i$, which means ρ does not correspond to an execution of $\mathcal{M}$, or ρ never reaches state h, which means the execution of $\mathcal{M}$ is not halting.

If $\mathcal{M}$ has a halting execution, then it can be converted into an execution ρ that complies to the $Span_i$ constraints and reaches the final state h. Hence $\rho \not\models \varphi_{2cm}$ and $\mathcal{A}_{\mathcal{M}} \not\models \varphi_{2cm}$.

Conversely, if $\mathcal{A}_{\mathcal{M}} \not\models \varphi_{2cm}$, then consider an execution ρ that does not verify φ_{2cm}. Execution ρ both reaches h and complies to the $Span_i$ constraints, hence encodes a halting execution of $\mathcal{M}$.

As a result, $\mathcal{M}$ has no halting execution if and only if

$$\mathcal{A}_{\mathcal{M}} \models \mathsf{F}\left(\left(\neg\mathsf{q}' \wedge \neg 2_{=1}\mathsf{q}\right) \vee \left(\neg\mathsf{p}' \wedge \neg 2_{=2}\mathsf{p}\right)\right) \vee \mathsf{G}\neg\mathsf{h}.$$

Remark that this formula does not have nested history or prediction modalities ($2_{\bowtie a}$ and $4_{\bowtie a}$). Hence SCL with a discrete semantics (evaluating the subformulas only upon entering a state) would also be undecidable. □

5.2 Model-checking branching time properties with internal clocks

In this section we consider the extension of CTL with model clocks, the corresponding fragment being denoted by $\mathsf{TCTL}^{\text{int}}_c$. Such a logic allows to reason about the sojourn times in different levels which is quite useful when designing real-time operating systems. For example, formula $\mathsf{A}\,(x_2 \leq 3)\,\mathsf{U}\,\mathit{safe}$ expresses that all executions reach a safe

state while spending less than 3 time units in level 2 (assuming x_2 is not updated during the execution). Model-checking is achieved by adapting a class graph construction for untiming ITA (Section 3) and adding information relevant to the formula. The problem is thus reduced to a CTL model checking problem on this graph.

Definition 9 Formulas of the timed logic $\mathsf{TCTL}_c^{\text{int}}$ are defined by the following grammar:

$$\psi ::= p \mid \psi \wedge \psi \mid \neg\psi \mid \sum_{i\geq 1} a_i \cdot x_i + b \bowtie 0 \mid \mathsf{A}\,\psi\,\mathsf{U}\,\psi \mid \mathsf{E}\,\psi\,\mathsf{U}\,\psi$$

where $p \in AP$ is an atomic proposition, x_i are model clocks, a_i and b are rational numbers such that $(a_i)_{i\geq 1}$ has finite domain, and $\bowtie \in \{>, \geq, =, \leq, <\}$.

As before we use the classical shorthands F, G, and boolean operators.

Let $\mathcal{A} = \langle \Sigma, AP, Q, q_0, F, pol, X, \lambda, lab, \Delta\rangle$ be an interrupt timed automaton and $S = \{(q, v, \beta) \mid q \in Q,\ v \in \mathbb{R}^X,\ \beta \in \{\top, \bot\}\}$, the set of configurations. The formulas of $\mathsf{TCTL}_c^{\text{int}}$ are interpreted over configurations[1] $s = (q, v, \beta)$.

The semantics of $\mathsf{TCTL}_c^{\text{int}}$ is defined as follows on the transition system $\mathcal{T}_\mathcal{A}$ associated with $\mathcal{A}$. For atomic propositions and a configuration $s = (q, v, \beta)$, with $lab(s) = lab(q)$:

$$\begin{array}{ll} s \models p & \text{iff } \ p \in lab(s) \\ s \models \sum_{i\geq 1} a_i \cdot x_i + b \bowtie 0 & \text{iff } \ v \models \sum_{i\geq 1} a_i \cdot x_i + b \bowtie 0 \end{array}$$

and inductively:

$$\begin{array}{lll} s \models \varphi \wedge \psi & \text{iff} & s \models \varphi \text{ and } s \models \psi \\ s \models \neg\varphi & \text{iff} & s \not\models \varphi \\ s \models \mathsf{A}\,\varphi\,\mathsf{U}\,\psi & \text{iff} & \text{for all } \rho \in Exec(s),\ \rho \models \varphi\,\mathsf{U}\,\psi \\ s \models \mathsf{E}\,\varphi\,\mathsf{U}\,\psi & \text{iff} & \text{there exists } \rho \in Exec(s) \text{ s. t. } \rho \models \varphi\,\mathsf{U}\,\psi \\ \text{with } \rho \models \varphi\,\mathsf{U}\,\psi & \text{iff} & \text{there is a position } \pi \in \rho \text{ s. t. } s_\pi \models \psi \\ & & \text{and } \forall \pi' <_\rho \pi,\ s_{\pi'} \models \varphi \vee \psi. \end{array}$$

The automaton $\mathcal{A}$ satisfies ψ if the initial configuration s_0 of $\mathcal{T}_\mathcal{A}$ satisfies ψ.

Theorem 4 *Model checking TCTL_c^{int} on interrupt timed automata can be done in 2-EXPTIME, and in PTIME when the number of clocks is fixed.*

The proof relies on a refinement of the class graph according to the comparisons in the formula to model-check. It is detailed in Appendix A and we show the resulting graph on an example below.

Example. Consider the ITA $\mathcal{A}_1$ (Fig. 3(a)) and the formula $\varphi_1 = \mathsf{E}\,\mathsf{F}(q_1 \wedge (x_2 > x_1))$. We assume that q_1 is a propositional property true only in state q_1. Initially, the set of expressions are $E_1 = \{x_1, 0\}$ and $E_2 = \{x_2, 0\}$. First the expression $-\frac{1}{2}x_1 + 1$ is added into E_2 since $x_1 + 2x_2 = 2$ appears on the guard in the transition from q_1 to q_2. Then expression 1 is added to E_1 because $x_1 - 1 < 0$ appears on the guard in the transition from q_0 to q_1. Finally expression x_1 is added to E_2 since $x_2 - x_1 > 0$ appears in φ_1. The iterative part of the procedure goes as follows. Since there is a transition from q_0 of level 1 to state q_1 of level 2, we compute all differences between expressions of E_2, then normalize them:

[1] The boolean value in the configuration is not actually used. The logic could be enriched to take advantage of this boolean, to express for example that a run lets some time elapse in a given state.

$$Z_0^1 = (0 = x_1 < \frac{2}{3} < 1 < 2) \qquad Z_1^1 = (0 < x_1 < \frac{2}{3} < 1 < 2) \qquad Z_2^1 = (0 < x_1 = \frac{2}{3} < 1 < 2)$$

$$Z_3^1 = (0 < \frac{2}{3} < x_1 < 1 < 2) \qquad Z_4^1 = (0 < \frac{2}{3} < x_1 = 1 < 2) \qquad Z_5^1 = (0 < \frac{2}{3} < 1 < x_1 < 2)$$

$$Z_6^1 = (0 < \frac{2}{3} < 1 < x_1 = 2) \qquad Z_7^1 = (0 < \frac{2}{3} < 1 < 2 < x_1)$$

$$Z_0^2 = (0 = x_2 < x_1 < -\frac{1}{2}x_1 + 1) \qquad Z_1^2 = (0 < x_2 < x_1 < -\frac{1}{2}x_1 + 1)$$

$$Z_2^2 = (0 < x_1 = x_2 < -\frac{1}{2}x_1 + 1) \qquad Z_3^2 = (0 < x_1 < x_2 < -\frac{1}{2}x_1 + 1)$$

$$Z_4^2 = (0 < x_1 < -\frac{1}{2}x_1 + 1 = x_2) \qquad Z_5^2 = (0 < x_1 < -\frac{1}{2}x_1 + 1 < x_2)$$

$$Z_6^2 = (0 = x_2 < -\frac{1}{2}x_1 + 1 < x_1) \qquad Z_7^2 = (0 < x_2 < -\frac{1}{2}x_1 + 1 < x_1)$$

$$Z_8^2 = (0 < -\frac{1}{2}x_1 + 1 = x_2 < x_1) \qquad Z_9^2 = (0 < -\frac{1}{2}x_1 + 1 < x_2 < x_1)$$

$$Z_{10}^2 = (0 < -\frac{1}{2}x_1 + 1 < x_1 = x_2) \qquad Z_{11}^2 = (0 < -\frac{1}{2}x_1 + 1 < x_1 < x_2)$$

Table 2 Time zones used in the class graph of $\mathcal{A}_1$ when checking φ_1.

- $x_1 - 0$ and $x_2 - 0$ yield no new expression.
- $x_2 - (-\frac{1}{2}x_1 + 1)$ and $0 - (-\frac{1}{2}x_1 + 1)$ with update $x_2 := 0$ both yield expression 2, that is added to E_1.
- $x_1 - (-\frac{1}{2}x_1 + 1)$ yields expression $\frac{2}{3}$, which is also added to E_1.

The sets of expressions are therefore $E_1 = \{x_1, 0, 1, \frac{2}{3}, 2\}$ and $E_2 = \{x_2, 0, -\frac{1}{2}x_1 + 1, x_1\}$. Remark that knowing the order between x_1 and $\frac{2}{3}$ will allow us to know the order between $-\frac{1}{2}x_1 + 1$ and x_1. The class graph $\mathcal{G}$ corresponding to $\mathcal{A}_1$ and φ_1 is depicted in Fig. 10. Note that we replaced x_1 by its value, since it is not changed by any update at level 2. Some time zone notations used in $\mathcal{G}$ are displayed in Table 2. In the class graph, states where the comparison $x_2 > x_1$ is *true* are greyed. Among these, the ones in which the class corresponds to state q_1 are doubly circled, *i.e.* states in which $q_1 \wedge (x_2 > x_1)$ is *true*. Applying standard CTL model checking procedure on this graph, one can prove that one of these states is reachable, hence proving that φ_1 is *true* on $\mathcal{A}_1$.

5.3 Model-checking TCTL with subscript

Note that in $\mathsf{TCTL}_c^{\text{int}}$, it is not possible to reason about time evolution independently of the level in which actions are performed. For example, properties *(P2) the system is error free for at least 50 t.u.* or *(P3) the system will reach a safe state within 7 t.u.* involve global time. In order to verify such properties, we introduce the fragment TCTL_p. This fragment is expressive enough to state constraints on earliest (and latest) execution time of particular sequences, like those reaching a recovery state after a crash. TCTL_p is the set of formulas where satisfaction of an *until* modality over propositions can be parameterized by a restricted form of time intervals.

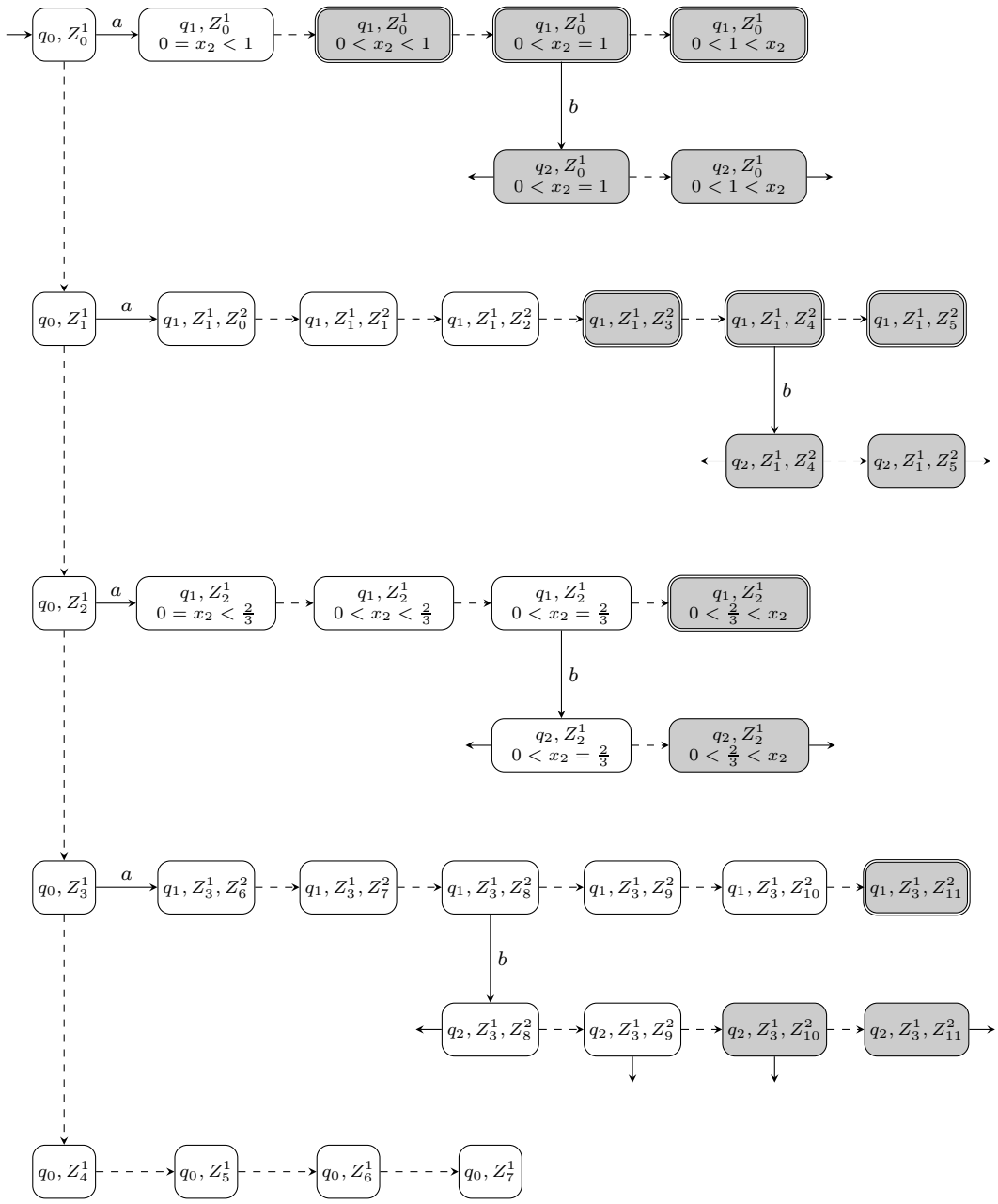


Fig. 10 The class automaton for $\mathcal{A}_1$ and formula φ_1.

Definition 10 Formulas of TCTL_p are defined by the following grammar:

$$\varphi_p := p \mid \varphi_p \wedge \varphi_p \mid \neg\varphi_p \quad \text{and} \quad \psi := \psi \wedge \psi \mid \neg\psi \mid \varphi_p \mid \mathsf{A}\,\varphi_p\,\mathsf{U}_{\bowtie \mathsf{a}}\,\varphi_p \mid \mathsf{E}\,\varphi_p\,\mathsf{U}_{\bowtie \mathsf{a}}\,\varphi_p$$

where $p \in AP$ is an atomic proposition, $a \in \mathbb{Q}^+$, and $\bowtie \in \{>, \geq, \leq, <\}$ is a comparison operator.

The properties given in introduction can be expressed by TCTL_p formulas as follows. Property $P2$: *the system is error free for at least 50 t.u.* corresponds to $\mathsf{A}\,(\neg error)\,\mathsf{U}_{\geq 50}\,\mathbf{t}$, while property $P3$: *the system will reach a safe state within 7 t.u.* is expressed by $\mathsf{A}\,\mathsf{F}_{\leq 7} safe$.

Formulas of TCTL_p are again interpreted over configurations of the transition system associated with an ITA. For configuration $s = (q, v, \beta)$, with $lab(s) = lab(q)$, the inductive definition is as follows:

$s \models p$ iff $p \in lab(s)$
$s \models \varphi \wedge \psi$ iff $s \models \varphi$ and $s \models \psi$
$s \models \neg\varphi$ iff $s \not\models \varphi$
$s \models \mathsf{A}\,\varphi_p\,\mathsf{U}_{\bowtie \mathsf{a}}\,\psi_p$ iff any execution $\rho \in Exec(s)$ is such that $\rho \models \varphi_p\,\mathsf{U}_{\bowtie \mathsf{a}}\,\psi_p$
$s \models \mathsf{E}\,\varphi_p\,\mathsf{U}_{\bowtie \mathsf{a}}\,\psi_p$ iff there exists an execution $\rho \in Exec(s)$ such that $\rho \models \varphi_p\,\mathsf{U}_{\bowtie \mathsf{a}}\,\psi_p$

where

$\rho \models \varphi_p\,\mathsf{U}_{\bowtie \mathsf{a}}\,\psi_p$ iff there exists a position π along ρ such that $Dur(\rho^{\leq \pi}) \bowtie a$, $s_\pi \models \psi_p$, and for any position $\pi' <_\rho \pi$, $s_{\pi'} \models \varphi_p$

Again $\mathcal{A} \models \psi$ if $s_0 \models \psi$.
We now prove that:

Theorem 5 *Model checking $TCTL_p$ on ITA is decidable.*

The proof consists in establishing procedures dedicated to the four different subcases:

- $\mathsf{E}\,p\,\mathsf{U}_{\leq \mathsf{a}}\,r$ and $\mathsf{E}\,p\,\mathsf{U}_{<\mathsf{a}}\,r$ (Proposition 6),
- $\mathsf{E}\,p\,\mathsf{U}_{\geq \mathsf{a}}\,r$ and $\mathsf{E}\,p\,\mathsf{U}_{>\mathsf{a}}\,r$ (Proposition 7),
- $\mathsf{A}\,p\,\mathsf{U}_{\geq \mathsf{a}}\,r$ and $\mathsf{A}\,p\,\mathsf{U}_{>\mathsf{a}}\,r$ (Proposition 8),
- $\mathsf{A}\,p\,\mathsf{U}_{\leq \mathsf{a}}\,r$ and $\mathsf{A}\,p\,\mathsf{U}_{<\mathsf{a}}\,r$ (Proposition 9),

where p and r are boolean combinations of atomic propositions.

Proposition 6 *Model checking formulas $\mathsf{E}\,p\,\mathsf{U}_{\leq \mathsf{a}}\,r$ and $\mathsf{E}\,p\,\mathsf{U}_{<\mathsf{a}}\,r$ over ITA is decidable in NEXPTIME and in NP if the number of clocks is fixed.*

Proof First consider the case of ITA$_-$. Both formulas are variants of reachability, with the addition of a time bound. Therefore, the proof is similar to the one of Proposition 5. Again using Lemma 2 on an ITA$_-$ with E transitions, we can look for a run satisfying one of these formulas and bounded by $B = (E + n)^{3n}$, because shortening longer runs can be can be done while preserving the property. Thus, the decision procedure again consists in guessing a path and building a linear program. The satisfaction of the formula is then checked by separately verifying on one side that the run satisfies $p\,\mathsf{U}\,r$, and on the other side, that the sum of all delays d_j satisfies the constraint in the formula. The complexity is the same as in Proposition 5.

In the case of ITA, the exponential blowup of the transformation into an equivalent ITA$_-$ does not affect the complexity of the model-checking procedure above, as in Theorem 2. □

Note that this problem can be compared with bounded reachability as studied in [17]. However, the models seem incomparable: while the variables (that have fixed non-negative rates in a state) are more powerful than interrupt clocks, the guards and updates are rectangular, which in particular forbids additive and diagonal constraints.

Proposition 7 *Model checking a formula* $\mathsf{E}\, p\, \mathsf{U}_{\geq \mathsf{a}}\, r$ *and* $\mathsf{E}\, p\, \mathsf{U}_{>\mathsf{a}}\, r$ *on an ITA is decidable in NEXPTIME and in NP if the number of clocks is fixed.*

Proof Let $\mathcal{A}$ be an ITA$_-$ with n interrupt clocks and E transitions, and $B = (E + n)^{3n}$. The algorithm to decide whether $\mathsf{E}\, p\, \mathsf{U}_{\geq \mathsf{a}}\, r$ (or $\mathsf{E}\, p\, \mathsf{U}_{>\mathsf{a}}\, r$) works as follows. It nondeterministically guesses a path of length smaller than or equal to B and builds the associated linear program (as in the proof of Proposition 5), then checks that:

- this path yields a run, which can be done by solving the linear program;
- there is a position π in this run at which r holds and before which p holds continuously;
- the sum of delays before π exceeds a (or strictly exceed in the case of $\mathsf{E}\, p\, \mathsf{U}_{>\mathsf{a}}\, r$).

If this first procedure fails, the algorithm nondeterministically guesses a path of length smaller or equal to $2B + 1$ and checks that:

- this path yields a run, which can be checked by a linear program as before,
- p holds on this path, but not necessarily in the last state reached,
- r holds in the last state of this path,
- either there is a transition e of level k that updates x_k appearing twice and separated by a sequence σ of transitions of level higher than k during which time elapses (globally) ; this last part can be checked with a linear program on the delays corresponding to this subrun.
- or there is a transition e of level k that does not update x_k appearing twice and separated by a sequence σ of transitions of level higher than k not updating x_k during which time elapses at levels strictly higher than k but not at level k.

The algorithm returns *true* if one of the previous procedure succeeds, and *false* otherwise. We shall now prove that this algorithm is both sound and complete.

Soundness. If the first procedure succeeds, then the path guessed is trivially a witness of $\mathsf{E}\, p\, \mathsf{U}_{\geq \mathsf{a}}\, r$ (or $\mathsf{E}\, p\, \mathsf{U}_{>\mathsf{a}}\, r$, accordingly). If the second procedure succeeds, then a witness for the formula can be built from the path guessed. Indeed, the path guessed satisfies $p\, \mathsf{U}\, r$, but not necessarily $p\, \mathsf{U}_{\geq \mathsf{a}}\, r$. Assume the sequence σ lets elapse δ time units ($\delta > 0$), by repeating $\lceil \frac{a}{\delta} \rceil$ times[2] the sequence σe, we obtain a run satisfying $p\, \mathsf{U}_{\geq \mathsf{a}}\, r$. Note that since either e updates the clock x_k or there are no updates nor time elapsing at level k, and σ happens at higher levels, the clock values in each instance of σe will be identical, hence this repetition will always be possible.

Completeness. Now consider a minimal witness ρ of length h for $\mathsf{E}\, p\, \mathsf{U}_{\geq \mathsf{a}}\, r$. Since ρ is minimal, r holds in the last state of ρ and p holds (at least) in every position before. If $h \leq B$, then the first procedure will consider ρ. Otherwise, $h > B$, it means that one of the following cases of Lemma 2 happens:

[2] This sequence may be repeated once more in the case of $p\, \mathsf{U}_{>\mathsf{a}}\, r$.

- The same transition e of level k leaving x_k unchanged appears twice separated by lazy or delayed transitions between states of level greater than or equal to k. In that case, the corresponding subrun can be replaced by a time step of the same duration, not changing the truth value of $p\,\mathsf{U}_{\geq \mathsf{a}}\,r$ on this new smaller run, thus violating the minimality hypothesis.
- The same transition e of level k updating clock x_k appears twice on the subrun $e_1 \dots e_{B+1}$, at positions i and j. In that case we have to distinguish two subcases either some time has elapsed between the two occurrences e_i and e_j of e, or the transitions were all instantaneous.
  - If no time has elapsed, the subrun between e_i and e_j can be removed without altering the truth value of $p\,\mathsf{U}_{\geq \mathsf{a}}\,r$ on this new run, which is smaller than ρ. Hence there is a contradiction with the minimality hypothesis.
  - Or some time elapsed during this subrun. Let ρ be decomposed into $\rho_0 e_i \sigma e_j \rho_j$. Then by applying Lemma 2 to ρ_j there exists a run ρ'_j of length smaller or equal to B such that $\rho' = \rho_0 e_i \sigma e_j \rho'_j$ is also a run. Note that $|\rho'| \leq 2B + 1$, that the last state of ρ' will be the same as the last state of ρ hence will satisfy r, and that p will also hold along ρ'. As a result ρ' will be considered by the second procedure.
- The same transition e of level k leaving x_k unchanged appears twice, with no time elapsing at level k between these occurrences. In that case, we again distinguish two subcases:
  - either no time elapsed (globally) the corresponding subrun can be removed, not changing anything to the rest of the execution nor to the satisfaction of $p\,\mathsf{U}_{\geq \mathsf{a}}\,r$, thus violating the hypothesis of minimality of ρ;
  - or time elapsed at higher levels and, by minimizing the subrun after the second occurrence as above, we deduce that the run will be considered by the second procedure.

The completeness proof is similar in the case of $\mathsf{E}\,p\,\mathsf{U}_{>\mathsf{a}}\,r$.

When $\mathcal{A}$ is an ITA, the exponential blowup of the transformation from ITA to ITA$_-$ does not affect the above complexity. □

While a witness is a finite path in the previous cases, it is potentially infinite for $\mathsf{A}\,p\,\mathsf{U}_{\geq \mathsf{a}}\,r$ or $\mathsf{A}\,p\,\mathsf{U}_{>\mathsf{a}}\,r$. The generation of an infinite run relies on the (nondeterministic) exploration of the class graph built in Section 3, thus has a much greater computational complexity.

Proposition 8 *Model checking a formula $\mathsf{A}\,p\,\mathsf{U}_{\geq \mathsf{a}}\,r$ and $\mathsf{A}\,p\,\mathsf{U}_{>\mathsf{a}}\,r$ on an ITA is decidable in 2-EXPTIME and in co-NP if the number of clocks is fixed.*

Proof We consider an ITA $\mathcal{A}$ with n interrupt clocks, E transitions and the bound $B = (n+2)^{12b \cdot E \cdot n^3}$ where b is the number of bits coding the constants in $\mathcal{A}$.

The algorithm to verify $\mathsf{A}\,p\,\mathsf{U}_{\geq \mathsf{a}}\,r$ (or $\mathsf{A}\,p\,\mathsf{U}_{>\mathsf{a}}\,r$) works as follows. It nondeterministically guesses a path of length smaller than or equal to B, builds its associated linear program, and checks that:

- this path yields a run ρ (by solving the linear program);
- this path is maximal, that means no transition can be fired from the last configuration of the run;

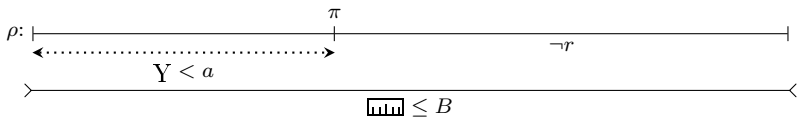


Fig. 11 Proof of Proposition 8: finite counterexample (Case 1).

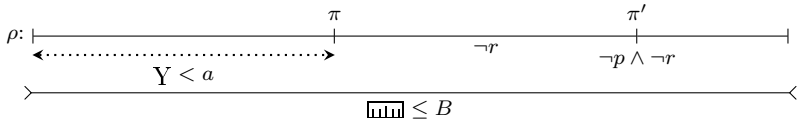


Fig. 12 Proof of Proposition 8: finite counterexample (Case 2).

- there is a position π in ρ occurring at a time stricly less than[3] a such that
  Case 1: either r does not hold from π (see Fig. 11)
  Case 2: or there is a position π' where neither p not r hold, and r does not hold between π and π' (see Fig. 12).

If this first procedure fails, then the algorithm guesses:

- a class K and a cycle $\mathcal{C}$ starting from K in the class graph (without building neither the graph nor the cycle), such that $\mathcal{C}$ contains at least a discrete step and only traverses classes where $\neg r$ holds;
- a path in the automaton of length smaller than or equal to the bound B;

and checks that:

- the path does yield a run ρ, that reaches a configuration (q, v, β) in class K (through a linear program);
- there is a position π in ρ occurring at time strictly less than[4] a after which r no longer holds.

Remark that the procedure cannot use solely the class graph, since the abstraction is not precise enough to check the existence of position π.

Soundness. We prove that the algorithm is sound: when one of the procedures succeeds, there exists a counterexample for formula $\mathsf{A}\, p\, \mathsf{U}_{\geq \mathsf{a}}\, r$ (or $\mathsf{A}\, p\, \mathsf{U}_{>\mathsf{a}}\, r$). In the case of the first procedure, it is trivial that the guessed run does not satisfy $p\, \mathsf{U}_{\geq \mathsf{a}}\, r$ (or $p\, \mathsf{U}_{>\mathsf{a}}\, r$). In the case of the second one, we show that there exists an infinite counterexample. Consider configuration (q, v, β), which is reachable by ρ. Since (q, v, β) belongs to class K, for any path σ starting from K in the class graph, there is a run in the automaton starting from (q, v, β) traversing configurations which belong to the classes traversed by σ. Since there is a cycle in the class graph, there is an infinite path in the class graph (iterating on this cycle), so there exists an infinite run in the ITA. Also, since $\neg r$ holds in the infinite path of the class graph, it holds in the run of the ITA, and the run is a counterexample for the formula.

[3] Less than or equal to a in the case of $\mathsf{A}\, p\, \mathsf{U}_{>\mathsf{a}}\, r$.
[4] Less than or equal to a in the case of $\mathsf{A}\, p\, \mathsf{U}_{>\mathsf{a}}\, r$.

Completeness. Assume there exists a finite counterexample ρ. Let $\mathcal{A}'$ be the ITA$_-$ accepting the same timed language as $\mathcal{A}$ and let E' denote the number of its transitions. Let $B' = (E' + 2n)^{3n}$ (the bound of Lemma 2), we have $B' \leq B$. If $|\rho| \leq B$, it will be detected by procedure 1. Otherwise let ρ' be the run corresponding to ρ in $\mathcal{A}'$. This run accepts the same timed word as ρ and its sequence of traversed states can be projected onto the sequence of corresponding states of ρ, by omitting states of the form $(q^-, -)$: any subsequence $(q_0^+, -) \to \cdots \to (q_{m-1}^+, -) \to (q_m^-, -) \to (q_m^+, -)$ in ρ' corresponds to the subsequence $q_0 \to \cdots \to q_{m-1} \to q_m$ in ρ. Note that $|\rho| \leq |\rho'|$ and that ρ' is also a counterexample for the formula (although in $\mathcal{A}'$). Since $|\rho'| > B \geq B'$, then one of the cases of case of Lemma 2 occurs. By removing transitions and maybe replacing them by some time elapsing, as in the proof of Proposition 7, a counterexample σ' of size $|\sigma'| \leq B' \leq B$ exists in $\mathcal{A}'$. Now consider the run σ in $\mathcal{A}$ which corresponds to σ'. We have $|\sigma| \leq |\sigma'| \leq B' \leq B$ and σ is still a counterexample. Therefore σ can be guessed by the first procedure.

If there exists an infinite counterexample ρ, consider its counterpart σ in the class graph. This counterpart is also infinite. More precisely, σ contains an infinite number of discrete transitions. Since σ traverses a finite number of classes, it contains a cycle $\mathcal{C}$ with at least one discrete transition. Choose any class K of this cycle and consider the prefix ρ_0 of ρ leading to a configuration in K. As in the case of a finite counterexample, there exists ρ_0' of length smaller than B reaching the same configuration. All $\mathcal{C}$, K and ρ_0' can be guessed by the second procedure, which will therefore succeed.

Procedure 1 operates in NEXPTIME (guessing a path of length B and solving a linear program of size polynomial w.r.t. B). Procedure 2 consists in looking for a specific cycle in the class graph which in can be done in time polynomial w.r.t. the size of the graph thus in 2-EXPTIME. The case where the clocks are fixed, is handled as usual. □

For formulas in case 4, a specific procedure can be avoided, since the algorithms of cases 2 and 3 can be reused:

Proposition 9 *Model checking a formula* $\mathsf{A}\, p\, \mathsf{U}_{\leq \mathsf{a}}\, r$ *and* $\mathsf{A}\, p\, \mathsf{U}_{<\mathsf{a}}\, r$ *on an ITA is decidable in 2-EXPTIME and in co-NP if the number of clocks is fixed.*

Proof Notice that $\mathsf{A}\, p\, \mathsf{U}_{\leq \mathsf{a}}\, r = (\mathsf{A}\, p\, \mathsf{U}_{\geq 0}\, r) \wedge \neg(\mathsf{E}\, \neg r\, \mathsf{U}_{>\mathsf{a}}\, \mathbf{t})$, and $\mathsf{A}\, p\, \mathsf{U}_{<\mathsf{a}}\, r = (\mathsf{A}\, p\, \mathsf{U}_{\geq 0}\, r) \wedge \neg(\mathsf{E}\, \neg r\, \mathsf{U}_{\geq \mathsf{a}}\, \mathbf{t})$. □

6 Language properties

In this section, we compare the expressive power of the previous models with respect to language acceptance. Recall that TL is strictly contained in CRTL. We prove that:

Theorem 6 *The families TL and ITL are incomparable. The families CRTL and ITL are incomparable.*

6.1 ITL is not contained in TL, nor in CRTL

The next proposition shows that ITA cannot be reduced to TA or CRTA. Observe that the automata used in the proof belong to ITA$_-$. Also, the language given for the first point of the proposition is very simple since it contains only words of length 2.

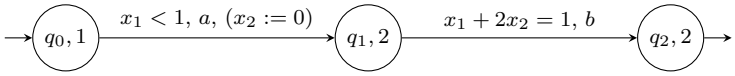


Fig. 13 An ITA $\mathcal{A}_3$ for L_3

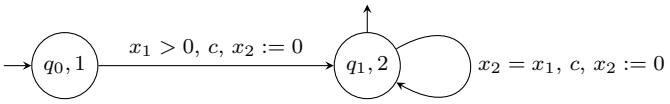


Fig. 14 An ITA $\mathcal{A}_4$ for L_4

Proposition 10
1. There exists a language in ITL whose words have bounded length which is not in TL.
2. There exists a language in ITL which is not in CRTL.

Proof To prove the first point, consider the ITA $\mathcal{A}_3$ in Fig. 13. Suppose, by contradiction, that $L_3 = \mathcal{L}(\mathcal{A}_3)$ is accepted by some timed automaton $\mathcal{B}$ (possibly with ε-transitions). Note that since we consider timed languages, we cannot assume that the granularity of $\mathcal{B}$ is 1. Let d be the granularity of $\mathcal{B}$, *i.e.* the gcd of all rational constants appearing in the constraints of $\mathcal{B}$ (thus each such constant can be written k/d for some integer k). Then the word $w = (a, 1 - 1/d)(b, 1 - 1/2d)$ is accepted by $\mathcal{B}$ through a finite path. Consider now the automaton $\mathcal{B}'$ in TA, consisting of this single path (where states may have been renamed). We have $w \in \mathcal{L}(\mathcal{B}') \subseteq \mathcal{L}(\mathcal{B}) = L_3$ and $\mathcal{B}'$ contains no cycle. Using the result in [11], we can build a timed automaton $\mathcal{B}''$ without ε-transition and with same granularity d such that $\mathcal{L}(\mathcal{B}'') = \mathcal{L}(\mathcal{B}')$, so that $w \in \mathcal{L}(\mathcal{B}'')$. The accepting path for w in $\mathcal{B}''$ contains two transitions : $p_0 \xrightarrow{\varphi_1, a, r_1} p_1 \xrightarrow{\varphi_2, b, r_2} p_2$. After firing the a-transition, all clock values are $1 - 1/d$ or 0, thus all clock values are $1 - 1/2d$ or $1/2d$ when the b-transition is fired. Let $x \bowtie c$ be an atomic proposition appearing in φ_2. Since the granularity of $\mathcal{B}''$ is d, the $\bowtie$ operator cannot be $=$ otherwise the constraint would be $x = 1/2d$ or $x = 1 - 1/2d$. If the constraint is $x < c$, $x \leq c$, $x > c$, or $x \geq c$, the path will also accept some word $(a, 1 - 1/d)(b, t)$ for some $t \neq 1 - 1/2d$. This is also the case if the constraint φ_2 is true. We thus obtain a contradiction with $\mathcal{L}(\mathcal{B}'') \subseteq L_3$, which ends the proof.

To prove the second point, consider the language:

$$L_4 = \{(c, \tau)(c, 2\tau) \dots (c, n\tau) \mid n \in \mathbb{N}, \tau > 0\}$$

accepted by the ITA $\mathcal{A}_4$ in Fig. 14. This language cannot be accepted by a CRTA (see [21]). □

6.2 TL is not contained in ITL

We now prove that there exists a language in TL that does not belong to ITL. Let L_5 be the language defined by

$$\begin{aligned} L_5 = \big\{ (a, \tau_1)(b, \tau_2) \dots (a, \tau_{2p+1})(b, \tau_{2p+2}) \mid p \in \mathbb{N}, \\ \forall 0 \leq i \leq p,\ \tau_{2i+1} = i + 1 \text{ and } i + 1 < \tau_{2i+2} < i + 2, \\ \forall 1 \leq i \leq p\ \tau_{2i+2} - \tau_{2i+1} < \tau_{2i} - \tau_{2i-1} \big\} \end{aligned}$$

Hence, the untimed language of L_5 is $(ab)^*$, there is an occurrence of a at each time unit and the successive occurrences of b come each time closer to the occurrence of a than previously. This language is in TL as can be checked on the TA $\mathcal{A}_5$ of Fig. 15 (first proposed in [4]).

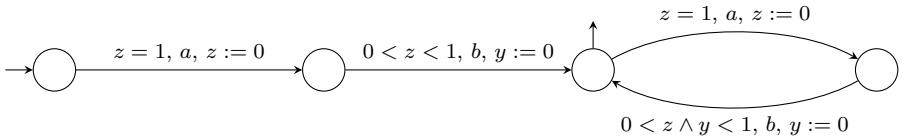


Fig. 15 A timed automaton $\mathcal{A}_5$ for L_5

Proposition 11 *The language L_5 does not belong to ITL.*

Proof Assume, by contradiction, that L_5 belongs to ITL. Then L_5 is accepted by an ITA$_-$ $\mathcal{A}$ with n clocks and E transitions. Let $B = (E+n)^{3n}$ and consider the timed word $w = (a,\tau_1)(b,\tau_2)\cdots(a,\tau_{2B+1})(b,\tau_{2B+2}) \in L_5$. Word w is accepted by a run ρ of $\mathcal{A}$, which can be assumed of minimal size. However, we know that $|\rho| > B$, so one of the three cases of Lemma 2 occurs in the B first transitions.

- Suppose a transition e of level k that updates x_k appears twice, separated by a subrun σ of level greater than or equal to k. Remark that the valuations after the first and the second occurrence of e are identical. We distinguish several subcases, depending on the word read along σe.
  - If σe reads the empty word ε, we write δ for the time spent during σe. If $\delta = 0$, then σe can be deleted without affecting neither the remainder of the run nor the accepted word, which contradicts the minimality of ρ. If $\delta \geq 1$, then some interval $[i, i+1]$ does not contain any b, which contradicts the definition of L_5. Otherwise, $0 < \delta < 1$. By deleting σe, we obtain an execution ρ' (accepted by $\mathcal{A}$) in which the suffix after e is shifted by δ. Therefore the following occurrence of letter a, which appeared in ρ at date $i \in \mathbb{N} \setminus \{0\}$, appears in ρ' at date $i - \delta$ which is not integral. So the word accepted by ρ' is not in L_5, which is a contradiction.
  - If σe reads more as than bs or more bs than as, by deleting σe we obtain a run accepting a word whose untiming is not in $(ab)^*$ thus does not belong to L_5.
  - If σe reads as many as as bs (and both letters at least once), by duplicating σe we obtain a run accepting a word where a same duration separates an a from the following b is repeated, thus violating the definition of L_5.
- Suppose a transition e of level k delayed or lazy occurs twice, separated by a subrun σ of level greater than or equal to k, such that σe does not update x_k. Then we can replace $e\sigma$ by a time step of the same duration and obtain a new run ρ', accepted by $\mathcal{A}$.
  - If $e\sigma$ reads ε, then ρ' contradicts the minimality of ρ.
  - If $e\sigma$ reads the word b, then ρ' accepts a word where a and b do not alternate, thus not in L_5.
  - If $e\sigma$ reads at least an a, then ρ' accepts a word with no a at a given integral date, therefore not in L_5.

– Otherwise, a transition e of level k appears twice separated by a subrun σ of level greater than or equal to k, such that σe does not update x_k nor lets time elapse at level k. The same disjunction as in the case of an update of x_k can be applied, since σe can either be deleted or duplicated.

Note that the feature preventing L_5 to be in ITL lies in the decreasing delays between the a's and their immediately following b. A language in ITL can record k different constant delays, using $k+1$ clocks. For instance on the alphabet $\Sigma = \{a_1, \ldots, a_k\}$, the language

$$M_k = \{(a_1, \tau_1) \ldots (a_k, \tau_k)(a_1, \tau_1 + 1) \ldots (a_k, \tau_k + 1) \ldots (a_1, \tau_1 + n) \ldots (a_k, \tau_k + n) \mid n \geq 1, \tau_1 \leq \tau_2 \leq \cdots \leq \tau_k \leq \tau_1 + 1\}$$

is accepted by an ITA$_-$ with $k+1$ clocks. Fig. 16 illustrates the case where $k = 3$, with all states lazy. We conjecture that M_k cannot be accepted by an ITA with k clocks.

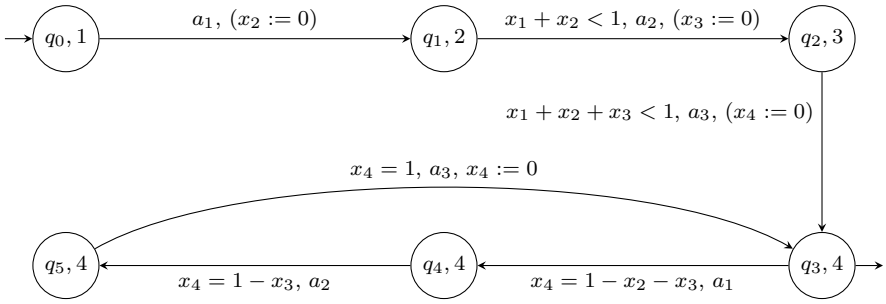


Fig. 16 An interrupt timed automaton for M_3

6.3 Closure under complementation and intersection

Proposition 12 *ITL is not closed under complementation.*

Proof We prove that the complement L_5^c of L_5 belongs to ITL. A timed word belongs to L_5^c iff one of the following assertions hold:

1. An a occurs not at a time unit.
2. An a is missing at some time unit that precedes some letter of the word.
3. A b occurs at a time unit.
4. There is no b in an interval $[i, i+1]$ with an a at time $i \in \mathbb{N}$.
5. There are two bs in an interval $[i, i+1]$ with an a at time $i \in \mathbb{N}$.
6. There is an occurrence of $abab$ such that the time difference between the two first occurrences is smaller than or equal to the time difference between the two last occurrences.

Since ITL is trivially closed under union, it is enough to prove that each assertion from the set above can be expressed by an ITA. The five first assertions are straightforwardly modeled by an ITA with a single clock (and ε-transitions) and we present in Fig. 17 an ITA with two clocks corresponding to the last one. □

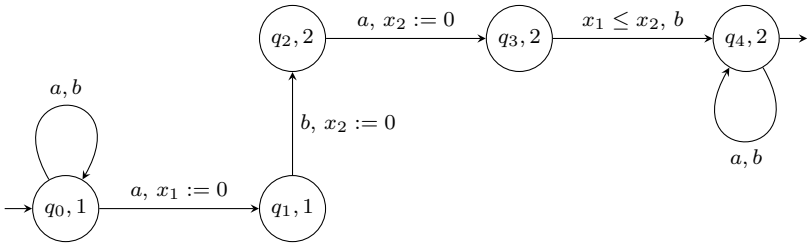


Fig. 17 An ITA for the language defined by assertion 6

Proposition 13 *ITL is not closed under intersection.*

Proof L_5 is the intersection of L'_5 and L''_5 defined as follows:

- The words of L'_5 are $(a,1)(b,\tau_1)\dots(a,n)(b,\tau_n)$, with $i < \tau_i < i+1$ for all i, $1 \leq i \leq n$.
- The words of L''_5 are $(a,\tau'_1)(b,\tau_1)\dots(a,\tau'_n)(b,\tau_n)$, with $\tau_{i+1} - \tau_i < 1$ for all i, $1 \leq i \leq n-1$.

Both languages are accepted by one-clock ITA (which are also one-clock TA). In case of L'_5, (1) the clock is reset at every occurrence of an a; (2) an a must occur when the clock is 1 and (3) a single b must occur when the clock is in $(0,1)$. In case of L''_5, (1) the clock is reset at every occurrence of a b (2) a b must occur when the clock is less than 1 except for the first b and (3) a single a must occur before every occurrence of a b. □

7 Combining ITA with CRTA

In the previous section, we proved that the class of languages defined by ITA and CRTA are incomparable. Here we provide a class containing both ITL and CRTL. In order to do so, we combine the models of ITA with CRTA.

7.1 An undecidable product

The first kind of combination possible is through synchronized product between an ITA and a CRTA. However, this turns out to be a too powerful model, since combining even a TA with an ITA yields the undecidability of the reachability problem.

Definition 11 If $\mathcal{I} = \langle \Sigma, Q_\mathcal{I}, q_0^\mathcal{I}, F_\mathcal{I}, pol_\mathcal{I}, X, \lambda_\mathcal{I}, \Delta_\mathcal{I} \rangle$ is an ITA (propositional variables and labeling are omitted) and $\mathcal{T} = \langle \Sigma, Q_\mathcal{T}, q_0^\mathcal{T}, F_\mathcal{T}, Y, \Delta_\mathcal{T} \rangle$ is a TA, then $\mathcal{I} \times \mathcal{T} = \langle \Sigma, Q_\mathcal{I} \times Q_\mathcal{T}, (q_0^\mathcal{I}, q_0^\mathcal{T}), F, pol, X, Y, \lambda, \Delta \rangle$ is an ITA×TA where:

- $pol(q_\mathcal{I}, q_\mathcal{T}) = pol_\mathcal{I}(q_\mathcal{I})$ and $\lambda(q_\mathcal{I}, q_\mathcal{T}) = \lambda_\mathcal{I}(q_\mathcal{I})$ are lifted from the ITA
- if $q_\mathcal{I} \xrightarrow{\varphi,a,u} q'_\mathcal{I} \in \Delta_\mathcal{I}$ and $q_\mathcal{T} \xrightarrow{\psi,a,v} q'_\mathcal{T} \in \Delta_\mathcal{T}$, then

$$(q_\mathcal{I}, q_\mathcal{T}) \xrightarrow{\varphi \wedge \psi, a, u \wedge v} (q'_\mathcal{I}, q'_\mathcal{T}) \in \Delta.$$

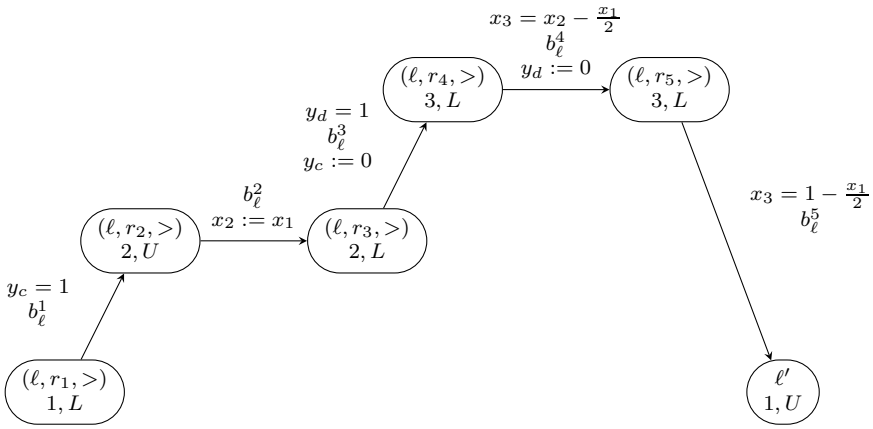


Fig. 18 Module $\mathcal{A}^{c++}_{c\geq d}(\ell, \ell')$ incrementing the value of c when $c \geq d$.

The semantics of an ITA×TA is a transition system over configurations

$$\left\{(q, v, w, \beta) \mid q \in Q, v \in \mathbb{R}^X, w \in \mathbb{R}^Y, \beta \in \{\top, \bot\}\right\}.$$

Discrete steps are defined analogously as in ITA (see Definition 2). In time steps, clocks of X evolve as in an ITA and clocks of Y as in a TA. More precisely, a time step of duration $d > 0$ is defined by $(q, v, w, \beta) \xrightarrow{d} (q, v', w', \top)$ where $v'(x_{\lambda(q)}) = v(x_{\lambda(q)}) + d$ and $v'(x) = v(x)$ for any other clock $x \in X$, and $w'(y) = w(y) + d$ for $y \in Y$.

Theorem 7 *Reachability is undecidable in the class ITA× TA.*

Proof (Sketch) The proof consists in encoding a two counter machine into an ITA×TA. Two classical clocks $\{y_c, y_d\}$ will keep the value of the counters by retaining a value $1 - \frac{1}{2^n}$ to encode n. Three interrupt clocks are used to change the value of the classical clocks through appropriate resets. The ITA×TA is defined through basic modules, corresponding to the four possible actions (incrementation or decrementation of c or d). Each module is itself composed of submodules: the first one compares the value of c to the one of d. The other one performs the action, but depends on the order between c and d.

For example, the submodule incrementing c when $c \geq d$ is depicted in Fig. 18. In this module, the value[5] of classical clocks is copied into interrupt clocks, updated thanks to linear updates allowed by ITA. the new values are copied into classical clocks by resetting them at the appropriate moment. The valuations of clocks during an execution of this module are given in Table 3.

Note that the policies are used in this product but they could be replaced by classical clocks.

The detailed proof can be found in Appendix B.

Other proofs of undecidability for hybrid systems mixing clocks and stopwatches have been developed (see for instance [24, Theorem 4.1] for a construction with a single stopwatch and 5 clocks). While this construction could have been adapted to

[5] Or rather the complement to 1 of the value.

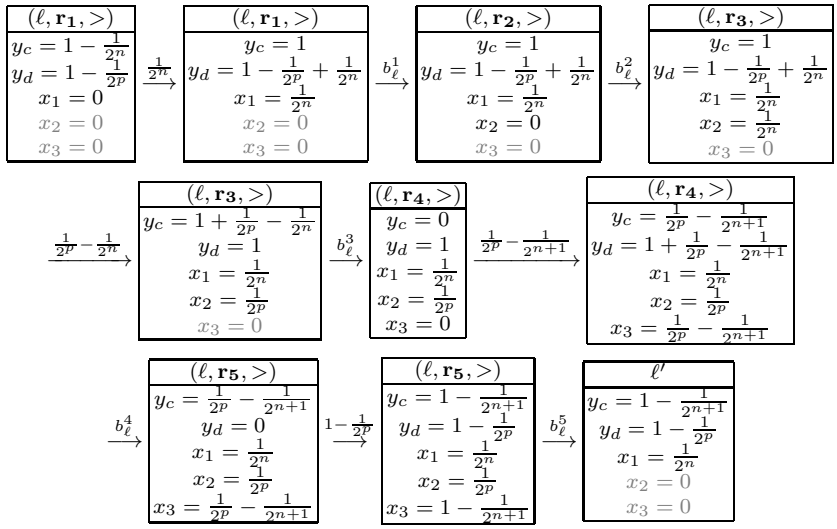


Table 3 Clock values in the unique run of $\mathcal{A}^{c++}_{c\geq d}(\ell, \ell')$. Irrelevant values of interrupt clocks are greyed.

our setting, this would have led to an ITA×TA with 5 classical clocks and 2 interrupt clocks.

7.2 A decidable product of ITA and CRTA: ITA$^+$

We define another synchronized product between ITA and CRTA, in the spirit of multi-level systems, for which reachability is decidable. This class, denoted by ITA$^+$, includes a set of clocks at an implicit additional level 0, corresponding to a basic task described as in a CRTA. In the definition below, since no confusion can occur, we aggregate the coloring function of CRTA and the level function of ITA, into a single function λ.

Definition 12 (ITA$^+$) An *extended interrupt timed automaton* is a tuple $\mathcal{A} = \langle Q, q_0, F, pol, X \uplus Y, \Sigma, \Omega, \lambda, up, low, vel, \Delta\rangle$, where:

- Q is a finite set of states, q_0 is the initial state and $F \subseteq Q$ is the set of final states.
- $pol : Q \mapsto \{L, U, D\}$ is the timing policy of states.
- $X = \{x_1, \ldots, x_n\}$ consists of n interrupt clocks and Y is a set of basic clocks,
- Σ is a finite alphabet,
- Ω is a set of colors, the mapping $\lambda : Q \uplus Y \mapsto \{1, \ldots, n\} \uplus \Omega$ associates with each state its level or its color, with $x_{\lambda(q)}$ the active clock in state q for $\lambda(q) \in \mathbb{N}$ and $\lambda(y) \in \Omega$ for $y \in Y$. For every state $q \in \lambda^{-1}(\Omega)$, the policy is $pol(q) = L$.
- up and low are mappings from Y to $\mathbb{Q}$ with the same constraints as CRTA (see Definition 4), and $vel : Q \mapsto \mathbb{Q}$ is the clock rate with $\lambda(q) \notin \Omega \Rightarrow vel(q) = 1$
- $\Delta \subseteq Q \times [\mathcal{C}(X \cup Y) \times (\Sigma \cup \{\varepsilon\}) \times \mathcal{U}(X \cup Y)] \times Q$ is the set of transitions. Let $q \xrightarrow{\varphi, a, u} q'$ in Δ be a transition.
  1. The guard φ is of the form $\varphi_1 \wedge \varphi_2$ with the following conditions. If $\lambda(q) \in \mathbb{N}$, φ_1 is an ITA guard on X and otherwise $\varphi_1 = true$. Constraint φ_2 is a CRTA guard on Y (also possibly equal to $true$).

2. The update u is of the form $u_1 \wedge u_2$ fullfilling the following conditions. Assignments from u_1 update the clocks in X with the constraints of ITA when $\lambda(q)$ and $\lambda(q')$ belong to $\mathbb{N}$. Otherwise it is a global reset of clocks in X. Assignments from u_2 update clocks from Y, like in CRTA.

Any ITA can be viewed as an ITA$^+$ with Y empty and $\lambda(Q) \subseteq \{1, \ldots, n\}$, and any CRTA can be viewed as an ITA$^+$ with X empty and $\lambda(Q) \subseteq \Omega$. Class ITA$^+$ combines both models in the following sense. When the current state q is such that $\lambda(q) \in \Omega$, the ITA part is inactive. Otherwise, it behaves as an ITA but with additional constraints about clocks of the CRTA involved by the extended guards and updates. The semantics of ITA$^+$ is defined as usual but now takes into account the velocity of CRTA clocks.

Definition 13 (Semantics of ITA$^+$) The semantics of an automaton $\mathcal{A}$ in ITA$^+$ is defined by the transition system $\mathcal{T}_\mathcal{A} = (S, s_0, \to)$. The set S of configurations is $\Big\{(q, v) \mid q \in Q,\ v \in \mathbb{R}^{X \cup Y},\ \beta \in \{\top, \bot\}\Big\}$, with initial configuration $(q_0, \mathbf{0}, \bot)$. An accepting configuration of $\mathcal{T}_\mathcal{A}$ is a pair (q, v) with q in F. The relation $\to$ on S consists of time steps and discrete steps, the definition of the latter being the same as before:

Time steps: Only the active clocks in a state can evolve, all other clocks are suspended. For a state q with $\lambda(q) \in \mathbb{N}$ (the active clock is $x_{\lambda(q)}$), a time step of duration $d > 0$ is defined by $(q, v, \beta) \xrightarrow{d} (q, v', \top)$ with $v'(x_{\lambda(q)}) = v(x_{\lambda(q)}) + d$ and $v'(x) = v(x)$ for any other clock x. For a state q with $\lambda(q) \in \Omega$ (the active clocks are $Y' = Y \cap \lambda^{-1}(\lambda(q))$), a time step of duration $d > 0$ is defined by $(q, v, \beta) \xrightarrow{d} (q, v', \top)$ with $v'(y) = v(y) + vel(q)d$ for $y \in Y'$ and $v'(x) = v(x)$ for any other clock x. In all states, time steps of duration $d = 0$ leave the system $\mathcal{T}_\mathcal{A}$ in the same configuration. When $pol(q) = U$, only time steps of duration 0 q are allowed.

Discrete steps: A discrete step $(q, v) \xrightarrow{a} (q', v')$ occurs if there exists a transition $q \xrightarrow{\varphi, a, u} q'$ in Δ such that $v \models \varphi$ and $v' = v[u]$. When $pol(q) = D$ and $\beta = \bot$, discrete steps are forbidden.

In order to illustrate the interest of the combined models, an example of a (simple) login procedure is described in Fig. 19 as a TA with interruptions at a single level.

First it immediately displays a prompt and arms a time-out of 1 t.u. handled by clock y (transition $init \xrightarrow{p} wait$). Then either the user answers correctly within this delay (transition $wait \xrightarrow{ok} log$) or he answers incorrectly or let time elapse, both cases with transition $wait \xrightarrow{er} init$, and the system prompts again. The whole process is controlled by a global time-out of 6 t.u. (transition $wait \xrightarrow{to} out$) followed by a long suspension (50 t.u.) before reinitializing the process (transition $out \xrightarrow{rs} init$). Both delays are handled by clock z. At any time during the process (in fact in state $wait$), a system interrupt may occur (transition $wait \xrightarrow{i} I$). If the time spent (measured by clock x_1) during the interrupt is less than 3 t.u. or the time already spent by the user is less than 3 t.u., the login process resumes (transition $I \xrightarrow{cont} init$). Otherwise the login process is reinitialized allowing again the 6 t.u. (transition $I \xrightarrow{rs} init$). In both cases, the prompt will be displayed again. Since invariants are irrelevant for the reachability problem we did not include them in the models. Of course, in this example state $wait$ should have invariant $y \leq 1 \wedge z \leq 6$ and state out should have invariant $z \leq 50$.

We extend the decidability and complexity results of the previous models when combining them with CRTA. Class ITA$^+_-$ is obtained in a similar way by combining ITA$_-$ with CRTA.

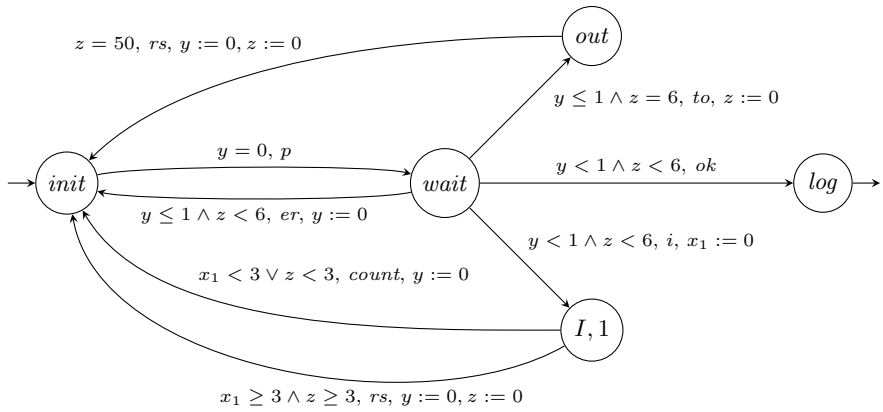


Fig. 19 An automaton for login in ITA$^+$

Proposition 14 *1. The reachability problem for ITA$^+_-$ is decidable in* NEXPTIME *and is* PSPACE*-complete when the number of interrupt clocks is fixed.*
2. The reachability problem for ITA$^+$ is decidable in NEXPTIME *and is* PSPACE*-complete when the number of interrupt clocks is fixed.*

Proof
Case of ITA$^+_-$. Let $\mathcal{A} = \langle Q, q_0, F, pol, X \uplus Y, \Sigma, \Omega, \lambda, up, low, vel, \Delta \rangle$ be an ITA$^+_-$, with $n = |X|$ the number of ITA clocks, $p = |Y|$ the number of CRTA clocks and $E = |\Delta|$ the number of transitions.

We first consider the reachability problem for two states q_i and q_f on the CRTA level (with $\lambda(q_i) \in \Omega$ and $\lambda(q_f) \in \Omega$). The procedure consists in performing a non deterministic search along an elementary path where the vertices are graph classes of the CRTA. Let (q, Z) be the current class, the procedure chooses non deterministically the next class (q', Z') and checks that there exists a configuration of (q, Z) and an execution only through states q'' with $\lambda(q'') \in \mathbb{N}$ that leads to a configuration of (q', Z'). This is solved as previously by non deterministically choosing an execution path, building a linear program related to the path (of exponential size) and solving it. Let us prove that such a path can be chosen whose length is in $O(p(E + 2n)^{3n})$.

Assume that there is a run π from $(q, v) \in (q, Z)$ to some configuration $(q', v') \in (q', Z')$ such that all intermediate states q'' are such that $\lambda(q'') \in \mathbb{N}$. We say that a transition e of π *usefully resets* a clock $y \in Y$ if it is the first transition of π that resets y. Observe that there are at most p useful resetting transitions and that between two such successive transitions (or before the first one or after the last one) the value of the clocks of Y are unchanged when transitions are fired.

We consider a subrun ρ between two such successive transitions (or before the first one or after the last one) from (q_1, v_1) to (q_2, v_2), with m_k the number of transitions of level k.

Using Lemma 2, we build a subrun ρ' from (q_1, v_1) to (q_2, v_2) of length smaller than $(E + 2n)^{3n}$. Concatenating the subruns, the useful resetting transitions and the initial transition, one obtains a run π' from (q, v) to (q', v') of length in $O(p(E + 2n)^{3n})$.

The key point ensuring correctness of the procedure is that the existence of a solution depends only on the starting class (q, Z) and not on the configuration inside

this class. This is due to the separation of guards and updates between the two kinds of clocks on the transitions.

When state q_i (resp. q_f) is not at the basis level, the procedure adds an initial (resp. final) guess also checked by a linear program. When the number of clocks is fixed the dominant factor is the path search in the class graph and PSPACE hardness follows from the result in TA.

Case of ITA$^+$. We transform the ITA part of the automaton in ITA$_-$ *via* the procedure of proposition 3 and apply the procedure for ITA$^+_-$. □

It is also possible to build a class graph for ITA^+, combining a class graph for ITA and a region graph for TA. This yields the regularity of the untimed language of an ITA$^+$, hence the strict inclusion in the languages accepted by a stopwatch automaton.

Let ITL$^+$ be the family of timed languages defined by ITA$^+$. The class ITL$^+$ syntactically contains ITL $\cup$ CRTL. We can however have a stronger result:

Proposition 15 *The class ITL$^+$* strictly *contains ITL $\cup$ CRTL.*

Proof Recall ITA $\mathcal{A}_4$ of Fig. 14, whose language L_4 is not in CRTL, and let Q_4 be its set of states. Also recall TA $\mathcal{A}_5$ of Fig. 15, whose language L_5 is not in ITL, with set of states Q_5. Let $\mathcal{A}_4 \otimes \mathcal{A}_5$ be the ITA$^+$ having $\mathcal{A}_5$ at level 0 and $\mathcal{A}_4$ at levels 1 and 2.

Formally, $\mathcal{A}_4 \otimes \mathcal{A}_5$ has set of states $Q_4 \cup Q_5$, which are all lazy. Interrupt clocks of $\mathcal{A}_4 \otimes \mathcal{A}_5$ are $\{x_1, x_2\}$ (active according to $\mathcal{A}_4$). Its basic clocks are $\{z, y\}$ of velocity 1. Both have the same color as states of Q_5. The bounding functions *up* (resp. *low*) map both z and y to 1 (resp. 0). Transitions of $\mathcal{A}_4 \otimes \mathcal{A}_5$ are the ones of $\mathcal{A}_4$ and $\mathcal{A}_5$, adding an unguarded, unlabeled transition from $\mathcal{A}_5$'s final state to $\mathcal{A}_4$'s initial one.

$\mathcal{A}_4 \otimes \mathcal{A}_5$ accepts timed words which start with an alternation of as and bs, with the b drawing always closer to its preceding a (as in $\mathcal{A}_5$), and then contains only cs separated by the same amount of time (as in $\mathcal{A}_4$). Since both CRTL and ITL are closed under projection, $\mathcal{L}(\mathcal{A}_4 \otimes \mathcal{A}_5)$ cannot be accepted by a CRTA nor an ITA. □

8 Conclusion

In this paper, we introduced and studied the model of Interrupt Timed Automata. This model is useful to represent timed systems with tasks organized over priority levels.

While ITA fall into the more general class of hybrid systems, the reachability problem is proved decidable for this subclass. For ITA, the reachability is in NEXPTIME, and PTIME when the number of clocks is fixed by building a class graph. Similar constructions yield decidability of the reachability problem on an extension of ITA where the lowest priority level can behave as a Controlled Real-Timed Automata. It also yields procedure for model checking CTL* formulas and timed CTL formulas constraining only the clocks of the system. Another fragment of interest was identified in timed CTL as decidable: the one where the only time constraints concern global earliest or latest execution times. On the other hand, model checking the linear time logic SCL is proved undecidable on ITA, implying that this is also the case for MITL.

On the expressiveness point of view, the class ITL is proved incomparable with both TL and CRTL, and is neither closed under complementation nor intersection. The expressiveness results are summed up in Fig. 20, where the grey zone represents undecidability of the reachability problem.

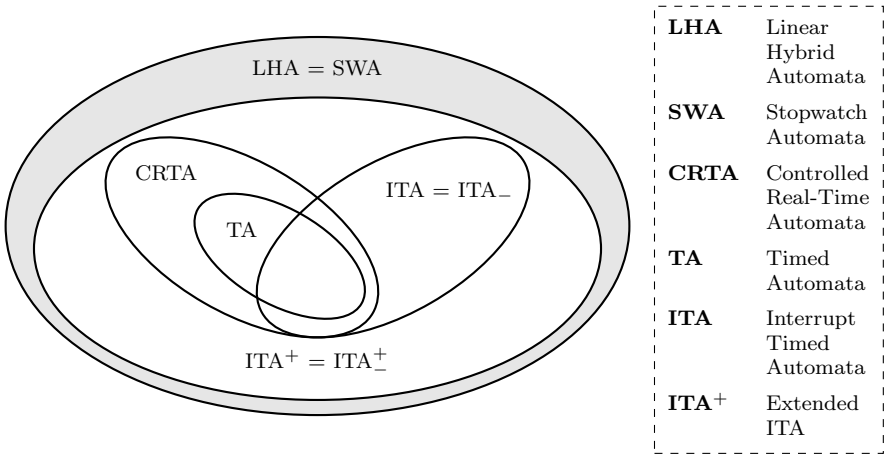


Fig. 20 Expressiveness of several timed formalisms with respect to timed languages.

Several problems remain open on the class of ITA. First of all, the effect of having both (limited) stopwatches and linear expressions in guards is combined in ITA, and it is not known which is the cause of the undecidability results presented in this paper. For instance, the undecidability of SCL may not hold without the possibility of complex updates. More generally, the expressive power of the subclass of ITA restricted with rectangular guards ($x + b \bowtie 0$) and only resets ($x := 0$) should be investigated. Also, it is conjectured that the class of ITA with $n + 1$ clocks is strictly more expressive than the class of ITA with n clocks. Regarding model-checking, the undecidability of full TCTL remains to be established. Finally, complexity bounds presented in this paper are only upper-bounds, and matching lower-bounds are still missing.

Acknowledgments The authors would like to thank the anonymous reviewers for their insightful comments. This work was supported by projects Dots (ANR-06-SETI-003, French government), ImpRo (ANR-2010-BLAN-0317, French government) and CoChaT (Digiteo 2009-27HD, Région Île de France).

References

1. Aceto, L., Burgueño, A., Larsen, K.G.: Model checking via reachability testing for timed automata. In: Proceedings of the 4th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'98). Volume 1384 of Lecture Notes in Computer Science, London, UK, Springer-Verlag (1998) 263–280.
2. Alur, R., Courcoubetis, C., Dill, D.L.: Model-checking in dense real-time. Information and Computation **104** (1993) 2–34.
3. Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A., Ho, P.H., Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: The algorithmic analysis of hybrid systems. Theoretical Computer Science **138** (1995) 3–34.
4. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science **126** (1994) 183–235.
5. Alur, R., Feder, T., Henzinger, T.A.: The benefits of relaxing punctuality. Journal of the ACM **43**(1) (janvier 1996) 116–146.
6. Asarin, E., Maler, O., Pnueli, A.: Reachability analysis of dynamical systems having piecewise-constant derivatives. Theoretical Computer Science **138**(1) (1995) 35–65.

7. Asarin, E., Schneider, G., Yovine, S.: Algorithmic analysis of polygonal hybrid systems, part I: Reachability. Theoretical Computer Science **379**(1-2) (2007) 231–265.
8. Behrmann, G., David, A., Larsen, K.G.: A tutorial on UPPAAL. In: Formal methods for the design of real-time systems (SFM-RT'04). Volume 3185 of Lecture Notes in Computer Science, Springer (2004) 200–236.
9. Bérard, B., Haddad, S.: Interrupt Timed Automata. In: Proceedings of the 12th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'09). Volume 5504 of Lecture Notes in Computer Science, York, GB, Springer (March 2009) 197–211.
10. Bérard, B., Haddad, S., Sassolas, M.: Real time properties for interrupt timed automata. In Markey, N., Wisjen, J., eds.: Proceedings of the 17th International Symposium on Temporal Representation and Reasoning (TIME'10), IEEE Computer Society (September 2010) 69–76.
11. Bérard, B., Petit, A., Diekert, V., Gastin, P.: Characterization of the expressive power of silent transitions in timed automata. Fundamenta Informaticae **36**(2-3) (1998) 145–182.
12. Bouyer, P.: Forward analysis of updatable timed automata. Formal Methods in System Design **24**(3) (2004) 281–320.
13. Bouyer, P., Brihaye, Th., Jurdziński, M., Lazić, R., Rutkowski, M.: Average-price and reachability-price games on hybrid automata with strong resets. In Cassez, F., Jard, C., eds.: Proceedings of the 6th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'08). Volume 5215 of Lecture Notes in Computer Science, Saint-Malo, France, Springer (September 2008) 63–77.
14. Bouyer, P., Chevalier, F., Markey, N.: On the expressiveness of TPTL and MTL. In: Proceedings of the 25th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'05). Volume 3821 of Lecture Notes in Computer Science, Springer (December 2005) 432–443.
15. Bozga, M., Daws, C., Maler, O., Olivero, A., Tripakis, S., Yovine, S.: KRONOS: A Model-Checking Tool for Real-Time Systems. In: FTRTFT. (1998) 298–302.
16. Brihaye, T., Bruyère, V., Raskin, J.F.: On model-checking timed automata with stopwatch observers. Information and Computation **204**(3) (2006) 408–433.
17. Brihaye, T., Doyen, L., Geeraerts, G., Ouaknine, J., Raskin, J.F., Worrell, J.: On reachability for hybrid automata over bounded time. In Aceto, L., Henzinger, M., Sgall, J., eds.: Proceedings (part II) of the 38th International Colloquium on Automata, Languages and Programming (ICALP'11). Volume 6756 of Lecture Notes in Computer Science, Springer (July 2011) 416–427.
18. Cassez, F., Larsen, K.G.: The impressive power of stopwatches. In Palamidessi, C., ed.: Proceedeings of the 11th International Conference on Concurrency Theory (CONCUR'00). Volume 1877 of Lecture Notes in Computer Science, Springer (2000) 138–152.
19. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV version 2: An opensource tool for symbolic model checking. In: Proceedings of the 14th International Conference on Computer Aided Verification (CAV'02). Volume 2404 of Lecture Notes in Computer Science, Springer (2002) 241–268.
20. Courcoubetis, C., Yannakakis, M.: Minimum and maximum delay problems in real-time systems. Formal Methods in System Design **1**(4) (1992) 385–415.
21. Demichelis, F., Zielonka, W.: Controlled timed automata. In Sangiorgi, D., de Simone Robert, eds.: Proceedings of the 9th International Conference on Concurrency Theory (CONCUR'98). Volume 1466 of Lecture Notes in Computer Science, London, UK, Springer-Verlag (1998) 455–469.
22. Emerson, E.A., Halpern, J.Y.: Decision procedures and expressiveness in the temporal logic of branching time. In: Proc. 14th annual ACM Symp. on Theory of Computing (Stoc'82), ACM (1982) 169–180.
23. Fersman, E., Krcal, P., Pettersson, P., Yi, W.: Task automata: Schedulability, decidability and undecidability. Information and Computation **205**(8) (2007) 1149–1172.
24. Henzinger, T.A., Kopke, P.W., Puri, A., Varaiya, P.: What's decidable about hybrid automata? Journal of Computer and System Sciences **57**(1) (1998) 94–124.
25. Henzinger, T.A., Nicollin, X., Sifakis, J., Yovine, S.: Symbolic model checking for real-time systems. Information and Computation **111(2)** (1994) 193–244.
26. Kesten, Y., Pnueli, A., Sifakis, J., Yovine, S.: Decidable integration graphs. Information and Computation **150**(2) (1999) 209–243.
27. Koymans, R.: Specifying real-time properties with metric temporal logic. Real-Time Systems **2** (1990) 255–299.

28. Lafferriere, G., Pappas, G.J., Yovine, S.: A new class of decidable hybrid systems. In: Proceedings of Hybrid Systems : Computation and Control. Volume 1569 of Lecture Notes in Computer Science, Springer (1999) 137–151.
29. Lafferriere, G., Pappas, G.J., Yovine, S.: Symbolic reachability computation for families of linear vector fields. Journal of Symbolic Computation **32**(3) (2001) 231–253.
30. Maler, O., Manna, Z., Pnueli, A.: From timed to hybrid systems. In Rozenberg, G., de Roever, W.P., Huizing, C., de Bakker, J.W., eds.: Real-time: theory in practice, REX workshop. Volume 600 of Lecture Notes in Computer Science, Springer-Verlag (1992) 447–484.
31. McManis, J., Varaiya, P.: Suspension automata: A decidable class of hybrid automata. In: Proceedings of the 6th International Conference on Computer Aided Verification (CAV'94), London, UK, Springer-Verlag (1994) 105–117.
32. Minsky, M.L.: Computation: finite and infinite machines. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1967).
33. Pnueli, A.: The temporal logic of programs. In: Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FoCS'77), Washington, DC, USA, IEEE Computer Society (1977) 46–57.
34. Queille, J.P., Sifakis, J.: Specification and verification of concurrent systems in CESAR. In Dezani-Ciancaglini, M., Montanari, U., eds.: Proceedings of the 5th International Symposium on Programming. Volume 137 of Lecture Notes in Computer Science, London, UK, Springer-Verlag (1982) 337–351.
35. Raskin, J.F., Schobbens, P.Y.: State clock logic: A decidable real-time logic. In Maler, O., ed.: Hybrid and Real-Time Systems. Volume 1201 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (1997) 33–47.
36. Roos, C., Terlaky, T., Vial, J.P.: Theory and Algorithms for Linear Optimization. An Interior Point Approach. Wiley-Interscience, John Wiley & Sons Ltd (1997).
37. Silberschatz, A., Galvin, P.B., Gagne, G.: Operating Systems Concepts. 8th edn. John Wiley & Sons, Inc. (jul 2008).
38. Sistla, A.P., Clarke, E.M.M.: The complexity of propositional linear temporal logics. Journal of the ACM **32** (jul 1985) 733–749.
39. Zaslavsky, T.: Facing up to arrangements: Face-count formulas for partitions of space by hyperplanes. AMS Memoirs **1**(154) (1975).

A Proof of Theorem 4

Let φ be a formula in $\mathsf{TCTL}_c^{\text{int}}$ and $\mathcal{A}$ an ITA with n levels and E transitions. Like in Section 3, the proof relies on the construction of a finite class graph. The main difference is in the computation of the n sets of expressions $E_1, \ldots, E_n$. Like before, each set E_k is initialized to $\{x_k, 0\}$ and expressions in this set are those which are relevant for comparisons with the current clock at level k. In this case, they include not only guards but also comparisons with the constraints from the formula. Recall that the sets are computed top down from n to 1, using the normalization operation.

- At level k, we may assume that expressions in guards of an edge leaving a state are of the form $\alpha x_k + \sum_{i<k} a_i x_i + b$ with $\alpha \in \{0, 1\}$. We add $-\sum_{i<k} a_i x_i - b$ to E_k.
- To take into account the constraints of formula φ, we add the following step: For each comparison $C \bowtie 0$ in φ, and for each k, with $\mathtt{norm}(C, k) = \alpha x_k + \sum_{i<k} a_i x_i + b$ ($\alpha \in \{0, 1\}$), we also add expression $-\sum_{i<k} a_i x_i - b$ to E_k.
- Then we iterate the following procedure until no new term is added to any E_i for $1 \leq i \leq k$.
  1. Let $q \xrightarrow{\varphi, a, u} q'$ with $\lambda(q') \geq k$ and $\lambda(q) \geq k$. If $C \in E_k$, then we add $C[u]$ to E_k.
  2. Let $q \xrightarrow{\varphi, a, u} q'$ with $\lambda(q') \geq k$ and $\lambda(q) < k$. For $C, C' \in E_k$, we compute $C'' = \mathtt{norm}(C[u] - C'[u], \lambda(q))$. If $C'' = \alpha x_{\lambda(q)} + \sum_{i<\lambda(q)} a_i x_i + b$ with $\alpha \in \{0, 1\}$, then we add $-\sum_{i<\lambda(q)} a_i x_i - b$ to $E_{\lambda(q)}$.

The proof of termination for this construction is similar to the one in Section 3.

We now consider the transition system $\mathcal{G}_\mathcal{A}$ whose set of configurations are the classes $R = (q, \{\preceq_k\}_{1 \leq k \leq \lambda(q)})$, where q is a state and $\preceq_k$ is a total preorder over E_k. The class R describes the set of valuations $[\![R]\!] = \{(q, v) \mid \forall k \leq \lambda(q)\ \forall (g, h) \in E_k,\ g[v] \leq h[v] \text{ iff } g \preceq_k h\}$. The set of transitions is defined as in Section 3. The transition system $\mathcal{G}_\mathcal{A}$ is again finite and time abstract bisimilar to $\mathcal{T}_\mathcal{A}$. Moreover, the truth value of each comparison $C = \sum_{i \geq 1} a_i \cdot x_i + b \bowtie 0$ appearing in φ can be set for each class R. Indeed, since for every k, both 0 and $\sum_{i \geq 1}^{k-1} a_i \cdot x_i + b$ are in the set of expressions E_k, the truth value of $C \bowtie 0$ does not change inside a class. Therefore, introducing a fresh propositional variable q_C for the constraint $C \bowtie 0$, each class R can be labelled with a truth value for each q_C. Deciding the truth value of φ can then be done by a classical CTL model-checking algorithm on $\mathcal{G}_\mathcal{A}$.

The complexity of the procedure is obtained by bounding the number of expressions for each level k by $(E + |\varphi| + 2)^{2^{n(n-k+1)}+1}$, and applying the same reasoning as for proposition 2.

B Proof of Theorem 7

We build an automaton in ITA×TA which simulates a deterministic two counter machine $\mathcal{M}$ (as in proof of Theorem 3).

Let $L_\mathcal{M}$ be the set of labels of $\mathcal{M}$. The automaton $\mathcal{A}_\mathcal{M} = \langle \Sigma, Q, q_0, F, pol, X \cup Y, \lambda, \Delta \rangle$ is built to reach its final location $Halt$ if and only if $\mathcal{M}$ stops. It is defined as follows:

- Σ consists of one letter per transition.
- $Q = L_\mathcal{M} \cup (L_\mathcal{M} \times \{k_0\}) \cup (L_\mathcal{M} \times \{k_1, k_2, r_1, \ldots, r_5\} \times \{>, <\})$, $q_0 = \ell_0$ (the initial instruction of $\mathcal{M}$) and $F = \{Halt\}$.
- $pol : Q \to \{Urgent, Lazy, Delayed\}$ is such that $pol(q) = Urgent$ iff either $q \in L_\mathcal{M}$ or $q = (\ell, q_2, \bowtie)$, and $pol(q) = Lazy$ in most other cases: some states $(\ell, k_i, \bowtie)$ are $Delayed$, as shown on Fig. 21 and 22.
- $X = \{x_1, x_2, x_3\}$ is the set of interrupt clocks and $Y = \{y_c, y_d\}$ is the set of standard clocks with rate 1.
- $\lambda : Q \to \{1, 2, 3\}$ is the interrupt level of each state. All states in $L_\mathcal{M}$ and $L_\mathcal{M} \times \{k_0, k_1, k_2\}$ are at level 1; so do all states corresponding to r_1. States corresponding to r_2 and r_3 are in level 2, while the ones corresponding to r_4 and r_5 are in level 3.
- Δ is defined through basic modules in the sequel.

The transitions of $\mathcal{A}_\mathcal{M}$ are built within small modules, each one corresponding to one instruction of $\mathcal{M}$. The value n of c (resp. p of d) in a state of $L_\mathcal{M}$ is encoded by the value $1 - \frac{1}{2^n}$ of clock y_c (resp. $1 - \frac{1}{2^p}$ of y_d).

The idea behind this construction is that for any standard clock y, it is possible to "copy" the value of $k - y$ in an interrupt clock x_i, for some constant k, provided the value of y never

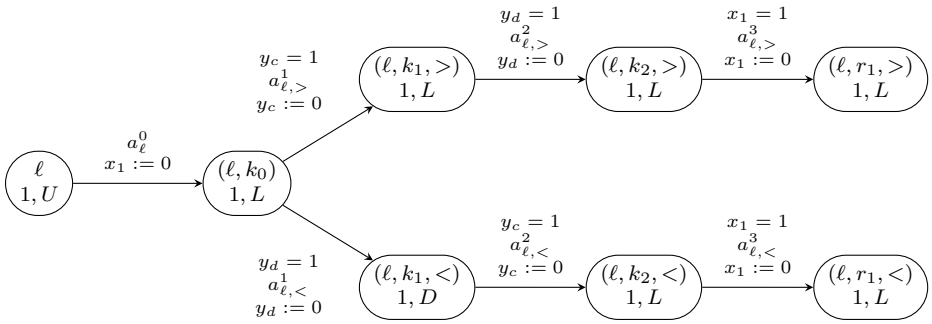


Fig. 21 Module taking into account the order between the values of c and d when incrementing c.

exceeds k. To achieve this, we start and reset the interrupt clock, then stop it when $y = k$. Note that by the end of the copy, the value of y has changed. Conversely, in order to copy the content of an interrupt clock x_i into a clock y, we switch from level i to level $i+1$ and reset y at the same time. When $x_{i+1} = x_i$, the value of y is equal to the value of x_i. Remark that the form of the guards on x_{i+1} allows us to copy the value of a linear expression on $\{x_1, \ldots, x_i\}$ in y.

For instance, consider an instruction labeled by ℓ incrementing c then going to ℓ', with the respective values n of c and p of d, from a configuration where $n \geq p$. The corresponding module $\mathcal{A}^{c++}_{c \geq d}(\ell, \ell')$ is depicted on Fig. 18 (see main text). In this module, interrupt clock x_1 is used to record the value $\frac{1}{2^n}$ while x_2 keeps the value $\frac{1}{2^p}$. Assuming that $y_c = 1 - \frac{1}{2^n}$, $y_d = 1 - \frac{1}{2^p}$ and $x_1 = 0$ in state $(\ell, r_1, >)$, the unique run in $\mathcal{A}^{c++}_{c \geq d}(\ell, \ell')$ will end in state ℓ' with $y_c = 1 - \frac{1}{2^{n+1}}$ and $y_d = 1 - \frac{1}{2^p}$. The intermediate clock values are shown in Table 3 (see main text).

The module on Fig. 18 can be adapted for the case of decrementing c by just changing the linear expressions in guards for x_3, provided that the final value of c is still greater than the one of d. It is however also quite easy to adapt the same module when $n < p$: in that case we store $\frac{1}{2^p}$ in x_1 and $\frac{1}{2^n}$ in x_2, since y_d will reach 1 before y_c. We also need to start y_d before y_c when copying the adequate values in the clocks. The case of decrementing c while $n \leq p$ is handled similarly. In order to choose which module to use according to the ordering between the values of the counters, we use the modules of Fig. 21 and 22. Fig. 21 represents the case when at label ℓ we have an increment of c whereas Fig. 22 represents the case when ℓ corresponds to decrementing c. In that last case the value of c is compared not only to the one of d, but also to 0, in order to know which branch of the *if* instruction is taken. Note that only one of the branches can be taken until the end[6]. Instructions involving d are handled in a symmetrical way.

Automaton $\mathcal{A}_\mathcal{M}$ is obtained by joining the modules described above through the states of $L_\mathcal{M}$. Let us prove that automaton $\mathcal{A}_\mathcal{M}$ simulates the two counter machine $\mathcal{M}$, so that $\mathcal{M}$ halts iff $\mathcal{A}_\mathcal{M}$ reaches the *Halt* state.

Let $\langle \ell_0, 0, 0 \rangle \langle \ell_1, n_1, p_1 \rangle \ldots \langle \ell_i, n_i, p_i \rangle \ldots$ be a run of $\mathcal{M}$. We show that this run is simulated in $\mathcal{A}_\mathcal{M}$ by the run $\langle l_0, \mathbf{0} \rangle \rho_0 \langle l_1, v_1 \rangle \rho_1 \ldots$ where ρ_i is either empty or a subrun through states in $\{(\ell_i, r_j, \bowtie) \mid j \in \{1, \ldots, 5\}, \bowtie \in \{>, <\}\}$ (*i.e.* subruns in modules like $\mathcal{A}^{c++}_{c \geq d}$ of Fig. 18). Moreover, it will be the case that

$$\forall i, \quad v_i(y_c) = 1 - \frac{1}{2^{n_i}} \quad \text{and} \quad v_i(y_d) = 1 - \frac{1}{2^{p_i}}$$

This holds at the beginning of the execution of $\mathcal{A}_\mathcal{M}$. Suppose that we have simulated the subrun up to $\langle \ell_i, n_i, p_i \rangle$. Then we are in state ℓ_i, with clock y_c being $1 - \frac{1}{2^{n_i}}$ and y_d being $1 - \frac{1}{2^{p_i}}$. The next configuration of $\mathcal{M}$, $\langle \ell_{i+1}, n_{i+1}, p_{i+1} \rangle$, depends on the content of instruction ℓ_i, and so does the outgoing transitions of state ℓ_i in $\mathcal{A}_\mathcal{M}$. We consider the case where ℓ_i decrements c

[6] State policies are used to treat the special cases, *e.g.* $y_c = y_d = 0$.

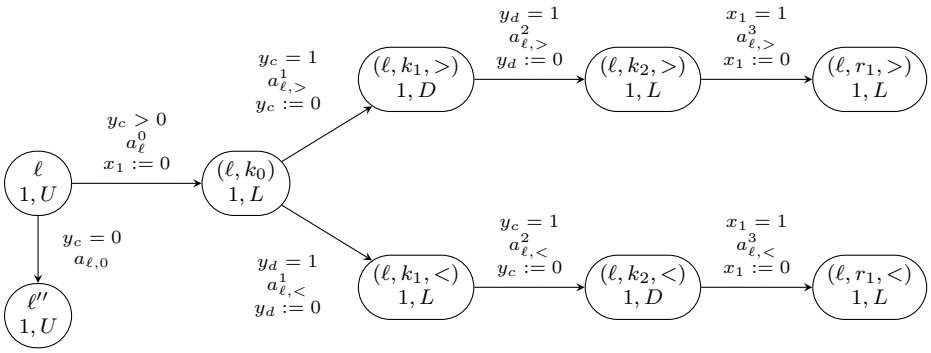


Fig. 22 Module taking into account the order between the values of c and d when decrementing c.

and goes to ℓ' if c is greater than 0 and goes to ℓ'' otherwise, the other ones being similar. We are therefore in the case of Fig. 22. If $n_i = 0$, the next configuration of $\mathcal{M}$ will be $\langle \ell'', n_i, p_i \rangle$. Conversely, in $\mathcal{A}_\mathcal{M}$, if $n_i = 0$ then $y_c = 0$, and there is no choice but to enter ℓ'', leaving all clock values unchanged (because ℓ_i is an *Urgent* state). The configuration of $\mathcal{A}_\mathcal{M}$ thus satisfies the property. If $n_i > 0$, the next configuration of $\mathcal{M}$ will be $\langle \ell', n_i - 1, p_i \rangle$. In $\mathcal{A}_\mathcal{M}$, the transition chosen is the one that corresponds to the ordering between n_i and p_i. In both cases, similarly to the example of $\mathcal{A}^{c++}_{c \geq d}(\ell, \ell')$, the run reaches state ℓ' with $y_c = 1 - \frac{1}{2^{n_i - 1}}$ and y_d as before, thus preserving the property. Hence $\mathcal{M}$ halts iff $\mathcal{A}_\mathcal{M}$ reaches the *Halt* state.

The automaton $\mathcal{A}_\mathcal{M}$ is indeed the product of an ITA $\mathcal{I}$ and a TA $\mathcal{T}$, synchronized on actions. Observe that in all the modules described above, guards never mix a standard clock with an interrupt one. Since each transition has a unique label, keeping only guards and resets on either the clocks of X or on those of Y yields an ITA and a TA whose product is $\mathcal{A}_\mathcal{M}$. □