Real Time Properties for Interrupt Timed Automata*

Béatrice Bérard† Serge Haddad‡ Mathieu Sassolas†

†Université Pierre & Marie Curie,
LIP6/MoVe, CNRS UMR 7606, Paris, France

‡École Normale Supérieure de Cachan,
LSV, CNRS UMR 8643, Cachan, France

Abstract

Interrupt Timed Automata (ITA) have been introduced to model multi-task systems with interruptions. They form a subclass of stopwatch automata, where the real valued variables (with rate 0 *or* 1*) are organized along priority levels. While reachability is undecidable with usual stopwatches, the problem was proved decidable for ITA.*

In this work, after giving answers to some questions left open about expressiveness, closure, and complexity for ITA, our main purpose is to investigate the verification of real time properties over ITA. While we prove that model checking a variant of the timed logic TCTL is undecidable, we nevertheless give model checking procedures for two relevant fragments of this logic: one where formulas contain only model clocks and another one where formulas have a single external clock.

1 Introduction

Context. Scheduling problems in multi-task systems are usually modeled with stopwatches, *i.e.* variables which evolve with rate 0 or 1 and can be tested and updated when discrete transitions are fired. Thus stopwatches model clocks that can be suspended and restarted with their former value, which makes them useful to express delay accumulation. However, adding such variables to finite automata yields the powerful model of Stopwatch Automata (SwA) [15, 10] where reachability has been proved undecidable. On the other hand, reachability is PSPACE-complete in the now classical model of Timed Automata (TA) [2, 3], where all variables are clocks, with single rate 1.

Restricting SwA to gain decidability, while retaining part of the power of stopwatches, is a difficult problem. A few models have been proposed, for instance Suspension Automata [17], Hierarchical Timed Automata [12], Task Automata [14] or Interrupt Timed Automata (ITA) [5]. We consider here the ITA model, specifically adapted to the description of multi-task systems with hierarchical interrupt levels in a single processor environment, like operating systems. As proved in [5], untiming languages accepted by ITA yields regular languages with the effective construction of a class graph generalizing the region automaton from [2].

While untimed properties like reachability or CTL model checking [13, 19, 11] are useful for such models, real time verification allows to obtain more precise results, for instance quantitative response time properties. Therefore, timed extensions of CTL have been defined, leading to different versions of Timed CTL (TCTL) [1, 16] for which several tools on TA have been developed [4, 9].

Contribution. Starting with studying closure properties and settling an expressiveness conjecture from [5], we improve complexity of reachability on ITA. We then focus on the verification of real time properties for ITA. These properties are expressed in TCTL_c, a timed extension of CTL where formulas involve model clocks as well as external clocks. This logic is a variant of the one in [16], also studied later from the expressivity point of view [8]. Unfortunately, we prove here that, contrary to reachability, model checking TCTL_c over ITA is undecidable. The result holds for a fixed two-clock formula, showing its robustness (which is not always the case in similar proofs). However, we propose two fragments for which decidability procedures can be found. In the first one, only model clocks are involved and we can 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-EXPSPACE (PSPACE if the number of clocks is fixed). Since the corresponding fragment cannot refer to global time, we consider a second fragment in which it is possible to reason on minimal or maximal delays. Properties like *(P2) the system is error free for at least 50 t.u.* or *(P3) the*

*Work partially supported by projects CoChaT (Digiteo 2009-27HD) and DOTS (ANR-06-SETI-003)

system will reach a safe state within 7 t.u. can be expressed. In this case, the decidability procedure relies on a new specific technique involving infinite runs.

Outline. Section 2 gives definitions for ITA, expressiveness, closure, and complexity results. We prove in Section 3 that model checking TCTL_c over ITA is undecidable and Section 4 presents model checking procedures for two fragments of TCTL_c.

2 Interrupt Timed Automata

Notations. The sets of natural, rational and real numbers are denoted respectively by $\mathbb{N}$, $\mathbb{Q}$ and $\mathbb{R}$. 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 the a_xs 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 of assignments of the form $x := C$ for a clock $x \in X$ and a linear expression C 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 containing $x := C_x$, the expression $C[u]$ is obtained by substituting x by C_x in C.

A clock valuation is a mapping $v : X \mapsto \mathbb{R}$ and we denote by $\mathbf{0}$ the valuation with value 0 for all clocks. The set of all clock valuations is $\mathbb{R}^X$ and we write $v \models \varphi$ when valuation v satisfies the clock constraint φ. 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.

Interrupt timed automata and timed automata. Interrupt Timed Automata (ITA) were introduced in [5] to model multi-task systems with interruptions.

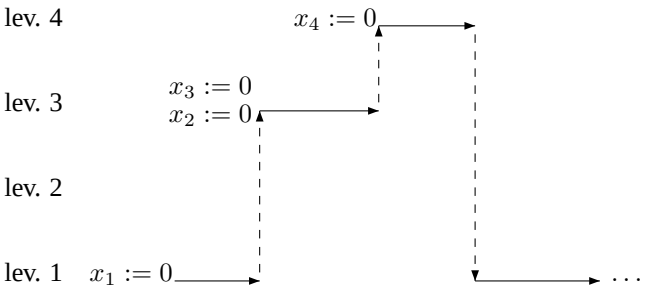


Figure 1. Interrupt levels and clocks in an ITA.

Given a set of tasks with different priority levels, 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 Figure 1.

We extend the definition by associating with states a timing policy which indicates whether time may (Lazy, default), may not (Urgent) or must (Delayed) elapse in a state. This feature could not be enforced by additional clock constraints like in TA and is needed to obtain the translation from ITA to ITA$_-$ (see below). We also add a labeling of states with atomic propositions, in view of interpreting logic formulas on these automata.

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* $lab : Q \to 2^{AP}$ *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. We call* $x_{\lambda(q)}$ *the active clock in state* q*. Let* $q \xrightarrow{\varphi,a,u} q'$ *in* Δ *be a transition with* $k = \lambda(q)$ *and* $k' = \lambda(q')$*. The guard* φ *contains only clocks from levels less than or equal to* k*: it is a conjunction of constraints of the form* $\sum_{j=1}^{k} a_j x_j + b \bowtie 0$*. 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* C_i *is of the form* $\sum_{j=1}^{i-1} a_j x_j + b$ *or* $C_i = x_i$ *(unchanged clock value) for* $1 \leq i \leq k'$ *and* $C_i = 0$ *otherwise;*
  - *if* $k' \geq k$ *then* C_i *is of the form* $\sum_{j=1}^{i-1} a_j x_j + b$ *or* $C_i = x_i$ *for* $1 \leq i \leq k$*,* $C_i = 0$ *for* $k < i$*.*

The class ITA$_-$ is the subclass of ITA where updates are restricted as follows. For a transition $q \xrightarrow{\varphi,a,u} q'$ of an automaton $\mathcal{A}$ in ITA$_-$, with $k = \lambda(q)$ and $k' = \lambda(q')$, there is no update (*i.e.* $x_i := x_i$ for all i) if $k' < k$ and if $k' \geq k$, the update u is of the form $\wedge_{i=1}^{n} x_i := C_i$ with C_k of the form $\sum_{j=1}^{k-1} a_j x_j + b$ or $C_k = x_k$, $C_i = 0$ if $k < i$ and $C_i = x_i$ if $i < k$. Thus, in an ITA$_-$, the only non trivial update (*i.e.*

not enforced by the semantics of the model) is an update of the clock of the current level, when the transition does not decrease the level.

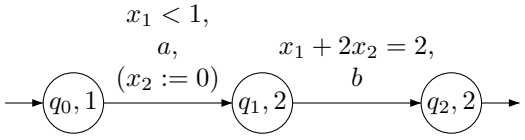
A configuration of the system consists of a state of the ITA, a clock valuation and a boolean value expressing whether time has elapsed since the last discrete transition.

Definition 2. *The semantics of an ITA $\mathcal{A}$ is defined by the transition system $\mathcal{T}_{\mathcal{A}} = (S, s_0, \rightarrow)$. The set S of configurations is $\{(q, v, \beta) \mid q \in Q,\ v \in \mathbb{R}^X,\ \beta \in \{\top, \bot\}\}$, with initial configuration $(q_0, \mathbf{0}, \bot)$. The relation $\rightarrow$ on S consists of two types of steps:*

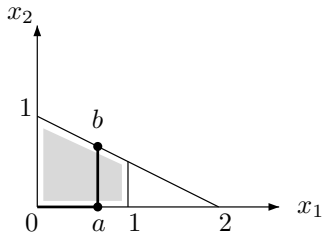
Time steps: *Only the active clock in a state can evolve, all other clocks are suspended. For a state q with active clock $x_{\lambda(q)}$, a time step of duration $d > 0$ is defined by $(q, v, \beta) \xrightarrow{d} (q, v', \top)$ with $v'(x_{\lambda(q)}) = v(x_{\lambda(q)}) + d$ and $v'(x) = v(x)$ 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.*

An ITA $\mathcal{A}_1$ is depicted in Figure 2(a), with two interrupt levels (and two interrupt clocks), with a geometric view of a possible trajectory in Figure 2(b).



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



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

Figure 2. An example of ITA and a possible execution.

We now briefly recall the classical model of timed automata (TA) [3] (in which 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, and X are defined as in an ITA and the set of transition is $\Delta \subseteq Q \times \mathcal{C}_0(X) \times \Sigma \times \mathcal{U}_0(X) \times Q$, 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 additionnal 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 $\forall x \in X,\ v'(x) = v(x) + d$.

A run of an automaton $\mathcal{A}$ in TA or in ITA is a path in the associated timed transition system, where time steps and discrete steps alternate. An *accepting run* is a finite run ending in a state of F. For such a run with label $d_1 a_1 d_2 \ldots d_n a_n$, we say that the word $(a_1, d_1)(a_2, d_1 + d_2) \ldots (a_n, d_1 + \cdots + d_n)$ (where ε actions are removed) is accepted by $\mathcal{A}$. The set $\mathcal{L}(\mathcal{A})$ contains the timed words accepted by $\mathcal{A}$. Interrupt Timed Languages or ITL (resp. Timed Languages or TL) denote the family of timed languages accepted by an ITA (resp. a TA). We also consider *maximal runs* which are either infinite or such that no discrete step is possible from the last configuration. We use the notion of (totally ordered) positions (which allow to consider multiple discrete actions simultaneously) along a maximal run [16]: for a run ρ, we denote by $<_\rho$ the strict order on positions and for position π along ρ, the corresponding configuration is denoted by s_π.

Expressiveness, closure, and complexity results. We end this section by closing some questions left open in [5] and improve complexity bounds for the reachability problem on ITA. In particular, while it was known that ITL is not contained in TL, the converse was not proved. We have:

Proposition 1. *The families TL and ITL are incomparable. ITL is neither closed under complementation, nor under intersection.*

These proofs rely on a specific *pumping lemma* for ITA. Note that incomparability of languages accepted by TA and ITA also proves that ITA are not in the same class than Hierarchical Timed Automata (HTA) from [12], since it was also proved that these HTA can be flattened into a network of TA.

Finally, we prove that ITA and ITA$_{-}$ have the same expressive power:

Proposition 2. *Any ITA can be translated into an ITA$_{-}$ accepting the same language, with the same set of clocks. The number of states and transitions is doubly exponential in the number of clocks.*

This transformation allows to reduce reachability for ITA to the same problem for ITA$_{-}$, where it is solved by bounding the length of a minimal path. The bound is exponential for ITA$_{-}$, but stays only doubly exponential for ITA, due to the conservation of the number of clocks. Thus, we have:

Proposition 3. *Reachability on ITA can be done in 2-NEXPTIME and in NP when the number of clocks is fixed.*

These results improve the ones of [5] where the upper bounds were in 2-EXPSPACE and in PSPACE when the number of clocks is fixed.

Detailed proofs for these results can be found in [6].

3 Model checking TCTL over ITA

3.1 Timed logic TCTL_c.

At least two different timed extensions of the branching time logic CTL have been proposed. The first one [1] adds subscripts to the U operator while the second one considers formula clocks [16]. Model checking of timed automata was proved decidable in both cases and compared expressiveness has been revisited later on [8].

In the variant below, CTL is enriched with both model clocks (set X), used in linear constraints, and formula clocks (set Y disjoint from X), used only in comparisons to constants and resets. Such linear constraints yield a more expressive logic, which raises the question of decidability both for TA and ITA.

Definition 4. *Formulas of the timed logic TCTL_c are defined by the following grammar:*

$$\psi ::= p \mid y + b \bowtie 0 \mid \sum_{i \in I} a_i \cdot x_i + b \bowtie 0 \mid y.\psi \mid$$

$$\mathsf{A}\,\psi\,\mathsf{U}\,\psi \mid \mathsf{E}\,\psi\,\mathsf{U}\,\psi \mid \psi \wedge \psi \mid \neg\psi$$

where $p \in AP$ is an atomic proposition, $y \in Y$ is a formula clock, x_i are model clocks, a_i and b are rational numbers such that $(a_i)_{i\in I}$ has finite support $I \subseteq \mathbb{N}$, and $\bowtie \in \{>, \geq, =, \leq, <\}$.

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 TCTL_c are interpreted over extended configurations of the form (q, v, β, w), also written as (s, w), where $s = (q, v, \beta) \in S$ and $w \in \mathbb{R}^Y$ is a valuation of the formula clocks[1]. The notions of (maximal) run and position are extended to these configurations in a natural way: the clock valuation w becomes $w + d$ in a time step of delay d and is unchanged in a discrete step. We denote by $Exec(s, w)$ the set of maximal runs starting from (s, w).

The semantics of TCTL_c is defined as follows. For atomic propositions and a configuration $(s, w) = (q, v, \beta, w)$:

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

and inductively:

$$\begin{array}{ll} (s,w) \models y.\psi & \text{iff } y \in Y \text{ and } (q, v, w[y := 0]) \models \psi \\ (s,w) \models \mathsf{A}\,\varphi\,\mathsf{U}\,\psi & \text{iff } \forall \rho \in Exec(s,w),\ \rho \models \varphi\,\mathsf{U}\,\psi \\ (s,w) \models \mathsf{E}\,\varphi\,\mathsf{U}\,\psi & \text{iff } \exists \rho \in Exec(s,w) \text{ s. t. } \rho \models \varphi\,\mathsf{U}\,\psi \end{array}$$

with $\rho \models \varphi\,\mathsf{U}\,\psi$ iff there is a position $\pi \in \rho$ s. t. $s_\pi \models \psi$ and $\forall \pi' <_\rho \pi,\ s_{\pi'} \models \varphi \vee \psi$

the cases for boolean operators are omitted.

3.2 Undecidability of TCTL_c model checking.

We now prove that model checking TCTL_c over ITA is undecidable. More precisely, let $\mathsf{TCTL}_c^{\text{ext}}$ be the fragment of TCTL_c containing only formula clocks, we have:

Theorem 1. *Model checking TCTL_c^{ext} over ITA is undecidable.*

The first step of the proof is the construction of automaton $\mathcal{A}_\mathcal{M}$, as a synchronized product between an interrupt timed automaton and a timed automaton, to simulate a two counter machine $\mathcal{M}$. In the second step, a TCTL_c formula with two external clocks is built to simulate the timed automaton part of the product. This formula does not depend on the two counter machine.

First step. We consider the class ITA×TA of automata built as a synchronized product between an interrupt timed automaton and a timed automaton over the same alphabet. Note that if accepted languages are considered, the language of such an automaton is the intersection of the language of an ITA and the language of a TA.

Lemma 1. *Reachability is undecidable in the class ITA×TA.*

Sketch. We build an automaton in ITA×TA which simulates a deterministic two counter machine. Recall that such a machine $\mathcal{M}$ consists of a finite sequence of labeled instructions L, 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''

[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.

Without loss of generality, we may assume that the counters have initial value zero. The behaviour of the machine is described by a (possibly infinite) sequence of configurations: $\langle \ell_0, 0, 0\rangle\langle \ell_1, n_1, p_1\rangle \dots \langle \ell_i, n_i, p_i\rangle \dots$, 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 [18].

The automaton $\mathcal{A}_{\mathcal{M}} = \langle \Sigma, AP, Q, q_0, F, pol, X \cup Y, \lambda, lab, \Delta\rangle$ 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, AP is defined in the sequel.
- $Q = L \cup (L \times \{k_0\}) \cup (L \times \{k_1, k_2, r_1, \dots, r_5\} \times \{>, <\})$, $q_0 = \ell_0$ (the initial instruction of $\mathcal{M}$) and $F = \{Halt\}$.
- $pol : Q \to \{Urgent, Lazy, Delayed\}$ is such that $pol(q) = Urgent$ iff either $q \in L$ 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 Figure 4.
- $X = \{x_1, x_2, x_3\}$ 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 are at level 1; so do all states corresponding to k_0, k_1, k_2 and 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.
- lab will be defined in the second step of the proof.
- 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 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 mimic the copy of the value of $k - y$ in an interrupt clock x_i, for some constant k, provided the value of y never 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 interrupt x_i by x_{i+1} and reset y at the same time. When $x_{i+1} = x_i$, clock y has the value of x_i. Remark that the form of the guards on x_{i+1} allows us to copy any linear expression on $\{x_1, \dots, x_i\}$ 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 Figure 3. In

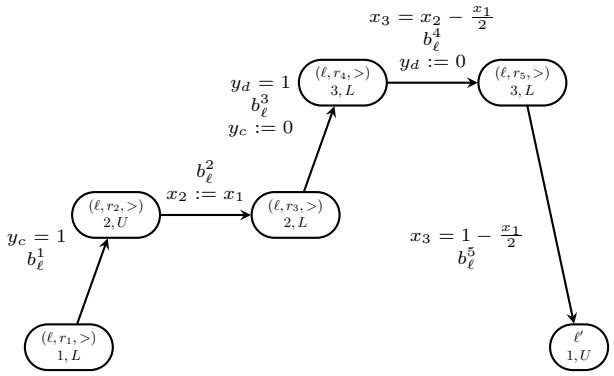


Figure 3. Module $\mathcal{A}^{c++}_{c \geq d}(\ell, \ell')$ incrementing the value of c when $c \geq d$.

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 module on Figure 3 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 module of Figure 4 which represents the case when at label ℓ we have an increment of c, or a similar one for decrementation. 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 of the module[2]. Instructions involving d are handled in a symmetrical way.

$\mathcal{A}_{\mathcal{M}}$ is obtained by joining the modules described above through the states of L. The automaton $\mathcal{A}_{\mathcal{M}}$ can actually be viewed as the product of an ITA $\mathcal{I}$ and a TA $\mathcal{T}$, synchronized on actions. It can be seen in all the modules described above that 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 $\mathcal{A}_{\mathcal{M}}$. □

Note that another notion of synchronized product between ITA and TA leads to the class ITA$^+$ where reachability is decidable [5].

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

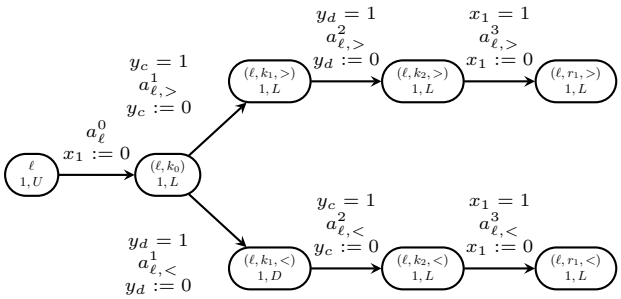


Figure 4. Module taking into account the order between the values of c and d when incrementing c.

Second step. To prove Theorem 1, we build from the automaton $\mathcal{A}_\mathcal{M}$ above a formula φ in TCTL_c simulating the TA $\mathcal{T}$, so that the ITA $\mathcal{I}$ satisfies φ iff $\mathcal{M}$ terminates. Formula φ expresses that (1) there is a run in $\mathcal{I}$ reaching the *Halt* state, and (2) for each module of $\mathcal{I}$, this run satisfies the constraints on the clocks y_c and y_d of $\mathcal{T}$.

The full proofs that the above construction is correct ($\mathcal{M}$ halts iff $\mathcal{A}_\mathcal{M}$ reaches the *Halt* state) and for this second step are given in [7]. Observe that state policies allow an encoding with two TA clocks; an additional one would be needed to simulate policies.

4 Decidable fragments

4.1 Model checking $\mathsf{TCTL}_c^{\text{int}}$

In this section we consider formulas with only model clocks, the corresponding fragment being denoted by $\mathsf{TCTL}_c^{\text{int}}$. For example property $P1$ in the introduction is expressed by $\mathsf{A}\, x_2 \leq 3\, \mathsf{U}\, \textit{safe}$. Model checking is achieved by adapting a class graph construction for untiming ITA and adding information relevant to the formula. The problem is thus reduced to a CTL model checking problem on this graph.

Theorem 2. *Model checking $\mathsf{TCTL}_c^{\text{int}}$ on interrupt timed automata can be done in 2-EXPSPACE, and in PSPACE when the number of clocks is fixed.*

Proof. The proof relies on a refinement of the class graph construction in [5], each class being divided into subclasses corresponding to truth values of comparisons in the given formula. Thus each comparison can be represented by a fresh propositional variable. The final step of the algorithm consists in applying standard CTL model-checking procedure.

Let φ be a formula in $\mathsf{TCTL}_c^{\text{int}}$ and $\mathcal{A}$ an ITA with n levels. In order to build the finite class graph, the first step consists in computing n sets of expressions $E_1, \ldots, E_n$. 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. The sets are then computed top down from n to 1. In that process, we use the k-normalization operator: for an expression $C = \sum_{i\geq 1} a_i x_i + b$, if $a_k = 0$, then $\mathtt{norm}(C,k) = \sum_{i=1}^{k-1} a_i x_i + b$, otherwise $\mathtt{norm}(C,k) = x_k + \sum_{i=1}^{k-1} \frac{a_i}{a_k} x_i + \frac{b}{a_k}$.

- At level k, we may assume (by normalization) 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 [5].

Consider the ITA $\mathcal{A}_1$ (Figure 2(a)) and the formula $\varphi_1 = \mathsf{E} \top \mathsf{U} (q_1 \wedge (x_2 > x_1))$. 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. After iteration, we obtain $E_1 = \{x_1, 0, 1, \frac{2}{3}, 2\}$ 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 next step is to build the class graph as 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 $[\![R]\!] = \{(q,v) \mid \forall k \leq \lambda(q)\, \forall (g,h) \in E_k,\ g[v] \leq h[v]$ iff $g \preceq_k h\}$. The set of transitions is defined by discrete and successor steps, whose details are developed in [5]. Just remark that the way the set of expressions is computed,

and more notably the inclusion of all differences between other expressions (up to normalization details), will enable us to know for each level the preorder between expressions after firing a discrete transition increasing the interrupt level. The transition system $\mathcal{G}_{\mathcal{A}}$ is 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 labeled 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 $\mathcal{G}_{\mathcal{A}}$.

On the example, we obtain the states in which $q_1 \wedge (x_2 > x_1)$ is *true* and conclude that φ_1 is *true* on $\mathcal{A}_1$.

The complexity of the procedure is obtained by bounding the number of expressions for each level k by $\max(2, |\Delta| + |\varphi|)^{2^{n(n-k+1)}+1}$, thus obtaining a triple exponential bound for the size of the graph, by storing the orderings. The 2-EXPSPACE complexity results in a standard way from a non deterministic search in this graph. □

Due to the linear constraints we conjecture that model checking $\mathsf{TCTL}_c^{\text{int}}$ on TA is undecidable. This would enforce the incomparability of TL and ITL from a decidability point of view.

4.2 Model checking a fragment of TCTL

The decidability of model-checking TCTL_c formulas over ITA has been studied above for two cases: (1) when there are 2 formula clocks, in which case the problem is undecidable (Theorem 1) and (2) when there is no formula clock, in which case the problem is decidable (Theorem 2).

The remaining case concerns formulas with only 1 formula clock, which can measure elapsing of global time. In this section, we prove the decidability of model checking ITA for a strict subset of this logic. TCTL_p is the set of formulas where satisfaction of an *until* modality over propositions can be parameterized by a time interval. Formulas of TCTL_p are defined by the following grammar:

$$\begin{array}{rcl} \varphi_p & := & p|\varphi_p \wedge \varphi_p|\neg\varphi_p \text{ and} \\ \psi & := & \varphi_p|\mathsf{A}\,\varphi_p\,\mathsf{U}_{\bowtie a}\,\varphi_p|\mathsf{E}\,\varphi_p\,\mathsf{U}_{\bowtie a}\,\varphi_p|\psi \wedge \psi|\neg\psi \end{array}$$

where $p \in AP$ is an atomic proposition, $a \in \mathbb{Q}^+$, and $\bowtie \in \{>, \geq, \leq, <\}$ is a comparison operator. This logic is indeed a subset of $TCTL_c$ with only one formula clock since a formula, say $\mathsf{A}\,p\,\mathsf{U}_{>a}r$, can be rewritten as $y.(\mathsf{A}\,p\,\mathsf{U}\,(r \wedge (y > a)))$. Properties $P2$ and $P3$ from introduction are expressed respectively as $\mathsf{A}\,\neg \textit{error}\,\mathsf{U}_{\geq 50}\,\top$ and $\mathsf{A}\,\top\,\mathsf{U}_{\leq 7}\,\textit{safe}$. Since ITA can be translated into ITA_{-}, the problem can be simplified by focusing on the subclass ITA_{-}. We prove that:

Theorem 3. *Model checking TCTL_p on ITA is decidable.*

The proof consists in establishing procedures dedicated to the 4 different subcases for ITA_{-}: (1) $\mathsf{E}\,p\,\mathsf{U}_{\leq a}\,r$ and $\mathsf{E}\,p\,\mathsf{U}_{<a}\,r$, (2) $\mathsf{E}\,p\,\mathsf{U}_{\geq a}\,r$ and $\mathsf{E}\,p\,\mathsf{U}_{>a}\,r$, (3) $\mathsf{A}\,p\,\mathsf{U}_{\leq a}\,r$ and $\mathsf{A}\,p\,\mathsf{U}_{<a}\,r$ and (4) $\mathsf{A}\,p\,\mathsf{U}_{\geq a}\,r$ and $\mathsf{A}\,p\,\mathsf{U}_{>a}\,r$, where p and r are boolean combinations of atomic propositions. Detailed proofs can be found in [7]. First, we have:

Lemma 2. *a. Model checking formulas $\mathsf{E}\,p\,\mathsf{U}_{\leq a}\,r$ and $\mathsf{E}\,p\,\mathsf{U}_{<a}\,r$ over ITA_{-} is decidable in NEXPTIME and NP when the number of clocks is fixed.*

b. Model checking a formula $\mathsf{E}\,p\,\mathsf{U}_{\geq a}\,r$ and $\mathsf{E}\,p\,\mathsf{U}_{>a}\,r$ on an ITA_{-} is decidable in NEXPTIME and NP when the number of clocks is fixed.

c. Model checking a formula $\mathsf{A}\,p\,\mathsf{U}_{\leq a}\,r$ and $\mathsf{A}\,p\,\mathsf{U}_{<a}\,r$ on an ITA_{-} is decidable in co-NEXPTIME and co-NP when the number of clocks is fixed.

Sketch. The main idea underlying these procedures for cases 1 and 2 is to obtain a maximal (exponential in the number of clocks) size for the runs on which it is sufficient to test the formula. Then 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 $\{x_i^j\}$, 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 satisfaction of the formula can be 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 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 [20], we obtain a procedure in NEXPTIME. If the number of clocks is fixed the number of variables is now polynomial w.r.t. the size of the problem.

For formulas in case 3, a specific procedure can be avoided: the result of case 2 can be reused since $\mathsf{A}\,p\,\mathsf{U}_{\leq a}\,r = (\mathsf{A}\,p\,\mathsf{U}\,r) \wedge \neg(\mathsf{E}\,\neg r\,\mathsf{U}_{>a}\,\top)$, and $\mathsf{A}\,p\,\mathsf{U}_{<a}\,r = (\mathsf{A}\,p\,\mathsf{U}\,r) \wedge \neg(\mathsf{E}\,\neg r\,\mathsf{U}_{\geq a}\,\top)$. □

While a counterexample is a finite path in the three previous cases, it is potentially infinite in case 4. Therefore, the proof is more difficult and the decidability procedure relies on a specific technique.

Lemma 3. *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.*

Sketch. We start by noticing that formula $\mathsf{A}\, p\, \mathsf{U}_{\geq \mathsf{a}}\, r$ is *true* on a configuration of an ITA$_-$ $\mathcal{A}$ if all the following conditions hold for paths starting in this configuration:

- all paths do satisfy $p\, \mathsf{U}\, r$,
- there is no path such that from a certain point where the time elapsed is strictly less than a, proposition r is *false* until both p and r are,
- there is no path such that from a certain point where the time elapsed is strictly less than a, proposition r is always *false*.

Using maximal paths, which are either infinite or finite but ending in a state from which no transition can be taken, is necessary for this last condition. □

5 Conclusion and related work

Several restrictions of stopwatch automata have been proposed to gain decidability results. For the model of suspension automata [17], reachability is decidable when stopwatches have value zero if suspended and satisfy some additional bounds. In the case of preemptive scheduling, the clocks in task automata from [14] can be updated by subtraction, which can be viewed as a kind of stopwatch simulation. Checking schedulability is proved decidable for several scheduling policies (and undecidable in general).

In this work we consider interrupt timed automata, where stopwatches are organized along hierarchical levels. Although model checking TCTL formulas with explicit clocks is undecidable, we obtain decidability for two subsets of real time properties: when only model clocks are used in the formula, with a complexity in 2-EXPSPACE, and for a subset of TCTL with subscripts. The case of formulas with internal clocks and only one external clock, remains open. We also plan to extend these results to ITA$^+$ which subsumes both TA and ITA.

Acknowledgments. We wish to thank the anonymous reviewers for their insightful comments and suggestions.

References

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

[2] R. Alur and D. L. Dill. Automata for modeling real-time systems. In *Proc. of the 17th Int. Colloquium on Automata, Languages and Programming*, pages 322–335, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

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

[4] G. Behrmann, A. David, and K. G. Larsen. A tutorial on UPPAAL. In *Formal methods for the design of real-time systems (SFM-RT'04)*, volume 3185 of *LNCS*, pages 200–236. Springer, 2004.

[5] B. Bérard and S. Haddad. Interrupt Timed Automata. In *Proc. of the 12th Int. Conf. on Foundations of Software Science and Computation Structures (FoSSaCS'09)*, volume 5504 of *LNCS*, pages 197–211. Springer, 2009.

[6] B. Bérard and S. Haddad. Interrupt Timed Automata: A step further. Technical Report LSV-09-1, Lab. Specification and Verification, ENS de Cachan, Cachan, France, Jan. 2009. 24 pages.

[7] B. Bérard, S. Haddad, and M. Sassolas. Verification on Interrupt Timed Automata. Technical Report LSV-09-16, Lab. Specification and Verification, ENS de Cachan, Cachan, France, July 2009. 27 pages.

[8] P. Bouyer, F. Chevalier, and N. Markey. On the expressiveness of TPTL and MTL. In *Proc. 25th Conf. on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'05)*, volume 3821 of *LNCS*, pages 432–443. Springer, Dec. 2005.

[9] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. KRONOS: A Model-Checking Tool for Real-Time Systems. In *FTRTFT*, pages 298–302, 1998.

[10] F. Cassez and K. G. Larsen. The impressive power of stopwatches. In *Proc. of concur 2000: concurrency theory*, pages 138–152. Springer, 1999.

[11] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking. In *Proc. International Conference on Computer-Aided Verification (CAV 2002)*, volume 2404 of *LNCS*, pages 241–268. Springer, 2002.

[12] A. David. *Hierarchical Modeling and Analysis of Timed Systems*. PhD thesis, Uppsala University, November 2003.

[13] E. A. Emerson and J. Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. In *Proc. 14th annual ACM Symp. on Theory of Computing (Stoc'82)*, pages 169–180. ACM, 1982.

[14] E. Fersman, P. Krcal, P. Pettersson, and W. Yi. Task automata: schedulability, decidability and undecidability. *Inf. Comput.*, 205(8):1149–1172, 2007.

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

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

[17] J. McManis and P. Varaiya. Suspension automata: a decidable class of hybrid automata. In *Proc. 6th Int. Conf. Computer Aided Verification (CAV'94)*, pages 105–117. Springer, 1994.

[18] M. L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967.

[19] J.-P. Queille and J. Sifakis. Specification and verification of concurrent systems in cesar. In *Proc. of the 5th colloquium on international symposium on programming*, pages 337–351, London, UK, 1982. Springer-Verlag.

[20] C. Roos, T. Terlaky, and J.-P. Vial. *Theory and Algorithms for Linear Optimization. An Interior Point Approach*. Wiley-Interscience, John Wiley & Sons Ltd, 1997.