Real Time Properties
for Interrupt Timed Automata∗

Béatrice Bérard† †Université Pierre & Marie Curie,
Serge Haddad‡ †École Normale Supérieure de Cachan,
Mathieu Sassolas† 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

∗Work partially supported by projects CoChaT (Digiteo 2009-27HD) and DOTS (ANR-06-SEIT-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 TCTLc over ITA is undecidable and Section 4 presents model checking procedures for two fragments of TCTLc.

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 \( a_x \)'s are in \( \mathbb{Q} \). We denote by \( C(X) \) the set of constraints obtained by conjunctions of atomic propositions of the form \( C \gg 0 \), where \( C \) is a linear expression over \( X \) and \( \gg \subseteq \{>,\geq,=,\leq,\ll\} \). The subset \( C_0(X) \) of \( C(X) \) contains constraints of the form \( x + b \gg 0 \). 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 \( 0 \) 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 \( \varphi \). 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.

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 \( A = (\Sigma, AP, Q, q_0, F, pol, X, \lambda, lab, \Delta) \), where:

- \( \Sigma \) 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,
- \( pol : Q \rightarrow \{\text{Lazy, Urgent, Delayed}\} \) is the timing policy of states,
- \( X = \{x_1, \ldots, x_n\} \) consists of \( n \) interrupt clocks,
- the mapping \( \lambda : Q \rightarrow \{1, \ldots, n\} \) associates with each state its level, and \( lab : Q \rightarrow 2^{AP} \) 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 \( \Delta \) be a transition with \( k = \lambda(q) \) and \( k' = \lambda(q') \). The guard \( \varphi \) contains only clocks from levels less than or equal to \( k \): it is a conjunction of constraints of the form \( \sum_{j=1}^{k} a_j x_j + b \gg 0 \). The update \( u \) is of the form \( \lambda_{i=1}^{k} x_i := C_i \) 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 \( A \) 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 \( \lambda_{i=1}^{n} x_i := C_i \) with \( C_i \) of the form \( \sum_{j=1}^{k-1} a_j x_j + b \) or \( C_i = x_i \) if \( k < i \) and \( C_i = 0 \) if \( i < k \). Thus, in an ITA_..., the only non trivial update (i.e.

| lev. 4 | \( x_4 := 0 \) |
| lev. 3 | \( x_3 := 0 \), \( x_2 := 0 \) |
| lev. 2 | |
| lev. 1 | \( x_1 := 0 \) |

Figure 1. Interrupt levels and clocks in an ITA.
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 \( A \) is defined by the transition system \( T_A = (S, s_0, \rightarrow) \). 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 \((q_0, 0, \bot)\). 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 \( d \geq 0 \) 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 \( T_A \) in the same configuration. When \( \text{pol}(q) = \text{Urgent} \), only time steps of duration \( 0 \) are allowed from \( q \).

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

An ITA \( A_1 \) 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).

![Figure 2](image)

**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 \( A = (\Sigma, Q, q_0, F, X, \Delta) \), where \( \Sigma, Q, q_0, F, \) and \( X \) are defined as in an ITA and the set of transition is \( \Delta \subseteq Q \times C_0(X) \times Q \times \mathbb{R}^X \times U_0(X) \times Q \), with guards in \( C_0(X) \) and updates in \( U_0(X) \).

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 \( A \) 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 \( a_1, a_2, \ldots, a_n \), we say that the word \( (a_1, d_1)(a_2, d_2)\ldots(a_n, d_n) \) (where \( \epsilon \) actions are removed) is accepted by \( A \). The set \( \mathcal{L}(A) \) contains the timed words accepted by \( A \). 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 \( \rho \), we denote by \( \rho_p \) the strict order on positions and for position \( \pi \) along \( \rho \), the corresponding configuration is denoted by \( \pi_\rho \).

**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-EXPSACE 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 TCTLc.

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 formulas clocks [16]. Model checking of timed automata 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 TCTLc are defined by the following grammar:

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

\[
A \psi U \psi \mid E \psi U \psi \mid \psi \land \psi \mid \neg \psi
\]

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 \(\infty \in \{\geq, \succ, =, \leq, <\}\).

Let \(A = (\Sigma, AP, Q, q_0, F, \text{pol}, X, \lambda, \text{lab}, \Delta)\) 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 TCTLc are interpreted over extended configurations of the form \((q, v, \beta, w)\), 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 \(v\) becomes \(v + d\) in a time step of delay \(d\) and is unchanged in a discrete step. We denote by \(E \text{exec}(s, w)\) the set of maximal runs starting from \((s, w)\).

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

\[
(q, v, \beta, w) \models p \iff p \in \text{lab}(q)
\]

\[
(q, v, \beta, w) \models y + b \succ 0 \iff w \models y + b \succ 0
\]

\[
(q, v, \beta, w) \models \sum_{i \geq 1} a_i \cdot x_i + b \succ 0 \iff w \models \sum_{i \geq 1} a_i \cdot x_i + b \succ 0
\]

and inductively:

\[
(s, w) \models y.\psi \iff y \in Y \text{ and } (q, v, w[y := 0]) \models \psi
\]

\[
(s, w) \models A \varphi U \psi \text{ iff } \forall \rho \in E \text{exec}(s, w), \rho \models \varphi U \psi
\]

\[
(s, w) \models E \varphi U \psi \text{ iff } \exists \rho \in E \text{exec}(s, w) \text{ s.t. } \rho \models \varphi U \psi
\]

with \(\rho \models \varphi U \psi\) iff there is a position \(\pi \in \rho\) s.t. \(\pi \models \psi\)

and \(\forall \pi' <_D \pi, \pi' \models \varphi \lor \psi\)

the cases for boolean operators are omitted.

3.2 Undecidability of TCTLc model checking.

We now prove that model checking TCTLc over ITA is undecidable. More precisely, let TCTLc\textsuperscript{ext} be the fragment of TCTLc containing only formula clocks, we have:

Theorem 1. Model checking TCTLc\textsuperscript{ext} over ITA is undecidable.

The first step of the proof is the construction of automaton \(A_M\) as a synchronized product between an interrupt timed automaton and a timed automaton, to simulate a two counter machine \(M\). In the second step, a TCTLc 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 \(M\) consists of a finite sequence of labeled instructions \(L\), which handle two counters \(c\) and \(d\), and ends at a special instruction with label \(\text{Halt}\). The other instructions have one of the two forms below, where \(e \in \{c, d\}\) represents one of the two counters:

- \(e := e + 1; \text{goto } \ell'\)

- if \(e > 0\) then \((e := e - 1; \text{goto } \ell')\) else goto \(\ell''\)

\(^1\)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: \((\ell_0, 0, 0)(\ell_1, n_1, p_1) \ldots (\ell_i, n_i, p_i) \ldots\), where \(n_i\) and \(p_i\) are the respective counter values and \(\ell_i\) is the label, after the \(i\)th instruction. The problem of termination for such a machine (“is the \textit{Halt} label reached?”) is known to be undecidable [18].

The automaton \(A_M = \langle \Sigma, AP, Q, q_0, F, pol, X \cup Y, \lambda, lab, \Delta \rangle\) is described by a (possibly infinite) sequence of configurations: \((\ell_0, 0, 0)\langle \ell_1, n_1, p_1) \ldots (\ell_i, n_i, p_i) \ldots\rangle\), \(q_0 = \ell_0\) (the initial instruction of \(M\) and \(F = \{\text{Halt}\}\).

\(\text{pol} : Q \rightarrow \{\text{Urgent, Lazy, Delayed}\}\) is such that \(\text{pol}(q) = \text{Urgent}\) iff either \(q \in L \) or \(q = (\ell, q_2, \omega)\), and \(\text{pol}(q) = \text{Lazy}\) in most other cases: some states \(\ell, k_1, \omega\) are \textit{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.

\(\lambda : Q \rightarrow \{1, 2, 3\}\) 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.

\(\text{lab}\) will be defined in the second step of the proof.

\(\Delta\) is defined through basic modules in the sequel.

The transitions of \(A_M\) are built within small modules, each one corresponding to one instruction of \(M\). The value \(n\) of \(c\) (resp. \(p\) of \(d\)) in a state of \(L\) is encoded by the value \(1 - \frac{n}{k}\) (resp. \(1 - \frac{p}{d}\) of \(y_c\)).

The idea behind this construction is that for any standard clock \(y\), it is possible to mimic the copy of the clock \(k-y\) 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 \(\{x_1, \ldots, x_i\}\) in \(y\).

For instance, consider an instruction labeled by \(\ell\) incrementing \(c\) then going to \(\ell'\), with the respective values \(n\) of \(c\) and \(p\) of \(d\), from a configuration where \(n \geq p\). The corresponding module \(A_{c \geq p}(\ell, \ell')\) is depicted on Figure 3. In this module, interrupt clock \(x_1\) is used to record the value \(\frac{1}{2}\) while \(x_2\) keeps the value \(\frac{1}{2}\). Assuming that \(y_c = 1 - \frac{1}{k}\), \(y_d = 1 - \frac{1}{d}\) and \(x_1 = 0\) in state \((\ell, r_1, >)\), the unique run in \(A_{c \geq d}(\ell, \ell')\) will end in state \(\ell'\) with \(y_c = 1 - \frac{1}{\ell'k}\) and \(y_d = 1 - \frac{1}{\ell'd}\).

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 \(\frac{1}{2}\) in \(x_1\) and \(\frac{1}{2}\) 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 \(\ell\) 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 \(\ell'\) instruction is taken. Note that only one of the branches can be taken until the end of the module2. Instructions involving \(d\) are handled in a symmetrical way.

\(A_M\) is obtained by joining the modules described above through the states of \(L\). The automaton \(A_M\) can actually be viewed as the product of an ITA \(I\) and a TA \(T\), 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 \(A_M\).

Note that another notion of synchronized product between ITA and TA leads to the class \(\mathcal{ITA}^+\) where reachability is decidable [5].

\footnote{State policies are used to treat the special cases, e.g. \(y_c = y_d = 0\).}
Second step. To prove Theorem 1, we build from the automaton $A_M$ above a formula $\varphi$ in TCTL$_c$ simulating the TA $T$, so that the ITA $T$ satisfies $\varphi$ iff $A_M$ terminates. Formula $\varphi$ expresses that (1) there is a run in $T$ reaching the $\text{halt}$ state, and (2) for each module of $T$, this run satisfies the constraints on the clocks $y_c$ and $yd$ of $T$.

To prove Theorem 1, we build from the automaton $A_M$ above a formula $\varphi$ in TCTL$_c$ simulating the TA $T$, so that the ITA $T$ satisfies $\varphi$ iff $A_M$ terminates. Formula $\varphi$ expresses that (1) there is a run in $T$ reaching the $\text{halt}$ state, and (2) for each module of $T$, this run satisfies the constraints on the clocks $y_c$ and $yd$ of $T$.

The full proofs that the above construction is correct ($\varphi$ in $T$ halts iff $A_M$ reaches the $\text{halt}$ 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 TCTL$_c^{\text{int}}$

In this section we consider formulas with only model clocks, the corresponding fragment being denoted by TCTL$_c^{\text{int}}$. For example property P1 in the introduction is expressed by $A x_2 \leq 3 \text{U safe}$. 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 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 $\varphi$ be a formula in TCTL$_c^{\text{int}}$ and $A$ 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 $C = \sum_{i=1}^{k} a_i x_i + b$, if $a_k = 0$, then $\text{norm}(C, k) = \sum_{i=1}^{k-1} a_i x_i + b$, otherwise $\text{norm}(C, k) = x_k + \sum_{i=1}^{k-1} a_i x_i + b / a_k$.

- 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 $\varphi$, we add the following step: For each comparison $C \cong 0$ in $\varphi$, and for each $k$, with $\text{norm}(C, k) = \alpha x_k + \sum_{i<k} a_i x_i + b$ with $\alpha \in \{0, 1\}$, we 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 $E_q$.  
2. Let $q \xrightarrow{\varphi,a,u} q'$ with $\lambda(q') \geq k$ and $\lambda(q) < k$. If $C' \in E_k$, we compute $C' = \text{norm}(C[u] - C'[u], \lambda(q)).$ If $C'' = \alpha x_k + \sum_{i<k} a_i x_i + b$ with $\alpha \in \{0, 1\}$, then we add $- \sum_{i<k} a_i x_i - b$ to $E_{\lambda(q')}$.

The proof of termination for this construction is similar to the one in [5].

Consider the ITA $A_1$ (Figure 2(a)) and the formula $\varphi_1 = E \cong_0 (q_1 \land (x_2 > x_1))$. 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 $x_1 + 2 x_2 = 2$ 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 1 is added to $E_2$ since $x_2 - x_1 > 0$ appears in $\varphi_1$. After iteration, we obtain $E_1 = \{x_1, 0, 1, \frac{3}{2}, 2\}$ and $E_2 = \{x_2, 0, - \frac{1}{2} x_1 + 1, x_1\}$. Remark that knowing the order between $x_1$ and $\frac{3}{2}$ 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 $G_A$ whose set of configurations are the classes $R = \{(q, \leq_k) \mid \forall k \leq \lambda(q)\}$, where $q$ is a state and $\leq_k$ is a total preorder over $E_k$. The class $R$ describes the set of values $[R] = \{(q, v) \mid \forall k \leq \lambda(q) \forall (g, h) \in E_k, v \leq h[v] \text{ iff } g \leq_k h\}$. 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 $G_A$ is finite and time abstract bisimilar to $\mathcal{T}_A$. Moreover, the truth value of each comparison $C = \sum_{i\geq 1} a_i \cdot x_i + b \gg 0$ appearing in $\varphi$ can be set for each class $R$. Indeed, since for every $k$, both $0$ and $\sum_{i\geq 1} a_i \cdot x_i + b$ are in the set of expressions $E_k$, the truth value of $C \gg 0$ does not change inside a class. Therefore, introducing a fresh propositional variable $q_C$ for the constraint $C \gg 0$, each class $R$ can be labeled with a truth value for each $q_C$. Deciding the truth value of $\varphi$ can then be done by a classical CTL model-checking algorithm on $G_A$.

On the example, we obtain the states in which $q_1 \land (x_2 > x_1)$ is true and conclude that $q_1$ is true on $A_1$. The complexity of the procedure is obtained by bounding the number of expressions for each level $k$ by $\max(2, |\Delta| + |\varphi|)\cdot (n-k+1)+1$, 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 $\text{TCTL}_c$ 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 $\text{TCTL}_c$ 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. $\text{TCTL}_p$ is the set of formulas where satisfaction of an until modality over propositions can be parameterized by a time interval. Formulas of $\text{TCTL}_p$ are defined by the following grammar:

$$
\varphi_p := p \varphi_p \land \varphi_p \land \neg \varphi_p \quad \text{and} \quad \psi := \varphi_p[A \varphi_p \Upsilon \varphi_p[E \varphi_p \Upsilon \varphi_p] \land \psi] \land \neg \psi
$$

where $p \in AP$ is an atomic proposition, $c \in \mathbb{Q}^*$, and $\gg \in \{\geq, \leq, \prec\}$ is a comparison operator. This logic is indeed a subset of $\text{TTCL}_p$, with only one formula clock since a formula, say $A p U_{\geq a} r$, can be rewritten as $y(A p U (r \land (y > a)))$. Properties P2 and P3 from introduction are expressed respectively as $A \neg \varphi U_{\geq 50} \top$ and $\top U_{\leq 7} \$.
Sketch. We start by noticing that formula $A p \cup U_{> a} r$ is true on a configuration of an ITA. If all the following conditions hold for paths starting in this configuration:

- all paths do satisfy $p \cup r$,
- 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