Béatrice Bérard
Serge Haddad
Mathieu Sassolas

Verification on Interrupt Timed Automata

Research Report LSV-09-16

July 2009

Laboratoire Spécification et Vérification

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

Ecole Normale Supérieure de Cachan
61, avenue du Président Wilson
94235 Cachan Cedex France

Verification on Interrupt Timed Automata

Béatrice Bérard[1], Serge Haddad[2], Mathieu Sassolas[1]

[1] Université Pierre & Marie Curie, LIP6/MoVe, CNRS UMR 7606, Paris, France
E-mail: {Beatrice.Berard, Mathieu.Sassolas}@lip6.fr

[2] Ecole Normale Supérieure de Cachan, LSV, CNRS UMR 8643, Cachan, France
E-mail: Serge.Haddad@lsv.ens-cachan.fr

Abstract. The class of Interrupt Timed Automata (ITA) has been introduced to model multi-task systems with interruptions in a single processor environment. This is a subclass of hybrid automata in which real valued variables consist of a restricted type of stopwatches (variables with rate 0 or 1) organized along levels. While reachability is undecidable with usual stopwatches, it was proved that this problem is decidable in ITA and that untimed languages of ITA are effectively regular. Here we investigate the problem of model checking timed extensions of CTL over ITA and show in contrast that this problem is undecidable. On the other hand, we prove that model checking is decidable for two relevant fragments of this timed logic: (1) the first one where formula contain only model clocks and (2) the second one where formulas have a single external clock.

1 Introduction

Context. Hybrid automata [11] are finite automata equipped with real valued variables, which evolve continuously according to some differential equations and can be tested and updated when discrete transitions are fired. This model is very expressive but it was proved that, even when the only variables are stopwatches, with rate equal to 0 or 1, reachability is undecidable [7]. On the other hand, restricting the variables to clocks, with rate 1, yields the well known model of Timed Automata (TA) where several verification question are decidable, in particular model checking TCTL [1], a timed extension of the temporal logic CTL [9,13]. Various classes have then been proposed between timed and hybrid automata, with the aim to keep a high expressive power together with decidable verification problems. The case of stopwatches is of particular interest since it allows to model time accumulation which is useful for instance in scheduling problems. The model of Interrupt Timed Automata (ITA) was introduced in [4] to describe multi-task systems with interruptions

in a single processor environment. An ITA is organized into interrupt levels, with a single clock active in this level, all clocks in lower levels being suspended and the clocks from higher levels not yet defined. As proved in [4], untiming languages accepted by ITA yields regular languages with the effective construction of a class graph generalizing the region automaton from [2,3]. Thus, ITA is one of the few models with stopwatches where reachability is decidable. Moreover, the complexity of reachability is improved by applying a linear programming technique to a syntactic restriction of ITA, with the same expressive power [5].

Contribution. In this work, we investigate other verification problems than reachability for ITA. It results from the class graph construction that model checking untimed properties from CTL or LTL is possible on ITA. Concerning timed logics, we introduce a timed extension TCTL_c of CTL, where formulas involve model clocks as well as external clocks. This logic is a variant of the one proposed in [10], also studied later in [6] from the expressivity point of view, and model checking this logic has been proved decidable for timed automata, with a construction similar to the region automaton. In contrast, we prove here that model checking TCTL_c over ITA is undecidable. However, we propose restrictions for which decidability procedures can be found. In the first one, only model clocks are involved and 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 restriction in which it is possible to reason on minimal or maximal delays. In this case, the decidability procedure relies on a linear programming technique, with a complexity in 2-NEXPTIME (NP if the number of clocks is fixed).

Outline. Section 2 gives definitions of ITA and the timed logics TCTL_c. 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 and Timed Logics

Notations. The sets of natural numbers, rational numbers 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 and $\bowtie \in \{>, \geq, =, \leq, <\}$ is a comparison operator. The

subset $\mathcal{C}_0(X)$ of $\mathcal{C}(X)$ contains constraints of the form $x + b \bowtie 0$. An update is a conjunction of affectations of the form $x := C$ for a clock x and a linear expression C. The set of all updates over X is written $\mathcal{U}(X)$ and $\mathcal{U}_0(X)$ denotes the subset of $\mathcal{U}(X)$ where the affectations are either 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 assigning the value 0 to 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 and an update u, 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 [4], to model multi-task systems with interruptions, in a single processor environment. 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 containing value 0. We give here a slightly more general definition as in [5], where states also have a timing policy which indicates whether time may or must elapse in a state, with lazy as default policy. We finally add a mapping labeling states with atomic propositions in a set AP, in view of defining logic formulas interpreted on these automata.

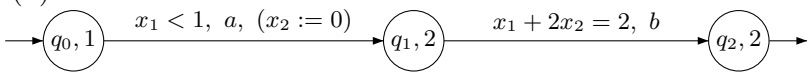
Definition 1. *An* interrupt timed automaton *is a tuple* $\mathcal{A} = \langle \Sigma, AP, Q, q_0, pol, F, X, \lambda, lab, \Delta \rangle$, *where:*

- 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,*
- *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. 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*

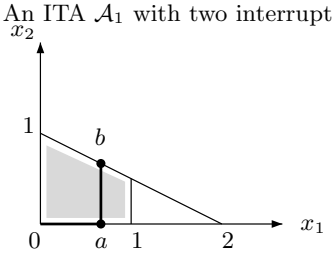
form $\sum_{j=1}^{k} a_j x_j + b \bowtie 0$. *The update* u *is of the form* $\wedge_{i=1}^{n} x_i := C_i$ *with:*

- *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* $C_i = x_i$ *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* $1 \leq i \leq k$, $C_i = 0$ *if* $k < i \leq k'$ *and* $C_i = x_i$ *if* $i > k'$.

An ITA $\mathcal{A}_1$ is depicted in Fig. 1(a), with two interrupt levels (and two interrupt clocks), with a geometric view of a possible trajectory in Fig. 1(b).



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



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

Fig. 1. An example of ITA and a possible execution.

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 $k < i \leq k'$ and $C_i = x_i$ otherwise. In words, in an ITA$_-$, the only possible relevant update (i.e. not enforced by the semantics of the model) is an update of the clock of the current level.

Any ITA can be exponentially translated into an equivalent ITA$_-$ [5] but with the same set of clocks.

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* $(q_0, \mathbf{0}, \bot)$. *A configuration* (q, v, β) *is accepting if* q *is in* F. *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 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. When $pol(q) = Urgent$, time steps from q are forbidden.*

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

For the sake of completeness, we briefly recall the classical model of timed automata (TA) [3]. Note that in this case, there is no need to include time policies, which can be enforced with 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,*
- *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, and which is maximal: either it is an infinite path or it is a finite path such that no discrete step is possible from the last configuration. We use the notion of (totally ordered) positions along a run [10]: for a run ρ, we denote by $<_\rho$ the strict order on positions and for position $\pi \in \rho$, the corresponding configuration is denoted by s_π.

Timed logic TCTL_c. At least two different timed extensions of the branching time logic CTL [9,13,8] have been proposed. The first one [1] adds subscripts to the U operator while the second one considers formula clocks [10]. Model checking timed automata was proved decidable in both cases and compared expressiveness has been revisited later on [6]. We use here a variant of the (more expressive) second type, using a set X of model clocks (from the automaton on which the formulas will be evaluated) and a disjoint set Y of formula clocks.

Definition 4. *The timed logic TCTL_c is defined by the following grammar:*

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

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 $(a_i)_{i\geq 1}$ has finite domain, and $\bowtie \in \{>, \geq, =, \leq, <\}$.

Let $\mathcal{A} = \langle \Sigma, AP, Q, q_0, pol, F, X, \lambda, lab, \Delta \rangle$ be an ITA 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[3]. The notions of 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 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:

$$\begin{array}{ll}
(s, w) \models y.\psi & \text{iff } y \in Y \text{ and } (q, v, w[y := 0]) \models \psi \\
(s, w) \models \mathsf{A}\,\varphi\,\mathsf{U}\,\psi & \text{iff } \forall \rho \in Exec(s, w),\ \rho \models \varphi\,\mathsf{U}\,\psi \\
(s, w) \models \mathsf{E}\,\varphi\,\mathsf{U}\,\psi & \text{iff } \exists \rho \in Exec(s, w) \text{ s. t. } \rho \models \varphi\,\mathsf{U}\,\psi \\
\text{with} & \\
\rho \models \varphi\,\mathsf{U}\,\psi & \text{iff there is a position } \pi \in \rho \text{ s. t. } s_\pi \models \psi \\
 & \quad \text{and } \forall \pi' <_\rho \pi,\ s_{\pi'} \models \varphi \vee \psi
\end{array}$$

the cases for boolean operators are omitted.

3 Model checking TCTL_c over ITA is undecidable

While both reachability and TCTL_c model checking are decidable in the class of timed automata, we prove in this section that model checking

[3] Here the boolean value appears in the configuration in order to be consistent with the semantics of ITA but is not actually used. One could imagine enriching the logic to take advantage of this boolean, to express for example that a run lets some time elapse in a certain state.

TCTL_c over ITA is undecidable. The first step of the proof is the construction of a synchronized product between an interrupt timed automaton and a timed automaton, to simulate a two counter machine. This proves that reachability is undecidable in the class containing products of an ITA and a TA. In the second step, a TCTL_c formula is built to simulate the timed automaton part of the product.

Reachability on ITA$\times$TA. We consider the class $\mathrm{ITA}_3\times\mathrm{TA}_2$ of automata built as a synchronized product between an interrupt timed automaton with 3 clocks and a timed automaton with 2 clocks 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.

Proposition 1. *Reachability is undecidable in the class $ITA_3\times TA_2$.*

Proof. We build an automaton in $\mathrm{ITA}_3\times\mathrm{TA}_2$ which simulates a deterministic two counters machine. Recall that such a machine $\mathcal{M}$ consists of a finite sequence of labeled instructions L, which handles 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 behaviour of the machine is described by a (possibly infinite) sequence of configurations: $\langle \ell_0, 0, 0\rangle\langle \ell_1, n_1, p_1\rangle \ldots \langle \ell_i, n_i, p_i\rangle \ldots$, 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 [12].

The automaton $\mathcal{A}_\mathcal{M} = \langle \Sigma, AP, Q, q_0, pol, F, X \cup Y, \lambda, \Delta\rangle$ is built to reach a final location if and only if $\mathcal{M}$ stops. It is defined as follows:

- Σ consists of one letter per transition, AP will be defined in the next step.
- $Q = L \cup (L \times \{k_0\}) \cup (L \times \{k_1, k_2, r_1, \ldots, r_5\} \times \{>, <\})$.
- $q_0 = \ell_0$ (the initial instruction of $\mathcal{M}$).
- $pol : Q \to \{Urgent, Lazy, Delayed\}$ 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 Fig. 3.
- $F = \{Halt\}$.

- $X = \{x_1, x_2, x_3\}$ 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 in 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 will be 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 $1 - \frac{1}{2^n}$ of clock y_c (resp. $1 - \frac{1}{2^p}$ of y_d) will encode the value n of c (resp p of d) in states of L.

The idea behind this construction is that for any standard clock y, it is possible to "copy" the value of $k - y$ in an interrrupt 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 $x_{i+1} = x_i$, 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 labelled by ℓ incrementing c then going to ℓ', with the respective values n of c and p of d, from a configuration where $c \geq d$. The corresponding module $\mathcal{A}^{c++}_{c \geq d}(\ell, \ell')$ is depicted on Fig. 2. 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 1.

The module on Fig. 2 can be adapted for the case of decrementing c by just changing the linear expressions in guards for x_3, provided 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. 3. Figure 3(a) represents the case when at label ℓ we have an increment of c whereas Fig. 3(b) 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

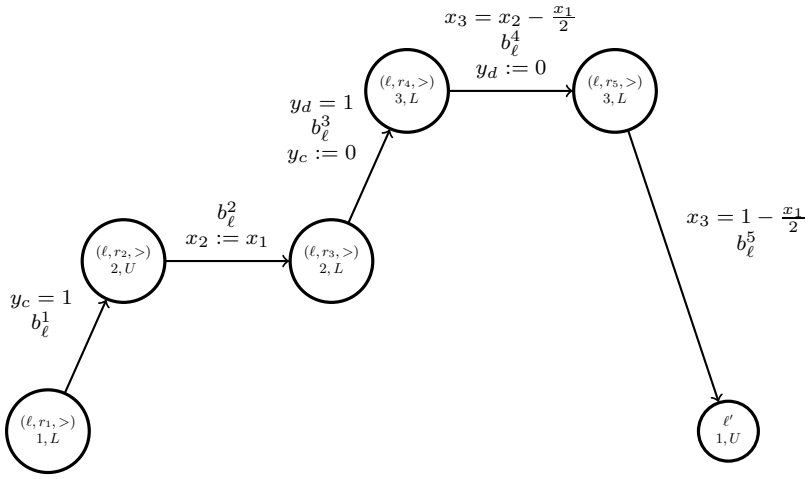


Fig. 2. Module $\mathcal{A}_{c\geq d}^{c++}(\ell, \ell')$ incrementing the value of c when $c \geq d$.

$(\ell, \mathbf{r_1}, >)$	$(\ell, \mathbf{r_1}, >)$	$(\ell, \mathbf{r_2}, >)$	$(\ell, \mathbf{r_3}, >)$
$y_c = 1 - \frac{1}{2^n}$	$y_c = 1$	$y_c = 1$	$y_c = 1$
$y_d = 1 - \frac{1}{2^p}$	$y_d = 1 - \frac{1}{2^p} + \frac{1}{2^n}$	$y_d = 1 - \frac{1}{2^p} + \frac{1}{2^n}$	$y_d = 1 - \frac{1}{2^p} + \frac{1}{2^n}$
$x_1 = 0$	$x_1 = \frac{1}{2^n}$	$x_1 = \frac{1}{2^n}$	$x_1 = \frac{1}{2^n}$
$x_2 = 0$	$x_2 = 0$	$x_2 = 0$	$x_2 = \frac{1}{2^n}$
$x_3 = 0$	$x_3 = 0$	$x_3 = 0$	$x_3 = 0$

$\xrightarrow{\frac{1}{2^n}}$ $\xrightarrow{b_\ell^1}$ $\xrightarrow{b_\ell^2}$ $\xrightarrow{\frac{1}{2^p} - \frac{1}{2^n}}$ $\cdots$

$\cdots \xrightarrow{\frac{1}{2^p} - \frac{1}{2^n}}$

$(\ell, \mathbf{r_3}, >)$	$(\ell, \mathbf{r_4}, >)$	$(\ell, \mathbf{r_4}, >)$
$y_c = 1 + \frac{1}{2^p} - \frac{1}{2^n}$	$y_c = 0$	$y_c = \frac{1}{2^p} - \frac{1}{2^{n+1}}$
$y_d = 1$	$y_d = 1$	$y_d = 1 + \frac{1}{2^p} - \frac{1}{2^{n+1}}$
$x_1 = \frac{1}{2^n}$	$x_1 = \frac{1}{2^n}$	$x_1 = \frac{1}{2^n}$
$x_2 = \frac{1}{2^p}$	$x_2 = \frac{1}{2^p}$	$x_2 = \frac{1}{2^p}$
$x_3 = 0$	$x_3 = 0$	$x_3 = \frac{1}{2^p} - \frac{1}{2^{n+1}}$

$\xrightarrow{b_\ell^3}$ $\xrightarrow{\frac{1}{2^p} - \frac{1}{2^{n+1}}}$ $\xrightarrow{b_\ell^4} \cdots$

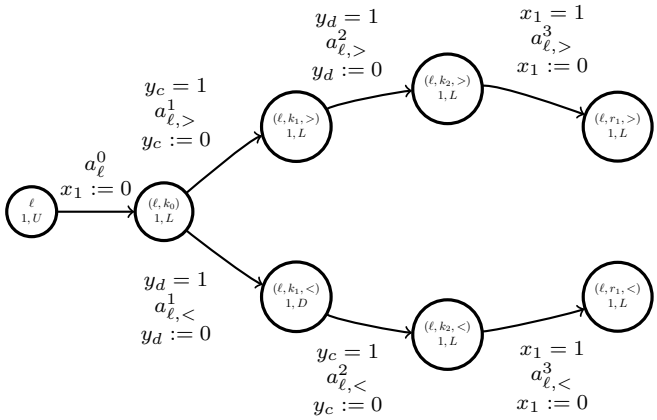
$\cdots \xrightarrow{b_\ell^4}$

$(\ell, \mathbf{r_5}, >)$	$(\ell, \mathbf{r_5}, >)$	ℓ'
$y_c = \frac{1}{2^p} - \frac{1}{2^{n+1}}$	$y_c = 1 - \frac{1}{2^{n+1}}$	$y_c = 1 - \frac{1}{2^{n+1}}$
$y_d = 0$	$y_d = 1 - \frac{1}{2^p}$	$y_d = 1 - \frac{1}{2^p}$
$x_1 = \frac{1}{2^n}$	$x_1 = \frac{1}{2^n}$	$x_1 = \frac{1}{2^n}$
$x_2 = \frac{1}{2^p}$	$x_2 = \frac{1}{2^p}$	$x_2 = 0$
$x_3 = \frac{1}{2^p} - \frac{1}{2^{n+1}}$	$x_3 = 1 - \frac{1}{2^{n+1}}$	$x_3 = 0$

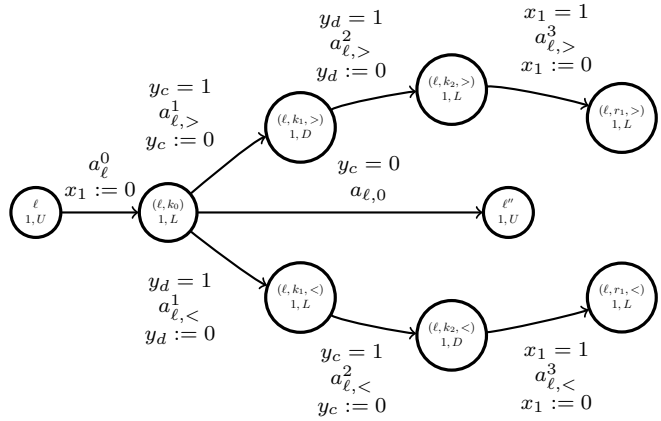
$\xrightarrow{1 - \frac{1}{2^p}}$ $\xrightarrow{b_\ell^5}$

Table 1. Clock values in the unique run of $\mathcal{A}_{c\geq d}^{c++}(\ell, \ell')$. Irrelevant values of interrupt clocks are greyed.

which branch of the *if* instruction is taken. Note that these modules are also deterministic with respect to time constraints. Instructions involving d are handled in a symmetrical way.



(a) Incrementing c



(b) Decrementing c

Fig. 3. Modules taking into account the order between the values of c and d.

To obtain automaton $\mathcal{A}_\mathcal{M}$, we join the modules described above through the states of L. The proof that the above construction is correct, *i.e.* that $\mathcal{A}_\mathcal{M}$ simulates $\mathcal{M}$ and therefore that $\mathcal{M}$ halts iff $\mathcal{A}_\mathcal{M}$ reaches the *Halt* state, is given in Appendix A.1.

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 abov 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 $\mathcal{A}_\mathcal{M}$. □

Undecidability of TCTL_c model checking. From the automaton $\mathcal{A}_\mathcal{M}$ of the above construction, we now build a formula φ in TCTL_c simulating the TA $\mathcal{T}$, so that the ITA $\mathcal{I}$ satisfies φ iff $\mathcal{M}$ terminates. More precisely, denoting by $\mathsf{TCTL}_c^{\text{ext}}$ the fragment of TCTL_c containing only formula clocks, we have:

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

Proof. As explained above, we split the automaton $\mathcal{A}_\mathcal{M}$ built in the proof of Proposition 1 into an ITA $\mathcal{I}$ and a TA $\mathcal{T}$, and we produce a $\mathsf{TCTL}_c^{\text{ext}}$ formula φ expressing the following:

- there is a run in $\mathcal{I}$ reaching the *Halt* state,
- for each module of $\mathcal{I}$, this run satisfies the constraints on the clocks y_c and y_d of $\mathcal{T}$.

Therefore, $\mathcal{I} \models \varphi$ iff $\mathcal{M}$ terminates. In this construction, all information about $\mathcal{M}$ is contained in $\mathcal{I}$ and φ does not depend on $\mathcal{M}$. The full proof is given in Appendix A.2. □

4 Decidable fragments

4.1 Checking $\mathsf{TCTL}_c^{\text{int}}$

In this section we prove that model-checking formulas without external clocks is decidable. We consider the fragment $\mathsf{TCTL}_c^{\text{int}}$ where only model cloks are used and adapt a class graph construction for untiming ITA, by 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^{int} on ITA can be done in 2-EXPSPACE, and in PSPACE when the number of clocks is fixed.*

Proof. Let φ be a formula in $\mathsf{TCTL}_c^{\text{int}}$ and $\mathcal{A}$ an ITA with n levels. Like in [4], the proof relies on the construction of a finite class graph, the first step being the computation of 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 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 $-\sum_{i<k} a_i x_i - b$ 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 [4].

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 by discrete and successor steps, whose details are developped 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 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 $\mathcal{G}_{\mathcal{A}}$. An example of this model checking procedure is shown in Appendix B.

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. □

4.2 Checking a fragment of TCTL

The decidability of model-checking TCTL_c formulas over ITA has been studied above for two cases:

- when there are 2 formula clocks, in which case the problem is undecidable (Theorem 1)
- 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 statisfaction of an *until* modality over propositions can be parameterized by a time interval. Formulas of TCTL_p are defined by the following grammar:

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

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 $\mathsf{A}\,p\,\mathsf{U}_{>a}q$, can be rewritten as $y.(\mathsf{A}\,p\,\mathsf{U}\,(r \wedge (y > a)))$. Since ITA can be translated into ITA_{-} (at exponential cost), the problem can be simplified by focusing on the subclass ITA_{-}. We prove that:

Theorem 3. *Model checking $TCTL_p$ on an ITA (resp. ITA_{-}) is decidable in 2-NEXPTIME (resp. NEXPTIME), and in both cases in NP if the number of clocks is fixed.*

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

Lemma 1. *a. 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.*

b. 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.*

Proof. The main idea underlying these procedures for cases 1 and 2 is to determine a maximal 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 [14], 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. The complete proofs for these cases are in Appendix A.3 and A.4. □

For formulas in case 3, a specific procedure can be avoided, since the result of case 2 can be reused:

Lemma 2. *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.*

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

Finally, we handle case 4, which is the most involved:

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.*

Proof. 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 pathes starting in this configuration:

– all pathes 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 the last condition. In the other cases, a counterexample will be a finite path, whereas here the counterexample is potentially infinite, which makes the proof more difficult. It can be found in Appendix A.5. □

5 Conclusion and future work

In this work we have shown that model checking TCTL formulas with explicit clocks on interrupt timed automata is undecidable. This places the class ITA in an intermediate position between hybrid automata, for which reachability is undecidable, and timed automata, for which model checking TCTL is decidable. We nevertheless propose specific model checking procedures for some subsets of this logic: namely (1) when no external clock is used in the formula, in which case the complexity is 2-EXPSPACE (PSPACE when the number of clocks is fixed), and (2) in a subset of TCTL with subscripts, in which case the complexity is in 2-NEXPTIME (NP when the number of clocks is fixed). We plan to refine this problem by addressing the case of formulas with internal clocks and only one external clock, which remains open.

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 seventeenth international 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. B Bérard and S Haddad. Interrupt Timed Automata. In *Proc. of the 12th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'09)*, volume 5504 of *Lecture Notes in Computer Science*, pages 197–211, York, GB, March 2009. Springer.
5. 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, January 2009. 24 pages.

6. P. Bouyer, F. Chevalier, and N. Markey. On the expressiveness of TPTL and MTL. In R. Ramanujam and Sandeep Sen, editors, *Proc. of the 25th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'05)*, volume 3821 of *Lecture Notes in Computer Science*, pages 432–443, Hyderabad, India, December 2005. Springer.
7. F. Cassez and K. G. Larsen. The impressive power of stopwatches. In *Proc. of concur 2000: concurrency theory*, pages 138–152. Springer, 1999.
8. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, 1986.
9. E. A. Emerson and J. Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. In *Proc. of the fourteenth annual ACM Symposium on Theory of Computing (Stoc'82)*, pages 169–180, New York, NY, USA, 1982. ACM.
10. 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.
11. O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In *In real-time: theory in practice, rex workshop, lncs 600*, pages 447–484. Springer-Verlag, 1992.
12. M. L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967.
13. 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.
14. C. Roos, T. Terlaky, and J.-Ph. Vial. *Theory and Algorithms for Linear Optimization. An Interior Point Approach*. Wiley-Interscience, John Wiley & Sons Ltd, 1997.

A Proof of selected properties

A.1 Correction of the construction of $\mathcal{A}_\mathcal{M}$ in the proof of Proposition 1

Let us prove that $\mathcal{A}_\mathcal{M}$ defined in Section 3 simulates exactly the two counter machine $\mathcal{M}$. Let $\langle \ell_0, 0, 0\rangle\langle \ell_1, n_1, p_1\rangle \dots \langle \ell_i, n_i, p_i\rangle \dots$ be a run of $\mathcal{M}$. We show that this run is simulated in $\mathcal{A}_\mathcal{M}$ by the run $\langle l_0, \mathbf{0}\rangle\rho_0\langle l_1, v_1\rangle\rho_1 \dots$ where ρ_i is either empty or a subrun through states in $\{(\ell_i, r_j, \bowtie) \mid j \in \{1, \dots, 5\}, \bowtie \in \{>, <\}\}$ (*i.e.* subruns in modules like $\mathcal{A}^{c++}_{c\geq d}$ of Fig. 2). 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 beginnning 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 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. 3(b). 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 $\mathcal{A}_\mathcal{M}$ simulates $\mathcal{M}$, and $\mathcal{M}$ halts iff $\mathcal{A}_\mathcal{M}$ reaches the *Halt* state.

A.2 Proof of Theorem 1

Let $\mathcal{M}$ be a two counter machine and let $\mathcal{A}_\mathcal{M} = \mathcal{I}||\mathcal{T}$ be the automaton built in the proof of Proposition 1, where $\mathcal{I}$ is an ITA with three clocks and $\mathcal{T}$ is a TA with two clocks, which will here be simulated by a formula. Let $AP = \{p, h, k_1, k_2, r_{23}, r_4, r_5\} \cup (\{r_1\} \times \{c, d\} \times \{++, --\} \times \{>, <\})$ be the set of atomic propositions. Proposition p is true in states labelled by $\ell \in L$. Proposition h is true only in the state labelled by *Halt*. Proposition k_i, for $i \in \{1, 2\}$, is true in states labelled by $(\ell, k_i, \bowtie)$. Proposition r_i, for $i \in \{4, 5\}$, is true in states labelled by $(\ell, r_i, \bowtie)$. Similarly, r_{23} is true in

states labelled by $(\ell, r_i, \bowtie)$, for $i \in 2, 3$. Proposition $(r_1, a, op, \bowtie)$ is true in state r_1 of the corresponding module. For example, $(r_1, c, ++, >)$ will be true in state $(\ell, r_1, >)$ if ℓ is an instruction of $\mathcal{M}$ incrementing counter c. Hence all these propositions will enable a $\mathrm{TCTL}_c^{\mathrm{ext}}$ formula to know in which state or in which kind of module the formula is evaluated. Let $\{y_c, y_d\}$ be two formula clocks and

$$\varphi_{\geq}^{c++} = \mathsf{A}\,((r_1, c, ++, >) \wedge (y_c < 1))\,\mathsf{U}\,(\mathsf{A}\,(y_c \geq 1 \wedge r_{23})$$

$$\mathsf{U}\,(y_d = 1 \wedge y_c.(\mathsf{A}\,r_4\,\mathsf{U}\,y_d.(\mathsf{A}\,r_5\,\mathsf{U}\,p))))$$

Formula $\varphi_{\geq}^{c++}$ is true if any run in the projection $\mathcal{I}_{c\geq d}^{c++}$ of a module $\mathcal{A}_{c\geq d}^{c++} = \mathcal{I}_{c\geq d}^{c++} || \mathcal{T}_{c\geq d}^{c++}$ respects the same constraints on clocks y_c and y_d as $\mathcal{A}_{c\geq d}^{c++}$ does. Other analogous formulas are created for the seven other types of modules. Let

$$\begin{aligned}\varphi_p^{c++} = p \Rightarrow [&(\mathsf{A}\,y_c < 1\,\mathsf{U}\,(y_c = 1 \wedge k_1 \wedge y_c.(\mathsf{A}\,y_d < 1\,\mathsf{U}\\ &\qquad (y_d = 1 \wedge k_2 \wedge y_d.(\mathsf{A}\,k_2\,\mathsf{U}\,\varphi_{\geq}^{c++})))))\\ \vee\ &(\mathsf{A}\,y_d < 1\,\mathsf{U}\,(y_d = 1 \wedge k_1 \wedge y_d.(\mathsf{A}\,y_c < 1\,\mathsf{U}\\ &\qquad (y_c = 1 \wedge k_2 \wedge y_c.(\mathsf{A}\,k_2\,\mathsf{U}\,\varphi_{<}^{c++})))))]\end{aligned}$$

and

$$\begin{aligned}\varphi_p^{c--} = p \Rightarrow [&(\mathsf{A}\,y_c < 1\,\mathsf{U}\,(y_c = 1 \wedge k_1 \wedge y_c.(\mathsf{A}\,y_d < 1\,\mathsf{U}\\ &\qquad (y_d = 1 \wedge k_2 \wedge y_d.(\mathsf{A}\,k_2\,\mathsf{U}\,\varphi_{>}^{c--})))))\\ \vee\ &(\mathsf{A}\,y_d < 1\,\mathsf{U}\,(y_d = 1 \wedge k_1 \wedge y_d.(\mathsf{A}\,y_c < 1\,\mathsf{U}\\ &\qquad (y_c = 1 \wedge k_2 \wedge y_c.(\mathsf{A}\,k_2\,\mathsf{U}\,\varphi_{\leq}^{c--})))))\\ \vee\ &(\mathsf{A}\,y_c = 0\,\mathsf{U}\,p)]\end{aligned}$$

Formula φ_p^{c++} will be true if the correct path is always taken in the modules that take into account the order between the values of the counters before entering the module that actually does the incrementation. Formula φ_p^{c--} is analogous, but with a third case for when the counter value is zero. Other similar formulas are created for the actions on the second counter d. Finally, let

$$\varphi = y_c.y_d.\mathsf{E} \bigwedge_{\substack{a\in\{c,d\}\\ *\in\{+,-\}}} \varphi_p^{a**}\,\mathsf{U}\,h$$

Formula φ states that there is a path reaching the halting state where all the formulas previously defined hold. If these formulas hold on a path ρ in $\mathcal{I}$, then ρ respects the same clock constraints on y_c and y_d as the corresponding run on $\mathcal{A}_{\mathcal{M}}$ does. Since $\mathcal{A}_{\mathcal{M}}$ exactly encodes the computations of $\mathcal{M}$, then φ is true in $\mathcal{I}$ if and only if $\mathcal{M}$ teminates.

A.3 Proof of Lemma 1.a (case 1)

This proof and the following rely on a counting lemma on ITA$_-$.

Lemma 4. *Let $\mathcal{A}$ be an ITA$_-$ with n clocks and $|\Delta|$ transitions, with $B = n^2(|\Delta|+2)^{3n}$. If ρ is a run in $\mathcal{A}$ of length greater than B, then:*

- *there exist a level k and two identical transitions e_i and e_j in ρ from a state of level k updating x_k, with only states of level $\geq k$ between e_i and e_j;*
- *if not, there exist a level k and two identical lazy or delayed transitions e_i and e_j in ρ, from a state of level k, leaving x_k unchanged, separated by states of level $\geq k$ and transitions leaving x_k unchanged;*
- *otherwise, there exist a level k and two identical urgent transitions e_i and e_j in ρ from a state of level k, separated by states of level $\geq k$ and urgent transitions leaving x_k unchanged.*

Proof. The proof consists of a counting argument and can be found in [5]. □

Suppose there is a run satisfying $p \,\mathsf{U}_{\leq \mathsf{a}}\, r$ on an ITA$_-$ $\mathcal{A}$, we consider a finite prefix $\rho = e_1 \dots e_h$, minimal in number of discrete transitions satisfying the formula. Remark that in this case the last transition of ρ reaches a state in which r is *true*. Suppose that $|\rho| > B = n^2(|\Delta|+2)^{3n}$. Then, one of the three cases of Lemma 4 happens:

- Either e_i updates x_k, in that case, the run $e_1 \dots e_i e_{j+1} \dots e_h$ is a smaller run satisfying $p \,\mathsf{U}_{\leq \mathsf{a}}\, r$, which is a contradiction.
- Or e_i does not update x_k then time elapses for x_k in the subrun $\rho' = e_{i+1} \dots e_j$. In this case, replacing this subrun in ρ by waiting the time elapsed for x_k yields a smaller run satisfying $p \,\mathsf{U}_{\leq \mathsf{a}}\, r$, which is also a contradiction.
- Or the subrun $e_{i+1} \dots e_j$ does not let time elapse for x_k, in which case deleting it in ρ does not affect the remainder and also yields a smaller run which still satisfies $p \,\mathsf{U}_{\leq \mathsf{a}}\, r$, again a contradiction.

The decision procedure is then obtained as explained in the main text on runs with length less than or equal to B.

A.4 Proof of Lemma 1.b (case 2)

Let $\mathcal{A}$ be an ITA$_-$ with n interrupt clocks and $|\Delta|$ transitions, and $B = n^2(|\Delta|+2)^{3n}$. From a run satisfying $p \,\mathsf{U}_{\geq \mathsf{a}}\, r$, we consider a minimal finite prefix $\rho = e_1 \dots e_h$ also satisfying the formula. If $h > 2B$, it means that one of the following cases of Lemma 4 happens:

– 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 for x_k between the two occurences e_i and e_j of e, or the transitions were all instantaneous for this clock. 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. On the other hand, if some time elapsed during this subrun then there is a run $\rho' = e'_1 \dots e'_{h'}$ such that:
  - $e_1 \dots e_j = e'_1 \dots e'_j$,
  - $e'_{j+1} \dots e'_{h'}$ satisfies $p \,\mathsf{U}\, r$,
  - $h' - j \leq B$.

  The last condition can be ensured by a reasonning of the same kind as in the proof of the first case of Lemma 1 (see Section A.3): a run which is too long can be reduced without changing the truth value of $p \,\mathsf{U}\, r$.
– The same transition e 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 $p \,\mathsf{U}_{\geq \mathsf{a}}\, q$ on this new smaller run, thus violating the minimality hypothesis.
– The same transition e appears twice, separated by urgent transitions. In that case, 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.

The decision procedure is again obtained by testing runs with length less than or equal to $2B + 1$: non deterministically guessing a path and building an associated linear program. 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 is greater than a. In case of failure of this first part of the procedure, it non deterministically guesses a path of length less than or equal to $B + 1$, 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,
– there is a transition e of level k appearing twice and separated by transitions of level higher than k,

– between these two occurences of e, time elapses for x_k. This last part can be checked with a linear program on the delays corresponding to this subrun.

From such a run one can obtain a run satisfying $p \,\mathsf{U}_{\geq \mathsf{a}}\, r$ by iterating the path between the two transitions e, hence the formula $\mathsf{E}\, p \,\mathsf{U}_{\geq \mathsf{a}}\, r$. The case of the formula $\mathsf{E}\, p \,\mathsf{U}_{> \mathsf{a}}\, r$ is handled similarly, the additional condition to the linear program being that the sum of delays should be strictly greater than a.

A.5 Proof of Lemma 3 (case 4)

We consider again an ITA$_-$ $\mathcal{A}$ with n interrupt clocks and $|\Delta|$ transitions and the bound $B = n^2(|\Delta| + 2)^{3n}$. First suppose there is a finite path of minimal length ρ for which there is a position where the time elapsed is less than a, from which r does not hold until $\neg p \wedge \neg r$ does. It can be shown that ρ contains at most B transitions. Indeed, if this is not the case then there is a transition e at level k appearing twice separated by a subrun σ containing only transitions of level $\geq k$ (see Lemma 4). Transition e can either reset the value of x_k, in which case σ can be simply deleted, or e does not change the value of x_k, in which case σ can be replaced by a time step of the same duration (possibly null). In both cases the fact that there is a position of global time less than a from which r does not hold until $\neg p \wedge \neg r$ does is not changed (the time of this position can only have been reduced). The new path being smaller, this violates the minimality condition.

Now suppose there is a finite maximal path of minimal length ρ satisfying $p \,\mathsf{U}\, r$ but for which there is a position where the time elapsed is less than a and from which r no longer holds. It can be shown by a technique similar to the one above that ρ contains at most B transitions.

Otherwise, suppose that all maximal paths ρ satisfying $p \,\mathsf{U}\, r$ and having a position p_a where time is less than a and after which r is no longer *true* are infinite. By the same reasoning as above, the minimal such position p_a can be assumed to appear after less than B transitions. For the infinite part, hence longer than B, in the suffix ρ' of ρ after position p_a, there is a transition e of level k repeated twice and separated by transitions of higher level. Two cases are possible.

(1) Either this transition updates x_k or the run between its two occurences does not let time elapse for k.
(2) For all such suffixes ρ', there is necessarily time elapsing in x_k between every two occurences of e separated by transitions of higher

level. Moreover, there is one of such subruns σ in which transitions are visited infinitely often. It can therefore be assumed that k is the minimal level traversed infinitely often by ρ'. Now recall the set of expressions used to build the class graph of $\mathcal{A}$ in proof of Theorem 2. Since the same transitions can be fired inifintely often, it means that from a certain point of the run, the ordering between the expressions of level k is constant. Moreover, all expressions except x_k will not change their value. Since some time has elapsed for x_k, it must have been because σ traverses a state of level k with delayed policy: otherwise the same run in which all these delays have been suppressed would also be possible, and a run satisfying condition (1) would exist.

In both cases, the path occuring before the first occurence of e we are considering, and the length of the path between the two occurences of e can both be bounded by B.

Therefore we consider the following decision procedure. It non deterministically guesses a maximal path in the ITA$_-$ whose length is less than or equal to the bound. This path should either not satisfy $p \,\mathsf{U}\, r$, or from some step (say the zth step), never satisfy r until the end of the run or a point where neither p nor r hold. In order to check that this path yields a run, it again 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, plus the condition that the zth transition considered before should occur at a time strictly less than a. If this procedure succeeds, then there is a finite counterexample to the formula $\mathsf{A}\, p \,\mathsf{U}_{\geq \mathsf{a}}\, r$, and the algorithm returns *false*. If this fails, it non deterministically guesses a path ρ of length less than or equal to twice the bound, in which from some step (say the zth step), r never holds, and which ends by a sub-path $e\sigma e$ where e is a transition of level k and σ a path containing states of level greater than or equal to k. Then the procedure is based on the previous case study.

1. If e updates x_k or $e\sigma$ does not let time elapse for x_k, then by repeating σe indefinitely, we would obtain an infinite path ρ' in which r never holds from a certain point on. By a linear program similar to above, it can be checked that ρ yields a run, and that the nth transition occured strictly before time a. Hence it is also possible to obtain a run from ρ', and there is an infinite counterexample so the algorithm returns *false*.

2. Otherwise, the linear program has to be tweaked in the following way: all delayed states of $e\sigma$ are supposed lazy while building the constraints for the end of the path: (1) it must execute in zero time and (2) all large guards involving variable x_k are changed in strict guards ($\leq$ becomes $<$, $\geq$ becomes $>$) (3) there is no guard with equality involving variable x_k. If ρ does yield a run, an infinite run could be obtained from it by repeating σe but staying in delayed states a time as small as needed in order not to change the evaluation of the guards. Therefore there is also an infinite counterexample, and the algorithm returns *false*.

If this procedure also fails, then the algorithm returns *true*.

In all cases above, the size of the linear programs is exponential w.r.t. the size of the ITA$_-$, which produces procedures in NEXPTIME (NP if the number of clocks is fixed).

B Example of model checking $\mathbf{TCTL}_c^{\mathbf{int}}$

Consider the ITA $\mathcal{A}_1$ (Fig. 1(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. 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:

- $x_1 - 0$ and $x_2 - 0$ yield no new expression.
- $x_2 - (-\frac{1}{2}x_1 + 1)$ and $0 - (-\frac{1}{2}x_1 + 1)$ 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. 4. 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 this state is reachable, hence proving that φ_1 is *true* on $\mathcal{A}_1$.

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

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

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

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

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

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

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

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

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

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

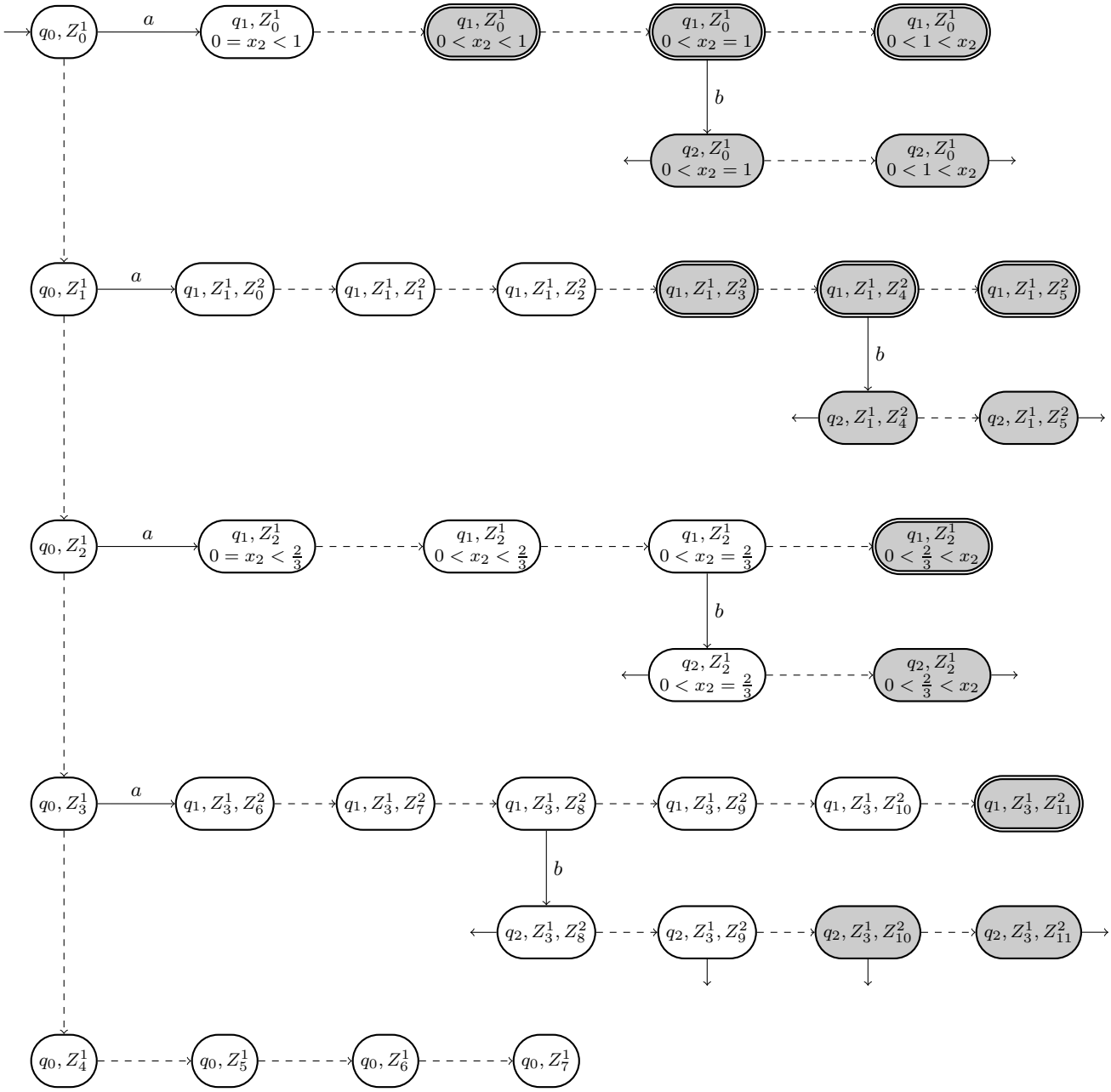


Fig. 4. The class automaton for $\mathcal{A}_1$ and formula φ_1.