Positive and negative results on the decidability of the model-checking problem for an epistemic extension of Timed CTL

Cătălin Dima*
LACL, Université Paris Est – Université Paris 12, 61 av. du Général de Gaulle, 94010 Créteil, France

Abstract

We present TCTLK, a continuous-time variant of the Computational Tree Logic with knowledge operators, generalizing both TCTL, the continuous-time variant of CTL, and CTLK, the epistemic generalization of CTL. Formulas are interpreted over timed automata, with a synchronous and perfect recall semantics, and the observability relation requires one to specify what clocks are visible for an agent.

We show that, in general, the model-checking problem for TCTLK is undecidable, even if formulas do not use any clocks – and hence CTLK has an undecidable model-checking problem when interpreted over timed automata. On the other hand, we show that, when each agent can see all clock values, model-checking becomes decidable.

1 Introduction

Combinations of temporal and epistemic logics offer a useful setting for the analysis of multi-agent systems. They have been successfully utilized for model-checking protocols like the Alternating Bit Protocol [14], or the Chaum's Dining Cryptographers Protocol [16, 9], whose functioning is related with participants' knowledge of the system state. Algorithms exist for the model-checking problem [15, 10, 4] (to cite only a few) and several tools have been designed for model-checking [11, 5].

In this paper we investigate one possibility to extend these results to the continuous case. We present a logic, called here TCTLK, which is based on an epistemic extension of the Computational Tree Logic, with the addition of freeze clock variables for expressing time. The semantics we consider here is a synchronous and perfect recall semantics [6, 7], and is given in terms of timed automata [2]. The observability relations used for defining the epistemic operators include the ability of agents to observe truth values of some atomic propositions (and the inability to observe some others), as well as their ability to observe *clock values*, as a generalization of the ability to observe only time passage. Recall that synchronicity means that agents may observe the exact current absolute time, while perfect recall means that, based on the history of observations they can make on system states, agents are able to distinguish histories, and not only the current global state of the system.

Our focus is on extending some results concerning an epistemic variant of CTL without common knowledge. In [4] we proved that model-checking formulas in such a logic is nonelementary, and requires a subset construction on the model. The question one would ask is then whether that construction can be adapted to the continuous time, provided that timed automata cannot be determinized [2].

We show that, indeed, model-checking for this timed version of epistemic CTL with synchrony and perfect recall is undecidable. Our result is even stronger: model-checking for CTL with "simple" knowledge operators with a continuous-time semantics is undecidable too, even without the freeze quantifiers from TCTL, and the result shows that only one unobservable clock suffices for undecidability. This also means that undecidability holds even if the agents are only able to observe time passage. The source of this strong result comes more from the expressive power of the models than from the logic itself, and is strongly related with the impossibility to generalize the subset construction for timed automata.

On the other hand, we show that if agents are able to fully observe clock values – and this includes the observability of the values for the freeze clocks of TCTL – then model-checking becomes decidable. The proof goes by a straightforward adaptation of the classical algorithm for TCTL model-checking [1], and is based on the fact that, with full clock observability, the subset construction only concerns the "untimed" part of the model.

Previous work on combining epistemic and continuous time expressivity includes [17]. There, a continuous time variant of CTLK is also presented, with a state-based semantics that includes state and clock observability, but not history or absolute time observability. Therefore, the logic in [17] has a decidable model-checking problem. However our logic is more expressive, due to its perfect recall semantics.

*Partially supported by the French ANR project SPREADS

The semantics of a timed automata will be given here in a weakly-monotonic setting which draws some similarities with the work of [13]. This avoids problems related to the density of real numbers, that lead sometimes authors to consider a "nonstrict" variant of the until operator, as in [3]. We introduce a weakly-monotonic presentation for both trajectories and runs. Trajectories are, in some sense, annotations of timed words [2] with clock information, and, as such, are a generalization of the concept of word in finite automata. Runs are continuous presentations of behaviors of timed automata, embodying the possibility to have, in a model, some finite control (locations) which is not captured by formulas. The observability relation is then introduced using projection and equivalence on trajectories. Equivalence is needed as there may be several trajectories representing the same behavior, due to the possibility to have *silent transitions* in a timed automaton. Also projection cannot be defined directly on runs, as it models "forgetting" some part of the observable state of a system.

We also note that the decidability result would still hold if we use a *bounded asynchronous* semantics. The idea is that agents would only have the possibility to observe the integer parts of the clock values, and of the differences between clocks, but not the exact clock values. The final section contains a brief discussion on this topic.

The rest of the paper is divided as follows: in Section 2 we lay the basis for the semantics of timed multi-agent systems. Section 3 gives the undecidability result, while Section 4 gives the decidability in the case of full observability of the clock values. A section with final remarks concludes the paper.

2 A weakly-monotonic semantics for TCTLK

In this section we will give a weakly-monotonic semantics of timed automata, in terms of (continuous) *trajectories*. A continuous trajectory is a description of a particular behavior of the automaton, describing at each time point the location in which the automaton is at that time point, and the values of all clocks. It is, in some sense, similar with words for finite automata. We will present the "time domain" of a trajectory in a weakly-monotonic fashion [13], in order to accomodate the possibility of a timed automaton to take more than one transition at a given instant.

We will utilize throughout this paper a set $\mathcal{X}$ of nonnegative real-valued variables called *clocks*. A *clock valuation* is a mapping $v : \mathcal{X} \to \mathbb{R}_{\geq 0}$. For any clock valuation v and set of clocks $X \subseteq \mathcal{X}$, $v[X := 0]$ denotes the clock valuation defined by $v[X := 0](x) = 0$ for $x \in X$ and $v[X := 0](x) = v(x)$ for $x \notin X$. As always, $\infty - \alpha = \infty$ for any $\alpha \in \mathbb{R}$.

Definition 1 *A* ***trajectory*** *over $\mathcal{X}$ and a set of state symbols Π is a pair $T = (\mathcal{I}, \theta)$, where $\mathcal{I}$ is a (finite or infinite) sequence of pairs of sets of state symbols and intervals*

- $\mathcal{I} = \big(S_i, [\alpha_{i-1}, \alpha_i]\big)_{1 \leq i < \eta}$ *with* $\eta \in \mathbb{N} \cup \{\infty\}$, $S_i \subseteq \Pi$, $\alpha_{i-1}, \alpha_i \in \mathbb{R}_{\geq 0}$, $\alpha_{i-1} \leq \alpha_i$ *and* $\alpha_0 = 0$.

and θ is a continuous mapping of clock intervals:

- $\theta = (\theta_i)_{1 \leq i < \eta}$ *with* $\theta_i : [\alpha_{i-1}, \alpha_i] \times \mathcal{X} \to \mathbb{R}_{\geq 0}$

subject to the following properties.

1. *For all $i < \eta$ and all $t, t' \in [\alpha_{i-1}, \alpha_i]$, $t < t'$, and for all $x \in \mathcal{X}$, $\theta_i(t', x) = \theta(t, x) + t' - t$.*

2. *For all $i < \eta - 1$ there exists a (possibly empty) set of clocks $X \subseteq \mathcal{X}$ such that $\theta_{i+1}(\alpha_i, \cdot) = \theta_i(\alpha_i, \cdot)[X := 0]$.*

Remark 1 *Note that for two successive elements of $\mathcal{I}$, $(S_i, [\alpha_{i-1}, \alpha_i])$, $(S_{i+1}, [\alpha_i, \alpha_{i+1}])$, intervals are adjacent.*

The *weakly-monotonic time domain* of a trajectory $T = (\mathcal{I}, \theta)$ is the set $\mathcal{I}_{intv} = \bigcup_{1 \leq i < \eta} \{i\} \times [\alpha_{i-1}, \alpha_i]$. Elements of $\mathcal{I}_{intv}$ will be called *weakly-monotonic time points* (or *w-points* for short). $\mathcal{I}_{intv}$ can be totally-ordered as usual: given $(n, t), (n', t') \in \mathcal{I}_{intv}$, we say that (n, t) *precedes* (n', t') and denote $(n, t) \prec (n', t')$ if either $n < n'$ or $n = n'$ and $t < t'$.

Remark 2 *Note that, for some $1 \leq i < \eta$, $\mathcal{I}_{intv}$ might contain only one element of the form (i, α). The intuition behind this is that the i-th state in the trajectory is transient, the system must pass through this state but it rests there for zero amount of time.*

Given a trajectory $T = (\mathcal{I}, \theta)$ and some w-point $(n, \beta) \in \mathcal{I}_{intv}$, we may define the *suffix of T starting with* (n, β), denoted by $T[(n, \beta)..] = \big(\mathcal{I}[(n, \beta)..], \theta[(n, \beta)..]\big)$, as follows:

1. $\mathcal{I}[(n, \beta)..] = \big(S_{i+n-1}, [\alpha'_{i-1}, \alpha'_i]\big)_{1 \leq i < \eta'}$ with $\eta' = \eta - n + 1$ and $\alpha'_i = \alpha_{i+n-1}$ for all i with $1 \leq i < \eta'$

2. $\theta[(n, \beta)..] = (\theta'_i)_{1 \leq i < \eta'}$ with $\theta'_i(t, x) = \theta_{i+n-1}(t, x)$ for all i with $1 \leq i < \eta'$.

In the same settings, we may define the *prefix of T upto* (n, β), denoted by $T[0..(n, \beta)] = \big(\mathcal{I}[0..(n, \beta)], \theta[0..(n, \beta)]\big)$, as follows:

1. $\mathcal{I}[0..(n, \beta)] = \big(S_i, [\alpha'_{i-1}, \alpha'_i]\big)_{1 \leq i < n+1}$ with $\alpha'_i = \alpha_i$ for $i < n$, and $\alpha'_n = \beta$.

2. $\theta[0..(n, \beta)] = (\theta'_i)_{1 \leq i \leq n}$ is defined by $\theta'_i(t, x) = \theta_i(t, x)$ for all $1 \leq i \leq n$, $t \in [0, \alpha'_i]$.

Another useful operation is *resetting some clock z at some w-point* (k, β): given a trajectory $T = (\mathcal{I}, \theta)$ and some w-point $(k, \beta) \in \mathcal{I}_{intv}$, the trajectory $T[z := 0 \text{ at } (k, \beta)] = (\mathcal{I}', \theta')$ is defined as follows:

- $\mathcal{I}' = (S'_i, [\alpha'_{i-1}, \alpha'_i])_{1\le i<\eta'}$ where $\eta' = \eta+1$ and $S'_i = S_i$ for $i \le k$, $S'_i = S_{i-1}$ for $k < i < \eta'$ and $\alpha'_i = \alpha_i$ for $i < k$, $\alpha'_k = \beta$ and $\alpha'_i = \alpha_{i-1}$ for $k < i < \eta'$.

- $\theta' = (\theta'_i)_{1\le i<\eta'}$ is defined by

$$\theta'_i(t,z) = \begin{cases} \theta_i(t,z), \text{ for } i \le k \text{ and } (i,t) \prec (k,\beta) \\ \theta_{i-1}(t,z), \text{ if } k < i < \eta', \exists j \le i-1, \theta_j(t,z) = 0 \\ \theta_{i-1}(t,z) - \theta_{i-1}(\beta,z), \text{ otherwise} \end{cases}$$

and, for $x \neq z$, $\theta'_i(t,x) = \theta'_i(t,x)$ for $i \le k$ and $(i,t) \prec (k,\beta)$, and $\theta'_i(t,x) = \theta'_{i-1}(t,x)$ otherwise.

Figure 1 gives an example of a trajectory T, and of $T[x := 0 \text{ at } (1,1)]$

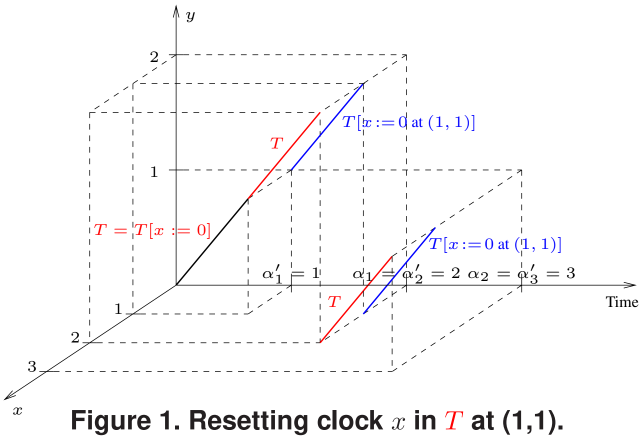

Figure 1. Resetting clock x in T at (1,1).

A trajectory is *initialized* if all the clocks are set to zero at the initial w-point, i.e.for all $x \in X$, $\alpha_1(0,x) = 0$.

2.1 Trajectories and observability

In this subsection we give the observability relation on trajectories, which represents the essence of the epistemic part of the logic. This requires the definition of a projection operator on trajectories, which tells what are the clock values and state symbols that are observable by some agent along some trajectory.

Given a set of clocks $\mathcal{X}' \subseteq \mathcal{X}$, a set of states $\Pi' \subseteq \Pi$ and a trajectory $T = (\mathcal{I}, \theta)$ where $\mathcal{I} = \big(S_i, [\alpha_{i-1}, \alpha_i]\big)_{1\le i<\eta}$ and $\theta = \big(\theta_i\big)_{1\le i<\eta}$, the **projection** of T **defined by** $(\Pi, \mathcal{X}')$ is the trajectory $T\big|_{\Pi,\mathcal{X}'} = (\mathcal{I}\big|_{\Pi,\mathcal{X}'}, \theta\big|_{\Pi,\mathcal{X}'})$ with $\mathcal{I}\big|_{\Pi,\mathcal{X}'} = \big(S_i \cap \Pi', [\alpha_{i-1}, \alpha_i]\big)_{1\le i<\eta}$ and $\theta\big|_{\Pi,\mathcal{X}'} = \big(\theta'_i\big)_{1\le i<\eta}$ where $\theta'_i(t,x) = \theta_i(t,x)$ for all $i < \eta, t \in [\alpha_{i-1}, \alpha_i]$, and $x \in \mathcal{X}$.

Our presentation of trajectories has the problem that some behavior can be presented by distinct trajectories: it is what we call the "stuttering" phenomenon. A trajectory is *stuttering-free* if either two consecutive intervals in $\mathcal{I}$ are labeled with distinct state symbols or if some clock is reset in between the two intervals. Formally, a trajectory $T = (\mathcal{I}, \theta)$ is **stuttering-free** if for all $i < \eta - 1$, if $S_i \neq S_{i+1}$ then there exists $x \in X$ with $\theta_{i+1}(\alpha_i, x) = 0 \neq \theta_i(\alpha_i, x)$.

Stuttering-free trajectories represent "normal forms" for behaviors of timed automata: consider two trajectories $T_j = (\mathcal{I}^j, \theta^j)$ $(j = 1,2)$, where $\mathcal{I}^j = \big(S^j_i, [\alpha^j_{i-1}, \alpha^j_i]\big)_{1\le i<\eta_j}$ and $\theta^j = \big(\theta^j_i\big)_{1\le i<\eta_j}$. Assume that T_2 is stuttering-free. We say that T_2 **represents** T_1 if there exists some surjective increasing function $\varphi : [1..\eta_1 - 1] \to [1..\eta_2 - 1]$ satisfying the following requirements:

- For all $i < j < \eta_1$, if $\varphi(i) = \varphi(j)$ then $S^1_i = S^1_j = S^2_{\varphi(i)}$.
- $\alpha^2_{\varphi(i)} = \max\big\{\alpha^1_j \mid \varphi(j) = \varphi(i)\big\}$.
- For all $i < \eta_1$ and $t \in [\alpha_{i-1}, \alpha_i]$, $\theta^2_{\varphi(i)}(t,x) = \theta^1_i(t,x)$.

We also say that two (arbitrary) trajectories T_1, T_2 are **identical**, denoted $T_1 \equiv T_2$, if there exists a third stuttering-free trajectory T_3 which represents both T_1 and T_2.

2.2 A weakly-monotonic semantics of timed automata

A *simple constraint* over $\mathcal{X}$ is a constraint of the form $x \in I$, where I is an interval with nonnegative integer bounds. For a simple constraint C and a clock valuation v, we denote as usual $v \models C$ if v satisfies the constraint C.

Definition 2 *A* ***timed automaton*** *[2] is a tuple $\mathcal{A} = (Q, \mathcal{X}, \Pi, \delta, \lambda, Q_0)$ where Q is a finite set of* locations*, $\mathcal{X}$ is a finite set of* clocks*, Π is a finite set of* state labels*, $\lambda : Q \to 2^\Pi$ is the* state-labeling function*, $Q_0 \subseteq Q$ is the set of* initial *locations, and δ is a finite set of tuples called* transitions *of the form (q, C, X, q'), where $q, q' \in Q$, $X \subseteq \mathcal{X}$, and C is a conjunction of simple constraints over $\mathcal{X}$.*

Our semantics of timed automata will be given in terms of runs, which are behaviors of timed automata along trajectories. This is just an extension of the classical notion of a run in a timed automaton, enhanced with the possiblity to have "transient" locations (in which control stays for a zero amount of time) and adapted such that each time point be associated with the current location.

A *run* in a timed automaton $\mathcal{A} = (Q, \mathcal{X}, \Sigma, \delta, \lambda, Q_0)$ is then a pair $R = (\mathcal{I}, \rho)$, where

- $\mathcal{I}$ is a sequence of pairs of locations and intervals, $\mathcal{I} = \big(q_i, [\alpha_{i-1}, \alpha_i]\big)_{1\le i<\eta}$, where $\eta \in \mathbb{N} \cup \{\infty\}$, $q_i \subseteq Q$ and $\alpha_{i-1}, \alpha_i \in \mathbb{R}_{\ge 0}$ with $\alpha_{i-1} \le \alpha_i$, and $\alpha_0 = 0$.
- $\rho = (\rho_i)_{1\le i<\eta}$ with $\rho_i : [\alpha_{i-1}, \alpha_i] \times \mathcal{X} \to \mathbb{R}_{\ge 0}$.

subject to the following constraints:

1. For all $1 \le i < \eta$ and all $t, t' \in [\alpha_{i-1}, \alpha_i]$, $t < t'$, and for all $x \in \mathcal{X}$, $\theta_i(t', x) = \theta(t,x) + t' - t$.

2. For all $1 \leq i < \eta - 1$ there exists some transition $(q_i, C_i, X_i, q_{i+1}) \in \delta$ for which $\rho_i(\alpha_i, \cdot) \models C_i$ and $\rho_{i+1}(\alpha_i, \cdot) = \rho_i(\alpha_i, \cdot)[X_i := 0]$.

We say that the run R is **unbounded** (or **has an unbounded domain**) if $\eta = \infty$ or $\alpha_{\eta-1} = \infty$.

At second item of the enumeration above, we say that the transition (q_i, C_i, X_i, q_{i+1}) *is associated with the w-point* (i, α_i) – and also associated with $(i+1, \alpha_i)$, both being w-points in $\mathcal{I}_{intv}$ for the run R.

Similarly to trajectories, prefix and postfix operators can be defined on runs. Hence, given a run $R = (\mathcal{I}, \rho)$ and some w-point $(n, \beta) \in \mathcal{I}_{intv}$, the prefix of R upto (n, t) is denoted $R[0..(n, \beta)] = \big(\mathcal{I}[0..(n, \beta)], \rho[0..(n, \beta)]\big)$ and the suffix of R starting with (n, t) is denoted $R[(n, \beta)..] = \big(\mathcal{I}[(n, \beta)..], \rho[(n, \beta)..]\big)$.

Reseting some clock along a run, $R[z := 0 \text{ at } (k, \beta)]$, can also be defined as for trajectories. However the result might not always be a run: it is a run only when there exists a loop (q_k, C, X, q_k) which can be associated with the w-point (k, β) in the result $R[z := 0 \text{ at } (k, \beta)]$.

Given a run $R = (\mathcal{I}, \rho)$ with $\mathcal{I} = \big(q_i, [\alpha_{i-1}, \alpha_i]\big)_{1 \leq i < \eta}$, the **trajectory generated by** R is the trajectory $\mathsf{traj}(R) = (\mathcal{I}', \rho)$ where $\mathcal{I} = \big(\lambda(q_i), [\alpha_{i-1}, \alpha_i]\big)_{1 \leq i < \eta}$.

A run $R = (\mathcal{I}, \rho)$ with $\mathcal{I} = \big(q_i, [\alpha_{i-1}, \alpha_i]\big)_{1 \leq i < \eta}$ and $\rho = (\rho_i)_{1 \leq i < \eta}$ is *accepting* if the first location q_0 is an initial location and the initial clock valuation is $\alpha_1(0, x) = 0$ for all $x \in \mathcal{X}$. A trajectory T is *accepted* if it is associated with an accepting run. Note that in our timed automata, all locations are "accepting".

3 TCTLK defined

The logic we investigate here, called the Timed Computational Tree Logic with knowledge operators and with synchronous and perfect recall semantics, denoted in the following as TCTLK, has the following syntax:

$$\phi ::= p \mid C \mid \phi \wedge \phi \mid \neg \phi \mid \phi \, A\mathcal{U} \, \phi \mid \phi \, E\mathcal{U} \, \phi \mid z \text{ in } \phi \mid K_A \phi$$

where $p \in \Pi$, the set of atomic propositions, C is a simple constraint over the set of clocks $\mathcal{X}$, y is some clock in $\mathcal{X}$, and $A \in Ag$ a finite set of agents.

The operator $z \text{ in } \phi$ is called the *freeze operator*, and CTLK is the logic defined by formulas not containing the freeze operator.

The semantics of TCTLK is given in terms of runs in a timed automaton which is endowed with some partial observability relations, one for each *agent* A in a finite set of agents Ag. Each partial observability relation models what is observable by some agent A and is induced by a subset of symbols $\Pi_A \subseteq \Pi$ and a subset of clocks $\mathcal{X}_A \subseteq \mathcal{X}$.

Formally, the semantics is given in terms of a multi-agent timed system, defined as follows:

Definition 3 *A* ***timed system with agents in*** Ag *is a tuple* $\mathcal{M} = (\mathcal{A}, \mathcal{Z}, (\Pi_A)_{A \in Ag}, (\mathcal{X}_A)_{A \in Ag})$ *where* $\mathcal{A}$ *is a timed automaton* $\mathcal{A} = (Q, \mathcal{X}, \Pi, \delta, \lambda, Q_0)$*,* $\mathcal{Z} \subseteq \mathcal{X}$ *is a subset of clocks (called* freeze clocks*) and for all* $A \in Ag$*,* $\Pi_A \subseteq \Pi$*,* $\mathcal{X}_A \subseteq \mathcal{X}$*. It is also assumed that* $\mathcal{A}$ *contains, for each location* $q \in Q$ *and each clock* $z \in \mathcal{Z}$*, a loop* $(q, true, \{z\}, q)$*.*

Remark 3 *Note that the last assumption in Definition 3 implies that for any run* R *in* $\mathcal{A}$ *and any w-point* (k, β) *in the run,* $R[z := 0 \text{ at } (k, \beta)]$ *is also a run in* $\mathcal{A}$*.*

The requirement that each location has a loop resetting a clock in $\mathcal{Z}$ does not represent a strong restriction: we may think that we are given a timed system $\mathcal{M}$ without the clocks $\mathcal{Z}$, and then we enhance $\mathcal{M}$ by appending clocks from $\mathcal{Z}$, such that the freeze quantifiers be interpretable.

For each agent $A \in Ag$, the **observability relation for** A is the following: given two runs R_1 and R_2 in $\mathcal{A}$, and some w-point (k, β) occuring in both, we say that R_1 and R_2 **cannot be distinguished by** A **upto** (k, β) if $\mathsf{traj}(R_1)[0..(k, \beta)]\big|_{\Pi_A, \mathcal{X}_A} \equiv \mathsf{traj}(R_2)[0..(k, \beta)]\big|_{\Pi_A, \mathcal{X}_A}$.

The semantics of TCTLK formulas is given in terms of tuples $(\mathcal{M}, R, k, \beta)$ consisting of an n-agent system $\mathcal{M}$, an *unbounded* run $R = (\mathcal{I}, \rho)$ in the underlying timed automaton of $\mathcal{M}$ and some w-point $(k, \beta) \in \mathcal{I}_{intv}$. The rules defining the semantics of TCTLK formulas are the following:

- $(\mathcal{M}, R, k, \beta) \models p$ if $p \in \lambda(q_0)$, q_0 being the first location in R.
- $(\mathcal{M}, R, k, \beta) \models C$ if $v \models C$ where v is the clock valuation at (k, β) in R, $v = \rho_k(\beta, \cdot)$ for $R = (\mathcal{I}, \rho)$.
- $(\mathcal{M}, R, k, \beta) \models \phi_1 \wedge \phi_2$ if $(\mathcal{M}, R, k, \beta) \models \phi_1$ and $(\mathcal{M}, R, k, \beta) \models \phi_2$.
- $(\mathcal{M}, R, k, \beta) \models \neg \phi$ if $(\mathcal{M}, R, k, \beta) \not\models \phi$.
- $(\mathcal{M}, R, k, \beta) \models z \text{ in } \phi$ if $(\mathcal{M}, R[z := 0 \text{ at } (k, \beta)], k, \beta) \models \phi$.
- $(\mathcal{M}, R, k, \beta) \models K_A \phi$ if for any run R' and any w-point (k', β) in R' with $\mathsf{traj}(R')[0..(k', \beta)]\big|_{\Pi_A, \mathcal{X}_A} \equiv \mathsf{traj}(R)[0..(k, \beta)]\big|_{\Pi_A, \mathcal{X}_A}$ we have that $(\mathcal{M}, R', k', \beta) \models \phi$.
- $(\mathcal{M}, R, k, \beta) \models \phi_1 \, E\mathcal{U} \, \phi_2$ if there exists some run $R' = (\mathcal{I}', \rho')$ for which $R[0..(k, \beta)] = R'[0..(k, \beta)]$ and there exists a w-point $(k', \beta') \in \mathcal{I}'_{intv}$ with $(k', \beta') \succeq (k, \beta)$, $(\mathcal{M}, R', k', \beta') \models \phi_2$ and for all $(k'', \beta'') \in \mathcal{I}'_{intv}$ for which $(k'', \beta'') \prec (k', \beta')$, $(k'', \beta'') \succeq (k, \beta)$, we have that $(\mathcal{M}, R', k'', \beta'') \models \phi_1$.
- $(\mathcal{M}, R, k, \beta) \models \phi_1 \, A\mathcal{U} \, \phi_2$ if for any run $R' = (\mathcal{I}', \rho')$ for which $R[0..(k, \beta)] = R'[0..(k, \beta)]$, there exists a w-point $(k', \beta') \in \mathcal{I}'_{intv}$ with $(k', \beta') \succeq (k, \beta)$,

$(\mathcal{M}, R', k', \beta') \models \phi_2$ and for all $(k'', \beta'') \in \mathcal{I}'_{intv}$ for which $(k'', \beta'') \prec (k', \beta'), (k'', \beta'') \succeq (k, \beta)$ we have that $(\mathcal{M}, R', k'', \beta'') \models \phi_1$.

The usual abbreviations apply here too, in particular

$$E\Diamond\,\phi = true\ E\mathcal{U}\,\phi \qquad A\Box\,\phi = \neg\,E\Diamond\,\neg\phi$$
$$A\Diamond\,\phi = true\ A\mathcal{U}\,\phi \qquad E\Box\,\phi = \neg\,A\Diamond\,\neg\phi$$
$$P_A\phi = \neg K_A\neg\phi$$

An example of a TCTLK formula is $P_A\varphi\wedge\neg\varphi$, with $\varphi = z$ in $E\Diamond(z \leq 3 \wedge danger)$. This might model a situation in which some sensor A might provoke a false alarm, as its observable state would indicate that it's possible that in less than 3 time units the system goes into a dangerous state, whereas this situation cannot occur in the current state.

4 Undecidability of model-checking

In this section we show that the model-checking problem is undecidable for CTLK with the continuous-time semantics. For a given timed n-agent system $\mathcal{M}$ and a formula ϕ, we denote $\mathcal{M} \models \phi$ if for any accepting run R in the underlying automaton, we have that $(\mathcal{M}, R, 0, 0) \models \phi$.

Definition 4 *The* ***model-checking*** *problem for* TCTLK *(*CTLK*) over continuous time domains is the following: given a timed n-agent system $\mathcal{M}$ and a* TCTLK *(*CTLK*) formula ϕ, does the property $\mathcal{M} \models \phi$ hold?*

Theorem 1 *The model-checking problem for* CTLK *over continuous time domains is undecidable.*

Proof: The proof idea is similar to many of the usual proofs of undecidability over continuous-time domains: we show that if some 2-counter machine has an infinite computation, then that computation can be simulated with an instance of the model-checking problem. (We utilize here 2-counter machines as programs utilizing only incrementing, and decrementing instructions for one of the two counters, or tests for zero.) Each configuration of the 2-counter program will be encoded into a unit-time interval of an accepted trajectory, with the value of each counter being encoded as the number of positions within the interval where some specific atomic proposition holds. It is assumed that the respective atomic propositions holds only at "sparse" points within any interval.

More specifically, each configuration (ℓ, ct_1, ct_2) of the 2-counter program, where ℓ is the label of the current instruction, is coded as a unit-length part of a trajectory in which ℓ holds as an atomic proposition. Also, during this part of the trajectory, there are exactly ct_1 point intervals where a new atomic proposition p_{ct_1} holds, and exactly ct_2 point intervals where p_{ct_2} holds.

We need to keep track, within each such unit interval, of the last instruction executed before reaching program label ℓ. This is the role of a set of new propositional symbols p_{instr} for all the eight type of instructions that may occur: $p_{\{ct_1++\}}, p_{\{ct_2++\}}, p_{\{ct_1--\}}, p_{\{ct_2--\}}, p_{\{\text{if } ct_1=0\}}, p_{\{\text{if } ct_1\neq 0\}}, p_{\{\text{if } ct_2=0\}}$, and $p_{\{\text{if } ct_2\neq 0\}}$. Note that we also need to recall that the last step in the program was the negative branch of a test instruction – this is the rôle of the label $p_{\{\text{if } ct_1\neq 0\}}$.

Then each transition from a configuration (ℓ, ct_1, ct_2) to another configuration (ℓ', ct'_1, ct'_2) is encoded as the concatenation of the trajectory which encodes the first configuration with the trajectory which encodes the second configuration. Moreover, the i-th position where p_{ct_1} holds in the part of the trajectory corresponding to ℓ is separated by exactly 1 time unit from the i-th position where p_{ct_1} holds in the part of the trajectory corresponding to ℓ'.

This unit separation property applies to instructions which do not modify counter ct_1. It does not apply to the *last* position when the transition from (ℓ, ct_1, ct_2) to (ℓ', ct'_1, ct'_2) is the effect of the instruction $\ell : ct_1{++}$. In this case, exactly one time unit before the last position of p_{ct_1} in the trajectory corresponding to ℓ' we must have a position where p_{ct_1} is false. A similar property holds when the instruction is $\ell : ct_1 - -$. In that case, exactly one time unit after the last position of p_{ct_1} in the interval corresponding to ℓ we must have a position where p_{ct_1} is false.

The above coding is done partly in the timed automaton model, partly in the formula that is checked against the model. The timed automaton simulates, along some trajectories, the control flow in the 2-counter program, and along some other trajectories, the positions where p_{ct_1} and p_{ct_2} occur are copied from each unit interval to another. The use of the temporal and epistemic operators allows us to iteratively construct an accepting trajectory which simulates, in the sense described in the previous paragraphs, the halting computation of the 2-counter program, if it exists.

The first task of the timed automaton is to simulate the control flow, that is, the possible successions of states that may occur if one ignores counter values. Namely, the timed automaton stays for one time unit in states labeled with some ℓ, then shifts to some other states labeled ℓ' if and only if, in the 2-counter program the control may flow from instruction ℓ to instruction ℓ'. In the case of test instructions like $\ell :$ if $ct_1 = 0$ then goto ℓ', the timed automaton simulates both the positive and the negative branch as follows:

- If, during the unit interval along which ℓ holds there is no occurrence of p_{ct_1}, then the control goes (after one time unit) to a state labeled ℓ'.
- Otherwise, the control goes to a state labeled ℓ'', which is the label of the instruction following ℓ in the 2-counter program.

As an example, consider the following simple program:

ℓ_1 : $ct_1{-}{-}$
ℓ_2 : if $ct_1 = 0$ goto ℓ_1
ℓ_3 : $ct_2{+}{+}$

The control flow in this program is modeled by the timed automaton in Figure 2:

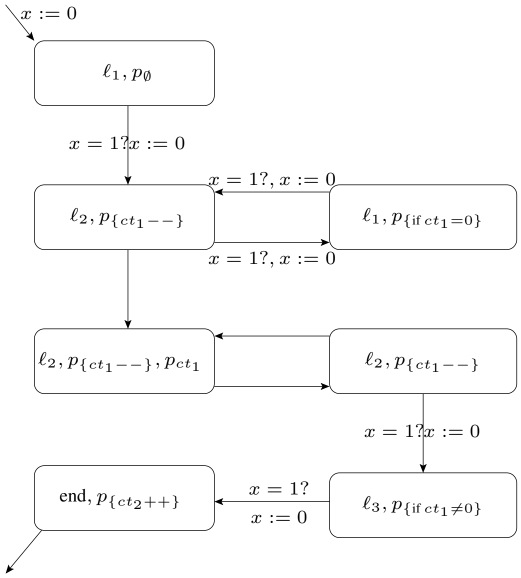

Figure 2. A timed automaton model of the control flow in a simple 2-counter program.

It should be noted that there may be several states labeled ℓ: some might be labeled with p_{ct_1}, and some not, and similarly with all the other propositional symbols that are used throughout the proof. So the states labeled ℓ_1, ℓ_2, ℓ_3 in Figure 2 are "macro-states" which can be further refined into "sub-states", according to whether one or more from the following propositional symbols hold in that state: $p_{ct_1}, p_{ct_2}, p^{last}_{ct_1}, p^{last}_{ct_2}, \overleftarrow{p_{ct_1}}, \overleftarrow{p_{\neg ct_1}}, \overleftarrow{p_{ct_2}}, \overleftarrow{p_{\neg ct_2}}, \overleftarrow{p^{last}_{ct_1}}, \overleftarrow{p^{last}_{ct_2}}$. The refinements have to satisfy the property that the duration of staying in each location is zero in locations where at least one of these symbols holds.

The simulation of control flow does not care about counter values: the mechanism used for "copying" the counter value from some unit interval to the next is different. It involves some branches in the timed automaton which copy just a single occurrence of p_{ct_1} within some interval to the next interval, at exactly one time unit distance. This can be done as in Figure 3, where the occurrence of the atomic proposition $\overleftarrow{p_{ct_2}}$ at the same position where p_{ct_2} holds signals the fact that exactly one time unit earlier we had a position where p_{ct_2} held too.

A similar trick needs to be used to specify that some position where p_{ct_2} is false occurs exactly one time unit after

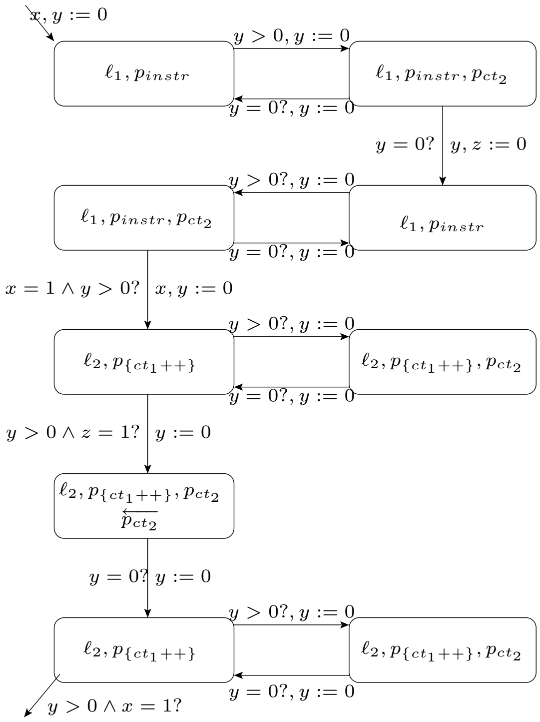

Figure 3. Copying the value of a counter that is not modified by an instruction: the case of copying the position of some p_{ct_2}.

a position where p_{ct_2} was false. This requires the use of another new propositional symbol $\overleftarrow{p_{\neg ct_2}}$, which, whenever occurring, signals that exactly one time unit before we had p_{ct_2} false. The branch of the timed automaton that does this trick is presented in Figure 4.

The tricks used in Figures 3 and 4 can be used for copying only one occurrence of a p_{ct_2}, resp only one point where p_{ct_2} is absent from the location label. How to copy *all* the occurrences of p_{ct_2} within an interval? It is here that the TCTLK formula comes into play, and especially the epistemic operators. We will consider an agent A that is able to observe the truth value for p_{ct_2} and for all propositional symbols of the form ℓ or p_{instr}, with ℓ being a label of some instruction in the 2-counter program, and $instr$ being such an instruction. On the other hand, A cannot observe the truth value for $\overleftarrow{p_{ct_2}}$ or $\overleftarrow{p_{\neg ct_2}}$. We also assume that A cannot observe the value of the clock z.

Suppose we have a trajectory T and a w-point (i, β) in T such that $T[0..(i, \beta)]$ simulates the behavior of the 2-counter program upto the moment when it just finished instruction ℓ_1. We would like to say that, starting from $(i + 1, \beta)$ and during the next unit interval, the positions where p_{ct_2} holds are the translations by one time unit of the positions where p_{ct_2} held in the previous unit interval. The following formula does this, if interpreted over the timed

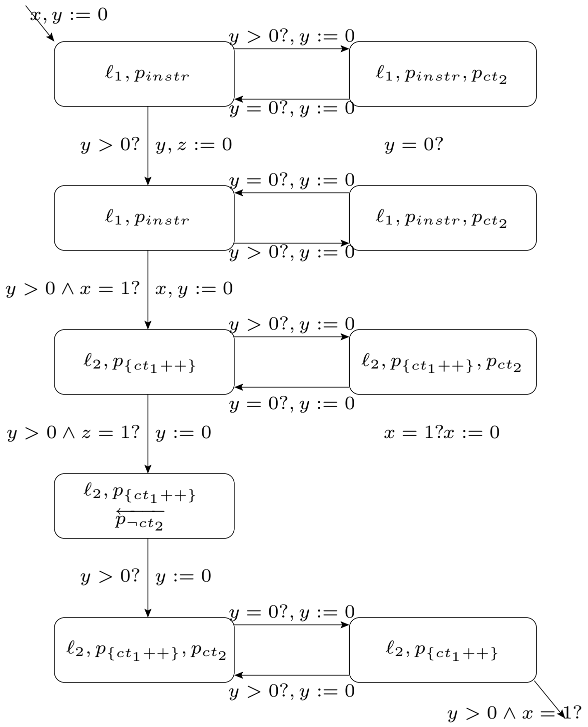

Figure 4. Copying the value of a counter that is not modified by the last instruction: the case of copying the position of some $\neg p_{ct_2}$.

automaton containing branches from Figures 3 and 4:

$$E\Box\Big(\big(\ell_2 \wedge p_{\{ct_1++\}} \wedge p_{ct_2} \rightarrow P_A(\overleftarrow{p_{ct_2}})\big)\wedge \big(\ell_2 \wedge p_{\{ct_1++\}} \wedge \neg p_{ct_2} \rightarrow P_A(\overleftarrow{p_{\neg ct_2}})\big)\Big) \quad (1)$$

This formula says that there exists a trajectory (in the timed automaton) that prolonges T starting with (i, β) and such that at each position (j, γ) where p_{ct_2} holds and we are at label ℓ_2 and the last instruction did not modify ct_2, then there must exist a trajectory T' which A cannot distinguish from T, and in which $\overleftarrow{p_{ct_2}}$ occurs at (j, γ).

Recall then that, by construction, the occurrence of $\overleftarrow{p_{ct_2}}$ at (j, γ) requires that, exactly one time unit before, p_{ct_2} must have been true in T'. And, on the other hand, T' and T must be indistinguishable by A, hence all positions before (i, β) where p_{ct_2} holds in T' must also be positions in T where p_{ct_2} holds. This is exactly what is needed to conclude that for each point (j, γ) where p_{ct_2} holds in T, during unit interval starting at (i, β), then (j, γ) preceded, at one time unit, by a position where p_{ct_2} holds.

The second part of the formula 1 specifies that all positions where p_{ct_2} is false are translations of positions where p_{ct_2} are false in the unit interval right before (i, β).

Figure 5 shows how to treat situations for the instruction $\{ct_1{+}{+}\}$, and hence the last position where p_{ct_1} holds in the unit interval labeled ℓ_2 is not a one time unit translation of the last position where p_{ct_1} held in the previous unit interval. The idea is to have a new propositional symbol $p^{last}_{ct_1}$, visible for A, which signals that we have reached the last p_{ct_1} within the interval where ℓ_2 holds. Also, at the same time $\overleftarrow{p_{\neg ct_1}}$ would be true, signalling that exactly one time unit before p_{ct_1} was false.

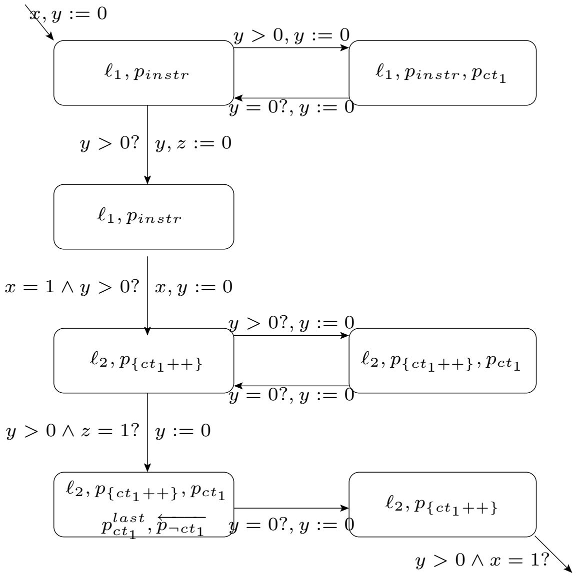

Figure 5. Copying the value of a counter when the last instruction incremented it: the case of inserting the last p_{ct_1}.

The formula which ensures that the last occurrence of p_{ct_1} is "new" and all the other occurrences are copied is then

$$E\Box\Big(\big(\ell_2 \wedge p_{\{ct_1++\}} \wedge p_{ct_1} \wedge \neg p^{last}_{ct_1} \rightarrow P_A(\overleftarrow{p_{ct_1}})\big)\wedge \big(\ell_2 \wedge p_{\{ct_1++\}} \wedge (p^{last}_{ct_1} \vee \neg p_{ct_1}) \rightarrow P_A(\overleftarrow{p_{\neg ct_1}})\big)\Big) \quad (2)$$

Finally, Figure 6 shows how to treat situations for the instruction $\{ct_1--\}$, and hence the last position where p_{ct_1} held in the previous unit interval is no longer copied in the interval where ℓ_2 holds. We utilize here symbol $p^{last}_{ct_1}$ that we have seen before and another new propositional symbol, $\overleftarrow{p^{last}_{ct_1}}$, not visible to A. The significance of $p^{last}_{ct_1}$ is the same as above, while $\overleftarrow{p^{last}_{ct_1}}$ signals that exactly one time unit before we had a position where p_{ct_1} held. The formula which ensures that the last occurrence of p_{ct_1} in the previous unit interval is "lost" and all the other occurrences are copied is then the following

$$E\Box\Big(\big(\ell_2 \wedge p_{\{ct_1--\}} \wedge (p^{last}_{ct_1} \vee p_{ct_1}) \rightarrow P_A(\overleftarrow{p^{last}_{ct_1}})\big)\wedge \big(\ell_2 \wedge p_{\{ct_1--\}} \wedge p_{\neg ct_1} \rightarrow P_A(\overleftarrow{p_{\neg ct_1}})\big)\Big) \quad (3)$$

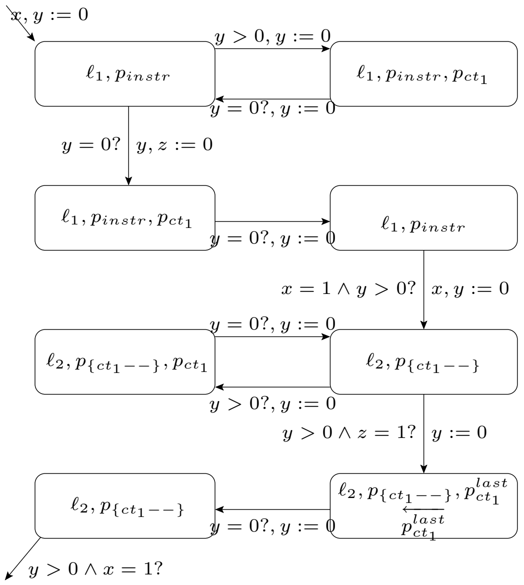

Figure 6. Copying the value of a counter when the last instruction decremented it: the case of deleting the last p_{ct_1}.

Summarizing, we construct a timed automaton $\mathcal{A}$ which is a union of several "branches", each branch being a refinement of the timed automaton in Figure 3 representing the control flow in the 2-counter program: one branch as in Figure 3 for each instruction not modifying a counter, one branch as in Figure 5 for each instruction incrementing a counter, one branch as in Figure 6 for each instruction decrementing a counter, and finally one branch as in Figure 4 for all instructions.

Then, if we denote $\phi_1^{\ell,keep\ ct_i}$ the subformula inside $E\Box$ in Formula 1, corresponding to instructions that keep unchanged the value of counter ct_i, and $\phi_2^{\ell,inc\ ct_i}$ the subformula inside $E\Box$ in Formula 2 corresponding to instructions that increment ct_i, and similarly for $\phi_3^{\ell,dec\ ct_i}$ for Formula 3, the actual formula that we will check against the timed automaton $\mathcal{A}$ is then

$$\Phi = E\Box\left(\neg\text{end} \wedge \bigwedge \phi_1^{\ell,keep\ ct_i} \wedge \bigwedge \phi_2^{\ell,incr\ ct_i} \wedge \bigwedge \phi_3^{\ell,dec\ ct_i}\right)$$

where the conjunction for $\phi_1^{\ell,keep\ ct_i}$ is taken for all instructions labeled ℓ which are preceded by an instruction not modifying counter ct_i, and similarly for all the other conjunctions, and also "end" is the "fictitious" label at the end of the 2-counter program.

Then Φ is satisfied in the timed automaton constructed as above if and only if the given 2-counter machine does not have a halting computation.

Note that our formula utilizes a single agent. Moreover, the essential ingredient for undecidability is that agent A cannot observe the value for the clock z – observing or not the other two clocks does not change the proof.

5 Model-checking with full observability of clock values

In this section we show how the classical model-checking algorithm for TCTL can be adapted to TCTLK when all the agents can observe all clock values. By *full observability of clock values* we mean that formulas of TCTLK are interpreted over multi-agent systems $\mathcal{M} = (\mathcal{A}, \mathcal{Z}, (\Pi_A)_{A\in Ag}, (\mathcal{X}_A)_{A\in Ag})$ in which $\mathcal{X}_A = \mathcal{X}$ for all $A \in Ag$. In the sequel M is the largest constraint occurring in the given timed automaton.

Given a timed automaton $\mathcal{A} = (Q, \mathcal{X}, \Pi, \delta, \lambda, Q_0)$, an $\mathcal{X}$-**region in** $\mathcal{A}$ is a convex set of points $\overline{r} \subseteq \mathbb{R}^n_{\geq 0}$ ($n = card(\mathcal{X})$) characterized by a clock constraint of the form

$$\bigwedge_{x\in Y} (x \in I_x) \wedge \bigwedge_{x,y\in Y} (x - y \in I_{xy}) \wedge \bigwedge_{x\in\mathcal{X}\setminus Y} (x \in]M, \infty[\,)$$

where I_x are non-negative integer-bound intervals whose upper bound are at most M, and I_{xy} are integer-bound intervals whose both upper and lower bounds are in between $-M$ and M. Also $Y \subseteq \mathcal{X}$ is some set of clocks. The set of regions in $\mathcal{A}$ is denoted $\mathsf{Reg}_{\mathcal{A}}$.

The *region automaton* for $\mathcal{A}$ is the tuple $\mathcal{R}_{\mathcal{A}} = \big(Q \times \mathsf{Reg}_{\mathcal{A}}, \delta_{\mathcal{R}}, S_0\big)$ where $S_0 = \big\{(q_0, \mathbf{0}_{\mathcal{X}}) \mid q_0 \in Q_0\big\}$ with $\mathbf{0}$ being the region where all clocks are zero, and

$$\begin{aligned}\delta_{\mathcal{R}} = \big\{&(q, \overline{r}) \to (q, \overline{r}') \mid \overline{r} \neq \overline{r}' \text{ and } \exists v \in \overline{r}, v' \in \overline{r}' \\ &\quad\text{for which } \exists t \in \mathbb{R}_{\geq 0} \text{ with } v' = v + t \text{ and} \\ &\quad \forall t' < t, v + t' \in \overline{r} \cup \overline{r}'\big\} \\ \cup \big\{&(q, \overline{r}) \to (q', \overline{r}') \mid \exists v \in \overline{r}, v' \in \overline{r}', (q, C, X, q') \in \delta \\ &\quad\text{for which } v \models C \text{ and } v' = v[X := 0]\big\}\end{aligned}$$

Here $\delta_{\mathcal{R}}$ defines *immediate transitions* between states in the region automaton. Transitions of the first type in $\delta_{\mathcal{R}}$ are called *time-passage*, while second type of transitions are called *discrete*. A run in $\mathcal{R}_{\mathcal{A}}$ is then simply a sequence of transitions in $\delta_{\mathcal{R}}$ agreeing on intermediary states.

A run $R = (\mathcal{I}, \rho)$ in $\mathcal{A}$ is *simulated by* a run $\sigma = \big((q_{i-1}, \overline{r}_{i-1}) \to (q_i, \overline{r}_i)\big)_{1\leq i\leq n}$ in $\mathcal{R}_{\mathcal{A}}$ if the following properties hold:

- All the w-points in $\mathcal{I}_{intv}$ fall into some state of σ, i.e. for any $(k, \beta) \in \mathcal{I}_{intv}$ there exists some $i \leq n$ such that $\rho_k(\beta, \cdot) \in \overline{r}_i$ and the k-th state in $\mathcal{I}$ is q_i.
- If the w-point (k, β) falls into state $(q_{i-1}, \overline{r}_{i-1})$ and the i-transition in σ is a time-passage transition, then there exists some $t \in \mathbb{R}_{\geq 0}$ with $(k, \beta + t) \in \mathcal{I}_{intv}$ which falls into state $(q_{i-1}, \overline{r}_i)$ (note that in this case $q_{i-1} = q_i$).

- All discrete transitions in σ correspond to discrete transitions in R: for any interval $[\alpha_{i-1}, \alpha_i]$ from $\mathcal{I}$, if $(i-1, \alpha_i)$ falls into state $(q_j, \overline{r}_j)$ then (i, α_i) falls into state $(q_{j+1}, \overline{r}_{j+1})$ and the transition associated with (i, α_i) is the same as the transition associated with the $(q_j, \overline{r}_j) \to (q_{j+1}, \overline{r}_{j+1})$.

Though the region automaton does not represent exactly all runs in $\mathcal{A}$, it has the following important property:

Proposition 1 *Any run in $\mathcal{A}$ is associated with a run in $\mathcal{R}_\mathcal{A}$, and the reverse holds too.*

Theorem 2 *The model-checking problem for* TCTLK *with full observability of clock values is decidable.*

Proof: The proof is a modification of the usual proof for TCTL model-checking by inserting a procedure for treating the case of subformulas of the type $K_A\psi$. Recall that TCTL model-checking works by iteratively labeling states in the region automaton with subformulas of the given formula ϕ. Here, for treating knowledge subformulas we need also to split some states according to the classes of trajectories along which the respective states can be reached, similar to the discrete-time case [4].

So assume we are given a timed system $\mathcal{M} = (\mathcal{A}, \mathcal{Z}, (\Pi_A)_{A\in Ag}, (\mathcal{X}_A)_{A\in Ag})$ and a formula ϕ that we want to model-check on $\mathcal{M}$. The technique is to transform the region automaton $\mathcal{R}_\mathcal{A}$ into another region automaton $\mathcal{R}'$ which embodies the trajectories of $\mathcal{A}$ and in which states may carry, besides region labels, additionnal labels like new propositional symbols representing subformulas of ϕ or pointers to states in $\mathcal{R}_\mathcal{A}$ which are "identically observable" as the current state.

Labeling some state $(q, \overline{r})$ with a new propositional symbol p_ψ signals the fact that on any run R which "passes through" $(q, \overline{r})$, at each moment of this pass the subformula ψ holds. That is,

> $(*)$ For $R = (\mathcal{I}, \rho)$ and $(i, t) \in \mathcal{I}_{intv}$ with $\rho_i(t, \cdot) \in \overline{r}$ and with the i-th state in $\mathcal{I}$ being q, we have that $(\mathcal{M}, R, i, t) \models \psi$.

We recall briefly the construction of $\mathcal{R}'$ for the case of some TCTL operators, and then present the construction for subformulas of the type $K_A\psi$.

For formulas $\psi_1\, E\mathcal{U}\, \psi_2$, suppose that, by induction, the states where ψ_1 holds are labeled with p_{ψ_1} and similarly for p_{ψ_2}. Then we add a new label $p_{\psi_1\, A\mathcal{U}\, \psi_2}$ to each state $(q, \overline{r})$ for which there exists some state $(q', \overline{r}')$ which is labeled by p_{ψ_2} and a path through $\mathcal{R}_\mathcal{A}$ such that all states on the path are labeled with p_{ψ_1}.

For $\psi_1\, A\mathcal{U}\, \psi_2$, we rather proceed by *not labeling* states $(q, \overline{r})$ if and only if there exists a set of states $S \subseteq Q \times \mathsf{Reg}_\mathcal{A}$ such that

- S is strongly connected and all states in S are labeled with p_{ψ_1} and are not labeled with p_{ψ_2}.
- There exists a path leading from $(q, \overline{r})$ to some state in S on which all states are labeled with p_{ψ_1} and not labeled with p_{ψ_2}.

For z in ψ, we label a state $(q, \overline{r})$ with $p_{z \text{ in } \psi}$ if and only if the state $(q, \overline{r}[z := 0])$ is labeled with p_ψ.

The last construction, for $\phi = K_A\psi$, no longer labels existing states, but needs to split the states of the given model in order to be able to identify states that have indistinguishable history, as seen by agent A.

So, assume we start with a region automaton $\mathcal{R} = \big(\overline{Q} \times \mathsf{Reg}_\mathcal{A}, \delta_\mathcal{R}, S_0\big)$ where $\overline{Q} = Q \times S$, S being the extra labeling information appended to each state. We also consider that the labeling function λ in $\mathcal{A}$ is extended to the states of the region automaton, so $\lambda(q, s, \overline{r}) = \lambda(q) \subseteq \Pi$.

We then construct the following region automaton $\mathcal{R}' = \big(\tilde{Q} \times \mathsf{Reg}_\mathcal{A}, \delta'_\mathcal{R}, \tilde{S}_0\big)$ as follows:

$$\tilde{Q} = \big\{(q, Q') \subseteq \overline{Q} \times 2^{\overline{Q}} \mid \lambda(q) \cap \Pi_A = \lambda(q') \cap \Pi_A \ \forall q' \in Q'\big\}$$
$$\tilde{S}_0 = \big\{(q_0, Q'_0, \mathbf{0}) \mid (q_0, \mathbf{0}) \in S_0\big\}$$
$$\delta'_\mathcal{R} = \big\{(q_1, Q'_1, \overline{r}_1) \to (q_2, Q'_2, \overline{r}_2) \mid (q_1, \overline{r}_1) \to (q_2, \overline{r}_2) \in \delta_\mathcal{R}\big\}$$

where $Q'_2 = \big\{q'_2 \in \overline{Q} \mid \lambda(q'_2) \cap \Pi_A = \lambda(q_2) \cap \Pi_A$ and $(q'_1, \overline{r}_1) \to (q'_2, \overline{r}_2) \in \delta_\mathcal{R}\big\}$.

Remark 4 *Note that the projection of each state in $\mathcal{R}'$ on its first and third component gives a bijection between runs in $\mathcal{R}$ and runs in $\mathcal{R}'$. Hence, any run in the initial timed automaton is simulated by some run in $\mathcal{R}'$ and vice-versa.*

This property of projection also ensures the fact that the state labeling with the new propositional variables p_χ remains correct: for all runs in the given timed automaton and all w-points (k, β) that fall in the state $(q, Q', \overline{r})$, if $(q, Q', \overline{r})$ is labeled with p_χ then $(\mathcal{M}, R, k, \beta) \models \chi$.

Then we label each state $(q, Q', \overline{r}) \in \tilde{Q} \times \mathsf{Reg}_\mathcal{A}$ with $K_A p_\psi$ if and only if all the states in Q' are labeled with p_ψ. (Note that $q \in Q'$ by construction.) The above remark ensures that for any run R in $\mathcal{A}$ and any w-point (k, β) within R that fall in the state $(q, Q', \overline{r})$ we must have $(\mathcal{M}, R, k, \beta) \models K_A\psi$.

The final observation is that, though the construction of $\mathcal{R}'$ modifies the region automaton by modifying the locations component in each state, it can still be combined with the previous procedures for treating temporal operators. These procedures can be applied to $\mathcal{R}'$ since it is associated with the same trajectories as $\mathcal{R}$. This is again a consequence of the Remark 3. In fact, one may even think of the procedure for obtaining $\mathcal{R}'$ as a location-splitting procedure in the given timed automaton.

To end the proof, we only need to check, at the end of the procedure, whether all the initial states in the resulting region automaton are labeled with p_ϕ.

The complexity of the above construction is nonelementary in the size of the given model. This complexity comes from the fact that each epistemic operator induces an exponential blowup in the size of the model.

Note also that the main aspect allowing the model-checking of TCTLK with full clock observability to be decidable is that we do not need to do subset construction on clock regions (which cannot be done!).

6 Final remarks and conclusions

We have given an epistemic extension of TCTL, with a synchronous and perfect recall semantics in which agents may observe clock values. The model-checking problem is undecidable for the new logic, but becomes decidable if any agent may observe all clocks.

The decidability result still holds if we modify the semantics to a more realistic situation with "bounded asynchronous" observability. This would build on the idea that agents are not able to observe the exact values of clocks, but only whether these values fall in the same region, i.e. satisfy the same clock constraints. If one properly defines the observability relation in this case (which would imply taking into account observability of intermediary clock values), then Theorem 2 would have the same proof.

References

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

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

[3] C. Baier and J. P. Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[4] C. Dima. Revisiting satisfiability and model-checking for CTLK with synchrony and perfect recall. In *Proceedings of CLIMA IX*, Lecture Notes in Artificial Intelligence, 2008. to appear.

[5] P. Gammie and R. van der Meyden. MCK: Model checking the logic of knowledge. In *Proceedings of CAV'04*, volume 3114 of *Lecture Notes in Computer Science*, pages 479–483. Springer, 2004.

[6] J.Y. Halpern and M.Y. Vardi. The complexity of reasoning about knowledge and time: Extended abstract. In *Proceedings of STOC'86*, pages 304–315, 1986.

[7] J.Y. Halpern and M.Y. Vardi. The complexity of reasoning about knowledge and time. I. Lower bounds. *Journal of Computer System Sciences*, 38(1):195–237, 1989.

[8] T. A. Henzinger and V. S. Prabhu. Timed alternating-time temporal logic. In *Proceedings of FORMATS'06*, volume 4202 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2006.

[9] M. Kacprzak, A. Lomuscio, A. Niewiadomski, W. Penczek, F. Raimondi, and M. Szreter. Comparing BDD and SAT based techniques for model checking Chaum's Dining Cryptographers Protocol. *Fundamenta Informaticae*, 72(1-3):215–234, 2006.

[10] A. Lomuscio and F. Raimondi. The complexity of model checking concurrent programs against CTLK specifications. In *Proceedings of DALT'06*, volume 4327 of *Lecture Notes in Computer Science*, pages 29–42. Springer, 2006.

[11] A. Lomuscio and F. Raimondi. MCMAS: A model checker for multi-agent systems. In *Proceedings of TACAS'06*, volume 3920 of *Lecture Notes in Computer Science*, pages 450–454. Springer, 2006.

[12] J. Ouaknine and J. Worrell. On metric temporal logic and faulty turing machines. In *Proceedings of FoSSaCS'06*, volume 3921 of *Lecture Notes in Computer Science*, pages 217–230. Springer, 2006.

[13] P. K. Pandya and Dang Van Hung. Duration calculus of weakly monotonic time. In *Proceedings of FTRTFT'98*, volume 1486 of *Lecture Notes in Computer Science*, pages 55–64. Springer, 1998.

[14] F. Raimondi and A. Lomuscio. Automatic verification of multi-agent systems by model checking via ordered binary decision diagrams. *Journal of Applied Logic*, 5(2):235–251, 2005.

[15] R. van der Meyden and N.V. Shilov. Model checking knowledge and time in systems with perfect recall (extended abstract). In *Proceedings of FSTTCS'99*, volume 1738 of *Lecture Notes in Computer Science*, pages 432–445, 1999.

[16] R. van der Meyden and K. Su. Symbolic model checking the knowledge of the dining cryptographers. In *Proceedings of CSFW-17*, pages 280–. IEEE Computer Society, 2004.

[17] B. Wozna and A. Lomuscio. A logic for knowledge, correctness, and real time. In *Proceedings of CLIMA V*, volume 3487 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2005.