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 [16], and the Chaum’s Dining Cryptographers Protocol [18, 11], whose functioning is related to the participants’ knowledge of the system state. Algorithms exist for the model-checking problem [17, 12, 5] (to cite only a few) and several tools have been designed for model-checking [13, 7].

In this paper we investigate a 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. Here we consider a synchronous and perfect recall semantics [8, 9], 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 synchrony 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 [5] 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 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 to the impossibility to generalize the subset construction to 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 [19]. 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 [19] has a decidable model-checking problem. However our logic is more expressive, due to its perfect recall semantics.
The semantics of a timed automata will be given here in a weakly-monotonic setting which draws some similarities with the work of [15]. 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. Due to space limitations the proofs are only sketched; a technical report [6] can be consulted for more detailed proofs.

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 [15], in order to accommodate the possibility of a timed automaton to take more than one transition at a given instant.

We will utilize throughout this paper a set $X$ of nonnegative real-valued variables called clocks. A clock valuation is a mapping $v : X \rightarrow \mathbb{R}_{\geq 0}$. For any clock valuation $v$ and set of clocks $X \subseteq \Gamma$, $v[X := 0]$ denotes the clock valuation defined by $v[X := 0](x) = 0$ for $x \in X$ and $v[X := 0](x) = v(x)$ for $x \not\in X$. As always, $-\infty - \alpha = \infty$ for any $\alpha \in \mathbb{R}$.

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

1. $I = \{S_i, [\alpha_{i-1}, \alpha_i]\}_{1 \leq i \leq n}$ with $n \in \mathbb{N} \cup \{\infty\}, S_i \subseteq I$, $\alpha_{i-1}, \alpha_i \in \mathbb{R}_{\geq 0}$, $\alpha_{i-1} \leq \alpha_i$ and $\alpha_0 = 0$.

and $\theta$ is a continuous mapping of clock intervals:

- $\theta = (\theta_i)_{1 \leq i \leq n}$ with $\theta_i : [\alpha_{i-1}, \alpha_i] \times X \rightarrow \mathbb{R}_{\geq 0}$ subject to the following properties.

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

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

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

The weakly-monotonic time domain of a trajectory $T = (I, \theta)$ is the set $I_{\text{inte}} = \bigcup_{1 \leq i \leq n} \{x\} \times [\alpha_{i-1}, \alpha_i]$. Elements of $I_{\text{inte}}$ will be called weakly-monotonic time points (or w-points for short). $I_{\text{inte}}$ can be totally-ordered as usual: given $(n, t), (n', t') \in I_{\text{inte}}$, 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 $1 \leq i < n$, $I_{\text{inte}}$ might contain only one element of the form $(i, \alpha)$. 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 $T = (I, \theta)$ and some w-point $(n, \beta) \in I_{\text{inte}}$, we define the suffix of $T$ starting with $(n, \beta)$, denoted by $T[(n, \beta)\ldots] = (I[(n, \beta)\ldots], \theta[(n, \beta)\ldots])$, as follows:

1. $I[(n, \beta)\ldots] = \{ (S_i, [\alpha_{i-1}, \alpha_i]^\prime) \}_{1 \leq i \leq \eta'}$ with $\eta' = n - \beta + 1$ and $\alpha_i' = \alpha_{i-1} + 1$ for all $i$ with $1 \leq i < \eta'$.

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

In the same settings, we may define the prefix of $T$ up to $(n, \beta)$, denoted by $T[0\ldots(n, \beta)] = (I[0\ldots(n, \beta)], \theta[0\ldots(n, \beta)])$, as follows:

1. $I[0\ldots(n, \beta)] = \{ (S_i, [\alpha_{i-1}, \alpha_i]) \}_{1 \leq i \leq n+1}^\prime$ with $\alpha_i' = \alpha_i$ for $i < n$ and $\alpha_n' = \beta$.

2. $\theta[0\ldots(n, \beta)] = (\theta_i')_{1 \leq i \leq n+1}$ 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 $(k, \beta)$: given a trajectory $T = (I, \theta)$ and some w-point $(k, \beta) \in I_{\text{inte}}$, the trajectory $T[z := 0]$ at $(k, \beta) = (I', \theta')$ is defined as follows:
• \( T' = (S'_1, [\alpha'_{i-1}, \alpha'_i])_{1 \leq i < \eta'} \) where \( \eta' = \eta + 1 \) and \( S'_1 = S_i \) for \( i \leq k \), \( S'_2 = S_{i-1} \) for \( k < i < \eta' \) and \( \alpha'_i = \alpha_i \) for \( i < k \), \( \alpha'_k = \beta \) and \( \alpha'_i = \alpha_{i-1} \) for \( k < i < \eta' \).

• \( \theta' = (\theta'_i)_{1 \leq i < \eta'} \) is defined by

\[
\theta'_i(t, z) = \begin{cases} 
\theta_i(t, z), & \text{for } i \leq k \text{ and } (i, t) < (k, \beta) \\
\theta_{i-1}(t, z), & \text{for } k < i < \eta' \text{ and } 2\leq i \leq i - 1, \theta_i(t, z) = 0 \\
\theta_{i-1}(t, z) - \theta_{i-1}(\beta, z), & \text{otherwise.}
\end{cases}
\]

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

Figure 1 shows some trajectory \( T \) and \( T[x := 0 \in (2, 2)] \):

![Figure 1. Resetting clock x in T at (2,2).](image)

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

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 \( T = (I, \theta) \) where \( I = (S_i, [\alpha_1, \alpha_i])_{1 \leq i < \eta} \) and \( \theta = (\theta_i)_{1 \leq i < \eta} \), the projection of \( T \) defined by \( (\Pi', \mathcal{X}') \) is the trajectory \( T|_{\Pi', \mathcal{X}'} = (I|_{\Pi', \mathcal{X}'}, \theta|_{\mathcal{X}'}) \) with \( I|_{\Pi', \mathcal{X}'} = (S_i \cap \Pi'_{i}[\alpha_1, \alpha_i])_{1 \leq i < \eta} \) and \( \theta|_{\mathcal{X}'} = (\theta'_i)_{1 \leq i < \eta} \) 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 \( I \) are labeled with distinct state symbols or if some clock is reset in between the two intervals. Formally, a trajectory \( T = (I, \theta) \) is stuttering-free if for all \( i < \eta - 1 \), if \( S_i \neq S_{i+1} \) then there exists \( x \in X \) with \( \theta_{i+1}(\alpha_i, x) = 0 = \theta_i(\alpha_i, x) \).

Stuttering-free trajectories represent “normal forms” for behaviors of timed automata: consider two trajectories \( T_2 = (I^1, \theta^1) \) \((j = 1, 2)\), where \( I^j = (S'_j, [\alpha_{i-1}, \alpha'_i])_{1 \leq i < \eta_j} \) and \( \theta^j = (\theta'_i)_{1 \leq i < \eta_j} \). Assume that \( T_2 \) is stuttering-free. We say that \( T_2 \) represents \( T_1 \) if there exists some surjective increasing function \( \varphi : [1..\eta - 1] \rightarrow [1..\eta - 2] \) satisfying the following requirements:

• For all \( i < j < \eta_1 \), if \( \varphi(i) = \varphi(j) \) then \( S'_i = S'_j = S^2_{\varphi(i)} \).

• \( \alpha^2_{\varphi(i)} = \max \{ \alpha^2_j \mid \varphi(j) = \varphi(i) \} \).

• 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 \( T_2 \).

A weakly-monotonic semantics of timed automata. A simple constraint over \( X \) 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 \( \mathcal{A} = (Q, \mathcal{X}, \Pi, \delta, \lambda, Q_0) \) where \( Q \) is a finite set of locations, \( \mathcal{X} \) is a finite set of clocks, \( \Pi \) is a finite set of state labels, \( \lambda : Q \rightarrow 2^\Pi \) is the state-labeling function, \( Q_0 \subseteq Q \) is the set of initial locations, and \( \delta \) is a finite set of tuples called transitions of the form \( (q, C, \mathcal{X}, q') \), 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 possibility 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 \( R = (I, \rho) \), where

\[
\rho = (\rho_i)_{1 \leq i < \eta} \quad \text{with} \quad \rho_i : [\alpha_{i-1}, \alpha_i] \times \mathcal{X} \rightarrow [\mathbb{R}_\geq 0].
\]

subject to the following constraints:

1. For all \( 1 \leq i < \eta \) and all \( t, t' \in [\alpha_{i-1}, \alpha_i] \), \( t < t' \), and for all \( x \in X \), \( \theta_i(t', x) = 0 = \theta_i(t, x) \).

2. For all \( 1 \leq i < \eta - 1 \) there exists some transition \( (q_i, C_i, \mathcal{X}_i, q_{i+1}) \in \delta \) for which \( \rho_i(\alpha_i, \cdot) \models C_i \) and \( \rho_{i+1}(\alpha_{i+1}, \cdot) = \rho_i(\alpha_i, \cdot) |_{\mathcal{X}_i := 0} \).

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

At second item of the enumeration above, we say that the transition \( (q_i, C_i, \mathcal{X}_i, q_{i+1}) \) is associated with the w-point.
called the agent following as TCTLK, has the following syntax: synchronous and perfect recall semantics, denoted in the computational Tree Logic with knowledge operators and with the freezed 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 \( A \), and some w-point \( (k, \beta) \) occurring in both, we say that \( R_1 \) and \( R_2 \) cannot be distinguished by \( A \) up to \( (k, \beta) \) if \( \text{traj}(R_1)[0..(k, \beta)]_{\Pi_A, X_A} \equiv \text{traj}(R_2)[0..(k, \beta)]_{\Pi_A, X_A} \).

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

3 TCTLK defined

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

\[\phi ::= p \mid C \mid \phi \land \phi \mid \neg \phi \mid \phi M \phi \mid \phi E \mu \phi \mid \phi K \phi\]

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 \( z \) in \( \phi \) is called the freeze operator. CTLK is the logic defined by formulas not containing the freeze operator.

The semantics of TCTLK is given in terms of runs in a multi-agent timed system, which is a timed automaton 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 \( X_A \subseteq \mathcal{X} \).

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

Remark 3 Note that the last assumption in Definition 3 implies that for any run \( R \) in \( A \) and any w-point \((k, \beta)\) in the run, \( R[z := 0 \text{ at } (k, \beta)] \) is also a run in \( A \).
An example of a TCTLK formula is $P_A \phi \land \neg \phi$, with $\phi = z$ in $E \Diamond (z \leq 3 \land \text{danger})$. 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 $M$ and a formula $\phi$, we denote $M \models \phi$ if for any accepting run $R$ in the underlying automaton, we have that $(M, R, 0, 0) \models \phi$.

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

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

**Proof:** 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 hold only at “sparse” points within any interval.

More specifically, each configuration $(\ell, ct_1, ct_2)$ of the 2-counter program, where $\ell$ is the label of the current instruction, is coded as a unit-length part of a trajectory in which $\ell$ 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 $\ell$. This is the role of a set of new propositional symbols $p_{ct_{ct_1}}$ for all the eight types of instructions that may occur: $P_{ct_1++}, P_{ct_1-+}, P_{ct_1--}, P_{ct_1-0}, P_{ct_1-0'}$. 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 role of the label $P_{ct_1-0'}$.

Then each transition from a configuration $(\ell, ct_1, ct_2)$ to another configuration $(\ell', ct_1', ct_2')$ 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 $\ell$ 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 $\ell'$.

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 $(\ell, ct_1, ct_2)$ to $(\ell', ct_1', ct_2')$ 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 $\ell'$ 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 $\ell$ 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 $\ell$, then shifts to some other states labeled $\ell'$ if and only if, in the 2-counter program the control may flow from instruction $\ell$ to instruction $\ell'$. In the case of test instructions like $\ell : ct_1 = 0$ then goto $\ell'$, the timed automaton simulates both the positive and the negative branch as follows:

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

It should be noted that there may be several states labeled $\ell$: 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.

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.

A similar trick needs to be used to specify that some position where $p_{ct_2}$ is false occurs exactly one time unit after
a position where \( p_{ctz} \) was false. This requires the use of another new propositional symbol \( \overline{p_{ctz}} \), which, whenever occurring, signals that exactly one time unit before we had \( p_{ctz} \) false.

The previous tricks can be used for copying only one occurrence of a \( p_{ctz} \), resp only one point where \( p_{ctz} \) is absent from the location label. How to copy all the occurrences of \( p_{ctz} \) 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 \( p_{ctz} \) and for all propositional symbols of the form \( \ell \) or \( p_{instr} \), with \( \ell \) 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 \( \overline{p_{ctz}} \) or \( \overline{p_{ctz}} \). We also assume that \( A \) cannot observe the value of the clock \( z \).

Suppose we have a trajectory \( T \) and a w-point \((i, \beta)\) in \( T \) such that \( T[0..(i, \beta)] \) simulates the behavior of the 2-counter program up to the moment when it just finished instruction \( \ell_1 \). We would like to say that, starting from \((i + 1, \beta)\) and during the next unit interval, the positions where \( p_{ctz} \) holds are the translations by one time unit of the positions where \( p_{ctz} \) held in the previous unit interval. The following formula does this:

\[
E \Box \left( (\ell_2 \land p_{(ct{z}++)} \land p_{ct{z}} \rightarrow P_A(\overline{p_{ctz}})) \land \right.
\]
\[
\left. (\ell_2 \land p_{(ct{z}++)} \land \overline{p_{ctz}} \rightarrow P_A(\overline{p_{ctz}})) \right) \tag{1}
\]

This formula says that there exists a trajectory (in the timed automaton) that prolongs \( T \) starting with \((i, \beta)\) and such that at each position \((j, \gamma)\) where \( p_{ctz} \) holds and we are at label \( \ell_2 \) and the last instruction did not modify \( ctz \), then there must exist a trajectory \( T' \) which \( A \) cannot distinguish from \( T \), and in which \( p_{ctz} \) occurs at \((j, \gamma)\).

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

The second part of the formula 1 specifies that all positions where \( p_{ctz} \) is false are positions where \( p_{ctz} \) are false in the unit interval right before \((i, \beta)\). First, consider the instruction \( ctz++ \), with the last position where \( p_{ctz} \) holds in the unit interval labeled \( \ell_2 \) not being a one time unit translation of the last position where \( p_{ctz} \) held in the previous unit interval. The idea is to have a new propositional symbol \( p_{ctz}^{last} \), visible for \( A \), which signals that we have reached the last \( p_{ctz} \) within the interval where \( \ell_2 \) holds. Also, at the same time \( \overline{p_{ctz}} \) would be true, signalling that exactly one time unit before \( p_{ctz} \) was false.

The formula which ensures that the last occurrence of \( p_{ctz} \) is “new” and all the other occurrences are copied is then

\[
E \Box \left( (\ell_2 \land p_{(ct{z}++)} \land \overline{p_{ctz}} \rightarrow P_A(\overline{p_{ctz}})) \land \right.
\]
\[
\left. (\ell_2 \land p_{(ct{z}++)} \land \overline{p_{ctz}} \rightarrow P_A(\overline{p_{ctz}})) \right) \tag{2}
\]

Then, for the instruction \( ctz- \), with the last position where \( p_{ctz} \) held in the previous unit interval being no longer copied in the interval where \( \ell_2 \) holds, we utilize here symbol \( p_{ctz}^{last} \) that we have seen before and another new propositional symbol, \( p_{ctz}^{last} \), not visible to \( A \). The significance of \( p_{ctz}^{last} \) is the same as above, while \( p_{ctz}^{last} \) signals that exactly one time unit before we had a position where \( p_{ctz} \) held. The formula which ensures that the last occurrence of \( p_{ctz} \) in the previous unit interval is “lost” and all the other occurrences are copied is then the following

\[
E \Box \left( (\ell_2 \land p_{(ct{z}--)} \land (p_{ctz}^{last} \lor p_{ctz}) \rightarrow P_A(\overline{p_{ctz}})) \land \right.
\]
\[
\left. (\ell_2 \land p_{(ct{z}--)} \land p_{ctz} \rightarrow P_A(\overline{p_{ctz}})) \right) \tag{3}
\]

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

\[
\Phi = E \Box (\neg\end \land \phi_1^{\ell,keep\ ctz} \land \phi_2^{\ell,inc\ ctz} \land \phi_3^{\ell,dec\ ctz})
\]

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

Then \( \Phi \) 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 \( \mathcal{M} = (\mathcal{A}, \mathcal{Z}, (\Pi_A)_{A \in \mathcal{A}}, (\mathcal{X}_A)_{A \in \mathcal{A}}) \) in which \( \mathcal{X}_A = \mathcal{X} \) for all
A \in A_g. In the sequel M is the largest constraint occurring in the given timed automaton.

Given a timed automaton A = (Q, X, \Pi, \delta, \lambda, Q_0), an X-region in A is a convex set of points \mathcal{T} \subseteq \mathbb{R}^n_{\geq 0} (n = card(X)) characterized by a clock constraint of the form
\[
\bigwedge_{x \in \mathcal{Y}} (x \in I_x) \land \bigwedge_{x, y \in \mathcal{Y}} (x - y \in I_{xy}) \land \bigwedge_{x \in \mathcal{X} \setminus \mathcal{Y}} (x \in ]M, \infty[)
\]
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 \neg M and M. Also \mathcal{Y} \subseteq \mathcal{X} is some set of clocks. The set of regions in A is denoted Reg_A.

The region automaton for A is the tuple R_A = (Q \times Reg_A, \delta_R, S_0) where S_0 = \{(q_0, 0x) \mid q_0 \in Q_0\} with 0 being the region where all clocks are zero, and
\[
\delta_R = \{(q, \mathcal{T}) \rightarrow (q', \mathcal{T'}) \mid \mathcal{T} \neq \mathcal{T'} \text{ and } \exists v \in \mathcal{T}, v' \in \mathcal{T'} \text{ for which } \forall t \in R_{\geq 0} \exists v'' = v + t \text{ and } \forall t' < t, v + t' \in \mathcal{T} \cup \mathcal{T'} (\text{time passage})
\]
\[
\cup \{(q, \mathcal{T}) \rightarrow (q', \mathcal{T'}) \mid \exists v \in \mathcal{T}, v' \in \mathcal{T'}, (q, C, X, q') \in \delta \text{ for which } v \models C \text{ and } v' = v[X := 0]\}
\]
Here \delta_R defines immediate transitions between states in the region automaton. A run in R_A is then simply a sequence of transitions in \delta_R agreeing on intermediary states.

A run R = (I, \rho) in A is simulated by a run \sigma = \{(q_{i-1}, \tau_{i-1}) \rightarrow (q_i, \tau_i)\}_{1 \leq i \leq n} in R_A if the following hold:

- All the w-points in I_{intv} fall into some state of \sigma, i.e. for any \((k, \beta) \in I_{intv}\), there exists some \(i \leq n\) such that \rho_k(\beta, \cdot) \in \mathcal{T}_i and the k-th state in I is q_k.

- If the w-point (k, \beta) falls into state (q_{i-1}, \tau_{i-1}) and the i-transition in \sigma is a time-passage transition, then there exists some t \in R_{\geq 0} with (k, \beta + t) \in I_{intv} which falls into state (q_{i-1}, \tau_i) (note that in this case q_{i-1} = q_{i}).

- All discrete transitions in \sigma correspond to discrete transitions in R: for any interval [\alpha_{i-1}, \alpha_i] from I, if (i - 1, \alpha_i) falls into state (q_{i-1}, \tau_i) then (i, \alpha_i) falls into state (q_{i+1, \tau_{i+1}}) and the transition associated with (i, \alpha_i) is the same as the transition associated with the (q_{i}, \tau_i) \rightarrow (q_{i+1}, \tau_{i+1}).

Though the region automaton does not represent exactly all runs in A, it has the following important property:

**Proposition 1** Any run in A is associated with a run in R_A, and the reverse holds too.

**Theorem 2** The model-checking problem 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 \phi. 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 [5].

So assume we are given a timed system M = (A, Z, \Pi, A \in A_g, (X_A) A \in A_g) and a formula \phi that we want to model-check on M. The technique is to transform the region automaton R_A into another region automaton R' which embodies the trajectories of A and in which states may carry, besides region labels, additional labels like new propositional symbols representing subformulas of \phi or pointers to states in R_A which are “identically observable” as the current state.

Labeling some state (q, \mathcal{T}) with a new propositional symbol \rho_q signals the fact that on any run R which “passes through” (q, \mathcal{T}), at each moment of this pass the subformula \psi holds. That is,
\[
(*) \text{ For } R = (I, \rho) \text{ and } (i, t) \in I_{intv} \text{ with } \rho_i(t, \cdot) \in \mathcal{T} \text{ and with the } i-th \text{ state in } I \text{ being } q, \text{ we have that } (M, R, i, t) \models \psi.
\]
Due to space limitations, we only recall the construction for z in \psi: we label a state (q, \mathcal{T}) with \rho_z in \psi if and only if the state (q, \mathcal{T}[z := 0]) is labeled with \rho_z.

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 R = (Q \times Reg_A, \delta_R, S_0) where Q = Q \times S, S being the extra labeling information appended to each state. We also consider that the labeling function \lambda in A is extended to the states of the region automaton, so \lambda(q, s, \mathcal{T}) = \lambda(q) \subseteq \Pi.

We then construct the following region automaton R' = (Q' \times Reg_A, \delta_R', S_0') as follows:
\[
\tilde{Q} = \{(q, q') \subseteq Q' \times 2^\mathcal{T} \mid \lambda(q) \cap \Pi_A = \lambda(q') \cap \Pi_A \forall q' \in Q'\}
\]
\[
\tilde{S}_0 = \{(q_0, Q_0', 0) \mid (q_0, 0) \in S_0\}
\]
\[
\delta_R' = \{(q_1, Q_1', \tau_1) \rightarrow (q_2, Q_2', \tau_2) \mid (q_1, \tau_1) \rightarrow (q_2, \tau_2) \in \delta_R\}
\]
where Q_2' = \{q_2' \in Q' \mid \lambda(q_2') \cap \Pi_A = \lambda(q_2) \cap \Pi_A \text{ and } (q_1', \tau_1) \rightarrow (q_2', \tau_2) \in \delta_R\}.

**Remark 4** Note that the projection of each state in R' on its first and third component gives a bijection between runs in R and runs in R'. Hence, any run in the initial timed automaton is simulated by some run in R' and vice-versa.
This property of projection also ensures the fact that the state labeling with the new propositional variables \( p_{\chi} \) remains correct: for all runs in the given timed automaton and all w-points \((k, \beta)\) that fall in the state \((q, Q', \tau)\), if \((q, Q', \tau)\) is labeled with \( p_{\chi} \) then \((\mathcal{M}, R, k, \beta) \models \chi \).

Then we label each state \((q, Q', \tau) \in \hat{Q} \times \text{Reg}_A \) with \( K_A p_{\psi} \) if and only if all the states in \( Q' \) are labeled with \( p_{\psi} \). (Note that \( q \in Q' \) by construction.) The above remark ensures that for any run \( \hat{R} \) in \( A \) and any w-point \((k, \beta)\) within \( R \) that fall in the state \((q, Q', \tau)\) 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 \( p_{\psi} \).

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.

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.

Acknowledgments The author is indebted to Dimitar Guelev for his suggestions for improving the presentation and for his careful reading of the manuscript.

References