Revisiting satisfiability and model-checking for CTLK with synchrony and perfect recall

Cătălin Dima

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

Abstract. We show that CTL with knowledge modalities but without common knowledge has an undecidable satisfiability problem in the synchronous perfect recall semantics. We also present an adaptation of the classical model-checking algorithm for CTL that handles knowledge operators.

1 Introduction

Combinations of temporal logics and epistemic logics offer a useful setting for the specification and analysis of multi-agent systems. They have been successfully utilized for model-checking protocols like the Alternating Bit Protocol [RL05], or the Chaum's Dining Cryptographers Protocol [vdMS04,KLN$^+$06], whose functioning is related with participants' knowledge of the system state.

Epistemic temporal logics have been studied since the mid-eighties, starting with [HV86,HV89]. These two studies led to the identification of 96 different logics, distinguished by semantics and/or the presence of common knowledge operators, and concern a number of decidability and undecidability results for the satisfiability problem in those logics. In particular, it is shown that Linear Temporal Logic (LTL) with knowledge modalities and no common knowledge has a decidable satisfiability problem in a synchronous and perfect recall semantics. However, the results proved in [HV89,HV86] only concern extensions of LTL, and though both studies mention also some results on branching logics, neither of the two concentrates on proving (un-)decidability for the epistemic extensions of branching time logics.

In this paper, we study the Computational Tree Logic with knowledge operators *and without common knowledge*, a logic that we denote as CTLK. We show that, contrary to the result on LTL, satisfiability within CTLK is undecidable under the synchronous and perfect recall semantics. This result contradicts the claim in [HV86] following which this logic (denoted there KB_n) would be decidable in nonlinear time.

Our proof of the undecidability of CTLK satisfiability is somewhat classical, in the sense that we code the computation of a Turing machine vertically in a tree. This proof technique was utilized many times in the literature for proving undecidability of various epistemic temporal logics, starting from [HV86] where it is proved that LTL *with common knowledge operators* and with various semantics has an undecidable satisfiability problem. We also cite the undecidability result of [vdM98] for LTL *with common knowledge*, which utilizes the same type of argument. Another paper which utilizes this argument is [vBP06], in which it is shown that several variants of branching-time logics *with common knowledge operators*[1] have an undecidable satisfiability problem. But, to our knowledge, this is the first time an epistemic temporal logic *without a common knowledge operator* is shown to have an undecidable satisfiability problem.

We also investigate here the model-checking problem for CTL with knowledge. The model-checking problem for (a generalized form of) a branching-time logic with knowledge operators and without common knowledge has been studied in [SG02], where the approach is to code the model-checking problem as a satisfiability problem in Chain Logic [ER66,Tho92].

We take here a direct approach, by adapting the classical model-checking algorithm of [CES86]. The extra procedure that is needed is a state labeling with knowledge formulas. This involves a subset construction on the given model, since one needs to identify all histories which may be identically observed by agent i, when one wants to label states with formulas involving the K_i modality. Note also that our approach is similar to the model-checking algorithm for LTL with knowledge from [vdMS99], which also involves a subset construction, optimized for achieving better complexity.

Our approach does not improve the worst-case complexity of the algorithm, since each nesting of knowledge operators induces an exponential explosion, thus leading to a nonelementary complexity. But we believe that our approach could be more practical for formulas with low nesting of knowledge operators. In the approach of [SG02], the system is first translated into the Chain Logic, which needs then to be coded into Monadic Second Order logic [Tho92], and then an MSO-based tool like Mona [EKM98] has to be applied. In such an approach, since the system coding creates some formula with quantifier alternation, some unnecessary determinization steps for the resulting Büchi automata are then needed. Our approach avoids this, as each non-knowledge operator requires only state relabeling, and no state explosion.

It is interesting to note that CTLK is not the only logic extending CTL in which satisfiability is undecidable but model-checking is decidable. The logic TCTL, a dense-time extension of CTL, bears the same problem [DW99].

The model-checking problem for a branching-time logic with knowledge operators has also been addressed in [LR06]. Their approach is to have state-based observation, and not trace-based observation, and this induces a PSPACE complexity of the model-checking problem. Note however that, in general, state-based observation is not a sychronous and perfect recall semantics.

We have not investigated here the possibility to adapt these results to other semantics, but we believe that our arguments can be extended to handle non-synchronous and/or non-perfect recall semantics.

---

[1] As stated in [vBP06] on page 5, $\mathcal{L}_{ETL}$ "contains all the [...] temporal and knowledge operators", that is, *common knowledge* is included too. Hence Theorem 24 refers to this branching temporal logic *with common knowledge*.

The paper is organized as follows: the next section gives the syntax and semantics of $CTLK_{prs}$. We then present the undecidability result in the third section, and the model-checking algorithm in the fourth section. We end with a section of conclusions and comments.

2 Syntax and semantics of $CTLK_{prs}$

We recall here the syntax and the semantics of CTL, the Computational Tree Logic, with knowledge modalities. Our semantics is a synchronous and perfect recall semantics which is based on observability of atomic proposition values, rather than on an "abstract" observability mapping on system states [FHV04]. We also give the semantics in a "tree-automata" flavor, as the models of a formula are presented as trees – which are unfoldings of transition systems.

We first fix some notations to be used through the paper. Given any set A, we denote by A^* the set of finite sequences over A. Hence, $\mathbb{N}^*$ denotes the set of *finite sequences of natural numbers*. The prefix order on A^* is denoted $\preceq$, hence $abcd \preceq abcde \preceq abcde$. For a partial function $f : A \nrightarrow B$, its *support* is the set of elements of A on which f is defined, and is denoted $\mathsf{supp}(f)$. The first projection of a partial function $f : A \nrightarrow B_1 \times \ldots \times B_n$ is denoted $f\big|_{B_1}$; similar notations are used for all the projections of f.

Given a set of symbols AP, which will denote in the sequel the set of *atomic propositions*, an *AP-labeled tree* is a partial function $t : \mathbb{N}^* \nrightarrow 2^{AP}$ that bears some additional properties, that we detail in the following. First, note that an element x in the support of t denotes a node of the tree, while $t(x)$ denotes the label of that node.

To be a tree, a mapping $t : \mathbb{N}^* \nrightarrow 2^{AP}$ has to satisfy the following properties:

1. The support of t is prefix-closed: for all $x \in \mathsf{supp}(t)$ and all $y \preceq x$, $y \in \mathsf{supp}(t)$.
2. Trees are "full": for all $x \in \mathsf{supp}(t)$, if $xi \in \mathsf{supp}(t)$ for some $i \in \mathbb{N}$, then for all $0 \leq j \leq i$, $xj \in \mathsf{supp}(t)$.
3. Trees are infinite: for all $x \in \mathsf{supp}(t)$ there exists $i \in \mathbb{N}$ s.t. $xi \in \mathsf{supp}(t)$.

For example, the subset of integers $\{\varepsilon, 1^*, 121^*, 131^*, 2, 21^*\}$ is the support of a tree, whereas $\{\varepsilon, 1, 2, 221^*\}$ is not, as it does not satisfy neither the fullness property (for node 2) nor the infinity property (for node 1). Here, ε denotes the empty sequence of integers, and 1^* denotes the set $1^* = \big\{1^n \mid n \geq 0\big\}$.

We say that t_1 is *similar with* a tree t_2 and denote this $t_1 \simeq t_2$ if, intuitively, t_2 is a rearrangement of t_1. Formally, $t_1 \simeq t_2$ if there exists a bijection $\varphi : \mathsf{supp}(t_1) \longrightarrow \mathsf{supp}(t_2)$ for which

- $\varphi(\varepsilon) = \varepsilon$.
- For all $x \in \mathsf{supp}(t_1)$ and $i \in \mathbb{N}$ for which $xi \in \mathsf{supp}(t_1)$ there exists $j \in \mathbb{N}$ such that $\varphi(x)j \in \mathsf{supp}(t_2)$ and $\varphi(xi) = \varphi(x)j$.
- For all $x \in \mathsf{supp}(t_1)$, $t_1(x) = t_2\big(\varphi(x)\big)$

A *finite path* in a tree t is a sequence of elements in the support of t, $(x_i)_{0 \leq i \leq k}$ with x_{i+1} being the immediate successor of x_i $(0 \leq i \leq k-1)$ w.r.t. the prefix order. An *infinite path* is an infinite sequence $(x_k)_{k\geq 0}$ with x_{k+1} being the immediate successor of x_k for all $k \geq 0$. A (finite or infinite) path is *initial* if it starts with ε, the tree root.

Next we define the observability relations for each agent. These relations are given by subsets $AP_i \subseteq AP$ of atomic propositions. The values of atoms in AP_i are supposed to be observable by agent i, and no other atoms are observable by i. Informally, an agent i does not distinguish whether the current state of the system is represented by the node x or by the node y in the tree if:

- x and y lie on the same level of the tree and
- The sequence of atomic propositions that agent i can observe *along the initial path that ends in* x is the same as the sequence of atomic propositions i can observe *along the initial path that ends in* y.

The first requirement makes this semantics *synchronous*, as it codes the fact that any agent knows the current absolute time, and the second requirement gives the *perfect recall* attribute of this semantics, as it encodes the fact that each agent records all the observations he has made on the system state, and updates his knowledge based on his recorded observations.

Formally, given an AP-labeled tree t, a subset $AP_i \subseteq AP$ and two positions x, y we denote $x \sim_{AP_i} y$ if

1. x and y are on the same level in the tree, i.e., $x, y \in \mathsf{supp}(t)$, $|x| = |y|$,
2. For any pair of nodes $x', y' \in \mathsf{supp}(t)$ with $x' \preceq x$, $y' \preceq y$ and $|x'| = |y'|$ we have that $t(x') \cap AP_i = t(y') \cap AP_i$.

Figure 1 gives an example of a (finite part of a) tree and some pairs of nodes which are or are not related by the two observability relations $\sim_{AP_1}$ and $\sim_{AP_2}$.

The logic we investigate here, which is the Computational Tree Logic with knowledge operators and with a synchronous and perfect recall semantics, denoted in the following as $CTLK_{prs}$, has the following syntax:

$$\phi ::= p \mid \phi \wedge \phi \mid \neg\phi \mid A\bigcirc\phi \mid \phi\, A\mathcal{U}\, \phi \mid \phi\, E\mathcal{U}\, \phi \mid K_i\phi$$

The semantics of $CTLK_{prs}$ is given in terms of tuples (t, x) where t is an AP-labeled tree, $x \in \mathsf{supp}(t)$ is a position in the tree, $AP_i \subseteq AP$ are some fixed subsets ($1 \leq i \leq n$), and $\sim_{AP_i}$ are the above-defined observability relations:

$(t,x) \models p$ if $p \in t(x)$

$(t,x) \models \phi_1 \wedge \phi_2$ if $(t,x) \models \phi_j$ for both $j = 1, 2$

$(t,x) \models \neg\phi$ if $(t,x) \not\models \phi$

$(t,x) \models A\bigcirc\phi$ if for all $i \in \mathbb{N}$ with $xi \in \mathsf{supp}(t)$, $(t, xi) \models \phi$

$(t,x) \models \phi_1\, A\mathcal{U}\, \phi_2$ if for any infinite path $(x_k)_{k\geq 0}$ in t with $x_0 = x$ there exists $k_0 \geq 1$ with $(t, x_{k_0}) \models \phi_2$ and $(t, x_j) \models \phi_1$ for all $0 \leq j \leq k_0 - 1$

$(t,x) \models \phi_1\, E\mathcal{U}\, \phi_2$ if there exists a finite path $(x_j)_{1\leq j\leq k_0}$ in t with $x_0 = x$, $(t, x_{k_0}) \models \phi_2$ and $(t, x_j) \models \phi_1$ for all $0 \leq j < k_0$

$(t,x) \models K_i\phi$ if for any $y \in \mathsf{supp}(t)$ with $x \sim_{AP_i} y$ we have $(t, y) \models \phi$

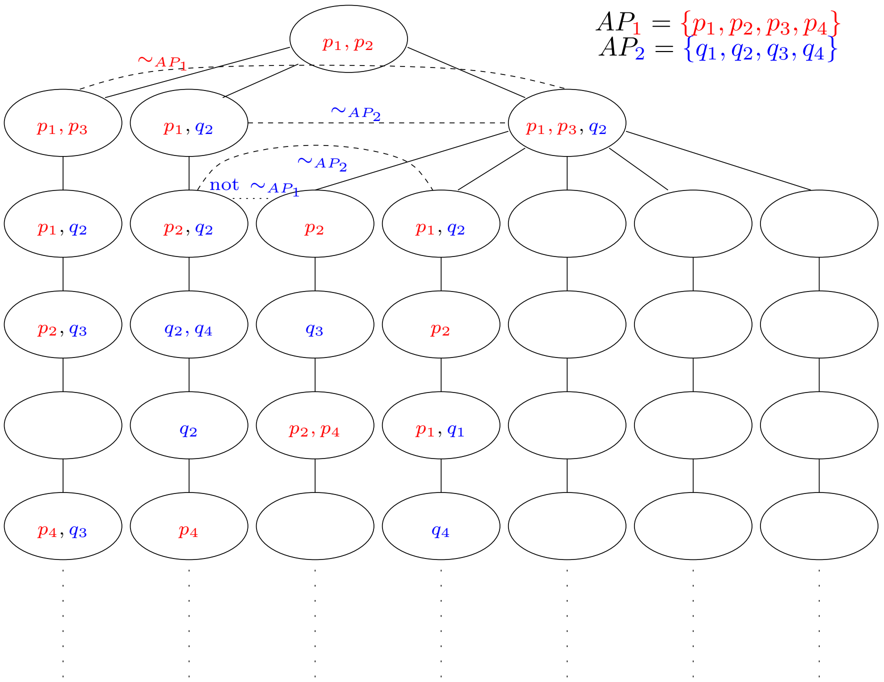

Fig. 1. An AP-tree, with some of the $\sim_{AP_1}$ relations and $\sim_{AP_2}$ relations represented as dashed lines. The dotted line shows two nodes that are not in $\sim_{AP_1}$ relation since the history that can be observed by agent 1 in both nodes is not the same, though the atomic propositions which are observed by 1 in each node is the same.

Remark 1. It is easy to note that for any pair of similar trees $t_1 \simeq t_2$, where the similarity relation is given by the bijection $\varphi : \mathsf{supp}(t_1) \to \mathsf{supp}(t_2)$, for any CTLK formula ϕ, and any position $x \in \mathsf{supp}(t_1)$, we have that

$$(t_1, x) \models \phi \text{ if and only if } (t_2, \varphi(x))$$

The usual abbreviations apply here too, in particular

$$\begin{array}{cc} E\Diamond\,\phi = \mathit{true}\ E\mathcal{U}\,\phi & A\Box\,\phi = \neg\,E\Diamond\,\neg\phi \\ A\Diamond\,\phi = \mathit{true}\ A\mathcal{U}\,\phi & E\Box\,\phi = \neg\,A\Diamond\,\neg\phi \\ P_i\phi = \neg K_i\neg\phi & E\bigcirc\,\phi = \neg\,A\bigcirc\,\neg\phi \end{array}$$

A formula ϕ is *satisfiable* if there exists a tree t and a position $x \in \mathsf{supp}(t)$ such that $(t, x) \models \phi$.

3 Undecidability of satisfiability

This section presents our undecidability result for the satisfiability problem in $CTLK_{prs}$. The undecidable problem that will be simulated is the following:

Problem 1 (Infinite Visiting Problem). Given a deterministic Turing machine, once it starts with a blank tape, does it visit all the cells of its tape?

It is easy to see that this problem is co-r.e.: just construct a multi-tape TM simulator, in which one of the tapes serves for memorizing all the previously visited configurations. The simulator machine also fixes, at start, a marker at some cell, as a "guess" that the R/W head will never go beyond that marker. It also memorizes, for each reachable configuration, only the part of the input tape up to the marker.

The simulator machine simulates one step of the given TM, and first checks whether it has reached a final state, in which case it halts. If not, it checks whether the new configuration has ever been reached by checking whether it is present between the memorized configurations. If yes, then it goes into an error state in which it fills all the input tape with an error symbol. If not, it appends the current configuration to the memorized configurations and continues.

As it might be possible that the marker be reached during computation, that is, that the simulator reaches more cells during its computation that the amount that was initially guessed, then the simulator pushes the marker one cell to the right, and appends one blank cell to each of the memorized configurations.

The first result of this paper is the following:

Theorem 1. *Satisfiability of $CTLK_{prs}$ formulas is undecidable.*

Proof. The main idea is to simulate the (complement of the) Infinite Visiting Problem 1 by a $CTLK_{prs}$ formula ϕ_T. Similarly to [vdM98], a tree that would satisfy ϕ_T would have configurations of the given TM coded as inital paths, and the observability relations $\sim_{AP_1}$ and $\sim_{AP_2}$ would be used to code transitions between configurations.

So take a deterministic Turing machine $T = (Q, \Sigma, \delta, q_0, F)$ with $\delta : Q \times \Sigma \to Q \times \Sigma \times \{L, R\}$. Assume, without loss of generality, that the transitions in δ always change state, and that δ is total – hence T may only visit all its tape cells, or cycle through some configuration, or halt because trying to move the head to the left when it points on the first tape cell.

The formula ϕ_T will be constructed over the set of atomic propositions consisting of:

1. Four copies of Q, denoted Q, Q', $\overline{Q}$ and $\overline{Q}'$.
2. Two copies of Σ, denoted Σ and Σ'.
3. An extra symbol $\perp$.

The symbol $\perp$ will be used as a marker of the right end of the available space on the input tape on which T will be simulated, and its position will be "guessed" at the beginning of the simulation. The utility of the copies of Q and Σ is explained in the following, along with the way computations of T are simulated.

The computation steps of T are simulated by inital paths in the tree satisfying ϕ_T, initial paths which can be of two types:

1. Type 1 paths, representing instantaneous configurations.
2. Type 2 paths, representing transitions between configurations.

In a type 1 path representing an instantaneous configuration (q, w, i) (where w is the contents of the tape and i is the head position), the configuration is coded using atomic propositions from Q and Σ in a straightforward way:

- The first node of the path (i.e. the tree root) bears no symbol – it is used as the "tape left marker".
- If we consider that all initial paths start with index 0 (which is the tree root), then the contents of cell j, say, symbol $w_j = a \in \Sigma$, is an atomic proposition that holds in the jth node of the path.
- Moreover, at each position j along the path only the tape symbol w_j holds. That is, satisfiability of symbols from Σ is mutually exclusive.
- i is the unique position on the path on which the atomic proposition q holds. That is, satisfiability of symbols from Q is also mutually exclusive.
- Whenever a symbol in $Q \cup \Sigma$ holds, the corresponding primed symbol in $Q' \cup \Sigma'$ holds too.
- The whole path contains a position at which $\perp$ holds and from there on it holds forever.
- No symbol from $Q \cup \Sigma$ holds when $\perp$ holds. This codes the finite amount of cell tapes used during simulation.
- At the point where $\perp$ holds, the symbols $\overline{q}$ and $\overline{q}'$ (recall that q is the current state of the Turing Machine). This is needed for coding the connection between type 1 paths and type 2 paths.

In a type 2 path representing a transition between two configurations, say, $(q, w, i) \vdash (r, z, j)$, the unprimed symbols (i.e. symbols from $Q \cup \Sigma$) along the path represent the configuration *before* the transition, while the primed symbols (i.e. symbols from $Q' \cup \Sigma'$) represent the configuration *after* the transition, in a way completely similar to the above description. Hence, we will have some position on the path where symbol q holds, and another position (before or after) where symbol r' holds. Also $\perp$ marks the limit of the available tape space, and $\overline{q}$ and $\overline{r}'$ hold wherever $\perp$ holds.

It then remains to connect type 1 paths (representing instantaneous configurations) with type 2 paths (representing transitions), by means of the observability relations. This connection will be implemented using two agents and their observability relations: one agent being able to see atomic propositions from $Q \cup \overline{Q} \cup \Sigma$, the other seeing $Q' \cup \overline{Q}' \cup \Sigma'$. Formally, $AP = AP_1 \cup AP_2 \cup \{\perp\}$ with:

$$AP_1 = Q \cup \overline{Q} \cup \Sigma \qquad\qquad AP_2 = Q' \cup \overline{Q}' \cup \Sigma'$$

Note that both agents cannot see the value of the symbol $\perp$.

More specifically, we will connect, by means of $\sim_{AP_1}$, each type 1 path represeting some configuration (q, w, i) with a type 2 path representing a transition $(q, w, i) \vdash (r, z, j)$. Then, by means of $\sim_{AP_2}$, we code the connection between that type 2 path and another type 1 path, which represents the configuration

(r, z, j). The first type of connection is imposed by means of the operator K_1, whereas the second type of connection is ensured by the employment of K_2.

We give in Figure 2 an example of the association between a part of the computation of a Turing machine and a tree. The tree presented in Figure 2 is associated with the following computation of the Turing machine: $(q_0 ab, 1) \vdash^{\delta_1} (q_1, cb, 2) \vdash^{\delta_2} (q_2, cBB, 3) \vdash^{\delta_3} (q_3, cBa, 2)$ where the transitions applied at each step are the following: $\delta_1(q_0, a) = (q_1, c, R)$, $\delta_2(q_1, b) = (q_2, B, R)$, $\delta_1(q_2, B) = (q_3, a, L)$. The tree simulates a "guess" that the Turing machine will utilize strictly less than 5 tape cells: on each run there are at most 4 tape cells simulated before the $\perp$ symbol. Here B is the blank tape symbol, whereas R, resp. L denote the commands "move head to the right", resp. "to the left". Note also that the tree node labeled with $\perp, \overline{q}_0, \overline{q}'_0$ is AP_1-similar with the node labeled $\perp, \overline{q}_0, \overline{q}'_1$, which is AP_2-similar with the node labeled $\perp, \overline{q}_1, \overline{q}'_1$. This connection implemented by the composition of $\sim_{AP_1}$ with $\sim_{AP_2}$ encodes the first step in the above computation.

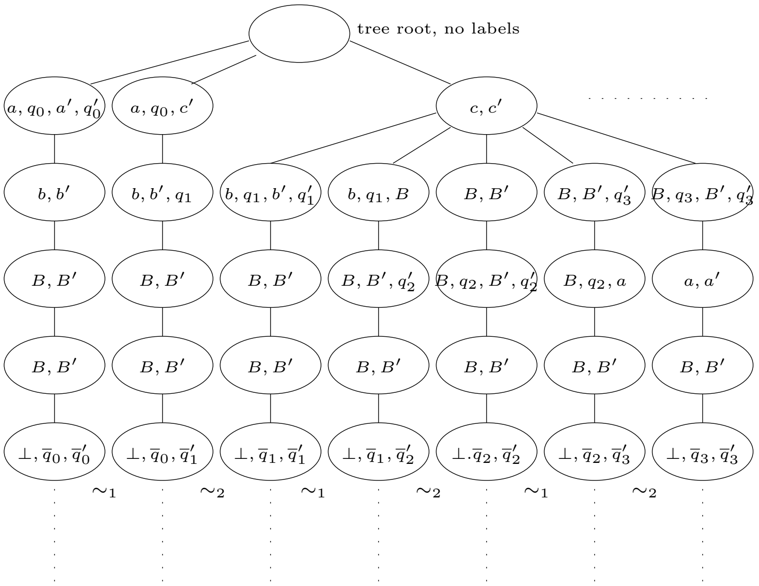
tree root, no labels
a, q_0, a', q'_0
a, q_0, c'
c, c'
. . . .
b, b'
b, b', q_1
b, q_1, b', q'_1
b, q_1, B
B, B'
B, B', q'_3
B, q_3, B', q'_3
B, B'
B, B'
B, B'
B, B', q'_2
B, q_2, B', q'_2
B, q_2, a
a, a'
B, B'
$\perp, \overline{q}_0, \overline{q}'_0$
$\perp, \overline{q}_0, \overline{q}'_1$
$\perp, \overline{q}_1, \overline{q}'_1$
$\perp, \overline{q}_1, \overline{q}'_2$
$\perp . \overline{q}_2, \overline{q}'_2$
$\perp, \overline{q}_2, \overline{q}'_3$
$\perp, \overline{q}_3, \overline{q}'_3$
$\sim_1$
$\sim_2$

Fig. 2. Simulating a sequence of transitions of a Turing machine within a CTLK tree.

Formally, ϕ_T is the conjunction of the following formulas:

1. ϕ_1, specifying that symbols in the same unprimed/primed/overlined set are mutually exclusive:

$$\phi_1 : A\Box\Big(\bigwedge_{q,r\in Q, q\neq r} \big(\neg(q\wedge r)\wedge\neg(q'\wedge r')\wedge\neg(\overline{q}\wedge\overline{r})\wedge\neg(\overline{q}'\wedge\overline{r}')\big)\wedge \\ \bigwedge_{a,b\in\varSigma, a\neq b}\big(\neg(a\wedge b)\wedge\neg(a'\wedge b')\big)\Big)$$

2. ϕ_2, specifying that on each inital path there exists a single occurrence of a state symbol, which marks the position of the R/W head:

$$\phi_2 : \big(A\Diamond\bigvee_{q\in Q} q\big)\wedge\big(A\Diamond\bigvee_{q\in Q} q'\big)\wedge\bigwedge_{q\in Q} A\Box\Big((q\rightarrow A\bigcirc A\Box\neg q)\wedge(q'\rightarrow A\bigcirc A\Box\neg q')\Big)$$

3. ϕ_3, which, in combination with ϕ_7 and ϕ_9 below, is used to encode the fact that the simulation of T is done on a finite tape, whose end is marked with $\bot$:

$$\phi_3 : A\Diamond\bot\wedge A\Box\Big(\bot\rightarrow A\Box\big(\bot\wedge\bigwedge_{z\in\varSigma\cup Q}(\neg z\wedge\neg z')\big)\Big)$$

4. ϕ_4 which copies the value of the current state of the configuration into the position of the end marker $\bot$ – this is useful for connecting configurations via K_1 and K_2:

$$\phi_4 : A\Box\big(q\rightarrow A\Box(\bot\rightarrow\overline{q})\big)\wedge A\Box\big(q'\rightarrow A\Box(\bot\rightarrow\overline{q}')\big)$$

5. ϕ_5, specifying that on a path of the first type the primed and unprimed symbols are the same:

$$\phi_5 : A\Box\bigwedge_{q\in Q}\Big(q\wedge q'\rightarrow A\Box\bigwedge_{a\in\varSigma} a\leftrightarrow a'\Big)\wedge A\Box\bigwedge_{q\in Q}\Big(E\Diamond(q\wedge q')\rightarrow\bigwedge_{a\in\varSigma} a\leftrightarrow a'\Big)$$

6. ϕ_6, specifying that on a path of the second type, the primed and unprimed symbols are almost everywhere the same, excepting the current position of the R/W head:

$$\phi_6 : A\Box\bigwedge_{(q,a)\in\mathsf{supp}(\delta)}\Big(q\wedge a\wedge\neg q'\rightarrow A\Box A\bigcirc\bigwedge_{c\in\varSigma} c\leftrightarrow c'\Big)\wedge \\ \bigwedge_{(q,a)\in\mathsf{supp}(\delta)}\Big(E\Diamond E\bigcirc(q\wedge a\wedge\neg q')\rightarrow\bigwedge_{c\in\varSigma} c\leftrightarrow c'\Big)\wedge$$

7. ϕ_7, specifying that if a path is of type 1 and encodes a configuration in which a certain transition can be applied, then there exists a path of type 2 in which that transition is applied. (The unique transition which will be applied is the subject of ϕ_8.) ϕ_7 also specifies that the "target" type 2 path carries the end marker at the same position as the "source" type 1 path:

$$\phi_7 : A\Box\bigwedge_{(q,a)\in\mathsf{supp}(\delta)}\Big(q\wedge a\wedge q'\rightarrow A\Box\big(\bot\rightarrow K_1\bot\wedge P_1\neg\overline{q}'\big)\Big)$$

8. ϕ_8, specifying that each transition which can be applied in a certain configuration must be applied in that configuration:

$$\begin{aligned}\phi_8 : A\Box & \bigwedge_{q\in Q, a\in \Sigma, \delta(q,a)=(r,b,dir)} \big(q \wedge a \wedge \neg q' \to b'\big) \wedge \\ A\bigcirc\, A\Box & \bigwedge_{q\in Q, a\in \Sigma, \delta(q,a)=(r,b,L)} \big(E\bigcirc(q \wedge a \wedge \neg q') \to r'\big) \wedge \\ A\Box & \bigwedge_{q\in Q, a\in \Sigma, \delta(q,a)=(r,b,R)} \big(q \wedge a \wedge \neg q' \to A\bigcirc\, r'\big)\end{aligned}$$

   Recall that T is deterministic, and hence in each configuration at most one transition can be applied. Note also that the $A\bigcirc$ operator is needed at the beginning of the second line above, in order to code the situations when the head is on the first cell and tries to move left – in such situations the machine halts, no next configuration exists, and therefore our formula ϕ_T needs to be unsatisfiable.

9. ϕ_9, specifying that the outcome of a transition, as coded in a type 2 path, is copied, via $\sim_{AP_2}$, into a type 1 path. ϕ_9 also specifies that the "target" type 1 path carries the end marker at the same position as the "source" type 2 path:

$$\phi_9 : \bigwedge_{q\in Q} A\Box\Big(q' \wedge \neg q \to A\Box\big(\bot \to K_2\bot \wedge P_2\overline{q}\big)\Big)$$

10. ϕ_{10}, encoding the initial configuration of T:

$$\phi_{10} : \neg\bot \wedge \Big(\bigwedge_{z\in Q\cup\Sigma} \neg z \wedge \neg z'\Big) \wedge E\bigcirc\Big(q_0 \wedge q_0' \wedge B \wedge B' \wedge E\bigcirc\big((B \wedge B')\, E\mathcal{U}\, \bot\big)\Big)$$

    Here B represents the blank symbol from Σ. Note that the first position in each path represents the beginning of the tape, hence it does not code any tape cell.

Hence the formula ϕ_T ensures the existence of a path coding the initial configuration of T, together with a guess of the amount of tape space needed for simulating T until it stops or repeats an already visited a configuration. ϕ_T will then ensure, by means of ϕ_7, ϕ_8 and ϕ_9, that once a type 1 path encoding an instantaneous configuration exists in the tree, and from that configuration some transition may be fired, then a type 2 path encoding that transition exists, and the resulting configuration is also encoded in another type 1 path of the tree. It then follows that $(t, \varepsilon) \models \phi_T$ for some tree t if and only if there exists a finite subset Z of inital paths such that Z represents the evolution of the Turing machine T, starting with a blank tape, and either halting in a final state, or re-entering periodically into a configuration – that is, iff the Infinite Visiting Problem has a negative answer for T. $\Box$

4 Model-checking $CTLK_{prs}$

In this section we present a direct approach to model-checking $CTLK_{prs}$. Our approach is to reutilize the classical state-labeling technique for CTL model-checking from [CES86], by adding a procedure that does state labeling with K_i formulas. This extra procedure requires that the system be "sufficiently expanded" such that each state be labeled with some knowledge formula that holds in the state. This implies the necessity of a subset construction.

An *n-agent system* is a finite representation of an AP-labeled tree. Formally, it is a tuple $\mathcal{A} = (Q, AP_1, \ldots, AP_n, AP_0, \pi, \delta, q_0)$ where Q is a finite set of *states*, $\pi : Q \to 2^{AP}$ is the state labeling (here $AP = \bigcup_{0 \le i \le n} AP_i$), $\delta : Q \to 2^Q$ is the state transition function, and $q_0 \subseteq Q$ is initial state. The n agents are denoted $1, \ldots, n$, and can observe respectively $AP_1, \ldots, AP_n$, whereas AP_0 is not observable by anyone.

We denote $\pi_i : Q \to AP_i$ $(0 \le i \le n)$ as the mapping defined by $\pi_i(S) = \pi(S) \cap AP_i$. We also abuse notation and use δ as both a function $\delta : Q \to 2^Q$ and a relation $\delta \subseteq Q \times Q$.

The *tree of behaviors generated by* $\mathcal{A}$ is the tree $t_\mathcal{A} : \mathbb{N} \mathrel{\circ\!\!\!\rightarrow} Q \times AP$ defined inductively as follows:

1. $\varepsilon \in \mathsf{supp}(t_\mathcal{A})$ and $t_\mathcal{A}(\varepsilon) = (q_0, \pi(q_0))$.
2. If $x \in \mathsf{supp}(t_\mathcal{A})$ with $t_\mathcal{A}(x) = (q, P)$ and $card(\delta(q)) = k$, then
   - x has exactly k "sons" in t, i.e. $xi \in \mathsf{supp}(t_\mathcal{A})$ iff $1 \le i \le k$
   - there exists a bijection $\sigma : \{1, \ldots, k\} \to \delta(q)$ such that for all $1 \le i \le k$, $t_\mathcal{A}(xi) = (\sigma(i), \pi(\sigma(i)))$

Note that $t_\mathcal{A}\big|_{AP}$, the projection of the tree generated by $\mathcal{A}$ onto AP, is an AP-labeled tree, and, as such, is a model for $CTLK_{prs}$. We then say that $\mathcal{A}$ is a *model* of a $CTLK_{prs}$ formula ϕ if $(t_\mathcal{A}\big|_{AP}, \varepsilon) \models \phi$.

Problem 2 (Model-checking problem for $CTLK_{prs}$). Given an n-agent system $\mathcal{A}$ and a formula ϕ, is $\mathcal{A}$ a model of ϕ?

Theorem 2. *The model-checking problem for $CTLK_{prs}$ is decidable.*

Proof. The technique that we use is to transform $\mathcal{A}$ into another n-agent system $\mathcal{A}'$ generating the same AP-tree (modulo reordering of brother nodes). The new system would have its state space Q' decomposed into states Q'_ψ which satisfy some subformula ψ of ϕ and some states $Q'_{\neg\psi}$ which do not satisfy it. Then we will add a new propositional symbol p_ψ to AP, which will be appended to the labels of Q_ψ (and will not be appended to the labels of $Q_{\neg\psi}$), and iterate the whole procedure by structural induction on ϕ. The new propositional symbol will not be observable by any agent, hence will be appended to AP_0. For the original CTL operators, it will be the case that $\mathcal{A}$ and $\mathcal{A}'$ are the same. Only for the K_i operators we will need to apply a particular subset construction to $\mathcal{A}$ in order to get $\mathcal{A}'$.

So consider first ϕ is in one of the forms $A \bigcirc p$, $p_1 \, A\mathcal{U} \, p_2$, $p_1 \, E\mathcal{U} \, p_2$ or $K_i p$, where p, p_1, p_2 are atomic propositions. The first three constructions presented below are exactly those used for model-checking CTL [CES86], that we re-state here for the sake of self-containment.

For all temporal operators, the main idea is to split the state space into those states that satisfy the formula and those that do not satisfy it. This can be done in linear time, as follows:

For the case of $\phi = A\bigcirc p$, we partition Q in two sets of states:

$$\begin{aligned} Q_{A\bigcirc p} &= \big\{q \in Q \mid \forall q' \in \delta(q), p \in \pi(q')\big\} \\ Q_{\neg\, A\bigcirc p} &= \big\{q \in Q \mid \exists q' \in \delta(q), p \notin \pi(q')\big\} \end{aligned}$$

The following lemma shows that our splitting is correct:

Lemma 1. *$(t_{\mathcal{A}}\big|_{AP}, x) \models A\bigcirc p$ if and only if for $q = t_{\mathcal{A}}(x)\big|_Q$ we have that $q \in Q_{A\bigcirc p}$.*

For the case of $\phi = p_1\, A\mathcal{U}\, p_2$, we partition again Q into two sets of states:

1. $Q_{\neg(p_1\, A\mathcal{U}\, p_2)}$ is the set of states $q \in Q$ for which there exists a subset of states $Q_q \subseteq Q$ such that:
   (a) For all $q' \in Q_q$, $p_1 \in \pi(q)$ and $p_2 \notin \pi(q)$.
   (b) Q_q is strongly connected w.r.t. δ.
   (c) There exists a path $\rho = (q_i)_{1\leq i\leq l}$ connecting q to some state $q' \in Q_q$ such that $p_1 \in \pi(q_i)$ and $p_2 \notin \pi(q_i)$ for all $1 \leq i \leq l$.
2. $Q_{p_1\, A\mathcal{U}\, p_2} = Q \setminus Q_{\neg(p_1\, A\mathcal{U}\, p_2)}$.

Similarly to the first case, we get:

Lemma 2. *$(t_{\mathcal{A}}\big|_{AP}, x) \models p_1\, A\mathcal{U}\, p_2$ if and only if for $q = t_{\mathcal{A}}(x)\big|_Q$ we have that $q \in Q_{p_1\, A\mathcal{U}\, p_2}$.*

For the case of $\phi = p_1\, E\mathcal{U}\, p_2$, the partition of Q is the following:

1. $Q_{p_1\, E\mathcal{U}\, p_2}$ is the set of states $q \in Q$ for which there exists some state $q' \in Q$ such that
   (a) $p_2 \in \pi(q')$
   (b) There exists a path $\rho = (q_i)_{1\leq i\leq l}$ connecting q to q' such that $p_1 \in \pi(q_i)$ for all $1 \leq i \leq l$.
2. $Q_{\neg(p_1\, E\mathcal{U}\, p_2)} = Q \setminus Q_{p_1\, E\mathcal{U}\, p_2}$.

As above, we also have:

Lemma 3. *$(t_{\mathcal{A}}\big|_{AP}, x) \models p_1\, E\mathcal{U}\, p_2$ if and only if for $q = t_{\mathcal{A}}(x)\big|_Q$ we have that $q \in Q_{p_1\, E\mathcal{U}\, p_2}$.*

The last construction, for $\phi = K_i p$, no longer labels existing states, but needs to split the states of the given model in order to be able to sufficiently distinguish states that have identical history, as seen by agent i. Therefore, we first build a system which generates the same AP-tree as $\mathcal{A}$, but contains sufficient information about the runs that are identically observable by agent i.

Intuitively, the new automaton is an unfolding of the deterministic automaton (without final states) which accepts the same language as $L(\mathcal{A})\big|_{AP_i}$, that is, the projection of the language of $\mathcal{A}$ onto AP_i.

Given a subset $R \subseteq Q$ and a set of atomic propositions $A \subseteq AP$, we denote

$$\delta_A(R) = \big\{r' \in Q \mid A \subseteq \pi(r') \text{ and } \exists r \in R \text{ with } r' \in \delta(r)\big\}$$

The new system is $\tilde{\mathcal{A}} = (\tilde{Q}, AP_1, \ldots, AP_n, AP_0, \tilde{\pi}, \tilde{\delta}, \tilde{q}_0)$ where:

$$\begin{aligned}\tilde{Q} &= \big\{(q, R) \mid R \subseteq Q, \forall r \in R, \pi_i(q) = \pi_i(r)\big\}\\ \tilde{q}_0 &= \big(q_0, \{q_0\}\big)\end{aligned}$$

and for all $(q, R) \in \tilde{Q}$,

$$\begin{aligned}\tilde{\pi}(q, R) &= \pi(q)\\ \tilde{\delta}(q, R) &= \big\{(q', R') \mid q' \in \delta(q), R' = \delta_{\pi_i(q')}(R)\big\}\end{aligned}$$

The following proposition says that the tree generated by $\tilde{\mathcal{A}}$ and the tree generated by $\mathcal{A}$ are the same, modulo node rearrangement:

Proposition 1. $t_{\tilde{\mathcal{A}}}\big|_{AP} \sim t_{\mathcal{A}}\big|_{AP}$.

This result is a consequence of the fact that $\tilde{\mathcal{A}}$ is an *in-splitting* of $\mathcal{A}$ [LM95], that is, the mapping $f : \tilde{Q} \to Q$ defined by $f(q, R) = q$ is a surjective mapping for which, for each $(q, R) \in \tilde{Q}$, there exists a bijection between $\delta(f(q, R))$ and $f(\delta(q, R))$, which preserves the AP-labels.

We then partition $\tilde{Q}$ into two sets of states according to whether $K_i p$ holds or not:

$$\begin{aligned}\tilde{Q}_{K_i p} &= \big\{(q, R) \in \tilde{Q} \mid p \in \pi(q) \text{ and } \forall r \in R, p \in \pi(r)\big\}\\ \tilde{Q}_{\neg K_i p} &= \tilde{Q} \setminus Q_{K_i p}\end{aligned}$$

Lemma 4. *$(t_{\tilde{\mathcal{A}}}\big|_{AP}, x) \models K_i p$ if and only if for $(q, R) = t_{\tilde{\mathcal{A}}}(x)\big|_{\tilde{Q}}$ we have that $(q, R) \in \tilde{Q}_{K_i p}$.*

It then only remains to augment AP in each system such that it includes a new atomic propostion p_ϕ, where ϕ is the formula that was used to partition the state space. Formally, assume that, from the initial system $\mathcal{A}$ and formula ϕ we construct the system $\mathcal{B} = (Q', AP_1, \ldots, AP_n, AP_0, \pi', \delta', q_0')$ (which in most of the cases above is again $\mathcal{A}$!), and that its state space is partitioned into Q_ϕ and $Q_{\neg\phi}$, with $Q_\phi \cup Q_{\neg\phi} = Q, Q_\phi \cap Q_{\neg\phi} = \emptyset$. We then construct $\mathcal{B}' = (Q', AP_1, \ldots, AP_n, AP_0', \pi'', \delta', q_0')$ by augmenting the set of atomic proposition with a new propositional symbol p_ϕ, $AP_0' = AP_0 \cup \{p_\phi\}$, and relabel accordingly all states: for all $q \in Q'$,

$$\pi''(q') = \pi'(q') \cup \big\{p_\phi \mid q \in Q_\phi\big\}$$

The final system $\mathcal{A}_{fin}$ gives the answer to the model-checking problem: $\mathcal{A} \models \phi$ if and only if the initial state in $\mathcal{A}_{fin}$ si labeled with p_ϕ. $\square$

The complexity of the algorithm is nonelementary, as each knowledge subformula involves a subset construction. It then follows that the complexity of model-checking CTL with knowledge is similar to the complexity of model-checking LTL with knowledge on a synchronous & perfect recall semantics – which, as shown by [vdMS99], is nonelementary. Note that this is in contrast with the classic case, in which model-checking CTL is easier than model-checking LTL [CES86].

5 Conclusions

We have shown that satisfiability is undecidable for CTL extended with knowledge operators, in a synchronous and perfect recall semantics. Our result holds in the absence of common knowledge operators, unlike some other well-known undecidability results on temporal epistemic logics.

We have also given a direct decision procedure for the model-checking problem for $CTLK_{prs}$, which extends the classical algorithm from [CES86]. The algorithm is based on a subset construction which associates with each state q the set of states within the model that give the same observed history for some agent as all runs that reach q.

The undecidability result was obtained while trying to give an automata-based procedure for model-checking CTL. The author tried to build a class of automata for which the "star-free" subclass would be equivalent with formulas in $CTLK_{prs}$. Unfortunately it appeared that no such automata had a decidable emptiness problem. The exact class of automata that would be equivalent with some extension of $CTLK_{prs}$ with "counting" capabilities is still to be identified.

Both our results rely on the synchrony and perfect recall assumptions. It would be interesting to investigate whether any of these can be relaxed, and also whether they can be translated in a non-learning semantics.

The author acknowledges the many discussions with Dimitar Guelev on temporal logics with knowledge, during his visit at the LACL in October-November 2007.

References

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions of Programming Languages and Systems*, 8(2):244–263, 1986.

[DW99] M. Dickhöfer and Th. Wilke. Timed alternating tree automata: The automata-theoretic solution to the TCTL model checking problem. In *Proceedings of ICALP'99*, volume 1644 of *LNCS*, pages 281–290. Springer, 1999.

[EKM98] J. Elgaard, N. Klarlund, and A. Møller. Mona 1.x: New techniques for ws1s and ws2s. In *Proceedings of CAV'98*, volume 1427 of *Lecture Notes in Computer Science*, pages 516–520. Springer, 1998.

[ER66] C.C. Elgot and M.O. Rabin. Decidability and undecidability of extensions of second (first) order theory of (generalized) successor. *Journal of Symbolic Logic*, 31(2):169–181, 1966.

[FHV04] R. Fagin, J. Halpern, and M. Vardi. *Reasoning about knowledge*. The MIT Press, 2004.

[HV86] 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. Online version at `https://www.cs.rice.edu/~vardi/papers/stoc86r1.pdf.gz`.

[HV89] 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.

[KLN$^+$06] 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.

[LM95] D. Lind and B. Marcus. *An Introduction to Symbolic Dynamics and Coding*. Cambridge University Press, 1995.

[LR06] 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.

[RL05] 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.

[SG02] N.V. Shilov and N.O. Garanina. Model checking knowledge and fixpoints. In *Proceedings of FICS'02*, pages 25–39, Extended version available as Preprint 98, Ershov Institute of Informatics, Novosibirsk, 2002.

[Tho92] W. Thomas. Infinite trees and automaton-definable relations over ω-words. *Theoretical Computer Science*, 103(1):143–159, 1992.

[vBP06] J. van Benthem and E. Pacuit. The tree of knowledge in action: Towards a common perspective. In Guido Governatori, Ian M. Hodkinson, and Yde Venema, editors, *Proceedings of AiML'06*, pages 87–106. College Publications, 2006.

[vdM98] R. van der Meyden. Common knowledge and update in finite environments. *Information and Computation*, 140(2):115–157, 1998.

[vdMS99] 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 *LNCS*, pages 432–445, 1999.

[vdMS04] Ron van der Meyden and Kaile Su. Symbolic model checking the knowledge of the dining cryptographers. In *Proceedings of the 17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004)*, pages 280–. IEEE Computer Society, 2004.