Non-elementary speed up for model checking synchronous perfect recall

Mika Cohen and **Alessio Lomuscio**

Department of Computing
Imperial College London

Abstract. We analyse the time complexity of the model checking problem for a logic of knowledge and past time in synchronous systems with perfect recall. Previously established bounds are k-exponential in the size of the system for specifications with k nested knowledge modalities. We show that the upper bound for positive (respectively, negative) specifications is polynomial (respectively, exponential) in the size of the system irrespective of the nesting depth.

1 Introduction

When reasoning about multi-agent systems it is common to assume that the agents evolve *synchronously* in a lock-step manner at each tick of a global clock and that each agent has a *perfect recall* of all the information it has been exposed to so far in the run. Synchronous systems with perfect recall are thus widely studied in artificial intelligence and knowledge representation [6]. Specifically, various work have addressed the completeness and decidability problem of linear time epistemic languages (i.e., those built on the logic LTLK) defined under the assumption of synchronous perfect recall [5].

Model checking is a key technology in computational logic for automatically verifying systems against properties expressed in temporal logic [3]. The model checking problem amounts to calculating the satisfaction of $M_S \models \phi_P$, where M_S is an appropriate model for a system S, and ϕ_P is a formula in temporal logic representing a system property P. More recently model checking has been extended to multi-agent systems and properties specified in temporal-epistemic logics [8, 10, 11]. While much progress has been made in developing efficient model checking tools for temporal-epistemic languages, the computational complexity of the model checking problem has received comparatively less attention.

The program complexity (i.e., the model checking complexity in terms of the size of the model) for synchronous systems with perfect recall is known to be k-EXPTIME for temporal-epistemic formulae with k nested epistemic modalities [15]. It is reasonable therefore to look for non-trivial language fragments with an easier complexity.

Many temporal-epistemic properties occurring in practice can be expressed without future-time modalities in the past-time fragment of LTLK [7]. In this paper we analyse the complexity of the model checking problem for past LTLK on synchronous systems with perfect recall. Specifically we show that the program complexity is in PTIME for positive specifications and in EXPTIME for negative specifications irrespective of the number of nested modalities. In general, the program complexity is shown to be k-EXPTIME for an arbitrary specification with k alternations between epistemic diamonds and negations. The number of these alternations is no larger than the non-elementary factor in existing complexity bounds, i.e., the nesting depth of epistemic diamonds, and may be arbitrarily smaller, thereby resulting in a non-elementary speed-up when compared to known procedures.

Overview of paper. The rest of the paper is organised as follows. In section 2 we interpret pure past-time LTL+K on synchronous systems with perfect recall. In Section 3 we outline the existing model checking algorithms. In Section 4 we present the improved upper bounds for the model checking problem. The new bounds are established by means of an automata construction given in Section 5. Appendix A presents basic definitions in automata-theory used in the proofs.

2 Past LTLK

We model computational systems as transition systems and express design requirements in the logic of knowledge and past time (past LTLK); this section summarises the basic definitions.

Transition system with path equivalences. We assume that the computational system under analysis is given as a *transition system* $\mathcal{S} = \langle S, R, I_0 \rangle$ consisting of a finite set S of *system states* s, a *transition relation* $R \subseteq S \times S$, and a non-empty set $I_0 \subseteq S$ of *initial states*. We assume each state s is a subset $s \subseteq P$ of a given set P of atomic propositions; intuitively, state s is made up of the atomic facts that hold at s.

A word is a finite sequence $\sigma \in S^*$ over alphabet S. We write $\sigma \cdot s$ for the result of appending state s to word σ, i.e., $\sigma \cdot s = s_0, s_1, \ldots, s_t, s$ if $\sigma = s_0, s_1, \ldots, s_t$. We write $\sigma' \leq \sigma$ if word σ' is a prefix of σ, i.e., $\leq$ is the least reflexive and transitive relation on S^* such that $\sigma \leq \sigma \cdot s$. A *(computation) path* of $\mathcal{S}$ is a word $\sigma = s_0, s_1, \ldots, s_t \in S^+$ such that $s_0 \in I_0$ and $(s_i, s_{i+1}) \in R$ for all $0 \leq i < t$. We write *last*(σ) for the last state s_t in the computation path σ.

We assume a finite set Ag of agents; each agent $a \in Ag$ observes a subset $P_a \subseteq P$ of the propositional atoms. Intuitively, the set $s \cap P_a$ represents the local view of agent a at the system state s. We say that two system states s and s' are *equivalent* w.r.t agent a, in symbols $s \sim_a s'$, if they agree on the propositions observed by agent a, i.e., if $s \cap P_a = s' \cap P_a$.

Definition 2.1. *Synchronous perfect recall equivalence w.r.t. agent a is the least reflexive relation $\sim_a \subseteq S^* \times S^*$ such that $\sigma \cdot s \sim_a \sigma' \cdot s'$ iff $\sigma \sim_a \sigma'$ and $s \sim_a s'$, for all $\sigma, \sigma' \in S^*$ and all $s, s' \in S$.*

In short, words over S^* are equivalent if they are of the same length and are point-wise equivalent.

Language of past LTLK LTLK [7] extends LTL with knowledge modalities interpreted through path equivalences. In this paper, we consider the following past-time fragment:

$$\phi \quad ::= \quad p \mid \neg\phi \mid \phi \wedge \phi' \mid \phi \vee \phi' \mid k_a\,\phi \mid \Diamond\phi \mid \ \ominus\,\phi \mid \overline{\ominus}\phi$$

where $p \in P$ is an atomic proposition and $a \in Ag$. The knowledge diamond k_a is read "Agent a considers it possible that", the temporal diamond $\Diamond$ is read "Once", the strong yesterday modality $\ominus$ is read "Yesterday", and the weak yesterday modality $\overline{\ominus}$ is read "Yesterday, if there was a yesterday". We omit the temporal modality for "since" for ease of presentation. The *length* $|\phi|$ of a formula ϕ is the number of symbols in ϕ excluding negations, i.e., the number of atomic propositions, conjunctions, disjunctions, and temporal/knowledge modalities appearing in ϕ.

A formula ϕ is *closed* if every knowledge diamond is within the scope of a negation, i.e., if ϕ is built from atoms, conjunction, disjunction, temporal modalities and negated arbitrary formulae:

$$\phi \quad ::= \quad p \mid \phi \wedge \phi' \mid \phi \vee \phi' \mid \Diamond\phi \mid \ \ominus\,\phi \mid \overline{\ominus}\phi \mid \neg\psi$$

where ψ is any formula possibly containing knowledge diamonds; a formula is *open* if not closed. A formula ϕ is in *normal form* if negations only apply to atoms and diamonds:

$$\begin{aligned}\phi \quad ::= \quad & p \mid \neg p \mid \phi \wedge \phi' \mid \phi \vee \phi' \mid k_a\,\phi \mid \neg k_a\,\phi \mid \\ & \Diamond\phi \mid \neg\Diamond\phi \mid \ \ominus\,\phi \mid \overline{\ominus}\phi\end{aligned}$$

We can reduce any formula to normal form by eliminating double negations and distributing negations over conjunctions, disjunctions, and yesterday modalities. Given a formula ϕ, the *complement* $\overline{\phi}$ is the formula $\neg\phi$ after reduction to normal form.[1] We introduce box modalities and implication in terms of complementation in the expected way: $K_a\phi$ ("Agent a knows that ϕ") abbreviates $\neg k_a\overline{\phi}$, while $\boxminus\phi$ ("Historically ϕ") stands for $\neg\Diamond\overline{\phi}$; and $\phi \to \phi'$ is short for $\overline{\phi}\vee\phi'$.

We will consider two language fragments. We say a formula ϕ is *negative* if negations in the formula only apply to atoms:

$$\phi \quad ::= \quad p \mid \neg p \mid \phi \wedge \phi' \mid \phi \vee \phi' \mid k_a\,\phi \mid \Diamond\phi \mid \ \ominus\,\phi \mid \overline{\ominus}\phi$$

A formula ϕ is *positive* if it can be obtained from a negative formula by substituting diamonds with boxes:

$$\phi \quad ::= \quad p \mid \neg p \mid \phi \wedge \phi' \mid \phi \vee \phi' \mid K_a\,\phi \mid \boxminus\,\phi \mid \ \ominus\,\phi \mid \overline{\ominus}\phi$$

where the box modalities are abbreviations defined as above.

Lemma 2.2. *If ϕ is positive then $\overline{\phi}$ is negative; if ϕ is negative then $\overline{\phi}$ is positive.*

Proof. By straightforward induction on positive and negative formulae respectively (using the equalities in footnote 1). □

[1] In detail, the complement $\overline{\phi}$ is defined inductively by: $\overline{p} = \neg p$; $\overline{\neg\phi} = \phi$; $\overline{\phi \wedge \phi'} = \overline{\phi} \vee \overline{\phi}$; $\overline{\phi \vee \phi'} = \overline{\phi} \wedge \overline{\phi}$; $\overline{\ominus\phi} = \overline{\ominus}\overline{\phi}$; $\overline{\overline{\ominus}\phi} = \ominus\overline{\phi}$; $\overline{k_a\phi} = \neg k_a\phi$; $\overline{\Diamond\phi} = \neg\Diamond\phi$.

Knowledge depth We consider two measures of the knowledge depth of a formula. The *nesting depth* $nd(\phi)$ is the number of knowledge diamonds applied consecutively in the formula ϕ; the inductive definition is as follows.

- $nd(p) = 0$
- $nd(k_a\phi) = 1 + nd(\phi)$.
- $nd(\Diamond\phi) = nd(\ominus\phi) = nd(\overline{\ominus}\phi) = nd(\neg\phi) = nd(\phi)$.
- $nd(\phi \wedge \phi') = nd(\phi \vee \phi') = max(nd(\phi), nd(\phi'))$.

Example 2.3. $nd((k_a k_b)^n p) = nd((K_a K_b)^n p) = 2n$.

The second measure of knowledge depth is the number of alternations from knowledge diamonds to negations when counting from the inside and out. In detail, the *alternation depth* $ad(\phi)$ is defined inductively by:

- $ad(p) = 0$.
- $ad(k_a\phi) = ad(\phi)$.
- $ad(\neg\phi) = 1 + ad(\phi)$, if ϕ is open.
- $ad(\neg\phi) = ad(\phi)$, if ϕ is closed.
- $ad(\Diamond\phi) = ad(\ominus\phi) = ad(\overline{\ominus}\phi) = ad(\phi)$
- $ad(\phi \wedge \phi') = ad(\phi \vee \phi') = max(ad(\phi), ad(\phi'))$.

Observe there are no alternations from knowledge diamonds to negations in negative formulae.

Lemma 2.4. $ad(\phi) = 0$ *for negative ϕ.*

Proof. By straightforward induction on negative ϕ. □

Example 2.5. $ad((k_a k_b)^n \neg p) = 0$.

Note also that there is at most one alternation in a positive formula.

Lemma 2.6. $ad(\phi) \leq 1$ *for positive ϕ.*

Proof. By induction on positive ϕ. Induction step, knowledge box: $ad(K_a\phi) = ad(\neg k_a\overline{\phi}) = 1 + ad(\overline{\phi}) \leq 1$ since $ad(\overline{\phi}) = 0$ by Lemmas 2.2 and 2.4. Induction step for temporal box $\boxminus$ is shown similarly. Remaining induction steps (and base cases) are straightforward. □

Example 2.7. $ad((K_a K_b)^n p) = ad(\neg((k_a k_b)^n \neg p)) = 1$.

As illustrated by Examples 2.3, 2.5 and 2.7, the alternation depth $ad(\phi)$ may be arbitrarily smaller than the nesting depth $nd(\phi)$, but it is never larger.

Lemma 2.8. $ad(\phi) \leq nd(\phi)$.

Proof. We show that:

$$\begin{aligned} ad(\phi) \quad &\leq \quad nd(\phi), \text{ if } \phi \text{ is closed.} && (1) \\ ad(\phi) \quad &< \quad nd(\phi), \text{ if } \phi \text{ is open.} && (2) \end{aligned}$$

by induction on ϕ. Base case, atomic ϕ. (1): $ad(p) = 0$ and $nd(p) = 0$. (2): p is closed. Induction step, negation: (1): Assume that ϕ is closed. Then $ad(\neg\phi) = ad(\phi) \leq$ (by the induction assumption) $\leq nd(\phi) = nd(\neg\phi)$. Assume instead ϕ is open. Then $ad(\neg\phi) = 1 + ad(\phi) <$ (by the induction assumption) $< 1 + nd(\phi)$ $= 1 + nd(\neg\phi)$, i.e., $ad(\neg\phi) \leq nd(\neg\phi)$. (2): $\neg\phi$ is closed. Induction step, knowledge diamond. (1): $k_a\phi$ is open. (2): $ad(k_a\phi) = ad(\phi)$ $\leq$ (by the induction assumption) $\leq nd(\phi) < nd(\phi) + 1 = nd(k_a\phi)$. Induction step, temporal diamond. (1): $ad(\Diamond\phi) = ad(\phi) \leq$ (by induction assumption) $\leq nd(\phi) = nd(\Diamond\phi)$. (2): If $\Diamond\phi$ is open then so is ϕ. Therefore, $ad(\Diamond\phi) = ad(\phi) <$ (by induction assumption since ϕ is open) $< nd(\phi) = nd(\Diamond\phi)$. The induction steps for remaining operators are shown similarly. □

Corollary 2.9. $ad(\overline{\phi}) \leq nd(\phi)$.

Proof. From Lemma 2.8 since $nd(\overline{\phi}) = nd(\phi)$. □

Semantics Satisfaction of the language above is defined as standard for atomic propositions, boolean operators and temporal modalities, while the knowledge modalities are interpreted by the path equivalences. Informally, $k_a\phi$ holds if proposition ϕ is consistent with the past and present observations of agent a.

Definition 2.10. *Assume a transition system $\mathcal{S}$. The satisfaction relation $\models$ between computation paths σ of $\mathcal{S}$ and formulae ϕ is defined inductively as follows:*

- $\sigma \models p$ *iff* $p \in last(\sigma)$
- $\sigma \models \neg\phi$ *iff* $\sigma \not\models \phi$.
- $\sigma \models \phi \wedge \phi'$ *iff* $\sigma \models \phi$ *and* $\sigma \models \phi'$.
- $\sigma \models \phi \vee \phi'$ *iff* $\sigma \models \phi$ *or* $\sigma \models \phi'$.
- $\sigma \models k_a\,\phi$ *iff* $\sigma' \models \phi$ *for some path* σ' *of* $\mathcal{S}$ *such that* $\sigma \sim_a \sigma'$.
- $\sigma \models \Diamond\phi$ *iff* $\sigma' \models \phi$ *for some path* $\sigma' \leq \sigma$.
- $\sigma \models \ominus\phi$ *iff* $\sigma' \models \ominus\phi$ *for some path* $\sigma' \leq \sigma$ *such that* $|\sigma'| = |\sigma| - 1 \geq 1$.
- $\sigma \models \overline{\ominus}\phi$ *iff* $\sigma' \models \ominus\phi$ *for all* $\sigma' \leq \sigma$ *such that* $|\sigma'| = |\sigma| - 1 \geq 1$.

where $|\sigma|$ is the length of the path σ.

The extension $[[\phi]]$ of formula ϕ in a given system $\mathcal{S}$ is the set of all computation paths σ of $\mathcal{S}$ such that $\sigma \models \phi$ in $\mathcal{S}$. A formula ϕ is valid in the given system, $\mathcal{S} \models \phi$, iff the extension $[[\phi]]$ consists of all computation paths of $\mathcal{S}$. We use the running example from [14].

Example 2.11 (Bounded message delivery [14])**.** *After waiting a random number of time steps, a sender agent sends a bit value to a receiver agent over a channel that delivers messages immediately or with a delay of one time step.*

We model the scenario as a transition system $\mathcal{S}$ with agents a, b and c representing the sender, receiver and channel respectively. We assume an atomic proposition $holds(i, v)$ read "Agent i holds the value v" for each agent $i \in \{a, b, c\}$ and each bit value $v \in \{0, 1\}$. We assume agent i observes only atomic propositions about agent i itself: $P_i = \{holds(i, 0), holds(i, 1)\}$. We define the transition relation R and the set I_0 if initial states such that the set of possible computation paths of the system is given by the regular expression: $\{holds(a, 0)\}^+ \cdot \{holds(c, 0)\}^? \cdot \{holds(b, 0)\}^* + \{holds(a, 1)\}^+ \cdot \{holds(c, 1)\}^? \cdot \{holds(b, 1)\}^*$.

It can be shown that it is always the case that if the receiver has held the bit value 0 for at least one time step then the receiver knows that the sender knows that the receiver holds the value 0:

$$\ominus\, holds(b, 0) \rightarrow K_b K_a\, holds(b, 0) \tag{3}$$

is valid in $\mathcal{S}$. More generally,

$$\ominus^n\, holds(b, v) \rightarrow (K_b K_a)^n\, holds(b, v) \tag{4}$$

is valid in $\mathcal{S}$; it is always the case that if the receiver has held the bit value for n time steps then n levels of knowledge of knowledge have been established. Moreover, the converse:

$$(K_b K_a)^n\, holds(b, v) \rightarrow \ominus^n\, holds(b, v) \tag{5}$$

is valid in $\mathcal{S}$; n levels of knowledge of knowledge can never be reached in less than n time steps.

Many specifications that occur in practice are either positive or negative; (3) and (4) above are positive, while (5) is negative.

3 Existing model checking procedure

Given a model $\mathcal{S}$ and a formula ϕ the model checking problem is the task of deciding whether ϕ is satisfied in $\mathcal{S}$, $\mathcal{S} \models \phi$. Depending on the underlying semantics the model checking problem has different complexity. In the case of synchronous systems with perfect recall the problem is an instance of infinite model checking (since satisfaction $\mathcal{S} \models \phi$ is in effect defined in terms of a Kripke model over the infinite set of possible computation paths of $\mathcal{S}$).

Yet, model checking synchronous systems with perfect recall is known to be decidable. Existing algorithms [1, 2, 4, 9, 13, 14, 15] are based on [14]. The algorithms transform the given system $\mathcal{S}$ into an equivalent, finite Kripke model by means of a subset construction for each nesting level of knowledge modalities in the formula ϕ. As a simple example, to check a formula with a single epistemic modality K_a the algorithms extend each state s in the given transition system with a "knowledge set" $X \subseteq S$ that encodes the states considered possible by agent a. In the induced Kripke model over extended states $\langle s, X\rangle$, it is assumed that $\langle s, X\rangle$ is related to $\langle t, Y\rangle$ if $\langle s, t\rangle \in R$ in the original system $\mathcal{S}$ and $Y = \{t' \mid t' \sim_a t\} \cap \{t' \mid \exists s' \in X$ such that $\langle s', t'\rangle \in R\}$. In other words the condition states that s is related to t in the original system and the updated knowledge set is consistent both with the latest observation and the earlier knowledge set. The temporal modalities for past and future are interpreted through this induced relation on extended states, while the knowledge modality is interpreted by checking that $\langle s, X\rangle \models K_a\phi$ iff $\langle s', X\rangle \models \phi$ for all $s' \in X$.

Each additional nesting level of knowledge modalities in the formula ϕ requires a further knowledge set construction. This leads to a Kripke model of size at least $\mathtt{exp}(nd(\phi), |S|)$, where $\mathtt{exp}$ is the iterated exponentiation operation defined inductively by: $\mathtt{exp}^0(x) = x$ and $\mathtt{exp}^{n+1}(x) = 2^{\mathtt{exp}^n(x)}$. Informally, $\mathtt{exp}^n(x)$ is a power tower

$$2^{2^{\cdot^{\cdot^{x}}}}$$

of 2s of hight n and with x at the top. For instance $\mathtt{exp}^1(5) = 2^5 = 32$ and $\mathtt{exp}^2(5) = 2^{32} = 4\,294\,967\,296$.

Example 3.1. *Since $nd((K_a K_b)^n p) = 2n$ model checking formula (4) using the existing techniques involves constructing a Kripke model of size greater than $\mathtt{exp}(2n, |S|)$.*

It follows that the existing model checking algorithms for synchronous systems with perfect recall run in time at least $\mathtt{exp}(nd(\phi), |S|)$ for a given ϕ in past LTLK. The upper bound is greater; we refer to [15] for details.

4 Improved upper bounds

In this section we improve the upper complexity bound for model checking past LTLK in synchronous systems with perfect recall.

We show that the model checking problem for a *positive* formula can be decided in time polynomial in the size of the system and in time exponential in the length of the formula.

Theorem 4.1. *The model checking problem for a positive formula ϕ can be decided in time $|S|^{2|\phi|}$.*

Existing bounds are by contrast $nd(\phi)$-exponential in the size $|S|$ of the system as we have just seen. Observe that the new upper bound for positive ϕ may in fact be dramatically better than the existing bounds even when the nesting depth $nd(\phi)$ is low: the size $|S|$ of the system is in practice often large.

Example 4.2. *The model checking problem for the positive formula (4) can be decided in time $|S|^{6(n+1)}$ according to Theorem 4.1. This improves on existing techniques that run in time at least $exp(2n, |S|)$.*

We show in addtion that the model checking problem for a *negative* formula can be solved in time exponential (in a polynomial) in the size of the system and in time doubly-exponential in the length of the formula.

Theorem 4.3. *The model checking problem for a negative formula ϕ can be decided in time $2^{2|S|^{|\phi|+1}}$.*

Again observe that if the size $|S|$ of the system is large the new upper bound for negative ϕ may be considerably better than the existing bounds even when the nesting depth $nd(\phi)$ is low.

Example 4.4. *The model checking problem for the negative formula (5) can be decided in time $2^{2|S|^{3n+4}}$ according to Theorem 4.3. Again this improves on existing techniques that run in time at least $exp(2n, |S|) = 2^{2^{\cdot^{\cdot^{|S|}}}}$.*

We show finally that the model checking problem for arbitrary formulae ϕ can be decided in time $ad(\overline{\phi})$-exponential (in a polynomial) in the size of the system.

Theorem 4.5. *The model checking problem for a formula ϕ can be decided in time $\mathrm{exp}(ad(\overline{\phi}), |S|^{|\phi|+ad(\overline{\phi})})^2$.*

While non-elementary, the new upper bound may itself be non-elementary better than the existing bounds: the height $ad(\overline{\phi})$ of the tower $\mathrm{exp}(ad(\overline{\phi}), |S|^{|\phi|+ad(\overline{\phi})})$ may be arbitrarily lower than the height $nd(\phi)$ of the tower $\mathrm{exp}(nd(\phi), |S|)$, and it is never worse (see Corollary 2.9).

We prove Theorems 4.1, 4.3, and 4.5 by way of the following automata-theoretic characterization of validity (we refer to the appendix for some standard terminology from automata theory).

Lemma 4.6. *For any formula ϕ there exists a corresponding automaton $\mathcal{A}_\phi$ of size at most $\mathrm{exp}(ad(\phi), |S|^{|\phi|+ad(\phi)})$ that accepts precisely all the computation paths satisfying ϕ in the given system.*

By Lemma 4.6 the model checking problem for a specification ϕ can be decided by checking whether the language accepted by the automaton $\mathcal{A}_{\overline{\phi}}$ corresponding to the complemented specification $\overline{\phi}$ is empty, i.e., by checking whether $L(\mathcal{A}_{\overline{\phi}}) = \emptyset$. This can be decided in time $\mathrm{exp}(ad(\overline{\phi}), |S|^{|\overline{\phi}|+ad(\overline{\phi})})^2$ since emptiness is decided in time quadratic in the size (number of locations) of an automaton. This establishes Theorem 4.5 since $|\overline{\phi}| = |\phi|$. From Theorem 4.5 we obtain Theorem 4.1 by Lemmas 2.2 and 2.4, and Theorem 4.3 by Lemmas 2.2 and 2.6.

Example 4.7. *To model check the positive formula (4) from Example 2.11 we build an automaton corresponding to its negative complement:*

$$\ominus^n \, holds(b, v) \wedge (k_b k_a)^n \neg \, holds(b, v) \tag{6}$$

and check whether the automaton has an empty language. To evaluate the negative formula (5), we form the automaton for the positive complement:

$$(K_b K_a)^n holds(b, v) \wedge \overline{\ominus}^n \, \neg holds(b, v) \tag{7}$$

and check whether the automaton has an empty language. By Lemmas 2.4, 2.6 and 4.6, the size of the automata for (6) and (7) are at most exponential in n and doubly-exponential in n respectively.

In the next section we prove Lemma 4.6 by constructing an appropriate automaton $\mathcal{A}_\phi$ corresponding to any formula ϕ.

5 Automata construction

We assume a transition system $\mathcal{S} = \langle S, R, I_0 \rangle$ and proceed to construct for any formula ϕ an automaton $\mathcal{A}_\phi$ whose language is the extension $[[\phi]]$ of the formula ϕ in the given system.

First we introduce some auxiliary operations on automata. The *scalar product* of an agent $a \in Ag$ and an automaton $\mathcal{A}$ over the alphabet S (the state space of the given system) is the result of replacing ("multiplying") every transition label s in $\mathcal{A}$ with all a-equivalent labels s'.

Definition 5.1 (Scalar multiplication)**.** *Assume an agent $a \in Ag$ and an automaton $\mathcal{A} = \langle S, Q, Q_0, \rho, F \rangle$ over the alphabet S. The product of a and $\mathcal{A}$ is the automaton $a \star \mathcal{A} = \langle S, Q, Q_0, \rho^{a\star}, F \rangle$ over the alphabet S where: $\rho^{a\star}(q, s) = \bigcup\{\rho(q, s') \mid s' \sim_a s\}$.*

In other words, for every transition $q \xrightarrow{s} q'$ in the automaton $\mathcal{A}$ between locations q and q' labelled by state s and for every equivalent state $s' \sim_a s$, there is a transition $q \xrightarrow{s'} q'$ in the automaton $a \star \mathcal{A}$ from q to q' labelled by the equivalent state s'. Consequently, the automaton $a \star \mathcal{A}_\phi$ accepts a word $\sigma \in S^*$ iff automaton $\mathcal{A}$ accepts some a-equivalent word $\sigma' \sim_a \sigma$.

Remaining auxiliary operations needed are standard. The automaton $\mathcal{A}(F') := \langle S, Q, Q_0, \rho, F' \rangle$ substitutes the set of accepting locations in the automaton $\mathcal{A}$ with the set $F' \subseteq Q$. The *complement* $\mathcal{A}^c := \mathcal{A}(Q - F)$ complements the set of accepting locations in $\mathcal{A}$. The *determinization* $\mathcal{A}^c$ determinizes the automaton $\mathcal{A}$ by means of a subset construction (see Appendix A). The automaton $\Diamond\mathcal{A}$ extends each location in $\mathcal{A}$ with a boolean variable that becomes true when we reach an accepting location $q \in F$ and which stays true from then on; an extended location is accepting in the automaton $\Diamond\mathcal{A}$ if the boolean variable is true. In detail,

$$\Diamond\mathcal{A} \quad := \quad \langle S, Q \times \{0, 1\}, Q_0 \times \{0\}, \rho^{\Diamond}, \{\langle q, 1 \rangle \mid q \in Q\} \rangle$$

where:

- $\rho^{\Diamond}(\langle q, 0 \rangle, s) := \{\langle q', 1 \rangle \mid q' \in \rho(q, s)\}$ if $q' \in F$.
- $\rho^{\Diamond}(\langle q, 0 \rangle, s) := \{\langle q', 0 \rangle \mid q' \in \rho(q, s)\}$ if $q' \notin F$.
- $\rho^{\Diamond}(\langle q, 1 \rangle, s) := \{\langle q', 1 \rangle \mid q' \in \rho(q, s)\}$.

Finally, the automaton $\mathcal{A}_\mathcal{S}$ corresponding to the given system $\mathcal{S}$ accepts precisely all computation path of $\mathcal{S}$ (see Appendix A).

We now have the auxiliary operations needed to construct the automaton $\mathcal{A}_\phi$ corresponding to a formula ϕ.

Definition 5.2 (Formulae to automata)**.** *The automaton $\mathcal{A}_\phi$ corresponding to a formula ϕ is defined inductively as follows:*

1. $\mathcal{A}_p := \mathcal{A}_\mathcal{S}([[p]])$.
2. $\mathcal{A}_{k_a\phi} := (a \star \mathcal{A}_\phi) \times \mathcal{A}_\mathcal{S}$.
3. $\mathcal{A}_{\neg\phi} := (\mathcal{A}_\phi)^c$, *if ϕ is closed.*
4. $\mathcal{A}_{\neg\phi} := ((\mathcal{A}_\phi)^d)^c \times \mathcal{A}_\mathcal{S}$, *if ϕ is open.*
5. $\mathcal{A}_{\phi\wedge\phi'} := \mathcal{A}_\phi \times \mathcal{A}_{\phi'}$.
6. $\mathcal{A}_{\phi\vee\phi'} := \mathcal{A}_\phi \overline{\times} \mathcal{A}_{\phi'}$.
7. $\mathcal{A}_{\Diamond\phi} := \Diamond\mathcal{A}_\phi$.

The automata for $\ominus\phi$ and $\overline{\ominus}\phi$ are constructed analogously to the automaton for $\Diamond\phi$.

In (2) and (4) in Definition 5.2 we form the cross product ($\times$) with the automaton $\mathcal{A}_{\mathcal{S}}$ corresponding to the given system so as to exclude words that are not possible computation paths of the system. The constructions (1), (5), (6) and (7) in Definition 5.2 for atoms, conjunction, disjunction and temporal modalities follow standard lines in compositional automata construction for past LTL (see [12]). The constructions (3) and (4) in Definition 5.2 for negation are novel. To obtain the automaton $\mathcal{A}_{\neg\phi}$ from the automaton $\mathcal{A}_\phi$ when ϕ is closed, we complement the set of accepting locations in $\mathcal{A}_\phi$. When ϕ is open, we first determinize $\mathcal{A}_\phi$ by means of a subset construction. Finally, the construction (2) in Definition 5.2 for the knowledge diamond is also novel as existing model checking techniques for synchronous perfect recall handle knowledge modalities quite differently (see Section 3).

Example 5.3. *The automaton for the second conjunct in formula (6) is* $(b \star (a \star \mathcal{A}_{(k_b k_a)^{n-1} \neg\, holds(b,v)}) \times \mathcal{A}_{\mathcal{S}}) \times \mathcal{A}_{\mathcal{S}}$.

Size of automata. It is easily seen that the alternation depth $ad(\phi)$ of a formula ϕ is equal to the number of subset construction performed when constructing the automaton $\mathcal{A}_\phi$ for the formula ϕ. In detail, we obtain the following lemma.

Lemma 5.4. *The size of $\mathcal{A}_\phi$ is at most* $\mathtt{exp}(ad(\phi), |S|^{|\phi|+ad(\phi)})$.

Proof. By induction on ϕ. Base case, atom.[2] $|\mathcal{A}_p| = |S| = \mathtt{exp}(ad(p), |S|^{|p|+ad(\overline{p})})$. Induction step, negation. Assume ϕ is closed. Then $|\mathcal{A}_{\neg\phi}| = |\mathcal{A}_\phi| \leq$ (by the induction assumption) $\leq \mathtt{exp}(ad(\phi), |S|^{|\phi|+ad(\phi)})$ = (since ϕ is closed) = $\mathtt{exp}(ad(\neg\phi), |S|^{|\neg\phi|+ad(\neg\phi)})$. Assume instead ϕ is open. Then $|\mathcal{A}_{\neg\phi}| = 2^{|\mathcal{A}_\phi|} \cdot |S| \leq$ (by the induction assumption) $\leq 2^{\mathtt{exp}(ad(\phi), |S|^{|\phi|+ad(\phi)})} \cdot |S| = \mathtt{exp}(ad(\phi)+1, |S|^{|\phi|+ad(\phi)}) \cdot |S|$ = (since ϕ is open) = $\mathtt{exp}(ad(\neg\phi), |S|^{|\neg\phi|+ad(\phi)}) \cdot |S| \leq \mathtt{exp}(ad(\neg\phi), |S|^{|\neg\phi|+ad(\phi)+1})$ = (since ϕ is open) = $\mathtt{exp}(ad(\neg\phi), |S|^{|\neg\phi|+ad(\neg\phi)})$. Induction step, knowledge diamond: $|\mathcal{A}_{k_a\phi}| = |\mathcal{A}_\phi| \cdot |S| \leq$ (by the induction assumption) $\leq \mathtt{exp}(ad(\phi), |S|^{|\phi|+ad(\phi)}) \cdot |S| \leq \mathtt{exp}(ad(\phi), |S|^{|\phi|+1+ad(\phi)}) = \mathtt{exp}(ad(k_a\phi), |S|^{|k_a\phi|+ad(k_a\phi)})$. Induction steps for remaining operators follows similarly from the equalities $|\mathcal{A}_{\diamond\phi}| = |\mathcal{A}_{\ominus\phi}| = |\mathcal{A}_{\overline{\ominus}\phi}| = 2 \cdot |S|$ and $|\mathcal{A}_{\phi\wedge\phi'}| = |\mathcal{A}_{\phi\vee\phi'}| = |\mathcal{A}_\phi| \cdot |\mathcal{A}_{\phi'}|$. □

Note that the upper bound in Lemma 5.4 depends on the conditional automata construction for negation; defining $\mathcal{A}_{\neg\phi} := ((\mathcal{A}_\phi)^d)^c \times \mathcal{A}_{\mathcal{S}}$ when ϕ is closed would lead to an upper bound which can be non-elementary worse than the upper bound in Lemma 5.4. For example, the automaton for the formula $(\boxminus k_a)^n p$ would then require $2 \cdot n$ subset constructions instead of the n subset constructions needed now (and needed by the existing model checking techniques).

On the other hand there are some more or less obvious optimizations that would reduce the exponential $|\phi| + ad(\phi)$ in Lemma 5.4. For example, the cross product with $\mathcal{A}_{\mathcal{S}}$ in Definition 5.2 is sometimes unnecessary and could be deferred. However, for ease of presentation we have chosen to ignore optimizations that do not effect the the number of iterated exponentiations.

Correctness To establish Lemma 4.6, it remains to be shown that the constructed automaton $\mathcal{A}_\phi$ is correct, $L(\mathcal{A}_\phi) = [[\phi]]$. First, we show that the construction for negation is correct. We proceed by way of three lemmas.

[2] $|\mathcal{A}_p| = |S| + 1$ but for ease of presentation we approximate this to $|S|$; the size of the state space is typically a very large number.

Lemma 5.5. *If ϕ is closed, then $\mathcal{A}_\phi$ is unambiguous.*

Proof. By induction on closed ϕ. Base case: $\mathcal{A}_p$ is deterministic (and so unambiguous) since $\mathcal{A}_{\mathcal{S}}$ is deterministic. Induction step for negation: Assume ψ is closed. By the induction assumption, $\mathcal{A}_\psi$ is unambiguous. Therefore $\mathcal{A}_{\neg\psi} = (\mathcal{A}_\psi)^c$ is unambiguous. Assume instead ψ is open. Then $\mathcal{A}_{\neg\psi} = ((\mathcal{A}_\psi)^d)^c \times \mathcal{A}_{\mathcal{S}}$ is ambiguous since $(\mathcal{A}_\psi)^d$ is deterministic and so unambiguous, $\mathcal{A}_{\mathcal{S}}$ is unambiguous and $\times$ preserves unambiguity. Induction steps for conjunction, disjunction and temporal modalities: $\times$, $\overline{\times}$, etc. preserve unambiguity. □

Let the language *produced* by any automaton $\mathcal{A}$ be the set $P(\mathcal{A})$ of words that would be accepted if we made every location an accept location, i.e., $\sigma \in P(\mathcal{A})$ iff there exists a run on word σ in $\mathcal{A}$. It follows immediately that the words accepted by the complement $\mathcal{A}^c$ of an unambiguous automaton $\mathcal{A}$ are precisely the words produced but not accepted by $\mathcal{A}$.

Lemma 5.6. *If $\mathcal{A}$ is unambiguous, then $L(\mathcal{A}^c) = P(\mathcal{A}) - L(\mathcal{A})$.*

Proof. Immediate. □

The third and final lemma says that automaton $\mathcal{A}_\phi$ produces exactly the set of possible computation paths of the given system:

Lemma 5.7. $P(\mathcal{A}_\phi) = L(A_{\mathcal{S}})$.

Proof. By induction on ϕ using:

$$P(\mathcal{A} \times \mathcal{A}') = P(\mathcal{A}) \cap P(\mathcal{A}') \quad (8)$$

$$P(A_{\mathcal{S}}) = L(A_{\mathcal{S}}) \quad (9)$$

where (9) follows from the fact that every location in $A_{\mathcal{S}}$ is an accepts location. Base case, atomic propositions: $P(\mathcal{A}_p) = P(A_{\mathcal{S}})$ = (by (9)) = $L(A_{\mathcal{S}})$. Induction step, negation. Assume ϕ is closed. Then $P(\mathcal{A}_{\neg\phi}) = P((\mathcal{A}_\phi)^c)$ = (by the induction assumption and the fact that the operation $(\cdot)^c$ of complementing the accept locations preserve the set of produced words) = $L(A_{\mathcal{S}})$. Assume instead ϕ is open. Then $P(\mathcal{A}_{\neg\phi}) = P(((\mathcal{A}_\phi)^d)^c \times A_{\mathcal{S}})$. But $((\mathcal{A}_\phi)^d)^c$ is deterministic and so produces every word, $P(((\mathcal{A}_\phi)^d)^c) = S^*$. By (8) and (9), $P(\mathcal{A}_{\neg\phi}) = L(A_{\mathcal{S}})$. Induction step, knowledge diamond: From (8) and (9) and the fact that $P(a \star \mathcal{A}_\phi) \supseteq P(\mathcal{A}_\phi)$. Induction steps, temporal modalities: The constructions for temporal modalities preserve the set of produced words. Induction step, disjunction and conjunction: From (8). □

By Lemmas 5.5, 5.6 and 5.7 we can conclude that the construction for negation is correct, $L(\mathcal{A}_{\neg\phi}) = L(A_{\mathcal{S}}) - L(\mathcal{A}_\phi)$. It follows that the automaton $\mathcal{A}_\phi$ is correct.

Lemma 5.8. $L(\mathcal{A}_\phi) = [[\phi]]$.

Proof. By induction on ϕ. The inductive case for negation follows from Lemmas 5.5, 5.6 and 5.7; remaining cases are routine. □

From Lemmas 5.4 and 5.8 we immediately obtain Lemma 4.6, which establishes the main theorem, Theorem 4.5.

6 Conclusion

We considered the model checking complexity for the logic of knowledge and past time in synchronous systems with perfect recall, a class of systems often seen in applications. Previous results show that the program complexity is in k-EXPTIME for specifications with k nested knowledge modalities. We proved that the program complexity is in PTIME (respectively, EXPTIME) for positive (respectively, negative) specifications irrespective of the number of nested modalities.

Acknowledgements The authors would like to thank Nir Piterman for valuable comments on earlier drafts of this paper.

A Finite automata

In this appendix we briefly recall some basic automata theory. An *automaton* over an alphabet Σ is a structure $\mathcal{A} = \langle \Sigma, Q, Q_0, \rho, F\rangle$ consisting of a finite set Q of locations, a non-empty set $Q_0 \subseteq Q$ of initial locations, a (non-deterministic) transition function $\rho : Q \times \Sigma \longrightarrow 2^Q$, and a set $F \subseteq Q$ of accepting locations. A *run* over a word $\sigma = (s_1, \ldots, s_n) \in \Sigma^*$ is a finite sequence $q_0, \ldots, q_n$ of locations such that $q_0 \in Q_0$ and $q_i \in \rho(q_{i-1}, s_i)$ for all $1 \leq i \leq n$. The run $q_0, \ldots, q_n$ is *accepting* if $q_n \in F$. The word $\sigma = s_1, \ldots, s_n$ is *accepted* by the automaton if some run over the word σ is accepting. The *language* accepted by the automaton $\mathcal{A}$ is the set $L(\mathcal{A})$ of words accepted by $\mathcal{A}$. The *size* $|\mathcal{A}|$ of automaton $\mathcal{A}$ is its number $|Q|$ of locations.

Automaton $\mathcal{A}$ is *deterministic* if there is exactly one initial location, $|Q_0| = 1$, and each letter $s \in \Sigma$ determines a functional transition relation, $|\rho(q, s)| = 1$ for $q \in Q$. Automaton $\mathcal{A}$ is *unambiguous* if every run on an accepted word is an accepting run. Specifically, any deterministic automaton is unambiguous. By means of a subset construction we can form a deterministic automaton $\mathcal{A}^d$ with the same language as $\mathcal{A}$, $L(\mathcal{A}^d) = L(\mathcal{A})$. In detail, $\mathcal{A}^d = \langle \Sigma, 2^Q, \{Q_0\}, \rho^d, F^d\rangle$ where $\rho^d(X, s) = \bigcup_{q \in X} \rho(q, s)$ and $F^d = \{X \subseteq Q : X \cap F \neq \emptyset\}$.

The *cross product* of two automata is an automaton $\mathcal{A} \times \mathcal{A}'$ that accepts the intersection of their languages, $L(\mathcal{A} \times \mathcal{A}') = L(\mathcal{A}) \cap L(\mathcal{A}')$; in detail, given two automata $\mathcal{A} = \langle \Sigma, Q, Q_0, \rho, F\rangle$ and $\mathcal{A}' = \langle \Sigma, Q', Q'_0, \rho', F'\rangle$ over the same alphabet Σ, the cross product $\mathcal{A} \times \mathcal{A}'$ is the automaton $\langle \Sigma, Q \times Q', Q_0 \times Q'_0, \rho \times \rho', F \times F'\rangle$ where $(\rho \times \rho')(\langle q, q'\rangle, s) = \rho(q, s) \times \rho'(q', s)$. Defined similarly, the *dual cross product* $\mathcal{A} \overline{\times} \mathcal{A}'$ of two automata accepts the union of their languages, $L(\mathcal{A} \overline{\times} \mathcal{A}') = L(\mathcal{A}) \cup L(\mathcal{A}')$; in detail, $\mathcal{A} \overline{\times} \mathcal{A}' := \langle \Sigma, Q \times Q', Q_0 \times Q'_0, \rho \times \rho', \{\langle q, q'\rangle : q \in F \text{ or } q' \in F'\}\rangle$. Automata $\mathcal{A} \times \mathcal{A}'$ and $\mathcal{A} \overline{\times} \mathcal{A}'$ are unambiguous if both $\mathcal{A}$ and $\mathcal{A}$ are.

Given a transition system $\mathcal{S} = \langle S, R, I_0\rangle$, we can form a corresponding automaton $\mathcal{A}_\mathcal{S}$ over alphabet S whose language consists precisely of all computation path of $\mathcal{S}$: we add a fresh state ι to S and make this the initial state in the automaton; we add a transition from the new initial state ι to each state in I_0; we label each transition $\langle s, s'\rangle$ (whether newly added or "old" from R) with the target state s'; finally we mark every location in S as accepting. In detail, $\mathcal{A}_\mathcal{S} := \langle S, S \cup \{\iota\}, \{\iota\}, \rho^R, S\rangle$ where $\rho^R(s, s') = \{s' \mid \langle s, s'\rangle \in R\}$ if $s \in S$ and $\rho^R(\iota, s') = \{s'\}$ if $s' \in I_0$ otherwise $\rho^R(\iota, s') = \emptyset$.

REFERENCES

[1] Rajeev Alur, Pavol Cerný, and Swarat Chaudhuri, 'Model checking on trees with path equivalences', in *TACAS*, eds., Orna Grumberg and Michael Huth, volume 4424 of *Lecture Notes in Computer Science*, pp. 664–678. Springer, (2007).

[2] Ananda Basu, Saddek Bensalem, Doron Peled, and Joseph Sifakis, 'Priority scheduling of distributed systems based on model checking', in *CAV '09: Proceedings of the 21st International Conference on Computer Aided Verification*, pp. 79–93, Berlin, Heidelberg, (2009). Springer-Verlag.

[3] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*, The MIT Press, Cambridge, Massachusetts, 1999.

[4] Catalin Dima, 'Revisiting satisfiability and model-checking for ctlk with synchrony and perfect recall', in *Proceedings of the 9th International Workshop on Computational Logic in Multi-Agent Systems (CLIMA IX)*, eds., Michael Fisher, Fariba Sadri, and Michael Thielscher, volume 5405 of *Lecture Notes in Computer Science*, pp. 117–131. Springer, (2008).

[5] R. Fagin, J. Y. Halpern, and M. Y. Vardi, 'What can machines know? On the properties of knowledge in distributed systems', *Journal of the ACM*, **39**(2), 328–376, (1992).

[6] Ronald Fagin, Joseph Y. Halpern, Moshe Y. Vardi, and Yoram Moses, *Reasoning about knowledge*, MIT Press, Cambridge, MA, USA, 1995.

[7] Tim French, Ron van der Meyden, and Mark Reynolds, 'Axioms for logics of knowledge and past time: Synchrony and unique initial states', in *Advances in Modal Logic*, eds., Renate A. Schmidt, Ian Pratt-Hartmann, Mark Reynolds, and Heinrich Wansing, pp. 53–72. King's College Publications, (2004).

[8] Peter Gammie and Ron van der Meyden, 'Mck: Model checking the logic of knowledge', in *CAV '04: Proceedings of the 6th International Conference on Computer Aided Verification*, eds., Rajeev Alur and Doron Peled, volume 3114 of *Lecture Notes in Computer Science*, pp. 479–483. Springer, (2004).

[9] Dimitar P. Guelev and Catalin Dima, 'Model-checking strategic ability and knowledge of the past of communicating coalitions', in *Proceedings of the 16th International Workshop on Declarative Agent Languages and Technologies (DALT08)*, eds., Matteo Baldoni, Tran Cao Son, M. Birna van Riemsdijk, and Michael Winikoff, volume 5397 of *Lecture Notes in Computer Science*, pp. 75–90. Springer, (2008).

[10] Magdalena Kacprzak, Wojciech Nabialek, Artur Niewiadomski, Wojciech Penczek, Agata Pólrola, Maciej Szreter, Bozena Wozna, and Andrzej Zbrzezny, 'Verics 2007 - a model checker for knowledge and real-time', *Fundam. Inform.*, **85**(1-4), 313–328, (2008).

[11] Alessio Lomuscio, Hongyang Qu, and Franco Raimondi, 'Mcmas: A model checker for the verification of multi-agent systems', in *CAV '09: Proceedings of the 21st International Conference on Computer Aided Verification*, pp. 682–688, Berlin, Heidelberg, (2009). Springer-Verlag.

[12] Amir Pnueli and Aleksandr Zaks, 'On the merits of temporal testers', in *25 Years of Model Checking*, eds., Orna Grumberg and Helmut Veith, volume 5000 of *Lecture Notes in Computer Science*, pp. 172–195. Springer, (2008).

[13] Nikolay V. Shilov and Natalya Olegovna Garanina, 'Model checking knowledge and fixpoints', in *FICS*, eds., Zoltán Ésik and Anna Ingólfsdóttir, volume NS-02-2 of *BRICS Notes Series*, pp. 25–39. University of Aarhus, (2002).

[14] Ron van der Meyden, 'Common knowledge and update in finite environments', *Inf. Comput.*, **140**(2), 115–157, (1998).

[15] Ron van der Meyden and Nikolay V. Shilov, 'Model checking knowledge and time in systems with perfect recall (extended abstract)', in *Proceedings of the 19th annual conference on Foundations of Software Technology and Theoretical Computer Science (FST TCS 1999)*, eds., C. Pandu Rangan, Venkatesh Raman, and Ramaswamy Ramanujam, volume 1738 of *Lecture Notes in Computer Science*, pp. 432–445. Springer, (1999).