Symbolic Model Checking Multi-Agent Systems against CTL*K Specifications

Jeremy Kong
Department of Computing
Imperial College London
jeremykong91@gmail.com

Alessio Lomuscio
Department of Computing
Imperial College London
a.lomuscio@imperial.ac.uk

ABSTRACT

We introduce a technique for model checking multi-agent systems against temporal-epistemic specifications expressed in the logic CTL*K. We present an algorithm for the verification of explicit models and use this to show that the problem is PSPACE-complete. We show that the technique is amenable to symbolic implementation via binary decision diagrams. We introduce MCMAS*, a toolkit based on the open-source model checker MCMAS which presently supports CTLK only, implementing the technique. We present the experimental results obtained and show its attractiveness compared to all other toolkits available.

Keywords

Model Checking; Epistemic Logic.

1. INTRODUCTION

Over the past 10 years considerable attention has been given to the problem of verifying multi-agent systems (MAS) against AI-inspired specifications, typically based on variants of temporal-epistemic logic. A key consideration for advances in this area has been the efficiency of the underlying methodology employed. Explicit techniques, where the state-space of the system is represented explicitly, were initially optimised by using SAT-based [29, 21] or BDD-based approaches [15, 31]; these have then been further enhanced by means of abstraction methodologies, including partial order reduction [20] and symmetry reduction [9].

While additional agent-based modalities have been added enriching the specification languages supported, including commitments [2] and various strategic operators [1], the underlying model of time has predominantly been the one offered by CTL, the branching-time computation tree logic [10]. This is in contrast with most work on reactive systems in which linear time logic, or LTL, is predominantly used [30]. While there are several reasons why MAS may require rich specifications including knowledge, commitments, etc., there seems to be no compelling reason for the underlying temporal model to be branching.

It is well known that LTL and CTL have different expressivity. While CTL permits expressions such as $AGEXreset$ representing "in all paths in the future it is possible to perform a reset in the next step", LTL admits statements such as $GFrequest$ expressing the fact that "a request is made infinitely often". MAS benefit from the power of LTL-based specifications just as distributed systems do. Indeed, new powerful logics introduced in AI such as LDL [16] have linear time features. There are clear advantages if both temporal variants are supported. Indeed, model checkers widely used for the analysis of reactive systems such as NuSMV [5] do so already.

The present apparent emphasis on CTL of some MAS techniques may be purely accidental. Indeed, the very first contributions on verification of MAS against temporal-epistemic specifications used LTL as the underlying model of time [28, 17]. In contrast, the first actual model checking toolkits for the verification of MAS supported, at least initially, CTL as the underlying temporal logic [24, 19]. In particular, MCMAS [22], released as open-source from 2004 onwards still supports CTLK only.

In this paper we introduce a technique for the verification of MAS against specifications in CTL*K. CTL*K is the combination of CTL* [11] with the epistemic logic S5 [14]. As is known, CTL* strictly subsumes both CTL and LTL; hence, by providing a technique for CTL* we also support LTL. The algorithm we report is based on the recursive descent based approach for CTL* and extends it by adding support to knowledge modalities. We show that the algorithm runs in polynomial space and has runtime exponential only in the size of the formula. We implemented the technique as an extension of the MCMAS model checker, thereby obtaining a toolkit that supports the whole of CTL*K. The experimental results we report indicate that the present checker suffers no penalty against the current MCMAS release and is the most efficient tool for verifying CTL*K specifications under observational semantics.

Related Work. The algorithm we introduce here is based on the method for reducing model checking of full-branching time logics to model checking of the respective linear-time logics introduced in [12], combined with the use of LTL model checking techniques such as that presented in [7, 33]. The method presented combines and extends these techniques by adding support for individual knowledge, distributed knowledge and common knowledge [14].

Our implementation is based on the open-source model checker MCMAS [22]. While the implementation for CTL*K uses a different labelling algorithm, we show that this does not impact the performance on the CTL fragment which is already supported by the checker. The implementation we

Appears in: *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017), S. Das, E. Durfee, K. Larson, M. Winikoff (eds.), May 8–12, 2017, São Paulo, Brazil.*

here report significantly extends the specification language supported.

MCK [25] supports the full logic CTL*K, although only the LTLK fragment is discussed in [15]. Differently from MCMAS, MCK supports a variety of semantics, including perfect recall (for one agent only) and clock semantics. Under observational semantics and all the scenarios we considered, the performance of the toolkit introduced in this paper was superior to that of MCK. The reasons for the difference in performance may be due to the different labelling algorithm employed. In addition the tools have different functionalities. For example we have also implemented support for counterexample generation relying on the techniques introduced in [8], which does not appear to be supported by MCK v1.1.0 in this setting.

In other work [27] introduced an approach to the verification of MAS against LTL and knowledge. However, this work is based on bounded model checking and supports only the existential fragment of the logic. Given this, it is not comparable to the present approach.

2. BACKGROUND

In this section we introduce the formalism of interpreted systems, which we use to model MAS. We then present the syntax and semantics of CTL*K.

2.1 Interpreted Systems

Interpreted systems, introduced in [14], are a well-known formalism for MAS. They provide a convenient means for reasoning about time and knowledge. Here we follow the presentation given in [22], where agents have locally defined transition relations, and the global transition relation is given by the composition of these. We assume a set of agents $A = \{1, \dots, n\}$ and an environment e.

Definition 1 (Interpreted Systems). *An interpreted system is a tuple $IS = \langle \{L_i, Act_i, P_i, \tau_i\}_{i \in A \cup \{e\}}, I, h\rangle$, where:*

- L_i *is a finite set of possible* local states *of agent* i.
- Act_i *is a finite set of possible* actions *of agent* i.
- $P_i : L_i \to 2^{Act_i} \setminus \{\emptyset\}$ *is a* local protocol *for agent $i \in A$, specifying which actions agent i can take from each local state.*
- $\tau_i : L_i \times Act_1 \times \ldots \times Act_n \times Act_e \to L_i$ *is a local* transition function*, returning the new local state of agent $i \in A$ after the agents and environment act in a given way. We assume that the agents' actions are be consistent with their protocols; i.e., if $l'_i = \tau_i(l_i, a_1, \dots, a_n, a_e)$, then for each $i \in \{1, \dots, n\}$, we have that $a_i \in P_i(l_i)$.*
- $I \subseteq L_1 \times \ldots \times L_n \times L_e$ *is the set of* initial global states.
- $h : AP \to 2^{L_1 \times \ldots \times L_n \times L_e}$*, where AP is a set of atomic propositions, is a* valuation function *identifying the global states in which an atomic proposition is true.*

The environment e is similarly associated with a local transition function $\tau_e : L_e \times Act_1 \times \ldots \times Act_n \times Act_e \to L_e$ and local protocol $P_e : L_e \to 2^{Act_e} \setminus \{\emptyset\}$. We often consider the set of *reachable global states* $G \subseteq L_1 \times \ldots \times L_n \times L_e$, reachable from I by applying the composition of the local transition functions. We also introduce the notion of an evolution *path* for an interpreted system IS, consisting of an infinite sequence of global states $g_0, g_1, \ldots$ such that for every i, $g_i \in G$, and if $i > 0$ then there exists actions $a_1, \dots, a_n, a_e$ such that for every local state l'_j in g_i, $l'_j = \tau_j(l_j, a_1, \dots, a_n, a_e)$ for each possible agent $j \in A \cup \{e\}$ (where l_j is the local state of agent j in state g_{i-1}). Furthermore, we denote with $\pi(i)$ the *i-th state* of a path $\pi = g_0, g_1, \ldots$; we denote with π^i the *i-th suffix* of a path π; i.e., $\pi^i = g_i, g_{i+1}, \ldots$. Observe that because paths are infinite sequences, a suffix of a path is also a path.

For $i \in A$ we also define a projection operator $l_i : G \to L_i$ denoting the local state of agent i from each global state.

Intuitively, interpreted systems describe a finite-state transition system consisting of multiple agents, which may synchronise with one another using joint actions [22]. We refer to [14] for more details.

2.2 The Logic CTL*K

We first introduce the syntax of CTL*K. This is an extension of full branching time logic with epistemic modalities, and allows us to quantify arbitrarily over paths and make assertions about agents' knowledge. For example, by using CTL*K we can express the property "on every path in which p is true infinitely often, it is always eventually possible for agent 1 to know that q is permanently true" (written as $A((GFp) \to G(E(FK_1(Gq)))))$. Formally, the syntax of CTL*K is defined as follows, where ψ refers to a *path formula* which may or may not hold on a given path, and ϕ refers to a *state formula* which may or may not hold at a given state.

$$\langle\psi\rangle ::= \phi \mid \neg\psi \mid \psi \wedge \psi \mid X\psi \mid \psi U \psi$$

$$\langle\phi\rangle ::= \top \mid p \mid \neg\phi \mid \phi \wedge \phi \mid E\psi \mid K_i\phi \mid E_\Gamma\phi \mid D_\Gamma\phi \mid C_\Gamma\phi$$

where p is an atomic proposition, $i \in A \cup \{e\}$ is an agent, and $\Gamma \subseteq A \cup \{e\}$ is a set of agents. We allow the standard abbreviations from propositional logic, for both state and path formulae ($\bot = \neg\top$, $\phi_1 \vee \phi_2 = \neg(\neg\phi_1 \wedge \neg\phi_2)$, $\phi_1 \to \phi_2 = \neg\phi_1 \vee \phi_2$ and $\phi_1 \leftrightarrow \phi_2 = (\phi_1 \to \phi_2) \wedge (\phi_2 \to \phi_1)$), for path quantification ($A\psi = \neg E \neg\psi$), and for the temporal modalities ($F\phi = (\top U \phi)$, $G\phi = \neg F \neg\phi$).

We now inductively define the semantics of CTL*K formulae. We first define this for path formulae. In what follows we assume that IS is an interpreted system, and π is a path in IS. We define satisfaction of a path formula as follows:

- $IS, \pi \models \phi$ iff $IS, \pi(0) \models \phi$, where ϕ is a state formula.
- $IS, \pi \models \neg\psi$ iff it is not the case that $IS, \pi \models \psi$.
- $IS, \pi \models \psi_1 \wedge \psi_2$ iff both $IS, \pi \models \psi_1$ and $IS, \pi \models \psi_2$.
- $IS, \pi \models X\psi$ iff $IS, \pi^1 \models \psi$.
- $IS, \pi \models \psi_1 U \psi_2$ iff $\exists j \geq 0$ such that $IS, \pi^j \models \psi_2$ and for $0 \leq k < j$ we have $IS, \pi^k \models \psi_1$.

Intuitively, $X\psi$ holds if ψ is true in the next state; $\psi_1 U \psi_2$ holds if ψ_1 is true until ψ_2 becomes true.

Observe that the first condition states that a state formula ϕ is satisfied on a path if it is satisfied in the first state of the path. Satisfaction for state formulae ϕ on states $g \in G$ is given as follows.

- $IS, g \models \top$.

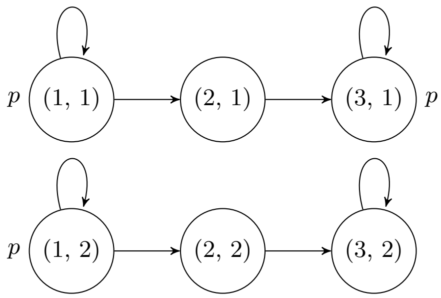

Figure 1: A simple interpreted system IS. The initial states are $(1,1)$ and $(1,2)$; $h(p)$ is true at precisely the states labelled with p.

- $IS, g \models p$ iff $g \in h(p)$.
- $IS, g \models \neg\phi$ iff it is not the case that $IS, g \models \phi$.
- $IS, g \models \phi_1 \wedge \phi_2$ iff both $IS, g \models \phi_1$ and $IS, g \models \phi_2$.
- $IS, g \models E\psi$ iff there exists a path π with $\pi(0) = g$ such that $IS, \pi \models \psi$.
- $IS, g \models K_i\phi$ iff for every state $g' \in G$ if $l_i(g) = l_i(g')$, then $IS, g \models \phi$.
- $IS, g \models E_\Gamma\phi$ iff for every state $g' \in G$ if $l_i(g) = l_i(g')$ for *some* agent $i \in \Gamma$, then $IS, g \models \phi$.
- $IS, g \models D_\Gamma\phi$ iff for every state $g' \in G$ if $l_i(g) = l_i(g')$ for *all* agents $i \in \Gamma$, then $IS, g \models \phi$.
- $IS, \pi \models C_\Gamma\phi$ iff for every state $g \in G$ if $g_1, \ldots$ are states such that $l_i(g) = l_i(g_1)$, $l_{k_1}(g_1) = l_{k_1}(g_2)$, $\ldots$, $l_j(g_n) = l_j(\pi(0))$ for some agents $i, j, k_1, \ldots \in \Gamma$ and positive integer n, then $IS, g \models \phi$.

Intuitively, we have $E\psi$ if there exists a path on which ψ is true. We use *observational semantics* for the epistemic modalities; hence the agents' knowledge is a function of their instantaneous private local state only: an agent knows ϕ if ϕ holds in every possible state of the world consistent with its local state. The group modalities E, D and C intuitively correspond to ϕ being known by everyone in a group, ϕ being distributed knowledge in the group, and ϕ being common knowledge in the group, respectively. We refer to [14] for more details.

We now introduce a small example to illustrate some subtleties of the semantics of CTL*K. Consider an interpreted system IS with global states $G = \{(x, y) \mid x \in \{1, 2, 3\}, y \in \{1, 2\}\}$ where for agent 1, $l_1(x, y) = x$. The global transition relation of IS is illustrated in Figure 1; we assume $I = \{(1,1), (1,2)\}$ and $h(p) = \{(1,1), (1,2), (3,1)\}$. Consider the CTL*K formula $E(GF(K_1\neg p))$. First observe that $(2,1)$ and $(2,2)$ are the only states satisfying $K_1\neg p$. It follows that $IS, s \not\models E(GF(K_1\neg p))$ for any $s \in G$, since any path going through either of $(2,1)$ or $(2,2)$ subsequently transitions to and remains in $(3,1)$ or $(3,2)$ respectively. So the formula is false on IS as it is satisfied in none of its initial states.

Notice that the formula considered is not directly expressible in LTLK, and is also semantically different from the CTLK formula $EG(EF(K_1\neg p))$ in CTLK. Indeed, observe that $IS, s \models EF(K_1\neg p)$, where $s \in \{(1,1), (1,2), (2,1), (2,2)\}$. It follows that $IS, (1,1) \models EG(EF(K_1\neg p))$ and also $IS, (1,2) \models EG(EF(K_1\neg p))$. Therefore the specification $EG(EF(K_1\neg p))$ is true in IS, because it is satisfied at all the initial states.

The specification language CTL*K is very expressive. It includes both CTLK and LTLK: CTLK is equivalent to the subset of CTL*K where path formulae are restricted to $X\phi$ and $\phi U \phi$, where ϕ is a state formula; LTLK is equivalent to the subset of CTL*K composed of $A\psi$ formulae. CTL*K is more expressive than the union between LTLK and CTLK as it can freely mix path and and state formulae. For example, it can express properties such as "there is a run of the system in which if at some point agent 1 knows that p, then at some later point in that run p becomes commonly known by the agents in Γ", expressed by the CTL*K formula $EF(K_ip \rightarrow FC_\Gamma p)$. This is an example of a specification that may be useful, for example, when reasoning about cooperation whereby knowledge might evolve from private to common in a computation path of interest. Note that our previous example $E(GF(K_1\neg p))$ is also not readily expressible in CTLK or LTLK.

In terms of practical usage, specifying properties in CTL*K offers the ability to freely mix path and state formulae thereby addressing many of the limitations of using CTLK or LTLK alone. These include the inability to express some fairness properties directly at specification level and to perform more complex quantification over paths, respectively. For example, consider a variant of the train, gate and controller scenario from [18]. We have N trains travelling on looping tracks, all of which go through a tunnel that can only accommodate one train. A controller gives signals to the trains indicating whether they may proceed or not. We assume that trains at any state might develop a fault by performing a *break* action causing them to skip their current turn (and remain in the current state). We keep track of whether a train i has performed the *break* action at the previous time step by using the proposition $broken_i$; we extend the controller C's responsibilities with the ability to evict trains from the tunnel. Similar scenarios are often used in safety-analysis, both in epistemic and temporal settings [3, 13], e.g., advanced avionics and autonomous vehicles. The logic CTL*K allows us to investigate properties, such as "on every path, it eventually becomes the case that when a train is broken, the controller knows about this and can evict the train on the next turn", expressed by:

$$\bigwedge_{i=1}^{N} AFG((\neg broken_i) \vee (K_C(broken_i) \wedge EX\neg tunnel_i))$$

This is a property of interest if it is desired that the system is fault-tolerant and can perform fault diagnosis correctly. This property is not expressible in either CTL nor LTL, as it uses both universal and existential path quantifiers. Further CTL*K specifications are discussed in Section 5.

Given an interpreted system IS, a state g and a CTL*K formula ϕ, the model checking problem involves determining whether $IS, g \models \phi$.

3. MODEL CHECKING CTL*K

We now present an algorithm which, given an arbitrary CTL*K formula ϕ and an interpreted system IS, computes

$[\phi]_{IS}$, the set of states of IS in which ϕ holds. We prove the correctness of the algorithm and show that it runs in polynomial space and with time exponential only in the size of the formula.

To do so, we use two mutually recursive functions LABEL and TRANS. The function LABEL(ϕ, IS) ultimately outputs the set of states in which the CTL*K state formula ϕ holds; TRANS(ψ, IS) outputs an LTL formula which holds precisely in the same set of states in which the CTL*K path formula ψ holds. The translation introduces a fresh atomic proposition p for each path formula which includes an epistemic modality or E as a principal connective; this is sufficient because both knowledge and path existence with respect to a formula are only dependent on the current state.

In the algorithm we assume the known procedures SAT_K, SAT_E, SAT_D and SAT_C for computing the set of states satisfying the corresponding epistemic modalities; see, e.g., [31]. We also use the procedure EXIST-PATH returning the states in IS from which there exists a path on which the LTL formula ϕ holds [7]. Finally the algorithm uses the procedure NEW-PROPOSITION returning a fresh atomic proposition when called. Finally, to check whether a CTL*K formula ϕ is satisfied at a given state g in a model IS, we check whether $g \in$ LABEL(ϕ, IS).

Algorithm 1 Labelling Algorithm

INPUT: CTL*K state formula ϕ, interpreted system IS
OUTPUT: $[\phi]_{IS}$

function LABEL(ϕ, IS)
  if ϕ is an atomic proposition p **then**
    return $h(p)$
  else if $\phi = \top$ **then**
    return G
  else if $\phi = \neg\phi_1$ **then**
    return $G \setminus$ LABEL(ϕ_1, IS)
  else if $\phi = \phi_1 \wedge \phi_2$ **then**
    return LABEL$(\phi_1, IS) \cap$ LABEL(ϕ_2, IS)
  else if $\phi = K_i\phi_1$ **then**
    return $SAT_K($LABEL$(\phi_1, IS), i)$
  else if $\phi = E_\Gamma\phi_1$ **then**
    return $SAT_E($LABEL$(\phi_1, IS), \Gamma)$
  else if $\phi = D_\Gamma\phi_1$ **then**
    return $SAT_D($LABEL$(\phi_1, IS), \Gamma)$
  else if $\phi = C_\Gamma\phi_1$ **then**
    return $SAT_C($LABEL$(\phi_1, IS), \Gamma)$
  else ▷ $\phi = E\psi$ for some path formula ψ
    return EXIST-PATH(TRANS$(\psi, IS), IS)$
  end if
end function

Observe that all functions naturally lend themselves to a symbolic implementation, as set operations, the SAT procedures for the epistemic modalities [31], and EXIST-PATH [7] can all be implemented symbolically. We now prove the correctness of the algorithm by showing that LABEL(ϕ, IS) returns precisely the set of states in which ϕ is true in IS.

THEOREM 1. *For any $g \in G$, $g \in$ LABEL(ϕ, IS) if and only if $g \in [\phi]_{IS}$.*

PROOF. By structural induction on ϕ.

- The propositional cases follow directly from the CTL*K semantics, and the inductive hypothesis (for $\neg$ and $\wedge$).

Algorithm 2 Translation Algorithm

INPUT: CTL* path formula ψ, interpreted system IS
OUTPUT: LTL formula ψ' where for any path π, $IS, \pi \models \psi \leftrightarrow IS, \pi \models \psi'$

function TRANS(ψ, IS)
  if ψ is an atomic proposition p **then**
    return p
  else if $\psi = \top$ **then**
    return $\top$
  else if $\psi = \neg\psi_1$ **then**
    return $\neg($TRANS$(\psi_1, IS))$
  else if $\psi = \psi_1 \wedge \psi_2$ **then**
    return TRANS$(\psi_1, IS) \wedge$ TRANS(ψ_2, IS)
  else if $\psi = X\psi_1$ **then**
    return $X($TRANS$(\psi_1, IS))$
  else if $\psi = \psi_1 U \psi_2$ **then**
    return TRANS$(\psi_1, IS) U$ TRANS(ψ_2, IS)
  else ▷ ψ is an epistemic modality or $E\psi'$
    $p' =$ NEW-PROPOSITION()
    $h(p') =$ LABEL(ψ, IS) ▷ ψ is a state formula
    return p'
  end if
end function

- For the epistemic modalities, by the inductive hypothesis, LABEL(ϕ_1, IS) returns $[\phi_1]_{IS}$. Furthermore, the SAT procedures for the epistemic modalities follow those presented in [31], which were shown to be correct.

- For $E\psi$, we rely on the correctness of EXIST-PATH from [7]. We show that TRANS preserves the semantics of ψ, i.e., for any path π, $IS, \pi \models \psi$ if and only if $IS, \pi \models$ TRANS(ψ, IS). Observe that the semantics of E as well as the epistemic modalities are dependent on the current state only, as a suitable atomic proposition p'. By inductive hypothesis the set of states in which p' holds for a given subformula ψ' is computed correctly by LABEL(ψ', IS). No other changes are made to the path formula ψ.

This concludes the analysis of all the inductive cases. □

We now explore the complexity of the verification problem. We assume an interpreted system to be given explicitly if all reachable states and epistemic and temporal relations are provided directly and need not be computed.

THEOREM 2. *Verifying explicit interpreted systems against CTL*K specifications is PSPACE-complete.*

PROOF. Hardness follows from CTL* model checking, which is PSPACE-hard [6].

To show membership, consider Algorithm 1. The total number of calls to LABEL(ϕ, IS) is polynomial in the size of the formula (and hence the size of the input), since we only call LABEL up to twice for each instance of an epistemic modality, path quantifier, $\neg$ or $\wedge$. Calls to TRANS(ψ, IS) do not introduce any of the above constructs. We can thus safely store all intermediate labellings computed by the algorithm. Since each labelling is bounded by $O(|IS|)$, we obtain $O(|\phi||IS|)$ space overall.

It remains to show that LABEL, the epistemic procedures, EXIST-PATH and TRANS run in polynomial space.

- LABEL – We make polynomially many calls to the other procedures, which are in polynomial space, and perform polynomially many set operations. It follows that the procedure runs in polynomial space.

- Epistemic procedures – By assumption all states and relations, including all epistemic accessibility relations, are given in the input. Following the algorithms in [31], we can iterate through the reachable states and maintain an accumulator for the result. We may need to maintain another temporary set for SAT_C when computing fix-points. In all cases, this is bounded by size $2|IS|$ and is thus polynomial.

- EXIST-PATH – By using Büchi automata we can determine the set of states from which there exists a path satisfying a given LTL formula in polynomial space; see, e.g., [33] for more details.

- TRANS – The number of epistemic modalities or path quantifiers present is polynomial, and thus we only introduce polynomially many additional atomic propositions. We also only call LABEL, which is polynomial space, polynomially many times.

We conclude that model checking interpreted systems against CTL*K specifications is PSPACE-complete. $\square$

Note that model checking explicit models against CTL* specifications is also PSPACE-complete [6]; therefore adding epistemic modalities (under observational semantics) does not increase the worst case verification complexity.

We now show that LABEL and TRANS have a runtime which is exponential only in the formula size. We first introduce Lemma 1 to show that we can recursively check subformulae of our initial formula whilst preserving this complexity.

LEMMA 1. *Suppose that for $i \in \{0, \dots, k\}$ we have $f(y_i, n) = O(p(n)2^{y_i})$ with $0 \le y_i$, where $p(n)$ is a polynomial which is positive for positive n. Also suppose that $\sum_{i=0}^{k} y_i \le x$ and that $f(x, n) = \left(\sum_{i=0}^{k} f(y_i, n)\right) + O(p(n)2^x)$. Then $f(x, n) = O(p(n)2^x)$.*

PROOF. From the definition of big-O notation we have that for each i, $f(y_i, n) \le c_i n 2^{y_i}$ for some positive c_i. Then, observe that

$$\begin{aligned} f(x,n) &= \left(\sum_{i=0}^{k} f(y_i,n)\right) + O(p(n)2^x) \\ &\le \left(\sum_{i=0}^{k} c_i p(n) 2^{y_i}\right) + O(p(n)2^x) \\ &\le \left(Cp(n)\sum_{i=0}^{k} 2^{y_i}\right) + dp(n)2^x \\ &\le (Cp(n))\,2^x + dp(n)2^x \\ &= (C+d)\,p(n)2^x \end{aligned}$$

for some positive constants C (e.g., the maximum of the c_i) and d. Thus, we obtain $f(x, n) = O(p(n)2^x)$. $\square$

We can now state the main complexity result of this section.

THEOREM 3. *Verifying explicitly specified interpreted systems against CTL*K specifications by means of Algorithm 1 is in $O(p(|IS|)2^{|\phi|})$ time for some polynomial $p(|IS|)$.*

PROOF. We apply Lemma 1, where $f(|\phi|, |IS|)$ is the time taken to compute LABEL(ϕ, IS). We consider the various cases:

1. Propositional cases (lines 2–9). These require up to 2 recursive calls to LABEL(ϕ, IS), as well as possibly 1 set operation. The set operation can clearly be computed in $O(|IS|)$ time by iterating through the sets of states involved, each of which is bounded by size $O(|IS|)$.

2. Epistemic modalities (lines 10–17). Each SAT procedure involves determining the states in which ϕ_1 does not hold (thus there is one subproblem), looking for states related to these, and then complementing the result. The second and third steps can be bounded by $O(p(|IS|))$ time.

3. $E\psi$ (lines 18–19). TRANS calls LABEL, but the sum of the sizes of the subformulas it is called on is clearly less than the size of the original formula. The procedure returns an LTL formula with size bounded by $|\phi|$. We have from [33] that EXIST-PATH can be performed in $O(|IS|2^{|\phi|})$ time.

We observe that in all cases, we have a finite number of subproblems with total size bounded by the original formula, and additional computation that is $O(p(|IS|)2^{|\phi|})$. Thus Lemma 1 applies and the overall time complexity is $O(p(|IS|)2^{|\phi|})$. $\square$

Having derived the model checking algorithm and studied its complexity, we now proceed to implement it.

4. IMPLEMENTATION AND EXPERIMENTAL RESULTS

We implemented the algorithms of Section 3 on top of version 1.2.2 of MCMAS [22], a symbolic model checker that uses BDD-based technology to perform verification. In the following the resulting implementation is referred to as MCMAS*.

For usability reasons, MCMAS* implements all the abbreviations from Section 2.2. These are used to convert input formulae and use only the minimal set of operators. MCMAS* uses Algorithm 1 on the converted input formula (say ϕ) to determine the states in which ϕ holds. MCMAS* then checks the validity of $InitStates \rightarrow \phi$, returning true if ϕ holds in every initial state of the model.

For Algorithm 1, the sets of states used throughout LABEL are represented using BDDs. We implemented Algorithm 2 by using recursive descent; the call NEW-PROPOSITION is readily supported by CUDD, the package used by MCMAS to perform operations on BDDs.

To realise MCMAS* we also implemented EXIST-PATH, as this is not provided within MCMAS. We did so by using the symbolic tableau construction in [7]. This involved encoding symbolically the tableau and the fairness constraints, and then performing model checking on the formula $EG\top$ with the relevant fairness constraints. We remark that MCMAS* performs the construction differently from the approach used in [7] in that this construction is done internally within

MCMAS*. In contrast, the approach in [7] reported the reprocessing of the user's input file, generating a separate SMV program for each specification. This enabled us to improve the performance of our implementation, since model checking CTL*K specifications would normally require multiple calls to EXIST-PATH for each specification to be checked. Had we followed the approach in [7], we would have generated separate ISPL files for each LTL formula, which would have required unnecessarily rebuilding the model at each recursive call, thereby resulting in a less efficient implementation. Observe that this issue does not arise in the context of LTL formulae as in that case, only one such translation step would be needed. Moreover, by maintaining the same model throughout the verification step, we also benefit from the CUDD cache and dynamic variable reordering features over multiple calls whilst only paying their cost once, thereby further boosting the performance of the checker.

A further issue we encountered is that the tableau algorithm maintains the consistency between the model and the tableau variables by using the same instance of the BDD variables; this is generally not possible in MCMAS, because its `Evaluation` variables need not correspond to single BDD variables in the model. Furthermore, by executing Algorithm 2, we may also generate fresh atomic propositions that are not part of the model itself. To reconcile these aspects, we explicitly augmented the tableau with a set of consistency rules to enforce this correspondence. Formally, before computing the verification step, for a given proposition p, we added a Boolean equality constraint between the tableau variable for the elementary formula p and the states of the model in $h(p)$.

In addition to the model checking algorithms, we also implemented the generation of counterexamples and witnesses for several classes of CTL*K formulas. The build and resulting code are available from [26]. MCMAS* inherits the support of fairness constraints from MCMAS.

Experimental Setup. To evaluate the proposed algorithms, we ran experiments on virtual machines with two 2.70GHz CPUs and 16 GB of RAM, running Ubuntu v15.10. We evaluated the performance of MCMAS* in two ways:

1. For specifications readily expressible in CTLK (which is subsumed by CTL*K), we compared the performance of the proposed CTL*K algorithm against the CTLK algorithm already implemented in MCMAS.

2. For specifications *not* expressible in CTLK, we compared the performance of MCMAS* against MCK version 1.1 [15], a state-of-the-art model checker supporting verification of CTL*K properties using symbolic model checking.

We verified specifications concerning the well-known *dining cryptographers* scenario [4], and considered how the algorithms scaled to different numbers N of cryptographers.

Comparison with MCMAS (CTLK fragment). We compared MCMAS* with MCMAS by verifying two equivalent safety properties: ϕ_0 (CTLK) and ϕ_0' (CTL*K):

$$\phi_0 = AG\left(\bigwedge_{i=0}^{N-1} \bigwedge_{j\in\{0,\ldots,N-1\},j\neq i} \neg K_i paid_j\right)$$

$$\phi_0' = A\left(G\left(\bigwedge_{i=0}^{N-1} \bigwedge_{j\in\{0,\ldots,N-1\},j\neq i} \neg K_i paid_j\right)\right)$$

	time (s)			BDD memory	
N	reach	CTLK	CTL*K	CTLK	CTL*K
3	0.022	0.001	0.003	8.81	9.07
4	0.057	0.002	0.005	8.97	9.34
5	0.090	0.002	0.008	9.17	9.77
6	0.172	0.004	0.013	9.55	10.42
7	0.223	0.006	0.020	10.03	11.24
8	0.287	0.009	0.030	10.31	11.95
9	0.493	0.012	0.041	10.80	12.86
10	0.662	0.015	0.058	11.48	14.14
12	2.001	0.037	0.135	15.56	21.34
15	3.544	0.183	0.511	27.13	38.74
20	22.270	0.468	1.280	49.12	45.49
25	38.071	0.847	3.036	46.75	57.72
30	60.966	SO	9.180	SO	59.25
40	151.075	SO	34.553	SO	91.88
50	633.898	SO	120.298	SO	179.47

Table 1: Verification results for MCMAS's CTLK and MCMAS*'s CTL*K algorithms on the Dining Cryptographers protocol, for specifications ϕ_0 and ϕ_0' respectively. All results for memory usage are presented in MB. We experienced stack overflows with MCMAS, owing to the large size of the formulae for $N \geq 30$.

These properties hold for any value of $N \geq 3$; they express that no cryptographer should ever know that any other cryptographer has paid. The results are presented in Table 1.

We observe that the CTLK algorithm performs better than the CTL*K algorithm, though both algorithms require time that is on the same order of magnitude. This is to be expected as the model checking problem for CTLK is P-complete [23] while the corresponding problem for for CTL*K is PSPACE-complete (from Theorem 2). Our implementation of LTL model checking requires us to construct the symbolic tableau and compose this with the model; as previously discussed, this involves CTL model checking of $EG\top$ on a larger model with an additional fairness constraint. Nonetheless, we observe that the algorithm still scales reasonably well (the protocol for $N = 50$ has approximately 1.94×10^{32} reachable states). This validates the findings in [7] where a similar performance differential was reported on the SMV model checker when comparing CTL specifications and the tableau algorithm for LTL. Also note that we do need to compute first the sets of states in which $K_i paid_j$ holds for each pair $i \neq j$, though this is also done for CTLK. Indeed, the SAT_K algorithm used to determine this is the same for both CTLK and CTL*K.

Further observe that in any case, the total runtime is consistently dominated by the computation of the reachable state space; so any difference in the labelling times is not significant in practical usage. In addition to the one here presented, we have conducted further experiments, leading us to conclude that the increase in expressivity has been obtained without a significance performance penalty.

Comparison with MCK (Full CTL*K). To compare the performance of MCMAS* to that of MCK, we used both model checkers to verify the liveness property ϕ_1 of the dining cryptographers scenario above. Throughout our experiments, we restricted ourselves to observational semantics only. Note that, while MCK offers some support for al-

	time (s)	
N	MCMAS*	MCK
3	0.033	4.163
4	0.072	142.917
5	0.110	240.713
6	0.201	SEG FAULT
8	0.338	SEG FAULT
10	0.735	SEG FAULT
15	4.131	SEG FAULT
20	23.511	SEG FAULT
30	61.826	SEG FAULT
40	159.949	SEG FAULT
50	551.877	SEG FAULT

Table 2: Verification results for MCMAS* and MCK on the Dining Cryptographers protocol for the property ϕ_1. We encountered segmentation faults when using MCK to verify ϕ_1 with $N \geq 6$ cryptographers.

ternative semantics, neither tool supports clock semantics nor perfect recall for the specification we checked. Furthermore, we forced both tools to use the sifting algorithm for dynamic variable reordering. This avoids uneven overheads from the underlying BDD packages, even if both tools use the CUDD package as BDD handler. Since MCK does not support counterexample generation in the following we focus on verification only.

Finally, we ensured that the explicit models encoded for each tool were bisimilar. The sources are accessible from [26].

$$\phi_1 = A\left(G\left(\neg paid_0 \rightarrow F\left(\psi_{none} \vee \psi_{someone}\right)\right)\right)$$
$$\psi_{none} = K_0\left(\bigwedge_{i=0}^{N-1} \neg paid_i\right)$$
$$\psi_{someone} = K_0\left(\bigvee_{i=1}^{N-1} paid_i\right) \wedge \bigwedge_{i=1}^{N-1} \neg K_0\left(paid_i\right)$$

In the specifications above ϕ_1 holds for any value of $N \geq 3$; it expresses that if cryptographer 0 has not paid, then he eventually either knows that no one has paid or knows that some other cryptographer has paid, but does not know whom. The results are presented in Table 2.

The tests reported suggest that MCMAS* appears able to cope better than MCK on larger state spaces; this is consistent with previous results concerning the scalability of MCMAS and MCK for verifying CTLK properties [22]. Further tests on different specifications (sources at [26]) suggest the same conclusion. We observe that both MCMAS* and MCK use the symbolic tableau construction to verify CTL* properties, which involves a reduction to potentially multiple CTL model checking problems with fairness constraints.

We also considered a variant of the dining cryptographers scenario, which allowed us to express further CTL*K specifications as well as to evaluate MCMAS*'s performance on those. The modified scenario consists of an *iterated* version of the protocol whereby the cryptographers play infinitely many rounds of the protocol. In this variant we further allowed the possibility of intruders being part of the group; specifically, the paying cryptographer may nondeterministically violate the protocol when making the announcement of the parity of the coins. We defined the proposition $obey_i$ to represent that cryptographer i is currently following the protocol, i.e. he declares "same" or "different" according to what the protocol rules; $\neg obey_i$ represents the fact that the cryptographer i is stating the opposite of what prescribed by the protocol. Non-paying cryptographers are assumed to follow the protocol throughout their executions. Under these circumstances, we can verify the CTL$^*K$ property ϕ_2:

$$\phi_2 = (paid_0) \rightarrow AG(\psi_{none} \rightarrow E\left(\psi_{arbitrary} \wedge G\left(\psi_{none}\right)\right))$$
$$\psi_{arbitrary} = GF obey_0 \wedge GF \neg obey_0$$
$$\psi_{none} = \bigwedge_{i=1}^{N-1} \neg K_i paid_0$$

ϕ_2 states that "If no one else knows the paying cryptographer has paid, then it is always possible for her to arbitrarily follow or violate the protocol (when announcing "same" or "different"), without any other cryptographer being able to deduce she is the payer". In the specification, we exploited the symmetry of the protocol and used cryptographer 0 as the payer.

We found that, assuming $N \geq 3$, the specification ϕ_2 holds. This is because the other cryptographers cannot share any information and, under observational semantics, they do not remember past announcements. Observe that ϕ_2 is not expressible in CTLK nor in LTLK. This may indicate a weakness of the protocol as it is somewhat non-tolerant to an element of disruption in the protocol. We can use this fact to derive variations of the protocol that are resistant to tampering. The results of the tests against ϕ_2 are presented in Table 3.

Notice that it is actually inevitable that *distributed knowledge* between cryptographers $N-1$ and 1 of the identity of the intruder (i.e., that cryptographer 0 must have paid) is realised since the remaining cryptographers collectively know the value of both coins that cryptographer 0 has seen. It can be checked that upon her first lie (which must happen at some point, since we require $GF\neg obey_0$) distributed knowledge of this fact is created. We do not report the verification results for this specification as distributed knowledge is not currently supported by MCK.

By comparing the algorithms' performance over ϕ_1 and ϕ_2, we observe that for the same value of N, both MCMAS* and MCK require noticeably more time to verify ϕ_2 than ϕ_1. We observe that the iterated protocol has a significantly larger state space, owing to the ability of the payer to disobey the protocol, as well as the possibility of having many more legitimate intermediate states during iterations. For example, for $N = 50$ the reachable state space for the iterated model has 3.25×10^{62} states as compared to 1.94×10^{32} of the original model. Since our algorithms scale linearly with increases in model size, verifying similar properties over larger models is expected to take longer. Furthermore, building the model (which involves iterating through its full depth) is also significantly slower for the modified version of the protocol. We were unable to obtain a comparable step-by-step overview of the time spent in each step of verification for MCK, and thus we based all reported results on the total run-time. This would include not just labelling the states in which the properties hold, but also parsing and building symbolic representations of the models being specified.

The results reported suggest that MCMAS* appears bet-

	time (s)	
N	MCMAS*	MCK
3	0.116	172.020
4	0.207	Timeout
5	0.370	Seg Fault
6	0.613	Seg Fault
8	2.111	Seg Fault
10	2.692	Seg Fault
15	32.371	Seg Fault
20	41.339	Seg Fault
30	416.815	Seg Fault
40	631.159	Seg Fault
50	1042.779	Seg Fault
60	Timeout	Seg Fault

Table 3: Verification results for MCMAS* and MCK on the iterative Dining Cryptographers protocol for the property ϕ_2. We encountered segmentation faults when using MCK to verify ϕ_2 with $N \geq 5$ cryptographers. We ran our tests with a timeout of 6 hours.

ter able to scale to larger state spaces than MCK on formulae that exploit the additional expressivity provided by CTL*K. This may be owing to different BDD representations or algorithms used, or other undocumented implementation details.

5. CONCLUSIONS

As we argued in the Introduction, a considerable amount of research in verification of MAS assumes a branching model of time. This is in contrast with work in reactive systems, distributed systems, and mainstream Software Engineering where time is assumed to be linear.

In this paper we have introduced verification algorithms for MAS against CTL*K specifications. This language allows us to refer to linear time, branching time and extensions of both, as well as the epistemic states of the agents in the system. We have shown that the verification problem is no harder than that of plain CTL*. We have also described MCMAS*, an extension of MCMAS, that supports CTL*K specifications and therefore significantly extends the range of problems that MCMAS-based techniques can be applied to. We have shown that the extension comes at little computational penalty against MCMAS.

In terms of future work, the technique we currently use for Exist-Path, though adopted from other state-of-the-art model checking tools such as MCK [15] and NuSMV, could potentially be optimised further. This could involve adopting relatively recent improvements in LTL model checking approaches, such as considering multiple symbolic tableau encodings in parallel [32]. The overall verification approach is modular with respect to the actual implementation of Exist-Path; therefore, any further subsequent improvements to LTL model checking performance should also be readily applicable to our algorithm. To improve the performance even further, we could investigate tuning aspects of the CUDD package (such as its cache parameters, variable ordering strategies, etc.) as well.

While MCK [15] also supports the verification of CTL*K properties. In our comparison we found MCMAS* more efficient in conducting verification. VerICS [27] also supports a fragment of CTL*K. However, since VerICS is based on bounded model checking, the underlying technique is tailored to the existential fragment of CTL*K only.

By significantly extending the specification language supported, we believe that a number of applications ranging from services, to security, robotics and beyond that use MCMAS as their underlying validation toolkit can be analysed in more detail, or in variants that could not be considered until now. Furthermore, advanced verification techniques built on MCMAS can now potentially be extended to support CTL*K specifications. We leave both subjects for further work.

REFERENCES

[1] N. Alechina, B. Logan, H. N. Nguyen, F. Raimondi, and L. Mostarda. Symbolic model-checking for resource-bounded ATL. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS15)*, pages 1809–1810, 2015.

[2] J. Bentahar, M. El-Menshawy, H. Qu, and R. Dssouli. Communicative commitments: Model checking and complexity analysis. *Knowledge-Based Systems*, 35:21–34, 2012.

[3] M. Bozzano and A. Villafiorita. The FSAP/NuSMV-SA safety analysis platform. *Software Tools for Technology Transfer*, 9(1):5–24, 2007.

[4] D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.

[5] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV2: An open-source tool for symbolic model checking. In *Proceedings of the 14th International Conference on Computer Aided Verification (CAV02)*, volume 2404 of *LNCS*, pages 359–364. Springer-Verlag, 2002.

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

[7] E. M. Clarke, O. Grumberg, and K. Hamaguchi. Another look at LTL model checking. In *Proceedings of the 6th International Conference on Computer Aided Verification (CAV94)*, volume 818 of *LNCS*, pages 415–427. Springer, 1994.

[8] E. M. Clarke, S. Jha, Y. Lu, and H. Veith. Tree-like counterexamples in model checking. In *Proceedings of the 17th Symposium on Logic in Computer Science (LICS02)*, pages 19–29, 2002.

[9] M. Cohen, M. Dam, A. Lomuscio, and H. Qu. A symmetry reduction technique for model checking temporal-epistemic logic. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI09)*, pages 721–726, 2009.

[10] E. A. Emerson and E. M. Clarke. Using branching-time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2(3):241–266, 1982.

[11] E. A. Emerson and J. Y. Halpern. "Sometimes" and "not never" revisited: on branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986.

[12] E. A. Emerson and C. L. Lei. Modalities for model checking: Branching time logic strikes back. In *Proceedings of the 12th ACM Symposium on Principles of Programming Languages*, pages 84–96, 1985.

[13] J. Ezekiel, A. Lomuscio, L. Molnar, and S. Veres. Verifying fault tolerance and self-diagnosability of an autonomous underwater vehicle. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI11)*, pages 1659–1664. AAAI Press, 2011.

[14] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge*. MIT Press, Cambridge, 1995.

[15] P. Gammie and R. van der Meyden. MCK: Model checking the logic of knowledge. In *Proceedings of 16th International Conference on Computer Aided Verification (CAV04)*, volume 3114 of *Lecture Notes in Computer Science*, pages 479–483. Springer, 2004.

[16] G. D. Giacomo and M. Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI'13)*, pages 854–860, 2013.

[17] W. Hoek and M. Wooldridge. Model checking knowledge and time. In *SPIN 2002 – Proceedings of the Ninth International SPIN Workshop on Model Checking of Software*, 2002.

[18] A. V. Jones and A. Lomuscio. A BDD-based BMC Approach for the Verification of Multi-Agent Systems. In *Proceedings of the 2009 Workshop on Concurrency, Specification, and Programming (CS&P09)*, pages 361–372. Warsaw University Press, 2009.

[19] M. Kacprzak, W. Nabialek, A. Niewiadomski, W. Penczek, A. Pólrola, M. Szreter, B. Woźna, and A. Zbrzezny. Verics 2007 - a model checker for knowledge and real-time. *Fundamenta Informaticae*, 85(1):313–328, 2008.

[20] A. Lomuscio, W. Penczek, and H. Qu. Partial order reduction for model checking interleaved multi-agent systems. *Fundamenta Informaticae*, 101(1–2):71–90, 2010.

[21] A. Lomuscio, W. Penczek, and B. Woźna. Bounded model checking knowledge and real time. *Artificial Intelligence*, 171(16-17):1011–1038, 2007.

[22] A. Lomuscio, H. Qu, and F. Raimondi. MCMAS: A model checker for the verification of multi-agent systems. *Software Tools for Technology Transfer*, 19(1):9–30, 2017.

[23] A. Lomuscio and F. Raimondi. The complexity of model checking concurrent programs against CTLK specifications. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS06)*, pages 548–550. ACM Press, 2006.

[24] A. Lomuscio and F. Raimondi. MCMAS: A model checker for multi-agent systems. In *Proceedings of Tools and Algorithms for Construction and Analysis of Systems 2006, Vienna*, volume 3920 of *Lecture Notes in Computer Science*, pages 450–454. Springer, 2006.

[25] MCK. MCK: Model checking knowledge. http://cgi.cse.unsw.edu.au/∼mck/pmck/.

[26] MCMAS*. MCMAS*. http://vas.doc.ic.ac.uk/software/mcmas/, 2016.

[27] A. Meski, W. Penczek, M. Szreter, B. W.-Szczesniak, and A. Zbrzezny. BDD-versus SAT-based bounded model checking for the existential fragment of linear temporal logic with knowledge: algorithms and their performance. *Autonomous Agents and Multi-Agent Systems*, 28(4):558–604, 2014.

[28] R. v. Meyden and H. Shilov. Model checking knowledge and time in systems with perfect recall. In *Proceedings of the 19th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS99)*, volume 1738 of *Lecture Notes in Computer Science*, pages 432–445. Springer, 1999.

[29] W. Penczek and A. Lomuscio. Verifying epistemic properties of multi-agent systems via bounded model checking. *Fundamenta Informaticae*, 55(2):167–185, 2003.

[30] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th International Symposium Foundations of Computer Science (FOCS77)*, pages 46–57, 1977.

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

[32] K. Y. Rozier and M. Y. Vardi. A multi-encoding approach for LTL symbolic satisfiability checking. In *International Symposium on Formal Methods*, pages 417–431. Springer, 2011.

[33] M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency*, pages 238–266. Springer, 1996.