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 CTLLK 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 $AGEx reset$ representing “in all paths in the future it is possible to perform a reset in the next step”, LTL admits statements such as $GF request$ 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 CTLLK 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

Copyright © 2017, International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.
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 $a_1, \ldots, a_n, a_e$ such that for every local state $l'_i$ in $g_i$, $l'_i = \tau_j(l_i, a_1, \ldots, a_n, a_e)$ 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 $\pi_1(i)$ the $i$-th state of a path $\pi = g_0, g_1, \ldots$; we denote with $\pi^i$ the $i$-th suffix of a path $\pi$; 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 $l_i : G \rightarrow L_i$ 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 $A((GFp) \rightarrow G(EF(K_1(Gq))))$). Formally, the syntax of CTL*K is defined as follows, where $\psi$ refers to a path formula which may or may not hold on a given path, and $\phi$ refers to a state formula which may or may not hold at a given state.

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

$$
\langle \phi \rangle ::= \top \mid p \mid \neg \phi \mid \phi \land \phi \mid E\psi \mid K\phi \mid E\psi \land \phi \mid D_{\psi} \phi \mid C_T \phi
$$

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 ($\top = \neg \bot$, $\phi \lor \phi_2 = \neg(\neg\phi_1 \land \neg\phi_2)$, $\phi_1 \rightarrow \phi_2 = \neg\phi_1 \lor \phi_2$ and $\phi_1 \Leftrightarrow \phi_2 = (\phi_1 \rightarrow \phi_2) \land (\phi_2 \rightarrow \phi_1)$), for path quantification ($A\psi = E\neg\psi$), and for the temporal modalities ($F\phi = (\Gamma U\phi)$, $G\phi = \neg F\neg\phi$).

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 $\pi$ is a path in $IS$. We define satisfaction of a path formula as follows:

- $IS, \pi \models \phi$ if $IS, \pi(0) \models \phi$, where $\phi$ is a state formula.
- $IS, \pi \models \neg \psi$ if it is not the case that $IS, \pi \models \psi$.
- $IS, \pi \models \psi_1 \land \psi_2$ if both $IS, \pi \models \psi_1$ and $IS, \pi \models \psi_2$.
- $IS, \pi \models X\psi$ if $IS, \pi^1 \models \psi$.
- $IS, \pi \models \psi_1 U\psi_2$ if $\exists j \geq 0$ 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 $\psi$ is true in the next state; $\psi_1 U\psi_2$ holds if $\psi_1$ is true until $\psi_2$ becomes true.

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

- $IS, g \models \top$. 

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 $A = \{1, \ldots, n\}$ 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 $i$.
- $P_i : L_i \rightarrow 2^{Act_i \setminus \emptyset}$ is a local protocol for agent $i \in A$, specifying which actions agent $i$ can take from each local state.
- $\tau_i : L_i \times Act_i \times \ldots \times Act_n \times Act_e \rightarrow L_i$ 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 consistent with their protocols; i.e., if $l'_i = \tau_i(l_i, a_1, \ldots, a_n, a_e)$, then for each $i \in \{1, \ldots, n\}$, 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.
- $h : AP \rightarrow 2^{L_1 \times \ldots \times L_n \times L_e}$, 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 $\tau_e : L_e \times Act_e \rightarrow L_e$ and local protocol $P_e : L_e \rightarrow 2^{Act_e \setminus \emptyset}$. 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
The states labelled with \( p \) are:

- \( IS,g \models p \) iff \( g \in h(p) \).
- \( IS,g \models \neg \phi \) if it is not the case that \( IS,g \models \phi \).
- \( IS,g \models \phi_1 \land \phi_2 \) iff both \( IS,g \models \phi_1 \) and \( IS,g \models \phi_2 \).
- \( IS,g \models E\psi \) iff there exists a path \( \pi \) 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 \( IS,g \models \phi \).
- \( IS,g \models E_x \phi \) iff for every state \( g' \in G \) if \( l_i(g) = l_i(g') \) for some agent \( i \in \Gamma \), then \( IS,g \models \phi \).
- \( IS,g \models D_x \phi \) iff for every state \( g' \in G \) if \( l_i(g) = l_i(g') \) for all agents \( i \in \Gamma \), then \( IS,g \models \phi \).
- \( IS,\pi \models C_i \phi \) iff for every state \( g \in G \) if \( g_1,\ldots \) are states such that \( l_i(g) = l_i(g_1), l_i(g_1) = l_i(g_2), \ldots, l_i(g_n) = l_i(\pi(0)) \) 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 \( \psi \) 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 \( \phi \) if \( \phi \) holds in every possible state of the world consistent with its local state. The group modalities \( E, D \) and \( C \) intuitively correspond to \( \phi \) being known by everyone in a group, \( \phi \) being distributed knowledge in the group, and \( \phi \) being common knowledge in the group, respectively. We refer to [14] for more details.

We now introduce a simple example to illustrate some subtleties of the semantics of CTLK. 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 CTLK 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 \( E(GF(K_1 \neg p)) \) is true in \( IS \), because it is satisfied at all the initial states.

The specification language CTLK is very expressive. It includes both CTLK and LTLK: CTLK is equivalent to the subset of CTLK where path formulae are restricted to \( X \phi \) and \( \phi U \phi \), where \( \phi \) is a state formula; LTLK is equivalent to the subset of CTLK composed of \( Ax \psi \) formulae. CTLK 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 \( \Gamma \)”, expressed by the CTLK formula \( EF(K_1 \neg p) \rightarrow FC_{\Gamma}(p) \). 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 CTLK 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 \( \neg \text{broken}_i \); 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 CTLK 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:

\[
\bigwedge_{i=1}^{N} AFG((\neg \text{broken}_i) \lor (KC(\text{broken}_i) \land EX(\neg \text{tunnel}_i)))
\]

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 or LTL, as it uses both universal and existential path quantifiers. Further CTLK specifications are discussed in Section 5.

Given an interpreted system \( IS \), a state \( g \) and a CTLK formula \( \phi \), the model checking problem involves determining whether \( IS,g \models \phi \).

### 3. MODEL CHECKING CTLK

We now present an algorithm which, given an arbitrary CTLK formula \( \phi \) and an interpreted system \( IS \), computes
returns precisely the set of states in which \( \phi \) 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 \( \text{Label} \) and \( \text{Trans} \). The function \( \text{Label}(\phi, IS) \) ultimately outputs the set of states in which the CTL'K state formula \( \phi \) holds; \( \text{Trans}(\psi, IS) \) outputs an LTL formula which holds precisely in the same set of states in which the CTL'K path formula \( \psi \) 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 \( \text{SAT}_K, \text{SAT}_E, \text{SAT}_D \) and \( \text{SAT}_C \) for computing the set of states satisfying the corresponding epistemic modalities; see, e.g., [31]. We also use the procedure \( \text{Exist-Path} \) returning the states in \( IS \) from which there exists a path on which the LTL formula \( \phi \) holds [7]. Finally the algorithm uses the procedure \( \text{New-Proposition} \) returning a fresh atomic proposition when called. Finally, to check whether a CTL'K formula \( \phi \) is satisfied at a given state \( g \) in a model \( IS \), we check whether \( g \in \text{Label}(\phi, IS) \).

\[
\text{Algorithm 1 Labeling Algorithm}
\]

INPUT: CTL'K state formula \( \phi \), interpreted system \( IS \)
OUTPUT: \([\phi]_{IS}\), the set of states of \( IS \) in which \( \phi \) 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 \( \text{Label} \) and \( \text{Trans} \). The function \( \text{Label}(\phi, IS) \) ultimately outputs the set of states in which the CTL'K state formula \( \phi \) holds; \( \text{Trans}(\psi, IS) \) outputs an LTL formula which holds precisely in the same set of states in which the CTL'K path formula \( \psi \) 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 \( \text{SAT}_K, \text{SAT}_E, \text{SAT}_D \) and \( \text{SAT}_C \) for computing the set of states satisfying the corresponding epistemic modalities; see, e.g., [31]. We also use the procedure \( \text{Exist-Path} \) returning the states in \( IS \) from which there exists a path on which the LTL formula \( \phi \) holds [7]. Finally the algorithm uses the procedure \( \text{New-Proposition} \) returning a fresh atomic proposition when called. Finally, to check whether a CTL'K formula \( \phi \) is satisfied at a given state \( g \) in a model \( IS \), we check whether \( g \in \text{Label}(\phi, IS) \).

\[
\text{Algorithm 2 Translation Algorithm}
\]

INPUT: CTL'K path formula \( \psi \), interpreted system \( IS \)
OUTPUT: LTL formula \( \psi' \) where for any path \( \pi \), \( IS, \pi \models \psi \leftrightarrow IS, \pi \models \psi' \)

1. function \( \text{Trans}(\psi, IS) \)
2. if \( \psi \) is an atomic proposition \( p \) then
3. return \( p \)
4. else if \( \psi = \top \) then
5. return \( \top \)
6. else if \( \psi = \neg \psi_1 \) then
7. return \( \neg(\text{Trans}(\psi_1, IS)) \)
8. else if \( \psi = \psi_1 \land \psi_2 \) then
9. return \( \text{Trans}(\psi_1, IS) \land \text{Trans}(\psi_2, IS) \)
10. else if \( \psi = X\psi_1 \) then
11. return \( X(\text{Trans}(\psi_1, IS)) \)
12. else if \( \psi = \psi_1 U \psi_2 \) then
13. return \( \text{Trans}(\psi_1, IS)U \text{Trans}(\psi_2, IS) \)
14. else if \( \psi \) is an epistemic modality or \( E\psi' \) then
15. \( p' = \text{New-Proposition}() \)
16. \( h(p') = \text{Label}(\psi, IS) \)
17. return \( p' \)
18. end if
19. 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 \( \text{Exist-Path} \) [7] can all be implemented symbolically. We now prove the correctness of the algorithm by showing that \( \text{Label}(\phi, IS) \) returns precisely the set of states in which \( \phi \) is true in \( IS \).

**Theorem 1.** For any \( g \in G \), \( g \in \text{Label}(\phi, IS) \) if and only if \( g \in [\phi]_{IS} \).

**Proof.** By structural induction on \( \phi \).

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

- For the epistemic modalities, by the inductive hypothesis, \( \text{Label}(\phi_1, IS) \) 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 \( \text{Exist-Path} \) from [7]. We show that \( \text{Trans} \) preserves the semantics of \( \psi \), i.e., for any path \( \pi, IS, \pi \models \psi \) if and only if \( IS, \pi \models \text{Trans}(\psi, IS) \). 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 \( \psi' \) is computed correctly by \( \text{Label}(\psi', IS) \). No other changes are made to the path formula \( \psi \).

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 \( \text{Label}(\phi, IS) \) is polynomial in the size of the formula (and hence the size of the input), since we only call \( \text{Label} \) up to twice for each instance of an epistemic modality, path quantifier, \( \neg \) or \( \land \). Calls to \( \text{Trans}(\psi, IS) \) 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 \( \text{Label} \), the epistemic procedures, \( \text{Exist-Path} \) and \( \text{Trans} \) 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 SATC 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. □

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 subformulas of our initial formula whilst preserving this complexity.

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

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

$$f(x, n) = \sum_{i=0}^{k} f(y_i, n) + O(p(n)2^x) \leq \sum_{i=0}^{k} c_i p(n) 2^{y_i} + O(p(n)2^x) \leq (C p(n) \sum_{i=0}^{k} 2^{y_i}) + d p(n)2^x \leq (C p(n)) 2^x + d p(n)2^x = (C + d) p(n)2^x$$

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

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($\phi$, IS)**. We consider the various cases:

1. Propositional cases (lines 2–9). These require up to 2 recursive calls to **Label($\phi$, 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 $\phi_1$ 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. $E\phi$ (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 $O(p(|IS|)2^{\|\phi\|})$.

□

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 $\phi$) to determine the states in which $\phi$ holds. MCMAS$^*$ then checks the validity of InitStates $\rightarrow \phi$, returning true if $\phi$ 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 re-processing 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 formulae. 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 cryptography 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: $\phi_0$ (CTLK) and $\phi'_0$ (CTL’K):

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

$$\phi'_0 = A \left( G \left( \bigwedge_{i=0}^{N-1} \bigwedge_{j \in \{0, \ldots, N-1\}, j \neq i} \neg K_i \text{paid}_j \right) \right)$$

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 $1.94 \times 10^{12}$ 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 $K_i \text{paid}_j$ 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 $\phi_1$ 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-

<table>
<thead>
<tr>
<th>$N$</th>
<th>reach (s)</th>
<th>CTLK</th>
<th>CTL’K</th>
<th>BDD memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.022</td>
<td>0.001</td>
<td>0.003</td>
<td>8.81</td>
</tr>
<tr>
<td>4</td>
<td>0.057</td>
<td>0.002</td>
<td>0.005</td>
<td>9.34</td>
</tr>
<tr>
<td>5</td>
<td>0.090</td>
<td>0.002</td>
<td>0.008</td>
<td>9.17</td>
</tr>
<tr>
<td>6</td>
<td>0.172</td>
<td>0.004</td>
<td>0.013</td>
<td>10.42</td>
</tr>
<tr>
<td>7</td>
<td>0.223</td>
<td>0.006</td>
<td>0.020</td>
<td>10.03</td>
</tr>
<tr>
<td>8</td>
<td>0.287</td>
<td>0.009</td>
<td>0.030</td>
<td>10.31</td>
</tr>
<tr>
<td>9</td>
<td>0.493</td>
<td>0.012</td>
<td>0.041</td>
<td>10.80</td>
</tr>
<tr>
<td>10</td>
<td>0.662</td>
<td>0.015</td>
<td>0.058</td>
<td>11.48</td>
</tr>
<tr>
<td>12</td>
<td>2.001</td>
<td>0.037</td>
<td>0.135</td>
<td>21.34</td>
</tr>
<tr>
<td>20</td>
<td>3.544</td>
<td>0.183</td>
<td>0.511</td>
<td>27.13</td>
</tr>
<tr>
<td>25</td>
<td>38.071</td>
<td>0.847</td>
<td>3.036</td>
<td>46.75</td>
</tr>
<tr>
<td>30</td>
<td>60.966</td>
<td>SO</td>
<td>9.180</td>
<td>SO</td>
</tr>
<tr>
<td>40</td>
<td>151.075</td>
<td>SO</td>
<td>34.553</td>
<td>SO</td>
</tr>
<tr>
<td>50</td>
<td>633.898</td>
<td>SO</td>
<td>120.298</td>
<td>SO</td>
</tr>
</tbody>
</table>

Table 1: Verification results for MCMAS’s CTLK and MCMAS’*’s CTL’K algorithms on the Dining Cryptographers protocol, for specifications $\phi_0$ and $\phi'_0$, 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 $N \geq 30$. 


Table 2: Verification results for MCMAS\(^*\) and MCK on the Dining Cryptographers protocol for the property \(\varphi_1\). We encountered segmentation faults when using MCK to verify \(\varphi_1\) with \(N \geq 6\) cryptographers.

\[
\begin{array}{c|cc}
N & \text{MCMAS}\(^*\) & \text{MCK} \\
\hline
3 & 0.033 & 4.163 \\
4 & 0.072 & 142.917 \\
5 & 0.110 & 240.713 \\
6 & 0.201 & \text{SEG FAULT} \\
8 & 0.338 & \text{SEG FAULT} \\
10 & 0.735 & \text{SEG FAULT} \\
15 & 4.131 & \text{SEG FAULT} \\
20 & 23.511 & \text{SEG FAULT} \\
30 & 61.826 & \text{SEG FAULT} \\
40 & 139.949 & \text{SEG FAULT} \\
50 & 551.877 & \text{SEG FAULT} \\
\end{array}
\]

The results reported suggest that MCMAS\(^*\) appears 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\(^*\) 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 non-deterministically violate the protocol when making the announcement of the parity of the coins. We defined the proposition \(\text{obey}_i\) to represent that cryptographer \(i\) is currently following the protocol, i.e. he declares “same” or “different” according to what the protocol rules; \(\neg \text{obey}_i\) 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 \(\varphi_2\):

\[
\varphi_2 = (\text{paid}_0) \rightarrow AG(\psi_{\text{none}} \rightarrow E (\psi_{\text{arbitrary}} \land G (\psi_{\text{none}})))
\]

\[
\psi_{\text{arbitrary}} = GF \neg \text{obey}_0 \land GF \neg \text{obey}_2
\]

\[
\psi_{\text{none}} = \bigwedge_{i=1}^{N-1} \neg K_i \text{paid}_i
\]

\(\varphi_2\) 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 \(\varphi_2\) holds. This is because the other cryptographers cannot share any information and, under observational semantics, they do not remember past announcements. Observe that \(\varphi_2\) 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 \(\varphi_2\) 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 \(GF \neg \text{obey}_0\) 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 \(\varphi_1\) and \(\varphi_2\), we observe that for the same value of \(N\), both MCMAS\(^*\) and MCK require noticeably more time to verify \(\varphi_2\) than \(\varphi_1\). 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 \(3.25 \times 10^{52}\) states as compared to \(1.94 \times 10^{52}\) 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-
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 for MAS against CTL 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 for MAS against CTL 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 for MAS against CTL 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 for MAS against CTL 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 for MAS against CTL 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 for MAS against CTL 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 for MAS against CTL 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 for MAS against CTL 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 for MAS against CTL 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 for MAS against CTL 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 for MAS against CTL 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 for MAS against CTL 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 for MAS against CTL 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 for MAS against CTL 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 for MAS against CTL 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 for MAS against CTL 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 for MAS against CTL where time is assumed to be linear.

Table 3: Verification results for MCMAS∗ and MCK on the iterative Dining Cryptographers protocol for the property \( \phi_2 \). We encountered segmentation faults when using MCK to verify \( \phi_2 \) with \( N \geq 5 \) cryptographers. We ran our tests with a timeout of 6 hours.

<table>
<thead>
<tr>
<th>( N )</th>
<th>time (s)</th>
<th>MCMAS∗</th>
<th>MCK</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.116</td>
<td>172.020</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.207</td>
<td>Timeout</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.370</td>
<td>SEG FAULT</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.613</td>
<td>SEG FAULT</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>2.111</td>
<td>SEG FAULT</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>2.692</td>
<td>SEG FAULT</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>32.371</td>
<td>SEG FAULT</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>41.339</td>
<td>SEG FAULT</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>416.815</td>
<td>SEG FAULT</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>631.159</td>
<td>SEG FAULT</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>1042.779</td>
<td>SEG FAULT</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>Timeout</td>
<td>SEG FAULT</td>
<td></td>
</tr>
</tbody>
</table>

The language extensions 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.

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


