The complexity of model checking concurrent programs against CTLK specifications*

Alessio Lomuscio and Franco Raimondi

Department of Computer Science
University College London – London, UK
{a.lomuscio,f.raimondi}@cs.ucl.ac.uk

Abstract. This paper presents complexity results for model checking formulae of **CTLK** (a logic to reason about time and knowledge in multi-agent systems) in concurrent programs. We apply these results to evaluate the complexity of verifying programs of two model checkers for multi-agent systems: MCMAS and Verics.

1 Introduction

Multi-agent systems (MAS) are a successful paradigm employed in the formalisation of many scenarios [33, 34], including communication protocols, security protocols, autonomous planning, etc. In many instances, MAS are modelled by means of multi-modal logics with modal operators to reason about temporal, epistemic, doxastic, and other properties of agents.

As MAS being modelled grow larger, however, automatic techniques are crucially required for the formal verification of MAS specifications. Accordingly, various authors have investigated the problem of verification for MAS [35, 3, 1, 13, 28, 18, 30]. In particular, [35, 3, 1] reduce the problem of model checking MAS to the verification of temporal-only models, while [28, 18, 30, 13] extend traditional model checking techniques to the verification of MAS. Model checking [9] was traditionally developed for the verification of hardware circuits using temporal logics. Various tools are available for the verification of temporal logics [28, 23, 7, 16], and complexity results for model checking temporal logics are well known [8, 31, 32, 19]. In contrast, model checking for MAS is still in its infancy. In particular, to the best of our knowledge, the complexity of model checking for MAS has been little explored [15].

In this paper we review various complexity results for temporal and multi-modal logics and we investigate the complexity of model checking the logic **CTLK** in concurrent programs. The main result of this paper is presented in Section 3, where we show that the problem of model checking formulae of **CTLK** in concurrent programs is PSPACE-complete. This result allows to establish complexity results for the problem of verifying MCMAS [22] and Verics [28] programs.

This paper is organised as follows. Temporal logics, model checking, and complexity classes are briefly reviewed in Sections 2.1 – 2.3; Section 2.4 introduces the logic

* The present paper is an extended version of [21]. The authors acknowledge support from the EPSRC (grants GR/S49353 and CASE CNA/04/04).

CTLK and presents some results for model checking extensions of temporal logics. Section 3 contains the main result of this paper: the proof of PSPACE-completeness for model checking **CTLK** in concurrent programs. Section 4 presents an application of this result to the evaluation of the complexity of verifying programs for two tools, MCMAS[22] and Verics [28]. We conclude in Section 5.

2 Notation and preliminaries

2.1 Temporal logics and model checking

CTL The language $\mathbf{L}_{CTL}$ of Computational Tree Logic (CTL, [24, 9]) is defined over a set of atomic formulae $AP = \{p, q, \dots\}$ as follows:

$$\varphi ::= p|\neg\varphi|\varphi \vee \varphi|EX\varphi|E[\varphi U\psi]|EG\varphi.$$

The remaining temporal operators to express eventuality and universality can be derived in standard way, for instance: $EF\varphi = E(\top U\varphi)$, and $AG\varphi = \neg EF\neg\varphi$ [17].

CTL formulae are interpreted in *Kripke models*. A Kripke model M for CTL is a tuple $M = (S, R, V, I)$ where S is a set of states, $R \subseteq S \times S$ is a serial transition relation (the *temporal* relation), $V : S \to 2^{AP}$ is an evaluation function, and $I \subseteq S$ is a set of initial states. A *path* $\pi =< \pi_0, \pi_1, \pi_2, \cdots >$ of M is an infinite sequence of states in S such that $(\pi_i, \pi_{i+1}) \in R$ for all $i \geq 0$.

Satisfiability of a CTL formula φ at a state $s \in S$ of a given model M is defined inductively as follows:

$s \models p$ iff $p \in V(s)$,
$s \models \neg\varphi$ iff $s \not\models \varphi$,
$s \models \varphi_1 \vee \varphi_2$ iff $s \models \varphi_1$ or $s \models \varphi_2$,
$s \models EX(\varphi)$ iff there exists a path π such that $\pi_0 = s$ and $\pi_1 \models \varphi$,
$s \models E[\varphi U\psi]$ iff there exists a path π such that $\pi_0 = s$ and a $k \geq 0$ such that $\pi_k \models \psi$ and $\pi_i \models \varphi$ for all $0 \leq i < k$,
$s \models EG(\varphi)$ iff there exists a path π such that $\pi_0 = s$ and $\pi_i \models \varphi$ for all $i \geq 0$.

We write $M \models \varphi$ if φ is satisfied at all states of the Kripke model M (notice that some authors write $M \models \varphi$ when φ is satisfied in the set of initial states I of M; the two approaches are equivalent from a complexity point of view).

Model checking Model checking is the problem of establishing (possibly in automatic way) whether or not a formula φ is satisfied on a given model M. While this check may be defined for a model M of any logic, traditionally the problem of model checking has been investigated mainly for temporal logics. Various tools have been developed for temporal logics [23, 16, 7, 28]. Typically, a tool for temporal logic model checking provides a programming language to describe a Kripke model S and implements efficient techniques for the automatic verification of formulae (see Section 2.3).

2.2 Turing machines and complexity classes

In this section we follow the presentation given in [29]. A k-string Turing machine ($k \geq 1$) is a tuple $TM = (K, \Sigma, \delta, s)$ where K is a set of states, Σ is a set of symbols (the

alphabet of TM), δ is a transition function, and $s \in K$ is an initial state. Additionally, a Turing machine TM is equipped with k "heads" (one for each string) to read symbols from a certain position on the string, signposted by a "cursor". The transition function $\delta : K \times \Sigma^k \rightarrow (K \cup \{h,'' yes'','' no''\}) \times (\Sigma \times \{\Rightarrow, \Leftarrow, -\})^k$ is the *program* of the machine and describes the evolution of the machine. The special symbols $\{\Rightarrow, \Leftarrow, -\}$ denote the direction of the cursor of TM, and $\{h,'' yes'','' no''\}$ are special halting states for TM. At the beginning of a run, TM is provided with an input string $x \in \Sigma^*$ and the heads are at the beginning of each string. We refer to [29] for more details.

The output of a Turing machine TM on input x is denoted by $TM(x)$, and it is defined to be yes (resp. no) if TM halts on state yes (resp. no) on input x. If the machine halts in state h, then $TM(x)$ is defined to be the string on the last tape. A language $L \subseteq \Sigma^*$ is *decided* by a Turing machine TM if, for all strings $x \in L$, $TM(x) = yes$.

A k-string *non-deterministic* Turing machine is a tuple $NTM = (K, \Sigma, \Delta, s)$, where Δ is a transition *relation* $\Delta \subseteq K \times \Sigma^k \times (K \cup \{h,'' yes'','' no''\}) \times (\Sigma \times \{\Rightarrow, \Leftarrow, -\})^k$

A language $L \subseteq \Sigma^*$ belongs to the *complexity class* TIME$(f(n))$ if there exists a deterministic Turing machine deciding L in time $f(n)$. A language $L \subseteq \Sigma^*$ belongs to the *complexity class* SPACE$(f(n))$ if there exists a deterministic Turing machine deciding L in space $f(n)$ [29]. NTIME and NSPACE are non-deterministic complexity classes defined analogously for non-deterministic Turing machines.

Important complexity classes are L (logarithmic space), NL (non-deterministic logarithmic space), P (polynomial time), NP (non-deterministic polynomial time), PSPACE (polynomial space). The following inclusions hold: $L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE$ [29].

2.3 Model checking concurrent programs

In many practical instances, when using model checkers, states and relations of temporal models are not listed *explicitly*. Instead, a *compact description* is usually given for a model M. Various techniques are available to provide succinct descriptions (variables, program constructors, etc). In this paper we focus on *concurrent programs* [19]. Concurrent programs offer a suitable framework to investigate the complexity of model checking when compact representations are used because, as exemplified in Section 4, various techniques can reduced concurrent programs[1].

Formally, a program is a tuple $D =< AP, AC, S, \Delta, s^0, L >$, where AP is a set of atomic propositions, AC is a set of actions, S is a set of states, $\Delta : S \times AC \rightarrow S$ is a transition function, s^0 is the initial state, and $L : S \rightarrow 2^{AP}$ is a valuation function. Given n programs $D_i =< AP_i, AC_i, S_i, \Delta_i, s_i^0, L_i >$ ($i \in \{1, \ldots, n\}$), a concurrent

[1] Notice that some authors [31] define the problem of establishing whether or not a formula φ holds on a model whose description is given in a compact way with the term *symbolic model checking*. On the other hand, other authors [25] define *symbolic model checking* to be a technique that "avoids building a state graph by using Boolean formulas to represent sets and relations." To avoid confusion, we will refer to symbolic model checking in the latter, stricter sense.

program $D_C =< AP_C, AC_C, S_C, \Delta_C, s^0_C, L_C >$ is defined as the parallel composition of the n programs D_i, as follows:

- $AP_C = \cup_{1 \leq i \leq n} AP_i$;
- $AC_C = \cup_{1 \leq i \leq n} AC_i$;
- $S_C = \prod_{1 \leq i \leq n} S_i$;
- $(s, a, s') \in \Delta_C$ iff
  - $\forall 1 \leq i \leq n$, if $a \in AC_i$, then $(s[i], a, s'[i]) \in \Delta_i$, where $s[i]$ is the i-th component of a state $s \in S$.
  - if $a \notin AC_i$, then $s[i] = s'[i]$;
- $L_C(s) = \cup_i L_i(s[i])$.

(in the remainder, we will drop the subscript C when this is clear from the context)

CTL formulae can be interpreted in a (concurrent) program D by using the standard Kripke semantics for CTL formulae in a model $M = (S, R, V)$. Indeed, the set of states S of M can be taken to be set of states S of D, the temporal relation R can be defined by Δ, and the evaluation function V can be defined by L (we refer to [19] for more details). By slight abuse of notation, we will sometimes refer to the programs D_i and to D with the term "Kripke models".

Summary of known results for temporal logics model checking Traditionally, the complexity of temporal logics model checking has been investigated assuming that models are given explicitly. In this approach, complexity is given as a function of the size of the model and of the size of the formula. Known results are reported in Table 1.

Logic	Complexity
CTL [8, 31]	P-complete
LTL [32]	PSPACE-complete
CTL* [8, 32]	PSPACE-complete
μ-calculus [19]	MC$\in$ NP $\cap$ co-NP

Table 1. The complexity of model checking for some temporal logics.

The complexity of model checking concurrent programs against CTL specifications is investigated in [19]; the authors analyse first the *program complexity* of model checking, i.e., the complexity of model checking as a function of the size of the model only (with a fixed formula). Results are presented in Table 2.

Based on these results, the authors of [19] employ automata-based techniques to evaluate the complexity of model checking as a function of the size of the formula and the sum of the sizes of the concurrent programs constituting D. Their results are presented in Table 3.

Logic	Program complexity
CTL	NLOGSPACE-complete
CTL*	NLOGSPACE-complete
μ-calculus	P-complete

Table 2. Program complexity of model checking for some temporal logics in concurrent programs.

Logic	Program complexity	Complexity
CTL	PSPACE-complete	PSPACE-complete
CTL*	PSPACE-complete	PSPACE-complete
μ-calculus	EXPTIME-complete	EXPTIME

Table 3. Program complexity and complexity of model checking for some temporal logics.

2.4 CTLK

CTLK is an extension of CTL with epistemic operators [10]. Well-formed **CTLK** formulae are defined by the following grammar:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid EX\varphi \mid EG\varphi \mid E[\varphi U \psi] \mid \ K_i\varphi.$$

The formula $K_i\varphi$ expresses the fact that agent i *knows* φ.

CTLK formulae can be interpreted in a Kripke model $M = (W, R_t, \sim_1, \ldots, \sim_n, V)$ where W is a set of states, $R_t \subseteq S \times S$ is a serial transition relation (the *temporal* relation), $\sim_i \subseteq S \times S$ are equivalence relations (the *epistemic* relations), and $V : S \rightarrow 2^{AP}$ is an evaluation function for a given set AP of atomic propositions. Formulae are interpreted in a standard way, by extending the interpretation of CTL formulae of Section 2.1 with the following:

$M, w \models K_i\varphi$ iff for all $w' \in W$, $\sim_i (w, w')$ implies $M, w' \models \varphi$,

Notice that **CTLK** is a multi-dimensional logic obtained by the fusion (or independent join) [12, 2] of CTL with $\mathbf{S5}^n$, where n is the number of distinct epistemic modalities.

CTLK formulae can be interpreted in concurrent programs: the temporal operators of **CTLK** are interpreted as in [19], while epistemic operators are evaluated by defining epistemic accessibility relations based on the equivalence of the components of the states of a concurrent program (a similar approach can be found in [10]). Specifically, let $D =< AP, AC, S, \Delta, s^0, L >$ be a concurrent program obtained by the parallel composition of n programs $D_i =< AP_i, AC_i, S_i, \Delta_i, s_i^0, L_i >$ ($i \in \{1, \ldots, n\}$). Notice that a state $s \in S$ is a tuple $(s_1, \ldots, s_n)$ such that, for all $i \in \{1, \ldots, n\}$, $s_i \in S_i$. We define two states $s = (s_1, \ldots, s_n)$ and $s' = (s'_1, \ldots, s'_n)$ to be related via the epistemic accessibility relation $\sim_i$ iff $s_i = s'_i$, i.e., two states of S are related via the epistemic relation $\sim_i$ iff the i-th components of the two states are identical.

Known results about model checking temporal-epistemic logics An upper bound for "explicit" model checking formulae on Kripke models is given by the following theorem.

Theorem 1. ([10], p.63) *Consider a Kripke model $M = (W, R_1, \ldots, R_n, V)$ for a normal modal logic (e.g. $\mathbf{S5}^n$, $\mathbf{K}$, etc.) and a formula φ. There is an algorithm that, given a model M and a formula φ, determines in time $O(|M| \times |\varphi|)$ whether or not $M \models \varphi$.*

The time complexity for model checking fusion (independent join) of logics can be derived using the following theorem [11]:

Theorem 2. *Let $M = (W, R_1, R_2, V)$ be a model for the fusion of two logics $\mathbf{L}_1$ and $\mathbf{L}_2$, and φ a formula of $\mathbf{L}_1 \oplus \mathbf{L}_2$ (where $\oplus$ denotes the fusion of two logics). The complexity of model checking for $\mathbf{L}_1 \oplus \mathbf{L}_2$ on input φ is:*

$$O(m_1 + m_2 + n \cdot n) + \sum_{i=1}^{2}((O(k) + O(n)) \cdot C_{\mathbf{L}_i}(m_i, n, k))$$

where $m_i = |R_i|$, $n = |W|$, $k = |\varphi|$, and $C_{\mathbf{L}_i}$ is the complexity of model checking for logic $\mathbf{L}_i$, as a function of m_i, n and k.

The following lower bound can be shown:

Lemma 1. *Model checking is P-hard for the logic $\mathbf{K}$, for $\mathbf{D}$, and for any normal logic obtained by fusion (aka independent join), in which one of the components is either $\mathbf{K}$, or $\mathbf{D}$, or CTL.*

Proof. Following the approach of [31] for CTL, by reduction of a P-complete problem to model checking. Consider SAM2CVP (synchronous alternating monotone fanout 2 circuit value problem [14]). Any circuit can be reduced to a Kripke model for $\mathbf{K}$ or for $\mathbf{D}$ (but not to models for other logics, such as $\mathbf{T}$, where accessibility relations are constrained). Consider then the formula $\varphi = \Diamond\Box\Diamond \ldots \Box\Diamond 1$. The circuit evaluate to 1 iff $M, w_0 \models \varphi$.

The lemma above gives an immediate P-completeness result for the logic $\mathbf{CTLK}$ *with common knowledge*. Indeed, a P-time algorithm is provided in [26] for model checking epistemic operators and common knowledge in $\mathbf{S5}^n$, and CTL is known to be P-complete (see Table 1).

3 The complexity of model checking CTLK in concurrent programs

Similarly to temporal logics, model checkers for multi-modal logics accept a "compact" description of Kripke models. In this section we present a proof for the PSPACE-completeness of the problem of model checking $\mathbf{CTLK}$ in concurrent programs; this result will be employed in Section 4 to investigate the complexity of existing tools.

Following the approach of Section 2.3, we will analyse the complexity of model checking a concurrent program $D =< AP, AC, S, \Delta, s^0, L >$ obtained by the parallel composition of n programs $D_i =< AP_i, AC_i, S_i, \Delta_i, s_i^0, L_i >$.

We first introduce some lemmas that will be used in the proof of the main theorem. Lemma 3 states that, if the formula $EG\varphi$ is true at a state s of a model M, then φ is true on a path of length $|M|$ starting from s *and vice-versa*. Corollary 4 states that, if $E[\varphi U \psi]$ is true at a state s of a model M, then there is a state s' on a path starting from s at a distance not greater than $|M|$ from s, in which $s' \models \psi$, and such that φ holds in all states from s to s'. Moreover, we report three well known theorems, as variations of these will be used in the proof of Theorem 7.

Theorem 3. *Given a Kripke model $M = (S, R, V, I)$ for CTL, a state $s \in S$, and a formula φ, $M, s \models EG\varphi$ iff there exists a path π starting from s of length $|\pi| \geq |M|$ s.t. $M, \pi_i \models \varphi$ for all $0 \leq i \leq |M|$.*

Proof. If $M, s \models EG\varphi$, then there exists a path π from s such that, for all $i \geq 0$, $M, \pi_i \models \varphi$; as the relation R is serial, this path is infinite (so, obviously, $|\pi| \geq |M|$).

Conversely, if there is a path π from s of length $|\pi| \geq |M|$, then such a path must necessarily include a backward loop. As $M, \pi_i \models \varphi$ for all i in this loop, it suffices to consider the (infinite) trace generated by this loop to obtain a (semantical) witness for $M, s \models EG$.

Theorem 4. *Given a Kripke model $M = (S, R, V, I)$ for CTL, a state $s \in S$, and two formulae φ and ψ, $M, s \models E[\varphi U \psi]$ iff there exists a path π starting from s s.t. $M, \pi_i \models \psi$ for some $i \leq |M|$, and $M, \pi_j \models \varphi$ for all $0 \leq j \leq i$.*

Proof. If $M, s \models E[\varphi U \psi]$, by the definition of the until operator, there must exist a state s' in which ψ holds, and φ holds in every state from s to s'. Moreover, the state s' cannot be at a "distance" greater than $|M|$ from s.

The other direction is obvious.

The proof of Theorem 7 requires a procedure for establishing whether or not two states $s, s' \in S$ of a Kripke model M are connected via a temporal path. Moreover, the same proof requires a procedure to convert a non-deterministic Turing machine into a deterministic one. Both problems are in fact instances of the same problem: reachability of two nodes in a graph. Formally, given a graph G and two nodes $(x, y) \in G$, REACHABILITY is the problem of establishing whether there is a path from x to y or not. The following known theorems are related to REACHABILITY.

Theorem 5. (Savitch's Theorem) REACHABILITY $\in \mathrm{SPACE}(\log^2(n))$.

Corollary 1. ([29], p.150) $\mathrm{NSPACE}(f(n)) \subseteq \mathrm{SPACE}(f^2(n))$.

Notice that, by Corollary 1, $\mathrm{NPSPACE} = \mathrm{PSPACE}$.

Theorem 6. ([29], p.153) $\mathrm{NSPACE}(f(n)) = \mathrm{co} - \mathrm{NSPACE}(f(n))$.

We are now ready to provide a proof for the main claim of this section:

Theorem 7. *Symbolic model checking for* $\mathbf{CTLK}$ *is PSPACE-complete.*

Proof. Proof idea: Given a formula of φ of $\mathbf{CTLK}$ and a concurrent program D, we define a non-deterministic polynomially-space bounded Turing machine T that halts in an accepting state iff $\neg\varphi$ is satisfiable in D (i.e. iff there exists a state $s \in S$ s.t. $D, s \models \neg\varphi$). Based on this, we conclude that the problem of model checking is in co-NPSPACE. From this, considering Corollary 1 and Theorem 6, we conclude that symbolic model checking for $\mathbf{CTLK}$ is PSPACE-complete (the lower bound being given by the complexity of symbolic model checking CTL).

Proof details: T is a multi-string Turing machine whose inputs are D and φ. T operates "inductively" on the structure of the formula φ (see also [6] for similar approaches), by calling other machines ("sub-machines") dealing with a particular logical operator. The input of T includes the states of the program S_i ($1 \leq i \leq n$), the transition relations, the evaluation functions and all the other input parameters of each Δ_i. This information can be stored on a single input tape, separated by appropriate delimiters. The formula φ is negated, and then it stored on the same tape. The following is a description of the "program" of T.

The machine T starts by guessing a state s and by verifying that s is reachable from the initial state; if it is not, the machine halts in a "no" state. The algorithm of Theorem 5 can be used here, but notice that a polynomial amount of space is needed to store a state of D (as it is the product of states of D_i); this algorithm uses the transition relations Δ_i encoded in the input tapes to verify reachability. In the remainder of this proof, we assume that whenever a new state is "guessed", it is also checked for reachability from the initial state.

The computation proceeds recursively on the structure of $\neg\varphi = \psi$ by calling one of the machines described below. Each machine accepts a state s and a formula, and returns either 0 (the formula is false in s) or 1 (the formula is true in s). Notice that each machine can call any other machine. The following is a description of the formula-specific machines:

- The machine T_p for atomic formulae simply checks whether or not the state is in $L(s)$; if it is, then the machine returns 1. Otherwise, it returns 0.
- The machine $T_\neg$ for formulae of the form $\psi = \neg\psi'$ calls the appropriate machine for ψ' and returns the opposite.
- The machine $T_\vee$ for disjunction of the form $\psi = \psi' \vee \psi''$ first calls the machine for ψ', and then for ψ'', and returns the appropriate result.
- The machine T_{EX} for formulae of the form $\psi = EX(\varphi')$ is as follows: Consider the machine that guesses a state $s' \in S$, checks whether it is reachable with a temporal transition from s, and then calls the sub-machine for φ' (if s' is not reachable, the machine halts in a "no" state). Notice that this sub-machine will return 1 iff it can "guess" an appropriate successor where φ' holds, and it uses at most a polynomial amount of space. By Corollary 1, it is possible to build a *deterministic* machine based on this non-deterministic machine returning either 0 or 1 in polynomial space; T_{EX} is taken to be this "deterministic" machine.
- The machine T_{EG} for formulae of the form $\psi = EG(\varphi')$ is as follows: consider a machine executing the following loop:

```
s-now = s;
counter = 0;
do
  guess a state s';
  check that s' is reachable from s-now;
      if s' is not reachable, return 0;
  if (f does not hold in s') then
     return 0;
  else
     s-now = s';
  end if
  if (counter > |M|)
    return 1;
  else
    counter = counter + 1;
  end if
end do
```

Based on Lemma 3, this machine guesses a path of length greater than $|M|$ (this value can be computed by considering the size of the input) in which φ' holds. When (and if) such a path is found, the machine returns 1 (notice that this machine uses a polynomial amount of space and always halts). By Corollary 1, it is possible to build a deterministic machine T_{EG} in PSPACE that returns 1 iff there exists a path of length greater than $|M|$ in which φ' holds.

– The machine T_{EU} for formulae of the form $\psi = E[\varphi' U \psi'']$ is as follows. Consider the machine executing this code:

```
s-now = s;
counter = 0;
do
  if ( psi'' holds in s-now) then
    return 1;
  else
    if ( psi' does not hold in s-now) then
      return 0;
    else
      guess a state s';
      check that s' is reachable from s-now;
         if s' is not reachable return 0;
      s-now = s';
      counter = counter + 1;
    end if
  end if
  if ( counter > |M| )
    return 0;
  end if
end do
```

This machine implements the idea of Corollary 4: it tries to find a state s' in which ψ'' holds and which is at a distance not greater than $|M|$ from s. As in the previous cases, the machine is non-deterministic, it uses a polynomial amount of space, and it always halts; thus, by Corollary 1, a deterministic machine T_{EU} can be built that uses only a polynomial amount of space.

– The machine T_K for formulae of the form $\psi = K_i\varphi'$ is as follows. Consider a sub-machine that guesses a state $s' \in S$, checks whether it is reachable with an *epistemic* transition from s (i.e. it checks whether the i-th component of the two states are equal), and then calls the sub-machine for $\neg\varphi'$. Notice that this sub-machine will return 1 iff it can "guess" a appropriate successor where $\neg\varphi'$ holds, and it uses at most a polynomial amount of space. By Corollary 1, it is possible to build a *deterministic* machine T_K based on this non-deterministic machine returning either 0 (if a state in which $\neg\varphi$ holds is reachable form s), or 1 (if no such state exists) in polynomial space.

Each of the machines above uses at most a polynomial amount of space, and there are at most $|\varphi|$ calls to this machines in each run of T. Thus, T uses a polynomial amount of space. □

Notice that this proof differs from the proof of PSPACE-completeness for symbolic model checking CTL presented in [19]. The authors of [19] investigate the complexity of various automata and apply these results to the verification of branching time logics. Unfortunately, it does not seem that their technique can easily extended to epistemic modalities. Thus, the proof above provides an alternative proof of the upper bounds for symbolic model checking CTL, which can be easily extended to CTLK.

4 Applications

MCMAS [22] and Verics [28] are two tools for the automatic verification of multi-agent systems via model checking. Both tools allow for the verification of **CTLK** formulae in Kripke models. MCMAS uses *interpreted systems* [10] to describe Kripke models in a succinct way. Verics employs *networks of automata*. Both approaches can be reduced to *concurrent programs*, and *vice-versa*; thus, Theorem 7 allows to establish PSPACE-completeness results for the problem of verifying MCMAS and Verics programs.

4.1 The complexity of model checking MCMAS programs

MCMAS [22] is a symbolic model checker for interpreted systems. Interpreted systems [10] provide a fine grain semantics for temporal and epistemic operators, based on a system of *agents*. Each agent is characterised by a set of local states, by a set of actions, by a protocol specifying the actions allowed in each local state, and by an evolution function for the local states. MCMAS accepts as input a description of an interpreted system and builds a *symbolic* representation of the model by using Ordered Binary Decision Diagrams (OBDDs, [4]). We refer to [10, 22, 30] for more details. An excerpt of a sample input file for MCMAS is reported in Figure 1.

```
Agent SampleAgent
 Lstate = {s0,s1,s2,s3};
 Lgreen = {s0,s1,s2};
 Action = {a1,a2,a3};
 Protocol:
   s0: {a1};
   s1: {a2};
   s2: {a1,a3};
   s3: {a2,a3};
 end Protocol
 Ev:
   s2 if ((AnotherAgent.Action=a7);
   s3 if Lstate=s2;
 end Ev
end Agent
```

Fig. 1. MCMAS input file (excerpt).

An interpreted systems described in MCMAS can be reduced to a concurrent program: each agent is associated with a program $D_i =< AP_i, AC_i, S_i, \Delta_i, s_i^0, L_i >$, where AC_i is the set of actions for agent i, S_i is the set of local states for agent i, and the evolution function Δ_i is the one provided for the agent.

In the formalism of interpreted systems an agent's evolution function may depend on the other agents' actions. Thus, we modify the definition of a concurrent program $D =< AP, AC, S, \Delta, s^0, L >$ obtained by the composition of n programs D_i (one for each agent), as follows:

- $AP = \cup_{1 \leq i \leq n} AP_i$,
- $AC = \prod_{1 \leq i \leq n} AC_i$,
- $S = \prod_{1 \leq i \leq n} S_i$,
- $(s, a, s') \in \Delta$ iff $\forall 1 \leq i \leq n$, $(s[i], a, s'[i]) \in \Delta_i$,
- $L(s) = \cup_i L_i(s[i])$.

Notice that, instead of taking the union, AC is now the *Cartesian* product of the agents' actions AC_i, and the transition function is modified accordingly. Thus, given an interpreted system and a **CTLK** formula φ described in the formalism of MCMAS, it is possible to obtain a concurrent program D of size equal to the original MCMAS description (modulo some constant), so that the Turing machine T defined in Section 3 can be employed to perform model checking of φ. Hence, we conclude that model checking MCMAS programs is in PSPACE.

Conversely, the problem of model checking a formula φ in the parallel composition of n programs $D_i =< AP_i, AC_i, S_i, \Delta_i, s_i^0, L_i >$ can be reduced to an MCMAS program. Indeed, it suffices to introduce an agent for each program, whose local states are S_i and whose actions are AC_i. The transition conditions for the agent can be taken to be Δ_i, augmented with the condition that a transition between two local states is enabled if all the agents including the same action in AC_i perform the transition labelled with the particular action.

It is worth noticing that the actual implementation of MCMAS requires, in the worst case, an exponential time to perform verification. Indeed, MCMAS uses OBDDs, and it is known [5] that OBDDs may have a size which is exponential in the number of variables used.

4.2 The complexity of model checking Verics programs

Verics [28] is a tool for the verification of various types of timed automata and for the verification of **CTLK** properties in multi-agent systems. In this section we consider only the complexity of verification of **CTLK** properties in Verics.

A multi-agent system is described in Verics by means of a network of (un-timed) automata [20]: each agent is represented as an automaton, whose states correspond to local states of the agent. In this formalism a single set of action is present, and automata synchronise over common actions.

The reduction from Verics code to concurrent programs is straightforward: each automaton is a program D_i and no changes are required for the parallel composition presented in Section 2.3, and similarly a concurrent program can be seen as a network of automata. Thus, we conclude that the problem of model checking Verics programs is PSPACE-complete.

Notice that the actual implementation of Verics performs verification by reducing the problem to a satisfiability problem for propositional formulae. Similarly to MCMAS, this reduction may lead to exponential time requirements in the worst case.

5 Conclusion

In this paper we have reviewed various results about the complexity of model checking for temporal logics, both for "explicit" and for symbolic model checking. We have extended some of these results to richer logics for reasoning about knowledge and time. In particular, we have presented Theorem 7 which provides a result for the complexity of symbolic model checking **CTLK**. To the best of our knowledge, no other complexity results for symbolic model checking temporal-epistemic logics are available, with the exception of [26, 27]. The authors of [26, 27] investigate the complexity of model checking for LTL extended with epistemic operators and common knowledge in synchronous/asynchronous systems with perfect recall. Let $\mathcal{L}_{X,U,K_1,\ldots,K_n,C}$ be the full language of this logic. Complexity results are presented in Table 4. Intuitively, model checking for these semantics is more complex than for the "standard" Kripke semantics (also called "observational" semantics by the authors), because perfect recall causes local states to be unbounded strings, thus "generating" an infinite set of worlds, upon which model checking should be performed.

Our work differs from [26, 27] in analysing the problem of *symbolic* model checking for the generic framework of concurrent programs, in which models are not described explicitly: in turn, the generic result in Theorem 7 provides a concrete methodology to investigate the complexity of verifying MCMAS and Verics programs.

Finally, the work presented here is similar in spirit to [15] where complexity results for the verification of ATL against simple reactive modules are presented.

Language	Complexity
$\mathcal{L}_{K_1,\ldots,K_n,C}$, synchronous	PSPACE-hard
$\mathcal{L}_{K_1,\ldots,K_n,C}$, asynchronous	undecidable
$\mathcal{L}_{X,K_1,\ldots,K_n,C}$, synchronous	PSPACE-complete
$\mathcal{L}_{X,U,K_1,\ldots,K_n}$, synchronous	non-elementary
$\mathcal{L}_{X,U,K_1,\ldots,K_n,C}$, synchronous	undecidable

Table 4. Complexity of MC for some perfect recall semantic.

References

1. M. Benerecetti, F. Giunchiglia, and L. Serafi ni. Model checking multiagent systems. *Journal of Logic and Computation*, 8(3):401–423, 1998.
2. P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2001.
3. R. H. Bordini, M. Fisher, C. Pardavila, and M. Wooldridge. Model checking AgentSpeak. In J. S. Rosenschein, T. Sandholm, W. Michael, and M. Yokoo, editors, *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-agent systems (AAMAS-03)*, pages 409–416. ACM Press, 2003.
4. R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.
5. R. E. Bryant. On the complexity of VLSI implementations and graph representations of boolean functions with application to integer multiplication. *IEEE Trans. Comput.*, 40(2):205–213, 1991.
6. A. Cheng. Complexity results for model checking. Technical Report RS-95-18, BRICS - Basic Research in Computer Science, Department of Computer Science, University of Aarhus, Feb. 1995.
7. 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 (CAV'02)*, volume 2404 of *LNCS*, pages 359–364. Springer-Verlag, 2002.
8. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verifi cation of fi nite-state concurrent systems using temporal logic specifi cations: A practical approach. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.
9. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.
10. R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge*. MIT Press, Cambridge, 1995.
11. M. Franceschet, A. Montanari, and M. de Rijke. Model checking for combined logics with an application to mobile systems. *Automated Software Engineering*, 11:289–321, 2004.
12. D. Gabbay and V. Shehtman. Products of modal logics, part 1. *Logic Journal of the IGPL*, 6(1):73–146, 1998.
13. P. Gammie and R. van der Meyden. MCK: Model checking the logic of knowledge. In *Proceedings of 16th International Conference on Computer Aided Verification (CAV'04)*, volume 3114 of *LNCS*, pages 479–483. Springer-Verlag, 2004.
14. R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory*. Oxford University Press, 1995.

15. W. v. Hoek, A. Lomuscio, and M. Wooldridge. On the complexity of practical atl model checking knowledge, strategies, and games in multi-agent systems. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems (AAMAS'06)*, Hakodake, Japan. ACM Press. To appear.
16. G. J. Holzmann. The model checker SPIN. *IEEE transaction on software engineering*, 23(5):279–295, 1997.
17. M. R. A. Huth and M. D. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, Cambridge, England, 2000.
18. M. Kacprzak, A. Lomuscio, and W. Penczek. From bounded to unbounded model checking for temporal epistemic logic. *Fundamenta Informaticae*, 63(2,3):221–240, 2004.
19. O. Kupferman, M. Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, 2000.
20. A. Lomuscio, T. Łasica, and W. Penczek. Bounded model checking for interpreted systems: preliminary experimental results. In M. Hinchey, editor, *Proceedings of FAABS II*, volume 2699 of *LNCS*. Springer Verlag, 2003.
21. 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 (AAMAS'06)*, pages 548–550, Hakodake, Japan, 2006. ACM Press.
22. A. Lomuscio and F. Raimondi. MCMAS: A model checker for multi-agent systems. In H. Hermanns and J. Palsberg, editors, *Proceedings of TACAS 2006, Vienna*, volume 3920, pages 450–454. Springer Verlag, March 2006.
23. K. McMillan. The SMV system. Technical Report CMU-CS-92-131, Carnegie-Mellon University, Feb. 1992.
24. K. McMillan. *Symbolic model checking: An approach to the state explosion problem*. Kluwer Academic Publishers, 1993.
25. K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.
26. R. Meyden. Common knowledge and update in finite environments. *Information and Computation*, 140(2):115–157, 1998.
27. R. v. Meyden and H. Shilov. Model checking knowledge and time in systems with perfect recall. In *Proceedings of Proc. of FST&TCS*, volume 1738 of *Lecture Notes in Computer Science*, pages 432–445, Hyderabad, India, 1999.
28. W. Nabialek, A. Niewiadomski, W. Penczek, A. Pólrola, and M. Szreter. VerICS 2004: A model checker for real time and multi-agent systems. In *Proceedings of the International Workshop on Concurrency, Specification and Programming (CS&P'04)*, volume 170 of *Informatik-Berichte*, pages 88–99. Humboldt University, 2004.
29. C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
30. F. Raimondi and A. Lomuscio. Automatic verification of multi-agent systems by model checking via OBDDs. *Journal of Applied Logic*, 2005. To appear in Special issue on Logic-based agent verification.
31. P. Schnoebelen. The complexity of temporal logic model checking. In *Proceedings of the 4th Conference Advances in Modal Logic (AiML'2002)*, volume 4 of *Advances in Modal Logic*, pages 437–459. King's College Publications, 2003.
32. A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logic. *Journal of the ACM*, 32(3):733–749, 1985.
33. M. Wooldridge. *Reasoning about Rational Agents*. MIT Press, 2000.
34. M. Wooldridge. *An introduction to multi-agent systems*. John Wiley, England, 2002.
35. M. Wooldridge, M. Fisher, M. Huget, and S. Parsons. Model checking multiagent systems with MABLE. In *Proceedings of the First International Conference on Autonomous Agents and Multiagent Systems (AAMAS-02)*, pages 952–959, Bologna, Italy, July 2002.