Parallel Model Checking for Temporal Epistemic Logic

Marta Kwiatkowska[1] and **Alessio Lomuscio**[2] and **Hongyang Qu**[1]

Abstract. We investigate the problem of the verification of multi-agent systems by means of parallel algorithms. We present algorithms for CTLK, a logic combining branching time temporal logic with epistemic modalities. We report on an implementation of these algorithms and present the experimental results obtained. The results point to a significant speed-up in the verification step.

1 Introduction

Temporal-epistemic logics are a well-known formalism to reason about multi-agent systems [6]. One of its recent applications has been the development of model checking techniques for the verification of multi-agent systems (MAS) specified by means of temporal-epistemic logics. Several approaches have been put forward in this direction. [14] introduced an approach based on bounded model checking for the verification of CTLK, the combination of CTL with epistemic logic. [7] used binary-decision diagrams to perform symbolic model checking on CTLK. This approach was also followed in the development of MCMAS [9], an open-source model checker for MAS. While these techniques and resulting toolkits are capable of checking very considerable state-space, they still suffer from the state-explosion problem. This is a well-known difficulty in verification resulting from the fact that the state-space grows exponentially with the number of variables in the program to be verified. In order to be able to verify large systems it remains of paramount importance to devise methodologies that mitigate this difficulty. Recent research has focused on techniques such as abstraction [5] and symmetry reduction [4] to alleviate the problem.

Even if significant gains can be achieved, ultimately any technique of this kind needs to confront the problem of computing and encoding a very large state-spaces by means of a serial program. For several years research in model checking has been able to benefit from continuously increasing computational power in the underlying single-core computer architectures. However, there are increasing signs that current CPU development is hitting the barriers of the underlying physics, thereby providing only limited scope for faster serial CPUs. Current CPUs already provide several execution cores; the number of cores in a single CPU is expected to increase significantly in the next few years. It is therefore of interest to develop model checking algorithms ready to reap the benefits of the underlying parallelism.

Steps in this direction have already been taken. In [16] the state space is partitioned and the overall model constructed by exploring in parallel the sub-models generated by the representatives in the partitions. This approach was applied to explicit model checking procedures; however, similar algorithms [8, 16] have been devised for symbolic, i.e., OBDD-based, representations as well. A difficulty of these approaches resides in the partitioning process. In a nutshell, if we are performing breath-first search and assign new states to a different thread, at times we are forced to cross-reference the partial sub-models to the different execution threads. This results in a computation overhead that may well offset any possible gain offered by the parallel search. To overcome this difficulty the standard algorithms for computing the set of states satisfying a logical formula [3] have been modified in order to minimise the communication overhead between threads.

In this paper we take inspiration from these observations to develop parallel approaches to symbolic model checking MAS specified by means of branching-time temporal epistemic logic. Specifically, we report on partitioning strategies for the representation of state spaces generated by MAS encoded as interpreted systems [13] and parallel algorithms for the satisfaction of epistemic operators, including distributed and common knowledge.

The rest of the paper is organised as follows. In Section 2 we recall the interpreted systems formalism for MAS and the temporal-epistemic logic CTLK. in Section 3 we give the sequential model checking procedures for CTLK. We introduce a partition strategy and parallel satisfaction algorithms for CTLK in Section 4. In Section 5 we evaluate the methodology by reporting the performance of the algorithms on four scalable models. Section 6 presents directions of future work.

2 Interpreted systems and CTLK

We model a MAS as an interpreted system [6], and follow the presentation in [10]. An interpreted system is composed of a set $\mathcal{A} = \{1, \ldots, n\}$ of agents and an environment e. We assume that at any given time each agent in the system is in a particular local state. We associate a set of instantaneous *local states* L_i to each agent $i \in \mathcal{A}$ and a set L_e to the environment.

To represent the instantaneous configuration of the whole MAS at a given time we use the notion of global state. A global state $s \in S$ is a tuple $s = (l_1, \ldots, l_n, l_e)$ where each component $l_i \in L_i$ represents the local state an agent i is in, together with the environment state. The set of all global states $S \subseteq L_1 \times \ldots \times L_n \times L_e$ is a subset of the Cartesian product of all local states and the local states for the environment. $I \subseteq S$ is a set of initial states for the system.

Each agent i has a repertoire of actions Act_i available, similarly has the environment. It is assumed $null \in Act_i$ for each agent i where $null$ is the null action. The action selection mechanism is given by the notion of local protocol $P_i : L_i \rightarrow 2^{Act_i}$ for any $i \in \mathcal{A}$; P_i is a function giving the set of possible actions that may be performed when in a given local state. In other words $P_i(l_i)$ represents the actions that may be performed by agent i when in the state l_i.

The evolution of the system is given by locked transitions for all

[1] Oxford University Computing Laboratory, UK.
{Marta.Kwiatkowska, Hongyang.Qu}@comlab.ox.ac.uk
[2] Department of Computing, Imperial College London, UK.
A.Lomuscio@imperial.ac.uk

the agents and the environment. The model assumes that each agent moves from local state to local state at each time tick. The transitions between local states depend on which actions have been performed by all agents in the system. So an agent's action may affect another agent's resulting next state. Formally, for each agent we assume a local transition function $\tau_i : L_i \times Act_1 \times \ldots \times Act_n \times Act_e \rightarrow L_i$ defining the local state for agent i resulting from a local state and and a joint action.

Local transitions may be combined together to give a joint transition function $\tau : S \times Act_1 \times \ldots \times \ldots Act_n \times Act_e \rightarrow S$ giving the overall transition function for the system. We write $(s, s') \in \mathcal{T}$ if $\tau(s, a_1, \ldots, a_n, a_e) = s'$ for some joint action $(a_1, \ldots, a_n, a_e)$.

We introduce paths to give an interpretation to a branching time language. A *path* $\pi = (s_0, s_1, \ldots, s_j)$ is a sequence of possible global states such that $(s_i, s_{i+1}) \in \mathcal{T}$ for each $0 \leq i < j$. For a path $\pi = (s_0, s_1, \ldots)$, we take $\pi(k) = s_k$.

Definition 1 (Models)**.** *A model $\mathcal{M} = (S, I, \mathcal{T}, \sim_1, \ldots, \sim_n, \mathcal{L})$ is a tuple such that:*

- *$S \subseteq L_1 \times, \ldots \times L_n \times L_e$ is the set of global states for the system,*
- *$I \subseteq S$ is a set of initial states for the system,*
- *$\mathcal{T}$ is the temporal relation for the system defined as above,*
- *For each agent $i \in \mathcal{A}$, $\sim_i$ is an epistemic indistinguishably relation defined by $(l_1, \ldots, l_n, l_e) \sim_i (l'_1, \ldots, l'_n, l'_e)$ if $l_i = l'_i$.*
- *$\mathcal{L} : S \rightarrow 2^{AP}$ is a labelling function over the set AP of atomic propositions.*

The above models allow us to interpret a temporal epistemic language. The relation $\mathcal{T}$ is used to interpret temporal operators whereas $\sim_i$ is used to interpret epistemic modalities [6]. In addition to knowledge for individual agent, we can define knowledge with respect to a group $\Gamma \subseteq \mathcal{A}$ of agents in the following way.

- Everybody knows: $\sim^E_\Gamma = \bigcup_{i \in \Gamma} \sim_i$. We have $s \ \sim^E_\Gamma \ s'$ iff $\forall i \in \Gamma$ such that $s \sim_i s'$.
- Distributed knowledge $\sim^D_\Gamma = \bigcap_{i \in \Gamma} \sim_i$. We have $s \ \sim^D_\Gamma \ s'$ iff $\exists i \in \Gamma$ such that $s \sim_i s'$.
- Common knowledge: $\sim^C_\Gamma = (\bigcup_{i \in \Gamma} \sim_i)^+$, where $^+$ denotes the reflexive transitive closure of the underlying relation.

The syntax of the temporal epistemic logic CTLK is given by the following BNF notation.

Definition 2 (Syntax of CTLK)**.**

$$\begin{aligned} \phi \quad ::= \quad & p \mid \neg\phi \mid \phi \vee \psi \mid EX\phi \mid E\phi U\psi \mid EG\phi \mid \\ & K_i\phi \mid E_\Gamma\phi \mid D_\Gamma\phi \mid C_\Gamma\phi. \end{aligned}$$

In the above definition, p is an atomic proposition, the connectives X, G and U are CTL path operators, standing for "next", "globally" and "until", respectively. E is the existential quantifier on paths. The modal connectives K_i, E_Γ, D_Γ and C_Γ stand for knowledge, everybody knows, distributed knowledge and common knowledge respectively. $K_i\phi$ means that agent i knows ϕ; $E_\Gamma\phi$ means all agents in group Γ know ϕ; $D_\Gamma\phi$ means one agent in group Γ knows ϕ; $C_\Gamma\phi$ means ϕ is common knowledge in group Γ. Other temporal modalities, e.g., F, and the universal path quantifier A can defined in terms of the above as usual.

When a CTLK formula ϕ is evaluated to true in a global state s in an IS $\mathcal{M}$, we say that ϕ is satisfied in s, denoted by $(\mathcal{M}, s) \models \phi$. Let $\mathcal{L}(s) \subseteq AP$ be set of atomic propositions satisfied in s.

Definition 3 (Satisfaction)**.** *Given an IS $\mathcal{M}$, the satisfaction of a CTLK formula ϕ in a global state s is recursively defined as follows.*

- *$(\mathcal{M}, s) \models p$ iff $p \in \mathcal{L}(s)$;*
- *$(\mathcal{M}, s) \models \neg\phi$ iff it is not the case that $(\mathcal{M}, s) \models \phi$;*
- *$(\mathcal{M}, s) \models \phi \vee \psi$ iff $(\mathcal{M}, s) \models \phi$ or $(\mathcal{M}, s) \models \psi$;*
- *$(\mathcal{M}, s) \models EX\phi$ iff there exists a path π starting at s such that $(\mathcal{M}, \pi(1)) \models \phi$.*
- *$(\mathcal{M}, s) \models EG\phi$ iff there exists a path π starting at s such that $(\mathcal{M}, \pi(k)) \models \phi$ for all $k \geq 0$;*
- *$(\mathcal{M}, s) \models E\phi U\psi$ iff there exists a path π starting at s such that for some $k \geq 0$, $(\mathcal{M}, \pi(k)) \models \psi$ and $(\mathcal{M}, \pi(j)) \models \phi$ for all $0 \leq j < k$;*
- *$(\mathcal{M}, s) \models K_i\phi$ iff for all $s' \in S$ if $s \sim_i s'$ then $(\mathcal{M}, s') \models \phi$.*
- *$(\mathcal{M}, s) \models E_\Gamma\phi$ iff for all $s' \in S$ if $s \ \sim^E_\Gamma \ s'$ then $(\mathcal{M}, s') \models \phi$.*
- *$(\mathcal{M}, s) \models D_\Gamma\phi$ iff for all $s' \in S$ if $s \ \sim^D_\Gamma \ s'$ then $(\mathcal{M}, s') \models \phi$.*
- *$(\mathcal{M}, s) \models C_\Gamma\phi$ iff for alls' $\in S$ if $s \ \sim^C_\Gamma \ s'$ then $(\mathcal{M}, s') \models \phi$.*

In model checking we are normally interested in checking whether a formula ϕ is satisfied in a model $\mathcal{M}$, which is equivalent to whether ϕ is satisfied in all initial states I, i.e.,

$$(\mathcal{M}, I) \models \phi \textit{ iff for all } s \in I, (\mathcal{M}, s) \models \phi.$$

3 Model Checking CTLK formulae

Given a MAS represented as an interpreted system and a specification $\phi \in CTLK$, the model checking problem involves checking whether $(\mathcal{M}, I) \models \phi$, i.e., establishing whether the formula ϕ is satisfied in the system starting from initial states. Symbolic approaches tackle this problem by computing the set of states in $\mathcal{M}$ that satisfy ϕ by means of the transition relation $\mathcal{T}$ and compare it against the set of initial states I in $\mathcal{M}$. Recall that sets can be easily represented in terms of ordered-binary decision diagrams (OBDDs); so any algorithm can be implemented directly on OBDDs [1].

Several procedures exist to calculate the set of reachable states. In Procedure 1, reported below, the function $image(next, \mathcal{T})$ returns the set of successor states of the set $next$ of states with respect to $\mathcal{T}$. The set $next$ of states is the *frontier* during the generation.

Procedure 1 $REACH(I, \mathcal{T})$

1: $S \Leftarrow \emptyset$; $next \Leftarrow I$; $S' \Leftarrow I$
2: **while** $S \neq S'$ **do**
3: $\quad S \Leftarrow S'$; $next = Image(next, \mathcal{T})$; $S' \Leftarrow S \cup next$;
4: **end while**
5: **return** S;

The second step in the model checking procedure is to calculate SAT_ϕ, the set of states in $\mathcal{M}$ that satisfy the formula ϕ. The procedure for calculating SAT_ϕ for $\phi \in CTLK$ is given in [15] and results from an extension of the algorithms given in [3] for CTL. Given that in the sequel we do not modify the algorithms for the temporal modalities, below we only report the cases for the epistemic modalities.

Procedure 2 reports the algorithm for the basic epistemic modality. In a nutshell we first compute the set of states for $\neg\phi$, then construct the set of states that can "see" by means of the epistemic relation a state satisfying $\neg\phi$, and finally we return the complement of this set.

Procedure 2 $SAT_K(\phi, i)$ for $K_i\phi$.

$X \Leftarrow SAT_{\neg\phi}$;
$Y \Leftarrow \{s \in S \mid \exists s' \in X$ such that $s \sim_i s'\}$;
return $\neg Y \cap S$;

The procedure for everybody knows (distributed knowledge, respectively) is similar to that above, except that the relation considered is the union (intersection, respectively) of the epistemic relations in Γ.

Procedure 3 $SAT_E(\phi, \Gamma)$ for $E_\Gamma\phi$.

$X \Leftarrow SAT_{\neg\phi}$;
$Y \Leftarrow \{s \in S \mid \exists s' \in X$ such that $\exists i \in \Gamma, s \sim_i s'\}$
return $\neg Y \cap S$;

Procedure 4 $SAT_D(\phi, \Gamma)$ for $D_\Gamma\phi$.

$X \Leftarrow SAT_{\neg\phi}$;
$Y \Leftarrow \{s \in S \mid \exists s' \in X$ such that $\forall i \in \Gamma, s \sim_i s'\}$
return $\neg Y \cap S$;

Computing C_Γ normally involves a fix point computation. For efficiency we use the algorithm below. Procedure 5 starts from the set of states where ϕ is not satisfied and repeatedly extends it by adding any state related by an agent in Γ to any of state in the working set. The set of states satisfying C_Γ is the complement of the result of the recursive computation above.

Procedure 5 $SAT_C(\phi, \Gamma)$ for $C_\Gamma\phi$.

$X \Leftarrow S; Y \Leftarrow SAT_{\neg\phi}$;
while $X \neq Y$ **do**
  $X \Leftarrow Y$;
  $Y \Leftarrow \{s \in S \mid \exists s' \in X$ and $\exists i \in \Gamma$ such that $s \sim_i s'\}$;
end while
return $\neg X \cap S$;

Since we can calculate the set S of reachable states and the set SAT_ϕ of states satisfying any formula $\phi \in CTLK$, we can now give the general model checking algorithm, reported in Procedure 6. Effectively, the algorithm checks whether the formula in consideration is true at all initial states ($I \subseteq SAT_\phi$).

Procedure 6 $CHECK(\mathcal{M}, \phi)$

if $I \subseteq SAT_\phi$ **then**
  return TRUE
else
  return FALSE
end if

4 Parallel model checking algorithm for CTLK

In this section we present the proposed parallel approach to verifying CTLK. Given a model $\mathcal{M}$ and a formula ϕ to be checked we follow the steps below.

1. We first partition the set I of initial states and assign each partition to a process.
2. We then compute the set of reachable states in each partition in parallel;
3. Finally, we carry out model checking checks simultaneously on all sets of reachable states.

The only communication requirement in the above is in Step 3, where it is possible that a process may require to access states being computed by another process.

In more detail, assume I is divided into m partitions $I_1, \ldots, I_m$. We define $\mathcal{M}_k = \langle S_k, I_k, \mathcal{T}_k, \sim_1^k, \ldots, \sim_n^k, \mathcal{L}_k\rangle$ ($1 \leq k \leq m$) to be a *submodel* of $\mathcal{M}$ if $S_k \subseteq S$ is the set of states reachable from the states in I_k, and $\sim_i^k$ ($\mathcal{L}_k$, $\mathcal{T}_k$, respectively) is the projection of $\sim_i$ ($\mathcal{L}$, $\mathcal{T}_k$, respectively) onto S_k, i.e, for all $s, s' \in S_k$, $(s, s') \in \mathcal{T} \Leftrightarrow (s, s') \in \mathcal{T}_k$, $s \sim_i^k s' \Leftrightarrow s \sim_i s'$ and $\mathcal{L}_k(s) = \mathcal{L}(s)$. Note that for constructing the set of reachable states from any I_k we can equally use the relations $\mathcal{T}, \sim_i$, for any $i \in A$.

In view of the remarks at the end of the previous section we begin by giving a general procedure for model checking that can be parallelised.

Procedure 7 $P\_CHECK(\mathcal{M}, \phi)$ for checking $(\mathcal{M}, I) \models \phi$

for $k = 1$ to m **do**
  if $CHECK_p(\mathcal{M}_k, \phi)$=FALSE **return** FALSE **end if**
end for
return TRUE

Clearly the **for** loop can be made parallel by means of m *parallel* processes (PPs), simply calculating the reachable states in the corresponding submodel and checking the satisfiability of the formula on it (see Procedure 8). Every PP executes step 1 in Procedure 8 independently and afterwards executes $CHECK_p(\mathcal{M}_k, \phi)$ with limited synchronisation with other PPs. We then generate a *control* process (CP) to collect the return values from the individual PPs, thereby implementing $P\_CHECK(\mathcal{M}, \phi)$.

Procedure 8 $SIMPLE\_PARA(k, \phi)$

1: $S_k \Leftarrow REACH(I_k)$;
2: $CHECK_p(\mathcal{M}_k, \phi)$;

The distributed procedure $CHECK_p(\mathcal{M}_k, \phi)$ to run on the submodel is identical to $CHECK(\mathcal{M}, \phi)$ apart the test of I_k against the set of states returned by the P_SAT_ϕ procedures.

Procedure 9 $CHECK_p(\mathcal{M}_k, \phi)$

if $I_k \subseteq P\_SAT_\phi$ **then**
  return TRUE
else
  return FALSE
end if

The P_SAT_ϕ procedure for the cases p, EX, EG, EU is the same as the sequential procedures SAT_ϕ, thereby reducing the synchronisations among PPs. The parallel procedures for K_i, E_Γ, D_Γ, C_Γ cases are defined as follows.

The procedure $P\_SAT_K(\phi, i, k, S_k)$ differs from the serial $SAT_K(\phi, i, k, S_k)$ by means of a loop to get all reachable states in which ϕ is not satisfied. The **for** loop needs to synchronise with other PPs: each PP k needs to get a copy of X_j ($1 \leq j \neq k \leq m$) from other PP j.

Procedure 10 $P\_SAT_K(\phi, i, k, S_k)$ for $K_i\phi$.

$X_k \Leftarrow P\_SAT(\neg\phi)$; $X \Leftarrow \emptyset$;
for $j = 1$ to m **do** $X \Leftarrow X \cup X_j$; **end for**
$Y \Leftarrow \{s \in S_k \mid \exists s' \in X$ such that $s \sim_i s'\}$;
return $\neg Y \cap S_k$;

The procedures $P\_SAT_E(\phi, \Gamma, k, S_k)$ and $P\_SAT_D(\phi, \Gamma, k, S_k)$ are obtained in the similar way from $SAT_E(\phi, \Gamma)$ and $SAT_D(\phi, \Gamma)$ respectively with similar synchronisation steps.

Procedure 11 $P\_SAT_E(\phi, \Gamma, k, S_k)$ for $E_\Gamma\phi$.

$X_k \Leftarrow P\_SAT(\neg\phi)$; $X \Leftarrow \emptyset$;
for $j = 1$ to m **do** $X \Leftarrow X \cup X_j$; **end for**
$Y \Leftarrow \{s \in S_k \mid \exists s' \in X$ such that $\exists i \in \Gamma, s \sim_i s'\}$
return $\neg Y \cap S_k$;

Procedure 12 $P\_SAT_D(\phi, \Gamma, k, G_k)$ for $D_\Gamma\phi$

$X_k \Leftarrow P\_SAT(\neg\phi)$; $X \Leftarrow \emptyset$;
for $j = 1$ to m **do** $X \Leftarrow X \cup X_j$; **end for**
$Y \Leftarrow \{s \in G_k \mid \exists s' \in X$ such that $\forall i \in \Gamma, s \sim_i s'\}$
return $\neg Y \cap G_k$;

The parallel procedure for $C_\Gamma\phi$, reported below, is more complex because of the fix point computation. Observe that in P_SAT_C we need to compute a double fix point. In fact, each PP k calculates set Y_k of states in which ϕ is not satisfied and broadcasts it to other PPs. Following this, and given S_k and $Y = \bigcup_{j=1}^{m} Y_j$, each PP k computes the set of states $Y'_k \subseteq S_k$ in which $C_\Gamma\phi$ is not satisfied. If there exists a PP k ($1 \leq k \leq m$) such that $Y_k \neq Y'_k$, then all PPs assign Y'_k to Y_k, rebroadcast Y_k, and re-compute Y'_k. This iteration is repeated until $Y_k = Y'_k$ for all $1 \leq k \leq m$. Since we only deal with systems with finite states, $P\_SAT_C(\phi, \Gamma, k, S_k)$ eventually terminates.

Procedure 13 $P\_SAT_C(\phi, \Gamma, k, S_k)$ for $C_\Gamma\phi$

$Y_k \Leftarrow P\_SAT(\neg\phi)$;
repeat
  $Y \Leftarrow \emptyset$; $F_k \Leftarrow$ FALSE;
  for $j = 1$ to m **do** $Y \Leftarrow Y \cup Y_j$; **end for**
  repeat
    $X \Leftarrow Y$;
    $Y \Leftarrow \{s \in S_k \mid \exists s' \in X$ and $\exists i \in \Gamma$ such that $s \sim_i s'\}$;
  until $X = Y$
  if $Y_k = Y$ **then** $F_k \Leftarrow$ TRUE; **else** $Y_k \Leftarrow Y$; **end if**
  $F \Leftarrow$ TRUE;
  for $j = 1$ to m **do** $F \Leftarrow F \wedge F_j$; **end for**
until $F =$ TRUE
return $\neg Y \cap S_k$;

The parallel Procedures 8, 9, and 10 are integrated into Procedure 7 thereby defining a parallel approach to verifying CTLK formulae on IS models. It can be shown by induction that all $P\_CHECK(\mathcal{M}, \phi)$ return the correct set of states.

Theorem 1 (Soundness and completeness)**.** *Given a CTLK formula ϕ, m partitions of initial states $I_1, \ldots, I_m$ and corresponding sets of reachable states $S_1, \ldots, S_m$, we have that $P\_CHECK(\mathcal{M}, \phi)$ if and only if $CHECK(\mathcal{M}, \phi)$.*

Proof. (Sketch) by induction on syntax of ϕ. □

Efficiency considerations. We now pursue different optimisation strategies that will be analysed experimentally in the next section. Observe that the parallel procedure above is as slow as the slowest PP. This is because of the communication required among PPs. It is a priori not trivial to identify a partitioning of the initial states so that all PPs share a similar load. Additionally, since in any implementation the computations above are performed on OBDDs, the variable reordering mechanisms make any prediction even harder.

In an attempt to distribute evenly the workload to the various PPs, we can partition I in a number of sets greater than the number of processes available. In this way the various PPs can perform their respective computations and can, when finished, move to the next partition. Several strategies are possible here. We can try to explore as many reachable states as possible, or attempt to run the check for satisfaction of the formula in question. The procedure $MERGE\_PARA(k, \phi, sp)$ below adopts the former line. Here, after a successful computation of the reachable state space from a partition, the next unexplored set of initial states is iteratively computed, before performing the check for the formula in question. $MERGE\_PARA(k, \phi, sp)$ is illustrated in Procedure 14 below where sp is a global pointer to the next available partition and $\overline{m}$ the number of parallel processes. Note that I'_k and S'_k are the initial states and reachable states of the submodel $\mathcal{M}'_k$ respectively.

Procedure 14 $MERGE\_PARA(k, \phi, sp)$

$S'_k \Leftarrow \emptyset$; $I'_k \Leftarrow \emptyset$;
repeat
  if $sp \leq m$ **then** $j \Leftarrow sp$; $sp \Leftarrow sp + 1$; **end if**
  $I'_k \Leftarrow I'_k \cup I_j$; $S'_k \Leftarrow S'_k \cup REACH(I_j)$;
until $sp > m$
$CHECK_p(\mathcal{M}'_k, \phi)$;

The procedure $SIMPLE\_PARA(k, \phi, sp)$ is a special case of $MERGE\_PARA(k, \phi, sp)$ such that $m = \overline{m}$.

In many cases, especially when the length of the formula ϕ is short, the time to generate the state space is predominant in the overall model checking time. However, the time spent performing P_SAT_ϕ is at times non-negligible. In some of these cases P_SAT_ϕ runs faster on a number of small OBDDs than on a single large one, even taking into count the extra synchronisations needed. The procedure $FULL\_PARA(k, \phi, sp)$, reported below, is an extension of $MERGE\_PARA$ where sets of reachable states are not merged, in an attempt to exploit the considerations above. Note that I^t_k and S^t_k are the initial and reachable states of the submodel $\mathcal{M}^t_k$ respectively.

Procedure 15 $FULL\_PARA(k, \phi, sp)$

$t \Leftarrow 0$;
repeat
  if $sp \leq m$ **then** $j \Leftarrow sp$; $sp \Leftarrow sp + 1$; **end if**
  $t \Leftarrow t + 1$; $I^t_k \Leftarrow I_j$; $S^t_k \Leftarrow REACH(I_j)$;
until $sp > m$
for $j = 1$ to t **do** $CHECK_p(\mathcal{M}^j_k, \phi)$; **end for**

Lastly, in order to demonstrate the impact of model checking procedures on OBDDs of different sizes, we also explore a final procedure, that we call $SEMI\_PARA(k, \phi)$. Procedure 16 is a simplification of $SIMPLE\_PARA(k, \phi)$. In $SEMI\_PARA(k, \phi)$, PP 1 collects S_k from all other PPs, and then constructs the set S. Then it executes the sequential model checking procedure $CHECK(\mathcal{M}, \phi)$. Other PPs terminate when they send their S_k to PP 1.

Procedure 16 $SEMI\_PARA(k, \phi)$

1: $S_k \Leftarrow REACH(I_k)$;
2: **if** $k = 1$ **then**
3: $S \Leftarrow \emptyset$;
4: **for** $j = 1$ to m **do** $S \Leftarrow S \cup S_j$; **end for**
5: $CHECK(\mathcal{M}, \phi)$;
6: **end if**

We analyse the performance of these variants below.

5 Experiments

We implemented the different model checking algorithms presented in Section 4 on top of MCMAS [9], an open-source model checker for temporal-epistemic logic. MCMAS was the natural choice as it already supports the semantics of Interpreted Systems, CTLK specification languages, and performs OBDD operations by means of the efficient CUDD library [17]. In a MCMAS model, each agent has a set of local variables and a local state is an evaluation of these variables. A global state is an evaluation over all variables in the system. The set of initial states is specified by a Boolean expression over variables, i.e., any global state that satisfies the expression is an initial state.

To allow parallel model checking in a model, we only need to reorganise the expression for the initial states. The new expression is of the form $e_s \wedge (\bigvee_{j=1}^{m} e_j)$. Any global states satisfying $e_s \wedge e_k$ is in partition I_k ($1 \leq k \leq m$) for $FULL\_PARA$ and $MERGE\_PARA$. For $SIMPLE\_PARA$ and $SEMI\_PARA$, partition I_k is constructed as $e_s \wedge (e_{(k-1)*(\delta_2+1)+1} \vee \ldots \vee e_{k*(\delta_2+1)})$ for $1 \leq k \leq \delta_1$, or $e_s \wedge (e_{(k-1)*\delta_2+\delta_1+1} \vee \ldots \vee e_{k*\delta_2+\delta_1})$ for $\delta_1 < k \leq \overline{m}$ where $\delta_1 = m \bmod \overline{m}$ and $\delta_2 = \lfloor m/\overline{m} \rfloor$.

In order to provide a thorough assessment we tested our implementation on four examples already used with MCMAS. These are: the dining cryptographers scenario [2], the card games [5] example, the NSPK protocol [12], and the muddy children puzzle [6]. We refer to the cited publications and MCMAS's documentation for more details. The experiments were performed on an AMD Phenom(tm) 9600B Quad-Core Processor with 8GB memory running Fedora 12 x86_64 Linux (kernel 2.6.31.5-127). Four parallel threads were generated for all the examples.

We found that in all examples the overall memory consumption was often three or four times higher than that in the publicly available MCMAS. This was entirely expected as each thread creates an independent BDD manager. The experiments were meant to check whether we can perform checks faster than on a single core. The tables below report the running time (in seconds) and memory (in MBs) for the examples discussed. Note that Seq represents sequential model checking procedure, and Semi (Simple, Merge and Full respectively) represents SEMI_PARA(SIMPLE_PARA, MERGE_PARA and FULL_PARA respectively).

Dining cryptographers [2]. In this example, we checked the following common knowledge formula specification

$$AG(even \rightarrow C_\Gamma(\bigwedge \neg paid_i)),$$

where $even$ represents an even number of cryptographers claiming that the two coins fell on the same side and $paid_i$ represents that the bill was paid by the i-th cryptographer. The set of initial states was split into $N+1$ partitions for MERGE- and FULL-PARA. We found the following results.

Table 1. Verification results for the dining cryptographers scenario.

N	States	Seq	Semi	Simple	Merge	Full
10	45056	21s	11s	12s	6s	5s
		20MB	67MB	69MB	60MB	58MB
14	9.8×10^5	128s	26s	56s	28s	15s
		51MB	91MB	99MB	83MB	84MB
18	2.0×10^7	160s	149s	186s	21s	48s
		55MB	174MB	159MB	82MB	96MB
22	3.9×10^8	2098s	6783s	6622s	85s	85s
		127MB	357MB	353MB	126MB	149MB
26	7.2×10^9	365s	161s	184s	58s	55s
		58MB	176MB	170MB	117MB	164MB
30	1.3×10^{11}	2823s	12771s	12009s	160s	496s
		105MB	427MB	412MB	176MB	205MB

Card games [5]. Here we used the formula presented in [5] for our tests:

$$AG\ (allred1 \rightarrow K_{player1}(AF\ win1)),$$

The specification states that it is always the case that if player 1 has only red cards, then he knows that eventually he will win the game. The initial states are partitioned based on the possible choices of each player's first card. The number of partitions is $(N-1)N$, where N is the number of total cards.

Table 2. Verification results for the card games example.

N	States	Seq	Semi	Simple	Merge	Full
8	8×10^4	1s	1s	1s	1s	1s
		13MB	48MB	48MB	54MB	54MB
10	7.4×10^8	62s	32s	33s	34s	22s
		48MB	150MB	140MB	215MB	213MB
12	$2,9\times10^{10}$	51242s	15160s	13974s	3569s	3960s
		1.4GB	2.5GB	2.2GB	1.9GB	1.9GB

NSPK protocol [12]. In this example we ran experiments on $n \in \{2,3,4\}$ number of agents, respectively, together with $(n+1)n$ partitions. The CTLK formulae verified in each case are listed below.

2. $AG(i2\_end \rightarrow K_{i2}\ agree\_i2\_i1)$.
3. $AG(i3\_end \rightarrow K_{i3}(agree\_i3\_i1 \vee agree\_i3\_i2))$.
4. $AG(i3\_end \rightarrow K_{i3}(agree\_i3\_i1 \vee agree\_i3\_i2))$.
   $AG(i4\_end \rightarrow K_{i4}(agree\_i4\_i1 \vee agree\_i4\_i2))$.

The first formula says that whenever agent 2 terminates, he knows that he and agent 3 agree on the protocol variables. The second one specifies that globally when agent 3 terminates, he knows that he agrees with either agent 1 or agent 2 on the protocol variables.

Muddy children [6]. The formula verified on this example is

$$AG(((K_{child1} muddy1) \vee (K_{child1} \neg muddy1)) \rightarrow saysknows1),$$

specifying that whenever child 1 knows whether or not he has muddy forehead, he will announce that he knows this. The initial states were partitioned into 8 disjunctive groups.

The results above demonstrate that the parallel algorithms offer good performance in the first three examples. The verification

Table 3. Verification results for the NSPK protocol.

N	States	Seq	Semi	Simple	Merge	Full
2	618	1s	1s	1s	1s	1s
		12MB	44MB	43MB	43MB	43MB
3	42240	7s	3s	2s	3s	2s
		50MB	170MB	159MB	143MB	143MB
4	2.9×10^6	391s	161s	134s	155s	156s
		647MB	1373MB	992MB	961MB	717MB

Table 4. Verification results for the muddy children puzzle.

n	States	Seq	Semi	Simple	Merge	Full
20	3.4×10^7	7s	8s	8s	8s	7s
		24MB	87MB	82MB	83MB	86MB
30	3.4×10^{10}	52s	75s	72s	66s	70s
		36MB	145MB	130MB	144MB	159MB
40	7.0×10^{13}	272s	227s	234s	285s	285s
		58MB	224MB	219MB	244MB	231MB
50	7.2×10^{16}	585s	1003s	1077s	1091s	1102s
		61MB	266MB	241MB	297MB	253MB
60	7.4×10^{19}	30521s	2318s	2167s	2216s	2295s
		87MB	490MB	311MB	342MB	352MB

time was reduced dramatically with significant gains being shown on bigger models. Generally speaking, the experimental results point to the fact that smaller size partitions can speed up the computation more, even with the same number of physical processor cores. Strong indications of this were given by the speed gained by the MERGE- and FULL-PARA algorithms. While some differences exist, the speed difference between SEMI- and SIMPLE-PARA, and between MERGE- and FULL-PARA, can be small.

Our parallel algorithms failed to accelerate the verification on half of the muddy children models. The biggest gain on this example was obtained when the model became very large (the last one in Table 4). We suspect this situation is caused by the regularity of the underlying OBDD structure; this is the only case we found where the speed advantage in the algorithms did not compensate for the communication overhead between the processes.

6 Conclusions

In this paper we have defined and explored a number of parallel model checking algorithms for the verification of multi-agent systems. These algorithms require only limited synchronisations among parallel processes/threads to evaluate epistemic operators, and leave the interpretation of CTL operators as it is in the sequential approach. The experimental results not only demonstrate the effectiveness of algorithms in a number of cases, but also suggest that more partitions of the set of initial states usually lead to shorter verification time. This is promising in view of the fact that the number of cores available in CPUs is expected to grow significantly, perhaps exponentially, in the years ahead.

There are many directions for future work. We are not satisfied with the performance of the algorithms on the muddy children example; a deeper investigation on the underlying OBDD structures is required to appreciate the result fully. It is also of interest to explore how the initial partitioning can affect the performance of the parallel model checking algorithms.

Acknowledgements

The first and third authors are partly supported by the European Commission FP 7 project CONNECT (IST Project Number 231167).

REFERENCES

[1] R. E. Bryant, 'Graph-based algorithms for boolean function manipulation', *IEEE Transactions on Computers*, **35**(8), 677–691, (1986).

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

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

[4] M. Cohen, M. Dam, A. Lomuscio, and H. Qu, 'A symmetry reduction technique for model checking temporal-epistemic logic', in *the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pp. 721–726, (2009).

[5] M. Cohen, M. Dam, A. Lomuscio, and F. Russo, 'Abstraction in model checking multi-agent systems', in *8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, pp. 945–952. IFAAMAS, (2009).

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

[7] 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*, pp. 479–483. Springer, (2004).

[8] O. Grumberg, T. Heyman, and A. Schuster, 'Distributed symbolic model checking for μ-calculus', *Formal Methods in System Design*, **26**(2), 197–219, (2005).

[9] A. Lomuscio, H. Qu, and F. Raimondi, 'MCMAS: A model checker for the verification of multi-agent systems.', in *Proceedings of CAV 2009*, LNCS 5643, pp. 682–688. Springer, (2009).

[10] A. Lomuscio, H. Qu, and M. Solanki, 'Towards verifying compliance in agent-based web service compositions', in *Proceedings of The Seventh International Joint Conference on Autonomous Agents and Multi-agent systems (AAMAS-08)*. IFAAMAS, (2008).

[11] A. Lomuscio and F. Raimondi, 'MCMAS: A model checker for multi-agent systems.', in *Proceedings of TACAS 2006*, volume 3920, pp. 450–454. Springer, (2006).

[12] R. M. Needham and M. D. Schroeder, 'Using encryption for authentication in large networks of computers', *Communications of the ACM*, **21**(12), 993–999, (1978).

[13] R. Parikh and R. Ramanujam, 'Distributed processes and the logic of knowledge', in *Logic of Programs*, volume 193 of *Lecture Notes in Computer Science*, pp. 256–268. Springer, (1985).

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

[15] 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).

[16] D. Sahoo, J. Jain, S. K. Iyer, and D. L. Dill, 'A new reachability algorithm for symmetric multi-processor architecture', in *the 3th International Symposium on Automated Technology for Verification and Analysis (ATVA 2005)*, LNCS 3707, pp. 26–38. Springer, (2005).

[17] F. Somenzi. CUDD: CU decision diagram package - release 2.4.2. `http://vlsi.colorado.edu/~fabio/CUDD/cuddIntro.html`, 2009.