Partial order reduction for model checking interleaved multi-agent systems

Alessio Lomuscio[1] and Wojciech Penczek[2] and Hongyang Qu[3]

[1] Department of Computing, Imperial College London, UK
[2] Institute of Computer Science, PAS, and University of Podlasie, Poland
[3] Oxford University Computing Laboratory, Parks Road, Oxford OX1 3QD, UK

Abstract. We investigate partial order reduction for model checking of multi-agent systems by focusing on interleaved interpreted systems. These are a particular class of interpreted systems, a mainstream MAS formalism, in which only one action at the time is performed in the system. We present a notion of stuttering-equivalence, and prove the semantical equivalence of stuttering-equivalent traces with respect to LTLK$_{-X}$, a linear temporal logic for knowledge without the next operator. We give an algorithm to reduce the size of the models before the model checking step and show it preserves LTLK$_{-X}$properties. We evaluate the technique by discussing an implementation and the experimental results obtained against well-known examples in the MAS literature.

1 Introduction

Several approaches have been put forward for the verification of MAS by means of model checking [3]. Some approaches are based on reducing the verification problem to the one of plain temporal logic and use existing tools for that task [1]. Others treat typical MAS modalities such as knowledge, correctness, cooperation, as first-class citizens and introduce novel algorithms for them, e.g., [17]. In an attempt to limit the state-space explosion problem (i.e., the problem that the state space of the system grows exponentially with the number of variables in the agents) two main symbolic approaches have been proposed: ordered binary decision diagrams [17, 16], and bounded model checking via propositional satisfiability [14]. These have produced positive results showing the ability to tackle state spaces of 10^{30} and above. However, in the standard literature of model checking reactive systems other, sometimes more efficient, approaches exists.

In particular, *partial order reduction* (e.g., [13]) is one of the most widely known techniques in verification of reactive systems. Still, the only approach to partial order reduction in a MAS context [9] presents theoretical results only, with no algorithm nor an implementation being discussed; as such it is difficult to assess how effective it is in concrete cases. In our experience, given their autonomous nature, MAS differ from standard reactive systems by displaying more "loosely coupled" behaviours. This makes the state-explosion problem even more challenging for MAS than it is already for reactive systems. It seems therefore of importance to conduct a systematic and comparative study of all possible techniques available to determine the most appropriate treatment to the verification problem.

In this paper we aim to make concrete progress in this area by studying a particular class of interpreted systems that we call *interleaved interpreted systems* (IIS). IIS are a special class of interpreted systems [5] in which only one action at a time is performed in a global transition. Several agents may be participating in the global action but, if so, they perform the same action, thereby synchronising at that particular time step. Many asynchronous reactive systems have been studied on similar semantics (see, e.g., [11, 6]) and it is easy to see that asynchronous MAS may be modelled within the formalism presented below.

In a nutshell, given a model M_S (representing a system S) and a formula ϕ_P (representing a specification property P to be checked) in the temporal logic LTL_{-X}(the linear temporal logic LTL without the neXt operator X), model checking via partial order reduction suggests to compute $M_S \models \phi_P$ by replacing M_S with a smaller model M'_S built on traces that are semantically equivalent (with respect to ϕ_P) to the ones of M_S. Of key importance in this line of work is not only to determine a notion of equivalence but also to present algorithms that can transform (in polynomial time) M_S into a suitable M'_S even without generating M_S. The literature of reactive systems has shown that in several scenarios this reduction can be very effective and brings results comparable or superior to the ones of other techniques including ordered-binary decision diagrams.

In this paper we draw inspiration from the above to conduct a similar exercise in the context of MAS logics. We begin in Section 2 by presenting IIS and the logic CTL*K_{-X}, and, in particular, LTLK_{-X}, a linear temporal logic for knowledge without the next operator. In Section 3 we proceed to present a notion of stuttering-equivalence with respect to IIS. We move on to describe a novel partial order algorithm that preserves LTLK_{-X}properties in Section 4. In Section 5 we present an implementation of the technique and report key experimental results. We conclude the paper in Section 6.

2 Preliminaries

We introduce here the basic technical background to the present paper. In particular we introduce the semantics of interpreted systems, properly augmented with suitable concepts for our needs, and the basic syntax we shall be using in the rest of the paper.

2.1 Interleaved Interpreted Systems

The semantics of *interpreted systems* provides a setting to reason about MAS by means of specifications based on knowledge and linear time. We report here the basic setting as popularised in [5]. Actions in interpreted systems are typically considered to be executed at the same round by all participants: this permits the modelling of synchronous systems in a natural way. While interpreted systems are typically considered in their synchronous variant here we look at the asynchronous case by assuming that only one local action may be performed at a given time in a global state. If more than one agent is active at a given round, all active agents perform the same (shared) action in the round. Differently from standard interpreted systems where, in principle, the agents' resulting local states depend on the actions performed by all the agents in the system, here we

assume the local states are only influenced by the same agent's action at the previous round. Note that it is still possible for agents to communicate by means of shared action. More formally, we proceed as follows:

We begin by assuming a MAS to be composed of n agents $\mathcal{A} = \{1, \ldots, n\}$[4]. We associate a set of *possible local states* $L_i = \{l_i^1, l_i^2, \ldots, l_i^{nl_i}\}$ and *actions* $Act_i = \{\epsilon_i, a_i^1, a_i^2, \ldots, a_i^{na_i}\}$ to each agent $i \in \mathcal{A}$. We call the special action ϵ_i the "null", or "silent" action of agent i; as it will be clear below the local state of agent i remains the same if the null action is performed. Also note we do not assume that the sets of actions of agents to be disjoint. We call $Act = \bigcup_{i \in \mathcal{A}} Act_i$ the union of all the sets Act_i. For each action a by $Agent(a) \subseteq A$ we mean all the agents i such that $a \in Act_i$, i.e., the set of agents potentially able to perform a.

Following closely the interpreted system model, we consider a *local protocol* modelling the program the agent is executing. Formally, for any agent i, the actions of the agents are selected according to a *local protocol* $P_i : L_i \rightarrow 2^{Act_i}$; we assume that $\epsilon \in P_i(l_i^m)$, for any l_i^m; in other words we insist on the null action to be enabled at every local state. For each agent i, we define an evolution (partial) function $t_i : L_i \times Act_i \rightarrow L_i$, where $t_i(l_i, \epsilon_i) = l_i$ for each $l_i \in L_i$. Note the local transition function considered here differs from the standard treatment in interpreted systems by depending only on the local action in question.

A *global state* $g = (l_1, \ldots, l_n)$ is a tuple of local states for all the agents in the MAS corresponding to an instantaneous snapshot of the system at a given time. Given a global state $g = (l_1, \ldots, l_n)$, we denote by $g^i = l_i$ the local component of agent $i \in \mathcal{A}$ in g. Given the notions above we can now define formally the global transitions we consider in this paper.

Definition 1 (Interleaved semantics). *Let G be a set of global states. The global interleaved evolution function $t : G \times Act_1 \times \cdots \times Act_n \rightarrow G$ is defined as follows: $t(g, act_1, \ldots, act_n) = g'$ iff there exists an action $a \in Act$ such that for all $i \in Agent(a)$, $act_i = a$ and $t_i(g^i, a) = g'^i$, and for all $i \in \mathcal{A} \setminus Agent(a)$, $act_i = \epsilon_i$ and $t_i(g^i, act_i) = g^i$. In brief we write the above as $g \xrightarrow{a} g'$.*

Similar to blocking synchronisation in automata, the above insists on all agents performing the same action in a global transition; additionally note that if an agent has the action being performed in its repertoire it must be performed for the global transition to be allowed. This assumes local protocols are defined in such a way to permit this; if a local protocol does not permit this, the local action cannot be performed and therefore the global transition does not comply with the definition of interleaving above. As we formally clarify below we only consider interleaved transitions here.

We assume that the global transition relation is total, i.e., that for any $g \in G$ there exists an $a \in Act$ such that $g \xrightarrow{a} g'$, for some $g' \in G$. A sequence of global states and actions $\pi = g_0 a_0 g_1 a_1 g_2 \ldots$ is called an interleaved path, or an interleaved run (or more simply a path or a run) originating at g_0 if there is a sequence of interleaved transitions from g_0 onwards, i.e., if $g_i \xrightarrow{a_i} g_{i+1}$ for every $i \geq 0$. The set of interleaved

[4] Note in the present study we do not consider the environment component. This may be added with no technical difficulty at the price of heavier notation.

paths originating from g is denoted as $\Pi(g)$. A state g is said to be *reachable* from g_0 if there is an interleaved path $\pi = g_0 a_0 g_1 a_1 g_2 \ldots$ such that $g = g_i$ for some $i \geq 0$.

Definition 2 (Interleaved Interpreted Systems). *Given a set of propositions PV, an interleaved interpreted system (IIS), also referred to as a model, is a 4-tuple $M = (G, \iota, \Pi, V)$, where G is a set of global states, $\iota \in G$ is an initial (global) state such that each state in G is reachable from ι, $\Pi = \bigcup_{g \in G} \Pi(g)$ is the set of all the interleaved paths originating from all states in G, and $V : G \rightarrow 2^{PV}$ is a valuation function.*

Figure 1 presents an interleaved interpreted system (the untimed version of the original Train-Gate-Controller (TGC) example in [15]) composed of three agents: a controller and two trains. Each train runs on a circular track and both tracks pass through a narrow tunnel, allowing one train only to go through it at any time. The controller controls the signal to let trains enter and leave the tunnel. In the figure, the initial states of the controller and the train are 'G' and 'W' respectively, and the transitions with the same label are synchronised. ϵ actions are omitted in the figure.

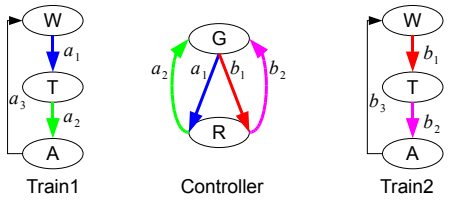

Fig. 1. An IIS of TGC composed of two trains

In order to define partial order reductions we need the following definitions.

Definition 3. *Let $i \in \mathcal{A}$, $g, g' \in G$, and $J \subseteq \mathcal{A}$.*

- $\sim_i \subseteq G \times G$ *is defined as:* $g \sim_i g'$ *iff* $g^i = g'^i$.
- $\sim_J = \bigcap_{j \in J} \sim_j$.
- $I \subseteq Act \times Act$ *is defined as follows:* $I = \{(a, b) \mid Agent(a) \cap Agent(b) = \emptyset\}$.

The first relation ($\sim_i$) is normally associated with the indistinguishably relation for the epistemic modality (see below), the second ($\sim_J$) corresponds to the indistinguishably relation for the epistemic modality of distributed knowledge in group J, whereas the third (I) is referred to as the independency relation in partial order approaches.

We say that two actions a, a' are dependent if $(a, a') \notin I$.

Definition 4 (Reduced Model). *Consider two models $M = (G, \iota, \Pi, V)$, $M' = (G', \iota', \Pi', V')$. If $G' \subseteq G$, $\iota' = \iota$ and $V' = V|G'$, then we write $M' \subseteq M$ and say that M' is a submodel of M, or that M' is a* reduced *model of M.*

We now define the syntax and semantics of our language.

2.2 Syntax of CTL*K$_{-X}$

Combinations of linear/branching time and knowledge have long been used in the analysis of temporal epistemic properties of systems [5, 7]. We recall the basic definitions here and adapt them to our purposes when needed.

Let PV be a finite set of propositions. First, we give a syntax of CTL*K$_{-X}$ and then restrict it to LTLK$_{-X}$ and other sublanguages. The state and path formulas of CTL*K$_{-X}$ are defined inductively as follows:

S1. every member of PV is a state formula,
S2. if φ and ψ are state formulas, then so are $\neg\varphi$ and $\varphi \wedge \psi$,
S3. if φ is a path formula, then $A\varphi$, $E\varphi$, and $K_i\varphi$ ($i \in \mathcal{A}$) are state formulas,
P1. any state formula φ is also a path formula,
P2. if φ, ψ are path formulas, then so are $\varphi \wedge \psi$ and $\neg\varphi$,
P3. if φ, ψ are path formulas, then so is $U(\varphi, \psi)$.

The modal operator A has the intuitive meaning "for all paths" whereas E - "there is a path". The operator U denotes the standard Until modality. K_i denotes knowledge of agent i: $K_i\phi$ is read as "agent i knows that ϕ". CTL*K$_{-X}$ consists of the set of all state formulae. The following abbreviations will be used: $true \stackrel{def}{=} \neg(p \wedge \neg p)$, for some $p \in PV$, $F\varphi \stackrel{def}{=} U(true, \varphi)$, $G\varphi \stackrel{def}{=} \neg F\neg\varphi$. As standard, F represents the temporal operator of "eventually" (in the future) and G corresponds to "forever" (in the future). We now define a variety of logics included in CTL*K$_{-X}$.

Definition 5.

- *LTLK$_{-X}$ $\subset$ CTL*K$_{-X}$ is the fragment of CTL*K$_{-X}$ in which all modal formulas are of the form $A\varphi$, where φ does not contain the state modalities A, E. As customary, we usually write φ instead of $A\varphi$ if confusion is unlikely.*
- *CTLK$_{-X}$ $\subset$ CTL*K$_{-X}$ is the fragment of CTL*K$_{-X}$ in which the state modalities A, E, and the path modalities U, F and G may only appear paired in the combinations AU, EU, AF, EF, AG, and EG.*
- *For any logic L and $J \subseteq \mathcal{A}$, we write L^J for the restriction of the logic L such that for each subformula $K_i\varphi$ we have $i \in J$.*

Next, we give the semantics to the logics used in this paper.

2.3 Semantics of CTL*K$_{-X}$

Let $M = (G, \iota, \Pi, V)$ be a model and let $\pi = g_0a_0g_1\cdots$ be an infinite path of G. Let π_i denote the suffix $g_ia_ig_{i+1}\cdots$ of π and $\pi(i)$ denote the state g_i. Satisfaction of a formula φ in a state g of M, written $(M, g) \models \varphi$, or just $g \models \varphi$, is defined inductively as follows:

S1. $g \models q$ iff $q \in V(g)$, for $q \in PV$,
S2. $g \models \neg\varphi$ iff not $g \models \varphi$,
$g \models \varphi \wedge \psi$ iff $g \models \varphi$ and $g \models \psi$,

S3. $g \models A\varphi$ iff $\pi \models \varphi$ for every path π starting at g,
$g \models E\varphi$ iff $\pi \models \varphi$ for some path π starting at g,
$g \models K_i\varphi$ iff $g' \models \varphi$ for every $g' \in G$ such that $g \sim_i g'$,
P1. $\pi \models \varphi$ iff $g_0 \models \varphi$ for any state formula φ,
P2. $\pi \models \neg\varphi$ iff not $\pi \models \varphi$,
$\pi \models \varphi \wedge \psi$ iff $\pi \models \varphi$ and $\pi \models \psi$,
P3. $\pi \models U(\varphi, \psi)$ iff there is an $i \geq 0$ such that $\pi_i \models \psi$ and $\pi_j \models \varphi$ for all $0 \leq j < i$.

3 Characterisation of LTLK$^{J}_{-X}$

Having now fixed the syntax and semantics on which to operate we now proceed to give a notion of behavioural equivalence, and to show this is preserved under the particular algorithm we introduce in the next section. To begin with we define a notion of action *invisibility*.

Definition 6. *An action $a \in Act$ is* invisible *in a model (G, ι, Π, V) if whenever $g \xrightarrow{a} g'$ for any two states $g, g' \in G$ we have that $V(g) = V(g')$.*

An action $a \in Act$ is J-invisible *in a model (G, ι, Π, V) if whenever $g \xrightarrow{a} g'$ for any two states $g, g' \in G$ we have that $V(g) = V(g')$ and $g \sim_J g'$.*

In other words, an action is invisible if its execution does not change the global valuation. An action is J-invisible if it is invisible and all local states in J are not changed by its execution (recall that all local states in $\mathcal{A} \setminus Agent(a)$ are not changed in the transition labelled with a either).

We denote the set of invisible (respectively, J-invisible) actions by $Invis$ ($Invis_J$, respectively), and we write $Vis = Act \setminus Invis$ (respectively, $Vis_J = Act \setminus Invis_J$) for the set of *visible* actions (*(J-)visible* actions, respectively).

Definition 7. *Let $\pi = g_0 a_0 g_1 a_1 \cdots$ be a (finite or infinite) path in a model M and $J \subseteq \mathcal{A}$. We define the* J-stuttering-free *projection $Pr_J(\pi)$ of a path π inductively as follows:*

- $Pr_J(g_0) = g_0$;
- $Pr_J(g_0 \cdots g_i) = Pr_J(g_0 \cdots g_{i-1})$ *if* $V(g_{i-1}) = V(g_i)$ *and* $g_{i-1} \sim_J g_i$;
  $Pr_J(g_0 \ldots g_i) = Pr_J(g_0 \ldots g_{i-1}) g_i$ *otherwise.*

Let $M = (G, \iota, \Pi, V)$ and $M' = (G', \iota', \Pi', V')$ be two models such that $M' \subseteq M$. In the following, we begin with the definition of *J-stuttering* among states. Then, we define *stuttering equivalence* of two paths $\pi, \pi' \in \Pi$ and extend it to *J-stuttering equivalence*. Finally, we present the notion of *J-stuttering trace equivalence* over states.

Definition 8 (J-stuttering of States). *Two states $g \in G$ and $g' \in G'$ are* J-stuttering*, denoted with $JKS(g, g')$, if $V(g) = V'(g')$ and $g \sim_J g'$.*

Definition 9 (Stuttering Equivalence). *A path $\pi = g_0 a_0 g_1 a_1 \cdots$ in M and a path $\pi' = g'_0 a'_0 g'_1 a'_1 \cdots$ in M' are called* stuttering equivalent*, denoted $\pi \equiv_s \pi'$, if there exists a partition $B_1, B_2 \ldots$ of the states of π, and a partition $B'_1, B'_2 \ldots$ of the states of π' such that for each $j \geq 0$ we have that B_j and B'_j are nonempty and finite, and for every state g in B_j and every state g' in B'_j we have $V(g) = V'(g')$.*

Definition 10 (J-stuttering Equivalence). *Two paths π in M and π' in M' are called* J-stuttering equivalent*, denoted $\pi \equiv_{Jks} \pi'$, if $\pi \equiv_s \pi'$ and for each $j \geq 0$ and for every state g in B_j and every state g' in B'_j we have $g \sim_J g'$,*

Definition 11 (J-stuttering Trace Equivalence). *Two states g in M and g' in M' are said to be* J-stuttering trace equivalent*, denoted $g \equiv_{Jks} g'$, if*

1. *for each infinite path π in M starting at g, there is an infinite path π' in M' starting at g' such that $\pi \equiv_{Jks} \pi'$;*
2. *for each infinite path π' in M' starting at g', there is an infinite path π in M starting at g such that $\pi' \equiv_{Jks} \pi$.*

Two models M and M' are J-stuttering trace equivalent *($M \equiv_{Jks} M'$), if $\iota \equiv_{Jks} \iota'$.*

The following theorem connects LTLK$^J_{-X}$ with J-stuttering trace equivalence:

Theorem 1. *Let M and M' be two J-stuttering trace equivalent models, where $M' \subseteq M$. Then, $M, \iota \models \varphi$ iff $M', \iota' \models \varphi$, for any LTLK$^J_{-X}$ formula φ over PV.*

Proof. By induction on the structure of φ. Due to the page limit, the proof is omitted.

This concludes our analysis of stuttering equivalent paths. We now give an algorithm that assures that given a model and a formula returns a potentially smaller model and show by means of the theorem above that the reduced model preserves the formula.

4 Partial order reductions

As mentioned above, the idea of verification by model checking with partial order reduction is to implement an algorithm that given a model can produce a smaller model that provably validates the same formulae of interest. This generation requires a notion of equivalence between paths and models. For the case of LTLK$^J_{-X}$ we show below that the notion of J-stuttering trace equivalence presented above suffices. The algorithm presented explores the given model and returns a reduced one. Traditionally, in partial order reduction the exploration is carried out either by depth-first-search (DFS) (see [6]), or double-depth-first-search (DDFS) [4].

In this context DFS is used to compute paths that will make up the reduced model by systematically exploring the possible computation tree and selecting only some of the possible paths generated. We proceed as follows. A stack represents the path $\pi = g_0 a_0 g_1 a_1 \cdots g_n$ currently being visited. For the top element of the stack, i.e., g_n, the following three operations are computed in a loop:

1. The set $en(g_n) \subseteq Act$ of enabled actions (not including the ϵ action) is identified and a subset $E(g_n) \subseteq en(g_n)$ of possible actions is heuristically selected.
2. For any action $a \in E(g_n)$ compute the successor state g' such that $g_n \xrightarrow{a} g'$, and add g' to the stack generating the path $\pi' = g_0 a_0 g_1 a_1 \cdots g_n a g'$. Recursively proceed to explore the submodel originating at g' in the same way by means of the present algorithm beginning at step 1.
3. Remove g_n from the stack.

The algorithm begins with a stack comprising of the initial state and terminates when the stack is empty. The model generated by the algorithm is a submodel of the original. The size of the submodel crucially depends on the ratio $E(g)/en(g)$. Clearly, if $E(g) = en(g)$ for all g explored there is no reduction, and the algorithm returns the whole model. The choice of $E(q)$ is constrained by the class of properties that must be preserved. In the rest of this section, we present the criteria based on the J-stuttering trace equivalence for the choice of $E(q)$ and give details of the DFS algorithm implementing them.

4.1 Preserving LTLK$^J_{-X}$ properties

In the sequel, let ϕ be a LTLK$^J_{-X}$formula to be checked over the model M with $J \subseteq \mathcal{A}$ such that for each subformula $K_i\varphi$ contained in ϕ, $i \in J$, and let M' be a submodel of M, generated by the algorithm. The states and the actions connecting states in M' construct a directed *state graph*. We give conditions defining a heuristics for the selection of $E(g)$ (such that $E(g) \neq en(g)$) while visiting state g in the algorithm below.

- **C1** No action $a \in Act \setminus E(g)$ that is dependent (see Definition 3) on an action in $E(g)$ can be executed before an action in $E(g)$ is executed.
- **C2** On every cycle in the constructed state graph there is at least one node g for which $E(g) = en(g)$, i.e., for which all the successors of g are expanded.
- **C3** All actions in $E(g)$ are invisible (see Definition 6).
- **CJ** For each action $a \in E(g)$, $Agent(a) \cap J = \emptyset$, i.e., no action in $E(g)$ changes local states of the agents in J.

The conditions $\mathbf{C1} - \mathbf{C3}$ are inspired from [12], whereas as we note below **CJ** is aimed at preserving the truth value of subformulae of the form $K_i\varphi$ for $i \in J$.

Theorem 2. *Let M be a model and $M' \subseteq M$ be the reduced model generated by the DFS algorithm described above in which the choice of $E(g')$ for $g' \in G'$ is given by* **C1, C2, C3, CJ** *above. The following conditions hold:*

a) M and M' are J-stuttering trace equivalent;
b) $M \models \phi$ iff $M' \models \phi$, for any $\phi \in$ LTLK$^J_{-X}$.

Proof. (Sketch) Although the setting is different the condition a) can be shown similarly to Theorem 3.11 in [13] stating that the conditions **C1, C2, C3** guarantee that the models M and M' are stuttering equivalent. Given condition **CJ** and $M' \subseteq M$, we can prove that the models M and M' are J-stuttering trace equivalent. The condition b) of the theorem follows from this and Theorem 1.

4.2 The DFS-POR algorithm

We now give details of a DFS algorithm implementing conditions **C1, C2, C3, CJ** for the choice of $E(g)$. We use two stacks: $Stack1$ represents the stack described above containing the global states to be expanded, whereas $Stack2$ represents additional information required to ensure condition **C2** is satisfied, i.e., each element in $Stack2$ is

the depth of $Stack1$ when its top element is fully explored. Initially, $Stack1$ contains the initial state, whereas $Stack2$ is empty. G is the set of the visited states. The algorithm DFS-POR does not generate the minimal J-stuttering equivalent model; however its computation overheads are negligible and, as we show in the section below, it still generates attractive results in several cases.

Algorithm 1 DFS-POR ()

```
1: g ⇐ Top(Stack1); reexplore ⇐ false;
2: if g = Element(Stack1, i) then
3:    depth ⇐ Top(Stack2);
4:    if i > depth then
5:       reexplore ⇐ true;
6:    else
7:       Pop(Stack1); return;
8:    end if
9: end if
10: if reexplore = false and g ∈ G then
11:    Pop(Stack1); return;
12: end if
13: G ⇐ G ∪ {g}; E(g) ⇐ ∅;
14: if en(g) ≠ ∅ then
15:    if reexplore = false then
16:       for all a ∈ en(g) do
17:          if a ∉ Vis and a ∉ Vis_J and ∀b ∈ en(g) \ {a} : (a, b) ∈ I then
18:             E(g) ⇐ {a}; break;
19:          end if
20:       end for
21:    end if
22:    if E(g) = ∅ then E(g) ⇐ en(g); end if
23:    if E(g) = en(g) then
24:       Push(Stack2, Depth(Stack1));
25:    end if
26:    for all a ∈ E(g) do
27:       g' ⇐ Successor(g, a); Push(Stack1, g'); DFS-POR();
28:    end for
29: end if
30: depth ⇐ Top(Stack2);
31: if depth = Depth(Stack1) then Pop(Stack2); end if
32: Pop(Stack1);
```

In the algorithm, the function $Top(s)$ returns the top element of the stack s; $Push(s, e)$ pushes the element e onto the top of the stack s; $Pop(s)$ removes the top element of the stack s; $Element(s, i)$ returns the i-th element of the stack s; $Depth(s)$ returns the depth (size) of the stack s; $Successor(g, a)$ returns the successor g' such that $g \xrightarrow{a} g'$.

Line 2 is used to detect a cycle. This can be implemented in the time complexity $\mathcal{O}(1)$ by using a hash table to index the state in $Stack1$. If a cycle is found, we check

whether at least one state is expanded fully in the cycle. This check is done in line 4 by comparing the top element of $Stack2$ and the index i of the repeated state in $Stack1$. If the check fails, we set $reexplore$ to true in order to fully expand the top state g in $Stack1$ to satisfy condition **C2**.

The lines 15-21 look for an action that is neither visible nor J-visible, and is independent of any other actions in $en(g)$. A set composed of such an action satisfies the conditions **C1, C3** and **CJ**. If no such action exists, we simply explore all enabled actions. This could be improved by searching for an appropriate subset of $en(g)$ to expand (e.g., [12] could be a starting point). In case $E(g) = en(g)$, we push the current depth of $Stack1$ onto the top of $Stack2$ for checking **C2**. When all actions in $E(g)$ are visited, we remove the top element of $Stack1$ and $Stack2$ properly.

We stress that DFS-POR is of linear complexity in the size of an IIS and the reduced model constructed.

5 Experimental Results

In order to evaluate the results above, we have implemented the DFS-POR algorithm to reduce models for LTLK^{J}_{-X} as well as the model checking algorithm for CTLK^{J}_{-X}. In doing so we are encouraged by the observation of the preceding section that the algorithm's complexity is linear both in the length of the formula and the size of a model. We have conducted experiments for two systems: the TGC of Section 2.1 and the Dining Cryptographers (DC) [2], discussed below.

Starting with TGC, we tested the property expressing that whenever the train 1 is in the tunnel, it knows that no other train is in the tunnel at the same time:

$$AG(\text{in\_tunnel}_1 \rightarrow K_{\text{train}_1} \bigwedge_{i=2}^{n} \neg \text{in\_tunnel}_i),$$

under the assumption that where n is the number of trains in the system, and the atomic proposition in_tunnel_i holds in the states where the train i is in the tunnel. We found that the size of the reduced state space $R(n)$ given by the algorithm is a function of the number of trains n, for $1 \leq n \leq 10$. This is compared to the size of the full state space $F(n)$ below:

- $F(n) = c_n \times 2^{n+1}$, for some $c_n > 1$,
- $R(n) = 3 + 4(n-1)$.

Note that the reduced state space is *exponentially smaller* than the original one.

As regards the DC scenario, we analysed a version with an arbitrary number of cryptographers. As in the original scenario [2], after all coins have been flipped each cryptographer observes whether the coins he can see fell on the same side or not. If he did not pay for dinner he states what he sees; if he did he states the opposite. Since our model is interleaved we assume the announcements are made in sequence; this does not affect the scenario. We used the algorithm to reduce the models preserving the protocol specification [8]:

$$AG((\text{odd} \wedge \neg \text{pay}_1) \rightarrow ((K_{\text{crypt}_1} \bigvee_{i=2}^{n} \text{pay}_i) \wedge (\bigwedge_{i=2}^{n} \neg K_{\text{crypt}_1} \text{pay}_i))),$$

where the atomic proposition 'odd' means that an odd number of announcements for different sides of the coins were paid, and the atomic proposition pay_i holds when cryptographer i is the payer. Table 1 displays the sizes of the full and reduced state spaces and the execution times (in seconds) on a machine running Linux Fedora 10 x86_64 on Intel CPU E4500 2.2GHz with 4GB memory. Notice that we get a substantial, certainly, more than linear, reduction in the number of states.

Number of	Full state space		Reduced state space	
cryptographers	size	time	size	time
3	864	0.30	448	0.12
4	6480	0.36	2160	0.13
5	46656	3.6	9984	0.86
6	326592	34	45248	5.2
7	2239488	308	202752	30
8	15116544	2827	900864	175

Table 1. Verification results for DC

The above results show the effectiveness of the technique.

6 Conclusions and Further work

As we argued in the introduction model checking multi-agent systems is now a rapidly maturing area of research with techniques and tools being rapidly applied to the validation of concrete MAS. While some techniques - notably ordered binary decision diagrams and bounded model checking have been redefined in a MAS setting - others, including abstraction and partial order reduction, are still largely unexplored. In particular, partial order reduction is one of the more traditional approaches, and it is therefore surprising that its study has not been systematically carried out yet in a MAS setting.

In this paper we tried to continue the preliminary analysis suggested in [9]. While only a notion of trace-equivalence is explored there, here we focused on interleaved interpreted systems, for which we were able to give stuttering equivalence preservation results, a linear algorithm preserving the validity on the models, as well as an implementation thereby evaluating the performance on two standard MAS scenarios. The results are very encouraging. In both examples the reductions presented are very considerable, possibly exponential.

Much remains to be done in this line. For instance, the partial order reduction technique presented here may be combined with ordered binary decision diagrams (for example within the MCMAS toolkit [10]) so that models are reduced first and then symbolically encoded. It should also be noted that the analysis presented here only applies to interleaved multi-agent systems and properties of LTLK^{J}_{-X}. The case of fully synchronous systems still remains to be tackled as well an extension to partial order reductions for CTL*K_{-X}. This extension can be obtained by strengthening the notion

of stuttering bisimulation as well the conditions for defining a heuristic for the selection of $E(g)$ of [6].

References

1. R. Bordini, M. Fisher, C. Pardavila, W. Visser, and M. Wooldridge. Model checking multi-agent programs with CASP. In *CAV'03*, volume LNCS 2725, pages 110–113. Springer-Verlag, 2003.
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, Cambridge, Massachusetts, 1999.
4. C. Courcoubetis, M. Vardi, P. Wolper, and M. Yannakakis. Memory-efficient algorithms for the verification of temporal properties. *Formal Methods in System Design*, 1(2/3):275–288, 1992.
5. R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge*. MIT Press, Cambridge, 1995.
6. R. Gerth, R. Kuiper, D. Peled, and W. Penczek. A partial order approach to branching time logic model checking. *Information and Computation*, 150:132–152, 1999.
7. J. Halpern, R. van der Meyden, and M. Y. Vardi. Complete axiomatisations for reasoning about knowledge and time. *SIAM Journal on Computing*, 33(3):674–703, 2003.
8. M. Kacprzak, A. Lomuscio, A. Niewiadomski, W. Penczek, F. Raimondi, and M. Szreter. Comparing BDD and SAT based techniques for model checking Chaum's dining cryptographers protocol. *Fundamenta Informaticae*, 63(2,3):221–240, 2006.
9. A. Lomuscio, W. Penczek, and H. Qu. Towards partial order reduction for model checking temporal epistemic logic. In *MoChArt*, LNCS 5348, pages 106–121. Springer, 2009.
10. A. Lomuscio, H. Qu, and F. Raimondi. MCMAS: A model checker for the verification of multi-agent systems. In *Proceedings of CAV 2009 (to appear)*. Springer Verlag, 2009.
11. K. L. McMillan. A technique of a state space search based on unfolding. *Formal Methods in System Design*, 6(1):45–65, 1995.
12. D. Peled. All from one, one for all: On model checking using representatives. In *Proceedings of the 5th International Conference on Computer Aided Verification*, LNCS 697, pages 409–423. Springer-Verlag, 1993.
13. D. Peled. Combining partial order reductions with on-the-fly model-checking. In *Proceedings of the 6th International Conference on Computer Aided Verification*, LNCS 818, pages 377–390. Springer-Verlag, 1994.
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. A. Puri and P. Varaiya. Verification of hybrid systems using abstractions. In *Hybrid Systems II*, LNCS 999, pages 359–369. Springer, 1995.
16. 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.
17. R. van der Meyden and K. Su. Symbolic model checking the knowledge of the dining cryptographers. In *Proceedings of the 17th IEEE Computer Security Foundations Workshop (CSFW'04)*, pages 280–291. IEEE Computer Society, 2004.