Partial order reductions for model checking temporal epistemic logics over interleaved multi-agent systems

Alessio Lomuscio[*]
Department of Computing
Imperial College London, UK

Wojciech Penczek
ICS PAS
and Univ. of Podlasie, Poland

Hongyang Qu[†]
Oxford University Computing
Laboratory, UK

ABSTRACT

We investigate partial order reduction for model checking 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. We present a notion of stuttering-equivalence, and prove the semantical equivalence of stuttering-equivalent traces with respect to linear and branching time temporal logics for knowledge without the next operator. We give algorithms to reduce the size of the models before the model checking step and show preservation properties. We evaluate the technique by discussing the experimental results obtained against well-known examples in the MAS literature.

Categories and Subject Descriptors

D.2 [**Software Engineering**]: Software/Program Verification

General Terms

Verification

Keywords

Verification of MAS, Model Checking, Partial order reduction

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., [31]. In an attempt to limit the state-space explosion problem (i.e., the difficulty 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 [31, 27], and bounded model checking via propositional satisfiability [24]. Both 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* [23] 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 [17] 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. 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 introducing partial order reduction on 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., [20, 7]). Several settings in MAS, where the moves are carried out following turns (e.g., games), or where joint actions are not considered (e.g., non-interacting robots), can also be easily modelled in this way.

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. Ideally the generation is conducted on the fly and M_S is actually never built explicitly. 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$_{-X}$and CTLK$_{-X}$. These temporal epistemic logics with knowledge are very commonly used in a MAS settings but appear here without the "next" operator because of standard inherent lim-

[*]Partly supported by EPSRC project EP/02727X.

[†]Supported by the European Commission Framework Program 7 project CONNECT (IST Project Number 231167).

Cite as: Partial order reductions for model checking temporal-epistemic logics over interleaved multi-agent systems, A. Lomuscio, W. Penczek and H. Qu, *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, van der Hoek, Kaminka, Lespérance, Luck and Sen (eds.), May, 10–14, 2010, Toronto, Canada, pp. XXX-XXX.

itations in the technique we present. In Section 3 we proceed to present a notion of stuttering-equivalence with respect to IIS. We move on to describe novel partial order algorithms that preserve LTLK$_{-X}$ and CTL*K$_{-X}$ 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. Further, we assume that 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.

We begin by assuming a MAS to be composed of n agents $\mathcal{A} = \{1, \ldots, n\}$[1]. 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 $Agent(a) \subseteq \mathcal{A}$ 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 2.1 (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 $t_i(g^i, act_i) = g'^i$. We denote the above as $g \xrightarrow{a} g'$.*

[1]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.

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 allow 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 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 2.2 (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) [26]) composed of three agents: a controller and two trains. Each train runs on a circular track and both tracks pass through a narrow tunnel (state 'T'), allowing one train only to go through it to state Away ('A') at any time. The controller operates the signal (Green ('G') and Red ('R')) to let trains enter and leave the tunnel. In the figure, the initial states of the controller and the train are 'G' and 'W' (Waiting) respectively, and the transitions with the same label are synchronised. Silent ϵ actions are omitted in the figure.

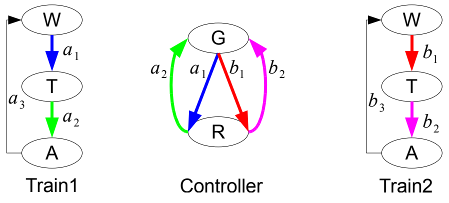

Figure 1: An IIS of TGC composed of two trains

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

DEFINITION 2.3. *Let $i \in \mathcal{A}$, $g, g' \in G$, and $J \subseteq \mathcal{A}$.*

- $\sim_i = \{(g, g') \in G \times G \mid g^i = g'^i\}$, $\quad \sim_J = \bigcap_{j \in J} \sim_j$.
- $I = \{(a, b) \in Act \times Act \mid Agent(a) \cap Agent(b) = \emptyset\}$.

The first relation ($\sim_i$) is 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. Notice that $\sim_\emptyset = G \times G$ while $\sim_{\mathcal{A}} = id_G$. We say that two actions a, a' are dependent if $(a, a') \notin I$. For each of the relations R given in Def. 2.3 by $x\ R\ y$ we mean that $(x, y) \in R$.

DEFINITION 2.4 (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, 9]. We recall the basic definitions here and adapt them to our purposes when needed. Let PV be a finite set of propositions. For generality, we first 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 $\neg\varphi$, $\varphi \wedge \psi$, and $K_i\varphi$ ($i \in \mathcal{A}$),

S3. if φ is a path formula, then $A\varphi$ and $E\varphi$ 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 path quantifier A has the intuitive meaning "for all paths" whereas E stands for "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: $true \overset{def}{=} \neg(p \wedge \neg p)$, for some $p \in PV$, $F\varphi \overset{def}{=} U(true, \varphi)$, $G\varphi \overset{def}{=} \neg F \neg\varphi$. As standard, F represents the temporal operator of "eventually" (in the future) and G corresponds to "forever" (in the future). Given their intuitive interpretation, sometimes we call A, E state modalities, and K_i, U, G, F path modalities. We now define a variety of logics included in CTL*K$_{-X}$.

DEFINITION 2.5.

- *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 Aand E. We write φ instead of $A\varphi$ if confusion is unlikely.*
- *ACTL*K$_{-X}$ $\subset$ CTL*K$_{-X}$ is the fragment of CTL*K$_{-X}$ in which the state modality E does not appear in any formula and the negation only appears in subformulas not containing any state or path modalities.*
- *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 $i \in J$.*

Observe the logics CTLK and LTLK without next operators are subsets of the logics above.

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

Let $M = (G, \iota, \Pi, V)$ be a model and let $\pi = g_0 a_0 g_1 \cdots$ be an infinite path of G. Let π_i denote the suffix $g_i a_i g_{i+1} \cdots$ 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 $g \models \psi$,
$g \models K_i\varphi$ iff $g' \models \varphi$ for every $g' \in G$ such that $g \sim_i g'$,

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,

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. EQUIVALENCES

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 3.1. *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 3.2. *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:*

- *$Pr_J(g_0) = g_0$;*
- *$Pr_J(g_0 \cdots g_i) = Pr_J(g_1 \cdots g_i)$ if $V(g_0) = V(g_1)$ and $g_0 \sim_J g_1$; $Pr_J(g_0 \ldots g_i) = g_0 Pr_J(g_1 \ldots g_i)$ otherwise.*

3.1 Equivalence preserving LTLK$^J_{-X}$

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 3.3 (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 3.4 (STUTTERING EQUIVALENCE). *A path π in M and a path π' in M' are called* stuttering equivalent*, denoted $\pi \equiv_s \pi'$, if there exists a partition $B_1, B_2 \dots$ of the states of π, and a partition $B'_1, B'_2 \dots$ 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 3.5 (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 3.6 (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 *denoted $M \equiv_{Jks} M'$, if $\iota \equiv_{Jks} \iota'$.*

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

THEOREM 3.7. *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. Part 1: First we prove that for each path $\pi = g_0 a_0 g_1 a_1 \dots$ of M if a_i is J-invisible, then $M, \pi_i \models \varphi$ iff $M, \pi_{i+1} \models \varphi$ for each LTLK$^J_{-X}$ formula φ.

By induction on the structure of φ. For $\varphi \in PV$ the thesis follows directly from the definition of J-invisibility. The case of $\wedge$ and $\neg$ is straightforward. For $\varphi = U(\phi, \psi)$ the thesis follows directly from the semantics of U and the inductive assumption. Consider $\varphi = K_i \psi$. If $M, \pi_i \models \varphi$, then it follows from $\pi(i) \sim_J \pi(i+1)$ that $M, \pi_{i+1} \models \psi$ and since $\sim_i$ is an equivalence relation we have $M, \pi_{i+1} \models K_i \psi$. A similar proof holds for $M, \pi_{i+1} \models \varphi$ implies $M, \pi_i \models \varphi$.

Part 2: Now, we prove the theorem itself, also by induction on the complexity of φ. In fact, we prove a stronger result, i.e., that from $g \equiv_{Jks} g'$, it follows that $M, g \models \varphi$ iff $M', g' \models \varphi$, for any LTLK$^J_{-X}$ formula φ over PV.

($\Rightarrow$): For $\varphi \in PV$ the thesis follows directly from the definition of $\sim_{Jks}$. The cases of $\wedge$ and $\neg$ are straightforward. Consider $\varphi = U(\phi, \psi)$ and a path π starting at g. We have $\pi \models U(\phi, \psi)$. Then, there is a path π' starting at g' such that $\pi \equiv_{Jks} \pi'$. By the inductive assumption we have that $\pi \models \phi$ iff $\pi' \models \phi$ and $\pi \models \psi$ iff $\pi' \models \psi$. Since $\pi \equiv_{Jks} \pi'$ we have that $Pr_J(\pi)_i \models \phi$ iff $Pr_J(\pi')_i \models \phi$ and $Pr_J(\pi)_i \models \psi$ iff $Pr_J(\pi')_i \models \psi$ for all $i \geq 0$. Thus, it follows from Part 1) that $\pi' \models \varphi$. So, clearly we have that if $M, g \models U(\phi, \psi)$, then $M, g' \models U(\phi, \psi)$.

Consider $\varphi = K_i \psi$ and let $M, g \models \varphi$. Let $G_\psi = \{g_1 \in G \mid g \sim_i g_1\}$. Consider g'_1 s.t. $g' \sim_i g'_1$. We have to show that $M', g'_1 \models \psi$. Since $g \equiv_{Jks} g'$, by transitivity of $\sim_i$ we have that $g'_1 \in G_\psi$. So, clearly $M, g'_1 \models \psi$. As $g'_1 \equiv_{Jks} g'_1$, it follows from the inductive assumption that $M', g'_1 \models \psi$. So, we get $M', g' \models \varphi$.

($\Leftarrow$) We consider only the case of $\varphi = K_i \psi$. The proof for other cases is similar to ($\Rightarrow$).

Let $\varphi = K_i \psi$ and let $M', g' \models \varphi$. Let $G'_\psi = \{g'_1 \in G' \mid g' \sim_i g'_1\}$. Consider g_1 s.t. $g \sim_i g_1$. We have to show that $M, g_1 \models \psi$. Consider a path in M starting at ι which contains g_1. Since $M \equiv_{Jks} M'$, there is a path in M' starting at ι', which contains a state $g'_2 \in G'$ such that $g_1 \equiv_{Jks} g'_2$. So, $g'_2 \in G'_\psi$ by transitivity of $\sim_i$. Thus, clearly $M', g'_2 \models \psi$. As $g_1 \equiv_{Jks} g'_2$, it follows from the inductive assumption that $M, g_1 \models \psi$. So, $M, g \models \varphi$. □

3.2 Equivalences preserving ACTL*K$^J_{-X}$ and CTL*K$^J_{-X}$

As before 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 (bi)-simulation* between M and M'. Then, we define *visible J-(bi)-simulation* between two models.

DEFINITION 3.8 (J-STUTTERING SIMULATION). *A relation $\sim_{jss} \subseteq G' \times G$ is a* J-stuttering simulation *between two models M and M' if the following conditions hold:*

1. *$\iota' \sim_{jss} \iota$,*
2. *if $g' \sim_{jss} g$, then $JKS(g, g')$ and for every path π of M, there is a path π' in M', a partition $B_1, B_2 \dots$ of π, and a partition $B'_1, B'_2 \dots$ of π' such that for each $j \geq 0$, B_j and B'_j are nonempty and finite, and every state in B'_j is related by $\sim_{jss}$ to every state in B_j.*

A relation $\sim_{jss}$ is a J-stuttering bisimulation *if both $\sim_{jss}$ and $\sim^T_{jss}$ (T denotes transposition) are J-stuttering simulations.*

Model M' J-stuttering simulates *model M ($M \leq_{jss} M'$) if there is a J-knowledge stuttering simulation between M and M'. Two models M and M' are called* J-stuttering simulation equivalent *if $M \leq_{jss} M'$ and $M' \leq_{jss} M$. Two models M and M' are called* J-visible bisimulation equivalent *if there is a J-visible bisimulation between M and M'.*

The following theorem connects ACTL*K$^J_{-X}$ with J-stuttering simulation equivalence:

THEOREM 3.9. *Let M and M' be two J-stuttering simulation equivalent models. Then, $M, \iota \models \varphi$ iff $M', \iota' \models \varphi$, for any ACTL*K$^J_{-X}$ formula φ over PV.*

PROOF. J-stuttering simulation equivalence is clearly stronger than stuttering simulation equivalence, which preserves ACTL$^*_{-X}$ [25]. Similarly to the proof of Theorem 3.7 one can show that ACTL*K$^J_{-X}$ is preserved by J-stuttering simulation equivalence. □

THEOREM 3.10. *Let M and M' be two J-stuttering bisimilar models. Then, $M, \iota \models \varphi$ iff $M', \iota' \models \varphi$, for any CTL*K$^J_{-X}$ formula φ over PV.*

PROOF. J-stuttering bisimulation is clearly stronger than stuttering bisimulation, which preserves CTL$^*_{-X}$ [7]. Similarly to the proof of Theorem 3.7 one can show that CTL*K$^J_{-X}$ is preserved by J-stuttering bisimulation. □

Next, we define two relations such that one is stronger than J-stuttering simulation, while the other one is stronger than J-stuttering bi-simulation. Both of them will be used for our partial order reductions.

DEFINITION 3.11 (J-VISIBLE SIMULATION). *A relation $\sim_{jkvs} \subseteq G' \times G$ is a* J-visible simulation *between the states of two models M and M' if*

- *$\iota' \sim_{jkvs} \iota$, and*

- *if $g' \sim_{jkvs} g$, then the following conditions hold:*
   1. $JKS(g, g')$.
   2. *If $g \xrightarrow{b} t$, then either b is J-invisible and $g' \sim_{jkvs} t$, or there exists a path $g' = g_0 \xrightarrow{a_0} g_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{n-1}} g_n \xrightarrow{b} t'$ in M' such that $g_i \sim_{jkvs} g$ for $i \leq n$, a_i is J-invisible for $i < n$ and $t' \sim_{jkvs} t$.*
   3. *If there is an infinite path $g = t_0 \xrightarrow{b_0} t_1 \xrightarrow{b_1} \cdots$, where b_i is J-invisible and $g' \sim_{jkvs} t_i$ for $i \geq 0$, then there exists an edge $g' \xrightarrow{c} g''$ such that c is J-invisible and $g'' \sim_{jkvs} t_j$ for some $j > 0$.*

A relation $\sim_{jkvs}$ is a J-visible bisimulation *if both $\sim_{jkvs}$ and $\sim_{jkvs}^T$ are J-visible simulations.*

Model M' J-visible simulates *model M (denoted $M \leq_{jkvs} M'$) if there is a J-visible simulation between the states of M and M'. Two models M and M' are called* J-visible simulation equivalent *if $M \leq_{jkvs} M'$ and $M' \leq_{jkvs} M$. Two models M and M' are called* J-visible bisimulation equivalent *if there is a J-visible bisimulation between the states of the two models.*

It is quite straightforward to show that J-visible bisimulation (simulation) is stronger that J-stuttering bisimulation (simulation, resp.).

This concludes our analysis of equivalences preserving LTLK$_{-X}$, ACTL*K$_{-X}$, and CTLK$_{-X}$. For each of the above mentioned logics, we now give an algorithm that for a given MAS and a formula returns a reduced model. By means of the theorems we show that the reduced model is equivalent to the full one.

4. PARTIAL ORDER REDUCTIONS

As mentioned above, the idea of verification by model checking with partial order reduction is to define an algorithm that given a model can produce a smaller model provably validating the same formulae of interest. This requires a notion of equivalence between models. For the case of LTLK$^J_{-X}$ we show below that the notion of J-stuttering trace equivalence presented above suffices. With respect to branching time, for CTL*K$_{-X}$ (ACTL*K$_{-X}$, respectively) we show we can use J-visible bisimulation (J-visible simulation equivalence, respectively). 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 [7]), or double-depth-first-search (DDFS) [4].

In this context DFS is used to compute paths that will make up the reduced model by exploring systematically the possible computation tree and selecting only some of the possible paths generated. In the following, 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 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 (see below).
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 thereby 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. Its size 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$^J_{-X}$

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' define 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 $a \in Act \backslash E(g)$ that is dependent (see Definition 2.3) on an action in $E(g)$ can be executed before an action in $E(g)$ is executed.

C2 For every cycle in the constructed state graph there is at least one node g in the cycle 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 3.1).

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 **C1**$-$**C3** are inspired from [22], 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 4.1. *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:*

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

PROOF. Although the setting is different it can be shown similarly to Theorem 3.11 in [23] that the conditions **C1, C2, C3** guarantee that the models M and M' are stuttering equivalent. More precisely, for each path $\pi = g_0 a_0 g_1 a_1 \cdots$ with $g_0 = \iota$ in M there is a stuttering equivalent path $\pi' = g'_0 a'_0 g'_1 a'_1 \cdots$ with $g'_0 = \iota$ in M' and a partition $B_1, \ldots, B_j, ..$ of the states of π and a partition $B'_1, \ldots, B'_j, ..$ of the states of π' satisfying for each $i, j \geq 0$ the following two conditions:

I. if $g_i \xrightarrow{a} g_{i+1}$ is a transition such that $g_i, g_{i+1} \in B_j$, then $a \in Invis$, and if $g'_i \xrightarrow{a'} g'_{i+1}$ is a transition such that $g'_i, g'_{i+1} \in B'_j$, then $a' \in Invis$,

II. if $g_i \xrightarrow{a} g_{i+1}$ is a transition such that $g_i \in B_j$ and $g_{i+1} \in B_{j+1}$, and $g'_{i'} \xrightarrow{a'} g'_{i'+1}$ is a transition such that $g'_{i'} \in B'_j$ and $g'_{i'+1} \in B'_{j+1}$, then $a = a'$.

Given condition **CJ**, for any two states g, g' in B_j or in B'_j we have that $JKS(g, g')$. Moreover, from the above and condition II one can show by induction that for each state $g \in B_j$ and $g' \in B'_j$ we have $JKS(g, g')$. Since, $M' \subseteq M$, we get that the models M and M' are J-stuttering trace equivalent. The second part of the theorem follows from this and Theorem 3.7. $\square$

4.2 Preserving ACTL*K$_{-X}$ and CTL*K$_{-X}$

Next, let ϕ be a ACTL*K$_{-X}$ or a CTL*K$_{-X}$ 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 above. We now give additional conditions, which together with **C1 - C3, CJ**, define a heuristics for the selection of $E(g)$ (such that $E(g) \neq en(g)$) while visiting state g

C3' $E(g)$ contains all the J-visible actions of $en(g)$; $en(g) \cap Vis^J \neq \emptyset$, i.e., there is at least one J-visible action in $en(g)$.

C4 $E(g)$ is a singleton set.

C5 $E(g)$ contains all the actions starting an infinite J-invisible path from g in M.

The above conditions are inspired from [25].

THEOREM 4.2. *Let M be a model and $M' \subseteq M$ be the reduced model generated by a DFS algorithm described above.*

a) If the choice of $E(g')$ for $g' \in G'$ is given by the conditions **C1, C2, C3'**, *and* **C5**, *then*

- *M and M' are J-visible simulation equivalent,*
- *$M \models \phi$ iff $M' \models \phi$, for any $\phi \in$ ACTL*K$^J_{-X}$.*

b) If the choice of $E(g')$ for $g' \in G'$ is given by the conditions **C1, C2, C3, C4**, *and* **CJ**, *then*

- *M and M' are J-visible bisimilar,*
- *$M \models \phi$ iff $M' \models \phi$, for any $\phi \in$ CTL*K$^J_{-X}$.*

PROOF. (sketch) Part a): Since M' is a sub-model of M, it is obvious that M J-visible simulates M'. In order to show the opposite, define the following relation: $\sim \subseteq G' \times G$ by $g' \sim g$ iff there exists a path $g' = g_0 \xrightarrow{a_0} g_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{n-1}} g_n = g$ such that a_i is J-invisible for all $i < n$. Let $\approx = \sim \cap (G' \times G)$. In [25] it is shown that if the reduction algorithm uses the conditions **C1, C2**, and the conditions weaker than **C3', C5** (visibility is used instead of J-visibility), then the relation weaker than $\approx$ is a visible simulation. Notice that with the change of visibility to J-visibility in the conditions, $\approx$ can be proved to be a J-visible simulation.
Part b:) Define $\sim \subseteq G \times G$ by $g \sim g'$ iff there exists a path $g_0 \xrightarrow{a_0} g_1 \xrightarrow{a_1} \ldots \xrightarrow{a_{n-1}} g_n = g'$, with $g_0 = g$ such that a_i is J-invisible and $\{a_i\}$ satisfies the condition **C1** from state g_i for $0 \leq i < n$. Consider $\approx = \sim \ (G \times G')$. It is shown in [7] that the relation weaker than $\approx$ is a visible bisimulation. By slightly modifying the original proof, one can show that $\approx$ is a J-visible bisimulation. □

4.3 The DFS-POR algorithm

We now give details of the DFS algorithm implementing conditions **C1, C2, C3**, and **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 is J-stuttering equivalent and produces attractive results in several cases.

Algorithm 1 DFS-POR ()

```
1:  $g \Leftarrow Top(Stack1)$; $reexplore \Leftarrow false$;
2:  if $g = Element(Stack1, i)$ then
3:      $depth \Leftarrow Top(Stack2)$;
4:      if $i > depth$ then
5:          $reexplore \Leftarrow true$;
6:      else
7:          $Pop(Stack1)$; return;
8:      end if
9:  end if
10: if $reexplore = false$ and $g \in G$ then
11:     $Pop(Stack1)$; return;
12: end if
13: $G \Leftarrow G \cup \{g\}$; $E(g) \Leftarrow \emptyset$;
14: if $en(g) \neq \emptyset$ then
15:     if $reexplore = false$ then
16:         for all $a \in en(g)$ do
17:             if $a \notin Vis$ and $a \notin Vis_J$ and $\forall b \in en(g) \setminus \{a\} : (a, b) \in I$
                then
18:                 $E(g) \Leftarrow \{a\}$; break;
19:             end if
20:         end for
21:     end if
22:     if $E(g) = \emptyset$ then $E(g) \Leftarrow en(g)$; end if
23:     if $E(g) = en(g)$ then
24:         $Push(Stack2, Depth(Stack1))$;
25:     end if
26:     for all $a \in E(g)$ do
27:         $g' \Leftarrow Successor(g, a)$; $Push(Stack1, g')$; DFS-POR();
28:     end for
29: end if
30: $depth \Leftarrow Top(Stack2)$;
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., [22] 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.

To add **C4** to the algorithm (to preserve CTL*K$^J_{-X}$), we simply change line 18 to be $E(g) \Leftarrow E(g) \cup \{a\}$, and change the condition in line 22 to $|E(g)| \neq 1$, where $|E(g)|$ is the cardinality of $E(g)$.

In order to adapt the algorithm for preserving ACTL*K$^J_{-X}$, we need to replace the **for** statement starting at line 16 with the following code.

$DependentOf(X1, X2)$ recursively computes the set $X' \subseteq$

Algorithm 2 Checking **C3'** and **C5**

```
1: $E(g) \Leftarrow (en(g) \cap Vis_J)$;
2: $E(g) \Leftarrow E(g) \cup \{a \in en(g) \setminus E(g) \mid$ $action$ a starts a loop composed of J-invisible actions$\}$;
3: $E(g) \Leftarrow DependentOf(E(g), en(g) \setminus E(g))$;
```

$X1 \cup X2$ such that for all $a \in X'$, either $a \in X1$ or there exists a set of actions $\{a_1, \ldots, a_m\} \subseteq X'$ with $(a_i, a_{i+1}) \notin I$ $(1 \leq i < m)$ and $a = a_1$, $a_m \in X1$. As proposed in [25], an action starts an infinite J-invisible path if it starts a local infinite invisible path in the local state space of each agent of J. Here a local infinite invisible path consists of only invisible actions from the agent, ignoring synchronisations with other agents.

5. EXPERIMENTAL RESULTS

In order to evaluate the results above, we have implemented the DFS-POR algorithms to verify specification properties in LTLK$^J_{-X}$, ACTL*K$_{-X}$, and CTL*K$^J_{-X}$. 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 three systems: the TGC of Section 2.1, the Dining Cryptographers (DC) [2], and the Write-Once cache coherence protocol (WO) [32, 33], 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[2]. In the case of the LTLK$^J_{-X}$ algorithm, 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. We also found that the algorithm for CTL*K$^J_{-X}$ gives the same reduction as the one for LTLK$^J_{-X}$. However, we found that the ACTL*K$^J_{-X}$ variant does not produce any reduction. The reason is that there is an invisible loop in the controller and the trains.

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 algorithms to reduce the models preserving the specification [15]:

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

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 an AMD Opteron clocked at 2.2GHz with 8GB memory running a vanilla Linux kernel 2.6.30. Notice that we get a substantial, certainly, more than linear, reduction in the number of states. In this case, we found the algorithm for CTL*K$^J_{-X}$ brought negligible benefits of less than 1%.

N	Full space		LTLK$^J_{-X}$		ACTL*K$^J_{-X}$	
	size	time	size	time	size	time
3	864	0.41	448	0.12	320	0.27
4	6480	0.49	2160	0.18	1760	0.30
5	46656	4.6	9984	1.8	9600	1.5
6	326592	44	45248	6.8	51072	7.7
7	2239488	465	202752	39	264192	57
8	15116544	4723	900864	228	1331712	380

Table 1: Verification results for DC.

The choice of the Write-One cache coherence protocol was inspired by [32], which analysed several cache coherence protocols in a knowledge setting. We followed some of the criteria presented in [32] while modelling WO: each cache contains a single bit, whose value can be either 0 or 1, the owner of the bit is either the main memory or a cache. The details of the protocol can be found in [33]. For the three algorithms, we tested the formula

$$AG((\text{Dirty}_1 \vee \text{Reserved}_1) \rightarrow (K_{\text{cache}_1} \bigwedge_{i=2}^{n} \text{Invalid}_i)),$$

where Dirty$_i$, Reserved$_i$ and Invalid$_i$ represent that cache i is in the state dirty, reserved, or invalid. Our implementation results in a substantial reduction only for the LTLK$^J_{-X}$ case, see Table 2.

N	Full space		LTLK$^J_{-X}$	
	size	time	size	time
2	322	0.27	82	0.13
3	3668	0.59	1066	0.29
4	40110	14.8	12402	5.6
5	426984	264	131402	77
6	4451778	3042	1311698	529
7			12585834	8870

Table 2: Verification results for WO.

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 [17]. 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 we found were very positive in the linear time case, less so for the branching time case. There are two reasons for that. Firstly, the equivalences induced by the branching time logics are much stronger than the equivalence induced by LTLK$^J_{-X}$. Secondly, for scenarios containing

[2]In the case of LTLK$^J_{-X}$ tests substitute AG with G in the formulas.

loops composed of J-invisible actions, the reductions preserving ACTL*K^{J}_{-X} are quite limited.

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 [18]), or combined with bounded model checking (for example within the VerICS toolkit [14]), 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. The case of fully synchronous systems still remains to be tackled.

7. REFERENCES

[1] R. Bordini, M. Fisher, C. Pardavila, W. Visser, and M. Wooldridge. Model checking multi-agent programs with CASP. In *Proc. of CAV03*, LNCS 2725, 110–113. Springer, 2003.

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

[3] E. Clarke, O. Grumberg, and D. 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] P. Gammie and R. van der Meyden. MCK: Model checking the logic of knowledge. In *Proc. of CAV04*, LNCS 3114, 479–483. Springer, 2004.

[7] 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.

[8] P. Godefroid. Using partial orders to improve automatic verification methods. In *Proc. of CAV90*, LNCS 531, 176 - 185. Springer, 1991.

[9] J. Halpern, R. Meyden, and M. Y. Vardi. Complete axiomatisations for reasoning about knowledge and time. *SIAM Journal on Computing*, 33(3):674–703, 2003.

[10] J. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. *Journal of the ACM*, 37(3):549–587, 1990.

[11] W. v. Hoek and M. Wooldridge. Model checking knowledge and time. In *Proc. of SPIN02*, LNCS 2318, 95–111. Springer, 2002.

[12] G. Holzmann and D. Peled. Partial order reduction of the state space. In *Proc. of SPIN95*, Montréal, Quebec, 1995.

[13] G. Holzmann, D. Peled, and M. Yannakakis. On nested depth first search. In *Proc. of SPIN96*, 23–32. AMS, 1996.

[14] M. Kacprzak, W. Nabialek, A. Niewiadomski, W. Penczek, A. Polrola, M. Szreter, B. Wozna, and A. Zbrzezny. VerICS 2007 - a Model Checker for Knowledge and Real-Time. Fundamenta Informaticae 85(1-4): 313-328, 2008.

[15] 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.

[16] M. E. Kurbán, P. Niebert, H. Qu, and W. Vogler. Stronger reduction criteria for local first search. In *Proc. of ICTAC06*, LNCS 4281, 108–122. Springer, 2006.

[17] A. Lomuscio, W. Penczek, and H. Qu. Towards partial order reduction for model checking temporal epistemic logic. In *Proc. of MoChArt08*, LNAI 5348, 106–121. Springer, 2009.

[18] A. Lomuscio, H. Qu, and F. Raimondi. MCMAS: A model checker for the verification of multi-agent systems. In *Proc. of CAV09*, LNCS 5643, 682–688. Springer, 2009.

[19] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[20] K. L. McMillan. A technique of a state space search based on unfolding. *Formal Methods in System Design*, 6(1):45–65, 1995.

[21] R. Parikh and R. Ramanujam. Distributed processes and the logic of knowledge. In *Logic of Programs*, LNCS 193, 256–268. Springer, 1985.

[22] D. Peled. All from one, one for all: On model checking using representatives. In *Proc. of CAV93*, LNCS 697, 409–423. Springer, 1993.

[23] D. Peled. Combining partial order reductions with on-the-fly model-checking. In *Proc. of CAV94*, LNCS 818, 377–390. Springer, 1994.

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

[25] W. Penczek, M. Szreter, R. Gerth, and Ruurd Kuiper. Improving Partial Order Reductions for Universal Branching Time Properties. *Fundamenta Informaticae*, 43(1-4): 245-267, 2000.

[26] A. Puri and P. Varaiya. Verification of hybrid systems using abstractions. In *Hybrid Systems II*, LNCS 999, 359–369. Springer, 1995.

[27] F. Raimondi and A. Lomuscio. Automatic verification of multi-agent systems by model checking via OBDDs. *Journal of Applied Logic*, 5(2):235-251, 2007.

[28] S. Rosenschein. Formal theories of ai in knowledge and robotics. *New Generation Computing*, 3:345–357, 1985.

[29] A. Valmari. A stubborn attack on state explosion. In *Proc. of CAV90*, LNCS 531, 156–165. Springer, 1990.

[30] W. van der Hoek, M. Roberts, and M. Wooldridge. Knowledge and social laws. In *Proc. of AAMAS05*, 674–681. ACM Press, 2005.

[31] R. van der Meyden and K. Su. Symbolic model checking the knowledge of the dining cryptographers. In *Proc. of CSFW04*, 280–291, IEEE. 2004.

[32] K. Baukus and R. van der Meyden. A Knowledge Based Analysis of Cache Coherence. In *Proc. of ICFEM04*, LNCS 3308, 99–114. Springer. 2004.

[33] J. Archibald and J.-l. Baer. Cache coherence protocols: Evaluation using a multiprocessor simulation model. *ACM Transactions on Computer Systems*, 4:273–298, 1986.