Automatic verification of multi-agent systems by model checking via ordered binary decision diagrams

Franco Raimondi, Alessio Lomuscio

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

Abstract

We present a methodology for the verification of multi-agent systems, whose properties are specified by means of a modal logic that includes a temporal, an epistemic, and a modal operator to reason about correct behaviour of agents. The verification technique relies on model checking via ordered binary decision diagrams. We present an implementation and report on experimental results for two scenarios: the bit transmission problem with faults and the protocol of the dining cryptographers.

1 Introduction

In the last two decades, the paradigm of multi-agent systems (MAS) has been employed successfully in several disciplines, including, for example, philosophy, economics, and software engineering. One of the reasons for the use of MAS formalisms in such different areas is the usefulness of ascribing autonomous and social behaviour to the components of a system of agents. This allows us to *abstract* from the details of the components, and to focus on the *interaction* among the various agents. The modelling of MAS in such scenarios is typically conducted by using logic-based formalisms [1,2].

Besides *abstracting* and *specifying* the behaviour of a complex system by means of MAS formalisms based on logic, recently researchers have been concerned with the problem of *verifying* MAS. Namely, if we model a real system by means of a MAS formalism, how can we *verify formally* that the system complies with certain desired properties? Formal verification is normally associated with traditional software engineering, where one wants to validate a piece of software or hardware against a specification. One of the most successful formal techniques to verification is *model checking* [3]. In this approach, a system S to be verified is represented by means of a logical model M_S encoding the computational traces of the system, and a property P to be checked is expressed via a logical formula φ_P. Verification via

model checking is defined as the problem of establishing whether or not $M_S \models \varphi_P$. Various tools have been built to perform this task automatically for temporal logic models (SMV [4], SPIN [5], NuSMV [6], and others), and several concrete systems have been tested.

Unfortunately, extending model checking techniques to the verification of MAS is not trivial. This is because model checking tools are tailored to standard reactive systems, and do not allow for the representation of the social interaction and the autonomous behaviour of the agents. Specifically, traditional model checking tools assume that M_S is "simply" a *temporal* model, while MAS need more complex formalisms. Typically, in MAS we want to reason about the epistemic, intentional, and doxastic properties of agents, as well as their temporal evolution. Hence, the logical models required are richer than the temporal model used in traditional model checking.

In this paper we consider the formalism of interpreted systems [7] to reason about temporal and epistemic properties of agents, and an extension of interpreted systems with modal operators to reason about correct behaviour [8]. Based on this formalism, we extend the model checking algorithm that appeared in [9] and we present an implementation relying on Ordered Binary Decision Diagrams (OBDDs) to verify temporal, epistemic, and correct behaviour modalities in interpreted systems.

The rest of the paper is organised as follows. In Section 2 we review the framework of deontic interpreted systems and model checking via OBDDs. In Section 3 we introduce a technique for the verification of deontic interpreted systems. An implementation of the algorithm is then discussed in Section 4. In Section 5 we test the soundness of our implementation by means of two examples: the bit transmission problem with faults and the protocol of the dining cryptographers. We discuss our results, in comparison with existing work, in Section 6, and we conclude in Section 7.

2 Preliminaries

In this section we briefly summarise the formalism of interpreted systems as presented in [7] to model MAS, and its extension to reason about the correct behaviour of agents as presented in [10]. After this, we summarise the approach to model checking via OBDDs.

2.1 Deontic interpreted systems and their temporal extension

An *interpreted system* [7] is a formalism representing a system of agents. Each agent i ($i \in \{1, \ldots, n\}$) in the system is characterised by a finite set of *local states* L_i and by a finite set of actions Act_i. Actions are performed in compliance with a protocol $P_i : L_i \rightarrow 2^{Act_i}$, specifying which actions may be performed in a given state. In this formalism, the environment in which agents "live" may be modelled by means of a special agent E. Associated with E are a set of local states L_E, a set of actions Act_E, and a protocol P_E. A tuple $g = (l_1, \ldots, l_n, l_E) \in L_1 \times \ldots \times L_n \times L_E$, where $l_i \in L_i$ for each i and $l_E \in L_E$, is called a *global state* and gives a description of the system at a particular instant of time. This description assumes a "global" time and it has been proven useful in many circumstances [7] (approaches in which time is local can be obtained by considering local clocks [11]).

The evolution of the agents' local states is described by a function $t_i : L_i \times L_E \times Act_1 \times \ldots \times Act_n \times Act_E \rightarrow L_i$, which returns a local state (the "next" local state) for agent i, given the "current" local state of the agent, the "current" local state of the environment, and all the agents' actions. Similarly, the evolution of the environment's local states is described by a function $t_E : L_E \times Act_1 \times \ldots \times Act_n \times Act_E \rightarrow L_E$. It is assumed that, in every state, agents evolve simultaneously (such a composition is usually referred to as a *lock-step* system): the evolution of the global states of the systems is described by a function $t : S \times Act \rightarrow S$, where $S = L_1 \times \ldots \times L_n \times L_E$, and $Act = Act_1 \times \ldots \times Act_n \times Act_E$. The function t is defined as $t(g, a) = g'$ iff for all $i, t_i(l_i(g), a) = l_i(g')$ and $t_E(l_E(g), a) = l_E(g')$, where $l_i(g)$ denotes the i-th component of global state g (corresponding to the local state of agent i). Given a set $I \subseteq S$ of possible *initial global states*, a set $G \subseteq S$ of *reachable global states* is generated by all the possible runs of the system.

In [10] the notion of *correct behaviour* of the agents is incorporated in the formalism. This is done by partitioning the set of local states into two sets: a non-empty set G_i of allowed (or correct, or "green") states, and a set R_i of disallowed (or faulty, or "red") states, such that $L_i = G_i \cup R_i$, and $G_i \cap R_i = \emptyset$.

To complete the description of a MAS, a set of atomic propositions AP is introduced, together with a *valuation relation* $h \subseteq AP \times S$. Finally, given a set of agents $\Sigma = \{1, \ldots, n\}$, we define a *deontic interpreted system* as the tuple:

$$DIS = \left\langle (G_i, R_i, Act_i, P_i, t_i)_{i \in \Sigma}, (G_E, R_E, Act_E, P_E, t_E), I, h \right\rangle.$$

It has been shown [12,7,10] that deontic interpreted systems can provide a semantics to reason about time, knowledge, and correct behaviour.

We analyse multi-agent systems by means of the following language:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid EX(\varphi) \mid EG\varphi \mid E(\varphi U \psi) \mid\ K_i\varphi \mid E_\Gamma\varphi \mid C_\Gamma\varphi \mid D_\Gamma\varphi \mid O_i(\varphi) \mid\ \hat{K}_i^\Gamma(\varphi)$$

In the grammar above, $p \in AP$ is an atomic proposition; EX and EG are temporal operators expressing, respectively, that there exists a next state in which φ holds, and that there exists a run in which φ holds globally; $E(\varphi U \psi)$ is a temporal operator expressing that there exists a run in which φ holds until ψ holds; Γ denotes a non-empty subset of the set of agents, $K_i\varphi$ is read as "agent i knows φ", $E_\Gamma\varphi$ is read as "everybody in group Γ knows φ", $C_\Gamma\varphi$ is read as "it is common knowledge in Γ that φ", and $D_\Gamma\varphi$ is read as "it is distributed knowledge in group Γ that φ [7]; $O_i\varphi$ expresses the fact that, *under all the correct alternatives for agent i, φ holds*; the operator $\hat{K}_i^\Gamma$ expresses the knowledge that agent i has *on the assumption that all agents in Γ are functioning correctly*. By slight abuse of notation, if Γ is a singleton $\Gamma = j$, we write $\hat{K}_i^j$, representing the knowledge of agent i under the assumption that agent j is functioning correctly. Notice that O_i *does not* represent obligations of agent i to φ; we refer to [8] for more details.

Given a deontic interpreted system DIS, we associate to DIS a model $M_{DIS} = (W, R_t, \sim_1, \ldots, \sim_n, R_1^O, \ldots, R_n^O, L)$ that can be used to interpret any formula φ, as follows:

- The set of possible worlds W is the set G of reachable global states.
- The temporal relation $R_t \subseteq W \times W$ relating two worlds (i.e. two global states) is defined by considering the temporal transition t. Two worlds w and w' are such that $R_t(w, w')$ iff there exists a joint action $a \in Act$ such that $t(g, a) = g'$, where t is the transition relation of DIS.
- The epistemic accessibility relations $\sim_i \subseteq\ W \times W$ are defined by considering the equality of the local components of the global states. Two worlds $w, w' \in W$ are such that $w \sim_i w'$ iff $l_i(w) = l_i(w')$ (i.e. two worlds w and w' are related via the epistemic relation $\sim_i$ when the local states of agent i in global states w and w' are the same [7]).
- The accessibility relations $R_i^O \subseteq W \times W$ are defined by considering the local i-th component of the global state g'. Two worlds $w, w' \in W$ are such that $R_i^O(w, w')$ iff $l_i(g') \in G_i$.
- The labelling relation $L \subseteq AP \times W$ is equivalent to the valuation relation h

Formulae can be interpreted in M_{DIS} in a standard way [7,3,10]. Let $\pi = (w_0, w_1, \ldots)$ be an infinite sequence of worlds such that for all i, $R_t(w_i, w_{i+1})$, and let $\pi(i)$ denote the i-th world in the sequence (notice that, following standard conventions, we assume that the temporal relation is serial and thus all computation paths are infinite). We write $M_{DIS}, w \models \varphi$ when a formula φ is true at a world w in a Kripke model M_{DIS}, associated with a deontic interpreted system DIS. Satisfaction is defined as follows.

$M_{DIS}, w \models p$ iff $(p, w) \in L$,

$M_{DIS}, w \models \neg\varphi$ iff $M_{DIS}, w \not\models \varphi$,

$M_{DIS}, w \models \varphi_1 \vee \varphi_2$ iff either $M_{DIS}, w \models \varphi_1$ or $M_{DIS}, w \models \varphi_2$,

$M_{DIS}, w \models EX\varphi$ iff there exists π such that $\pi(0) = w$, and $M_{DIS}, \pi(1) \models \varphi$,

$M_{DIS}, w \models EG\varphi$ iff there exists a path π such that $\pi(0) = w$, and $M_{DIS}, \pi(i) \models \varphi$ for all $i \geq 0$,

$M_{DIS}, w \models E(\varphi U \psi)$ iff there exists a path π such that $\pi(0) = w$, and there exists $k \geq 0$ such that $M_{DIS}, \pi(k) \models \psi$, and $M_{DIS}, \pi(j) \models \varphi$ for all $0 \leq j < k$,

$M_{DIS}, w \models K_i\varphi$ iff for all $w' \in W$, $w \sim_i w'$ implies $M_{DIS}, w' \models \varphi$,

$M_{DIS}, w \models O_i\varphi$ iff for all $w' \in W$, $R_i^O(w, w')$ implies $M_{DIS}, w' \models \varphi$,

$M_{DIS}, w \models \hat{K}_i^\Gamma\, \varphi$ iff for all $w' \in W$ and for all $j \in \Gamma$, $w \sim_i w'$ and $R_j^O(w, w')$ implies $M_{DIS}, w' \models \varphi$,

$M_{DIS}, w \models E_\Gamma\varphi$ iff for all $w' \in W$, $R_\Gamma^E(w, w')$ implies $M_{DIS}, w' \models \varphi$,

$M_{DIS}, w \models C_\Gamma\varphi$ iff for all $w' \in W$, $R_\Gamma^C(w, w')'$ implies $M_{DIS}, w' \models \varphi$,

$M_{DIS}, w \models D_\Gamma\varphi$ iff for all $w' \in W$, $R_\Gamma^D(w, w')'$ implies $M_{DIS}, w' \models \varphi$

In the definition above, the relation R_Γ^E is defined as the union of the epistemic relations for the agents in Γ: $R_\Gamma^E = \bigcup_{i\in\Gamma} \sim_i$; the relation R_Γ^D is defined as the intersection of the epistemic relations for agents in Γ: $R_\Gamma^D = \bigcap_{i\in\Gamma} \sim_i$; the relation R_Γ^C is the transitive closure of R_Γ^E. Other standard temporal modalities AX, EF, AF, AG, AU can be derived in a standard way [3].

We say that a formula φ is true in model and we write $M_{DIS} \models \varphi$ if $M_{DIS}, w \models \varphi$ for all $w \in W$. Similarly to [7], we say that a formula φ *is true in a deontic interpreted systems* DIS, and we write $DIS \models \varphi$, if $M_{DIS} \models \varphi$. Thus, *a formula is true in a deontic interpreted system if it is true in the associated Kripke model*.

2.2 Model checking techniques

The problem of **model checking** can be defined as establishing whether or not a model M satisfies a formula φ ($M \models \varphi$). Though M could be a model for any logic, traditionally the problem of building tools to perform model checking automatically

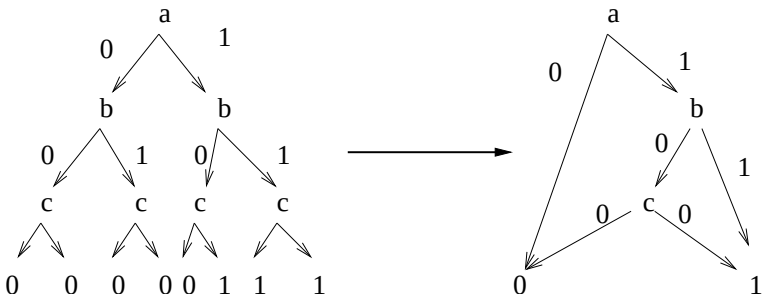

Fig. 1. OBDD representation for $a \wedge (b \vee c)$.

has been investigated mostly for *temporal* logics [3,13]. The model M is usually represented by means of a dedicated programming language, such as PROMELA [5] and SMV [14]. In many approaches, the model for the program is not built explicitly, but *symbolically*. Techniques to achieve this are based on ordered binary decision diagrams, SAT translations [15], or other algebraic structures. These approaches are often referred to as *symbolic model checking* techniques; other approaches exist, notably with automata [5]. For the purposes of this paper, we review briefly symbolic model checking using OBDDs.

OBDDs are an efficient representation for the manipulation of boolean functions. As an example, consider the boolean function $a \wedge (b \vee c)$. The truth table of this function would be 8 lines long. Equivalently, one can evaluate the truth value of this function by representing the function as a directed graph, as exemplified on the left-hand side of Figure 1. As it is clear from the picture, under certain assumptions, this graph can be simplified into the graph pictured on the right-hand side of Figure 1. This "reduced" representation is called the OBDD of the boolean function. Besides offering a compact representation of boolean functions, OBDDs of different functions can be composed efficiently. In [16] algorithms are provided for the manipulation and composition of OBDDs.

OBDDs are used in the verification of the model checking of systems specified by means of formulae of CTL, a logic used to reason about branching time [13]. Here states of the model and relations are represented by means of boolean formulae. A CTL formula is identified with a set of states: the states of the model satisfying the formula. As a set of states can be represented as a boolean formula, each CTL formula can be characterised by a boolean formula. Thus, the problem of model checking for CTL is reduced to the construction of boolean formulae. This is achieved by composing OBDDs, or by computing fix-points of operators on OBDDs; we refer to [13] for the details. Using this technique, systems with a state space in the region of 10^{40} have been verified. This technique will be extended in the next section to the verification of deontic interpreted systems.

3 Model checking deontic interpreted systems

In this section we present an algorithm for the verification of temporal, epistemic, and correctness modalities for MAS. Our approach is similar, in spirit, to the traditional model checking techniques for the logic CTL. Indeed, we start by representing the various parameters of a deontic interpreted system by means of boolean formulae. Then, we provide an algorithm based on this representation for the verification of formulae in the model associated with the deontic interpreted system.

Given a deontic interpreted system:

$$DIS = \left\langle \left(G_i, R_i, Act_i, P_i, t_i\right)_{i\in\Sigma}, \left(G_E, R_E, Act_E, P_E, t_E\right), I, h \right\rangle$$

note that the number $nv(i)$ of boolean variables required to encode the local states of an agent i is $nv(i) = \lceil log_2|L_i| \rceil$. Similarly, to encode an agent's action, the number $na(i)$ of boolean variables w_i required is $na(i) = \lceil log_2|Act_i| \rceil$. Thus, a global state g can be encoded as a boolean vector $(v_1, \ldots, v_N)$, where $N = \sum_i nv(i)$. A joint action $a = (a_1, \ldots, a_n, a_e) \in Act_1 \times \ldots \times Act_n \times Act_E$ can be encoded as a boolean vector $(w_1, \ldots, w_M)$, where [1] $M = \sum_i na(i)$. In turn, a boolean vector can be identified with a boolean formula, represented by a conjunction of literals, i.e. a conjunction of variables or their negation. In this way, a set of global states (or joint actions) can be expressed as the disjunction of the boolean formulae encoding each global state in the set.

Having encoded local states, global states, and actions by means of boolean formulae, all the remaining parameters can be expressed as boolean functions too. Indeed, since the protocols relate local states to sets of actions, they can also be expressed as boolean formulae. Similarly, the evolution functions can be translated into boolean formulae. The set of initial states is easily translated, while h can be translated into a boolean function which is true when a proposition is true in a given global state.

In addition to the parameters presented above, the algorithm for model checking presented below requires the definition of n boolean functions $R_i^K(g, g')$ (one for each agent) representing the epistemic accessibility relation, the definition of n boolean functions $R_i^O(g, g')$ representing the accessibility relations for the correctness operator, and the definition of a boolean function $R_t(g, g')$ representing the temporal transitions. Notice that we use the same symbols R_i^O and R_t to denote relations in $W \times W$ and boolean functions operating on boolean variables . The intended meaning should be clear from the context. The boolean function $R_t(g, g')$ can be obtained from the evolution functions t_i by quantifying over actions. This quantification can be translated into a propositional formula using a disjunction

[1] In this translation process the environment is treated as a standard agent.

(see [3] for a similar approach to boolean quantification):

$$R_t(g, g') = \bigvee_{a \in Act} [(t(g, a, g') \wedge P(g, a)]$$

where $P(g, a)$ is a boolean formula imposing that each component of the joint action a is consistent with the agents' protocols in the global state g. The above gives the desired boolean relation between global states. The set of *reachable* states is also needed by the algorithm: the set G of reachable global states can be expressed symbolically by a boolean formula, and it can be computed as the fix-point of the operator

$$\tau(Q) = (I(g) \vee \exists g'(R_t(g', g) \wedge Q(g'))$$

The fix-point of τ can be computed by iterating from $\tau(\emptyset)$ as standard (see [3]).

We now have all the ingredients in place to present the algorithm $SAT(\varphi)$ to compute the set of global states (expressed as a boolean formula) in which a formula φ holds, denoted by $[\![\varphi]\!]$. The following are input parameters for the algorithm:

- the boolean variables $(v_1, \ldots, v_N)$ and $(w_1, \ldots, w_M)$ encoding global states and joint actions;
- the boolean functions $P_i(v_1, \ldots, v_N, w_1, \ldots, w_M)$ encoding the protocols of the agents;
- the function $h(p)$ returning the set of global states in which the atomic proposition p holds. We assume that the global states are returned encoded as a boolean function of the variables $(v_1, \ldots, v_N)$;
- the set of initial states I, encoded as a boolean formula;
- the set of reachable states G, encoded as a boolean formula.
- the boolean function R_t encoding the temporal transition;
- n boolean functions encoding the accessibility relations R_i^K;
- n boolean functions encoding the accessibility relations R_i^O.

The algorithm is as follows:

$SAT(\varphi)$ {

φ is an atomic formula: return $h(\varphi)$;

φ is $\neg\varphi_1$: return $G \setminus SAT(\varphi_1)$;

φ is $\varphi_1 \wedge \varphi_2$: return $SAT(\varphi_1) \cap SAT(\varphi_2)$;

φ is $EX\varphi_1$: return $SAT_{EX}(\varphi_1)$;

φ is $E(\varphi_1 U \varphi_2)$: return $SAT_{EU}(\varphi_1, \varphi_2)$;

φ is $EG\varphi_1$: return $SAT_{EG}(\varphi_1)$;

φ is $K_i\varphi_1$: return $SAT_K(\varphi_1, i)$;

φ is $O_i\varphi_1$: return $SAT_O(\varphi_1, i)$;

φ is $\hat{K}_i^{\Gamma}\ \varphi_1$: return $SAT_{KH}(\varphi_1, i, \Gamma)$;

φ is $E_\Gamma\varphi_1$: return $SAT_E(\varphi_1, \Gamma)$;

φ is $D_\Gamma\varphi_1$: return $SAT_D(\varphi_1, \Gamma)$;

φ is $C_\Gamma\varphi_1$: return $SAT_C(\varphi_1, \Gamma)$;

}

In the algorithm above, SAT_{EX}, SAT_{EG}, SAT_{EU} are the standard procedures for CTL model checking [13], in which the temporal relation is R_t and, instead of temporal states, global states are considered. The procedures $SAT_K(\varphi, i)$, $SAT_O(\varphi, i)$, $SAT_{KH}(\varphi, i, \Gamma)$, $SAT_E(\varphi, \Gamma)$, $SAT_D(\varphi, \Gamma)$, and $SAT_C(\varphi, \Gamma)$ are defined using the appropriate accessibility relation. These procedures are presented below.

$SAT_K(\varphi, i)$ {

X = $SAT(\neg\varphi)$;

Y = $\{g \in G \mid \exists g' \in X$ s.t. $R_i^K(g, g')\}$

return $\neg$Y $\cap G$;

}

$SAT_O(\varphi, i)$ {

X = $SAT(\neg\varphi)$;

Y = $\{g \in G \mid \exists g' \in X$ s.t. $R_i^O(g, g')\}$

return $\neg$Y $\cap G$;

}

$SAT_{KH}(\varphi, i, \Gamma)$ {
X = $SAT(\neg\varphi)$;
Y = $\{g \in G \mid \exists g' \in X$ s.t. $R_i^K(g, g')$ and $R_j^O(g, g')$ for all $j \in \Gamma\}$
return $\neg$Y $\cap G$;
}

$SAT_E(\varphi, \Gamma)$ {
X = $SAT(\neg\varphi)$;
Y = $\{g \in G \mid \exists g' \in X$ s.t. $R_\Gamma^E(g, g')\}$
return $\neg$Y $\cap G$;
}

$SAT_D(\varphi, \Gamma)$ {
X = $SAT(\neg\varphi)$;
Y = $\{g \in G \mid \exists g' \in X$ s.t. $R_\Gamma^D(g, g')\}$
return $\neg$Y $\cap G$;
}

$SAT_C(\varphi, \Gamma)$ {
X = $SAT(\varphi)$;
Y = G;
while (X != Y) {
X = Y;
Y = $\{g \in G | \exists g' \in G$ s.t. $g' \in SAT(\varphi)$ and $g' \in X$ and $R_\Gamma^E(g, g')\}$
}
return Y;
}

The procedure $SAT_K(\varphi, i)$ operates by computing the set of global states X, cor-

responding the set of states in which the negation of φ holds. Then, the procedure computes the pre-image of this set with respect to the epistemic relation $\sim_i$ and returns the complement of this set with respect to the set of reachable states (this algorithm is based on the efficient implementation of the procedure to compute existential boolean quantifications; see [16,17]). The procedures $SAT_O(\varphi, i)$, $SAT_{KH}(\varphi, i, \Gamma)$, $SAT_E(\varphi, \Gamma)$, and $SAT_D(\varphi, \Gamma)$ implement a similar algorithm for the modalities $O_i, \hat{K}_i^\Gamma, E_\Gamma$, and D_Γ.

The procedure $SAT_C(\varphi, \Gamma)$ is based on the equivalence [7]

$$C_\Gamma \varphi \Leftrightarrow E_\Gamma(\varphi \wedge C_\Gamma \varphi)$$

which implies that $[\![C_\Gamma \varphi]\!]$ is the fix-point of the (monotonic) operator $\tau(Q) = [\![E_\Gamma(\varphi \wedge (Q))]\!]$. Hence, $[\![C_\Gamma \varphi]\!]$ can be obtained by iterating $\tau(G)$.

Notice that all the parameters can be encoded as OBDDs. Moreover, all the operations inside the algorithms can be performed on OBDDs.

The algorithm presented here computes the set of states in which a formula holds, but we are usually interested in checking whether or not a formula holds in the whole model. SAT can be used to verify whether or not a formula φ holds in a model by comparing two set of states: the set $SAT(\varphi)$ and the set of reachable states G. As sets of states are expressed as OBDDs, verification in a model is reduced to the comparison of the OBDDs for $SAT(\varphi)$ and for G.

4 Implementation

In this section we introduce MCMAS, a tool that implements the algorithms presented in Section 3. MCMAS is released under the terms of the GNU General Public License (GPL); the implementation is available for download [18].

In MCMAS, deontic interpreted systems are described by using the language ISPL (Interpreted Systems Programming Language). Figure 2 gives a short example of this language. We refer to the files available online [18] for the full syntax of ISPL. Formulae to be checked are provided at the end of the specification file, using an intuitive syntax.

Figure 3 lists the main components of MCMAS. Steps 2 to 6, inside the dashed box, are performed automatically upon invocation of the tool. These steps are coded mainly in C++. and can be summarised as follows:

- In step 2, the input ISPL file is parsed using standard tools. In this step various parameters are stored in temporary lists; such parameters include agents' names,

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

Fig. 2. ISPL example

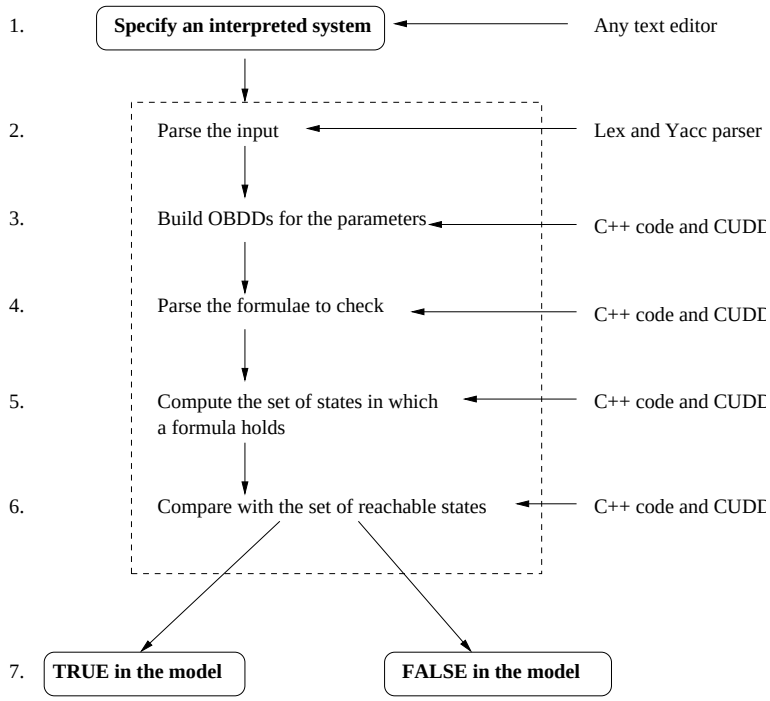

Fig. 3. Software structure

local states, actions, protocols, etc.

- In step 3, the lists obtained in step 2 are traversed to build the OBDDs for the verification algorithm. OBDDs are created and manipulated using the CUDD library [17]. In this step the number of variables needed to represent local states and actions are computed; following this, all the OBDDs are built by translating the boolean formulae for protocols, evolution functions, valuation, etc. Also, the set of reachable states is computed using the operator τ presented in Section 3.
- In step 4, the formulae to be checked are read from a text file, and parsed appro-

priately.
- In step 5, verification is performed by running the algorithm of Section 3. At the end of step 5, an OBDD representing the set of states in which a formula holds is computed.
- In step 6, the OBDD for the set of reachable states is compared with the OBDD corresponding to each formula. If the two are equivalent, the formula holds in the model and the tools produce a positive output. If the two are not equivalent, the tool produces a negative output.

MCMAS can be run from the command line, and accepts various options to modify verbosity, to inspect OBDDs statistics and memory usage, to enable variable reordering in the OBDDs (see [17]), etc. These options can be used to determine the "critical" points, and to fine tune the performance of the tool.

MCMAS is written in C/C++ and it has been successfully compiled on various platforms, including PowerPC (Mac OS X 10.2 and 10.3), Intel (various Pentium versions using Linux 2.4 and 2.6), and SPARC (SunOS 5.8 and 5.9). The source code has been compiled with gcc/g++ from version 2.95 till version 3.4.

5 Examples

In this section we exemplify and evaluate MCMAS by means of two examples: the bit transmission problem and the protocol of the dining cryptographers.

5.1 The bit transmission problem with faults

In the bit-transmission problem [7] a *sender* S wants to send the value of a bit to a *receiver* R, by using an unreliable communication channel. In this example, the channel may drop messages, but cannot tamper messages. One protocol for achieve communication is as follows. S immediately starts sending the bit to R, and continues to do so until it receives an acknowledgement from R. R does nothing until it receives the bit; from then on, it sends messages acknowledging the receipt to S. S stops sending the bit to R when it receives the first acknowledgement from R.

This scenario is extended in [8] to deal with failures. In particular, here we assume that R may fail to behave as intended. There are different kinds of faults that we can consider for R. Following [8], we discuss two examples; in the first, R may fail to send acknowledgements when it receives a message. In the second, R may send acknowledgements even if it has not received any message.

5.1.1 Deontic interpreted systems for the bit transmission problem

It is possible to represent the scenario described above by means of the formalism of deontic interpreted systems to reason about the correct behaviour of the components, as presented in [8]. To this end, a third agent agent called E (environment) is introduced to model the unreliable communication channel. The local states of the environment record the possible combinations of messages that have been sent in a round, either by S or R. Hence, four possible local states are taken for the environment: $L_E = \{(.,.), (sendbit, .), (., sendack), (sendbit, sendack)\}$, where '.' represents configurations in which no message has been sent by the corresponding agent. The actions Act_E for the environment correspond to the transmission of messages between S and R on the unreliable communication channel. It is assumed that the communication channel can transmit messages in both directions simultaneously, and that a message travelling in one direction can get through while a message travelling in the opposite direction is lost. The set of actions Act_E for the environment can be taken as $Act_E = \{S{-}R,\ S{\rightarrow},\ {\leftarrow}R,\ -\}$. The action $S{-}R$ represents the action in which the channel transmits any message successfully in both directions. The action $S{\rightarrow}$ represents a successful communication from S to R but unsuccessful from R to S. The action ${\leftarrow}R$ represents a successful communication from R to S but unsuccessful from S to R. Finally, the action $-$ represents the environment stopping messages in either direction. We assume the following constant function for the protocol of the environment P_E:

$$P_E(l_E) = Act_E = \{S{-}R,\ S{\rightarrow},\ {\leftarrow}R,\ -\}, \quad \text{for all } l_E \in L_E.$$

The evolution function for E records simply the actions of Sender and Receiver.

We model sender S by considering the set $L_S = \{0, 1, (0, ack), (1, ack)\}$ consisting of four possible local states. They represent the value of the bit S is attempting to transmit, and whether or not S has received an acknowledgement from R.

The set of actions Act_S for S can be taken as $Act_S = \{sendbit(0), sendbit(1), \lambda\}$; they represent the action of sending a bit of value 0, the action of sending a bit of value 1, and the null action.

The protocol for S is defined as follows:

$$P_S(0) = sendbit(0), \quad P_S(1) = sendbit(1),$$
$$P_S((0, ack)) = P_S((1, ack)) = \lambda.$$

The transition conditions for S are listed in Table 1.

We now consider two possible faulty behaviours for R.

Final state	**Transition condition**
$(0, ack)$	($L_S = 0$ and $Act_R = sendack$ and $Act_E = S{-}R$) or ($L_S = 0$ and $Act_R = sendack$ and $Act_E = {\leftarrow}R$)
$(1, ack)$	($L_S = 1$ and $Act_R = sendack$ and $Act_E = S{-}R$) or ($L_S = 1$ and $Act_R = sendack$ and $Act_E = {\leftarrow}R$)

Table 1
Transition conditions for S.

Final state	**Transition condition**
0	($Act_S = sendbit(0)$ and $L_R = \epsilon$ and $Act_E = S{-}R$) or ($Act_S = sendbit(0)$ and $L_R = \epsilon$ and $Act_E = S{\rightarrow}$)
1	($Act_S = sendbit(1)$ and $L_R = \epsilon$ and $Act_E = S{-}R$) or ($Act_S = sendbit(1)$ and $L_R = \epsilon$ and $Act_E = S{\rightarrow}$)
$(0, f)$	$L_R = 0$ and $Act_R = \epsilon$
$(1, f)$	$L_R = 1$ and $Act_R = \epsilon$

Table 2
Transition conditions for R.

Faulty receiver – 1: In this case we assume that R may fail to send acknowledgements when it is supposed to. To this end, we introduce the following local states for R: $L'_R = \{0, 1, \epsilon, (0, f), (1, f)\}$. The state ϵ is used to record the fact that in the run R has not received any message from S yet; 0 and 1 denote the value of the bit received. The local states (i, f) ($i = \{0, 1\}$) are *faulty* or *red* states denoting that, at some point in the past, R received a bit but failed to send an acknowledgement.

We model the set of allowed actions for R as $Act_R = \{sendack, \lambda\}$ and its protocol for R as:

$$P'_R(\epsilon) = \lambda, P'_R(0) = P'_R(1) = \{sendack, \lambda\},$$
$$P'_R((0, f)) = P'_R((1, f)) = \{sendack, \lambda\}.$$

The transition conditions for R are listed in Table 2.

Faulty receiver – 2: In this second case we assume that R may send acknowledgements without having received a bit first. We model this scenario with the following set of local states L''_R for R:

$$L''_R = \{0, 1, \epsilon, (0, f), (1, f), (\epsilon, f)\}.$$

The meaning of the local states $\epsilon, 0, 1, (0, f)$ and $(1, f)$ is as above; (ϵ, f) is a further *faulty* state corresponding to the fact that, at some point in the past, R sent an acknowledgement without having received a bit first. The actions allowed are the same as in the previous example. The protocol is defined as follows:

$$P''_R(\epsilon) = \{sendack, \lambda\},$$
$$P''_R(0) = P''_R(1) = sendack,$$
$$P''_R((0, f)) = P''_R((1, f)) = P''_R((\epsilon, f)) = \{sendack, \lambda\}.$$

The evolution function is a simple extension of Table 2.

For more details of both cases we refer to [8].

For both examples, we introduce the following atomic propositions: $AP = \{\mathbf{bit = 0}, \mathbf{bit = 1}, \mathbf{recbit}, \mathbf{recack}\}$. Correspondingly, we introduce the following valuation function:

$$\begin{aligned}
h(\mathbf{bit = 0}) &= \{g \in G \mid \text{ either } l_S(g) = 0 \text{ or } l_S(g) = (0, ack)\} \\
h(\mathbf{bit = 1}) &= \{g \in G \mid \text{ either } l_S(g) = 1 \text{ or } l_S(g) = (1, ack)\} \\
h(\mathbf{recbit}) &= \{g \in G \mid \text{ either } l_R(g) = 1, \text{ or } l_R(g) = 0, \\
&\qquad \text{or } l_R(g) = (0, f) \text{ or } l_R(g) = (1, f)\} \\
h(\mathbf{recack}) &= \{g \in G \mid l_S(g) = (1, ack) \text{ or } l_S(g) = (0, ack)\}
\end{aligned}$$

The parameters above describe two deontic interpreted systems, one for each faulty behaviour of R; we refer to these deontic interpreted systems with DIS_{BTP1} and DIS_{BTP2}.

Given the set AP above, we can evaluate various properties of DIS_{BTP1} and DIS_{BTP2} hold. For example, consider the following temporal and epistemic specifications:

$$AG(\mathbf{recack} \rightarrow (K_S(K_R(\mathbf{bit = 0}) \vee K_R(\mathbf{bit = 1})))) \quad (1)$$
$$AG(\mathbf{recack} \rightarrow (\hat{K}_S^R(K_R(\mathbf{bit = 0}) \vee K_R(\mathbf{bit = 1})))) \quad (2)$$

Formula 1 captures the fact that it is always true that, upon receipt of an acknowledgement, S knows that R knows the value of the bit. Formula 2 expresses a similar concept, but by using knowledge under the assumption of correct behaviour. In Section 5.3 we will verify in an automatic way that Formula 1 holds in DIS_{BTP1} but not in DIS_{BTP2}. This means that the faulty behaviour of R in DIS_{BTP1} does not affect the key property of the system. On the contrary, Formula 2 holds in both DIS_{BTP1}

and DIS_{BTP2}; hence, a particular form of knowledge is retained irrespective of the fault under consideration.

5.2 The protocol of the dining cryptographers

The protocol of the dining cryptographers was introduced in [19], and model checking of its properties was discussed in [20]. We report the original wording from [19].

"*Three cryptographers are sitting down to dinner at their favourite three-star restaurant. Their waiter informs them that arrangements have been made with the maitre d'hotel for the bill to be paid anonymously. One of the cryptographers might be paying for the dinner, or it might have been NSA (U.S. National Security Agency). The three cryptographers respect each other's right to make an anonymous payment, but they wonder if NSA is paying. They resolve their uncertainty fairly by carrying out the following protocol:*

Each cryptographer flips an unbiased coin behind his menu, between him and the cryptographer on his right, so that only the two of them can see the outcome. Each cryptographer then states aloud whether the two coins he can see – the one he flipped and the one his left-hand neighbour flipped – fell on the same side or on different sides. If one of the cryptographers is the payer, he states the opposite of what he sees. An odd number of differences uttered at the table indicates that a cryptographer is paying; an even number indicates that NSA is paying (assuming that the dinner was paid for only once). Yet if a cryptographer is paying, neither of the other two learns anything from the utterances about which cryptographer it is."[19]

The aim of this protocol is to allow anonymous broadcasting of messages. Notice that the same protocol works for any number of cryptographers greater or equal to three (see [19]).

For the purposes of this paper, we consider a variation of the protocol in which we assume that the first cryptographer may be faulty.

5.2.1 Deontic interpreted system for the dining cryptographers

We introduce three agents C_i ($i = \{1, 2, 3\}$) to model the three cryptographers, and one agent E for the environment. In our representation the environment is used to select non-deterministically the identity of the payer and the results of the coin tosses. This makes a total of 32 possible local states for the environment. We assume that the environment can perform only one action, the null action. Therefore, the protocol is simply mapping every local state to the null action. Also, there is no evolution of the local states for the environment. We model the local states of the

cryptographers as a string containing three parameters representing, respectively, whether or not the coins that a cryptographer can see are equal, whether or not the cryptographer is the payer, and the number of "different" utterances reported. Considering that all these parameters are not initialised at the beginning of the run, there are 27 possible combinations of these, hence 27 possible local states are required for every agent. For each cryptographer, the actions allowed are "say nothing", "say equal", "say different", and these actions are performed in compliance with the protocol stated above. We refer to the ISPL code available online for the details of the protocol and of the evolution function.

We define the following set of atomic propositions to reason about this scenario: $AP = \{\mathbf{paid_1}, \mathbf{paid_2}, \mathbf{paid_3}, \mathbf{even}, \mathbf{odd}\}$ and consider the following valuation function:

$$
\begin{aligned}
h(\mathbf{paid_1}) &= \{g \in G \mid l_{C_1}(g) = \langle *\text{Paid}* \rangle\} \\
h(\mathbf{paid_2}) &= \{g \in G \mid l_{C_2}(g) = \langle *\text{Paid}* \rangle\} \\
h(\mathbf{paid_3}) &= \{g \in G \mid l_{C_2}(g) = \langle *\text{Paid}* \rangle\} \\
h(\mathbf{even}) &= \{g \in G \mid l_{C_i}(g) = \langle *\text{Even}* \rangle \text{ for every } i\} \\
h(\mathbf{odd}) &= \{g \in G \mid l_{C_i}(g) = \langle *\text{Odd}* \rangle \text{ for every } i\}
\end{aligned}
$$

$\langle *\text{Paid}* \rangle$ denotes a local state in which the string contains the value Paid (i.e. the cryptographer paid for dinner). $\langle *\text{Even}* \rangle$ and $\langle *\text{Odd}* \rangle$ are defined similarly. We can now express formally various properties of this deontic interpreted system, denoted with DIS_{DC1}. For example:

$$
\begin{aligned}
DIS_{DC1} \models (\mathbf{odd} \wedge \neg \mathbf{paid_1}) \rightarrow AX(K_{C1}(\mathbf{paid_2} \vee \mathbf{paid_3}) \wedge \neg K_{C1}(\mathbf{paid_2}) \\
\wedge \neg K_{C1}(\mathbf{paid_3}))
\end{aligned}
$$

This formula expresses the claim made at the beginning of this section: if the first cryptographer did not pay for dinner and the number of "different" utterances is odd, then the first cryptographer knows that either the second or the third cryptographer paid for dinner; moreover, in this case, the first cryptographer does not know which of these two is the payer. Analogously, it is possible to check that, if a cryptographer paid for dinner, then there will be an odd number of "different" utterances, that is:

$$
DIS_{DC1} \models (\mathbf{paid_1} \vee \mathbf{paid_2} \vee \mathbf{paid_3}) \rightarrow AF(\mathbf{odd})
$$

Consider now the group Γ of the three cryptographers. An interesting property to

check is the following:

$$DIS_{DC1} \models \mathbf{even} \rightarrow AX(C_\Gamma(\neg\mathbf{paid_1} \wedge \neg\mathbf{paid_2} \wedge \neg\mathbf{paid_3}))$$

This formula expresses the fact that, in presence of an even number of "different" utterances, it is common knowledge that none of the cryptographers paid for the dinner. Hence, in this protocol common knowledge can be achieved anonymously.

Finally, we consider the case where C_1 may not follow its protocol, and say the opposite of what he should. This faulty behaviour may be described by another deontic interpreted system, denoted with DIS_{DC2}. We do not give the description of DIS_{DC2} explicitly; it is similar, in spirit, to DIS_{BTP2} and the code is available online. In this case we have that:

$$DIS_{DC2} \not\models (\mathbf{odd} \wedge \mathbf{paid_2}) \rightarrow AX(K_2(\mathbf{paid_1} \vee \mathbf{paid_3}) \wedge \neg K_2(\mathbf{paid_1}) \wedge \neg K_2(\mathbf{paid_3}))$$

However, if C_2 assumes that C_1 and C_3 are working correctly, the following formula is true:

$$DIS_{DC2} \models (\mathbf{odd} \wedge \mathbf{paid_2}) \rightarrow AX(\hat{K}_2^{\{1,3\}}(\mathbf{paid_1} \vee \mathbf{paid_3}) \wedge \neg\, \hat{K}_2^{\{1,3\}}(\mathbf{paid_1}) \wedge \neg\, \hat{K}_2^{\{1,3\}}(\mathbf{paid_3}))$$

All the above formulae were correctly verified by the tool.

5.3 Experimental results

In this section we present the experimental results obtained with MCMAS for the verification of the examples in Section 5.1 and 5.2. To evaluate the performance of the tool, we analyse space and time requirements. Following standard conventions, we define the size of a deontic interpreted system as $|DIS| = |S| + |R|$, where $|S|$ is the size of the state space and $|R|$ is the size of the relations. In our case, we define $|S|$ as the number all the possible combinations of local states and actions.

5.3.1 Experimental results for the bit transmission problem

We have encoded the deontic interpreted systems and the formulae introduced in Section 5.1 in ISPL. Appendix 1 reports a coding of DIS_{BTP1} in ISPL. MCMAS correctly reported DIS_{BTP1} as satisfying both formulae, and DIS_{BTP2} as not satisfying Formula (1), while satisfying Formula (2).

$\|M\|$	OBDDs **nodes**	**Memory (MBytes)**
$\approx 4 \cdot 10^6$	$\approx 10^3$	≈ 4.5

Table 3
Memory requirements for the bit transmission problem.

Model construction	**Verification**	**Total**
0.045sec	$<$0.01sec	0.05sec

Table 4
Running time (for one formula) for the bit transmission problem.

In this example there are 4 local states and 3 actions for S, 5 (or 6) local states and 2 actions for R, and 4 local states and 4 actions for E. In total, we have $|S| \approx 2 \cdot 10^3$. To define $|R|$ consider the sum of the temporal, the epistemic and the deontic relations. We approximate $|R|$ as $|S|^2$, hence $|M| = |S| + |R| \approx |S|^2 \approx 4 \cdot 10^6$.

To quantify the memory requirements we consider the maximum number of nodes allocated to the OBDDs. Notice that this figure over-estimates the number of nodes required to encode the state space and the relations. Further, we report the total memory used by the tool (in MBytes). The formulae of both DIS_{BTP1} and DIS_{BTP2} required a similar amount of memory and nodes. The average experimental results are reported in Table 3.

In addition to space requirements, we carried out some tests on time requirements. The running time is the sum of the time required for building all the OBDDs for the parameters and the actual running time for the verification. We ran the tool on a 1.2 GHz AMD Athlon with 256 MBytes of RAM, running Debian Linux with kernel 2.4.20. The average results are listed in Table 4.

5.3.2 Experimental results for the protocol of the dining cryptographers

We have encoded the interpreted systems introduced in Section 5.2 in ISPL (a copy of the code is included in the downlodable files). It is not difficult to extend the description of the system to a number of cryptographers greater than three. In this section we take advantage of this fact to perform an evaluation of the scalability of MCMAS.

Similarly to the previous section, we define the size of the model as $|M| = |S| + |R|$. We tested the formulae presented in Section 5.2.1 (more tests can be found in [18]); they were all correctly verified. The experimental results for memory and time requirements are reported in Tables 5 and 6.

N.Crypt.	$\|M\|$	OBDDs nodes	Memory (MBytes)
3	$\approx 7 \cdot 10^{13} (46)$	$\approx 10^4$	≈ 4.4
4	$\approx 2 \cdot 10^{18} (62)$	$\approx 6 \cdot 10^4$	≈ 5.2
5	$\approx 2 \cdot 7.5^{22} (76)$	$\approx 8 \cdot 10^4$	≈ 5.6
6	$\approx 1.2 \cdot 10^{27} (90)$	$\approx 1.6 \cdot 10^5$	≈ 7.1
7	$\approx 2 \cdot 10^{31} (104)$	$\approx 1.7 \cdot 10^5$	≈ 7.5
8	$\approx 1.3 \cdot 10^{36} (120)$	$\approx 1.2 \cdot 10^7$	≈ 230

Table 5
Memory requirements for the protocol of the dining cryptographers.

N.Crypt.	Model construction	Verification	Total
3	1.1sec	0.1sec	1.2sec
4	5.1	0.1	5.2
5	18.7	0.1	18.8
6	125.9	0.2	126.2
7	649	0.1	649
8	9643	1	9644

Table 6
Running time (for one formula) for the protocol of the dining cryptographers.

6 Related work and discussion

Various ideas have previously been put forward to verify MAS. In [21], M. Wooldridge et al. present the MABLE language for the specification of MAS. In this work, non-temporal modalities are translated into nested data structures (in the spirit of [22]). A similar approach can be found in Bordini et al. [23]: in this work, a modified version of the AgentSpeak(L) language [24] is used to specify agents and to exploit existing model checkers. Both the works of M. Wooldridge et al. and of Bordini et al. use the temporal model checker SPIN to perform an automatic verification.

The works of van der Meyden and Shilov [25], and van der Meyden and Su [20], are concerned with verification of interpreted systems. They consider the verification of a particular class of interpreted systems, namely the class of synchronous systems with perfect recall. An algorithm for model checking is introduced in the first paper using automata, and [20] suggests the use of ordered binary decision diagrams (OBDDs) for this approach. Recently, the tool MCK [26] has been developed to implement the techniques of [20].

Another line of research is concerned with SAT-based techniques for the verifi-

cation of multi-modal logics that model MAS. An algorithm for bounded model checking a subset of CTLK (a logic that augments the standard CTL with epistemic modalities) is introduced in [27] and an implementation is provided in [28], while [29] extends [27] by considering modalities to reason about the "correct behaviour" of agents in the formalism of deontic interpreted systems. An algorithm for unbounded model checking of the full language of CTLK is introduced in [30].

This paper differs from the works above in various respects. Differently from [21,23], instead of relying on existing model checkers, we extend the algorithms that appeared in [9] and we present an implementation of the algorithms to verify properties of MAS that is self-contained. We argue that it is more natural to express properties such as *knowledge* and *correct behaviour* directly, instead of translating a MAS system (and its properties) into temporal structures.

Our implementation uses OBDDs and, in this respect, our work differs from all the SAT-based approaches. Although [26] does use OBDDs, it restricts the verification to a particular class of interpreted systems, and does not consider operators to reason about the "correct behaviour" of agents, as we do here.

Unfortunately, due to the difference in semantics for MAS, it is currently impossible to compare MCMAS's performance with other model checkers for MAS (such as MCK [26] and Verics [28,31]) on common examples. Conventional model checkers for temporal logics have been employed to verify MAS scenarios, by translating MAS specifications into pure-temporal specifications [21]. Performance comparisons between MCMAS and other temporal-only model checkers is thus limited to simple examples, such as the bit transmission problem, for which it is possible to provide a "temporal" translation. However, the small size of the example does not allow for meaningful results. The translation of the protocol of the dining cryptographer (with a number of cryptographers greater than 5) would allow for better comparisons, but its feasibility is limited by the huge size of the temporal description for such examples.

Though we lack meaningful comparisons, the results presented in Section 5.3 show that MCMAS allows for the verification on examples whose size would be intractable with non-symbolic model checkers. For the case of the dining cryptographers, the difference in performance between 7 and 8 cryptographers shown in Table 6 is caused by the failure of the OBDD library in finding a compact order for the boolean variables in a reasonable time. This, in turn, causes MCMAS to store larger data structures, which consume considerably more processing time.

7 Conclusion

In this paper we have extended a major verification technique for reactive systems — symbolic model checking via OBDDs — to verify non-temporal properties of multi-agent systems. We presented an OBDD-based technique for the verification of MAS modelled in the formalism of deontic interpreted systems, a programming language for deontic interpreted systems (ISPL), and a tool implementing these ideas (MCMAS).

We tested our implementation by means of two examples: the bit transmission problem with faults and the protocol of the dining cryptographers. These examples suggested that our framework can be employed successfully in the analysis of communication and security protocols.

The scalability results from the second examples are encouraging, and we believe that this verification methodology may be applied in various real-life scenarios.

References

[1] M. Wooldridge, Reasoning about Rational Agents, MIT Press, 2000.

[2] M. Wooldridge, An introduction to multi-agent systems, John Wiley, England, 2002.

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

[4] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, L. J. Hwang, Symbolic model checking: 10^{20} states and beyond, Information and Computation 98 (2) (1992) 142–170.

[5] G. J. Holzmann, The model checker SPIN, IEEE transaction on software engineering 23 (5).

[6] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, A. Tacchella, NUSMV2: An open-source tool for symbolic model checking, in: Proceedings of the 14th International Conference on Computer Aided Verification (CAV'02), Vol. 2404 of LNCS, Springer-Verlag, 2002, pp. 359–364.

[7] R. Fagin, J. Y. Halpern, Y. Moses, M. Y. Vardi, Reasoning about Knowledge, MIT Press, Cambridge, 1995.

[8] A. Lomuscio, M. Sergot, A formalisation of violation, error recovery, and enforcement in the bit transmission problem, Journal of Applied Logic 2 (1) (2004) 93–116.

[9] F. Raimondi, A. Lomuscio, Automatic verification of deontic interpreted systems by model checking via OBDDs, in: Proceedings of the Sixteenth European Conference on Artificial Intelligence (ECAI04), IOS PRESS, 2004, pp. 53–57.

[10] A. Lomuscio, M. Sergot, Deontic interpreted systems, Studia Logica 75 (1) (2003) 63–92.

[11] B. Woźna, A. Lomuscio, W.Penczek, Bounded model checking for knowledge and real time, in: Proceedings of the 4th International Joint Conference on Autonomous Agents and Multi-agent systems (AAMAS'05), ACM Press, 2005.

[12] A. Lomuscio, Knowledge sharing among ideal agents, Ph.D. thesis, School of Computer Science, University of Birmingham, Birmingham, UK (June 1999).

[13] M. R. A. Huth, M. D. Ryan, Logic in Computer Science: Modelling and Reasoning about Systems, Cambridge University Press, Cambridge, England, 2000.

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

[15] A. Biere, A. Cimatti, E. Clarke, Y. Zhu, Symbolic model checking without BDDs, in: Proc. of TACAS'99, Vol. 1579 of LNCS, Springer-Verlag, 1999, pp. 193–207.

[16] R. E. Bryant, Graph-based algorithms for boolean function manipulation, IEEE Transactions on Computers 35 (8) (1986) 677–691.
URL `http://www.cs.cmu.edu/ bryant/pubdir/ieeetc86.ps`

[17] F. Somenzi, CUDD: CU decision diagram package - release 2.4.0, http://vlsi.colorado.edu/∼fabio/CUDD/cuddIntro.html.

[18] F. Raimondi, A. Lomuscio, MCMAS - A tool for verification of multi-agent systems, http://www.cs.ucl.ac.uk/staff/f.raimondi/MCMAS/.

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

[20] R. van der Meyden, K. Su, Symbolic model checking the knowledge of the dining cryptographers, in: 17th IEEE Computer Security Foundations Workshop, 2004, pp. 280–291.

[21] M. Wooldridge, M. Fisher, M. Huget, S. Parsons, Model checking multiagent systems with MABLE, in: Proceedings of the First International Conference on Autonomous Agents and Multiagent Systems (AAMAS-02), Bologna, Italy, 2002.

[22] M. Benerecetti, F. Giunchiglia, L. Serafini, Model checking multiagent systems, Journal of Logic and Computation 8 (3) (1998) 401–423.

[23] R. H. Bordini, M. Fisher, C. Pardavila, M. Wooldridge, Model checking AgentSpeak, in: J. S. Rosenschein, T. Sandholm, W. Michael, M. Yokoo (Eds.), Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-agent systems (AAMAS-03), ACM Press, 2003, pp. 409–416.

[24] A. S. Rao, Decision procedures for propositional linear-time Belief-Desire-Intention logics, in: M. Wooldridge, J. P. Müller, M. Tambe (Eds.), Intelligent Agents II (LNAI 1037), Springer-Verlag: Heidelberg, Germany, 1996, pp. 33–48.

[25] R. v. Meyden, H. Shilov., Model checking knowledge and time in systems with perfect recall., in: Proceedings of Proc. of FST&TCS, Vol. 1738 of Lecture Notes in Computer Science, Hyderabad, India, 1999, pp. 432–445.

[26] P. Gammie, R. van der Meyden, Mck: Model checking the logic of knowledge, in: Proceedings of 16th International Conference on Computer Aided Verification (CAV'04), Vol. 3114 of LNCS, Springer-Verlag, 2004, pp. 479–483.

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

[28] VerICS, http://www.ipipan.waw.pl/staff/w.penczek/abmpw/index-ang.htm.

[29] B. Woźna, A. Lomuscio, W. Penczek, Bounded model checking for deontic interpreted systems, in: Proc. of the 2nd Workshop on Logic and Communication in Multi-Agent Systems (LCMAS'04), Vol. 126 of ENTCS, Elsevier, 2004, pp. 93–114.
URL `http://www.sciencedirect.com/science/journal/15710661`

[30] M. Kacprzak, A. Lomuscio, W. Penczek, From bounded to unbounded model checking for temporal epistemic logic, Fundamenta Informaticae 63 (2,3) (2004) 221–240.

[31] A. Lomuscio, T. Łasica, W. Penczek, Bounded model checking for interpreted systems: preliminary experimental results, in: M. Hinchey (Ed.), Proceedings of FAABS II, Vol. 2699 of LNCS, Springer Verlag, 2003.

Appendix 1: ISPL code for DIS_{BTP1}

```
Agent Sender
Lstate = {s0,s1,s0ack,s1ack};
Lgreen = {s0,s1,s0ack,s1ack};
Action = {sb0,sb1,nothing};
Protocol:
                s0: {sb0};
                s1: {sb1};
                s0ack: {nothing};
                s1ack: {nothing};
        end Protocol

Ev:
  s0ack if (  ( (Lstate=s0) and (Receiver.Action=sendack) and [...]
or ( (Lstate=s0) and (Receiver.Action=sendack) [...] ) );
          s1ack if (  ( (Lstate=s1) and (Receiver.Action=sendack) and [...]
    [...]
end Ev
end Agent

Agent Receiver
Lstate = {empty,r0,r1,r0f,r1f};
Lgreen = {empty,r0,r1};
Action = {nothing,sendack};
Protocol:
empty: {nothing};
r0: {sendack,nothing};
r1: {sendack,nothing};
r0f: {sendack,nothing};
r1f: {sendack,nothing};
end Protocol
Ev:
  r0 if ( ( (Sender.Action=sb0) and (Lstate=empty) and [...]
          r1 if ( ( (Sender.Action=sb1) and (Lstate=empty) and [...]
  r0f if ( (Lstate = r0 ) and (Action=nothing) );
  r1f if ( (Lstate = r1 ) and (Action=nothing) );
end Ev
end Agent

Agent Environment
        Lstate = {S,R,SR,none};
Lgreen = {S,R,SR,none};
        Action = {S,SR,R,none};
        Protocol:
                S: {S,SR,R,none};
                R: {S,SR,R,none};
                SR: {S,SR,R,none};
                none: {S,SR,R,none};
end Protocol
Ev:
  S if ( ( (Sender.Action=sb0) or (Sender.Action=sb1) ) and
       (Receiver.Action=nothing));
          SR if [...]
  [...]
end Ev
end Agent

Evaluation
        recbit if ( (Receiver.Lstate=r0) or (Receiver.Lstate=r1) or
     (Receiver.Lstate=r0f) or (Receiver.Lstate=r1f) );
        recack if ( (Sender.Lstate=s0ack) or (Sender.Lstate=s1ack) );
        bit0 if ( (Sender.Lstate=s0) or (Sender.Lstate=s0ack));
        bit1 if ( (Sender.Lstate=s1) or (Sender.Lstate=s1ack) );
end Evaluation
```

```
InitStates
( (Sender.Lstate=s0) or (Sender.Lstate=s1) ) and
( Receiver.Lstate=empty ) and
( Environment.Lstate=none );
end InitStates

Groups
g1 = {Sender,Receiver};
end Groups

Formulae
  AG(recack -> ( K(Sender,K(Receiver,bit0) or K(Receiver,bit1))));
  AG(recack -> ( KH(Sender,Receiver,K(Receiver,bit0) or K(Receiver,bit1))));
end Formulae
```