Automatic verification of deontic interpreted systems by model checking via OBDD's

Franco Raimondi[1] and **Alessio Lomuscio**[1]

Abstract. We present an algorithm for the verification of multiagent systems specified by means of a modal logic that includes a temporal, an epistemic, and a deontic operator. Verification is performed by model checking on OBDD's. We present an implementation of the algorithm and report on experimental results for the bit transmission problem with faults.

1 Introduction

In the last two decades, the paradigm of multiagent systems (MAS) has been employed successfully in several fields, including, for example, philosophy, economics, and software engineering. One of the reasons for the use of MAS formalism in such different fields 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.

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 abstract a real system by means of the formalism of MAS, how can we *verify formally* that the system complies with certain desired properties? Formal verification has been investigated successfully in the field of software engineering. Typically, in software engineering we want to verify whether or not a system behaves as it is supposed to. One of the most successful formal approaches to verification is *model checking*. In this approach, the system S to be verified is represented by means of a logical model M_S representing the computational traces of the system, and the 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, and many real-life scenarios have been tested.

Unfortunately, extending model checking techniques for the verification of MAS does not seem 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 is "simply" a *temporal* model, while MAS need more complex formalisms. Typically, in MAS we want to reason about epistemic, intentional, and doxastic properties of agents, and their temporal evolution. Hence, the logical models required are richer than the temporal model used in traditional model checking.

Various ideas have been put forward to verify MAS. In [18], 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 [1]). Bordini et al. [2] use a modified version of the AgentSpeak(L) language [17] to specify agents and to exploit existing model checkers. Both the works of M. Wooldridge et al. and of Bordini et al. translate the specification into a SPIN specification on which the verification step is performed by means of existing tools. Effectively, the attitudes for the agents are reduced to predicates, and the verification involves only the temporal verification of those. In [8] a methodology is provided to translate a deontic interpreted system into SMV code, but the verification is limited to static deontic and epistemic properties, i.e. the temporal dimension is not present, and the approach is not fully symbolic. The works of van der Meyden and Shilov [12], and van der Meyden and Su [13], are concerned with verification of interpreted systems. They consider the verification of a particular class of interpreted systems, namely the class of synchronous distributed systems with perfect recall. An algorithm for model checking is introduced in the first paper using automata, and [13] suggests the use of OBDD's for this approach, but no algorithm or implementation is provided.

In this paper, instead of relying on existing model checkers, we build upon the algorithm presented in [16] to verify properties of MAS by means of model checking via OBDD's. In particular, in this work we investigate the verification of epistemic properties of MAS, and of the "correct" behaviour of agents. Verification of epistemic properties was presented in [16] and motivated by the long-standing need to represent epistemic and informational states of the agents. But in complex systems, reasoning about the "correct" behaviour (as opposed to behaviours that may be described as "faulty" or "unwanted") is also crucial. As an example, consider a client-server interaction in which a server fails to respond as quickly as it is supposed to to client's requests. This is an unwanted behaviour that may, in certain circumstances, crash the client. In MAS it is unfeasible to impose hard-wired constraints in the agents' behaviours to avoid all possible unwanted situations: more promising seems to be to allow for some non-critical faulty behaviour to happen and to verify what properties hold if this does or not does happen. The purpose of this paper is to present a technique and its implementation that allows for the automatic verification of properties expressing the correctness of behaviours of agents as well as their epistemic states.

The rest of the paper is organised as follows. In Section 2 we review the framework of deontic interpreted systems and model checking via OBDD's. In Section 3 we introduce an algorithm for the verification of deontic interpreted systems. An implementation of the algorithm is then discussed in Section 4. In Section 5 we test our implementation by means of an example: the bit transmission problem with faults. We conclude in Section 6.

[1] Department of Computer Science, King's College London London, UK email:{franco,alessio}@dcs.kcl.ac.uk. This research was partly supported by EPSRC (grant GR/S49353/01) and the Nuffield Foundation (grant NAL/00690/G).

2 Preliminaries

In this section we briefly summarise the formalism of interpreted systems as presented in [5] to model a MAS, and its extension to reason about correct behaviour as presented in [9]. After this, we briefly summarise the approach to model checking via OBDD's.

An **interpreted system** [5] is a semantic structure representing a system of agents. Each agent i ($i \in \{1, \ldots, n\}$) is characterised by a set of finite *local states* L_i and by a set of actions Act_i that may be performed. Actions are performed in compliance with a protocol $P_i : L_i \rightarrow 2^{Act_i}$. A tuple $g = (l_1, \ldots, l_n) \in L_1 \times \ldots, L_n$, where $l_i \in L_i$ for each i, is called a *global state* and gives a description at a particular instance of time of the system. Given a set I of *initial global states*, the evolution of the system is described by n evolution functions t_i (this definition is equivalent to the definition of a single evolution function t as in [5]): $t_i : L_1 \times \ldots \times L_n \times Act_1 \times \ldots \times Act_n \rightarrow L_i$. In this formalism, the environment in which agents "live" is usually modelled by means of a special agent E; we refer to [5] for more details. The set I, the functions t_i and the protocols P_i generate a set of *computations* (also called *runs*). Formally, a computation π is a sequence of global states $\pi = (g_0, g_1, \ldots)$ such that for each pair $(g_j, g_{j+1}) \in \pi$, there exists a set of actions a enabled by the protocols such that $t(g_j, a) = g_{j+1}$. $G \subseteq (L_1 \times \ldots \times L_n)$ denotes the set of *reachable* global states in the MAS.

In [9] the notion of *correct behaviour* of the agents is incorporated in this formalism. This is done by dividing the set of local states into two sets, a non-empty set G_i of allowed (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 $G_i \cap R_i = \emptyset$. Given a set of agents $A = \{1, \ldots, n\}$ with corresponding local states, protocols, and transition functions, a countable set of propositional variables $\mathcal{P} = \{p, q, \ldots\}$, and a valuation function for the atoms $\mathcal{V} : \mathcal{P} \rightarrow 2^G$, a deontic interpreted systems is a tuple $DIS = (G, I, \Pi, R_1^O, \ldots, R_n^O, R_1^K, \ldots, R_n^K, \mathcal{V})$. In the above G is the finite set of reachable global states for the system, $I \subseteq G$ is the set of initial states, and Π is the set of possible computations in the system. $R_i^O, i \in A$, is a relation between global states defined by $gR_i^O g'$ iff $l_i(g') \in G_i$, i.e. if the local state of i in g' is a "green" state. $R_i^K, i \in A$, is defined by $gR_i^K g'$ iff $l_i(g) = l_i(g')$, i.e. if the local state of agent i is the same in g and in g'. Some issues are related to the generation of the reachable states in the system given a set of protocols and transition relations; since they do not influence this paper we do not report them here.

Deontic interpreted systems can be used to evaluate formulae involving various modal operators. Besides the standard boolean connectives, the language considered in [9] includes:

- A deontic operator $O_i\varphi$ [9], denoting the fact that *under all the correct alternatives for agent i, φ holds.*
- An epistemic operator $K_i\varphi$ [5], whose meaning is *agent i knows φ.*
- A particular form of knowledge is also expressed via the operator $\widehat{K}_i^j$ [9]: this is the knowledge that an agent i has *on the assumption that agent j is functioning correctly.*

We extend this language by introducing the following temporal operators: $EX(\varphi), EG(\varphi), E(\varphi U \psi)$. Formally, the language we use is defined as follows:

$$\varphi ::= \quad p \mid \neg\varphi \mid \varphi \vee \varphi \mid EX\varphi \mid EG\varphi \mid E(\varphi U \varphi) \mid K_i\varphi \mid O_i\varphi \mid \widehat{K}_i^j \varphi$$

We now define the semantics for this language. Given a deontic interpreted system DIS, a global state g, and a formula φ, satisfaction is defined as follows:

$DIS, g \models p$	iff	$g \in \mathcal{V}(p)$,
$DIS, g \models \neg\varphi$	iff	$g \not\models \varphi$,
$DIS, g \models \varphi_1 \vee \varphi_2$	iff	$g \models \varphi_1$ or $g \models \varphi_2$,
$DIS, g \models EX\varphi$	iff	there exists a computation $\pi \in \Pi$ such that $\pi_0 = g$ and $\pi_1 \models \varphi$,
$DIS, g \models EG\varphi$	iff	there exists a computation $\pi \in \Pi$ such that $\pi_0 = g$ and $\pi_i \models \varphi \; \forall i \geq 0$,
$DIS, g \models E(\varphi U \psi)$	iff	there exists a computation $\pi \in \Pi$ such that $\pi_0 = g$ and a $k \geq 0$ such that $\pi_k \models \psi$ and $\pi_i \models \varphi$ for all $0 \leq i < k$,
$DIS, g \models K_i\varphi$	iff	$\forall g' \in G, gR_i^K g'$ implies $g' \models \varphi$
$DIS, g \models O_i\varphi$	iff	$\forall g' \in G, gR_i^O g'$ implies $g' \models \varphi$
$DIS, g \models \widehat{K}_i^j \varphi$	iff	$\forall g' \in G, gR_i^K g'$ and $gR_j^O g'$ implies $g' \models \varphi$

In the definition above, π_j denotes the global state at place j in computation π. Other temporal modalities can be derived, namely AX, EF, AF, AG, AU. We refer to [5, 9, 14] for more details.

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 has been investigated almost only for *temporal* logics [4, 7]. The model M is usually represented by means of a dedicated programming language, such as PROMELA[6] or SMV [11]. The verification step avoids building the model M explicitly from the program; instead, various techniques have been investigated to perform a *symbolic* representation of the model and the parameters needed by verification algorithms. Such techniques are based on automata [6], ordered binary decision diagrams (or OBDD's [3]), or other algebraic structures. These approaches are often referred to as *symbolic model checking* techniques. The key idea of model checking temporal logics using OBDD's is to represent the model M and all the parameters needed by the algorithms by means of boolean functions. These boolean functions can then be encoded as OBDD's, and the verification step can operate directly on these. The verification is performed using fixpoint characterisation of the temporal logics operators. We assume some familiarity with OBDD-based symbolic model checking, and we refer to [3, 11] for more details. Using this technique, systems with a state space in the region of 10^{40} have been verified.

3 Model checking deontic interpreted systems

In this section we present an algorithm for the verification of deontic, epistemic, and temporal modalities for MAS, extending the work that appeared in [16]. 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 the system by means of boolean formulae. Then, we provide and algorithm based on this representation for the verification step. The whole technique uses deontic interpreted systems as its underlying semantics.

As boolean formulae are built using boolean variables, we begin by computing the required number of boolean variables. To encode local states of an agent, the number of boolean variables required is $nv(i) = \lceil log_2 |L_i| \rceil$. To encode actions, the number of variables required is $na(i) = \lceil log_2 |Act_i| \rceil$. Hence, a global state $g = (v_1, \ldots, v_N)$ can be encoded by means of $N = \sum_i nv(i)$ boolean variables, and a joint action $a = (a_1, \ldots, a_M)$ can be encoded by means of $M = \sum_i na(i)$ boolean variables. Having encoded local states, global states, and actions by means of boolean variables, all the remaining parameters can be expressed as boolean

functions as follows. The protocols relate local states to set of actions, and can be expressed as boolean formulae. The evolution functions can be translated into boolean formulae, too. Indeed, the definition of t_i can be seen as specifying a list of *conditions* $c_{i,1}, \ldots, c_{i,k}$ under which agent i changes the value of its local state. Each $c_{i,j}$ relates conditions on global state and actions with the value of "next" local state for i. The evaluation function $\mathcal{V}$ associates a set of global states to each propositional atom, and so it can be translated into a boolean function.

In addition to these parameters, the algorithm presented below requires the definition of a boolean function $R_t(g, g')$ representing a temporal relation between g and g'. $R_t(g, g')$ can be obtained from the evolution functions t_i by quantifying over actions. The quantification over actions above can be translated into a propositional formula using a disjunction (see [11, 4] 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 the joint action a must be consistent with the agents' protocols in global state g and $t(g, a, g')$ is a "global" transition condition obtained from as a boolean function of the conditions t_i. R_t gives the desired boolean relation between global states.

We now present the algorithm SAT to compute the set of global states in which a formula φ holds. The following are input parameters of the algorithm:

- the boolean variables $(v_1, \ldots, v_N)$ and $(a_1, \ldots, a_M)$ to encode global states and joint actions;
- the boolean functions $P_i(v_1, \ldots, v_N, a_1, \ldots, a_M)$ to encode the protocols of the agents;
- the function $\mathcal{V}(p)$ 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 $(v_1, \ldots, v_N)$;
- the set of initial states I, encoded as a boolean function;
- the set of reachable states G. This can be computed as the fix-point of the operator $\tau = (I(g) \vee \exists g'(R_t(g', g) \wedge Q(g'))$ where $I(g)$ is true if g is an initial state and Q denotes a set of global states. The fix-point of τ can be computed by iterating $\tau(\emptyset)$ by standard procedure (see [11]);
- the boolean function R_t to encode the temporal transition;
- n boolean functions to encode the accessibility relations R_i^K (these functions are defined using equivalence on local states of G);
- n boolean functions to encode the accessibility relations R_i^O.

The algorithm is as follows:

$SAT(\varphi)$ {
  φ is an atomic formula: return $\mathcal{V}(\varphi)$;
  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 $\widehat{K}_i^j\ \varphi_1$: return $SAT_{KH}(\varphi_1, i, j)$;
}

In the algorithm above, SAT_{EX}, SAT_{EG}, SAT_{EU} are the standard procedures for CTL model checking [7], in which the temporal relation is R_t and, instead of temporal states, global states are considered. The procedures $SAT_K(\varphi, i)$, $SAT_O(\varphi, i)$ and $SAT_{KH}(\varphi, i, j)$ are defined using the appropriate accessibility relation and are presented below.

$SAT_K(\varphi, i)$ {
  X = $SAT(\neg\varphi)$;
  Y = $\{g \in G | K_i(g, g')$ and $g' \in X\}$
  return $\neg$Y;
}

$SAT_O(\varphi, i)$ {
  X = $SAT(\neg\varphi)$;
  Y = $\{g \in G | R_i^O(g, g')$ and $g' \in X\}$
  return $\neg$Y;
}

$SAT_{KH}(\varphi, \Gamma)$ {
  X = $SAT(\neg\varphi)$;
  Y = $\{g \in G | R_i(g, g')$ and $R_j^O(g, g')$ and $g' \in X\}$
  return $\neg$Y;
}

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

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 OBDD's, verification in a model is reduced to the comparison of the two OBDD's for $SAT(\varphi)$ and for G.

4 Implementation

In this section we introduce an implementation of the algorithm presented in Section 3. This extends to deontic states the tool presented in [16]. The implementation is available for download [15].

To define a deontic interpreted system we need to represent, for each agent:

- a list of local states, and a list of "green" local states;
- a list of actions;
- a protocol for the agent;
- an evolution function for the agent.

To complete the specification of a deontic interpreted system, it is also necessary to define the following parameters:

- an evaluation function;
- a set of initial states (expressed as a boolean condition);
- optionally, a set of groups for group modalities

In our implementation, the parameters listed above are provided via a text file. Due to space limitations we refer to the files available online for a full example of specification of an interpreted system.

5 An example: the bit transmission problem with faults

The bit-transmission problem involves two agents, a *sender* S, and a *receiver* R, communicating over a faulty communication channel.

The channel may drop messages but will not flip the value of a bit being sent. S wants to communicate some information (the value of a bit) to R. One protocol for achieving this 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 acknowledgements of receipt to S. S stops sending the bit to R when it receives an acknowledgement.

This scenario is extended in [10] to deal with failures. In particular, here we assume that R may fail or not behave as intended. There are different kind of faults that we can consider for R. Following [10], 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 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, as presented in [10, 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 L_E 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 is: $Act_E = \{S{-}R,\ S{\rightarrow},\ {\leftarrow}R,\ -\}$. "$S{-}R$" represents the action in which the channel transmits any message successfully in both directions, "$S{\rightarrow}$" that it transmits successfully from S to R but loses any message from R to S, "${\leftarrow}R$" that it transmits successfully from R to S but loses any message from S to R, and "$-$" that it loses any messages sent 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 the actions of Sender and Receiver.

We model sender S by considering 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: $L_S = \{0, 1, (0, ack), (1, ack)\}$. The set of actions Act_S for S is: $Act_S = \{sendbit(0), sendbit(1), \lambda\}$. 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, that we model below.

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 "$\epsilon$" is used to denote the fact that R did not receive any message from S; "0" and "1" denote the value of the received bit. The 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. The set of allowed actions for R is: $Act_R = \{sendack, \lambda\}$. The protocol for R is the following:

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

The transition conditions for R are listed in Table 2.

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 – 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 local states "ϵ", "0", "1", "$(0, f)$" and "$(1, f)$" are 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. The actions allowed are the same as in the previous example. The protocol is defined as follows:

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

The evolution function is a simple extension of Table 2.

For both examples, we introduce the following evaluation function:

$$\begin{aligned}
\mathcal{V}(\mathbf{bit = 0}) &= \{g \in G | l_S(g) = 0 \text{ or } l_S(g) = (0, ack)\} \\
\mathcal{V}(\mathbf{bit = 1}) &= \{g \in G | l_S(g) = 1 \text{ or } l_S(g) = (1, ack)\} \\
\mathcal{V}(\mathbf{recbit}) &= \{g \in G | l_R(g) = 1 \text{ or } l_R(g) = 0\} \\
\mathcal{V}(\mathbf{recack}) &= \{g \in G | l_S(g) = (1, ack) \text{ or } l_S(g) = (0, ack)\}
\end{aligned}$$

The evaluation function $\mathcal{V}$ and the parameters above generate two deontic interpreted systems, one for each faulty behaviour of R; we refer to these deontic interpreted systems as DIS_1 and DIS_2. In these systems we can evaluate various properties, for example:

$$AG(recack \rightarrow (K_S\big(K_R(\mathbf{bit = 0}) \vee K_R(\mathbf{bit = 1})\big))) \quad (1)$$

$$AG(recack \rightarrow (\widehat{K}^R_S\big(K_R(\mathbf{bit = 0}) \vee K_R(\mathbf{bit = 1})\big))) \quad (2)$$

Formula 1 above captures the fact that globally, upon receipt of an acknowledgement, S knows that R knows the value of the bit. Formula 2 expresses the same idea but using knowledge under the assumption of correct behaviour. In the next section we will verify in an automatic way that Formula 1 holds in DIS_1 but not in DIS_2. This means that the faulty behaviour of R in DIS_1 does not affect the key property of the system. On the contrary, Formula 2 holds in both DIS_1 and DIS_2; hence, a particular form of knowledge is retained irrespective of the fault.

5.2 Experimental results

We have encoded the deontic interpreted system and the formulae introduced in the previous section by means of a text file. The model checker here presented correctly reported DIS_1 as satisfying both formulae and DIS_2 not satisfying Formula (1) while satisfying Formula (2).

To evaluate the performance of the tool, we first analyse the space requirements. Following the 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. In the example above, 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, $|S| \approx 2 \cdot 10^3$. To define $|R|$ we must take into account that, in addition to the temporal relation, there are also the epistemic and deontic relations. Hence, we define $|R|$ as the sum of the sizes of temporal, epistemic, and 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 for OBDD's. Notice that this figure overestimates 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 examples required a similar amount of memory and nodes. The average experimental results are reported in Table 3.

$\|M\|$	OBDD's nodes	Memory (MBytes)
$\approx 4 \cdot 10^6$	$\approx 10^3$	≈ 4.5

Table 3. Memory requirements.

In addition to space requirements, we carried out some test on time requirements. The running time is the sum of the time required for building all the OBDD's 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.

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

Table 4. Running time (for one formula).

We see these as encouraging results. We have been able to check formulae with nested temporal, epistemic and deontic modalities in less than 0.1 seconds on a standard PC, for a non-trivial model. Also, the number of OBDD's nodes is orders of magnitude smaller than the size of the model.

6 Conclusion

In this paper we have extended a major verification technique for reactive systems — symbolic model checking via OBDD's — to verify non-temporal properties of multiagent systems. We provided an algorithm and its implementation, and we tested our implementation by means of an example: the bit transmission problem with faults. The results obtained are encouraging, and we estimate that our tool could be used in larger examples.

REFERENCES

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

[2] R. H. Bordini, M. Fisher, C. Pardavila, and M. Wooldridge, 'Model checking AgentSpeak', in *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'03)*, (July 2003).

[3] R. E. Bryant, 'Graph-based algorithms for boolean function manipulation', *IEEE Transaction on Computers*, 677–691, (August 1986).

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

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

[6] G. J. Holzmann, 'The model checker spin', *IEEE transaction on software engineering*, **23**(5), (May 1997).

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

[8] A. Lomuscio, F. Raimondi, and M. Sergot, 'Towards model checking interpreted systems', in *Proceedings of MoChArt*, Lyon, France, (August 2002).

[9] A. Lomuscio and M. Sergot, 'On multi-agent systems specification via deontic logic', in *Proceedings of ATAL 2001*, ed., J.-J Meyer. Springer Verlag, (July 2001). To Appear.

[10] A. Lomuscio and M. Sergot, 'Violation, error recovery, and enforcement in the bit transmission problem', in *Proceedings of DEON'02*, London, (May 2002).

[11] K. L. McMillan, *Symbolic model checking: An approach to the state explosion problem*, Kluwer Academic Publishers, 1993.

[12] R. van der Meyden and N. V. Shilov, 'Model checking knowledge and time in systems with perfect recall', *FSTTCS: Foundations of Software Technology and Theoretical Computer Science*, **19**, (1999).

[13] R. van der Meyden and K. Su. Symbolic model checking the knowledge of the dining cryptographers. Submitted, 2002.

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

[15] F. Raimondi and A. Lomuscio. A tool for verification of deontic interpreted systems. http://www.dcs.kcl.ac.uk/pg/franco/mcdis-0.1.tar.gz.

[16] F. Raimondi and A. Lomuscio, 'Verification of multiagent systems via ordered binary decision diagrams: an algorithm and its implementation', in *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'04)*, (July 2004).

[17] A. S. Rao, 'AgentSpeak(L): BDI agents speak out in a logical computable language', *Lecture Notes in Computer Science*, **1038**, 42–52, (1996).

[18] M. Wooldridge, M. Fisher, M.P. Huget, and S. Parsons, 'Model checking multi-agent systems with MABLE', in *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'02)*, eds., M. Gini, T. Ishida, C. Castelfranchi, and W. Lewis Johnson, pp. 952–959. ACM Press, (July 2002).