Model checking knowledge, strategies, and games in multi-agent systems

Franco Raimondi and Alessio Lomuscio
Technical report RN0501, revised version 30/08/2005.

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

Abstract. We present an OBDD-based methodology for verifying knowledge and strategies in multi-agent systems specified by the formalism of interpreted systems. To this end, we investigate the interpretation of ATL and epistemic formulae in various classes of interpreted systems, we present model checking algorithms and their implementation, and report experimental results.

1 Introduction

Model checking was traditionally put forward to verify specifications given in *temporal* logics [5]. Recently, however, researchers have extended model checking techniques to other modal logics, including some typical multi-agent systems (MAS) logics, thereby making it possible to verify formally a range of multi-agent systems. Examples of efforts on this line include [24, 4, 7, 18, 21]. These works share the model checking approach but differ in the choice of the logic specification language, and in the specific model checking technique employed.

In parallel developments, Alur et al [3] introduced Alternating-time Temporal Logic (ATL), a logic to reason about *strategies* in multi-player games. Model checking approaches for this logic have been developed [2, 15]. Recently, van der Hoek and Wooldridge proposed the logic ATEL [10]. ATEL extends ATL with epistemic operators; the semantics is defined over runs of a multi-agent system. However, it has been argued [11, 14, 12, 13] that the interpretation of ATL operators in ATEL might not correspond entirely to the original spirit of ATL [3], and various solutions have been put forward [14, 12, 13] to express ATL operators in a semantics based on MAS.

Against this background, and in parallel with this discussion, [10, 8, 19] suggested different techniques to reduce the problem of ATEL model checking to standard ATL model checking, with the idea of using MOCHA, the only existing model checker for ATL, for model checking MAS. In addition, the algorithms proposed in [15] are based on a reduction of the problem of model checking ATEL to a boolean satisfiability problem; however, no implementation is available yet.

The research of methodologies and the developments of tools for the formal verification of time, knowledge, and strategies of agents is motivated by their possible applications. Indeed, it has long been recognised that epistemic logics play an important role in the specification of many scenarios, such as games [13], planning [10], communication protocols [17], and security [9]. In this paper we try to make further progress

in this line by investigating interpretations of epistemic and ATL operators in interpreted systems, a mainstream semantics for MAS, and by presenting a model checking methodology for it. In particular:

- we consider various classes of interpreted systems and we explore in details the interpretation of ATL operators in these classes;
- we present model checking algorithms for ATL operators in various classes of interpreted systems based on Ordered Binary Decision Diagrams (OBDD's);
- we present an extension for MCMAS [20] (a tool for the automatic verification of temporal and epistemic operators in interpreted systems) that supports ATL operators.

Differently from previous approaches, our tool does not involve the translation or the reduction of the problem of model checking to plain ATL, and it allows for the automatic verification of ATL operators in different classes of interpreted systems, thereby allowing to express what agents may bring about, and what they may enforce. To our knowledge, this is not supported by any implementation. The tool is available for download from [20] under the terms of the GPL license.

The rest of the paper is organised as follows. In Section 2 we review the logic ATL and the formalism of interpreted systems upon which our tool is based. In Section 3 we discuss different classes of interpreted systems for the evaluation of ATL operators. In section 4 we introduce the algorithms for model checking and we present the tool MCMAS. In Section 5 we evaluate the effectiveness of our approach by verifying three examples and we report some experimental results. We conclude in Section 6.

2 Preliminaries

In this section we review the main formalisms that we shall use in the remainder of this paper. We first introduce the logic ATL, then we describe the formalism of interpreted systems to model multi-agent systems.

2.1 ATL

The syntax of the temporal logic ATL (Alternating-time Temporal Logic) [3] is defined as follows. Let AP be a finite set of atomic propositions, let $\Sigma = \{1, \ldots, n\}$ be a set of players, and let $\Gamma \subseteq \Sigma$ be a subset of the set of players. Well-formed ATL formulae are defined by the following grammar:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle\langle\Gamma\rangle\rangle X\varphi \mid \langle\langle\Gamma\rangle\rangle G\varphi \mid \langle\langle\Gamma\rangle\rangle(\varphi U \psi)$$

In [3], the formula $\langle\langle\Gamma\rangle\rangle \cdot \varphi$ is read as "the set of agents Γ can enforce $\cdot\varphi$" (where $\cdot$ denotes a temporal modality). ATL can be seen as an extension of the temporal logic CTL [5] where the path quantifiers E and A are replaced with quantification over a set of players. Indeed, quantification over the set of all players is read as the existential quantifier of CTL, while quantification over the empty set of agent is read as the universal quantifier of CTL.

Traditionally, the semantics of ATL formulae is given in terms of *concurrent game structures* (CGS). A CGS is a tuple $< \Sigma, S, AP, h, d, \delta >$ where Σ is a set of players, S is a finite set of states, AP is a set of atomic propositions, $h : AP \rightarrow 2^S$ is a labelling function, $d_i : \Sigma \times S \rightarrow I\!N$ is the number of moves available to a player i in a state (moves are labelled with natural numbers), and $\delta : S \times d_1 \times \ldots \times d_N \rightarrow S$ is an evolution function that associates a state to a (current) state and a set of moves, one for each player. A *strategy for player* i is a function f_i that maps sequences of states to a natural number, corresponding to a move available to player i at the end of the sequence: $f_i : S^+ \rightarrow I\!N$, such that $f_i(s) < d_i(s)$ for all states in the sequence. Given a state $s \in S$, a set of players Γ, and a set of strategies $F_\Gamma = \{f_i | i \in \Gamma\}$, the set $out(s, F_\Gamma) \subseteq S^+$ is the set of sequences that the group Γ can *enforce* in s. A sequence of states $s_0, s_1, \ldots$ is denoted with π, and $\pi(i) = s_i$ denotes the state at place i in the sequence. Satisfaction of an ATL formula in a state $s \in S$ of a given CGS is defined as follows:

$s \models p$ iff $s \in h(p)$,
$s \models \neg\varphi$ iff $s \not\models \varphi$,
$s \models \varphi_1 \vee \varphi_2$ iff $s \models \varphi_1$ or $s \models \varphi_2$,
$s \models \langle\langle\Gamma\rangle\rangle X\varphi$ iff there exists a set of strategies F_Γ s.t. for all computations $\pi \in out(s, F_\Gamma)$, $\pi(1) \models \varphi$.
$s \models \langle\langle\Gamma\rangle\rangle G\varphi$ iff there exists a set of strategies F_Γ s.t. for all computations $\pi \in out(s, F_\Gamma)$ and $\forall i \geq 0$, $\pi(i) \models \varphi$.
$s \models \langle\langle\Gamma\rangle\rangle(\varphi U \psi)$ iff there exists a set of strategies F_Γ s.t. for all computations $\pi \in out(s, F_\Gamma)$, $\exists i \geq 0$ s.t. $\pi(i) \models \psi$ and $\forall 0 \leq j < i$, $\pi(j) \models \varphi$

It is worth noticing here that the definitions above assume that *every player has complete information about the system and perfect recall*. Under this assumption, it has been proven in [3] that the model checking problem for ATL is P-complete. If a player has *incomplete information*, then his strategy can depend only on the *observable* part of the history of the game; in this case, the model checking problem for ATL is undecidable (see [3], p.708). However, if the *perfect recall* assumption is relaxed (i.e. if the players do not remember all the history of the game, and the strategy in a state is a function of the particular state only), then the problem of model checking is in P^{NP} [22]. The complexity of model checking ATL under various assumptions is reported in Table 1.

ir	**iR**	**Ir**	**IR**
P^{NP}	Undecidable	P	P

Table 1. Complexity of model checking ATL, from [22] (I/i = complete/incomplete information; R/r = perfect/imperfect recall).

2.2 Interpreted systems

An **interpreted system** [6] is a formal description of a set of agents $\Sigma = \{1, \ldots, n\}$. Each agent $i \in \Sigma$ is characterised by a finite set of *local states* L_i and by a finite set of actions Act_i that may be performed. Actions are performed in compliance with a protocol $P_i : L_i \rightarrow 2^{Act_i}$. Notice that this definition of protocols allows for *non-determinism* in the system. The environment in which agents "live" may be modelled by means of a special agent E, modelled by a set of local states L_E, a set of actions Act_E, and a protocol P_E. A tuple $g = (l_1, \ldots, l_n, l_e) \in L_1 \times \ldots \times L_n \times L_E$ is called a *global state* and gives a description of the system at a particular instant of time. The evolution of the agents' local states is described by a function $t_i : L_i \times L_E \times Act_1 \times \ldots \times Act_n \rightarrow L_i$ which gives the "next" local state as a function of the current local state of the agent, the environment, and all the other agents' actions. It is assumed that, in every state, agents evolve simultaneously (such a system is usually referred to as *lock-step* system). The evolution of the (global states of the) system may be described by a function $t : G \times Act \rightarrow G$, where $G \subseteq (L_1 \times \ldots \times L_n \times L_E)$ denotes the set of *reachable* global states, and $Act = Act_1 \times \ldots \times Act_n \times Act_E$ denotes the set of "joint" actions (similarly, one can reason about the joint actions of a group of agents $\Gamma \subseteq \Sigma$). The function t is the composition of all the functions t_i, and it is defined by $t(g, a) = g'$ iff $\forall i, t_i(l_i(g), a) = l_i(g')$, where $l_i(g)$ denotes the local state of agent i in global state g. The set G of reachable global states is obtained by considering all the possible evolutions of the system from a set of *initial* global states, denoted with I. Finally, to complete the description of a MAS, a set of atomic propositions AP is introduced, together with a valuation function $h : AP \rightarrow 2^G$.

In symbols, we will denote an interpreted systems IS with a tuple $IS = \langle (L_i, Act_i, P_i, t_i)_{i \in \Sigma}, I, h \rangle$. Interpreted systems have been proven a suitable semantics for reasoning about temporal and epistemic properties of agents [6, 16]. The standard semantics for epistemic and temporal modalities in interpreted systems is given Section 3.

3 ATL operators in interpreted systems

In this section we discuss how interpreted systems can provide a semantics for a rich language, including operators to reason about time, knowledge, and strategies. Formally, the syntax of the language we consider is defined by the following grammar:

$$\begin{aligned} \varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid EX(\varphi) \mid EG\varphi \mid E(\varphi U \psi) \mid \\ K_i\varphi \mid E_\Gamma\varphi \mid C_\Gamma\varphi \mid D_\Gamma\varphi \mid \\ \langle\langle\Gamma\rangle\rangle X(\varphi) \mid \langle\langle\Gamma\rangle\rangle G(\varphi) \mid \langle\langle\Gamma\rangle\rangle[\varphi U \psi] \end{aligned}$$

In the grammar above, $\Gamma \subseteq \Sigma$ denotes a 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", and $C_\Gamma\varphi$ is read as "it is common knowledge in Γ that φ" [6]. Other temporal and ATL operators expressing eventuality (F), and the universal quantifier (A) for paths, can be derived in a standard way; we refer to [3, 5] for the details.

To provide a semantics for well-formed formulae we proceed as follows. Given an interpreted system IS, it is possible to associate a Kripke model M_{IS} to IS. In this way, formulae of the language presented above can be interpreted in M_{IS}. The *associated model* $M_{IS} = (W, R_t, (\sim_i)_{i\in\Sigma}, L)$ for a given interpreted system $IS = \langle (L_i, Act_i, P_i, t_i)_{i\in\Sigma}, I, h\rangle$ is defined as follows:

- The set of possible worlds W is defined as the set G of reachable states of the interpreted system. The set G can be obtained from the set of initial states I by iterating the evolution function defined by the protocols and the evolution functions of each agent.
- The temporal relation $R_t \subseteq W \times Act \times W$ relating two worlds (i.e. two global states) by means of a joint action is obtained by considering the composition of the protocol and of the evolution functions (see Section 4 for details).
- The epistemic accessibility relations $\sim_i$ are defined considering the equivalence of the local components of the global states: $g \sim_i g'$ iff $l_i(g) = l_i(g')$, i.e. iff the local states of agent i in global states g and g' are the same [6].
- The labelling function $L : AP \rightarrow 2^W$ is equivalent to the evaluation function h.

Formulae can be interpreted in M_{IS} in a standard way [6, 5, 3]. For convenience, we report here the semantics for ATL and epistemic formulae:

$M_{IS}, w \models K_i\varphi$	iff for all $w' \in W$, $w \sim_i w'$ implies $M_{IS}, w' \models \varphi$
$M_{IS}, w \models \langle\langle\Gamma\rangle\rangle X\varphi$	iff there exists a joint action a for the agents in Γ s.t. for all joint actions a' of the agents in $\Sigma\backslash\Gamma$, all temporal transitions labelled with the joint actions $< a, a' >$ lead to a state w' s.t. $M_{IS}, w' \models \varphi$.
$M_{IS}, w \models \langle\langle\Gamma\rangle\rangle G\varphi$	iff $M_{IS}, w \models \varphi$ and for all temporal paths π from w and, for all states w' of π, there is a joint action for the agents in Γ s.t. for all the joint actions of the agents in $\Sigma\backslash\Gamma$ all temporal transitions labelled with the joint actions $< a, a' >$ lead to a state w'' s.t. $M_{IS}, w'' \models \varphi$.
$M_{IS}, w \models \langle\langle\Gamma\rangle\rangle(\varphi U\psi)$	iff for all temporal paths π starting from w, the agents in Γ may perform joint actions along the paths s.t. eventually ψ will hold, and φ holds along the paths until then.

In the definition above, $< a, a' >$ denotes a joint action obtained by the concatenation of a joint action a for agents in Γ and a joint action a' for agents in $\Sigma\backslash\Gamma$.

Following [6] (p.118), we write $IS \models \varphi$ if $M_{IS} \models \varphi$, where $\models$ denotes the standard notion of satisfiability in a Kripke model. Notice that the semantics presented above corresponds to the memoryless, imperfect information semantics of ATL (denoted by **"ir"** in Table 1). A "partial" memory semantics could be defined in this formalism by adding a vector to the local states of an agent, containing the list of previous local states. We will denote this option with the term *bounded recall*.

It is known that non-intuitive results may arise when evaluating ATL operators in MAS logics [14, 12, 13]. We investigate this issue in the next subsections by considering various classes of interpreted systems

3.1 Non-deterministic interpreted systems

The most general class of interpreted systems is the one defined in Section 2.2. An example of an interpreted system in this class and its associated model is reported in Figure 1[1] .

```
Agent 1
  Local States = { l1',l1'',l1'''};
  Actions = { a1, a2, nothing };
  Protocol:
    In l1' : {a1,a2};
    In l1'' : { nothing };
    In l1''' : {nothing };
  End Protocol
  Evolution function:
    next(Local State) = l1'' if ( Local State = l1') and ( ( ( Action = a1 )
                and  ( Local State of Agent 2 = l2' ) ) or
                 ( ( Action = a2 ) and ( Local state of Agent 2 = l2'' ) ) )
    next(Local State) = l1''' if ( Local State = l1') and ( ( ( Action = a1 )
                and  ( Local State of Agent 2 = l2'' ) ) or
                 ( ( Action = a2 ) and ( Local state of Agent 2 = l2' ) ) )
    If none of the above is true, then do not change Local State
  end Evolution function
end Agent1

Agent 2
  Local States = { l2', l2'' };
  Actions = { nothing };
  Protocol: nothing;
  Evolution: do not change local state
end Agent 2

Initial States: ( Agent 1 Local State = l1' ) and ( Agent 2 Local state = l2' or l2'')
```

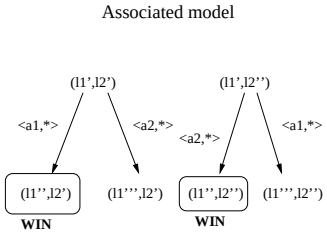

Fig. 1. Interpreted system description (left) and associated model (right)

Notice that the interpretation of ATL formulae in associated models *does not* express the intended meaning of the original ATL operators. In particular, similarly to ATEL, ATL formulae interpreted in models associated to the class of non-deterministic interpreted systems express what agents *may bring about*, maybe by guessing moves, and not what agents may *enforce*. Indeed, consider the example of Figure 1, and define an evaluation function assigning the proposition **win** to the global states `(l1'',l2')` and `(l1'',l2'')`. Then, the formula $\langle\langle Agent1 \rangle\rangle X\, win$ is true in the initial state, but it is clearly not the case that Agent 1 can enforce **win**. Indeed, Agent 1 *cannot distinguish* between the two initial states `(l1',l2')` and `(l1',l2'')`, because its local state is the same in this two global states (i.e. they are the *same* state from its point of view). However, the agent can win by "guessing" the action `a1` in global state `(l1',l2')`, and the action action `a2` in global state `(l1',l2'')`.

3.2 Deterministic interpreted systems

The reason the meaning of the ATL operators is different on interpreted systems is that these allow for agents to run non-deterministic protocols, i.e., the same agent may

[1] This is a reduced version of the card game presented in [14, 12].

perform different actions in a given local state. To follow the spirit of the original CGS more closely, we can focus on the subclass of interpreted systems, whose protocols are deterministic, i.e., protocols in which only one action is associated to a given local state: $P_i : L_i \rightarrow Act_i$.

We define an interpreted system to be *deterministic* iff the protocol of each agent is *deterministic* (a deterministic protocol associates a unique action to each local state). Notice that models associated to deterministic interpreted systems do not have a branching temporal structure. Instead, the system evolves linearly from one (or more) initial global states (note that the evolution function t is always deterministic).

Since agents are not allowed to "guess" actions, the evaluation of ATL operators in models associated to deterministic interpreted systems expresses the original "enforcement" meaning of ATL operators on CGS [3].

3.3 Γ-uniform interpreted systems

Deterministic interpreted systems are closest to the original CGS, but, as exemplified in Section 5, in many circumstances the class of deterministic interpreted systems is too restrictive to be used in the specification of MAS scenarios In these circumstances it is useful to reason about non-deterministic interpreted systems that at least are consistent in their selection of actions in a given local state.

By extending the concepts of [14, 13], we define an agent to be *uniform* if the agent performs the same action in epistemically equivalent global states. A group of agents $\Gamma \subseteq \Sigma$ is *uniform* if every agent in the group is uniform. We say that an interpreted system is Γ-*uniform* if all agents in Γ are uniform, whereas agents in $\Sigma \backslash \Gamma$ may choose their actions freely according to their protocol. Notice that, for a given interpreted system IS and a group of agents Γ, there may be several Γ-uniform restrictions of IS (but at most $\prod_{i \in \Gamma} A_i$, where A_i is the number of non-deterministic choices for actions of an agent). We denote with $\{IS\}_\Gamma$ the set of Γ-uniform interpreted systems that can be obtained from an interpreted system IS.

Similarly to the general case, we can associate a Kripke model to each Γ-uniform interpreted system in $\{IS\}_\Gamma$. We say a formula φ is true in a class of Γ-uniform interpreted systems, and we write $IS \models_\Gamma \varphi$, if φ is true in at least one of the models associated with the interpreted system in $\{IS\}_\Gamma$.

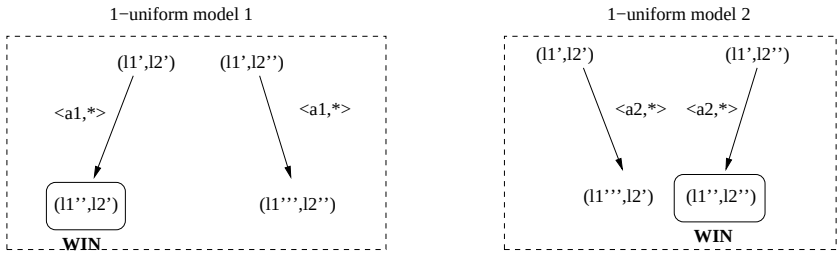

Fig. 2. Agent1-uniform models for the interpreted system of Figure 1.

As an example, the uniform models for the example of Figure 1 are presented in Figure 2. As expected, for the interpreted system of Figure 1, in the initial states we have $IS \models \langle\langle Agent1 \rangle\rangle X(win)$, while Figure 2 shows that $IS \not\models_{Agent1} \langle\langle Agent1 \rangle\rangle X(win)$.

4 Symbolic model checking techniques for interpreted systems

In this section we present the algorithms for the verification of temporal, epistemic, and ATL operators in the various classes of interpreted systems presented above. Our approach is similar, in spirit, to the traditional model checking techniques for the temporal logic CTL [5], and to the approach presented in [21]. Indeed, we reduce the problem of verification of a formula in an interpreted system to the verification of the equivalence between two boolean formulae. We begin by presenting the model checking algorithm for the class of non-deterministic interpreted systems (which can also be used for deterministic interpreted systems); then, we present a methodology for model checking formulae in Γ-uniform interpreted systems.

4.1 Model checking algorithm

To evaluate epistemic, temporal, and ATL operators in models associated to non-deterministic interpreted systems we proceed as follows. Given an interpreted system $IS = \langle (L_i, Act_i, P_i, t_i)_{i \in \Sigma}, I, h \rangle$, we begin by computing the number of boolean variables $v_i (i \in I\!N)$ required to encode the local states of an agent, denoted with $nv(i)$: $nv(i) = \lceil log_2 |L_i| \rceil$. Similarly, to encode an agent's action, the number of boolean variables $w_i (i \in I\!N)$ 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 can be encoded as a boolean vector $(w_1, \ldots, w_M)$, where $M = \sum_i na(i)$. 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 set of actions, they can also be expressed as boolean formulae. The evolution functions can be translated into boolean formulae, too. The set of initial states is easily translated, while h can be translated into a function returning a set of states, i.e. a boolean function.

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, and the definition of a boolean function $R_t(g, g')$, representing a temporal transition between the global states g and g'. $R_t(g, g')$ 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 (see [5] 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 global state g and t(g,a,g') is a boolean formula imposing that there is a temporal transition from g to g', labelled with the joint action a. The above gives the desired boolean relation between global states. Also, the set of *reachable* states is 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 $\tau(\emptyset)$ by standard procedure (see [5]).

Figure 3 presents the algorithm for model checking, based on the parameters presented above. $SAT(\varphi)$ computes the set of global states (expressed as a boolean formula) in which φ holds. The support procedures SAT_K and SAT_X are presented in Figure 4, while SAT_G, SAT_U, SAT_E, and SAT_C are defined in a standard way (see also [3, 10, 21] for a similar approach). The main idea is to express SAT_G and SAT_U as fix-point of operators based on SAT_X. The procedure $SAT_X(\varphi, \Gamma)$ uses a double quantification on actions and returns the set of states from which there exists an action for the agents in Γ such that, for all actions of the agents in $\Sigma \backslash \Gamma$, a transition is enabled such that in the next state φ holds. In the support procedures, Act_Γ denotes a joint action performed by group Γ. By comparing the OBDD representing the set of states in which a formula φ holds with the OBDD for the set of reachable states it is possible to establish whether or not $IS \models \varphi$.

$SAT(\varphi)$ {
  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 $\langle\langle\Gamma\rangle\rangle X\varphi_1$: return $SAT_X(\varphi_1, \Gamma)$;
  φ is $\langle\langle\Gamma\rangle\rangle(\varphi_1 U \varphi_2)$: return $SAT_U(\varphi_1, \varphi_2, \Gamma)$;
  φ is $\langle\langle\Gamma\rangle\rangle G\varphi_1$: return $SAT_G(\varphi_1, \Gamma)$;
  φ is $K_i(\varphi)$: return $SAT_K(\varphi, i)$;
  φ is $E_\Gamma(\varphi)$: return $SAT_E(\varphi, \Gamma)$;
  φ is $C_\Gamma(\varphi)$: return $SAT_C(\varphi, \Gamma)$;
}

Fig. 3. Model checking algorithm

4.2 Model checking Γ-uniform interpreted systems

The algorithm presented in Figure 3 can also be used for the evaluation of formulae in Γ-uniform interpreted systems. Indeed, the algorithm can be applied to the symbolic encoding of each model associated to each interpreted system in $\{IS\}_\Gamma$.

To evaluate epistemic, temporal, and ATL operators in Γ-uniform interpreted systems we proceed as follows:

– The set of deterministic joint protocols P_Γ is computed by imposing that agents in Γ adhere to a deterministic protocol, as defined in the previous section. An element

$SAT_X(\varphi, \Gamma)$ {
Y = $\{g | (\exists a \in Act_\Gamma, g' \in G)$ s.t. $(\forall b \in Act_{\Sigma \setminus \Gamma}).[R_t(g, g')$ and $t(g, \langle a, b \rangle, g')$
and $g' \in SAT(\varphi)$ and (a, b) is consistent with the protocols in $g']\}$
return Y;
}

$SAT_K(\varphi, i)$ {
X = $SAT(\neg\varphi)$;
Y = $\{g \in G$ s.t. $R_i^K(g, g')$ and $g' \in X\}$
return ¬Y;
}

Fig. 4. Support procedure for SAT_X

of P_Γ is a tuple of deterministic protocols, one for each agent in Γ. Notice that there are at most $\prod_{i \in \Gamma} |A_i$ different elements in P_Γ, where A_i is the number of non-deterministic choices available to an agent.

– Each element $s \in P_\Gamma$, together with the protocols of the agents in $\Sigma \setminus \Gamma$, is used to define a Γ-uniform interpreted systems IS_s, obtained by considering a restriction of Γ.
– The set of Γ-uniform interpreted systems $\{IS\}_\Gamma$ is defined as $\{IS\}_\Gamma = \{IS_s | s \in P_\Gamma\}$.

By verifying whether $M_s \models \varphi$ or not for all M_s associated with elements of $\{IS\}_\Gamma$ it is possible to verify whether $IS \models_\Gamma \varphi$ or not.

4.3 MCMAS

In this section we present MCMAS, a tool that implements the algorithms presented in Section 4. In MCMAS, interpreted systems are described using the language ISPL (Interpreted Systems Programming Language). Figure 5 gives a short example of this language. We refer to the files available online [20] for the full syntax of ISPL. Formulae to be checked are provided at the end of the specification file, using an intuitive syntax.

The tool automatically parses the specification and builds the relevant parameters, stored as OBDD's using the library provided by [23]. As discussed in Section 3, formulae can be evaluated either in models associated to non-deterministic interpreted systems, or in the class of Γ-*uniform* interpreted systems. MCMAS accepts a command-line parameter to establish which class should be considered.

When the class of Γ-*uniform* interpreted systems is chosen, MCMAS determines the set Γ by including in Γ all the agents appearing in φ. To optimise the verification process, the set P_Γ is computed first, and then for each $s \in P_\Gamma$ a model is generated and verification in performed on this model. If the formula is true in the model, then the verification process ends; otherwise, another element $s \in P_\Gamma$ is chosen and the loop is repeated.

To determine whether or not a formula holds in the interpreted system, its OBDD is compared with the OBDD representing the set of reachable states, and the appropriate output is produced.

MCMAS can be run from the command line, and accepts various options to modify verbosity, to inspect OBDD's statistics and memory usage, to enable variable reordering in the OBDD's (see [23]), 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 3.3.

```
Agent SampleAgent
 Lstate = {s0,s1,s2,s3};
 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. 5. ISPL example

5 Examples and experimental results

5.1 Connect 4 and its variations

Connect Four is a two player board game; the board has 7 columns and 6 rows, and it is placed vertically. The players have two sets of discs of different colours and they drop alternatively the discs in one of the non-full columns: because of gravity columns get filled bottom-up. The winner is the first player who can get four discs in line (vertically, horizontally, or diagonally). This is a game with perfect information: the only information relevant for the players is the configuration of the board, and this is publicly available.

Connect 4 has been analysed in [1] using a theorem proving approach, and it has been shown that using the standard board the first player has a strategy to win the game (the first move of this consist on placing his first disc in the middle column of the board). However, if the first player places the first disc in the first (or last) column,

then the second player can be shown to have a winning strategy at that point. In all the remaining cases, the second player can force a draw. These properties can be stated formally using the following formulae, where we also express that these facts are known to the players.

$$init \rightarrow K_1(\langle\langle player1 \rangle\rangle F(win1))$$

$$player1\_first\_a \rightarrow K_2(\langle\langle player2 \rangle\rangle F(win2))$$

$$player1\_first\_b \rightarrow K_2(\langle\langle player2 \rangle\rangle F(draw))$$

In the formulae above, the atomic proposition $init$ is true at the beginning of the game; the atomic proposition $win1$ (resp. $win2$) is true in a winning position for player 1 (resp. player 2); the atomic proposition $player1\_first\_a$ is true if the configuration of the board is such that the first player put a disc in the leftmost column and the second player did not move yet; similarly, the atomic proposition $player1\_first\_b$ is true if player 1 put a disc in the second column from the left; finally, the atomic proposition $draw$ is true if the board is full and none of the player is in a winning position.

Notice that the formulae above are true in the standard board 7x6, but they do not hold on smaller boards, e.g. on a 5x5 board. In this case, none of the players has a strategy to force a win from the initial state. Differently form [1] where these properties were proven as theorems, in this paper we verify these claims with MCMAS. It is not difficult to encode the game in the formalism of interpreted systems: two players can be encoded as two distinct agents, and the board can be encoded by means of a special agent (the environment). The ISPL code for various instances of this example is available for download from [20]. As this is a game with *perfect* information, we do not need to consider uniform strategies.

Due to memory limitations of our machines, we could verify boards up to the size of 5x5. We considered the following variations of Connect 4:

- *Fair 1* is a variation in which the second player is allowed a further move when the first reaches a winning position. If the second player can place four discs in a line with this last move, then the game ends in a draw.
- *Fair 2* is a variation in which the second player is allowed to "skip" a move. This may be useful for avoiding so-called "traps" by the first player.
- *Connect 3* is a variation in which only 3 discs in a line are necessary to win.

We could analyse with MCMAS the effect of these variations on the outcomes Connect 4; the results are reported in Table 2. It is interesting to notice that the rules above do not affect the ability of the first player to force a win in the case of *Connect 3*.

Verifications of these scenarios took from 9 minutes for smaller boards to 1h 48minutes for 5x5 boards on a 2.8GHz Intel Pentium IV, 1Gb of RAM, running Linux 2.6.8. Detailed results for the 5x4 board are presented in Table 3.

5.2 Nim

Nim is a two player game where players in turns remove any number of objects from one of a certain number of heaps (usually 3). Typically, at the beginning of the game, 3 heaps are presents, with 3, 4, and 5 objects in each. The player who takes the last object

Size:	**4x4**	**5x4**	**5x5**
Connect 4	No	No	No
Connect 4 + Fair 1	No	No	No
Connect 4 + Fair 2	No	No	No
Connect 3 + Fair 1	Yes	Yes	Yes
Connect 3 + Fair 2	Yes	Yes	Yes

Table 2. Variations of Connect 4: can the first player force a win?

Size:	**5x4**
Connect 4	1h12min
Connect 4 + Fair 1	1h12min
Connect 4 + Fair 2	1h21min
Connect 3 + Fair 1	26min54sec
Connect 3 + Fair 2	17min36sec

Table 3. Variations of Connect 4: verification time for three formulae

wins. There exists a variation of this game, called *Misère*, in which the player who takes the last object loses.

As in the previous example, this is a game with *perfect* information. It is not difficult to encode the game in the formalism of interpreted systems and the corresponding ISPL code is available for download from [20].

We considered two examples with 3-4-5 heaps, and with 5-5-5 heaps. We could confirm the known result that the first player can force a win *both* for the Nim and the Misère scenario. To this end, we checked the two formulae:

$$init \rightarrow \langle\langle player1 \rangle\rangle [\neg player2\_removelast \; U \; player1\_removelast]$$

$$init \rightarrow \langle\langle player1 \rangle\rangle [\neg player1\_removelast \; U \; player2\_removelast]$$

We could verify these formulae 18 seconds (for 3-4-5 heaps) and in 4 minutes and 8 seconds (for 5-5-5 heaps) using a 2.8GHz Intel Pentium IV, 1Gb of RAM, running Linux 2.6.8.

5.3 A simple card game

This example is presented in [12] and in [14] to analyse the effects of incomplete information in MAS: an agent (the player) plays a simple card game against another agent, the environment. There are just three cards in the deck: Ace (A), King (K), and Queen (Q); A wins over K, K wins over Q, and Q wins over A. In the initial state no cards are distributed; in the first step, the environment gives a card to the player and takes a card for itself. In the second step, the player can either keep its card, or change it. A description of this scenario in terms of interpreted systems can be easily obtained, and it is available in the downloadable files from [20]. We want to verify whether or not the player has a strategy to win the game in the initial state, i.e. we want to verify the following formula:

$$init \rightarrow \langle\langle player \rangle\rangle F(player\_win)$$

Differently from the original intended meaning of ATL operators, this formula is true if it is evaluated with MCMAS in the associated model: indeed, the player can always *guess* a move to win. However, the formula is false if it is evaluated in the class of *uniform* interpreted systems. This result confirms the intuition that the player does not have a *uniform* strategy to win the game.

Verification of this scenario only took 0.15 seconds on a 2.8GHz Intel Pentium IV, 1Gb of RAM, running Linux 2.6.8.

6 Conclusions

In this paper we have identified three classes of interpreted systems to evaluate temporal, epistemic, and ATL operators. In the first class agents may guess moves; therefore, ATL operators express what agents may *bring about*. The second class is constituted by deterministic agents; therefore, ATL operators express what agents may *enforce*. The third class comprises Γ-uniform interpreted systems, non-deterministic systems in which a group of agents Γ chooses consistently to perform certain actions. Model checking algorithms for these classes together with an implementation have also been presented.

While we see our results as encouraging, we also acknowledge that more work is needed before MCMAS could be considered a mature product. In particular, we did not tackle the issue of *fairness constraints*, and we did not consider the issue of counter-example generation for unsatisfiable formulae (counter-examples could be used in the field of planning for MAS). We leave these issues and comparisons with MOCHA for further work.

References

1. V. Allis. A knowledge-based approach of connect-four. Master's thesis, Vrije Universiteit, Amsterdam, 1989.
2. R. Alur, T. Henzinger, F. Mang, S. Qadeer, S. Rajamani, and S. Tasiran. MOCHA: Modularity in model checking. In *Proceedings of the 10th International Conference on Computer Aided Verification (CAV'98)*, volume 1427 of *LNCS*, pages 521–525. Springer-Verlag, 1998.
3. R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002.
4. R. H. Bordini, M. Fisher, C. Pardavila, and M. Wooldridge. Model checking AgentSpeak. In J. S. Rosenschein, T. Sandholm, W. Michael, and M. Yokoo, editors, *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-agent systems (AAMAS-03)*, pages 409–416. ACM Press, 2003.
5. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.
6. R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge*. MIT Press, Cambridge, 1995.
7. P. Gammie and R. van der Meyden. Mck: Model checking the logic of knowledge. In *Proceedings of 16th International Conference on Computer Aided Verification (CAV'04)*, volume 3114 of *LNCS*, pages 479–483. Springer-Verlag, 2004.
8. V. Goranko and W. Jamroga. Comparing semantics for logics of multi-agent systems. *Synthese*, 139(2):241–280, 2004.

9. J.Y. Halpern and R. Pucella. Modeling adversaries in a logic for security protocol analysis. In *Formal Aspects of Security, First International Conference, (FASec'02)*, volume 2629 of *LNCS*, pages 115–132. Springer-Verlag, 2003.
10. W. van der Hoek and M. Wooldridge. Tractable multiagent planning for epistemic goals. In M. Gini, T. Ishida, C. Castelfranchi, and W. L. Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'02)*, pages 1167–1174. ACM Press, 2002.
11. W. Jamroga. Some remarks on alternating temporal epistemic logic. In B. Dunin-Kęplicz and R. Verbrugge, editors, *Proceedings of the International Workshop on Formal Approaches to Multi-Agent Systems (FAMAS'03)*, pages 133–140, 2004.
12. W. Jamroga. *Using Multiple Models of Reality. On Agents who Know how to Play Safer.* PhD thesis, University of Twente, Enschede, The Netherlands, 2004.
13. W. Jamroga and W. van der Hoek. Agents that know how to play. *Fundamenta Informaticae*, 62:1–35, 2004.
14. G. Jonker. Feasible strategies in alternating-time temporal epistemic logic. Master's thesis, University of Utrech, The Netherlands, 2003.
15. M. Kacprzak and W. Penczek. A SAT-based approach to unbounded model checking for alternating-time temporal epistemic logic. *Synthese*, 142:203–227, 2004.
16. A. Lomuscio. *Knowledge Sharing among Ideal Agents*. PhD thesis, School of Computer Science, University of Birmingham, Birmingham, UK, June 1999.
17. A. Lomuscio and M. Sergot. A formalisation of violation, error recovery, and enforcement in the bit transmission problem. *Journal of Applied Logic*, 2(1):93–116, March 2004.
18. W. Nabiałek, A. Niewiadomski, W. Penczek, A. Półrola, and M. Szreter. VerICS 2004: A model checker for real time and multi-agent systems. In *Proceedings of the International Workshop on Concurrency, Specification and Programming (CS&P'04)*, volume 170 of *Informatik-Berichte*, pages 88–99. Humboldt University, 2004.
19. S. van Otterloo, W. van der Hoek, and M. Wooldridge. Knowledge as strategic ability. *ENCTS*, 85(2):1–23, 2003.
20. F. Raimondi and A. Lomuscio. MCMAS - A tool for verification of multi-agent systems. http://www.cs.ucl.ac.uk/staff/f.raimondi/MCMAS/.
21. F. Raimondi and A. Lomuscio. Verification of multiagent systems via ordered binary decision diagrams: an algorithm and its implementation. In N. R. Jennings, C. Sierra, L. Sonenberg, and M. Tambe, editors, *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'04)*, volume II. ACM, July 2004.
22. P.Y. Schobbens. Alternating-time logic with imperfect recall. *ENTCS*, 85(2):1–12, 2004.
23. F. Somenzi. CUDD: CU decision diagram package - release 2.4.0. http:/ /vlsi.colorado.edu/ ∼fabio/CUDD/cuddIntro.html.
24. M. Wooldridge, M. Fisher, M. Huget, and 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, July 2002.