Model Checking Knowledge, Strategies, and Games in Multi-Agent Systems

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

ABSTRACT

We present an OBDD-based methodology for verifying time, 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.

General Terms

Theory;Verification,Algorithms

Keywords

Model checking multi-agent systems; ATL

1. INTRODUCTION

Model checking was traditionally put forward to verify specifications given in *temporal* logics [6]. 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 against temporal, epistemic, and other modalities. Examples of efforts on this line include [25, 4, 8, 19, 22]. These works share the model checking approach but differ in the choice of the logic-based specification language, and in the specific model checking technique employed.

In parallel developments, Alur et al. introduced Alternating-time Temporal Logic (ATL), a logic to reason about *strategies* in multi-player games [3]. Model checking approaches for this logic have been developed [2, 16]. More recently, van der Hoek and Wooldridge proposed the logic ATEL [11]. ATEL extends ATL with epistemic operators whose semantics is defined over runs of a multi-agent system. However, it has been argued [12, 15, 13, 14] that the interpretation of ATL operators in ATEL might not correspond entirely to the original spirit of ATL [3]. Following [11], various solutions have been put forward [15, 13, 14, 1, 10] to express ATL operators in a semantics based on MAS. Some of these issues relate back to the seminal work by Moore [18].

AAMAS'06 May 8–12 2006, Hakodate, Hokkaido, Japan.

Against this background, and in parallel with this discussion, [11, 9, 20] 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 [16] are based on a reduction of the problem of model checking ATEL to a boolean satisfiability problem; however, no implementation is available yet.

In this paper we try to make further progress in this line by investigating interpretations of epistemic and ATL operators on interpreted systems, a mainstream semantics for MAS, and by presenting a verification methodology for it. In particular:

- we consider various classes of interpreted systems and we explore in detail the interpretation of ATL operators on these classes;
- we present model checking algorithms for ATL operators on various classes of interpreted systems;
- we present an extension for MCMAS [21] (a tool for the automatic verification of temporal and epistemic operators in interpreted systems) supporting ATL operators.

In contrast to previous approaches, our tool does not involve the translation or the reduction of the problem of model checking to plain ATL, and it permits the automatic verification of ATL operators on different classes of interpreted systems, thereby making it possible 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 [21] under the terms of the GPL license.

The rest of the paper is organised as follows. In Section 2 we review theoretical preliminaries, and we compare briefly semantics for ATL and for interpreted systems. 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 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, and we compare the semantics of interpreted systems with ATL semantics.

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. Let $p \in AP$ be an atomic proposition; 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 players Γ can enforce $\cdot\varphi$" (where $\cdot$ denotes a temporal modality). ATL can be seen as an extension of the temporal logic CTL [6] where the path quantifiers E and A are replaced with quantification over sets of computations which players can "enforce". Indeed, quantification over all the possible computations is read as the existential quantifier of CTL, while quantification over the empty set 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 an evaluation function, $d : \Sigma \times S \rightarrow I\!N$ is the number of moves available to a player i in a state (moves are labelled with natural numbers), and δ is an evolution function that determines the evolution of the system. Various definitions of δ are possible, we refer to [3] for more details. 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, i.e., $f_i : S^+ \rightarrow I\!N$, with $f_i(s) < d(i, s)$ 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, as defined in [3], p. 685. A sequence of states $s_0, s_1, \ldots$ such that each $s_i, i \geq 0$ is related to s_{i+1} via the evolution function δ is denoted with π, and $\pi(i) = s_i$ denotes the i-th state in the sequence. Sequences of states are usually called *computations*. 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 are given with the intuition that *every player has complete information about the system and perfect recall* (see Section 2.3 for further details).

2.2 Interpreted systems

An interpreted system [7] is a formal description of the computations carried out by a set of agents $\Sigma = \{1, \ldots, n\}$. Each agent $i \in \Sigma$ is characterised by a finite set of *private 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 protocol P_E. A tuple $g = (l_1, \ldots, l_n, l_e) \in L_1 \times \cdots \times L_n \times L_E$ 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 $t_i : L_i \times L_E \times Act_1 \times \cdots \times Act_n \times Act_E \rightarrow L_i$ 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 (the evolution function for the environment is defined as $t_E : L_E \times Act_1 \times \cdots \times Act_n \times Act_E \rightarrow L_E$). It is assumed that, in every state, agents evolve simultaneously (this assumption corresponds to the definition of *Moore synchronous* CGS, in which the evolution function δ prescribes a simultaneous evolution of players). The evolution of the (global states of the) system may be described by a function $t : G \times Act \rightarrow G$, where $G \subseteq (L_1 \times \cdots \times L_n \times L_E)$ denotes the set of *reachable* global states, and $Act = Act_1 \times \cdots \times Act_n \times Act_E$ 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 $\forall i, t_i(l_i(g), l_E(g), a) = l_i(g')$, where $l_i(g)$ denotes the local state of agent i in global state g and $a \in Act$. 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 an interpreted system, a set of atomic propositions AP is introduced, together with a valuation function $h : AP \rightarrow 2^G$.

Formally, an interpreted systems IS is a tuple $IS = \left\langle (L_i, Act_i, P_i, t_i)_{i \in \Sigma \cup \{E\}}, I, h \right\rangle$. Interpreted systems have been proven a suitable formalism for reasoning about temporal and epistemic properties of agents [7, 17]. The standard semantics of interpreted systems is given Section 3.

2.3 On the relationship between interpreted systems and concurrent game structures

Interpreted systems and concurrent game structures are closely related. Consider an interpreted system

$$IS = \left\langle (L_i, Act_i, P_i, t_i)_{i \in \Sigma \cup \{E\}}, I, h \right\rangle$$

and a concurrent game structure

$$CGS =< \Sigma, S, AP, h, d, \delta >:$$

- both structures comprise a set of agents (or players) Σ, and both structures comprise a set of states, called *global* states in IS;
- the function d, which returns the number of moves available to a player in a state of a CGS, intuitively corresponds to the protocols P_i in IS;
- the evolution function δ of CGS is an "accessibility" relation between states, as the evolution function t of IS, defined in terms of the evolution functions t_i;
- the valuation functions h of CGS and IS both label (global) states with propositions.

Differently from CGS, however, global states of interpreted systems have a structure, being tuples of *private local states*. This subtle difference is key in understanding the difference in the semantics for ATL operator in CGS and in interpreted systems presented in the next section. Indeed, CGS assume that every player

has *perfect information*, i.e., every player is fully aware of the state $s \in S$ at every time instant. Conversely, in interpreted systems an agent is aware of its private local state only.

A further difference between interpreted systems and CGS is the definition of strategies. In [3], a strategy is defined as a function from *sequences* of states to an action. In the case of interpreted systems, instead, we define below a strategy for agent i to be a function from a (single) local state to an action of agent i (see the next Section for a discussion about this). In this sense, strategies and protocols in interpreted systems are closely related because they associate actions to states and, in the case of *deterministic* interpreted systems (see Section 3.2), strategies and protocols are the same mathematical object. In the remainder of the paper we will refer to protocols when describing interpreted systems, and to strategies when reasoning about ATL formulae in interpreted systems.

Explanations of other technical similarities and differences between the two semantics is not the aim of the present paper.

3. ATL OPERATORS IN INTERPRETED SYSTEMS

In this section we discuss how interpreted systems can provide a semantics for a language that includes operators to reason about time, knowledge, and strategies. Formally, let AP be a set of atomic propositions and let $p \in AP$ be an atomic proposition; the syntax of the language we consider is defined by the following grammar:

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

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", $C_\Gamma\varphi$ is read as "it is common knowledge in Γ that φ" [7], $D_\Gamma\varphi$ is read as "it is distributed knowledge in Γ that φ, and $\langle\langle\Gamma\rangle\rangle \cdot \varphi$ (where $\cdot$ is one of the temporal operators X, G, or U) is read as "group Γ can enforce $\cdot\varphi$". Other temporal and ATL operators expressing eventuality (F), and the universal quantifier (A) for paths, can be derived in a standard way. Notice that, although the temporal operators can be derived from the ATL operators, we treat them separately to stress the difference between temporal reasoning and strategic reasoning; we refer to [3, 6] for details.

To provide a semantics for well-formed formulae we proceed as follows. Given an interpreted system IS, we associate a Kripke model M_{IS} to IS. The *associated model* $M_{IS} = (G, t, (\sim_i)_{i\in\Sigma}, L)$ for a given interpreted system $IS = \left\langle (L_i, Act_i, P_i, t_i)_{i\in\Sigma\cup\{E\}}, I, h \right\rangle$ is defined as follows:

- The set of possible worlds G is the set of reachable states of IS. The set G can be obtained from the set of initial states I by iterating the evolution function t defined by the protocols and the evolution functions of each agent in Section 2.2.
- The temporal relation $t \subseteq G \times Act \times G$ relating two worlds by means of a joint action is the relation defined in Section 2.2.
- Each epistemic accessibility relations $\sim_i$ is defined by the equality 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 [7].
- The labelling function $L : AP \to 2^G$ is equivalent to the evaluation function h.

Temporal and epistemic formulae are interpreted on M_{IS} in a standard way; we refer to [7, 6, 3] for details. Similarly to Section 2.1, we define a strategy to be a function $f_i : L_i \to Act_i$ from (local) states to actions for agent i, with the requirement that $f_i(l_i(g)) \in P_i(l_i(g))$, i.e. $f_i(l_i(g))$ is one of the actions enabled by the protocol P_i for agent i when its local state is $l_i(g)$. Notice that this definition corresponds to the memoryless, imperfect information semantics of ATL [23]. Different partial recall semantics could be defined by using vectors of local states. Given a group of agents Γ and a state $g \in G$, let $F_\Gamma = \{f_i | i \in \Gamma\}$ be a set of strategies; as in Section 2.1, we denote with $out(g, F_\Gamma)$ the set of computations that group Γ can enforce at state g (computations are defined as in Section 2.1). For convenience, we report here the semantics for ATL and epistemic formulae.

- $M_{IS}, g \models K_i\varphi$ iff for all $g' \in G$, $g \sim_i g'$ implies $M_{IS}, g' \models \varphi$.
- $M_{IS}, g \models \langle\langle\Gamma\rangle\rangle X\varphi$ iff there exists a set of strategies F_Γ such that, for all computations $\pi \in out(g, F_\Gamma)$, $\pi(1) \models \varphi$.
- $M_{IS}, g \models \langle\langle\Gamma\rangle\rangle G\varphi$ iff there exists a set of strategies F_Γ such that, for all computations $\pi \in out(g, F_\Gamma)$ and for all $i \geq 0$, $\pi(i) \models \varphi$.
- $M_{IS}, g \models \langle\langle\Gamma\rangle\rangle[\varphi U \psi]$ iff there exists a set of strategies F_Γ such that for all computations $\pi \in out(g, F_\Gamma)$, there exists $i \geq 0$ such that $\pi(i) \models \psi$, and for all $0 \geq j < i$ $\pi(j) \models \varphi$.

Following [7] (p.118), we write $IS \models \varphi$ if $M_{IS} \models \varphi$.

It is known that non-intuitive results may arise when evaluating ATL operators in MAS logics [15, 13, 14, 1]. We investigate this issue for the semantics of interpreted systems 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. Consider the following simple example of an interpreted system IS[1] composed by two agents Ag_1 and E (the environment); the agent Ag_1 is modelled by means of three local states $L_1 = \{l_1^1, l_1^2, l_1^3\}$ and two possible actions $Act_1 = \{a_1^1, a_1^2\}$. The protocol for Ag_1 prescribes that every action is allowed in every state. The second agent E is modelled by means of two local states $L_E = \{l_E^1, l_E^2\}$, and we assume that no action is performed by E (and thus no protocol is required). In the initial state the system can be either in global state (l_1^1, l_E^1) or in global state (l_1^1, l_E^2). The evolution function for Ag_1 is defined below:

Loc. st. Ag_1	Loc. st. E	Act. Ag_1	Next state
l_1^1	l_E^1	a_1^1	l_1^2
l_1^1	l_E^1	a_1^2	l_1^3
l_1^1	l_E^2	a_1^1	l_1^3
l_1^1	l_E^2	a_1^2	l_1^2

We assume that in every other state there is no change in local state for Ag_1, and we assume that the local state of E never changes. We introduce an evaluation function h assigning the proposition **WIN** to the global states (l_1^2, l_E^1) and (l_1^2, l_E^2). The model associated to IS is depicted in Figure 1.

In this simple example the formula $\langle\langle Ag_1\rangle\rangle X(\mathbf{WIN})$ is true in the initial states of IS; however, it is not the case that Ag_1 is capable of *enforcing* a state in which the proposition **WIN** holds. Indeed, Ag_1 cannot distinguish between the global states (l_1^1, l_E^1) and (l_1^1, l_E^2),

[1]This is a reduced version of the card game presented in [15, 13].

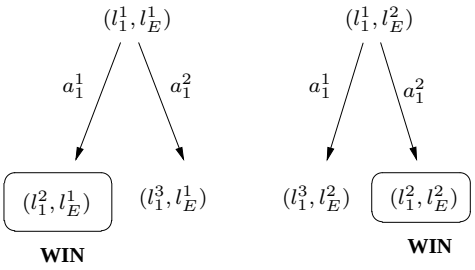

Figure 1: A simple example.

because its local state is the same in the two global states. We conclude that, similarly to what happens in the case of ATEL, ATL operators in non-deterministic interpreted systems do not express what agents may enforce, but what agents may *bring about*. In our example, the formula $\langle\langle Ag_1 \rangle\rangle X(\textbf{WIN})$ has to be read "Ag_1 may bring about a state in which the proposition **WIN** holds, maybe by *guessing* moves".

3.2 Deterministic interpreted systems

The reason the meaning of the ATL operators is subtly different in interpreted systems is that these allow for agents to run non-deterministic protocols, i.e., the same agent may 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, since we assume the transition function to be deterministic, the models associated to deterministic interpreted systems are not branching but linear.

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

3.3 Γ-uniform interpreted systems

Deterministic interpreted systems are close in spirit to the original interpretation of ATL in CGS, but, as exemplified later 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 [15, 14], we define an agent to be *uniform* if throughout a run the agent performs always the same action when in a particular local state. 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. We define a Γ-uniform interpreted system IS_Γ to be *compatible* with an interpreted system IS if (i) it contains the same agents, with the same set of local states, actions, and evolution function, and (ii) the protocols P_i^Γ for the Γ agents in IS_Γ are a *restriction* of the protocols P_i of IS, in the sense that for all local states $l_i \in L_i$, $P_i^\Gamma(l_i) \in P_i(l_i)$ (i.e. only one action is selected in P_i^Γ, and the action is one of the actions specified in P_i).

Notice that, for a given interpreted system IS and a group of agents Γ, there may be several Γ-uniform interpreted systems compatible with IS (but at most $\prod_{i \in \Gamma} |Act_i|^{|L_i|}$). We denote with $\{IS\}_\Gamma$ the set of Γ-uniform interpreted systems that are compatible with a given interpreted system IS.

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

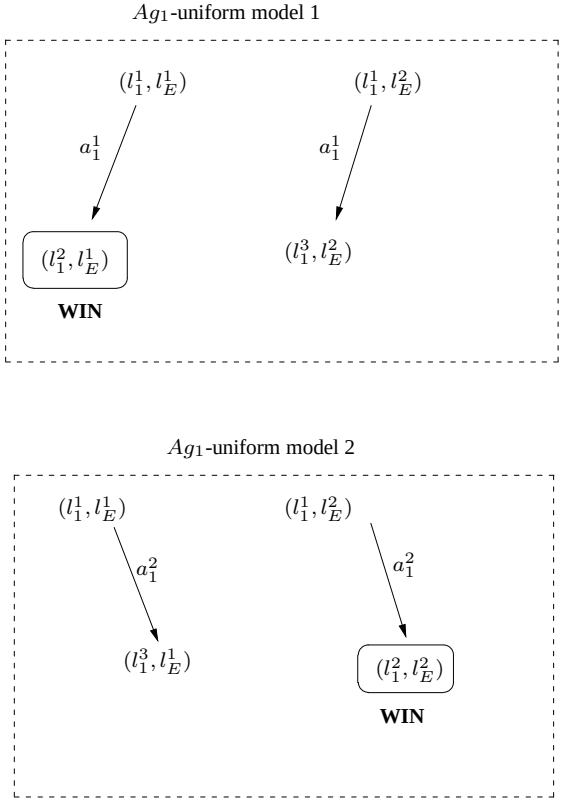

Figure 2: Ag_1-uniform interpreted systems for the interpreted system of Figure 1.

As an example, the uniform interpreted systems for the example of Figure 1 are presented in Figure 2. In this example, in the initial states we have $IS \models \langle\langle Ag_1 \rangle\rangle X(win)$, but $IS \not\models_{Ag_1} \langle\langle Ag_1 \rangle\rangle X(win)$, because there is no Ag_1-uniform interpreted system compatible with the original interpreted system in which the formula is true. Intuitively, a uniform agent chooses one of the actions allowed by the protocols, and it behaves consistently with its choice, i.e., the agent will always choose the same action in epistemically equivalent states.

4. OBDD-BASED MODEL CHECKING 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 [6], and to the approach presented in [22].

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 $IS = \left\langle (L_i, Act_i, P_i, t_i)_{i \in \Sigma \cup \{E\}}, I, h \right\rangle$, note that the number of boolean variables $v_i (i \in I\!N)$ required to encode the local states of an agent is $nv(i) = \lceil log_2 |L_i| \rceil$. Similarly, to encode an agent's action, the number of boolean variables $w_i (i \in I\!N)$ required is $na(i) = \lceil log_2 |Act_i| \rceil$. A global state g can be encoded as a boolean vector $\overline{v} = (v_1, \ldots, v_N)$, where $N = \sum_i nv(i)$. Similarly, a joint action a can be encoded as a boolean vector $\overline{w} = (w_1, \ldots, w_M)$, 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 (and similarly a set of joint actions) can be expressed as the disjunction of the boolean formulae encoding each global state in the set. Given this, protocols can be encoded by implications between boolean formulae representing local states and actions (see [22] for more details). Let $P(\overline{v}, \overline{w})$ be the boolean formula obtained by taking the conjunction of the boolean formulae encoding the protocols for all the agents. The evolution function t_i for a single agent can also be expressed as a boolean formula; we denote with $t(\overline{v}, \overline{w}, \overline{v'})$ the boolean function obtained by taking the conjunction of all the boolean formulae encoding the evolution functions for the agents. Notice that evolution functions require two sets of boolean variables for encoding states; in the above $\overline{v'} = (v'_1, \ldots, v'_N)$ denotes the additional set of boolean variables required. The set of initial states can be encoded by means of a boolean formula $I(\overline{v})$, as well as the evaluation function h.

In addition to the parameters presented above, the algorithm for model checking presented below requires the definition of n boolean functions $R_i^K(\overline{v}, \overline{v'})$ (one for each agent) representing the epistemic accessibility relation, and the definition of a boolean function $R_t(\overline{v}, \overline{v'})$, representing a temporal transition between the global states g and g' encoded by means of the boolean vectors $(v_1, \ldots, v_N)$ and $(v'_1, \ldots, v'_N)$. $R_t(\overline{v}, \overline{v})$ can be obtained from the boolean function t by quantifying over actions. This quantification can be translated into a propositional formula using a disjunction (see [6] for a similar approach to boolean quantification):

$$R_t(\overline{v}, \overline{v'}) = \bigvee_{\overline{w} \in Act} [(t(\overline{v}, \overline{w}, \overline{v'}) \wedge P(\overline{v}, \overline{w})]$$

The formula above provides a boolean relation between global states that can be used in the evaluation of temporal operators. 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 $\tau(Q) = (I(\overline{v}) \vee (\exists(\overline{v'})).(R_t(\overline{v'}, \overline{v}) \wedge Q(\overline{v'})$. Intuitively, $\tau(Q)$ computes the sets of states that are reachable from Q in a single temporal step. The fix-point of τ can be computed by iterating $\tau(\emptyset)$ as standard (see [6]).

Figure 3 presents the algorithm for model checking, based on the parameters presented above. In the following, Act_Γ denotes a joint action performed by group Γ. Let $a \in Act_\Gamma$ and $b \in Act_{\Sigma \setminus \Gamma}$: we denote with (a, b) the joint action defined by the concatenation of a and b, with the appropriate reordering of elements, if needed. $SAT(\varphi)$ 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, SAT_C, and SAT_D are defined using the ideas of [3, 11, 22]. The

$SAT(\varphi)$ {
 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 $SAT_C(\varphi, \Gamma)$;
 φ is $D_\Gamma(\varphi)$: return $SAT_D(\varphi, \Gamma)$;
}

Figure 3: Model checking algorithm

$SAT_X(\varphi, \Gamma)$ {
 Y = $\{g | (\exists a \in Act_\Gamma, g' \in G)$ s.t. $(\forall b \in Act_{\Sigma \setminus \Gamma}).[R_t(g, g')$ and $t(g, (a, b), g')$ 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 $\neg$Y;
}

Figure 4: Support procedure for SAT_X and SAT_K

key difference in our work is the definition of temporal transitions by means of interpreted systems' parameters (i.e. actions, protocols, evolution functions, etc.) as presented above. 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 $\Sigma \setminus \Gamma$, a transition is enabled such that in the next state φ holds. In the support procedures, by slight abuse of notation, we use a compact notation for global states and actions instead of boolean variables and vectors; the intended meaning should be clear from the context.

The algorithm presented above is designed to be compatible with the representation of boolean formulae by means of Ordered Binary Decision Diagrams (OBDDs, [5]): OBDDs are an efficient technique for the representation and manipulation of boolean formulae. Due to space limitation, we refer to [5] for more details about this technique.

The model checking algorithm above terminates by comparing the OBDD representing the set of states in which a formula φ holds with the OBDD for the set of reachable states. Notice that if the two OBDDs are equal, then the Boolean formulae generating them are propositionally equivalent, therefore $IS \models \varphi$.

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, the set $\{IS\}_\Gamma$ is built as follows:

- The set of deterministic joint protocols[2] P_Γ is computed by imposing that all the agents in Γ adhere to a deterministic

[2]A joint protocol extends the definition of protocol to a set of agents. Formally, given a set of agents Γ, a joint protocol is de-

protocol, and by imposing that elements of P_Γ are *restrictions* (in the sense of Section 3.3) of the protocols of IS. An element of P_Γ can be seen as a tuple of deterministic protocols, one for each agent in Γ. Notice that there are at most $\prod_{i\in\Gamma} |A_i|^{|L_i|}$ different elements in P_Γ, where A_i is the maximum number of non-deterministic choices available to an agent.

- For each element $p \in P_\Gamma$, an interpreted system IS_p is defined as:
  - The set of agents, their local states, actions, and evolution functions for IS_p are equal to the corresponding elements of IS.
  - The protocols for the agents in Γ in IS_p are defined by p, while the protocols for agents in $\Sigma\backslash\Gamma$ are equal to the protocols in IS.
- The set of Γ-uniform interpreted systems $\{IS\}_\Gamma$ is defined as $\{IS\}_\Gamma = \{IS_p | p \in P_\Gamma\}$.

To check $IS \models_\Gamma \varphi$, we check whether $M_{IS_p} \models \varphi$ for some $p \in P_\Gamma$.

It is worth noting that the time complexity of verifying $IS \models_\Gamma \varphi$ by means of the algorithm presented above exceeds, in the worst case, the time complexity of $IS \models \varphi$ by a factor of $\prod_{i\in\Gamma} |A_i|^{|L_i|}$. The space complexity of the verification, instead, remains the same because the space employed in the verification of each model M_s associated with Γ-uniform interpreted systems in $\{IS\}_\Gamma$ can be reused.

4.3 MCMAS

In this section we present MCMAS, a tool that implements the algorithms presented in Section 4.2. 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 [21] for the full syntax of ISPL. Formulae to be checked are provided at the end of the specification file.

The tool automatically parses the specification and builds the relevant parameters, stored as OBDDs by using the library provided in [24]. 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 select 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 under the scope of ATL operators in φ (notice that other implementation choices are possible without modifying the structure of MCMAS, for instance providing a separate list of uniform agents, irrespective of φ). To optimise the verification process, the set P_Γ is computed first, and then for each $p \in P_\Gamma$ a model is generated as in Section 4.2 and checked. If the formula is true in the model, the verification process ends with success; otherwise, another element $p \in P_\Gamma$ is chosen and the loop is repeated until all the elements of P_Γ have been tested, and the process terminates with failure.

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 [24]), 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.

fined as a function from the Cartesian product of the local states of the agents in Γ to the power set of the Cartesian product of the actions of the agents in Γ.

```
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
```

Figure 5: ISPL example

5. EXAMPLES AND EXPERIMENTAL RESULTS

5.1 Nim

Nim is a two player game where players in turns remove any number of objects from one of a certain number of heaps. Typically, 3 heaps are presents and the game starts with 3 objects in the first heap, 4 in the second, and 5 in the third. The player who takes the last object wins. In a variation of this game, called *Misère*, the player who takes the last object loses.

This is a game with *perfect* information. It is not difficult to encode the game in the formalism of interpreted systems, by imposing that only one agent makes a move at any given time step. The corresponding ISPL code is available for download from [21].

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:

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

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

(where "init" is a proposition true in the initial state of the system).

We could verify these formulae in 18 seconds (for 3-4-5) 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. The model for 3-4-5 heaps requires 51 boolean variables for the symbolic encoding, while 57 boolean variables are needed for the encoding of the 5-5-5 heaps (corresponding approximately to models of size 10^{15} and 10^{17} respectively).

5.2 A simple card game

This example is presented in [13] and in [15] 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

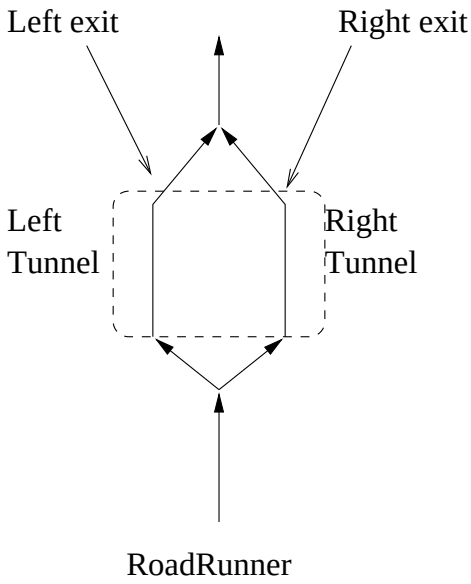

Figure 6: Diagram for RoadRunner and Coyote

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 [21]. 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 when evaluated with MCMAS: 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 cannot enforce a winning state.

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

5.3 RoadRunner and Coyote

This example illustrates further the different meaning of ATL operators in Γ-uniform interpreted systems and in non-deterministic interpreted systems.

RoadRunner is running in a hilly region of the desert; the main road splits in two small lanes just before the entrance of two tunnels below a mountain. RoadRunner can pick randomly either tunnel; the tunnels are identical and very narrow. Coyote knows RoadRunner has to take one of the two tunnels, and so he has bought a special tunnel-blocking device from ACME Inc. to catch RoadRunner. The device may be placed in front of either exit of the tunnel (see Figure 6).

The example can be modelled as an interpreted system IS by taking two agents, one for Coyote and one for RoadRunner. RoadRunner is described by means of two local states, "left" and "right", and by means of the single action "run"; local states for RoadRunner do not change. Coyote is modelled by means of three local states: "planning", "catch", and "fail". The initial state of Coyote is "planning" and in this local state Coyote is allowed to perform one of the two actions "place Left" or "place Right" denoting where he places the special device. The evolution function for Coyote prescribes that the next local state for Coyote is "catch" if he places the special device in front of the tunnel chosen by RoadRunner.

We introduce the proposition "catch", which is true if Coyote catches RoadRunner. The source code of the example is available from [21]. By using MCMASCoyote discovers that:

$$IS \models \text{init} \rightarrow \langle\langle Coyote \rangle\rangle X(\text{catch})$$

i.e. it is the case that in the initial state Coyote has a strategy to catch RoadRunner.

In fact, any external observer could verify that Coyote knows this very well:

$$IS \models K_{\text{Coyote}}(\text{init} \rightarrow \langle\langle Coyote \rangle\rangle X(\text{catch}))$$

Unfortunately, immediately after placing the ACME device in front of the tunnel, Coyote realises that he was assuming a *lucky* guess on where to place the device. Indeed, under the assumption Coyote is uniform $\Gamma = \{\text{Coyote}\}$, the formula turns to be false

$$IS' \not\models_{\Gamma} \text{init} \rightarrow \langle\langle Coyote \rangle\rangle X(\text{catch})$$

and, as expected, Coyote miserably fails in catching RoadRunner.

From RoadRunner's point of view, however, it is more useful and prudent to reason about what the clumsy Coyote may bring about. Thus, RoadRunner should be more interested in the verification of the non-deterministic interpreted system to discover that it is possible that Coyote catches him. In other words, to analyse the scenario from RoadRunner's point of view, we can check:

$$IS \models K_{\text{RoadRunner}}(\text{init} \rightarrow \langle\langle Coyote \rangle\rangle X(\text{catch}))$$

and also

$$IS \models K_{\text{RoadRunner}}(\text{init} \rightarrow EX(\text{catch})).$$

Given this Coyote should perhaps not be laughing so much.

Verification of all the formulae above took 0.19 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; here, 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. We introduced model checking algorithms for these classes together with an implementation. Examples have also been presented to motivate the use of the various classes.

Acknowledgements

We gratefully acknowledge Michael Wooldridge and Wojciech Jamroga for their valuable comments on an earlier version of this paper. Any current mistakes rest with the authors.

7. REFERENCES

[1] T. Agotnes. Action and knowledge in Alternating-time Temporal Logic. *Synthese*, 2005. Special issue on Knowledge, Rationality and Action.

[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] R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transaction on Computers*, 35(8):677–691, 1986.

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

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

[8] 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.

[9] V. Goranko and W. Jamroga. Comparing semantics for logics of multi-agent systems. *Synthese*, 139(2):241–280, 2004.

[10] W. Hoek, W. Jamroga, and M. Wooldridge. A logic for strategic reasoning. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 157–164, New York, NY, USA, 2005. ACM Press.

[11] W. 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.

[12] 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.

[13] W. Jamroga. *Using Multiple Models of Reality. On Agents who Know how to Play Safer.* PhD thesis, University of Twente, Enschede, The Netherlands, 2004.

[14] W. Jamroga and W. van der Hoek. Agents that know how to play. *Fundamenta Informaticae*, 62:1–35, 2004.

[15] G. Jonker. Feasible strategies in alternating-time temporal epistemic logic. Master's thesis, University of Utrech, The Netherlands, 2003.

[16] M. Kacprzak and W. Penczek. A SAT-based approach to unbounded model checking for alternating-time temporal epistemic logic. *Synthese*, 142:203–227, 2004.

[17] A. Lomuscio. *Knowledge Sharing among Ideal Agents*. PhD thesis, School of Computer Science, University of Birmingham, Birmingham, UK, June 1999.

[18] R. C. Moore. A formal theory of knowledge and action. In J. Allen, J. Hendler, and A. Tate, editors, *Readings in Planning*, pages 480–519. Kaufmann, San Mateo, CA, 1990.

[19] W. Nabialek, A. Niewiadomski, W. Penczek, A. Pólrola, 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.

[20] S. Otterloo, W. van der Hoek, and M. Wooldridge. Knowledge as strategic ability. *ENCTS*, 85(2):1–23, 2003.

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

[22] F. Raimondi and A. Lomuscio. Automatic verification of multi-agent systems by model checking via OBDDs. *Journal of Applied Logic*, 2005. To appear in Special issue on Logic-based agent verification.

[23] P. Y. Schobbens. Alternating-time logic with imperfect recall. In *International workshop on Logic and Communication in Multi-Agent Systems (LCMAS03)*, volume 85(2), pages 1–12, 2004.

[24] F. Somenzi. CUDD: CU decision diagram package - release 2.4.0. http:/ /vlsi.colorado.edu/ ∼fabio/CUDD/cuddIntro.html, 2005.

[25] 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.