Model Checking Epistemic Properties of Interpreted Systems

Franco Raimondi
Department of Computer Science - King's College
Strand, London WC2R 2LS
franco@dcs.kcl.ac.uk

Abstract. Multi-agent systems are often taken as a paradigm in the specification of complex systems because of their ability to abstract away from implementation details. Different kinds of modal logics have been used to model agents' knowledge/beliefs/desires and their evolution with time. In this paper we investigate how model checking techniques can be applied to some problems of verification in multi-agent systems. We present results achieved in the verification of static epistemic properties of two examples encoded in the formalism of Interpreted Systems: the bit transmission problem and the protocol of the dining cryptographers.

1 Introduction

Rational agents in computer science [28] can be seen as the abstraction of any piece of hardware or software enjoying autonomy, social ability, reactivity, and pro-activity. The use of *multi-agent systems* (MAS) in the design of complex systems, where implementations details are unnecessary or unknown [17], has been proven an appealing and successful paradigm: examples include information retrieval using software agents, online auctions, protocols for communication, etc.

In the past twenty years, various logical theories have been developed to formalise *knowledge*, *beliefs*, *desires*, *intentions* and other *intentional attitudes* (in the sense of [9]) of MAS. Such logics include Cohen and Levesque's theory of intention [8], Rao and Georgeff BDI architecture [21], Wooldridge's Logic for Rational Agents (LORA, [26]), and Interpreted Systems by Fagin, Halpern, Moses and Vardi [10].

Comparatively, less effort has been put in the *verification* of MAS. As suggested in [11], model checking techniques can be applied in the verification of MAS. Various results along these lines have been achieved recently by Benerecetti, Wooldrige, Bordini, van der Meyden and others [1, 14, 23, 27, 2, 22, 13, 19]. In this paper we employ Interpreted Systems [10] to specify MAS because they provide a *computationally grounded* theory of agency.

Proceedings of the Eighth ESSLLI Student Session
Balder ten Cate (editor)

The notion of *computationally grounded* theory of agency was introduced by M. Wooldridge in [25] to denote a theory that can be interpreted in terms of some concrete computational model.

By means of two well-known examples from the AI arena (the bit transmission problem [10] and the protocol of the dining cryptographers [5]), we present different methodologies that can be employed in the verification of epistemic, temporal and deontic operators in Interpreted Systems.

The rest of the paper is organised as follows: in the next section we introduce the formalism of Interpreted Systems and model checking. Section 3 presents the main results in model checking "static" formulae. We conclude in Section 4 by suggesting future development and by comparing our work to the state of the art in this field.

2 Preliminaries

Due to space considerations, we refer to [10, 15] for details on Interpreted Systems extended with deontic operators. Model checking is explored fully in [7].

2.1 Interpreted Systems

Consider n agents in a system and n non-empty sets $L_1, \ldots, L_n$ of local states, one for every agent of the system. Elements of L_i will be denoted by $l_1, l_1', l_2, l_2', \ldots$. For every agent of the system there is a set Act_i of actions that the agent can perform. Actions are not executed randomly, but following particular specifications called protocols. A protocol P_i for agent i is a function from the set L_i of local states to a non-empty set of actions Act_i

$$P_i : L_i \to 2^{Act_i}.$$

A *system of global states for n agents S* is a non-empty subset of the cartesian product $L_1 \times \ldots \times L_n$. A global state g of a system S is a tuple of the form $g = (l_1, \ldots, l_n)$. $l_i(g)$ denotes the local state of agent i in global state g.

The evolution of the system can be modelled by means of a transition function π from global states and joint actions to global states:

$$\pi : S \times Act \to S$$

where $Act = Act_1 \times \ldots \times Act_n$ is the set of joint actions for the system. This defines temporal flows on the set of global states. A *run* r is a function from time to global states: $r : N \to S$. Hence, a run is a sequence of global states obtained by applying the function π to global states and joint actions. Notice that, in the definition of π, all the local states are updated synchronously at each temporal step.

Given a set of propositional variables P, an *Interpreted* System of global states is a pair $IS = (S, h)$ where S is a system of global states and

$h : S \rightarrow 2^P$ is an interpretation function; intuitively, h returns the propositional variables true in a global state. Interpreted systems can then be used to model time and knowledge. The logic CTLK (Computation Tree Logic of Knowledge [13]) is an extensions of CTL to include epistemic operators. For instance, epistemic modalities K_i (one for each agent) are interpreted as follows:

$$(IS, g) \models K_i\, \varphi \quad \text{if for all } g' \; l_i(g) = l_i(g') \text{ implies } (IS, g') \models \varphi.$$

(intuitively, $K_i\,\varphi$ captures the idea that agent i knows φ, i.e. φ holds in every state that i considers possible).

The notion of interpreted systems can be extended to incorporate the idea of correct functioning behaviour of some or all of the components [16]. Given n agents and n non-empty sets $G_1, \ldots, G_n$, a *deontic system of global states* is any system of global states defined on $L_1 \supseteq G_1, \ldots, L_n \supseteq G_n$. G_i is called the *set of green states for agent i*. The complement of G_i with respect to L_i, denoted with R_i, is called the *set of red states for agent i*. Deontic systems of global states are used to interpret other modalities on top of CTLK, such as the following one:

$$(IS, g) \models O_i\, \varphi \quad \text{if for all } g' \text{ we have that } l_i(g') \in G_i \text{ implies } (IS, g') \models \varphi.$$

$O_i\,\varphi$ is used to represent that φ holds in all (global) states in which agent i is functioning correctly. Another concept of particular interest is the knowledge that an agent i has *on the assumption that the system (the environment, agent j, group of agents X) is functioning correctly.* We employ the (doubly relativised) modal operator $\widehat{K}_i^j$ for this notion, which is interpreted as follows:

$$(IS, g) \models \widehat{K}_i^j\, \varphi \quad \text{if for all } g' \text{ such that } l_i(g) = l_i(g') \text{ and } l_j(g') \in G_j \text{ we have that } (IS, g') \models \varphi.$$

The resulting logic for modalities K_i is $S5_n$; this models agents with complete introspection capabilities and veridical knowledge. Completeness results can be shown for the O_i as well (see [15]).

2.2 Model Checking and SMV

Given a program P and a property that can be represented as a logical formula φ in some (temporal) logic, model checking techniques allow for the automatic verification of whether or not a model M_P, representing the program P, satisfies the formula φ.

In the last two decades there have been great advances in the effectiveness of this approach thanks to sophisticated data manipulation techniques. Techniques based on Binary Decision Diagrams (BDDs, [3]) have been used to develop model checkers that are able to check large number of states [4]. Alternative approaches using automata have also been developed [24].

Software tools originated from these lines of research. SPIN (see [12]) exploits automata theory and related algorithms, while SMV [18] uses BDDs to represent states and transitions. In this paper we will use NuSMV, a novel implementation of SMV [6].

By means of these softwares a large number of systems ranging from communication protocols to hardware components have been verified. These are not agent systems, but standard distributed processes.

3 Static model checking of Interpreted Systems

In this section we present a methodology to check *epistemic formulae* expressing properties of an Interpreted System. We are interested in the verification of formulae involving epistemic modalities only, and we do not consider temporal operators in the formulae we want to check. We call this methodology *static model checking*.

Given an Interpreted System IS, it is possible to build an SMV program P_{IS} such that the set of states in the temporal model generated by P_{IS} has a one-to-one correspondence with global states of the Interpreted System. The SMV program is constructed as follows[1]:

1. We declare a variable for each agent, where the values of local states are stored.
2. We declare an array of actions for each agent. Intuitively, this array contains the actions "enabled" in a local state, with respect to the protocol for each agent.
3. Actions are synchronised with local states, for each agent and in each state.
4. The conditions for the evolution of the SMV model are obtained from the evolution function of the Interpreted System.
5. Propositions are defined using the function h of the Interpreted System.

The translation into SMV is manual when deontic operators have to be checked, as in Section 3.1. In Section 3.2 the SMV code is generated automatically. In both cases, there is a one-to-one correspondence between global states of the Interpreted System and SMV states.

NuSMV provides an operator to compute the set of "reachable states"; this set is needed to build epistemic and deontic relations between states to evaluate formulae involving epistemic and deontic operators. Specifically, we have built a parser that takes the set of reachable states as input and produces a model with epistemic and deontic relations in the format of Akka, a Kripke model editor[2]. We chose to use Akka because the Kripke

[1] The methodology presented here is a revised version of the one in [14].
[2] `http://turing.wins.uva.nl/~lhendrik/`

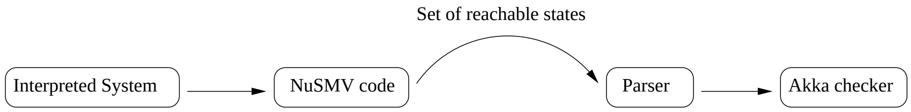

Figure 1.1: The methodology for static model checking.

model can be provided via a simple text file. Moreover, Akka allows for the verification of multiple modalities. Unfortunately, the representation of states in Akka is not symbolic, but in most cases the set of reachable states is orders of magnitude smaller than the full cartesian product of local states, and reachable states are computed symbolically by NuSMV.

The overall methodology is summarised in Fig. 1.1. In our examples, the translation into SMV code, the computation of reachable states with NuSMV, and the parsing of NuSMV output required less than one second on a 1GHz PC with 256MBytes of RAM.

3.1 Example: the bit transmission problem with faults

The bit-transmission problem [10] involves two agents, a *sender* S, and a *receiver* R, communicating over a faulty communication channel. The channel may drop messages but will not flip the value of a bit being sent. S wants to communicate some information—the value of a bit for the sake of the example—to R. One protocol for achieving this is as follows. S immediately starts sending the bit to R, and continues to do so until it receives an acknowledgement from R. R does nothing until it receives the bit; from then on it sends acknowledgements of receipt to S. S stops sending the bit to R when it receives an acknowledgement. Note that R will continue sending acknowledgements even after S has received its acknowledgement. Intuitively S will know for sure that the bit has been received by R when it gets an acknowledgement from R. R, on the other hand, will never be able to know whether its acknowledgement has been received since S does not answer the acknowledgement. Using the methodology suggested we can check mechanically that the protocol above guarantees that when sender receives the acknowledgement it then knows (in the information-theoretic sense defined in Section 2) that the receiver knows the value of the bit (see [14]). We exemplify the methodology with a slightly more complicated scenario: we assume that the receiver R may send acknowledgements even when it is not supposed to. We define the following local states for S and R:

$$L_S = G_S = \{0, 1, (0, ack), (1, ack)\}, \quad R_S = \emptyset.$$

$$G_R = \{0, 1, \epsilon\},\ R_R = \{(0, f), (1, f), (\epsilon, f)\}, L_R = G_R \cup R_R.$$

The local states for S represent the value of the bit S is attempting to transmit (0 or 1), and whether or not S has received an acknowledgement from R ($(0, ack)$ or $(1, ack)$). For R, the local states 0 and 1 represent the value of the received bit; ϵ is a circumstance under which no bit has been received yet; the last three faulty states correspond the fact that R sent a "faulty" acknowledgement. The faulty line can be modelled using another agent. We report here the protocol of the receiver as an example:

$$P_R(\epsilon) = \lambda,$$
$$P_R(0) = P_R(1) = sendack$$
$$P_R((0,f)) = P_R((1,f)) = P_R((\epsilon,f)) = \{sendack, \lambda\}$$

Due to space constraints we cannot report here the full definition of all the parameters (actions, protocols and evolution function), but these can be found in [14] and in the NuSMV code available online at
`http://www.dcs.kcl.ac.uk/pg/franco/is/btpfaults2.smv`.

After translating the Interpreted System into SMV language, we can obtain the set of reachable states running NuSMV; then, with Akka, it is possible to check that none of the epistemic formulae that were true for the basic case (no faults) hold in this version of the protocol. However, a particular form of knowledge still holds. Intuitively if S could make the assumption of R's correct functioning behaviour, then, upon receipt of an acknowledgement, it would make sense for it to assume that R does know the value of the bit; this is exactly the meaning of $\widehat{K}_S^R$. And indeed, using Akka we are able to check the validity of the following formulae in the model, formalising the ideas expressed above[3].

$$IS \models \mathbf{recack} \rightarrow \widehat{K}_S^R(K_R\,(\mathbf{bit} = \mathbf{0}) \vee K_R\,(\mathbf{bit} = \mathbf{1}))$$
$$IS \models \mathbf{recack} \wedge (\mathbf{bit} = \mathbf{0}) \rightarrow \widehat{K}_S^R\, K_R\,(\mathbf{bit} = \mathbf{0})$$

3.2 Example: the protocol of the dining cryptographers

The methodology presented in the previous section can be improved: we can specify an Interpreted System using XML and generate the SMV code mechanically using a Java translator [19]. This improved methodology is illustrated in Fig. 1.2. We did not include deontic operators in the Java translator and in the XML specification, but nevertheless we have been able to check a well-known example, the protocol of the dining cryptographers. The general aim of the protocol is to allow an untraceable broadcasting of messages in multi-agent systems, and is originally introduced with the following example:

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

[3]The interpretation for the atoms can be found in [14].

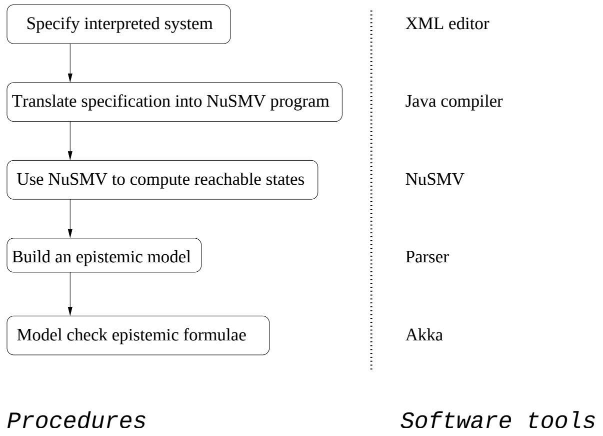

Figure 1.2: An improved methodology for static model checking.

is paying. They resolve their uncertainty fairly by carrying out the following protocol: Each cryptographer flips an unbiased coin behind his menu, between him and the cryptographer on his right, so that only the two of them can see the outcome. Each cryptographer then states aloud whether the two coins he can see – the one he flipped and the one his left-hand neighbour flipped – fell on the same side or on different sides. If one of the cryptographers is the payer, he states the opposite of what he sees. An odd number of differences uttered at the table indicates that a cryptographer is paying; an even number indicates that NSA is paying (assuming that the dinner was paid for only once). Yet if a cryptographer is paying, neither of the other two learns anything from the utterances about which cryptographer it is." [5]

As with the example of the bit transmission problem, we can model the protocol with Interpreted Systems. We introduce three agents C_i, $i = 1, 2, 3$, to model the three cryptographers, and one agent E for the Environment. By running the Java translator, NuSMV, and Akka, we can formally check that the following propositions hold in the Interpreted System of the dining cryptographers (details can be found in [19]):

$$IS \models \mathbf{odd} \rightarrow (\neg \mathbf{paid_1} \rightarrow (K_{C_1}(\mathbf{paid_2} \vee \mathbf{paid_3}) \\ \wedge \\ \neg K_{C_1}(\mathbf{paid_2}) \wedge \neg K_{C_1}(\mathbf{paid_2}))))$$

$$IS \models \mathbf{even} \rightarrow K_{C_1}(\neg \mathbf{paid_1} \wedge \neg \mathbf{paid_2} \wedge \neg \mathbf{paid_3})$$

These two formulae confirm the correctness of the statement: if the first cryptographer did not pay for the dinner and there is an odd number of differences in the utterances, then the first cryptographer knows that either the second or the third cryptographer paid for the dinner; moreover, in this case, the first cryptographer does not know which one of the remaining cryptographers is the payer. Conversely, if the number of differences in the utterances is odd, then the first cryptographer knows that nobody paid for the dinner.

4 Conclusions

In this paper we have presented results on model checking knowledge in multi-agent systems with two examples. Though the methodology applies to "static" formulae only, we think that these results are particularly interesting because we have been able to model check knowledge in a system that has a clear correspondence with computational states of agents, and is logically complete.

We have explored the issue of verification in MAS. Recently, different works have tackled this subject. In [27], M. Wooldridge et al. present the MABLE language for the specification of MAS. Modalities are modelled as nested data structures, in the spirit of [1]. Bordini et al [2] use a modified version of the AgentSpeak(L) language [20] to specify agents and to exploit existing model checkers.

The works of van der Meyden and Shilov [22], and van der Meyden and Su [23], are probably more related to this paper. They consider the verification of a particular class of Interpreted Systems, namely the class of synchronous distributed systems with perfect recall. An algorithm for model checking is introduced in the first paper, but the algorithm is not implemented and specification is not considered. In [23] verification is limited to a specific class of temporal specifications.

We have obtained preliminary results in the verification of a richer language involving temporal operators and epistemic operators, along different extensions of the methodology presented here. In parallel, we are currently investigating theoretical aspects of the $\widehat{K_i^j}$ operator, extending the work of [15].

Bibliography

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

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

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

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

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

[6] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: A new symbolic model verifier. In N. Halbwachs and D. Peled, editors, *Computer Aided Verification*, volume 1633 of *Lecture Notes in Computer Science*, pages 495–??, 1999.

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

[8] P. R. Cohen and H. J. Levesque. Intention is choice with commitment. *Artificial Intelligence, AI*, 42(2–3):213–261, Mar. 1990.

[9] D. C. Dennett. *The intentional stance*. The MIT Press, Massachusetts, 1987. 388 pages, 1987.

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

[11] J. Halpern and M. Vardi. *Model checking vs. theorem proving: a manifesto*, pages 151–176. Artificial Intelligence and Mathematical Theory of Computation. Academic Press, Inc, 1991.

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

[13] A. Lomuscio and W. Penczek. Bounded model checking for interpreted systems. Technical report, Institute of Computer Science of the Polish Academy of Sciences, 2002.

[14] A. Lomuscio, F. Raimondi, and M. Sergot. Towards model checking interpreted systems. In *Proceedings of MoChArt*, Lyon, France, August 2002.

[15] A. Lomuscio and M. Sergot. Investigations in grounded semantics for multi-agent systems specifications via deontic logic. Technical report, Imperial College, London, UK, 2000.

[16] A. Lomuscio and M. Sergot. On multi-agent systems specification via deontic logic. In J.-J. Meyer, editor, *Proceedings of ATAL 2001*. Springer Verlag, July 2001. To Appear.

[17] J. McCarthy. Ascribing mental qualities to machines. In M. Ringle, editor, *Philosophical Perspectives in Artificial Intelligence*. Harvester Press, 1979.

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

[19] F. Raimondi and A. Lomuscio. A tool for specification and verification of epistemic and temporal properties of multi-agent system. Submitted, 2003.

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

[21] A. S. Rao and M. P. Georgeff. Modeling rational agents within a BDI-architecture. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, pages 473–484. Morgan Kaufmann Publishers, Apr. 1991.

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

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

[24] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Symposium on Logic in Computer Science (LICS'86)*, pages 332–345, Washington, D.C., USA, June 1986. IEEE Computer Society Press.

[25] M. Wooldridge. Computationally grounded theories of agency. In E. Durfee, editor, *Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS 2000).* IEEE Press, July 2000.

[26] M. Wooldridge. *Reasoning about rational agents.* MIT Press, July 2000.

[27] M. Wooldridge, M. Fisher, M.-P. Huget, and S. Parsons. Model checking multi-agent systems with MABLE. 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 952–959. ACM Press, July 2002.

[28] M. Wooldridge and N. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2), 1995.