MCMAS: a Model Checker for Multi-Agent Systems

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

1 Overview

This paper presents MCMAS, a model checker for Multi-Agent Systems (MAS). Differently from traditional model checkers, MCMAS permits the automatic verification of specifications that use epistemic, correctness, and cooperation modalities, in addition to the standard temporal modalities. These additional modalities are used to capture properties of various scenarios (including communication and security protocols, games, etc.) that may be difficult or unnatural to express with temporal operators only; a small number of applications are presented in Section 4. Agents are described in MCMAS by means of the dedicated programming language ISPL (Interpreted Systems Programming Language). The approach is symbolic and uses ordered binary decision diagrams (OBDDs), thereby extending standard techniques for temporal logic to other modalities distinctive of agents. MCMAS and all the examples presented in this paper are available for download [14] under the terms of the GPL license.

2 Theoretical background

Interpreted systems [5] provide the formal semantics for MCMAS programs. In the formalism of interpreted systems, each agent is characterised by a set of *local states* and by a set of local *actions* that are performed following a local *protocol*. Given a set of initial states, the system evolves in compliance with an evolution function that determines how the local state of an agent changes as a function of its local state and of the other agents' actions. The evolution of all the agents' local states describes a set of *runs* and a set of *reachable states*. These can be used to interpret formulae involving temporal operators, epistemic operators to reason about what agents *know*, operators to reason about the *correct behaviour* of the agents, and ATL operators expressing states of affairs that agents can enforce. Due to space limitations, we refer to [5, 13, 10, 1, 7] for a detailed presentation of this formalism, and for theoretical results on completeness, decidability, complexity, etc.

Interpreted systems' specifications can be given by means of ISPL programs: a simple example is depicted in Figure 1 (a). We refer to the files available online for the full syntax of ISPL.

Given an interpreted system and a formula in the syntax of ISPL, MCMAS computes the set of states in which the formula holds and compares it to the set of reachable states. The methodology used to calculate this set extends the standard fix-point boolean characterisation for temporal operators [4] to epistemic, correctness, and cooperation operators. We refer to [15] for more details.

3 Implementation

Figure 1 (b) shows the structure of the implementation of MCMAS. The tool can be run from the command line and accepts various options to modify verbosity, to inspect OBDDs statistics [16] and memory usage, and to enable variable reordering in the

* Corresponding author. The authors acknowledge EPSRC grants CN04/04 and GR/S49353/01.

OBDDs. The tool is written in C/C++ and it has been compiled on various platforms, including PowerPC (Mac OS X 10.2 and 10.3), x86 (various CPUs running Linux 2.4 and 2.6), SPARC (SunOS 5.8 and 5.9), and Windows using Cygwin. The source code has been compiled with gcc/g++ from version 2.95 until version 3.4.

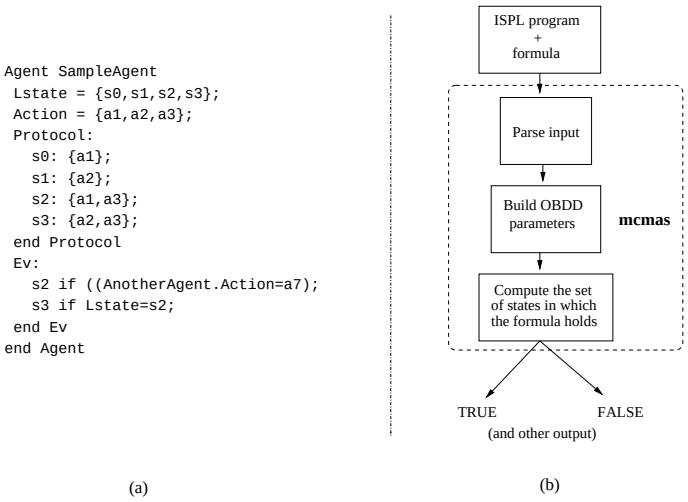

Fig. 1. Implementation structure and ISPL example

4 Examples and experimental results

Various ISPL programs are available for download from [14]. We consider here three of them to illustrate different verification scenarios.

Communication protocol: the bit transmission protocol with faults. In this example from [5], an agent (the Sender) wants to communicate the value of a bit to another agent (the Receiver) using a faulty line that may drop messages. To achieve this, the Sender starts sending messages to the Receiver; when the Receiver receives the bit, it starts sending acknowledgements back to the Sender. The protocol terminates when the Sender receives the acknowledgement. This scenario can be described in terms of interpreted systems, and it is easy to verify with MCMAS the key specification of the protocol: $\mathbf{recack} \rightarrow AG(K_S(K_R(\mathbf{bit}).)$
This formula expresses formally that, upon receipt of an acknowledgement, the Sender will forever know that the Receiver knows the value of the bit. This scenario is extended in [11] to include faulty behaviours of the Receiver. In particular, it is possible to model two faulty scenarios. In the first one, the Receiver "forgets" to send acknowledgements when it receives the bit. In the second scenario, the Receiver may send faulty acknowledgements without receiving the bit first. It is possible to verify with MCMAS the key specification of the protocol still holds in the first case, but it fails in the second. More complex specifications, referring explicitly to violations in the local behaviours, can also be verified.

Strategic games: a simple card game. This example is presented in [8] and in [9]: 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. The following ATL formula can be checked by MCMAS: $\langle\langle player \rangle\rangle F(\mathbf{win})$.
The formula expresses that the player may *always* bring about a winning state, by randomly selecting the correct action. In addition to this, MCMAS supports an operator that considers only *feasible strategies* in the sense of [9, 8]. These are strategies that cannot be "guessed". MCMAS correctly verifies that in this example the player *does not* have a feasible strategy to win.

Anonymity example: the protocol of the dining cryptographers. The protocol of the dining cryptographers is introduced in [3] to describe a scenario in which information is exchanged anonymously. The scenario consists of three cryptographers having dinner at a restaurant. When the waiter informs them that the charge has been covered already, they would like to find out whether it is one of them, or the company they work for who paid for it. In order to guarantee the anonymity of the payer (in case it is one of them), they proceed as follows: each of them flips a coin behind a menu on the right hand side of his dish and observes this coin and the coin at his left (flipped by another cryptographer). If the cryptographer did not pay for the dinner, then he announces whether the two coins he can see are equal or different. However, if the cryptographer paid for the dinner, he says the opposite of what he sees. It is possible to check that, if a cryptographer did not pay for the dinner and he hears an odd number of "different" utterances, then he knows that one of the remaining cryptographers paid for the dinner, but he cannot say who. This property is captured by the following formula:
$(\neg \mathbf{paid_1} \wedge \mathbf{odd}) \rightarrow AX(K_{C_1}(\mathbf{paid_2} \vee \mathbf{paid_3}) \wedge \neg K_{C_1}(\mathbf{paid_2}) \wedge \neg K_{C_1}(\mathbf{paid_3}))$
Notice that the same protocol works for any number of cryptographers greater or equal than three, thereby allowing for an evaluation of the scalability of MCMAS. The ISPL code for various instances of the protocol, including some in which cheating cryptographers are introduced, is available form [14].

5 Discussion

Differently from previous approaches [2], MCMAS does not involve the translation nor the reduction of the model checking problem for MAS to other available model checkers. Our technique is based on OBDDs but, differently from [6], we consider various modalities on top of the epistemic and the temporal ones, and our semantics does not assume perfect recall. The tool presented in [12] allows for the verification of temporal, epistemic, and correct behaviour operators but, differently form MCMAS, it uses bounded and un-bounded techniques, and it has a different input language.

Average experimental results for the example of the dining cryptographers on a 2.8 GHz Pentium 4 running Linux 2.4.20 with 1 Gbytes of RAM are presented in Table 1. We see these results as encouraging, considering that optimisation techniques have not been included in MCMAS yet, and that the code is currently under active development. In particular, we aim at including *fairness constraints* in the verification process, and counter-example generation for false formulae.

References

1. R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002.

N.Crypt.	States (n. bool var)	OBDDs nodes	Memory (MBytes)	Time (sec)
3	$\approx 7 \cdot 10^{13} (47)$	$\approx 10^4$	≈ 4.4	0.37
4	$\approx 2 \cdot 10^{18} (63)$	$\approx 6 \cdot 10^4$	≈ 5.2	3.9
5	$\approx 2 \cdot 7.5^{22} (77)$	$\approx 8 \cdot 10^4$	≈ 5.6	12.6
6	$\approx 1.2 \cdot 10^{27} (91)$	$\approx 1.6 \cdot 10^5$	≈ 7.1	64.5
7	$\approx 2 \cdot 10^{31} (105)$	$\approx 1.7 \cdot 10^5$	≈ 7.5	168.8
8	$\approx 1.3 \cdot 10^{36} (121)$	$\approx 1.2 \cdot 10^7$	≈ 450	28788

Table 1. Experimental results.

2. R. Bordini, M. Fisher, C. Pardavila, W. Visser, and M. Wooldridge. Model checking multi-agent programs with CASP. In *Proceedings of the 15th International Conference on Computer Aided Verification (CAV'03)*, volume 2725 of *LNCS*, pages 110–113. Springer-Verlag, 2003.
3. D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.
4. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.
5. R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge*. MIT Press, Cambridge, 1995.
6. 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.
7. 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.
8. W. Jamroga and W. van der Hoek. Agents that know how to play. *Fundamenta Informaticae*, 62:1–35, 2004.
9. G. Jonker. Feasible strategies in alternating-time temporal epistemic logic. Master's thesis, University of Utrech, The Netherlands, 2003.
10. A. Lomuscio and M. Sergot. Deontic interpreted systems. *Studia Logica*, 75(1):63–92, 2003.
11. 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.
12. 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.
13. W. Penczek and A. Lomuscio. Verifying epistemic properties of multi-agent systems via bounded model checking. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-agent systems (AAMAS'03)*, pages 209–216. ACM, 2003.
14. F. Raimondi and A. Lomuscio. MCMAS - A tool for verification of multi-agent systems. http://www.cs.ucl.ac.uk/staff/f.raimondi/MCMAS/.
15. 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.
16. F. Somenzi. CUDD: CU decision diagram package - release 2.4.0. http:/ /vlsi.colorado.edu/ ∼fabio/CUDD/cuddIntro.html.