MCMAS: A Model Checker for the Verification of Multi-Agent Systems*

Alessio Lomuscio[1], Hongyang Qu[1], and Franco Raimondi[2]

[1] Imperial College London, UK
[2] University College London, UK

1 Overview

While temporal logic in its various forms has proven essential to reason about reactive systems, agent-based scenarios are typically specified by considering high-level agents attitudes. In particular, specification languages based on epistemic logic [7], or logics for knowledge, have proven useful in a variety of areas including robotics, security protocols, web-services, etc. For example, security specifications involving anonymity [4] are known to be naturally expressible in epistemic formalisms as they explicitly state the lack of different kinds of knowledge of the principals.

More generally, various extensions of temporal logic have been studied in agents and AI contexts to represent properties of autonomous systems. In addition to epistemic operators, at the very core of these approaches is the importance of deontic modalities expressing norms and compliance/violation with respect to previously agreed commitments, and ATL-like modalities expressing cooperation among agents.

While these languages have been long explored and appropriate semantics developed, until recently there has been a remarkable gap in the availability of efficient symbolic model checking toolkits supporting these. In this paper we describe MCMAS, a symbolic model checker specifically tailored to agent-based specifications and scenarios. MCMAS [12] supports specifications based on CTL, epistemic logic (including operators of common and distributed knowledge) [7], Alternating Time Logic [2], and deontic modalities for correctness [16]. The release described in this abstract is a complete rebuild of a preliminary experimental checker [14]. The model input language includes variables and basic types and it implements the semantics of interpreted systems, thereby naturally supporting the modularity present in agent-based systems. MCMAS implements OBDD-based algorithms optimised for interpreted systems and supports fairness, counter-example generation, and interactive execution (both in explicit and symbolic mode). MCMAS has been used in a variety of scenarios including web-services, diagnosis, and security. MCMAS is released under GNU-GPL.

2 Multi-Agent Systems Formalisms

Multi-Agent Systems (MAS) formalisms are typically built on extensions of computational tree logic (CTL). For the purposes of this abstract we consider specifications

* The research described in this paper is partly supported by the European Commission Framework 6 funded project CONTRACT (IST Project Number 034418).

given in the following language $\mathcal{L}$ built from a set of propositional atoms $p \in P$, and a set of agents $i \in A$ ($G \subseteq A$ denotes a set of agents):

$$\phi ::= \neg\phi \mid \phi \wedge \phi \mid \langle\langle G\rangle\rangle X\phi \mid \langle\langle G\rangle\rangle F\phi \mid \langle\langle G\rangle\rangle[\phi U\psi] \mid K_i\phi \mid D_G\phi \mid C_G\phi \mid O_i\phi.$$

$\mathcal{L}$ extends ATL (hence CTL) by considering epistemic modalities representing "agent i knows ϕ" ($K_i\phi$), "group G has distributed knowledge of ϕ" ($D_G\phi$), "group G has common knowledge of ϕ" ($C_G\phi$), and deontic modalities encoding "whenever agent i is working correctly ϕ holds" ($O_i\phi$). The ATL modalities above are read as customary: $\langle\langle G\rangle\rangle X\phi$ stands for "group G can enforce ϕ at the next step" and $\langle\langle G\rangle\rangle F\phi$ stands for "group G can enforce ϕ at some point in the future". As standard, CTL modalities for AG, AU, AX and their existential counterparts may be derived from the ATL modalities, and, similarly, epistemic modalities for E_G ("everyone in G knows") may be rewritten as a conjunction of appropriate K_i, $i \in G$, formulas. The specification language above is very rich as it includes AI-based modalities representing various notions of knowledge [7], deontic conditions [16], ATL-style modalities for cooperation [2], as well as standard CTL.

A computationally grounded semantics for the family of MAS formalisms above (in the sense of [19], i.e., one in which the interpretation to all modalities is defined in terms of the computational states of the system) can be given by suitably extending *interpreted systems*. Interpreted systems [7], originally proposed for linear time only, are an agent-based semantics where the components, or agents, are defined by a set of possible local states, a set of actions that they may perform according to their local protocol, and transition functions returning the target local state given the current local state and the set of actions performed by all agents. An environment (described similarly to an agent) is also modelled as part of the system. ATL and CTL modalities are interpreted on the induced temporal relation given by the protocols and transition functions [15], the epistemic modalities are defined on the equivalence relations built on the equality of local states [7], and the deontic modalities are interpreted on "green states", i.e., subsets of local states representing states of locally correct behaviour for the agent in question. Specifically, satisfaction for the epistemic modalities is defined by $(IS, s) \models K_i\phi$ iff for all $s' \in S$ we have that $s \sim_i s'$ implies $(IS, s') \models \phi$, where IS is an interpreted system, s, s' reachable global states, and $\sim_i$ is defined on the local equality of global states, i.e., $s \sim_i s'$ iff $l_i(s) = l_i(s')$ where l_i is the function returning the local state of agent i in a given global state. Satisfaction for common knowledge is defined by $(IS, s) \models C_G\phi$ iff for all $s' \in S$ we have that $s \sim^* s'$ implies $(IS, s') \models \phi$, where $\sim^*$ is the reflexive and transitive closure of the union of the relations $\sim_i, i \in G$. We refer to the user manual available from [12] for satisfaction of distributed knowledge D_G and correctness modalities O_i, as well as more details and examples.

The language $\mathcal{L}$ has been used to specify a wide range of scenarios in application areas such as web-services, security, and communication protocols. For example, in a communication protocol we can use $EF(K_{sender}(K_{receiver}(bit = 0)))$ to specify that at some point in the future the sender will know that the receiver knows that the bit being sent is equal to 0; in a game-based setting, we can write $AGO_{p_1}\langle\langle p_1, p_2\rangle\rangle X(p_1\_p_2win)$ to represent that it is always the case that, as long as player 1 is functioning correctly, player1 and player2 can together force a win at any step.

The complexity of the model checking problem of $\mathcal{L}$ against compact representations (e.g., via reactive modules, or ISPL modules as below) is given by its more expensive fragment (ATL) and so it is EXPTIME-complete [9]. Note, however, that the problem of checking its temporal-epistemic-deontic fragment is only PSPACE-complete [13], i.e., the same as CTL [11].

3 The MCMAS Toolkit

MCMAS is implemented in C++ and compiled for all major platforms. It exploits the CUDD [18] library for BDD operations. MCMAS implements standard algorithms for CTL and ATL [3,2], and dedicated BDD-based algorithms for the epistemic and deontic operators [17], in particular, the algorithms for satisfaction for K_i and C_G are sketched in Algorithm 1 (S represents the set of reachable states).

Algorithm 1. Algorithms for $SAT_K(\phi, i)$(left) and $SAT_C(\phi, G)$ (right)

```
1: $X \Leftarrow SAT(\neg\phi)$;
2: $Y \Leftarrow \{s \in S \mid \exists s' \in X$ s.t. $s \sim_i s'\}$;
3: return $\neg Y \cap S$;
```

```
1: $X \Leftarrow \Sigma; Y \Leftarrow SAT(\neg\phi)$;
2: while $X \neq Y$ do
3:     $X \Leftarrow Y$;
4:     $Y \Leftarrow \{s \in S \mid \exists s' \in X$ and $i \in G$ s.t. $s \sim_i s'\}$;
5: end while
6: return $\neg Y \cap S$;
```

A number of optimisations are implemented in MCMAS in an attempt to minimise the memory consumption and verification time. For example, the checker does not build a single OBDD for the global transition relation, but performs any required operation against the local evolutions. Also, MCMAS does not compute the union of equivalence relations $\sim_i$ in Algorithm 1 when checking common knowledge, but instead repeatedly operates on all $\sim_i$. MCMAS provides counterexamples and witnesses for a wide range of formulas including epistemic modalities thereby giving guidance to the user. The algorithm used to return witnesses and counterexamples is novel and inspired by the tree-like construction of [5].

MCMAS takes ISPL (Interpreted Systems Programming Language) descriptions as input. An ISPL file fully describes a multi-agent system (both the agents and the environment), and it closely follows the framework of interpreted systems described above. We refer to the user manual for examples and usage. Essentially, an ISPL agent is described by giving the agents' possible local states, their actions, protocols, and local evolution functions. Local states are defined by using variables of type `Boolean`, `integer`, and `enumeration`. An optional section `RedStates` permits the definition of non-green states by means of any Boolean formula on the variables of the local states to interpret the correctness modalities O_i. The local transition function is given as a set of *evolution items* of the form `A if C`, where `C` is a Boolean condition over local variables, global variables (see below), and actions by the agents and the environment, and `A` is a set of assignments on the agent's local variables. All variables not present in `A` remain constant in the local transition. Any enabling condition and synchronisation among the agents is specified in `C`; note also that any non-deterministic

Table 1. ISPL snippet for the Train/Gate/Controller (Agent T2, similar to Agent T1, is omitted)

```
Agent Environment
Vars: s: {g, r}; end Vars
Actions={E1, L1, E2, L2};
Protocol:
 s=g: {E1, E2};
 s=r: {L1, L2};
end Protocol
Evolution:
 s=g if s=r and ((Action=L1 and T1.Action=L1)
           or (Action=L2 and T2.Action=L2));
 s=r if s=g and ((Action=L1 and T1.Action=E1)
           or (Action=L2 and T2.Action=E2));
end Evolution
end Agent
```

```
Agent T1
Vars: s: {w, t, a}; end Vars
Actions={E1, L1, B1};
Protocol:
 s=w: {E1}; s=t: {L1}; s=a: {B1};
end Protocol
Evolution:
 s=w if s=a and Action=B1;
 s=t if s=w and Action=E1 and
        Environment.Action=E1;
 s=a if s=t and Action=L1 and
        Environment.Action=L1;
end Evolution
end Agent
```

behaviour may be specified by using several evolution items. Compared to that of an agent, an environment definition may have additional features, including the definition of global variables observable to some or all agents. Table 1 shows an example of a self-explanatory ISPL file for the Train/Gate/Controller scenario with two trains.

An ISPL file also contains sections for the definition of the initial states (given by Boolean conditions on the agents' local variables), any fairness constraints, groups of agents (to be used in epistemic and ATL formulas) and the actual formulae in the language $\mathcal{L}$ to be checked. The interpretation for the propositional atoms used in the specifications is also given; among these the predefined atoms `GreenStates`, and `RedStates` have their interpretation fixed to the locally green local states and their set complement respectively. We refer to the user manual for more details and examples.

The graphical user interface (GUI) is an essential part of the MCMAS release. It is built as an Eclipse plug-in and provides a rich number of functionalities, some of which reported below.

ISPL program editing. The GUI guides the user to create and edit ISPL programs by performing dynamic syntax checking (an additional ISPL parser was implemented in ANTLR for this). The GUI also provides outline view, text formatting, syntax highlighting, and content assist automatically.

Interactive execution mode. The user can use MCMAS interactively to explore the model. This can be done both in symbolic and explicit way. The explicit exploration does not require installation of the checker itself and is provided entirely by the GUI. Obviously large models are best explored symbolically. Users can choose which state to visit among the possibilities presented, backtrack, etc.

Counterexample display. The user can launch the verification process via the GUI which, in turns, calls the checker. The GUI shows which specifications are satisfied and which are not. For a wide range of specifications (see the user manual) the user can visualise counterexamples or witnesses (the Graphviz package is used to display the submodel representing the counterexample/witness). The user has a variety of options once a submodel is displayed including inspecting the agents' states in the system, projecting the whole system onto agents, etc.

4 Experimental Results and Conclusions

MCMAS has been used in our group and in a limited number of other institutions to verify a range of scenarios, including agent-based web services, networking protocols, and security protocols. Some of these examples are available from the MCMAS website. To evaluate the tool we discuss the experimental results obtained while verifying the protocol of the *dining cryptographers* [4]. This is a scalable anonymity protocol in which lack of knowledge needs to be preserved following a round of announcements. We refer to [8,10] for more details.

On a Linux x86_64 machine with Intel Core 2 Duo 2.2GHz and 4GB memory, we tested the protocol (code on the website) against two temporal epistemic specifications: $AG((\text{odd} \wedge \neg \text{payer}_1) \rightarrow ((K_{\text{cryptographer}_1} \bigvee_{i=2}^{n} \text{payer}_i) \wedge (\bigwedge_{i=2}^{n} \neg K_{\text{cryptographer}_1} \text{payer}_i)))$, $AG(\text{even} \rightarrow C_{\{\text{cryptographer}_1,\ldots,\text{cryptographer}_n\}} \neg(\bigvee_{i=1}^{n} \text{payer}_i))$. We checked the second formula specifically to evaluate the performance of the tool against common knowledge.

Table 2. Verification results for the dinning cryptographers protocol

n **crypts**	**possible**	**reachable**	**knowledge**		**common knowledge**	
	states	**states**	**bdd memory** (MB)	**time** (s)	**bdd memory** (MB)	**time** (s)
10	1.86×10^{11}	33792	12.5	1	12.5	1
11	2.23×10^{12}	73728	12.4	3	12.6	2
12	2.67×10^{13}	159744	12.8	4	12.9	4
13	3.21×10^{14}	344064	28.2	23	28.4	23
14	3.85×10^{15}	737280	15.8	14	16.1	13
15	4.62×10^{16}	1.57×106	17.1	24	18.0	24
16	5.55×10^{17}	3.34×106	42.3	149	42.3	150
17	6.66×10^{18}	7.07×106	60.0	612	60.0	612
18	7.99×10^{19}	1.49×107	222.8	2959	222.8	2959

The table reports the results for different numbers of cryptographers, indicated in the first column. The size of the state space (equal to 3×12^n) is reported in the second column, and the third reports the number of actual reachable states in the corresponding model. Memory usage and time required for the verification of the two formulas follow in the last four columns respectively.

A direct efficiency comparison with other toolkits is problematic. Apart from the different input languages, other tools with overlapping functionalities support different variable types making any comparison difficult. In terms of pure size of the model, we found that MCMAS can explore the full state space of models whose size is approximately two orders of magnitude larger than most examples available for temporal-epistemic model checkers [8,6,20], and comparable to the size of the models analysed with BDD-based temporal-only model checkers such as NuSMV.

As mentioned in the introduction, MCMAS is a complete reimplementation of the original proof-of-concept described in [14]. Compared to the original prototype, the current version is several orders of magnitude faster. This is due to improved algorithms for the verification of epistemic and ATL modalities, and the computation of the reachable state space. Additionally, the revised input language now enables the user to write code that naturally generates smaller models, e.g., by using globally observable variables in the environment. Several functionalities, e.g., counterexample generation, witnesses, fairness, a fully-fledged GUI, etc., are also now included.

From the point of view of supported functionalities, MCMAS is the only checker we are aware of that supports the specification language $\mathcal{L}$ described above. Epistemic modalities are also treated in [8] although not via an observation-based semantics as here and not with the CUDD package. BMC based approaches for epistemic modalities have also been presented [6]: a comparison with [6] reveals the known advantages and disadvantages of BDD vs SAT-based approaches. Finally, ATL is of course supported by MOCHA [1]. However, MOCHA is an on-the-fly checker tailored to assume/guarantee analysis, whose efficiency crucially depends on the learning of successful decompositions and assumptions for the scenario under analysis.

References

1. Alur, R., Henzinger, T., Mang, F., Qadeer, S., Rajamani, S., Tasiran, S.: MOCHA: Modularity in model checking. In: Y. Vardi, M. (ed.) CAV 1998. LNCS, vol. 1427, pp. 521–525. Springer, Heidelberg (1998)
2. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. Journal of the ACM 49(5), 672–713 (2002)
3. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic model checking: 10^{20} states and beyond. Information and Computation 98(2), 142–170 (1990)
4. Chaum, D.: The dining cryptographers problem: Unconditional sender and recipient untraceability. Journal of Cryptology 1(1), 65–75 (1988)
5. Clarke, E., Lu, Y., Jha, S., Veith, H.: Tree-like counterexamples in model checking. In: the 17^{th} IEEE Symposium on Logic in Computer Science. IEEE Computer Society Press, Los Alamitos (2002)
6. Dembiński, P., Janowska, A., Janowski, P., Penczek, W., Pólrola, A., Szreter, M., Woźna, B., Zbrzezny, A.: VerICS: A tool for verifying Timed Automata and Estelle specifications. In: Garavel, H., Hatcliff, J. (eds.) TACAS 2003. LNCS, vol. 2619, pp. 278–283. Springer, Heidelberg (2003)
7. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Reasoning about Knowledge. MIT Press, Cambridge (1995)
8. Gammie, P., van der Meyden, R.: MCK: Model checking the logic of knowledge. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 479–483. Springer, Heidelberg (2004)
9. van der Hoek, W., Lomuscio, A., Wooldridge, M.: On the complexity of practical atl model checking knowledge, strategies, and games in multi-agent systems. In: Proceedings of AAMAS 2006, pp. 946–947. ACM Press, New York (2006)
10. Kacprzak, M., Lomuscio, A., Niewiadomski, A., Penczek, W., Raimondi, F., Szreter, M.: Comparing BDD and SAT based techniques for model checking Chaum's dining cryptographers protocol. Fundamenta Informaticae 63(2,3), 221–240 (2006)
11. Kupferman, O., Vardi, M.Y., Wolper, P.: An automata-theoretic approach to branching-time model checking. Journal of the ACM 47(2), 312–360 (2000)
12. Lomuscio, A., Qu, H., Raimondi, F.: MCMAS, `http://www-lai.doc.ic.ac.uk/mcmas/`

13. Lomuscio, A., Raimondi, F.: The complexity of model checking concurrent programs against CTLK specifications. In: Proceedings of the 5th international joint conference on Autonomous agents and multiagent systems (AAMAS 2006), pp. 548–550. ACM Press, New York (2006)
14. Lomuscio, A., Raimondi, F.: MCMAS: A model checker for multi-agent systems. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006. LNCS, vol. 3920, pp. 450–454. Springer, Heidelberg (2006)
15. Lomuscio, A., Raimondi, F.: Model checking knowledge, strategies, and games in multi-agent systems. In: Proceedings of AAMAS 2006, pp. 161–168. ACM Press, New York (2006)
16. Lomuscio, A., Sergot, M.: Deontic interpreted systems. Studia Logica 75(1), 63–92 (2003)
17. Raimondi, F., Lomuscio, A.: Automatic verification of multi-agent systems by model checking via OBDDs. Journal of Applied Logic 5(2), 235–251 (2005)
18. Somenzi, F.: CUDD: CU decision diagram package - release 2.4.1 (2005), `http://vlsi.colorado.edu/~fabio/CUDD/cuddIntro.html`
19. Wooldridge, M.: Computationally grounded theories of agency. In: Proceedings of ICMAS, International Conference of Multi-Agent Systems, pp. 13–22. IEEE Press, Los Alamitos (2000)
20. Wooldridge, M., Fisher, M., Huget, M., Parsons, S.: Model checking multiagent systems with MABLE. In: Proceedings of AAMAS 2002, pp. 952–959 (2002)