Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence (IJCAI-23)
Demonstrations Track

Practical Model Reductions for Verification of Multi-Agent Systems

Wojciech Jamroga[1,2] , **Yan Kim**[1]

[1]Interdisciplinary Centre for Security, Reliability, and Trust, SnT, University of Luxembourg
[2]Institute of Computer Science, Polish Academy of Science, Warsaw, Poland
{wojciech.jamroga, yan.kim}@uni.lu

Abstract

Formal verification of intelligent agents is often computationally infeasible due to state-space explosion. We present a tool for reducing the impact of the explosion by means of state abstraction that is (a) easy to use and understand by non-experts, and (b) agent-based in the sense that it operates on a modular representation of the system, rather than on its huge explicit state model.

1 Introduction

Multi-agent systems (MAS) [Wooldridge, 2002; Shoham and Leyton-Brown, 2009] describe interactions of autonomous agents, often assumed to be intelligent and/or rational. With the development of Internet and social networks, the impact of MAS on everyday life is becoming more and more significant. At the same time, their complexity is rapidly increasing. In consequence, formal methods for analysis and verification of MAS are badly needed.

Verification and model reduction. Algorithms and tools for verification have been in constant development for 40 years, with temporal model checking being most popular [Baier and Katoen, 2008; Clarke *et al.*, 2018]. The main obstacle for *practical* use of those techniques is state-space explosion. Model checking of MAS with respect to their *modular representations* ranges from **PSPACE**-complete to undecidable [Schnoebelen, 2003; Jamroga, 2015]. A possible way to mitigate the complexity is by model reductions, such as abstraction refinement [Clarke *et al.*, 2000] and partial-order reduction [Peled, 1993]. Unfortunately, lossless reductions (i.e., ones that produce fully equivalent models) are usually too weak, in the sense that the resulting model is still too large for feasible verification.

Towards practical abstraction. In this work, we revisit the idea of lossy state abstraction [Cousot and Cousot, 1977; Clarke *et al.*, 1994], and in particular *may/must abstraction* [Godefroid *et al.*, 2001] that potentially removes relevant information about the system, but produces arbitrarily small reduced models. Such verification works best with users who are knowledgeable about the application domain, as its conclusiveness crucially depends on what aspects of the model are being removed. Ideally, the user should be a domain expert, which often implies no in-depth knowledge of verification algorithms. This calls for a technique that is easy to use and understand, preferably supported by a Graphical User Interface (GUI). Moreover, the abstraction should be *agent-based* in the sense that it operates on modular representations of the MAS, and does not require to generate the full explicit-state model before the reduction. The theoretical backbone of our abstraction scheme is presented in [Jamroga and Kim, 2022]. Here, we report on the implementation, and show its usefulness through case studies.

Contribution. We propose a tool for reduction of MAS models by removing an arbitrary subset of variables from the model specification. After the user selects the variables to be removed, the tool can produce two new model specifications: one guaranteed to overapproximate, and one to underapproximate the original model. Then, the user can verify properties of the original model by model checking the new specifications with a suitable model checker. Our model specifications are in the form of *MAS Graphs* [Jamroga and Kim, 2022], a variant of automata networks with asynchronous execution semantics and synchronization on joint action labels [Priese, 1983; Jamroga *et al.*, 2020]. As the model checker of choice, we use UPPAAL [Behrmann *et al.*, 2004], one of the few temporal model checkers with GUI.

Our tool provides a simple command-line interface, where the user selects the input file with a model specification prepared in UPPAAL, the variables to be abstracted away, and the abstraction parameters. It outputs a file with the over-(resp. under-)approximating model specification, that can be opened in UPPAAL for scrutiny and verification. The source code and examples are available at `https://tinyurl.com/ijcai-demo`. Importantly, the abstraction uses modular representations for input and output; in fact, it does *not* involve the generation of the global state space at all. To our best knowledge, this is the first tool for practical user-defined model reductions in model checking of MAS.

Related work. The existing implementations of state abstraction for temporal model checking concern mostly automated abstraction. In particular, CEGAR [Clarke *et al.*, 2000; Clarke *et al.*, 2003] has been implemented for NuSMV [Cimatti *et al.*, 2002], and 3-valued abstraction [Godefroid *et al.*, 2001; Godefroid, 2014] was implemented in Yasm [Gurfinkel *et al.*, 2006] and YOGI [Godefroid *et al.*, 2010]. In each case, abstraction involves the generation of the global state space,

which is the main bottleneck when verifying MAS. Other, user-defined abstraction schemes have been defined only theoretically [Shoham and Grumberg, 2004; Ball and Kupferman, 2006; Dams and Grumberg, 2018], and also require to generate all global states and/or transitions. The approaches in [Cohen *et al.*, 2009; Belardinelli *et al.*, 2019] come closest to our work, as they use modular representations of the state space. However, they both need a global representation of the transition space, and no implementation is reported.

2 Formal Background

MAS graphs and templates. To specify the system to be verified, we use *MAS graphs*, based on standard models of concurrency, and compatible with UPPAAL model specifications. A *MAS graph* is a multiset of *agent graphs*, possibly sharing a set of *global variables*. Each agent graph includes finitely many *locations* and *private variables* that, together, define its local state space. Moreover, *edges* between locations determine the local transition relation. Each edge can be labelled with a randomized *selection* command, boolean *precondition*, *synchronisation* command, and/or a *postcondition* updating the values of some variables. A synchronizing edge can only be taken with a complementary one in another agent. An example agent graph is shown in Figure 1.

A *MAS template* treats each agent graph as a template, and specifies the number of its instances that occur in the verified system (each differing only by the value of variable `id`).

Models. Every MAS graph G can be transformed to its *combined MAS graph*: technically, a single agent graph $comb(G)$ given by the asynchronous product of the agent graphs in G. Each location in $comb(G)$ is a tuple of agents' locations in G. Moreover, the set of variables in $comb(G)$ is the union of all variables occurring in G. A *global model* is obtained from $comb(G)$ by unfolding it to the labelled transition system where states are defined by combined locations and valuations of all the variables. Such models are usually huge, and create an important bottleneck in model checking MAS.

Formal verification and model reduction. Our tool addresses model checking of temporal properties expressed in the well known branching-time logic $\mathbf{CTL}^\star$ [Emerson, 1990]. To mitigate the impact of state-space explosion, we use *state abstraction*, i.e., a method that reduces the state space by clustering similar *concrete states* into a single *abstract state*. In order for the scheme to be practical, it must be easy to use, and avoid the generation of the concrete global model. We summarize the details of our abstraction scheme in the next section.

3 Abstraction by Removal of Variables

Our tool employs the abstraction scheme of [Jamroga and Kim, 2022], and produces specifications of two abstract models: a *may-abstraction* (that overapproximates the concrete states and transitions) and a *must-abstraction* (that underapproximates them). Consequently, if a universal $\mathbf{CTL}^\star$ formula is true in the *may*-abstraction, then it must be true in the concrete model, and if it is false in the *must*-abstraction, then it must be false in the concrete model.

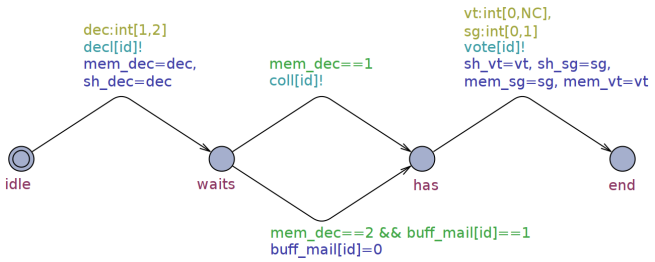

Figure 1: *Voter* template. The agent first declares if she prefers to receive the election package by post (`dec=2`) or in person (`dec=1`). Then, she waits until it can be collected, and casts the ballot together with her voting card. The *select* label for edge `idle→waits` (resp. `has→end`) specifies a nondeterministic choice of the value of variable `dec` $\in \{1, 2\}$ (resp. `vt` $\in \{1, \ldots, NC\}$ and `sg` $\in \{0, 1\}$)

Variable removal. In the simplest variant, the abstraction concerns a complete removal of some variables $V \subseteq Var$ from the model specification. For example, one might remove variables `mem_vt`, `mem_sg` from the agent graph in Figure 1, i.e., the voter's memory of the cast vote and the voting declaration status. Selection of the right variables to remove requires a good understanding of the application domain; we assume that it is provided by the user. Roughly speaking, the abstraction procedure takes the combined MAS graph $comb(G)$, computes an approximation of the reachable values for every $v \in V$, and processes the edges of $comb(G)$ by substituting the occurrences of v at location ℓ with the values $u \in appr(v, \ell)$. If $appr(v, \ell)$ overapproximates (resp. underapproximates) the actual reachable values of v at ℓ, then the resulting model is a *may* (resp. *must*)-abstraction of G.

Variable merge and scoping. More generally, a subset of variables can be merged into a fresh variable by means of a user-defined mapping function. For example, `mem_sg` and `mem_vt` can be merged into a boolean variable `valid` given by `(mem_sg*mem_vt>0)`, indicating the validity of the vote.

Additionally the user can specify the scope of abstraction, i.e., a subset of locations where the abstraction is applied.

Abstraction on MAS templates. In some cases, approximation of variable domains on the combined MAS graph is computationally infeasible. An alternative is to compute it directly on the MAS template by the right approximation of the synchronization edges. On the down side, this sometimes results in largely suboptimal abstract models, i.e., ones more likely to produce inconclusive verification results.

4 Architecture

The main components of the tool are: (1) local domain approximation and (2) generation of abstract model specifications. Additionally, the tool allows to perform simple pre-processing and code analysis, and to store parameters in a configuration file. Each component can be called from command line, possibly followed by a list of arguments:

- `configure`: sets the parameters in the configuration file;
- `unfold`: produces the combined MAS graph;
- `approx`: computes an approximation of the local domain;

#V	Concrete		Abstract (A1)		Abstract (A2)		Abstract (A3)	
	#St	**t**	**#St**	**t**	**#St**	**t**	**#St**	**t**
1	31	0	23	0	22	0	18	0
2	529	0.1	217	0.1	214	0.1	120	0.1
3	10891	0.1	2203	0.1	2440	0.1	838	0.1
4	2.3e+5	0.9	22625	1	29938	0.1	5937	0.1
5	5.1e+6	25	2.3e+5	1	3.7e+5	1	42100	0.6
6	memout		2.3e+6	20	4.9e+6	23	2.9e+5	5
7	memout		2.2e+7	304	memout		2.0e+6	33
8	memout		memout		memout		1.4e+7	357

Table 1: Verification of φ_{bstuff} on models with 3 candidates. $\#V$ is the number of Voter instances. We report the model checking performance for the concrete model, followed by *may*-models obtained by abstractions A1, A2, and A3

#Ag	Concrete		Abstract		
	#St	**t**	**#St**	**Reduct**	**t**
2	165	0	38	76.97	0
3	8917	0.1	555	93.78	0
4	4.6e+5	1.5	10247	97.77	0.1
5	2.1e+7	123	1.5e+5	99.29	1.2
6	memout		2.8e+6	–	42
7	memout		4.1e+7	–	682
8	memout		memout		

Table 2: Verification of φ_{compr} on models of social AI. $\#Ag$ is the number of agents. "Reduct" shows the level of reduction of the state space (in %)

- `abstract`: generates an abstract model specification based on the provided approximation of local domain;
- `info`: lists the variables, locations, and edges in the model.

Local domain approximation. Takes a subset of variables V, a target template ('ext' for the combined MAS graph) and an abstraction type $\mathrm{t} \in \{\text{upper}, \text{lower}\}$, and computes a t-approximation of the local domain over V. The result is saved to a JSON file, where location identifiers are mapped to an array of evaluation vectors.

Abstract model generation. Takes the mapping function with an upper-approximation (resp. lower-approximation) of the local domain, and computes the corresponding may-abstraction (resp. must-abstraction). The mapping function specifies the target agent name or template name, the scope of abstraction, variables to be removed, and possibly a merge variable. We assume that the input provided by the user is correct; some debugging might be added in the future.

5 Experimental Results

We have evaluated the tool by means of experiments on two benchmarks: a simple postal voting scenario and gossip learning for social AI. The model specifications are available for download with the tool. The experiments have been performed in combination with UPPAAL v4.1.24 (32-bit) on a machine with Intel i7-8665U 2.11 GHz CPU, running Ubuntu 22.04. We report the results for *may-abstractions*, typically more useful for universal branching-time properties.

Postal voting. We use a scalable family of MAS graphs proposed in [Kim *et al.*, 2022] to model a simplified postal voting system. The system consists of NV Voters, voting for NC candidates, and a single Election Authority, and proceeds in four subsequent phases: collection of voting declarations, preparation and distribution of election packages, ballot casting, and tallying. The verification concerns a variant of resistance to ballot stuffing, expressed by formula φ_{bstuff}:

```
A[](b_recv<=ep_sent && ep_sent<=NV)
```

where `b_recv` and `ep_sent` are variables storing the number of received ballots and sent election packages, respectively. For the experiments, we try the following abstractions:

A1: removes variables `mem_vt` and `mem_sg` from the Voter template, i.e., the voter's memory of the cast vote and the voting declaration status;

A2: removes variables `mem_dec` at Voter's locations {`has`, `voted`} and variable `dec_recv` at Authority's location {`coll_vts`}, i.e., the information about how the election package has been delivered;

A3: the combination of A1 and A2.

The results in Table 1 present the numbers of states in the global model generated during the verification, as well as the verification running times (in seconds), including the generation of abstract model specifications where applicable. Formula φ_{bstuff} is satisfied in all the reported instances; all three abstractions have been conclusive on it.

Social AI. The second series of experiments uses the specifications of gossip learning for social AI [Heaven, 2013; Hegedüs *et al.*, 2021], proposed in [Kurpiewski *et al.*, 2023]. The system consists of a ring network of AI agents, acting in three phases: data gathering, learning, and sharing of knowledge. The goal of the agents is to collectively reach knowledge of quality $mqual \geq 2$. The system includes also an attacker who can impersonate any agent and fake its quality level. The model specification given as asynchronous MAS [Jamroga *et al.*, 2020] and coded in the input language of the STV model checker [Kurpiewski *et al.*, 2021] was manually translated into the input language of UPPAAL. Afterwards, we hardcoded the attacker's strategy to always share the lowest quality model, and verified formula φ_{compr}:

```
A[](exists(i:int[1,NA])(impersonated!=i &&
    (!AI(i).wait || AI(i).mqual<2))).
```

φ_{compr} says that, on all execution paths, at least one AI agent is compromised. The model checking performance is shown in Table 2. We have been able to conduct verification for concrete models with up to 5 agents (4 honest AI and 1 attacker), and up to 7 agents after applying a *may*-abstraction that discards all variables except for $mqual$ in the AI template.

6 Conclusions

We propose a tool for practical model reductions in multi-agent systems. The tool addresses state-space explosion by removal of selected variables from the model while preserving the truth of **ACTL** formulas. The experiments show significant gains in terms of verification time as well as memory, with minimal time used by the abstraction procedure.

In the future, we plan to extend our tool to abstractions preserving temporal-epistemic and strategic properties in combination with the MCMAS and STV model checkers [Lomuscio *et al.*, 2017; Kurpiewski *et al.*, 2021].

Acknowledgments

The work was supported by NCBR Poland and FNR Luxembourg under the PolLux/FNR-CORE projects STV (POLLUX-VII/1/2019) and SpaceVote (POLLUX-XI/14/SpaceVote/2023), as well as the CHIST-ERA grant CHIST-ERA-19-XAI-010 by NCN Poland (2020/02/Y/ST6/00064).

References

[Baier and Katoen, 2008] C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.

[Ball and Kupferman, 2006] T. Ball and O. Kupferman. An abstraction-refinement framework for multi-agent systems. In *Proceedings of Logic in Computer Science (LICS)*, pages 379–388. IEEE, 2006.

[Behrmann *et al.*, 2004] G. Behrmann, A. David, and K.G. Larsen. A tutorial on UPPAAL. In *Formal Methods for the Design of Real-Time Systems: SFM-RT*, number 3185 in LNCS, pages 200–236. Springer, 2004.

[Belardinelli *et al.*, 2019] Francesco Belardinelli, Alessio Lomuscio, and Vadim Malvone. An abstraction-based method for verifying strategic properties in multi-agent systems with imperfect information. In *Proceedings of AAAI*, pages 6030–6037, 2019.

[Cimatti *et al.*, 2002] A. Cimatti, E.M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, M. Sebastiani, and A Tacchella. NuSMV2: An open-source tool for symbolic model checking. In *Proceedings of Computer Aided Verification (CAV)*, volume 2404 of *Lecture Notes in Computer Science*, pages 359–364, 2002.

[Clarke *et al.*, 1994] E.M. Clarke, O. Grumberg, and D.E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542, 1994.

[Clarke *et al.*, 2000] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *Proceedings of CAV*, volume 1855 of *Lecture Notes in Computer Science*, pages 154–169. Springer, 2000.

[Clarke *et al.*, 2003] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM*, 50(5):752–794, 2003.

[Clarke *et al.*, 2018] E.M. Clarke, T.A. Henzinger, H. Veith, and R. Bloem, editors. *Handbook of Model Checking*. Springer, 2018.

[Cohen *et al.*, 2009] Mika Cohen, Mads Dam, Alessio Lomuscio, and Francesco Russo. Abstraction in model checking multi-agent systems. In *AAMAS (2)*, pages 945–952. Citeseer, 2009.

[Cousot and Cousot, 1977] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages*, pages 238–252, 1977.

[Dams and Grumberg, 2018] Dennis Dams and Orna Grumberg. Abstraction and abstraction refinement. In *Handbook of Model Checking*, pages 385–419. Springer, 2018.

[Emerson, 1990] E.A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 995–1072. Elsevier, 1990.

[Godefroid *et al.*, 2001] P. Godefroid, M. Huth, and R. Jagadeesan. Abstraction-based model checking using modal transition systems. In *Proceedings of CONCUR*, volume 2154 of *Lecture Notes in Computer Science*, pages 426–440, 2001.

[Godefroid *et al.*, 2010] Patrice Godefroid, Aditya V. Nori, Sriram K. Rajamani, and SaiDeep Tetali. Compositional may-must program analysis: unleashing the power of alternation. In *Proceedings of POPL*, pages 43–56. ACM, 2010.

[Godefroid, 2014] Patrice Godefroid. May/must abstraction-based software model checking for sound verification and falsification. In Orna Grumberg, Helmut Seidl, and Maximilian Irlbeck, editors, *Software Systems Safety*, volume 36 of *NATO Science for Peace and Security Series, D: Information and Communication Security*, pages 1–16. IOS Press, 2014.

[Gurfinkel *et al.*, 2006] Arie Gurfinkel, Ou Wei, and Marsha Chechik. Yasm: A software model-checker for verification and refutation. In *Proceedings of CAV*, volume 4144 of *Lecture Notes in Computer Science*, pages 170–174. Springer, 2006.

[Heaven, 2013] Douglas Heaven. Social AI likes to gossip. *New Scientist*, 218(2923):20, 2013.

[Hegedüs *et al.*, 2021] István Hegedüs, Gábor Danner, and Márk Jelasity. Decentralized learning works: An empirical comparison of gossip learning and federated learning. *J. Parallel Distributed Comput.*, 148:109–124, 2021.

[Jamroga and Kim, 2022] Wojciech Jamroga and Yan Kim. Practical abstraction for model checking of multi-agent systems. *arXiv preprint arXiv:2202.12016*, 2022.

[Jamroga *et al.*, 2020] W. Jamroga, W. Penczek, T. Sidoruk, P. Dembiński, and A. Mazurkiewicz. Towards partial order reductions for strategic ability. *Journal of Artificial Intelligence Research*, 68:817–850, 2020.

[Jamroga, 2015] W. Jamroga. *Logical Methods for Specification and Verification of Multi-Agent Systems*. ICS PAS Publishing House, 2015.

[Kim *et al.*, 2022] Yan Kim, Wojciech Jamroga, and Peter Y.A. Ryan. Verification of the socio-technical aspects of voting: The case of the Polish postal vote 2020. In *Proceedings of STAST*, 2022. To appear, available at https://arxiv.org/abs/2210.10694.

[Kurpiewski *et al.*, 2021] Damian Kurpiewski, Witold Pazderski, Wojciech Jamroga, and Yan Kim. STV+Reductions: Towards practical verification of

strategic ability using model reductions. In *Proceedings of AAMAS*, pages 1770–1772. ACM, 2021.

[Kurpiewski *et al.*, 2023] Damian Kurpiewski, Wojciech Jamroga, and Teofil Sidoruk. Towards modelling and verification of social explainable AI. In *Proceedings of ICAART*, 2023. To appear, available at https://arxiv.org/abs/2302.01063.

[Lomuscio *et al.*, 2017] A. Lomuscio, H. Qu, and F. Raimondi. MCMAS: An open-source model checker for the verification of multi-agent systems. *International Journal on Software Tools for Technology Transfer*, 19(1):9–30, 2017.

[Peled, 1993] Doron A. Peled. All from one, one for all: on model checking using representatives. In Costas Courcoubetis, editor, *Proceedings of CAV*, volume 697 of *Lecture Notes in Computer Science*, pages 409–423. Springer, 1993.

[Priese, 1983] L. Priese. Automata and concurrency. *Theoretical Computer Science*, 25:221–265, 1983.

[Schnoebelen, 2003] Ph. Schnoebelen. The complexity of temporal model checking. In *Advances in Modal Logics, Proceedings of AiML 2002*. World Scientific, 2003.

[Shoham and Grumberg, 2004] Sharon Shoham and Orna Grumberg. Monotonic abstraction-refinement for CTL. In *Proceedings of TACAS*, volume 2988 of *Lecture Notes in Computer Science*, pages 546–560. Springer, 2004.

[Shoham and Leyton-Brown, 2009] Y. Shoham and K. Leyton-Brown. *Multiagent Systems - Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2009.

[Wooldridge, 2002] M. Wooldridge. *An Introduction to Multi Agent Systems*. John Wiley & Sons, 2002.