Practical Model Reductions for Verification of Multi-Agent Systems

Wojciech Jamroga\textsuperscript{1,2}, Yan Kim\textsuperscript{1}
\textsuperscript{1}Interdisciplinary Centre for Security, Reliability, and Trust, SnT, University of Luxembourg
\textsuperscript{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 [Schwoebelen, 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 Yasmin [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 UPAL 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 \( \text{comb}(G) \) given by the asynchronous product of the agent graphs in \( G \). Each location in \( \text{comb}(G) \) is a tuple of agents’ locations in \( G \). Moreover, the set of variables in \( \text{comb}(G) \) is the union of all variables occurring in \( G \). A global model is obtained from \( \text{comb}(G) \) 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 CTL* [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 CTL* 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.

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;

![Voter template.](Image)

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

Variable removal. In the simplest variant, the abstraction concerns a complete removal of some variables \( V \subseteq \text{Var} \) from the model specification. For example, one might remove variables \( \text{mem_vt}, \text{mem_sg} \) 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 \( \text{comb}(G) \), computes an approximation of the reachable values for every \( v \in V \), and processes the edges of \( \text{comb}(G) \) by substituting the occurrences of \( v \) at location \( \ell \) with the values \( u \in \text{appr}(v, \ell) \). If \( \text{appr}(v, \ell) \) overapproximates (resp. underapproximates) the actual reachable values of \( v \) at \( \ell \), 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, \( \text{mem_sg} \) and \( \text{mem_vt} \) can be merged into a boolean variable \( \text{valid} \) given by \( (\text{mem_sg} \& \text{mem_vt}) > 0 \), 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.
We report the results for may-abstractions, which have been conclusive on it.

To conduct verification for concrete models with up to 5 agents (4 honest AI and 1 attacker), we have used 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 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, which have been conclusive on it.

### 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, which have been conclusive on it.

### 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].

---

<table>
<thead>
<tr>
<th>V</th>
<th>Concrete</th>
<th>Abstract (A1)</th>
<th>Abstract (A2)</th>
<th>Abstract (A3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>#V</td>
<td>#St</td>
<td>t</td>
<td>#St</td>
<td>t</td>
</tr>
<tr>
<td>1</td>
<td>31</td>
<td>0.1</td>
<td>23</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>10891</td>
<td>0.1</td>
<td>2305</td>
<td>0.1</td>
</tr>
<tr>
<td>3</td>
<td>2.3e+5</td>
<td>0.1</td>
<td>23625</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3.1e+6</td>
<td>0.1</td>
<td>2.3e+6</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>memout</td>
<td>2.3e+6</td>
<td>20</td>
<td>4.9e+6</td>
</tr>
<tr>
<td>7</td>
<td>memout</td>
<td>2.3e+7</td>
<td>304</td>
<td>memout</td>
</tr>
<tr>
<td>8</td>
<td>memout</td>
<td>memout</td>
<td>memout</td>
<td>1.4e+7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#Ag</th>
<th>Concrete</th>
<th>Abstract</th>
</tr>
</thead>
<tbody>
<tr>
<td>#St</td>
<td>t</td>
<td>#St</td>
</tr>
<tr>
<td>2</td>
<td>165</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>8917</td>
<td>0.1</td>
</tr>
<tr>
<td>4</td>
<td>4.6e+5</td>
<td>1.5</td>
</tr>
<tr>
<td>5</td>
<td>2.1e+7</td>
<td>123</td>
</tr>
<tr>
<td>6</td>
<td>memout</td>
<td>2.8e+6</td>
</tr>
<tr>
<td>7</td>
<td>memout</td>
<td>4.1e+7</td>
</tr>
<tr>
<td>8</td>
<td>memout</td>
<td>memout</td>
</tr>
</tbody>
</table>

Table 1: Verification of \( \varphi_{bstuff} \) 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.

- 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 \( t \in \{ \text{upper}, \text{lower} \} \), 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, which have been conclusive on it.

### 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 (POLLEX-VII/1/2019) and SpaceVote (POLLEX-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


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


