Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning
Main Track

Practical Abstraction for Model Checking 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

Model checking of multi-agent systems (MAS) is known to be hard, both theoretically and in practice. A smart abstraction of the state space may significantly reduce the model, and facilitate the verification. We propose and study an intuitive agent-based abstraction scheme, based on the removal of variables in the representation of a MAS. This allows to achieve a desired reduction of a state space without generating the global model of the system. Moreover, the process is easy to understand and control even for domain experts with little knowledge of computer science. We formally prove the correctness of the approach, and evaluate the gains experimentally on a family of postal voting models.

1 Introduction

Multi-agent systems (MAS) describe interactions of autonomous agents, often assumed to be intelligent and/or rational. The theoretical foundations of MAS are mostly based on modal logic and game theory (Wooldridge 2002; Shoham and Leyton-Brown 2009). In particular, the temporal logics **CTL**, **LTL**, and $\mathbf{CTL}^\star$ provide formalizations of many relevant properties, including reachability, liveness, safety, and fairness (Emerson 1990). Algorithms and tools for verification have been in constant development for 40 years, with temporal model checking being the most popular approach (Baier and Katoen 2008; Clarke et al. 2018).

Complexity and state-space explosion. However, formal verification of MAS is known to be hard, both theoretically and in practice. The state-space explosion is a major obstacle here, as faithful models of real-world systems are huge and infeasible even to generate, let alone verify. In consequence, model checking of MAS with respect to their *modular representations* ranges from **PSPACE**-complete to undecidable (Schnoebelen 2003; Bulling et al. 2010). No less importantly, it is often unclear how to create the input model, especially if the system to be modelled involves human behaviour (Jamroga et al. 2020b). Specification is error-prone and difficult to debug and validate, and most model-checkers for MAS do not even have a graphical user interface.[1] In realistic cases, one does not really know if what is verified and what we *think* we verify are indeed the same thing.

Dealing with state-space explosion. Much work has been done to contain the state-space explosion by smart representation and/or reduction of input models. Symbolic model checking based on SAT- or BDD-based representations of the state/transition space (McMillan 1993; McMillan 2002; Penczek and Lomuscio 2003; Kacprzak et al. 2004; Lomuscio and Penczek 2007; Huang and van der Meyden 2014; Lomuscio et al. 2017) fall into the former group. Model reduction methods include partial-order reduction (Peled 1993; Gerth et al. 1999; Jamroga et al. 2020a), equivalence-based reductions (de Bakker et al. 1984; Alur et al. 1998; Belardinelli et al. 2021), and state abstraction (Cousot and Cousot 1977), see Section 2 for a detailed discussion.

Towards practical abstraction. A smart abstraction of the state space may reduce the model to manageable size by clustering "similar" concrete states into *abstract states*, which should facilitate verification. Unfortunately, such clustering may remove essential information from the model, thus making the verification of the abstract model inconclusive for the original model. Lossless abstractions can be obtained by means of abstraction-refinement (Clarke et al. 2000) but, typically, they are difficult to compute or provide insufficient reduction of the model – quite often both.

In consequence, one has to live with abstractions that only approximate the concrete model. Moreover, crafting a good abstraction is an art that relies on the domain expertise of the modeller. Since domain experts are seldom computer scientists or specialists in formal methods, the theoretical formulation of abstraction as an arbitrary mapping from the concrete to the abstract state space has little appeal. Moreover, model checking tools typically do not support abstraction, so doing one would require to manipulate the input specification code, which is a difficult task in itself. What we need is a simple and intuitive methodology for selecting information to be removed from a MAS model, and for its automated removal that preserves certain guarantees. Last but not least, practical abstraction should be applied on modular representations of MAS, unlike the theoretical concept that is usually defined with respect to explicit models of global states.

Contribution. In this paper, we suggest that the conceptually simplest kind of abstraction consists in removing a domain

---

[1] Notable exceptions include UPPAAL (Behrmann et al. 2004) and STV (Kurpiewski et al. 2021).

variable from the specification of the input model. This can be generalized to the merging of several variables into a single one, and possibly clustering their valuations. It is also natural to restrict the scope of abstraction to a part of the input graph. As the main technical contribution, we propose a correct-by-design method to generate such abstractions. We prove that the abstractions preserve the valuations of temporal formulae in Universal $\mathbf{CTL}^\star$ ($\mathbf{ACTL}^\star$). More precisely, our *may*-abstractions preserve the falsity of $\mathbf{ACTL}^\star$ properties, so if $\varphi \in \mathbf{ACTL}^\star$ holds in the abstract model, it must also hold in the original one. Conversely, our *must*-abstractions preserve the truth of $\mathbf{ACTL}^\star$ formulae, so if $\varphi \in \mathbf{ACTL}^\star$ is false in the abstract model, it must also be false in the original one. We evaluate the efficiency of the method by verifying a scalable model of postal voting in UPPAAL. The experiments show that the method is user-friendly, compatible with a state of the art verification tool, and capable of providing significant computational gains.

2 Related Work

State abstraction was introduced in the 1970s (Cousot and Cousot 1977), and studied intensively in the context of temporal properties (Clarke et al. 1994; Godefroid and Jagadeesan 2002). Automatically generated lossless abstractions have been defined through abstraction-refinement (Dams and Grumberg 2018; Clarke et al. 2000; Shoham and Grumberg 2004). In particular, counterexample-guided abstraction refinement was proposed in (Clarke et al. 2000; Clarke et al. 2003), and implemented in NuSMV (Cimatti et al. 2002). Unfortunately, lossless abstraction often results in abstract models that are still too large for practical verification. In this paper, we focus on lossy may/must abstractions, based on user-defined equivalence relations.

This kind of abstractions have been studied in (Dams et al. 1997; Godefroid et al. 2001; Godefroid and Jagadeesan 2002; Godefroid 2014), and implemented in Yasm (Gurfinkel et al. 2006) and YOGI (Godefroid et al. 2010). More specific variants for multi-agent systems were also proposed in (Enea and Dima 2008; Cohen et al. 2009; Lomuscio et al. 2010b). Moreover, abstractions for strategic properties have been investigated in (de Alfaro et al. 2004; Ball and Kupferman 2006), and specifically for MAS in (Kouvaros and Lomuscio 2017; Belardinelli and Lomuscio 2017; Belardinelli et al. 2019). In all those cases, the abstraction method is defined directly on the concrete model, i.e., it requires to first generate the concrete global states and transitions, which is exactly the bottleneck that we want to avoid.[2] In contrast, our method operates on modular (and compact) model specifications, both for the concrete and the abstract model. Data abstraction methods for infinite-state MAS (Belardinelli et al. 2011; Belardinelli et al. 2017) come somewhat close in that respect, but they still generate explicit abstract models. Moreover, they can be only used to *falsify* universal $\mathbf{CTL}^\star$ formulae, which is arguably the less interesting kind of approximation.

Last but not least, most of the existing works have been defined only theoretically (with the exceptions mentioned above), and their usability has never been considered from the perspective of a user with no intimate knowledge of verification techniques.

3 Preliminaries

We start by introducing the models and formulae which serve as an input to model checking.

3.1 MAS Graphs

To represent the behaviour of a multi-agent system, we use modular representations inspired by reactive modules (Alur and Henzinger 1999), interleaved interpreted systems (Lomuscio et al. 2010a; Jamroga et al. 2020a), and in particular by the way distributed systems are modelled in UPPAAL (Behrmann et al. 2004).

Let *Var* be a finite set of typed variables over finite domains.[3] By *Eval*(*Var*) we denote a set of evaluations, i.e., functions mapping variables $v \in Var$ to values from their domains $dom(v)$. *Cond* is a set of logical conditions (also called *guards*) over *Var*, possibly involving arithmetic operators. Let *Chan* be a finite set of asymmetric one-to-one synchronization channels. We define the set of synchronizations as $Sync = \{c!, c? \mid c \in Chan\} \cup \{-\}$, with $c!$ and $c?$ for sending and receiving on a channel c, respectively, and "$-$" for no synchronization.

Definition 1 (Agent graph)**.** An *agent graph* is a tuple $G = (Loc, Var, l_0, g_0, Act, Effect, \hookrightarrow)$, consisting of:

- *Loc*: a non-empty finite set of *locations*;
- *Var*: a finite set of typed *variables* over finite domains;
- $l_0 \in Loc$: the initial location;
- $g_0 \in Cond$: the initial condition;
- *Act*: a set of *actions*, with $\tau \in Act$ for "do nothing";
- $Effect : Eval(Var) \times Act \mapsto Eval(Var)$: the *effect* of an action. We assume $Effect(\eta, \tau) = \eta$;
- $\hookrightarrow \subseteq Loc \times Label \times Loc$: a set of *labelled edges* with labels from $Label \subseteq Cond \times Sync \times Act$, which will be used to define the local transition relation.

Instead of $(l, labl, l') \in \hookrightarrow$, we will often write $l \xhookrightarrow{g:ch\,\alpha} l'$, where $g = cond(labl)$, $ch = sync(labl)$ and $\alpha = act(labl)$. Also, we will omit $ch = -$.

Each condition $g \in Cond$ can be associated with its set of satisfying evaluations $Sat(g) = \{\eta \in Eval(Var) \mid \eta \models g\}$. An edge labelled by $labl \in Label$ is *locally enabled* for evaluation $\eta \in Eval(Var)$ iff $\eta \models cond(labl)$. For simplicity, we assume that $Sat(g_0) = \{\eta_0\}$, i.e., each variable $v \in Var$ is initialized by its default value $v_0 = \eta_0(v)$.

Furthermore, every action $\alpha \in Act \setminus \{\tau\}$ can be associated with a non-empty sequence of atomic assignments (also called *updates*) of the form $\alpha^{(1)}\alpha^{(2)} \ldots \alpha^{(m)}$.

Without loss of generality, we assume that the variables in $Var = \{v_1, \ldots, v_k\}$ are ordered in an arbitrary way. Thus, the evaluation of $V \subseteq Var$ can be seen as a vector

---

[2] (Cohen et al. 2009; Belardinelli et al. 2019) use modular representations of the concrete state space, but they do need a global representation of the concrete transition space, and they generate the global abstract model explicitly.

[3] We consider only variables with finite domains, in line with most model checking algorithms and tools for MAS.

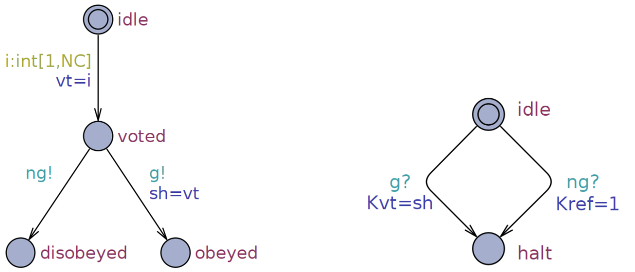
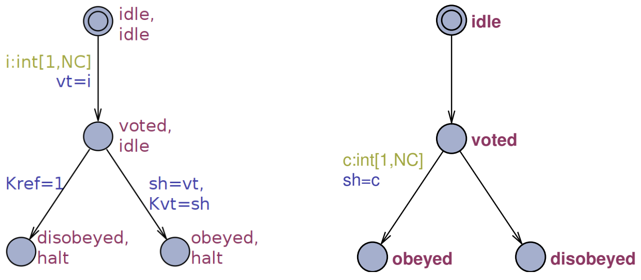


Figure 1: MAS graph for ASV: Voter graph G^{Voter} (left) and Coercer graph $G^{Coercer}$. The set of shared variables is $Var_{sh} = \{sh\}$, and the initial condition $g_0 = (v{=}0)$ for all $v \in Var$



(a) (b)

Figure 2: (a) Combined MAS graph of ASV. (b) *May*-abstraction $\mathcal{A}^{may}_{\{x\}}(G^{Voter}, ASV)$ (right).

$\eta(V) = [\eta(v_{i_1}), \ldots, \eta(v_{i_l})]$ for $i_j \in \{1, \ldots, k\}$, $i_j < i_{j+1}$. Moreover, we say that $\eta_1 \in Eval(Var_1)$ and $\eta_2 \in Eval(Var_2)$ *agree on* variables $V \subseteq Var_1 \cap Var_2$ (denoted $\eta_1 =_V \eta_2$) if $\eta_1(V) = \eta_2(V)$, i.e., $\eta_1(v) = \eta_2(v)$ for all $v \in V$.

Let $V \subseteq Var$ and $r \in Cond \cup Act \cup Eval(Var)$. By $r[V = c]$, we denote the substitution of all free occurrences of variables V in r by the constant vector $c \in dom(V)$. The definition of substitution for guards and updates is straightforward. For an action $\alpha \in Act$, the substitution $\alpha[V = c]$ is more nuanced; the details are presented in Alg. 3.

Definition 2 (MAS graph)**.** A *MAS graph* is a multiset of agent graphs additionally parameterized by a set of shared (global) variables. We assume w.l.o.g. that all local variables have unique names.[4] Then, the set of shared variables can be seen as those that occur in at least two different agent graphs.

Example 1 (ASV)**.** As the running example, we use a variation of the Asynchronous Simple Voting scenario of (Jamroga et al. 2020a). Its MAS graph $ASV = \{\!|Var_{sh}, G^{Voter}, G^{Coercer}|\!\}$ is shown in Fig. 1. The system is parameterized by the number of candidates NC.

The voter starts by nondeterministically selecting one of the candidates (`i:int[1,NC]`), for whom the vote will be cast (`idle`→`voted`). Then, she decides to either give the proof of how she voted to the coercer (`voted`→`obeyed`), or to refuse it (`voted`→`disobeyed`). Both options require executing a synchronous transition (using channels `g` and `ng`) with the coercer. In turn, the coercer either gets the proof and learns for whom the vote was cast, or becomes aware of the voter's refusal.

3.2 Models of MAS Graphs

We define the execution of a MAS graph by its *unwrapping*.

Definition 3 (Combined MAS graph)**.** Let $MG = \{\!|Var_{sh}, G^1, \ldots, G^n|\!\}$ be a MAS graph having a set of shared variables Var_{sh}. The *combined MAS graph* of MG is the agent graph $G_{MG} = (Loc, Var, l_0, g_0, Act, Effect, \hookrightarrow)$, where $Var = \bigcup_{i=1}^{n} Var_i$, $Loc = Loc^1 \times \ldots \times Loc^n$, $l_0 = (l_0^1, \ldots, l_0^n)$, $g_0 = g_0^1 \wedge \ldots \wedge g_0^n$, $Act = \bigcup_{i=1}^{n} Act^i$.

Relation $\hookrightarrow$ is obtained inductively by the following rules (where $l_i, l_i' \in Loc^i$, $l_j, l_j' \in Loc^j$, $c \in Chan^i \cap Chan^j$ for two agent graphs G^i and G^j of distinct indices $1 \leq i, j \leq n$):

$$\frac{l_i \xhookrightarrow{g_i:c!\alpha_i}_i l_i' \wedge l_j \xhookrightarrow{g_j:c?\alpha_j}_j l_j'}{(l_i, l_j) \xhookrightarrow{g_i \wedge g_j:(\alpha_j \circ \alpha_i)} (l_i', l_j')} \qquad \frac{l_i \xhookrightarrow{g_i:\alpha_i}_i l_i'}{(l_i, l_j) \xhookrightarrow{g_i:\alpha_i} (l_i', l_j)}$$

$$\frac{l_i \xhookrightarrow{g_i:c?\alpha_i}_i l_i' \wedge l_j \xhookrightarrow{g_j:c!\alpha_j}_j l_j'}{(l_i, l_j) \xhookrightarrow{g_i \wedge g_j:(\alpha_i \circ \alpha_j)} (l_i', l_j')} \qquad \frac{l_j \xhookrightarrow{g_j:\alpha_j}_j l_j'}{(l_i, l_j) \xhookrightarrow{g_j:\alpha_j} (l_i, l_j')}$$

Lastly, the effect function is defined by:

$$Effect(\alpha, \eta) = \begin{cases} Effect^i(\alpha, \eta) & \text{if } \alpha \in Act^i \\ Effect(\alpha_i, Effect(\alpha_j, \eta)) & \text{if } \alpha = \alpha_i \circ \alpha_j \end{cases}$$

Example 2. The combined MAS graph G_{ASV} for asynchronous simple voting of Example 1 is depicted in Fig. 2a.

Intuitively, the combined MAS graph is an asynchronous composition of the agent graphs in *MG*. Note that by the construction of combined MAS graph, its edges are always labelled by $labl \in Label$, s.t. $sync(labl) = -$. To turn it into a model, we still need to instantiate the variables in combined MAS graph with their possible values.

Definition 4 (Model)**.** A *model* is a tuple $M = (St, I, \longrightarrow, AP, L)$, where St is a set of states, $I \subseteq St$ is a non-empty set of initial states, $\longrightarrow \subseteq St \times St$ is a transition relation, AP is a set of atomic propositions, $L : St \to 2^{AP}$ is a labelling function. We assume $\longrightarrow$ to be serial, i.e., there is at least one outgoing transition at every state. We also assume that St includes only states reachable from I.

Nodes and edges in an agent graph G correspond to *sets* of states and transitions, defined by the unwrapping of G.

Definition 5 (Unwrapping)**.** The *unwrapping* of an agent graph G is a model $\mathcal{M}(G) = (St, I, \longrightarrow, AP, L)$, where:

- $St = Loc \times Eval(Var)$,
- $I = \{\langle l_0, \eta\rangle \in St \mid \eta \in Sat(g_0)\}$,
- $\longrightarrow = \longrightarrow_0 \cup \{(s, s) \in St \times St \mid \neg\exists s' \in St \,.\, s \longrightarrow_0 s'\}$, where $\longrightarrow_0 = \{(\langle l, \eta\rangle, \langle l', \eta'\rangle) \in St \times St \mid \exists\, l \xrightarrow{g:\alpha} l' \,.\, \eta \in Sat(g) \wedge \eta' = Effect(\alpha, \eta)\}$,[5]
- $AP = Loc \cup Cond$,
- $L(\langle l, \eta\rangle) = \{l\} \cup \{g \in Cond \mid \eta \in Sat(g)\}$.

---

[4] This can be achieved, e.g., by prefixing the identifiers of local variable with the name of its agent graph.

[5] We add loops wherever necessary to make the relation serial.

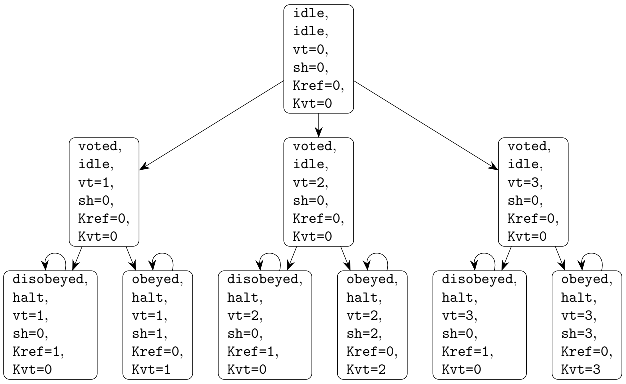


Figure 3: Unwrapping $\mathcal{M}(ASV)$ for ASV with $NC = 3$

The unwrapping $\mathcal{M}(MG)$ of a MAS graph *MG* is given by the unwrapping of its combined graph.

Intuitively, each state in the unwrapping specifies a location in the MAS graph plus a tuple of values for all the variables. Moreover, the atomic statements in AP allows us to indicate a location, or refer to a Boolean condition. By $AP(V)$, we will denote the subset of propositions that do not use any variables from outside V.

Example 3. The unwrapping of the MAS graph for asynchronous simple voting with 3 candidates is shown in Fig. 3.

Definition 6 (Runs, paths, local domain)**.** Let M be a model. A *run* in M is a sequence of states $s_0 s_1 \ldots$, such that $s_i \in St$ and $s_i \longrightarrow s_{i+1}$ for every i. For a finite run $\pi = s_0 s_1 \ldots s_n$, let $len(\pi) = n$ denote its length. By $\pi[k]$ and $\pi[i,j]$ we denote the k-th state of π and the fragment of π from index i to j. A *path* is an infinite run. The sets of all runs in M, all paths in M, and all paths starting from state s are denoted by *Runs*(M), *Paths*(M), and *Paths*(s). Similarly, *Runs*^{t} denotes the set of runs of fixed length $t \in \mathbb{N}^+ \cup \{\infty\}$.

A *local domain* is a function $d : Loc \mapsto \mathcal{P}(Eval(Var))$ that maps each location l to the set of evaluations reachable at l (i.e., for which there exists a corresponding state in the model). By $d(l)|_V = \{\eta(V) \mid \eta \in d(l)\}$ we denote the restriction of $d(l)$ that considers only the values of $V \subseteq Var$.

3.3 Branching-Time Logic $\mathbf{ACTL}^\star$

To specify requirements, we use the *universal fragment of the branching-time logic* $\mathbf{CTL}^\star$ (Emerson 1990), denoted $\mathbf{ACTL}^{\star}$[6] with A ("for every path") as the only path quantifier. The syntax for $\mathbf{ACTL}^\star$ over a set of atomic propositions AP is formally given by:

$$\begin{array}{lcl} \psi & ::= & \top \mid \bot \mid a \mid \neg a \mid \psi \land \psi \mid \psi \lor \psi \mid \mathsf{A}\varphi \\ \varphi & ::= & \psi \mid \varphi \land \varphi \mid \varphi \lor \varphi \mid \mathrm{X}\varphi \mid \varphi \,\mathrm{U}\, \varphi \mid \varphi \,\mathrm{R}\, \varphi \end{array}$$

where $a \in AP$, and $\mathrm{X}, \mathrm{U}, \mathrm{R}$ stand for "next", "until" and "release" respectively. Formulae ψ are called state formulae, and φ are called path formulae. The semantics of $\mathbf{ACTL}^\star$ is given with respect to states s and paths π of a model M.

$$\begin{array}{ll} M, s \models a & \text{iff } a \in L(s) \\ M, s \models \mathsf{A}\varphi & \text{iff } M, \pi \models \varphi \text{ for all } \pi \in \textit{Paths}(s) \\ M, \pi \models \psi & \text{iff } M, \pi[0] \models \psi \\ M, \pi \models \mathrm{X}\varphi & \text{iff } M, \pi[1,\infty] \models \varphi \\ M, \pi \models \varphi_1 \,\mathrm{U}\, \varphi_2 & \text{iff } \exists j \,.\, (M, \pi[j,\infty] \models \varphi_2 \land \\ & \quad \forall 0 \le i < j \,.\, M, \pi[i,\infty] \models \varphi_1) \\ M, \pi \models \varphi_1 \,\mathrm{R}\, \varphi_2 & \text{iff } \forall j \,.\, (M, \pi[j,\infty]) \models \varphi_2 \lor \\ & \quad \exists j \,.\, (M, \pi[j,\infty] \models \varphi_1 \land \\ & \quad \forall 0 \le k \le j \,.\, M, \pi[k,\infty] \models \varphi_1) \end{array}$$

The clauses for Boolean connectives are standard. Additional temporal operators "sometime" and "always" can be defined as $\mathrm{F}\psi \equiv \top \,\mathrm{U}\, \psi$ and $\mathrm{G}\psi \equiv \psi \,\mathrm{U}\, \bot$. Model M satisfies formula ψ (written $M \models \psi$) iff $M, s_0 \models \psi$ for all $s_0 \in I$.

Example 4. Model $M = \mathcal{M}(ASV)$ in Fig. 3 satisfies the $\mathbf{ACTL}^\star$ formula $\mathsf{AG}(\neg\texttt{obeyed} \lor \texttt{Kvt=vt})$, saying that if Voter obeys, Coercer gets to know how she voted, and the formula $\mathsf{AG}(\neg\texttt{disobeyed} \lor \texttt{Kref=1})$, expressing that she cannot disobey Coercer's instructions without his knowledge. It does not satisfy $\mathsf{AF}(\texttt{Kvt>0})$, saying that Coercer will eventually get to know how Voter voted.

4 Variable Abstraction for MAS Graphs

In this section, we propose how to automatically reduce MAS graphs by simplifying their structure of local variables. As the starting point, we take the idea of may/must abstractions (Dams et al. 1997; Godefroid and Jagadeesan 2002). Typically, they take concrete states and cluster them according to a given equivalence relation. The *may* model includes transitions of type $\exists\exists$, i.e., $[s_1] \longrightarrow [s_2]$ in the abstract model iff $\exists_{s_1' \in [s_1]} \exists_{s_2' \in [s_2]} s_1' \longrightarrow s_2'$ in the concrete model. The *must* model includes transitions of type $\forall\exists$, i.e., $[s_1] \longrightarrow [s_2]$ iff $\forall_{s_1' \in [s_1]} \exists_{s_2' \in [s_2]} s_1' \longrightarrow s_2'$. Correctness of the abstraction is proved by showing that the concrete model simulates the *must* model, and is simulated by the *may* model.

4.1 Main Idea

In our case, concrete states are pairs $\langle l, \eta \rangle$. Arguably the simplest equivalence is given by removing a subset of variables $V \subseteq Var$. That is, we will cluster states $\langle l_1, \eta_1 \rangle$ and $\langle l_2, \eta_2 \rangle$ iff $l_1 = l_2$ and η_1, η_2 agree on the variables in $Var \setminus V$.

Moreover, we want the abstraction $\mathcal{A}$ to transform the MAS graph $MG = \{\!|Var_{sh}, G^1, \ldots, G^n|\!\}$ so that:

(i) computation of the abstraction is *agent-based*, i.e., $\mathcal{A}(MG) = \{\!|Var_{sh}, \mathcal{A}(G^1), \ldots, \mathcal{A}(G^n)|\!\}$;
(ii) the abstract agent graphs $\mathcal{A}(G^i)$ have the same structure of locations as their concrete versions G^i;
(iii) the only change results from removal of a subset of local variables V, or simplifying their domains of values.

The may-abstraction $\mathcal{A}^{may}(MG)$ should *over-approximate MG*, in the sense that every transition in *MG* has its counterpart in $\mathcal{A}^{may}(MG)$. Consequently, every formula of type $\mathsf{A}\varphi$ that holds in the model $\mathcal{M}(\mathcal{A}^{may}(MG))$ must also hold in the model $\mathcal{M}(MG)$. Likewise, the must-abstraction $\mathcal{A}^{must}(MG)$ should *under-approximate MG*, in the sense that all transitions in $\mathcal{A}^{must}(MG)$ have their counterparts in *MG*. Thus,

---

[6] Not to be confused with "Action $\mathbf{CTL}$" of (Nicola and Vaandrager 1990).

Algorithm 1: Abstraction of MAS graph MG wrt V

```
for $MG = \{\!| Var_{sh}, G^1, \dots, G^n |\!\}$ compute the combined graph $G_{MG}$
compute the approximate local domain $d$ for $V$ in $G_{MG}$
foreach agent graph $G^i \in MG$ do
    compute abstract graph $\mathcal{A}(G^i)$ w.r.t. $d_i$
return $\mathcal{A}(MG) = \{\!| Var_{sh}, \mathcal{A}(G^1), \dots, \mathcal{A}(G^n) |\!\}$
```

whenever Aφ is false in $\mathcal{M}(\mathcal{A}^{must}(MG))$, it is also false in $\mathcal{M}(MG)$.

The general structure of the procedure is shown in Alg. 1. First, we approximate the set of reachable evaluations $d(l)|_V$ in every location of the combined MAS graph G_{MG} by means of Alg. 2, discussed in Section 4.2. Then, the output is used to transform the agent graphs G^i in MG, one by one, by detecting and transforming the occurrences of the variables in $V \cap Var^i$. This is implemented by function `ComputeAbstraction` (Algorithm 3), which will be presented in detail in Sections 4.3–4.5.

4.2 Approximating the Domains of Variables

Given MAS graph MG, the approximation of reachable values for a set of variables $V \subseteq Var$ is defined in two variants. The upper-approximation of local domain (denoted d^+) for every $l \in Loc$ initializes $d^+(l)|_V = \varnothing$, and then adds new, possibly reachable values of V whenever they are produced on an edge coming to l. The lower-approximation (denoted d^-) initializes $d^-(l)|_V = dom(V)$, and iteratively removes the values might be unreachable. To this end, function `ApproxLocalDomain` is parameterized by symbols d_0 and $\otimes$, such that $d_0 = \varnothing$ and $\otimes = \cup$ for the upper-approximation, and $d_0 = dom(V)$ and $\otimes = \cap$ for the lower-approximation. Note that d_0 is simply a neutral element of the operation $\otimes$.

Furthermore, for an approximation of local domain d^*, where $* \in \{+,-\}$, defined on $Loc = Loc^1 \times \ldots \times Loc^n$, by d^*_i we denote a reduced to the i-th location component "narrowing" of that, where $1 \leq i \leq n$. Intuitively, for $l_j \in Loc_i$ the value of $d^*_i(l_j)$ is defined as $\otimes_{l \in Loc^1 \times \ldots \times Loc^{i-1} \times \{l_j\} \times Loc^{i+1} \times \ldots \times Loc^n} d^*(l)$.

Detailed description of Alg. 2. `ApproxLocalDomain` takes the combined MAS graph G_{MG}, and traverses it using a modified version of a priority-BFS algorithm (Cormen et al. 2009). It begins with the complement of the coarsest approximation of the local domain d_0, and starting from l_0 systematically explores locations of the graph, iteratively refining $d^*(l)|_V$ for $* \in \{+,-\}$ with each visit at l. This proceeds until a stable approximation is obtained. Each location l must be visited at least once, and whenever some of its predecessors l' get their approximations $d^*(l')|_V$ refined, the location l must be processed again.

The max-priority queue Q stores the locations that must be visited (possibly anew). Within the queue, the higher traversal priority is given to locations with greater reachability index $r(l)$, defined as the number of locations $l' \neq l$ reachable from l. This will reduce the number of potential re-visits in comparison with the generic FIFO variant.

Algorithm 2: Approximation of local domain for $V \subseteq Var$

```
ApproxLocalDomain($G = G_{MG}$, $V$)
1      foreach $l \in Loc$ do
2          $l.d := d_0$
3          $l.p := \varnothing$
4          $l.color$ := white
5      $l_0.d := \{\eta(V) \mid \eta \in Sat(g_0)\}$
6      $Q := \varnothing$
7      Enqueue($Q, l_0$)
8      while $Q \neq \varnothing$
9          $l :=$ ExtractMax($Q$)
10         VisitLoc($l, V$)
11         if $l.color \neq black$ then
12             foreach $l' \in Succ^{=1}(l)$ do                    } enqueue immediate-
13                 $Q :=$ Enqueue($Q, l'$)                          } neighbours
14                 $l'.p := l'.p \cup \{l\}$
15             $l.color$ = black
16     return $\{\langle (V, l), l.d \rangle \mid l \in Loc\}$
VisitLoc($l$, $V$)
17     $\kappa := l.d$                                              }
18     foreach $l' \in l.p,\ l' \xhookrightarrow{g:\alpha} l$ do               }
19         $l.d := l.d \otimes$ ProcEdge($l', g, \alpha, l, V$)        } process incoming edges
20     $l.p = \varnothing$                                          }
21     if $\kappa \neq l.d$ then                                    }
22         $l.color$ := grey                                        }
23     $\lambda := l.d$                                             }
24     foreach $l \xhookrightarrow{g:\alpha} l$ do                          }
25         $l.d := l.d \otimes$ ProcEdge($l, g, \alpha, l, V$)         } process self-loops
26     if $\lambda \neq l.d$ then                                   }
27         $l.color$ := grey                                        }
28         go to 23                                                 }
ProcEdge($l$, $g$, $\alpha$, $l'$, $V$)
29     $\delta_0 := \{\eta \in Sat(g) \mid \eta(V) \in l.d\}$
30     let $\alpha := \alpha^{(1)} \dots \alpha^{(m)}$
31     for $i = 1$ to $m$ do
32         $\delta_i := \{\eta' = Effect(\alpha^{(i)}, \eta) \mid \eta \in \delta_{i-1}\}$
33     return $\{\eta(V) \mid \eta \in \delta_{\max(m,0)}\}$
```

The algorithm associates with each location l its attributes $l.colour \in \{white, grey, black\}$, the set of relevant predecessors $l.p \subseteq Loc \setminus \{l\}$, and the current approximation of the local domain $l.d$. The colour indicates if the location has not been visited yet (*white*), its $l.d$ has been refined (*grey*), or it has been visited and closed (*black*). The set $l.p$ indicates which predecessors of l had their approximations updated, which may lead to a refined $l.d$.

In lines 1–5, the locations are initialized with *white*, the empty set of predecessors, and the initial approximation d_0. Lines 6–7 initialize the queue with location l_0. The while-loop of lines 7–15 describes the visit in location l. In `VisitLoc`, after the edges from $l.p$ were taken into account for $l.d$, the $l.p$ is reset (line 20). Self-loops are processed until $l.d$ stabilizes (lines 23–28). The function `ProcEdge` explores the possible transitions, and gradually computes the image (restricted by V) associated with updates from α on evaluations satisfying the guard g and having their V counterpart in $l.d$. Lastly, if l changes its colour to grey from either black or

$l \in Loc$	$r(l)$	$d^-(l)\big\|_{vt}$	$d(l)\big\|_{vt}$	$d^+(l)\big\|_{vt}$
$\langle$`idle,idle`$\rangle$	3	$\{0\}$	$\{0\}$	$\{0\}$
$\langle$`voted,idle`$\rangle$	2	$\varnothing$	$\{1,2,3\}$	$\{1,2,3\}$
$\langle$`obeyed,halt`$\rangle$	0	$\varnothing$	$\{1,2,3\}$	$\{1,2,3\}$
$\langle$`disobeyed,halt`$\rangle$	0	$\varnothing$	$\{1,2,3\}$	$\{1,2,3\}$

Table 1: Reachability index r of locations and reachable values of vt from lower-approximation d^-, exact local domain d and upper-approximation d^+ in ASV with 3 candidates

Algorithm 3: Abstraction by variable removal

```
ComputeAbstraction($G = G^i$, $V$, $d = d_i$)
    $\hookrightarrow_a := \varnothing$
    foreach $l \xhookrightarrow{g:ch\ \alpha} l'$ do
        foreach $c \in d(l)|_V$ do
            $g' := g[V = c]$
            $\delta_0 = \{\eta \in Sat(g) \mid \eta(V) = c\}$
            let $\alpha' = \alpha^{(1)} \dots \alpha^{(m)}$
            for $i = 1$ to $m$ do
                $\delta_i = \{\eta' = Effect(\eta, \alpha^{(i)}) \mid \eta \in \delta_{i-1}\}$
                if $lhs(\alpha^{(i)}) \in V$ then
                    $\alpha^{(i)} := \tau$
            $A = \prod_{i=1}^{m} \{\alpha^{(i)}[V = \eta(V)] \mid \eta \in \delta_i\}$
            $\hookrightarrow_a := \hookrightarrow_a \cup (\bigcup_{\alpha' \in A} \{l \xhookrightarrow{g':ch\ \alpha'} l'\})$
    $\hookrightarrow := \hookrightarrow_a$
    $g_0 := g_0[V = \eta_0(V)]$
    $Var^i := Var^i \setminus V$
    return $G$
```

white, then all the immediate neighbours are enqueued to be inspected, adding l to the list of their relevant predecessors, and changing its colour to black (lines 10–15).

The algorithm halts and returns a stable approximation d (line 16) when the queue is empty and all the locations are *black*. It runs in polynomial time w.r.t. the number of locations and joint valuations of the removed variables. Note that the subsequent approximations $l.d$ are weakly monotonic (i.e., $l.d \subseteq l.d'$ for d^+, and $l.d \supseteq l.d'$ for d^-). Since the sets of locations and edges are finite, and so are the variable domains, termination is guaranteed.

Example 5. The local domain and its approximations obtained by `ApproxLocalDomain` for variable vt in the combined ASV graph of Example 2 can be found in Tab. 1.

4.3 Abstraction by Removal of Variables

The simplest form of abstraction consists in the complete removal of a given subset of variables $V \subseteq Var$ from the MAS graph. To this end, we use the approximation of reachable values of V, produced by `ApproxLocalDomain`. More precisely, we transform every edge between l and l' that includes variables $V' \subseteq V$ in its guard and/or its update into a set of edges (between the same locations), each obtained by substituting V' with a different value $C \in d(l)|_{V'}$, see Alg. 3. The abstract agent graph obtained by removing variables V from G in the context of MG is denoted by $\mathcal{A}_{\{V\}}(G, MG)$. Whenever relevant, we will use $\mathcal{A}^{may}$ (resp. $\mathcal{A}^{must}$) to indicate the variant of the abstraction.

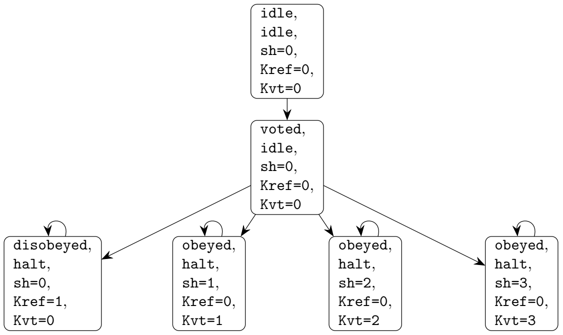


Figure 4: Unwrapping for the *may*-abstraction $\mathcal{A}^{may}_{\{x\}}(ASV) = \{\!|Var_{sh},\ \mathcal{A}^{may}_{\{x\}}(G^{Voter}, ASV),\ G^{Coercer}|\!\}$

Example 6. The result of removing variable vt from the voter graph, according to the upper-approximation of the domain presented in Tab. 1, is shown in Fig. 2b. Note that its unwrapping (shown in Fig. 4) is distinctly smaller than the original one (Fig. 3). Still, as we will formally prove in Section 5, all the paths of the model in Fig. 3 are appropriately represented by the model in Fig. 4.

4.4 Merging Variables and Their Values

A more general variant of variable abstraction assumes a collection of mappings $F = \{f_1, \dots, f_m\}$. Each mapping $f_i : Eval(X_i) \mapsto Eval(z_i)$ merges the local variables $X_i \subseteq Var^j$ of some agent graph G^j to a fresh variable z_i. The abstraction based on f_i removes variables X_i from graph G^j, and replaces them with z_i that "clusters" the values of X_i into appropriate abstraction classes. We will use $Args_R(f_i) = X_i$ and $Args_R(F) = \bigcup_{i=1}^{m} Args_R(f_i)$ to refer to the variables removed by f_i and F. $Args_N(f_i) = \{z_i\}$ and $Args_N(F) = \bigcup_{i=1}^{m} Args_N(f_i)$ refer to the new variables.

Note that the procedure in Section 4.3 can be seen as a special case, with a sole mapping f merging V to a fresh variable z with the singleton domain $dom(z) = \{\eta_0(z)\}$.

4.5 Restricting the Scope of Abstraction

The abstraction scheme can be further generalised by considering a set of mappings $F = \{(f_1, Sc_1), \dots, (f_m, Sc_m)\}$, with each $f_i : Eval(X_i) \mapsto Eval(z_i)$ applied in some agent graph G^j, and $Sc_i \subseteq Loc^j$ defining the scope of f_i. That is, mapping f_i is applied only in the locations $l \in Sc_i$ by assigning $f_i(X_i)$ to z_i, and resetting the value of each $v \in X_i$ to v_0. Outside of Sc_i, the variables in X_i stay intact, and the new variable z_i is assigned an arbitrary default value.

The abstract agent graph obtained by function `ComputeAbstraction` from G in the context of MG via F is denoted by $\mathcal{A}_F(G, MG)$. Consequently, the abstraction of $MG = \{\!|Var_{sh}, G^1, \dots, G^n|\!\}$ becomes

$$\mathcal{A}_F(MG) = \{\!|Var_{sh}, \mathcal{A}_F(G^1, MG) \dots, \mathcal{A}_F(G^n, MG)|\!\}.$$

The general algorithm is presented in detail in Section A.2 of the supplementary material (https://tinyurl.com/3eukkrkb).

5 Correctness of Variable Abstraction

We will now prove that the abstraction scheme preserves the truth values of $\mathbf{ACTL}^{\star}$ formulae if the computation of variable domain d produces the right approximation of their reachable values. In essence, we show that the abstraction always produces an approximation of the runs in the concrete MAS graph, which induces an appropriate simulation relation, and thus guarantees (one-way) preservation of $\mathbf{ACTL}^{\star}$.

5.1 Simulations between Models

We first recall a notion of simulation between models (Baier and Katoen 2008; Clarke et al. 2018; Cohen et al. 2009).

Definition 7. Let $M_i = (\mathit{St}_i, I_i, \longrightarrow_i, \mathit{AP}_i, L_i)$, $i = 1, 2$ be a pair of models, and let $\mathit{AP} \subseteq \mathit{AP}_1 \cap \mathit{AP}_2$ be a subset of atomic propositions. Model M_2 *simulates* model M_1 over AP (written $M_1 \preceq_{\mathit{AP}} M_2$) if there exists a *simulation relation* $\mathcal{R} \subseteq \mathit{St}_1 \times \mathit{St}_2$ over AP, such that:

(i) for every $s_1 \in I_1$, there exists $s_2 \in I_2$ with $s_1 \mathcal{R} s_2$;
(ii) for each $(s_1, s_2) \in \mathcal{R}$:
    (a) $L_1(s_1) \cap \mathit{AP} = L_2(s_2) \cap \mathit{AP}$, and
    (b) if $s_1 \to s_1'$ then there is $s_2 \to s_2'$ such that $s_1' \mathcal{R} s_2'$.

Additionally, for a pair of reachable states s_1, s_2 in M_1, M_2 such that $(s_1, s_2) \in \mathcal{R}$, we say that the pointed model (M_2, s_2) simulates (M_1, s_1) over AP, and denote it by $(M_1, s_1) \preceq_{\mathit{AP}} (M_2, s_2)$.

Theorem 1. *For $(M_1, s_1) \preceq_{\mathit{AP}} (M_2, s_2)$ and any $\mathbf{ACTL}^{*}$ state formula ψ, built of propositions from AP only, it holds that:*

$$M_2, s_2 \models \psi \quad \textit{implies} \quad M_1, s_1 \models \psi \qquad (*)$$

The proof is standard, see e.g. (Baier and Katoen 2008).

Remark. In our abstraction scheme, the set of joint atomic propositions AP, underlying the simulation relation, consists of Boolean conditions and a subset of variables that are not removed from the MAS graph.

5.2 May-Abstractions of MAS Graphs

Let $M_1 = \mathcal{M}(\mathit{MG}_1), M_2 = \mathcal{M}(\mathit{MG}_2)$ be models resulting from unwrapping of MAS graphs $\mathit{MG}_1, \mathit{MG}_2$. We start with a notion of correspondence between states and runs. Then, we use it to define the concept of may-approximation. The following is straightforward.

Lemma 2. *Let $V \subseteq \mathit{Var}$ and $V' = \mathit{Var} \setminus V$, $\alpha \in \mathit{Act}$ and $\mathit{Effect}(\alpha, \eta_1) = \eta_1'$. For any $\eta_2 \in \mathit{Eval}(\mathit{Var})$, we have:*
$\eta_1 =_V \eta_2 \Rightarrow (\mathit{Effect}(\alpha[V' = \eta_1(V')], \eta_2) = \eta_2' \Rightarrow \eta_1' =_V \eta_2')$

Definition 8. Let $s_i \in \mathit{St}_i$ and $s_i = \langle l_i, \eta_i \rangle$ for $i = 1, 2$. State s_2 *corresponds to* a state s_1 over variables $V \subseteq \mathit{Var}_1 \cap \mathit{Var}_2$ (denoted $s_1 \simeq_V s_2$) iff $l_1 = l_2$ and $\eta_1 =_V \eta_2$.

Moreover, run $\pi_2 \in \mathit{Runs}(M_2)$ *corresponds to* run $\pi_1 \in \mathit{Runs}(M_1)$ with respect to V (denoted $\pi_1 \simeq_V \pi_2$) iff:

(i) $\mathit{len}(\pi_1) = \mathit{len}(\pi_2) = t$, and
(ii) for every $1 \leq i \leq t$, it holds that $\pi_1[i] \simeq_V \pi_2[i]$.

5.3 Variable Abstraction Is Sound

We prove now that the abstraction method, based on upper-approximation of local domain, is indeed a simulation.

Lemma 3. *Let MG be a MAS graph and d^+ be an upper-approximation of a local domain defined on $V \subseteq \mathit{Var}$. Then, for any state $\langle l, \eta \rangle$ in $\mathcal{M}(\mathit{MG})$, it must be that $\eta(W) \in d^+(l)|_W$ for any $W \subseteq V$.*

Remark. if $g \in \mathit{Cond}$ and $\mathit{Sat}(g) = \{\eta_1, \ldots, \eta_k\}$, then $g \cong \bigvee_{1 \leq i \leq k} \bigwedge_{v \in \mathit{Var}} (v = \eta(v))$.

Let $\mathit{MG} = \{\!|\mathit{Var}_{sh}, G^1, \ldots, G^n|\!\}$, where $G^i = (\mathit{Var}^i, \mathit{Loc}^i, l_0^i, g_0^i, \mathit{Act}^i, \mathit{Effect}^i, \hookrightarrow^i)$, $G_{\mathit{MG}} = (\mathit{Var}_1, \mathit{Loc}, l_0, g_0, \mathit{Act}, \mathit{Effect}, \hookrightarrow)$, $\hat{\mathit{MG}} = \mathcal{A}_F^{\mathit{may}}(\mathit{MG})$, $G_{\hat{\mathit{MG}}} = (\mathit{Var}_2, \mathit{Loc}, l_0, \hat{g_0}, \mathit{Act}, \mathit{Effect}, \hat{\hookrightarrow})$.

Theorem 4. *Let $M_1 = \mathcal{M}(\mathit{MG})$ and $M_2 = \mathcal{M}(\mathcal{A}_F^{\mathit{may}}(\mathit{MG}))$, s.t. $M_i = (\mathit{St}_i, I_i, \longrightarrow_i, \mathit{AP}_i, L_i)$ for $i = 1, 2$, $V = \mathit{Args}_R(F)$, $Z = \mathit{Args}_N(F)$, $\overline{V} \subseteq \mathit{Var}_1 \cap \mathit{Var}_2$. Then, a relation $\mathcal{R} \subseteq \mathit{St}_1 \times \mathit{St}_2$, where $\langle l_1, \eta_1 \rangle \mathcal{R} \langle l_2, \eta_2 \rangle$ iff $l_1 = l_2 \wedge \eta_1 =_{\overline{V}} \eta_2$, is a simulation relation over $\mathit{AP} = \mathit{AP}_1(\overline{V}) \cap \mathit{AP}_2(\overline{V})$ between M_1 and M_2.*

Proof. Here, we will present a proof for a simpler case - variable removal; proof for a general case is only technically more involved and can be found in supplementary material.

Recall that for $\mathit{Sat}(g_0) = \{\eta_0\}$ it holds $g_0 \cong \bigwedge_{v \in \mathit{Var}_1} v{=}\eta_0(v)$. In variable removal scenario $\mathit{Var}_2 = \mathit{Var}_1 \setminus V$ and $\overline{V} = \mathit{Var}_2$. Therefore $g_0 \cong (\bigwedge_{v \in \overline{V}} v{=}\eta_0(v)) \wedge (\bigwedge_{v \in V} v{=}\eta_0(v))$, $\hat{g_0} \cong (\bigwedge_{v \in \overline{V}} v{=}\hat{\eta_0}(v))$. The construction of $\mathcal{A}_F^{\mathit{may}}$ sets $\hat{g_0} = g_0[V{=}\eta_0(V)]$ (and by assumption there must be at least one evaluation that satisfies that), which means that $g_0 \cong \hat{g_0} \wedge (\bigwedge_{v \in V} v{=}\eta_0(v))$ and $\hat{g_0} \cong (\bigwedge_{v \in \overline{V}} v = \eta_0(v))$. From this and the fact that the sets of locations for MG and $\mathcal{A}_F^{\mathit{may}}(\mathit{MG})$ are the same, we can conclude that the condition (i) of Definition 7 must hold.

Now we show that condition (ii) of Definition 7 holds as well. By construction, each concrete $(l, \mathit{labl}, l') \in\hookrightarrow$ from MG will have (at least one) matching abstract edge $(l, \hat{\mathit{labl}}, l') \in \hat{\hookrightarrow}$, where $\hat{\mathit{labl}} = \mathit{labl}[V{=}c]$ for some $c \in d^+(l)|_V$. Therefore, for any $\langle l, \eta_1 \rangle \mathcal{R} \langle l, \eta_2 \rangle$ and $\langle l, \eta_1 \rangle \longrightarrow_1 \langle l', \eta_1' \rangle$ that was induced by an edge $(l, \mathit{labl}_1, l') \in\hookrightarrow$, where $\eta_1 \models \mathit{cond}(\mathit{labl}_1)$ and $\mathit{Effect}(\mathit{act}(\mathit{labl}_1), \eta_1) = \eta_1'$, there must exist an edge $(l, \mathit{labl}_2, l') \in \hat{\hookrightarrow}$, where $\mathit{labl}_2 = \mathit{labl}_1[V{=}c]$ for some $c \in d^+(l)|_V$, that induces $\langle l, \eta_2 \rangle \longrightarrow_2 \langle l', \eta_2' \rangle$, and by Lem. 2 $\eta_2' =_{\overline{V}} \eta_1'$ and concludes $\langle l', \eta_1' \rangle \mathcal{R} \langle l', \eta_2' \rangle$. ☐

We can now state our main theoretical result.

Theorem 5. *Let MG be a MAS graph, and F a set of mappings as defined in Section 4.5. Then, for every formula ψ of $\mathbf{ACTL}^{\star}$ that includes no variables being removed or added by F:*

$$\mathcal{M}(\mathcal{A}_F^{\mathit{may}}(\mathit{MG})) \models \psi \quad \textit{implies} \quad \mathcal{M}(\mathit{MG}) \models \psi.$$

Proof. Follows directly from Theorems 1 and 4. ☐

5.4 Must-Abstractions of MAS Graphs

An analogous result can be obtained for must-abstraction.

Lemma 6. *Let MG be a MAS graph and* d^- *be a lower-approximation of a local domain defined for* $V \subsetneq Var$*. By the very nature of* d^-*, for any reachable location* $l \in Loc$ *it can have at most one element* $|d^-(l)|_V| \leq 1$*. Moreover, when* $d^-(l)|_V = \{c\}$ *there must exist reachable in* $\mathcal{M}(MG)$ *state* $\langle l, \eta\rangle$*, where* $\eta(V) = c$*.*

Theorem 7. *Let* $M_1 = \mathcal{M}(MG)$ *and* $M_2 = \mathcal{M}(\mathcal{A}^{must}_F(MG))$*, s.t.* $M_i = (St_i, I_i, \longrightarrow_i, AP_i, L_i)$ *for* $i = 1, 2$*,* $V = Args_R(F)$*,* $Z = Args_N(F)$*,* $\overline{V} \subseteq Var_1 \cap Var_2$*. Then, a relation* $\mathcal{R} \subseteq St_2 \times St_1$*, where* $\langle l_2, \eta_2\rangle \mathcal{R} \langle l_1, \eta_1\rangle$ *iff* $l_2 = l_1 \wedge \eta_2 =_{\overline{V}} \eta_1$*, is a simulation relation over* $AP = AP_1(\overline{V}) \cap AP_2(\overline{V})$ *between* M_2 *and* M_1*.*

The proof is analogous to that of Theorem 4, see the supplementary material for details.

Theorem 8. *For each formula* $\psi \in \mathbf{ACTL}^\star$ *including no variables removed by* F*:*

$$\mathcal{M}(\mathcal{A}^{must}_F(MG)) \not\models \psi \textit{ implies } \mathcal{M}(MG) \not\models \psi.$$

5.5 Abstraction on MAS Templates

When some agent graphs in the MAS graph are instantiations of a single template, one can apply abstraction directly on the template. This typically results in a coarser abstraction of the original MAS graph, but such abstractions are exponentially faster to compute, as the size of the model underlying the MAS graph is exponential in the size of the agent template.

Definition 9 (MAS template)**.** A *MAS template* is a compact representation of a MAS graph *MG* as a tuple $MT = (Var_{sh}, Const_{sh}, (GT^1, \#^1), \ldots, (GT^k, \#^k))$ which lists pairs of agent templates GT^i and the number of their instances $\#^i$ in *MG*, as well as the sets of shared variables Var_{sh} and shared constants $Const_{sh}$.

An agent template GT^i is just an agent graph, instantiated in *MG* by $\#^i$ copies through adding their id's $j = 1, \ldots, \#^i$ as prefixes to the locations and local variables in GT^i.

In order to avoid unfolding the MAS template into a MAS graph, we approximate the potential synchronization between instances of agent templates when doing abstraction. More precisely, the upper-approximation of a local domain d_i in agent template GT^i is computed on $upsync(GT^i)$ that discards all the *synchronisation labels* from the edges in GT^i. Analogously, the lower-approximation of a local domain d_i in agent template GT^i is computed on $lowsync(GT^i)$ that discards all the *edges with synchronisation labels* from GT^i.

Theorem 9. *Let* MT *be a MAS template, corresponding to the MAS graph MG. Then* $\mathcal{A}^{may}(upsync(MT))$ *induces a may-abstraction of MG, and* $\mathcal{A}^{must}(lowync(MT))$ *induces a must-abstraction of MG.*

Proof. Follows directly from the fact that discarding synchronisation labels results in a coarser upper-approximation of the local domain, and discarding the edges with synchronisation labels results in a coarser lower-approximation of d_i. ☐

6 Case Study and Experimental Results

We evaluate our abstraction scheme on a real-life scenario.

6.1 Case Study: Integrity of Postal Voting

As input, we use a scalable family of MAS graphs that specify a simplified postal voting system. The system consists of a single agent graph for the Election Authority (depicted in Fig. 5a) and NV instances of eligible Voters (Fig. 5b).

Each voter can vote for one of the NC candidates. The voter starts at the location `idle`, and declares if she wants to receive the election package with the voting declaration and the ballot by post, or to pick it up in person. Then, the voter waits until the package can be collected, which leads to location `has`. At that point, she sends the forms back to the authority, either filled in or blank (e.g., by mistake). The authority collects the voters' intentions (at location `coll_dec`), distributes the packages (at `send_ep`), collects the votes, and computes the tally (at `coll_vts`). A vote is added to the tally only if the declaration is signed and the ballot is filled.

In the experiments, we verify the formula

$$\varphi_{bstuff} \equiv \mathsf{AG}(\textstyle\sum_{\mathsf{i}=1}^{NC} \mathsf{tally[i]} \leq \sum_{\mathsf{j}=1}^{NV} \mathsf{pack\_sent[j]} \leq NV)$$

expressing a variant of *resistance to ballot stuffing*. More precisely, the formula says that the amount of sent packages can never be higher than the number of voters, and there will be no more tallied votes than packages. The formula is satisfied in all considered instances of our voting model.

Due to space limitations, we only present results for *may*-abstraction – arguably, the more important case, since it can be used to prove an $\mathbf{ACTL}^\star$ formula true in a model. Experimental results for *must*-abstraction are shown in the supplementary material.

6.2 Results of Experiments

We have used the following abstractions:

- Abstraction 1: globally removes variables `mem_sg` and `mem_vt`, i.e., the voters' memory of the cast vote and whether the voting declaration has been signed;
- Abstraction 2: removes the voter's memory of her decision (variable `mem_dec`) at locations {`has`, `voted`}, and `dec_recv` at {`coll_vts`};
- Abstraction 3: combines Abstractions 1 and 2.

The verification has been performed with the 32-bit version of UPPAAL 4.1.24 on a laptop with Intel i7-8665U 2.11 GHz CPU, running Ubuntu 22.04. The abstract models were generated using a script in node.js.[7] The results are presented in Table 2. Each row lists the scalability factors (i.e., the number of voters and candidates), the size and verification time for the original model (so called "concrete model"), and the results for Abstractions 1, 2, and 3. "Memout" indicates that the verification process ran out of memory. The columns 'ta' and 'tv' stand for the abstract model generation and verification time, respectively. In all the completed cases, the verification of the abstract model was conclusive (i.e., the output was "true" for all the instances in Table 2).

---

[7] Implementation prototype and utilized models can be found at https://tinyurl.com/363pvpu5 and https://tinyurl.com/3eukkrkb.

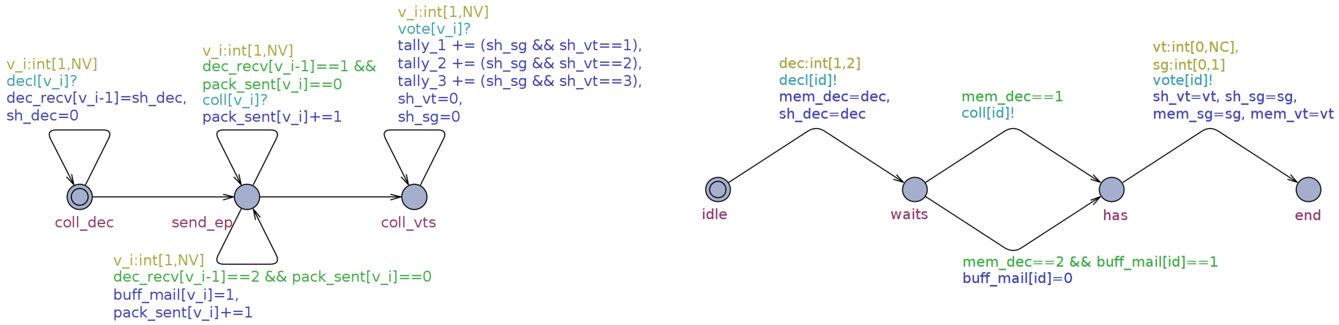


Figure 5: MAS graph for simplified postal voting: (a) Election Authority graph (left), (b) Voter graph (right).

conf	Concrete		Abstract 1			Abstract 2			Abstract 3		
NV,NC	#St	tv (sec)	ta (sec)	#St	tv (sec)	ta (sec)	#St	tv (sec)	ta (sec)	#St	tv (sec)
1,1	2.30e+1	0	0.03	1.90e+1	0	0.07	1.80e+1	0	0.16	1.60e+1	0
1,2	2.70e+1	0	0.03	2.10e+1	0	0.08	2.00e+1	0	0.06	1.70e+1	0
1,3	3.10e+1	0	0.03	2.30e+1	0	0.06	2.20e+1	0	0.05	1.80e+1	0
2,1	2.41e+2	0	0.02	1.41e+2	0	0.06	1.26e+2	0	0.06	9.30e+1	0
2,2	3.69e+2	0	0.02	1.77e+2	0	0.04	1.66e+2	0	0.03	1.06e+2	0
2,3	5.29e+2	0	0.02	2.17e+2	0	0.06	2.14e+2	0	0.04	1.20e+2	0
3,1	2.99e+3	0.01	0.02	1.14e+3	0	0.07	9.72e+2	0.01	0.05	5.67e+2	0
3,2	6.08e+3	0.01	0.02	1.62e+3	0.01	0.05	1.57e+3	0	0.04	6.93e+2	0
3,3	1.09e+4	0.04	0.02	2.20e+3	0.02	0.03	2.44e+3	0	0.05	8.38e+2	0.01
4,1	3.98e+4	0.12	0.02	9.57e+3	0.05	0.08	7.94e+3	0.03	0.08	3.54e+3	0.02
4,2	1.06e+5	0.55	0.01	1.52e+4	0.08	0.08	1.60e+4	0.05	0.06	4.62e+3	0.04
4,3	2.36e+5	0.95	0.01	2.26e+4	0.12	0.08	2.99e+4	0.07	0.08	5.94e+3	0.06
5,1	5.46e+5	1.48	0.02	8.17e+4	0.36	0.19	6.71e+4	0.18	0.25	2.23e+4	0.13
5,2	1.90e+6	6.42	0.02	1.43e+5	0.76	0.18	1.69e+5	0.50	0.23	3.09e+4	0.23
5,3	5.16e+6	24.95	0.02	2.30e+5	1.43	0.22	3.79e+5	1.16	0.22	4.21e+4	0.39
6,1	7.58e+6	31.34	0.01	7.03e+5	4.39	0.55	5.79e+5	1.92	0.44	1.41e+5	0.92
6,2	3.41e+7	170.25	0.01	1.34e+6	10.87	0.50	1.82e+6	7.64	0.40	2.07e+5	1.83
6,3	`memout`		0.01	2.31e+6	20.31	0.84	4.87e+6	22.67	0.40	2.97e+5	4.70
7,1	`memout`		0.01	6.05e+6	46.75	2.34	5.07e+6	22.16	1.91	8.89e+5	8.34
7,2	`memout`		0.02	1.25e+7	149.84	1.33	1.98e+7	107.95	2.01	1.38e+6	16.11
7,3	`memout`		0.02	2.28e+7	304.86	2.49	`memout`		2.35	2.08e+6	30.75
8,1	`memout`		0.02	5.20e+7	482.66	10.30	`memout`		8.04	5.61e+6	66.44
8,2	`memout`		0.19	`memout`		12.17	`memout`		7.58	9.15e+6	150.86
8,3	`memout`		0.07	`memout`		9.52	`memout`		7.99	1.44e+7	348.99
9,1	`memout`		0.12	`memout`		70.49	`memout`		64.96	3.53e+7	474.43
9,2	`memout`		0.06	`memout`		68.46	`memout`		71.69	`memout`	

Table 2: Experimental results for model checking of φ_{bstuff} in may-abstractions of postal voting

The results show significant gains. In particular, for the variant with $NC = 3$ candidates, our *may*-abstractions allowed to reduce the state space by orders of magnitude, and increase the main scalability factor by 3, i.e., to verify up to 9 instead of 6 voters.

7 Conclusions

In this paper, we present a correct-by-design method for model reductions that facilitate formal verification of MAS. Theoretically speaking, our reductions are *agent-based may/must abstractions* of the state space. Crucially, they transform the specification of the system at the level of agent graphs, without generating the global model. No less importantly, they are easy to use, come with a natural methodology, and require almost no technical knowledge from the user. All that the user needs to do is to select a subset of variables to be removed from the MAS graph representing the system. It is also possible to define mappings that merge information stored in local variables of an agent module.

We prove that the abstractions always generate a correct abstract MAS graph, i.e., one that provides a lower (resp. upper) bound for the truth values of formulae to be verified. Moreover, we demonstrate the effectiveness of the method on a case study involving the verification of a postal voting procedure using UPPAAL. As shown in the experiments, simple abstractions allow to verify state spaces larger by several orders of magnitude. Clearly, the efficiency of the method depends on the right selection of variables and the abstraction scope; ideally, that should be provided by a domain expert.

In the future, we want to combine variable abstraction with abstractions that transform locations in a MAS graph. Even more importantly, we plan to extend the methodology from branching-time properties to formal verification of strategic ability (Alur et al. 2002). We also note that the procedure is generic enough to be used in combination with other techniques, such as partial-order reduction (Jamroga et al. 2020a). Finally, an implementation as an extension of the STV model checker (Kurpiewski et al. 2021) is considered.

Acknowledgments

We thank Marius Belly-Le Guilloux, Damian Kurpiewski, Peter Y.A. Ryan, and Masoud Tabatabaei for comments and discussion. The work was supported by NCBR Poland and FNR Luxembourg under the PolLux/FNR-CORE projects STV (POLLUX-VII/1/2019 and C18/IS/12685695/IS/STV/Ryan), SpaceVote (POLLUX-XI/14/SpaceVote/2023 and C22/IS/17232062/SpaceVote) and PABLO (C21/IS/16326754/PABLO), as well as the CHIST-ERA grant CHIST-ERA-19-XAI-010 by NCN Poland (2020/02/Y/ST6/00064).

References

Alur, R., and Henzinger, T. A. 1999. Reactive modules. *Formal Methods in System Design* 15(1):7–48.

Alur, R.; Henzinger, T.; Kupferman, O.; and Vardi, M. 1998. Alternating refinement relations. In *Proceedings of CONCUR*, volume 1466 of *Lecture Notes in Computer Science*, 163–178.

Alur, R.; Henzinger, T. A.; and Kupferman, O. 2002. Alternating-time Temporal Logic. *Journal of the ACM* 49:672–713.

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

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

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

Belardinelli, F., and Lomuscio, A. 2017. Agent-based abstractions for verifying alternating-time temporal logic with imperfect information. In *Proceedings of AAMAS*, 1259–1267. ACM.

Belardinelli, F.; Lomuscio, A.; and Patrizi, F. 2011. Verification of deployed artifact systems via data abstraction. In *Proceedings of ICSOC*, volume 7084 of *Lecture Notes in Computer Science*, 142–156. Springer.

Belardinelli, F.; Kouvaros, P.; and Lomuscio, A. 2017. Parameterised verification of data-aware multi-agent systems. In *Proceedings of IJCAI*, 98–104. ijcai.org.

Belardinelli, F.; Lomuscio, A.; and Malvone, V. 2019. An abstraction-based method for verifying strategic properties in multi-agent systems with imperfect information. In *Proceedings of AAAI*, 6030–6037.

Belardinelli, F.; Condurache, R.; Dima, C.; Jamroga, W.; and Knapik, M. 2021. Bisimulations for verifying strategic abilities with an application to the ThreeBallot voting protocol. *Information and Computation* 276:104552.

Bulling, N.; Dix, J.; and Jamroga, W. 2010. Model checking logics of strategic ability: Complexity. In Dastani, M.; Hindriks, K.; and Meyer, J.-J., eds., *Specification and Verification of Multi-Agent Systems*. Springer. 125–159.

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

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

Clarke, E. M.; Grumberg, O.; Jha, S.; Lu, Y.; and Veith, H. 2000. Counterexample-guided abstraction refinement. In *Proceedings of CAV*, volume 1855 of *Lecture Notes in Computer Science*, 154–169. Springer.

Clarke, E. M.; Grumberg, O.; Jha, S.; Lu, Y.; and Veith, H. 2003. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM* 50(5):752–794.

Clarke, E.; Henzinger, T.; Veith, H.; and Bloem, R., eds. 2018. *Handbook of Model Checking*. Springer.

Cohen, M.; Dam, M.; Lomuscio, A.; and Russo, F. 2009. Abstraction in model checking multi-agent systems. In *Proceedings of (AAMAS*, 945–952. IFAAMAS.

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; and Stein, C. 2009. *Introduction to algorithms*. MIT press.

Cousot, P., and Cousot, R. 1977. 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*, 238–252.

Dams, D., and Grumberg, O. 2018. Abstraction and abstraction refinement. In *Handbook of Model Checking*. Springer. 385–419.

Dams, D.; Gerth, R.; and Grumberg, O. 1997. Abstract interpretation of reactive systems. *ACM Trans. Program. Lang. Syst.* 19(2):253–291.

de Alfaro, L.; Godefroid, P.; and Jagadeesan, R. 2004. Three-valued abstractions of games: Uncertainty, but with precision. In *Proceedings of Logic in Computer Science (LICS)*, 170–179. IEEE Computer Society.

de Bakker, J.; Bergstra, J.; Klop, J. W.; and Meyer, J. C. 1984. Linear time and branching time semantics for recursion with merge. *Theor. Comput. Sci.* 34:135–156.

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

Enea, C., and Dima, C. 2008. Abstractions of multi-agent systems. *International Transactions on Systems Science and Applications* 3(4):329–337.

Gerth, R.; Kuiper, R.; Peled, D.; and Penczek, W. 1999. A partial order approach to branching time logic model checking. In *Proceedings of ISTCS*, 130–139. IEEE.

Godefroid, P., and Jagadeesan, R. 2002. Automatic abstraction using generalized model checking. In *Proceedings of Computer Aided Verification (CAV)*, volume 2404 of *Lecture Notes in Computer Science*, 137–150. Springer.

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

Godefroid, P.; Nori, A. V.; Rajamani, S. K.; and Tetali, S. 2010. Compositional may-must program analysis: unleashing the power of alternation. In *Proceedings of POPL*, 43–56. ACM.

Godefroid, P. 2014. May/must abstraction-based software model checking for sound verification and falsification. In *Software Systems Safety*, volume 36. IOS Press. 1–16.

Gurfinkel, A.; Wei, O.; and Chechik, M. 2006. Yasm: A software model-checker for verification and refutation. In *Proceedings of CAV*, volume 4144 of *Lecture Notes in Computer Science*, 170–174. Springer.

Huang, X., and van der Meyden, R. 2014. Symbolic model checking epistemic strategy logic. In *Proceedings of AAAI Conference on Artificial Intelligence*, 1426–1432.

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

Jamroga, W.; Kim, Y.; Kurpiewski, D.; and Ryan, P. Y. A. 2020b. Towards model checking of voting protocols in uppaal. In *Proceedings of E-Vote-ID*, volume 12455 of *Lecture Notes in Computer Science*, 129–146. Springer.

Kacprzak, M.; Lomuscio, A.; and Penczek, W. 2004. Verification of multiagent systems via unbounded model checking. In *Proceedings of AAMAS*, 638–645. IEEE Computer Society.

Kouvaros, P., and Lomuscio, A. 2017. Parameterised verification of infinite state multi-agent systems via predicate abstraction. In *Proceedings of AAAI*, 3013–3020.

Kurpiewski, D.; Pazderski, W.; Jamroga, W.; and Kim, Y. 2021. STV+Reductions: Towards practical verification of strategic ability using model reductions. In *Proceedings of AAMAS*, 1770–1772. ACM.

Lomuscio, A., and Penczek, W. 2007. Symbolic model checking for temporal-epistemic logics. *SIGACT News* 38(3):77–99.

Lomuscio, A.; Penczek, W.; and Qu, H. 2010a. Partial order reductions for model checking temporal-epistemic logics over interleaved multi-agent systems. *Fundamenta Informaticae* 101(1-2):71–90.

Lomuscio, A.; Qu, H.; and Russo, F. 2010b. Automatic data-abstraction in model checking multi-agent systems. In *Model Checking and Artificial Intelligence*, volume 6572 of *Lecture Notes in Computer Science*, 52–68. Springer.

Lomuscio, A.; Qu, H.; and Raimondi, F. 2017. 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.

McMillan, K. 1993. *Symbolic Model Checking: An Approach to the State Explosion Problem*. Kluwer Academic Publishers.

McMillan, K. 2002. Applying SAT methods in unbounded symbolic model checking. In *Proceedings of Computer Aided Verification (CAV)*, volume 2404 of *Lecture Notes in Computer Science*, 250–264.

Nicola, R. D., and Vaandrager, F. W. 1990. Action versus state based logics for transition systems. In *Semantics of Systems of Concurrent Processes, Proceedings of LITP Spring School on Theoretical Computer Science*, volume 469 of *Lecture Notes in Computer Science*, 407–419. Springer.

Peled, D. A. 1993. All from one, one for all: on model checking using representatives. In Courcoubetis, C., ed., *Proceedings of CAV*, volume 697 of *Lecture Notes in Computer Science*, 409–423. Springer.

Penczek, W., and Lomuscio, A. 2003. Verifying epistemic properties of multi-agent systems via bounded model checking. In *Proceedings of AAMAS*, 209–216. New York, NY, USA: ACM Press.

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

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

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

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