Verification of Multi-Agent Properties in Electronic Voting: A Case Study

Damian Kurpiewski[1,3], Wojciech Jamroga[1,2], Łukasz Maśko[1]
Łukasz Mikulski[3,1], Witold Pazderski[1]
Wojciech Penczek[1], and Teofil Sidoruk[1,4] *

[1] *Institute of Computer Science, Polish Academy of Sciences*
ul. Jana Kazimierza 5, 01-248 Warsaw, Poland

[2] *Interdisciplinary Centre for Security, Reliability, and Trust, SnT, University of Luxembourg*
29 Av. John F. Kennedy, 1855 Luxembourg, Luxembourg

[3] *Faculty of Mathematics and Computer Science, Nicolaus Copernicus Univeristy*
ul. Chopina 12/18, 87-100 Toruń, Poland

[4] *Faculty of Mathematics and Information Science, Warsaw University of Technology*
ul. Koszykowa 75, 00-662 Warsaw, Poland

Abstract

Formal verification of multi-agent systems is hard, both theoretically and in practice. In particular, studies that use a single verification technique typically show limited efficiency, and allow to verify only toy examples. Here, we propose some new techniques and combine them with several recently developed ones to see what progress can be achieved for a real-life scenario. Namely, we use fixpoint approximation, domination-based strategy search, partial order reduction, and parallelization to verify heterogeneous scalable models of the Selene e-voting protocol. The experimental results show that the combination allows to verify requirements for much more sophisticated models than previously.

Keywords: multi-agent systems, formal verification, e-voting.

1 Introduction

Multi-agent systems (MAS) provide models and methodologies for analysis of systems that feature interaction of multiple autonomous components [64,69,71]. Formal specification and verification of such systems becomes essential due to

* The authors acknowledge the support of the National Centre for Research and Development, Poland (NCBR), and the Luxembourg National Research Fund (FNR), under the PolLux/FNR-CORE project STV (POLLUX-VII/1/2019). W. Penczek and T. Sidoruk acknowledge the support of CNRS/PAS under the project MOSART.

the dynamic development of AI solutions that enter practical applications [4,5]. In particular, it is crucial to assess requirements that refer to *strategic abilities* of agents and their groups, such as the ability of a passenger to leave an autonomous cab (preferably alive), or the inability of an intruder to take remote control of the cab.

Specification and verification of MAS. Properties of this kind can be conveniently specified in *modal logics of strategic ability*, of which alternating-time temporal logic **ATL*** [7,8] is probably the most popular. Logic-based methods for MAS are relatively well studied from the theoretical perspective [21,25,26,34], including theories of agents and agency [10,11,19,60,70] semantic issues [2,3,23,31,42,63], meta-logical properties [14,32], and the complexity of model checking [13,24,32,63,67]. There is even a number of model checking approaches and tools. Unfortunately, they only admit temporal properties [9,22,44,45,49], deal with the less practical case of perfect information strategies [6,17,46,47,48], treat imperfect information with limited interest and effectiveness [54], or have restricted verification capabilities [5,50,52]. No less importantly, attempts to verify actual requirements on realistic agent systems have been scarce.

In this paper, we combine and extend some of the recent advances in model checking of modal specifications for MAS [37,39,51], and apply them to see how far we can get with the verification of an existing e-voting protocol. Anonymous, coercion-resistant, and verifiable e-voting procedures have been proposed and studied for over 10 years now [16,27,43,62], including implementations and their use in real-life elections [1,15,20]. This makes e-voting a great case for testing verification algorithms and tools, being developed for MAS.

Contribution. Verification for modal logics of strategic ability is hard, both theoretically and in practice. Likely, no single technique suffices to deal with it alone. Here, we try the "all out" approach, and combine several techniques, developed recently by our team, to verify properties of the Selene e-voting protocol [61]. We use the algorithms of *fixpoint approximation* [37], *depth-first* and *domination-based strategy search* [51], as well as *partial order reduction* [40,41]. To apply the latter, we extend it to handle strategic-epistemic properties, and prove the correctness of the extension. We also propose and study a distributed variant of the depth-first and domination-based synthesis. We evaluate the power of the combined approach on a new model of Selene, consisting of voters, coercers, and the election infrastructure. While our model does not yet match the complexity of a real-life election, it goes beyond typically used examples.

Related work. Formal verification of voting protocols typically focuses on their cryptographic aspects. The multi-agent and social interaction in the models is limited, and the verification restricts to temporal and bisimulation-based properties. Examples include the automated analysis of Selene [12] and Electryo [72] using the Tamarin prover for security protocols. Theorem proving in first-order logic was also used to capture some socio-technical factors of Helios

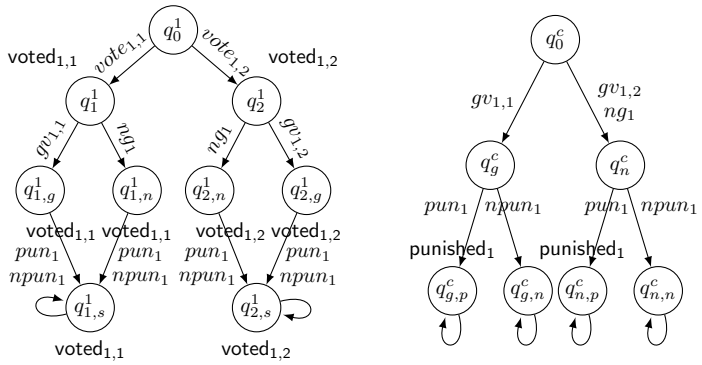

Fig. 1. ASV$_1^2$: agents *Voter$_1$* (left) and *Coercer* (right)

in [56]. Similarly, the interactive theorem prover Coq for higher-order logic was used in [29,33].

A slightly more detailed multi-agent model of the Prêt à Voter protocol was used in [35], but the verification concerned only simple temporal formulas of **CTL**. In [38], strategic abilities in Prêt à Voter were considered, but the strategies were hand-crafted rather than synthesized in the verification process. A preliminary formalization of receipt-freeness and coercion resistance using **ATL** was shown in [65], but no verification was proposed. Perhaps the closest work to the present one was our previous attempt to model and verify Selene in [36], but the model used there was extremely simple, and the performance results were underwhelming.

2 Preliminaries

We first recall the models of asynchronous interaction in MAS, defined in [40] and inspired by [53,59].

2.1 Models of Asynchronous Interaction

Definition 2.1 [Asynchronous MAS] An *asynchronous multi-agent system (AMAS)* S consists of n agents $\mathcal{A} = \{1, \ldots, n\}$, each associated with a tuple $A_i = (L_i, Evt_i, R_i, T_i, PV_i, V_i)$ including a set of *local states* $L_i = \{l_i^1, l_i^2, \ldots, l_i^{n_i}\}$, a nonempty finite set of *events* $Evt_i = \{\alpha_i^1, \alpha_i^2, \ldots, \alpha_i^{m_i}\}$, and a *repertoire of choices* $R_i : L_i \to 2^{2^{Evt_i}}$. For each $l_i \in L_i$, $R_i(l_i) = \{E_1, \ldots, E_m\}$ is a nonempty list of nonempty choices available to i at l_i. If the agent chooses $E_j = \{\alpha_1, \alpha_2, \ldots\}$, then only an event in E_j can be executed at l_i within the agent's module. Moreover, $T_i : L_i \times Evt_i \rightharpoonup L_i$ is a (partial) *local transition function* such that $T_i(l_i, \alpha)$ is defined iff $\alpha \in \bigcup R_i(l_i)$.

Agents are endowed with mutually disjoint, finite and possibly empty sets of *local propositions* PV_i, and their *valuations* $V_i : L_i \to 2^{PV_i}$. $Evt = \bigcup_{i \in \mathcal{A}} Evt_i$ is the set of all events, and $Agent(\alpha) = \{i \in \mathcal{A} \mid \alpha \in Evt_i\}$ the set of agents who have access to event α. $PV = \bigcup_{i \in \mathcal{A}} PV_i$ is the set of all propositions.

Note that each agent "owns" the events affecting its state, but some of the events may be shared with other agents. Such events can only be executed synchronously by all the involved parties. This way, the agent can influence how the states of the other agents evolve. Moreover, the agent's strategic choices are restricted by its repertoire function. Assigning sets rather than single events in R_i, which subsequently determines the type of strategy functions in Section 2.2, is a deliberate decision, allowing to avoid certain semantic issues that fall outside the scope of this paper. We refer the reader to [40] for the details.

The following example demonstrates a simple AMAS, while also introducing some key concepts that will be expanded upon in our model of the real-world protocol SELENE (discussed in Section 3). In particular, there is a *coercer* agent, whose goal is to ensure that the voter(s) select a particular candidate. To that end, the coercer may threaten them with punishment, e.g. if they refuse to cooperate by not sharing the ballot, or openly defy by voting for another candidate.

Example 2.2 [Asynchronous Simple Voting] Consider a simple voting system ASV_n^k with $n+1$ agents (n voters and 1 coercer). Each $Voter_i$ agent can cast her vote for a candidate $\{1,\ldots,k\}$, and decide whether to share her vote receipt with the *Coercer* agent. The coercer can choose to punish the voter or refrain from it. A graphical representation of the agents for $n=1, k=2$ is shown in Fig. 1. We assume that the coercer only registers if the voter hands in a receipt for candidate 1 or not. The repertoire of the coercer is defined as $R_c(q_0^c) = \{\{gv_{1,1}, gv_{1,2}, ng_1\}\}$ and $R_c(q_g^c) = R_c(q_n^c) = \{\{pun_1\}, \{npun_1\}\}$, i.e., the coercer first *receives* the voter's decision regarding the receipt, and then *controls* whether the voter is punished or not. Analogously, the voter's repertoire is given by: $R_1(q_0^1) = \{\{vote_{1,1}\}, \{vote_{1,2}\}\}$, $R_1(q_j^1) = \{\{gv_{1,j}\}, \{ng_1\}\}$ for $j=1,2$, and $R_1(q_{1,g}^1) = R_1(q_{1,n}^1) = R_1(q_{2,g}^1) = R_1(q_{2,n}^1) = \{\{pun_1, npun_1\}\}$.

Notice that the coercer cannot determine which of the events $gv_{1,1}, gv_{1,2}, ng_1$ will occur; this is entirely under the voter's control. This way we model the situation where it is the decision of the voter to show her vote or not. Similarly, the voter cannot avoid punishment by choosing the strategy allowing only $npun_1$, because the choice $\{npun_1\}$ is *not* in the voter's repertoire. She can only execute $\{pun_1, npun_1\}$, and await the decision of the coercer.

The execution semantics is based on interleaving with synchronization on shared events. Note that for a shared event to be executed, it must be done jointly by all agents who have it in their repertoires.

Definition 2.3 [Interleaved interpreted systems] Let S be an AMAS with n agents, and let $I \subseteq L_1 \times \ldots \times L_n$. The *full model* $IIS(S,I)$ extends S with: (i) the set of initial states I; (ii) the set of global states $St \subseteq L_1 \times \ldots \times L_n$ that collects all the configurations of local states, reachable from I by T (see below); (iii) the (partial) *global transition function* $T : St \times Evt \rightharpoonup St$, defined by $T(g_1, \alpha) = g_2$ iff $T_i(g_1^i, \alpha) = g_2^i$ for all $i \in Agent(\alpha)$ and $g_1^i = g_2^i$ for all

$i \in \mathcal{A} \setminus Agent(\alpha)$;[1] (iv) the *global valuation* of propositions $V : St \to 2^{PV}$, defined as $V(l_1, \ldots, l_n) = \bigcup_{i \in \mathcal{A}} V_i(l_i)$.

We will sometimes write $g_1 \xrightarrow{\alpha} g_2$ instead of $T(g_1, \alpha) = g_2$.

Definition 2.4 [Enabled events] Event $\alpha \in Evt$ is *enabled* at $g \in St$ if $g \xrightarrow{\alpha} g'$ for some $g' \in St$, i.e., $T(g, \alpha) = g'$. The set of such events is denoted by $enabled(g)$.

Let $A = \{a_1, \ldots, a_k\} \subseteq \mathcal{A} = \{1, \ldots, n\}$ and $\overrightarrow{E}_A = (E_{a_1}, \ldots, E_{a_k})$ for some $k \leq n$, such that $E_i \in R_i(g^i)$ for every $i \in A$. Event $\beta \in Evt$ is *enabled by the vector of choices* $\overrightarrow{E}_A$ *at* $g \in St$ iff, for every $i \in Agent(\beta) \cap A$, we have $\beta \in E_i$, and for every $i \in Agent(\beta) \setminus A$, it holds that $\beta \in \bigcup R_i(g^i)$. That is, the "owners" of β in A have selected choices that admit β, while all the other "owners" of β *might* select choices that do the same. We denote the set of such events by $enabled(g, \overrightarrow{E}_A)$. Clearly, $enabled(g, \overrightarrow{E}_A) \subseteq enabled(g)$.

Some combinations of choices enable no events. To account for this, the models of AMAS are augmented with "silent" ϵ-loops, added when no "real" event can occur.

Definition 2.5 [Undeadlocked IIS] Let S be an AMAS, and assume that no agent in S has ϵ in its alphabet of events. The *undeadlocked model of* S, denoted $IIS^{\epsilon}(S, I)$, extends the model $IIS(S, I)$ as follows:

- $Evt_{IIS^{\epsilon}(S,I)} = Evt_{IIS(S,I)} \cup \{\epsilon\}$, where $Agent(\epsilon) = \emptyset$;
- For each $g \in St$, we add the transition $g \xrightarrow{\epsilon} g$ iff there is a selection of some agents' choices $\overrightarrow{E}_A = (E_{a_1}, \ldots, E_{a_k})$, $E_i \in R_i(g^i)$, such that $enabled_{IIS(S,I)}(g, \overrightarrow{E}_A) = \emptyset$. Then, we also fix $enabled_{IIS^{\epsilon}(S,I)}(g, \overrightarrow{E}_A) = \{\epsilon\}$.

We use the term *model* to refer to subgraphs of $IIS(S, I)$ as well as $IIS^{\epsilon}(S, I)$.

Example 2.6 The undeadlocked model of ASV_1^2 is shown in Figure 2. Note that it contains no ϵ-transitions, since no choices of the voter and the coercer can cause a deadlock.

2.2 Reasoning About Strategies and Knowledge

Let PV be a set of propositions and $\mathcal{A}$ the set of all agents. The syntax of *alternating-time logic* **ATL*** [8,63] is given by:

$$\varphi ::= \mathsf{p} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle\langle A \rangle\rangle \gamma, \qquad \gamma ::= \varphi \mid \neg\gamma \mid \gamma \wedge \gamma \mid \mathrm{X}\,\gamma \mid \gamma\,\mathrm{U}\,\gamma,$$

where $\mathsf{p} \in PV$, $A \subseteq \mathcal{A}$, X stands for "next", U for "until", and $\langle\langle A \rangle\rangle \gamma$ for "agent coalition A has a strategy to enforce γ". Temporal operators F ("eventually") and G ("always"), Boolean connectives, and constants are defined as usual.

[1] g^i denotes agent i's state in $g = (l_1, \ldots, l_n)$, i.e., $g^i = l_i$.

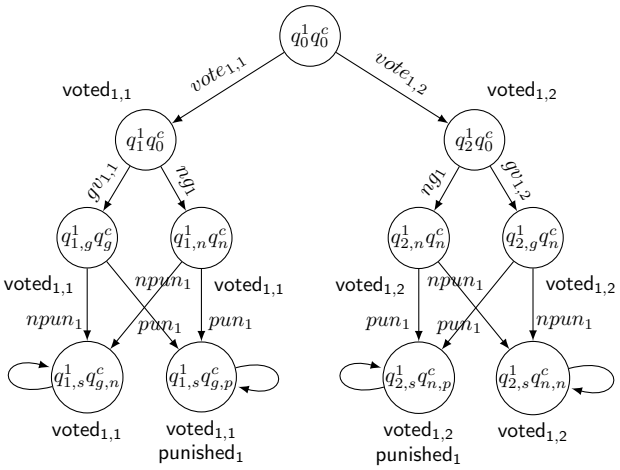

Fig. 2. The undeadlocked model $IIS^{\epsilon}(ASV_1^2)$

Example 2.7 *Coercion feasibility* against voter i can be expressed by formula $\langle\!\langle Coercer \rangle\!\rangle \mathrm{F}\,\mathsf{punished_i}$ (the coercer can ensure that the voter is eventually punished).

Strategic ability of agents. Following [40], a *positional imperfect information strategy (*ir*-strategy) for agent i* is defined by a function $\sigma_i \colon L_i \to 2^{Evt_i}$, such that $\sigma_i(l) \in R_i(l)$ for each $l \in L_i$. Note that σ_i is uniform by construction, as it is based on local, and not global states. The set of such strategies is denoted by Σ_i^{ir}. Joint strategies Σ_A^{ir} for $A = \{a_1, \ldots, a_k\} \subseteq \mathcal{A}$ are defined as usual, i.e., as tuples of strategies σ_i, one for each agent $i \in A$. By $\sigma_A(g) = (\sigma_{a_1}(g), \ldots, \sigma_{a_k}(g))$, we denote the joint choice of coalition A at global state g. An infinite sequence of global states and events $\pi = g_0\alpha_0 g_1\alpha_1 g_2 \ldots$ is called a *path* if $g_j \xrightarrow{\alpha_j} g_{j+1}$ for every $j \geq 0$. The set of all paths in model M starting at state g is denoted by $\Pi_M(g)$.

Definition 2.8 [Standard outcome] Let $A \subseteq \mathcal{A}$. The *standard outcome* of strategy $\sigma_A \in \Sigma_A^{\mathrm{ir}}$ in state g of model M is the set $out_M^{\mathrm{Std}}(g, \sigma_A) \subseteq \Pi_M(g)$ such that $\pi = g_0\alpha_0 g_1\alpha_1 \cdots \in out_M(g, \sigma_A)$ iff $g_0 = g$, and for each $m \geq 0$ we have that $\alpha_m \in enabled_M(g_m, \sigma_A(g_m))$.

Definition 2.9 [Reactive outcome] The *reactive outcome* is the set $out_M^{\mathrm{React}}(g, \sigma_A) \subseteq out_M^{\mathrm{Std}}(g, \sigma_A)$ such that $\pi = g_0\alpha_0 g_1\alpha_1 \cdots \in out_M^{\mathrm{React}}(g, \sigma_A)$ iff $\alpha_m = \epsilon$ implies $enabled_M(g_m, \sigma_A(g_m)) = \{\epsilon\}$.

Intuitively, the standard outcome collects all the paths where agents in A follow σ_A, while the others freely choose from their repertoires. The reactive outcome includes only those outcome paths where the opponents cannot miscoordinate on shared events. Let $x \in \{\mathrm{Std}, \mathrm{React}\}$. The ir-semantics of $\langle\!\langle A \rangle\!\rangle \gamma$ in asynchronous MAS [8,63,40] is defined by the clause:

$M, g \models^x \langle\langle A \rangle\rangle \gamma$ iff there is a strategy $\sigma_A \in \Sigma_A^{\mathrm{ir}}$ such that for all $\pi \in out_M^x(g, \sigma_A)$ we have $M, \pi \models^x \gamma$.

Adding knowledge operators. The following relations capture the notion of indistinguishability between states, needed to define semantics for the epistemic modality.

Definition 2.10 [Indistinguishable states] For each $i \in \mathcal{A}$, the relation $\sim_i = \{(g, g') \in St \times St \mid g^i = g'^i\}$ denotes that states g, g' are *indistinguishable* for agent i. The relation $\sim_J = \bigcap_{j \in J} \sim_j$ extends it to the distributed knowledge of a group of agents $J \subseteq \mathcal{A}$.

By $\mathbf{ATL}^*\mathbf{K}$, we denote the extension of $\mathbf{ATL}^*$ with knowledge operators K_i where $\mathrm{K}_i\psi$ means "agent i knows that ψ". Note that temporal and strategic operators cannot be nested inside ψ. The semantics of $\mathrm{K}_i\psi$ can be defined by the clause [26,68]:

$M, g \models \mathrm{K}_i\psi$ iff $M, g' \models \psi$ for every g' such that $g' \sim_i g$.

In e-voting, epistemic properties arise due to information exchanged by cryptographic protocols, but also published in plaintext, e.g., on the Web Bulletin Board.

Subjective ability. The asynchronous semantics of $\langle\langle A \rangle\rangle \gamma$ in [40] is based on the notion of "objective" ability, i.e., it suffices that there exists a strategy σ_A which enforces γ on outcome paths from the objective starting point g of the model. The more popular notion of "subjective" ability requires that σ_A succeeds on all outcome paths from the states that A might consider as possible starting points, cf. [14] for an in-depth discussion. The "subjective" semantics of strategic operators can be defined as:

$M, g \models_S^x \langle\langle A \rangle\rangle \gamma$ iff there is a strategy $\sigma_A \in \Sigma_A^{\mathrm{ir}}$ such that, for each $\pi \in \bigcup_{i \in A} \bigcup_{g' \sim_i g} out_M^x(g', \sigma_A)$, we have $M, \pi \models_S^x \gamma$.

Example 2.11 Let $M = IIS^{\epsilon}(ASV_n^k, \{g_0\})$ where $g_0 = (q_0^1, \ldots, q_0^n, q_0^c)$. That is, M is the undeadlocked model of the AMAS in Example 2.2 with g_0 as its sole initial state. Note that the Std and React semantics coincide on M, as it includes no ϵ-transitions. Clearly, $M, g_0 \models \langle\langle \mathit{Coercer} \rangle\rangle \mathrm{F}\ \mathsf{punished_i}$ for both the objective and subjective semantics. On the other hand, the stronger requirement $\langle\langle \mathit{Coercer} \rangle\rangle \mathrm{F}\ K_i \mathsf{punished_i}$ does not hold in (M, g_0).

In this paper, we focus on the (more popular) subjective semantics. Moreover, we only use formulas with no next step operators X and no nested strategic modalities, which is essential for the application of partial-order reduction. The corresponding "simple" subset of $\mathbf{ATL}^*$ (resp. $\mathbf{ATL}^*\mathbf{K}$) is denoted by $\mathbf{sATL}^*$ (resp. $\mathbf{sATL}^*\mathbf{K}$). The restriction is less prohibitive than it seems at a glance. First, the X operator is of little value for asynchronous systems. Secondly, nested strategic modalities would only allow us to express an agent's ability to endow another agent with ability (or deprive the other agent of ability). Such properties are sometimes interesting, e.g., one may want to require that $\langle\langle \mathit{Voter}_1 \rangle\rangle \mathrm{G}\ \neg\langle\langle \mathit{Coercer} \rangle\rangle \mathrm{F}\ \mathsf{punished_1}$ (the voter can keep the coercer unable

to punish the voter). Still, simpler properties like $\langle\!\langle \mathit{Voter}_1 \rangle\!\rangle \mathrm{G}\, \neg \mathsf{punished}_1$ and $\langle\!\langle \mathit{Voter}_1 \rangle\!\rangle \mathrm{G} \bigwedge_{j=1,\ldots,k} \neg K_{\mathit{Coercer}} \mathsf{voted}_{1,\mathrm{j}}$ are usually of more immediate interest.

Finally, we remark that, for subjective ability, epistemic operators are definable with strategic operators, since $K_i\varphi \equiv \langle\!\langle i \rangle\!\rangle \bot \,\mathrm{U}\, \varphi$. Moreover, $M, g \models \langle\!\langle i \rangle\!\rangle \gamma$ always implies $M, g \models K_i \langle\!\langle i \rangle\!\rangle \gamma$ in the subjective semantics.

3 How to Specify Voters and Coercers

To keep the paper self-contained, in this section we provide a short description of the Selene voting protocol and its formal specification.

3.1 Short Description of Selene

Selene [61] is an electronic voting protocol aimed to provide an effective mechanism for *voter verifiability* and *coercion resistance*. On the one hand, the voter receives a piece of evidence that allows to check if her vote has been registered correctly. On the other hand, she can present a fake vote evidence to the coercer, thus convincing him that she voted according to the coercer's request.

The protocol proceeds as follows. Before the election, the Election Authority (EA) sets up the system, generating the election keys used for the encryption and decryption of the votes and preparing the vote trackers, one per voter. The trackers are then encrypted and mixed to break any link between the voter and her tracker, and published on the Web Bulletin Board (WBB).

In the voting phase, each voter fills in, encrypts, and signs her vote, and sends it to the system. After several intermediate steps, a pair (Vote_v, tr_v), consisting of the decrypted ballot and the tracker of v, is published on the WBB for each $v \in \mathit{Voters}$. At this stage, no voters know their trackers. All the votes are presented in plaintext on the WBB. Thus, the tally of the election is open to a public audit.

At the final stage, the voters receive their trackers by an independent channel (e.g., sms). If the voter has not been coerced, then she requests the special term α_v, which allows for obtaining the correct tracker tr_v. If she was coerced to fill her ballot in a certain way, she sends a description of the requested vote to the election server. After such a request, a fake term α'_v is sent to the voter, which can be presented to the coercer. The α'_v token, together with the public commitment of the voter, reveals a tracker pointing out to a vote compatible with the coercer's demand.

Selene uses the ElGamal key encryption scheme, and relies on multiplicative homomorphism and non-interactive zero-knowledge proofs of knowledge that accompany all the transformations of data presented in WBB. In this work we abstract away from the cryptography and focus on the interaction between the involved agents.

3.2 Asynchronous MAS for Selene

In our model of Selene, we concentrate on the non-cryptographic interaction between the agents. In this sense, the present model is similar to [36], with three important differences.

First, the new modeling is fully modular and scalable. To this end, we use AMAS as input (rather than concurrent game structures or concurrent interpreted systems). Secondly, we have defined a flexible specification language for AMAS, based on asynchronous agent templates similar to those of UPPAAL [9], and implemented an interpreter for it. Besides modularity and scalability, this allowed us to adapt and use partial order reduction for our models. Thirdly, our specifications include much more details of the SELENE procedure than the skeletal model in [36].

3.3 Agents

There are 4 templates for agents in the modeling: the Election Authority (EA) that handles the generation and distribution of trackers and provides the Web Bulletin Board for voters and coercers, the Coercer, the standard Voter, and the Coerced Voter interacting with a coercer. The templates are written in a simple specification language created to provide the input for our algorithms. Each template consists of the name of the agent, the number of instances of that template in the model (e.g., the number of voters), the initial state, the list of transitions, and the agent's repertoire of choices (called "PROTOCOL" here). The Coerced Voter and Coercer templates for an election with 2 candidates are shown in Figures 3 and 4. The full specification is available at github.com/blackbat13/stv/blob/master/models/Selene.txt.

A template always begins with the keyword *Agent*, the agent's name, and the number of instances. The next line specifies the initial state, followed by the list of local transitions. Each transition starts with an event name, optionally preceded by the keyword *shared* if the event is shared with another agent. Then, the source and the target states are given. Optionally, the transition specification can also include a precondition (in the form of a simple Boolean formula) and/or a postcondition (via a list of updates specifying propositions and their new values). These conditions are only a technical shortcut that allows us to write clearer and shorter specifications. The keyword *aID* represents the ID of the current agent and is automatically replaced when preparing local models of agents. For example, the template VoterC[2] would produce two agents, $Voter_{C1}$ and $Voter_{C2}$. Then, each instance of the keyword *aID* will be replaced with $Voter_{C1}$ for the first coerced voter, and with $Voter_{C2}$ for the second one.

Consider the template in Figure 3. The first step of the coerced voter is her interaction with the coercer who can request the vote for a particular candidate (events *coerce1_VoterC1* and *coerce2_VoterC1*). The next step is the event *start_voting* synchronized with the *EA*. When the election has begun, the voter can create her commitment, fill in the vote, encrypt it, and send it to the EA. After that, she waits for the publication of votes, which is controlled by the EA. When the votes are published on the WBB, the voter can decide to compute the false alpha term and the false tracker using publicly available data, or she can just wait for her real tracker. The last few steps for the VoterC agent consist of checking the WBB, verifying the vote, and interacting with the

```
Agent VoterC[1]:
init start
shared coerce1_aID: start -> coerced [aID_required=1]
shared coerce2_aID: start -> coerced [aID_required=2]
select_vote1: coerced -> prepared [aID_vote=1, aID_prep_vote=1]
select_vote2: coerced -> prepared [aID_vote=2, aID_prep_vote=2]
shared is_ready: prepared -> ready
shared start_voting: ready -> voting
shared aID_vote: voting -> vote [Coercer1_aID_vote=?aID_vote, Coercer1_aID_revote=?
    aID_revote]
shared send_vote_aID: vote -> send
revote_vote_1: send -[aID_revote==1]> voting [aID_vote=?aID_required, aID_revote=2]
skip_revote_1: send -[aID_revote==1]> votingf
revote_vote_2: send -[aID_revote==2]> voting [aID_vote=?aID_required, aID_revote=3]
skip_revote_2: send -[aID_revote==2]> votingf
final_vote: send -[aID_revote==3]> votingf [aID_vote=?aID_prep_vote]
skip_final: send -[aID_revote==3]> votingf
shared send_fvote_aID: votingf -> sendf
shared finish_voting: sendf -> finish
shared send_tracker_aID: finish -> tracker
shared finish_sending_trackers: tracker -> trackers_sent
shared give1_aID: trackers_sent -> interact [Coercer1_aID_tracker=1]
shared give2_aID: trackers_sent -> interact [Coercer1_aID_tracker=2]
shared not_give_aID: trackers_sent -> interact [Coercer1_aID_tracker=0]
shared punish_aID: interact -> ckeck [aID_punish=true]
shared not_punish_aID: interact -> check [aID_punish=false]
shared check_tracker1_aID: check -> end
shared check_tracker2_aID: check -> end
PROTOCOL: [[coerce1_aID, coerce2_aID], [punish, not_punish]]
```

Fig. 3. Voter template

```
Agent Coercer[1]:
init coerce
shared coerce1_VoterC1: coerce -> coerce [aID_VoterC1_required=1]
shared coerce2_VoterC1: coerce -> coerce [aID_VoterC1_required=2]
shared start_voting: coerce -> voting
shared VoterC1_vote: voting -> voting
shared finish_voting: voting -> finish
shared finish_sending_trackers: finish -> trackers_sent
shared give1_VoterC1: trackers_sent -> trackers_sent
shared give2_VoterC1: trackers_sent -> trackers_sent
shared not_give_VoterC1: trackers_sent -> trackers_sent
to_check: trackers_sent -> check
shared check_tracker1_Coercer1: check -> check
shared check_tracker2_Coercer1: check -> check
to_interact: check -> interact
shared punish_VoterC1: interact -> interact
shared not_punish_VoterC1: interact -> interact
finish: interact -> end [aID_finish=1]
PROTOCOL: [[give1_VoterC1, give2_VoterC1, not_give_VoterC1]]
```

Fig. 4. Coercer template

coercer again. The voter can show one of her trackers (the false one or the real one) to the coercer, who then either punishes her or does not.

Additionally, the voter can do *revoting*, i.e., cast her vote multiple times, which is a well-known technique to counter in-house coercion by family members.

3.4 Specification of Properties

To specify interesting properties of the voting system, we use simple formulas of **ATL*K**. In the experiments, we will concentrate on the property of *coercion-vulnerability*, using the following formula:

$$\varphi_{vuln,i,k} \equiv \langle\langle Coercer \rangle\rangle \mathrm{G}\left((\mathsf{end} \wedge \mathsf{revote}_{\mathsf{v1}} = \mathsf{k} \wedge \mathsf{voted}_{\mathsf{v1}} = \mathsf{i}) \rightarrow \mathrm{K}_{Coercer} \mathsf{voted}_{\mathsf{v1}} = \mathsf{i}\right)$$

Formula $\varphi_{vuln,i}$ says that the coercer has a strategy so that, at the end of the election, if the voter has effectively voted for candidate i, the coercer knows

about it. This can be seen as the opposite of *receipt-freeness* and *coercion-resistance* formalizations in [43,65]. Note that the formula is parameterized by the name i of the preferred candidate of the coercer, as well as the number k of revoting rounds that the coercer is able to observe and learn the value of the cast vote. Proposition $\mathsf{revote}_{\mathsf{v1}}$ corresponds to the number of revoting rounds performed by the first voter.

We will use the above models and formulas in our verification experiments in Section 6.

3.5 Model definition

Model file consists of definitions of agents templates and various properties. Empty lines and Lines starting with % are comments lines and are ignored by the parser. Properties that can be defined in the model file are described below. Each proposition starts with uppercase property name followed by a colon.

PERSISTENT List of propositions names that should be considered as persistent when generating the model. Persistent proposition retains its value after it was set. Non-persistent propositions are present only in the state that is the result of the transition that created this proposition.

REDUCTION List of propositions names used for the partial-order reduction method.

FORMULA ATL formula to be verified.

SHOW_EPISTEMIC Boolean value used only in the graphical interface. If set to *true* links of the epistemic relations will be displayed in the model view.

3.6 Agent definition

Agent definition consists of a set of non-empty lines specifying the local model of the agent. Each agent definition is a template that can be used to generate multiple agents of a given type. Individual definitions are explained below.

Header The first line should specify the name of the agent template and number of agents of this type, given in the following format: `Agent AgentName[count]:`. For example, in case of `Agent Voter[3]:` three agents will be generated: *Voter1*, *Voter2* and *Voter3*.

init First line of the agent specification, name of the initial state.

Local transition Transition is defined in format:
`actionName: state1 -[preCondition]> state2 [propositions].`
Propositions are given in a form of a comma-separated list of variable definitions, i.e. `[prop1=true, prop2=false, prop3=2]`. Precondition can be any boolean formula that can be evaluated in Python.

Shared transition Shared transitions are defined similarly to local transitions, but are prefixed with *shared* keyword.

Dynamic names In order to simplify the specification dynamic names can be used. Every occurence of the keyword *aID* in the agent template specification will be replaced with the name of the computed agent.

4 Verification Algorithms

Model checking strategic ability under imperfect information is hard, both theoretically and in practice. In this section, we present two recently developed techniques that we use to tackle the high complexity of verification: fixpoint approximation and dominance-based depth-first strategy search (Sections 4.1 and 4.3). We also propose a novel approach based on distributed strategy synthesis for **sATL*****K** (Section 4.4).

4.1 Verification by Fixpoint Approximation

The main idea of the fixpoint approximations method presented in [37] is that sometimes, instead of the exact model-checking, it suffices to provide a lower and an upper bound for the output. In particular, given a formula ϕ, we construct two translations $tr_L(\phi)$ and $tr_U(\phi)$, such that $tr_L(\phi) \Rightarrow \phi \Rightarrow tr_U(\phi)$. In other words, if the lower bound translation $tr_L(\phi)$ evaluates to true, then the original formula ϕ must hold in the model. Similarly, if the upper bound translation $tr_U(\phi)$ is verified as false, then ϕ is also false.

The approximations are built on fixpoint-definable properties. For the upper bound translation we just compute the given formula under the perfect information assumption. For the lower bound we rely on translations that map the formula of $\mathbf{ATL}^*_{\mathbf{ir}}$ to an appropriate variant of alternating μ-calculus, see [37] for more details.

4.2 Depth-First Strategy Search

Further, we have implemented a depth-first search algorithm for strategy synthesis. The typical recursive DFS-based approach needed to be adapted due to the presence of epistemic classes and the nondeterministic outcome of the coalition's actions. In case of nondeterminism due to multiple possible transitions, it is possible that decisions taken in one of the branches may determine some choices in the other branches. If such a decision leads to a locally winning strategy, it may need to be changed since decisions in nodes being members of the same epistemic class can influence the outcome of transitions in other branches. The proposed algorithm allows to backtrack and change locally winning strategy when no winning strategy is found in another branch.

Clearly, even for relatively small models with hundreds of states, the space of strategies is too big to perform a full search. To address this, we used our DFS algorithm as the backbone for implementing more refined methods, namely **Domino DFS** and two variants of parallel model checking, see Sections 4.3 and 4.4.

4.3 Domination-Based Strategy Search

The strategy space grows exponentially with respect to the number of states and transitions in the model (on top of the state space explosion). However, in reality it often suffices to only check a subset of possible strategies. This idea was used in the **Domino DFS** algorithm [51]. In that method, a notion of strategy dominance was proposed, according to which the dominated strategies are omitted during the search. Although in the worst case scenario all the

strategies in the model need to be checked, the experimental results in [51] showed that in some cases the method achieves significant improvement in performance.

4.4 Parallel Implementation

The verification process itself has exponential nature with complexity of $O(k^n)$ for a single agent (where k is the number of possible actions and n is the number of states in the model). To reduce the search time, parallel computation can be used. As a basis and a reference, we used a sequential version of the recursive DFS-based strategy search. The biggest advantage of the recursive approach is its ability to perform multiple, independent searches at a time. Such a solution may be very effective for distributed computing architectures, when the search space can be divided independently and distributed between computing nodes with separate memory spaces. The downside is that all the nodes obtain copies of the same model, but then they all may compute independently looking for different solutions.

We have defined and implemented two different approaches to parallelizing depth-first strategy search.

Distributed Strategy Search: Simple Branching. The simple approach tries to concurrently execute a number of instances of sequential search, but assuming different strategy prefixes (i.e., such sequences of actions starting at a starting state, which correspond to deterministic paths in a graph – except for the last one, where there may be a branch). To achieve this, the algorithm first executes a breadth-first search to determine a set of potential different strategy prefixes up to a number equal to a value given as parameter. Due to the way in which the prefixes are selected, they correspond to single paths and they can be identified with single states. The algorithm tries to expand the prefixes by performing a BFS graph traversal.

Similarly to most BFS-based algorithms, a queue of pending prefixes is used. Initially, this queue contains an empty prefix denoting only one process to be executed. In each step of the algorithm, the first pending prefix is selected. The algorithm tries to expand this prefix by adding actions available in a state to which this prefix leads. For every deterministic action in such state, a new prefix is generated by appending this action to the current prefix. So obtained new prefix is added to the pending queue for further expansion. If the action is nondeterministic, the prefix obtained by appending this action to the current prefix is added to the set of resulting prefixes and is not expanded any more. The current prefix is then discarded. The loop stops when either there are no more pending prefixes or the number of prefixes in both the result set and the pending queue exceeds the given parameter. In the latter case, all the pending prefixes are copied to the resulting set.

After a set of prefixes is determined, the main process spawns a number of children processes. Each subprocess tries to find a winning strategy using the sequential algorithm, assuming that the strategy starts with actions from its assigned prefix. The main process waits for the results from its children; if any

of them reports that a winning solution has been found, all the other children are terminated and the search ends.

Distributed Strategy Search: Flexible Version. Our second approach uses concurrent execution to examine parallel branches of a single strategy in a flexible way. To this end, it directly expands the single–threaded DFS method. In this approach, parallel threads cooperate in building the same subgraph of the model that corresponds to the outcome of the candidate strategy. The initial master thread spawns new worker threads whenever it finds a state in which selection of the same action leads to multiple states. A sequential algorithm in such case would try to recursively build a substrategy considering all the possible targets one by one. In the concurrent version, an additional worker thread is created for each possible outcome of the same action.

In order to ease the synchronization between threads, only the main thread is allowed to spawn new threads due to parallel branches. It is also assumed that only one parallel thread is allowed to select actions for states from any epistemic class. If any worker thread reaches a state in such a class, it must wait until the master thread fixes the action for this class. Synchronization is also needed when one thread meets a node which is already examined by another thread – since the nodes cooperate, it is not necessary to redo the same work, therefore the second node is suspended until the node is checked by the first one. The main thread is allowed to intercept the work already performed by a worker thread.

5 Taming State Space Explosion

The main obstacle in model checking of MAS logics is the prohibitive complexity of models, in particular due to the state-space explosion. Partial order reduction (POR) is a well-known technique for state space reduction, dating back more than three decades [30,57,66]. The idea is to restrict the set of all enabled transitions to a representative (i.e. provably sufficient) subset, based on some underlying notion of equivalence. Crucially, this occurs while generating the unfolding of the system, so the full model, which may be far too large, is never created.

POR has been defined for variants of **LTL** and **CTL**, including temporal-epistemic logics [53]. Furthermore, the reduction for **LTL** was recently adapted (notably, at the same computational cost) to the much more expressive logic **sATL*** [39,41], allowing to leverage the existing algorithms and tools for the verification of strategic abilities. In [40], POR for **sATL*** was shown to still work under an improved execution semantics for AMAS, which we also adopt in this paper. Note, however, that our formalization of coercion-vulnerability in Section 3.4 is a *strategic-epistemic property*, and hence those results do not cover this case. Moreover, the reductions in [39,40,41] are *not* correct if we allow for nested modalities. The question is: can we adapt the scheme so that it works if we only allow to nest *epistemic* operators?

In this section, we prove that the answer is affirmative. We also show that the reduction is correct for the subjective semantics of ability, whereas the

previous works used the less intuitive (and less popular) objective semantics.

5.1 Conceptual Machinery

We first recall the concept of stuttering equivalence. Intuitively, two paths are stuttering equivalent if they can be divided into corresponding finite segments, each satisfying exactly the same propositions. If all states in corresponding segments are also indistinguishable for agents $i \in J$, then we say the paths are $J-$ stuttering equivalent.

Definition 5.1 [(J-)stuttering equivalence] Paths $\pi, \pi' \in \Pi_M(g)$ are *stuttering equivalent*, denoted $\pi \equiv_s \pi'$, if there exists a partition $B_0 = (\pi[0], \ldots, \pi[i_1 - 1]),\ B_1 = (\pi[i_1], \ldots, \pi[i_2 - 1]),\ \ldots$ of the states of π, and an analogous partition $B'_0, B'_1, \ldots$ of the states of π', such that for each $j \geq 0 : B_j$ and B'_j are nonempty and finite, and $V(g) \cap PV = V(g') \cap PV$ for every $g \in B_j$ and $g' \in B'_j$.

If $\pi \equiv_s \pi'$, and additionally it holds that $\forall j > 0 \quad \forall g \in B_j, g' \in B'_j : g \sim_J g'$, then paths π and π' are *J-stuttering equivalent*, denoted $\pi \equiv_s^J \pi'$.

States g and g' are *stuttering path equivalent* (resp. *J-stuttering path equivalent*), denoted $g \equiv_s g'$ (resp. $g \equiv_s^J g'$), iff for every path π starting from g, there is a path π' starting from g' such that $\pi' \equiv_s \pi$ (resp. $\pi' \equiv_s^J \pi$), and for every path π' starting from g', there is a path π starting from g such that $\pi \equiv_s \pi'$ (resp. $\pi \equiv_s^J \pi'$).

Models M and $M' \subseteq M$ are *stuttering path equivalent* (resp. *J-stuttering path equivalent*), denoted $M \equiv_s M'$ (resp. $M \equiv_s^J M'$), iff they have the same initial states, and for each initial state $\iota_i \in I$ and each path $\pi \in \Pi_M(\iota_i)$, there is a path $\pi' \in \Pi_{M'}(\iota_i)$ such that $\pi \equiv_s \pi'$ (resp. $\pi \equiv_s^J \pi'$).

The POR algorithm uses the notions of invisible and independent events. Intuitively, an event is invisible iff it does not change the valuations of the propositions.[2] Two events are independent iff at least one is invisible and they are not in the same agent's repertoire. We designate a subset of agents $A \subseteq \mathcal{A}$ whose events are visible by definition.

Definition 5.2 [Invisibility and independence of events] Let $M = IIS^{\epsilon}(S, I)$, and $A \subseteq \mathcal{A}$. An event $\alpha \in Evt$ is *invisible* wrt. A and PV if $Agent(\alpha) \cap A = \emptyset$ and for each two global states $g, g' \in St$ we have that $g \xrightarrow{\alpha} g'$ implies $V(g) \cap PV = V(g') \cap PV$. The set of all invisible events for A, PV is denoted by $Invis_{A,PV}$, and its closure, i.e. the set of visible events, by $Vis_{A,PV} = Evt \setminus Invis_{A,PV}$.

The notion of *independence* $Ind_{A,PV} \subseteq Evt \times Evt$ is defined as: $Ind_{A,PV} = \{(\alpha, \alpha') \in Evt \times Evt \mid Agent(\alpha) \cap Agent(\alpha') = \emptyset\} \setminus (Vis_{A,PV} \times Vis_{A,PV})$. Events $\alpha, \alpha' \in Evt$ are called *dependent* if $(\alpha, \alpha') \notin Ind_{A,PV}$. If it is clear from the context, we omit the subscript PV.[3] Note that ϵ events are always

[2] This technical concept of invisibility is not connected to any agent's view, unlike in [55].

[3] The sets of agents' local propositions PV_i are explicitly disjoint in our model (cf. Definition 2.1), allowing for simpler checking of event independence in the actual implementation of

invisible and independent from others, since they do not modify propositions and we have that $Agent(\epsilon) = \emptyset$ (by Definition 2.5).

The reduced model (or *submodel*) $M' \subseteq M$ obtained with POR extends the same AMAS S as $M = IIS^{\epsilon}(S, I)$. In particular, we have $St' \subseteq St$, $I = I'$, T is an extension of T', and $V' = V|_{St'}$. Note that, for each $g \in St'$, it holds that $\Pi_{M'}(g) \subseteq \Pi_M(g)$.

M' is generated by modifying the standard DFS [28], so that for each g, the successor state g_1 such that $g \xrightarrow{\alpha} g_1$ is selected from $E(g) \cup \{\epsilon\}$ such that $E(g) \subseteq enabled(g) \setminus \{\epsilon\}$. That is, the algorithm always selects ϵ, plus a subset of the enabled events at g. This modified DFS is called for each initial state of the model, and we have $\Pi_{M'} = \bigcup_{g \in I} \Pi_{M'}(g)$. The conditions on the heuristic selection of $E(g)$ given below are inspired by [18,41,58].

C1 Along each path π in M that starts at g, each event that is dependent on an event in $E(g)$ cannot be executed in π without an event in $E(g)$ being executed first in π. Formally, $\forall \pi \in \Pi_M(g)$ such that $\pi = g_0\alpha_0 g_1\alpha_1 \ldots$ with $g_0 = g$, and $\forall b \in Evt$ such that $(b, c) \notin Ind_A$ for some $c \in E(g)$, if $\alpha_i = b$ for some $i \geq 0$, then $\alpha_j \in E(g)$ for some $j < i$.

C2 If $E(g) \neq enabled(g) \setminus \{\epsilon\}$, then $E(g) \subseteq Invis_A$.

C3 For every cycle in M' containing no ϵ-transitions, there is at least one node g in the cycle for which $E(g) = enabled(g) \setminus \{\epsilon\}$, i.e., all the successors of g are expanded.

Submodel $M' \subseteq M$ generated with this algorithm satisfies property $\mathbf{AE}_A$ [40]: $\forall\, \sigma_A \in \Sigma_A^{\mathrm{ir}} \quad \forall\, \iota_i \in I \quad \forall\, \pi \in out^x_M(\iota_i, \sigma_A) \quad \exists\, \pi' \in out^x_{M'}(\iota_i, \sigma_A) : \quad \pi \equiv_s \pi'$, where $x \in \{\mathrm{Std}, \mathrm{React}\}$.

5.2 POR for sATL*K

We will show that the reduction algorithm for **sATL*** [41,40] can be applied also to formulas that include the knowledge operator (in subformulas of the form $\mathrm{K}_i\varphi$), provided that $J \subseteq A$. That is, any agents in these epistemic subformulas are added to the set $A \subseteq \mathcal{A}$ that parametrises the relations of invisibility and independence.

Theorem 5.3 *Let S be an AMAS, $J \subseteq A \subseteq \mathcal{A}$, $M = IIS^{\epsilon}(S, I)$, and let $M' \subseteq M$ be the reduced model generated by DFS with the choice of $E(g')$ for $g' \in St'$ given by conditions* **C1-C3**. *Then, for any starting state $\iota_i \in I$ and any* **sATL*K** *formula φ over PV that refers only to coalitions $\hat{A} \subseteq A$, we have that $M, \iota_i \models \varphi$ iff $M', \iota_i \models \varphi$.*

Proof. First, note that conditions **C1-C3** remain unchanged from the reduction algorithm for **sATL*** [40]. Thus, by [**?**, Theorems A.8 and A.9], [4] we have that:

POR.

[4] The conference paper [40] does not include a technical appendix with proofs of Theorems A.8 and A.9, so we refer to its extended arXiv manuscript here.

(*) M and M' are stuttering path equivalent. For each path $\pi = g_0\alpha_0 g_1\alpha_1\ldots$ with $g_0 = \iota_i$ in M, there is a stuttering equivalent path $\pi' = g'_0\alpha'_0 g'_1\alpha'_1\ldots$ with $g'_0 = \iota_i$ in M' such that $Evt(\pi)|_{Vis_A} = Evt(\pi')|_{Vis_A}$, i.e., π and π' have the same maximal sequence of visible events for A.

(**) M and M' satisfy structural condition $\mathbf{AE}_A$.

That is, we have $M, \iota_i \models \varphi$ iff $M', \iota_i \models \varphi$ for all non-epistemic φ. To extend the reasoning to any **sATL*****K** formula, we first show that the full and reduced model are also J-stuttering equivalent, which then allows to prove that epistemic subformulas are preserved in the reduced model M'. Finally, we show that these subformulas can be replaced with equivalent new propositions, effectively reducing the problem to the previously proven case for **sATL***.

Because $J \subseteq A$ (and so all transitions of the agents in group J are visible), it follows directly from **C2** that if $E(g) \neq enabled(g)\setminus\{\epsilon\}$, then $Agent(\alpha)\cap J = \emptyset$ for any event $\alpha \in E(g)$. This is a direct analogue of the extra condition **CJ** from [53]. Together with (*), this implies that the full and reduced model are also J-stuttering equivalent:

(***) $M \equiv_s^J M'$.

Consider any subformula $\varphi = \mathrm{K}_i\psi$. As per the syntax of **sATL*****K**, temporal operators and strategic modalities cannot be nested inside K_i, so φ is a purely epistemic formula that only contains knowledge operator(s) and propositional variables with Boolean connectives. Now, we will show it follows from (***) that epistemic subformulas are preserved in the reduced model, i.e., for any state g such that $g \equiv_s^J g'$, we have $M, g \models \varphi$ iff $M', g' \models \varphi$:

($\Rightarrow$) Assume that $M, g \models \mathrm{K}_i\psi$. Let $St_\psi = \{g_\psi \in St \mid g \sim_i g_\psi\}$, and take g'_ψ such that $g' \sim_i g'_\psi$. We need to show that $M', g'_\psi \models \psi$. From $g \equiv_s^J g'$ and by transitivity of relation $\sim_i$, we have that $g'_\psi \in St_\psi$. So, clearly $M, g'_\psi \models \psi$. As $g_\psi \equiv_s^J g'_\psi$, it follows from the inductive assumption that $M', g'_\psi \models \psi$. Hence, $M', g' \models \mathrm{K}_i\psi$.

($\Leftarrow$) Assume that $M', g' \models \mathrm{K}_i\psi$. Let $St'_\psi = \{g'_\psi \in St' \mid g' \sim_i g'_\psi\}$, and take g_ψ such that $g \sim_i g_\psi$. We need to show that $M, g_\psi \models \psi$. Consider a path $\pi \in M$ that contains g_ψ. From (***), there is a path $\pi' \in M'$, which contains a state $g''_\psi \in St'$, such that $g_\psi \equiv_s^J g''_\psi$. By transitivity of $\sim_i$, we get that $g''_\psi \in St'_\psi$, and thus $M', g''_\psi \models \psi$. As $g_\psi \equiv_s^J g''_\psi$, it follows from the inductive assumption that $M, g_\psi \models \psi$. Hence, $M, g \models \mathrm{K}_i\psi$.

From the above we get that any epistemic subformula $K_i\psi$ holds in the reduced model M' iff it holds in the corresponding state of the (J-stuttering equivalent) full model M. Now, we introduce auxiliary propositional variables to replace epistemic subformulas, including nested ones.

Consider subformulas $\varphi_0, \varphi_1, \ldots$, where $\varphi_0 = \varphi$, and for all $i > 0$, φ_i is an epistemic subformula nested in φ_{i-1}. Note that in the reduced model M', one can add a set of new propositional variables $PV' = \bigcup_i \{\mathsf{sat}_{\varphi_i}\}$ to PV, and extend the valuation function accordingly, so that we have $V' : St' \rightarrow$

$2^{PV \cup PV'}$, and sat_{φ_i} is true in state $g' \in St'$ iff $M', g' \models \varphi_i$. That is, for each epistemic (sub)formula φ_i, a new proposition sat_{φ_i} is added, whose valuation in each state $g' \in St'$ corresponds to the satisfaction of φ_i in that state of M'. Then, for the formula $\varphi' = \mathsf{sat}_{\varphi_0}$, it clearly holds that $M', g' \models \varphi'$ iff $M', g' \models \varphi$. Furthermore, since epistemic subformulas only refer to agents $i \in J$ and we have that $J \subseteq A$, it follows that $Vis_{A,PV} = Vis_{A,PV \cup PV'}$ and $Ind_{A,PV} = Ind_{A,PV \cup PV'}$. That is, replacing epistemic subformulas with new propositions in this manner does not affect the visibility or independence of events wrt. A and PV. Hence, we also have $M', g' \models \varphi'$ iff $M, g \models \varphi$, from (*) and (**) and by [**?**, Theorems A.8 and A.9], as φ' is a **sATL*** formula. But from the construction of φ', we have $M', g' \models \varphi'$ iff $M', g' \models \varphi$, so it also holds that $M, g \models \varphi$ iff $M', g' \models \varphi$ for any **sATL*****K** formula φ, both for standard outcomes [**?**, Theorem A.8] and under the opponent reactivity condition [**?**, Theorem A.9]. Thus, in particular we have that $M, \iota_i \models \varphi$ iff $M', \iota_i \models \varphi$, QED. □

5.3 POR for Subjective Semantics of Ability

Recall the subjective semantics of strategic ability in Section 2.2. We will show that the reduction scheme remains applicable in this setting, i.e., when the set of initial states is comprised of previously designated subset $I \subseteq St$, plus all states indistinguishable from those in I for coalition agents.

Theorem 5.4 *Let* $x \in \{\mathrm{Std}, \mathrm{React}\}$, $\hat{A} \subseteq A$ *and* $I_{\hat{A}} = I \cup \{g \in St \mid \exists_{\iota_i \in I} \exists_{j \in \hat{A}} : g \sim_j \iota_i\}$. *Let* $M = IIS(S, I_{\hat{A}})$, *and let* M' *be the reduction of* M *generated by POR using conditions* ***C1-C3*** *for the choice of ample sets. Then, for any initial state* $\iota_i \in I_{\hat{A}}$ *and any* **sATL*****K** *formula* φ *that refers only to coalition* $\hat{A}$, *we have that* $M, \iota_i \models^x_S \varphi$ *iff* $M', \iota_i \models^x_S \varphi$.

Proof. [Proof sketch] For each $\iota_i \in I_{\hat{A}}$, take M_i constructed by DFS starting from ι_i (i.e., with a single initial state), and let M'_i be the reduction of M_i generated by POR.

Take any path $\pi \in \Pi_M$. Clearly, $\pi \in \Pi_{M_i}$ for some $i > 0$. By Theorem 5.3, we have $M_i \equiv^J_s M'_i$, so there is a J-stuttering equivalent path $\pi' \in \Pi_{M'_i}$. From the construction by DFS, $\Pi_{M'} = \bigcup_{i \in I_{\hat{A}}} \Pi_{M'_i}$. Hence, $\pi' \in \Pi_{M'}$, which implies that $M \equiv^J_s M'$. (*)

Take any joint strategy $\sigma_{\hat{A}}$. The subjective outcome of $\sigma_{\hat{A}}$ in M (resp. M') is the sum of objective outcomes of $\sigma_{\hat{A}}$ in M_i (resp. M'_i). But from $\mathbf{AE}_A$, $out^x_{M_i}(\iota_i, \sigma_{\hat{A}}) \equiv_s out^x_{M'_i}(\iota_i, \sigma_{\hat{A}})$. So, analogously to the reasoning for (*), it follows that $\bigcup_{i \in I_{\hat{A}}} out^x_{M_i}(\iota_i, \sigma_{\hat{A}}) \equiv_s \bigcup_{i \in I_{\hat{A}}} out^x_{M'_i}(\iota_i, \sigma_{\hat{A}})$. Hence, M and M' also satisfy $\mathbf{AE}_A$. (**)

Since (*) and (**), the thesis follows from Theorem 5.3, as in the case for objective semantics of strategic ability. □

6 Experiments and Results

In this section, we give a brief summary of the experimental results.

Models and formulas. The scalable class of models has been described

#A	#R	Full Model					Reduced Model					Result
		#st	#tr	Seq.	Par.	Appr.	#st	#tr	Seq.	Par.	Appr.	
4	3	3.63e4	7.46e4	0.003	0.009	1.121	2.60e4	5.99e4	0.001	0.002	0.184	True
4	5	5.62e4	1.15e5	0.004	0.003	0.345	4.01e4	9.26e4	0.002	0.002	0.283	True
4	10	1.06e5	2.18e5	0.009	0.005	0.691	7.55e4	1.74e5	0.004	0.002	0.563	True
5	3	1.55e6	5.91e6	0.158	0.004	14.78	1.09e6	4.65e6	0.112	0.021	12.99	True
6	3	7.61e7	4.98e8	0.524	0.051	41.24	5.34e7	3.82e8	0.427	0.042	37.35	True
7	3	model generation timeout					model generation timeout					-

Table 1
Verification of $\varphi_{vuln,i,k}$ for the first candidate ($i = 1$) and $k = \#R$ revotes

#Ag	#R	Full Model					Reduced Model					Result
		#st	#tr	Seq.	Par.	Appr.	#st	#tr	Seq.	Par.	Appr.	
4	3	3.63e4	7.46e4	0.003	0.010	1.103	2.60e4	5.99e4	0.002	0.003	0.166	True
4	5	5.62e4	1.15e5	0.004	0.005	0.348	4.01e4	9.26e4	0.003	0.003	0.280	True
4	10	1.06e5	2.18e5	0.008	0.009	0.700	7.55e4	1.74e5	0.005	0.004	0.567	True
5	3	1.55e6	5.91e6	0.160	0.055	14.03	1.09e6	4.65e6	0.112	0.053	12.49	True
6	3	7.61e7	4.98e8	0.602	0.083	42.44	5.34e7	3.82e8	0.501	0.057	38.20	True
7	3	model generation timeout					model generation timeout					-

Table 2
Verification of $\varphi_{vuln,i,k}$ for the last candidate ($i = \#C$) and $k = \#R$ revotes

#Ag	#R	Full Model					Reduced Model					Result
		#st	#tr	Seq.	Par.	Appr.	#st	#tr	Seq.	Par.	Appr.	
4	3	3.63e4	7.46e4	0.303	0.317	1.128	2.60e4	5.99e4	0.202	0.205	0.179	False
4	5	5.62e4	1.15e5	0.524	0.592	0.325	4.01e4	9.26e4	0.411	0.503	0.280	False
4	10	1.06e5	2.18e5	0.721	0.668	0.459	7.55e4	1.74e5	0.525	0.512	0.364	False
5	3	1.55e6	5.91e6	2.146	1.257	0.981	1.09e6	4.65e6	1.513	1.003	0.583	False
6	3	7.61e7	4.98e8	5.232	3.228	1.892	5.34e7	3.82e8	4.986	2.427	1.092	False
7	3	model generation timeout					model generation timeout					-

Table 3
Verification of $\varphi_{vuln,i,k}$ for the first candidate ($i = 1$) and $k = \#R - 1$ revotes

#Ag	#R	Full Model					Reduced Model					Result
		#st	#tr	Seq.	Par.	Appr.	#st	#tr	Seq.	Par.	Appr.	
4	3	3.63e4	7.46e4	0.302	0.311	0.180	2.60e4	5.99e4	0.201	0.213	0.126	False
4	5	5.62e4	1.15e5	0.519	0.584	0.310	4.01e4	9.26e4	0.410	0.475	0.283	False
4	10	1.06e5	2.18e5	0.742	0.627	0.462	7.55e4	1.74e5	0.558	0.544	0.370	False
5	3	1.55e6	5.91e6	2.160	1.358	0.942	1.09e6	4.65e6	1.621	1.009	0.519	False
6	3	7.61e7	4.98e8	5.504	3.516	1.903	5.34e7	3.82e8	5.110	2.380	1.112	False
7	3	model generation timeout					model generation timeout					-

Table 4
Verification of $\varphi_{vuln,i,k}$ for the last candidate ($i = \#C$) and $k = \#R - 1$ revotes

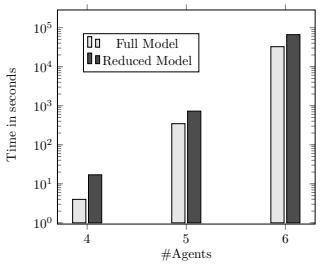

Fig. 5. Model generation times for 3 candidates 3 revotes

in detail in Section 3. They can be configured using four parameters: the number of voters (V), the number of coerced voters (CV), the number of candidates (C) and the number of revoting turns (R). In all experiments we

use configurations with one coerced voter and three candidates. We also use the *coercion-vulnerability formula*

$$\varphi_{vuln,i,k} \equiv \langle\langle Coercer \rangle\rangle \mathrm{G}\,((\mathsf{end} \wedge \mathsf{revote}_{\mathsf{v1}} = \mathsf{k} \wedge \mathsf{voted}_{\mathsf{v1}} = \mathsf{i}) \rightarrow \mathrm{K}_{Coercer}\mathsf{voted}_{\mathsf{v1}} = \mathsf{i})$$

saying that the coercer has a strategy so that, at the end of the election, if the voter has effectively voted for candidate i, then the coercer knows about it. This can be seen as the opposite of *receipt-freeness* and *coercion-resistance* formalizations in [43,65], see Section 3.4 for more details.

Configuration of the experiments. The test platform was a server equipped with ninety-six 2.40 GHz Intel Xeon Platinum 8260 CPUs, 991 GB RAM, and 64-bit Linux. All times are given in seconds. The algorithms have been implemented in C++. The code is available on github: github.com/blackbat13/ATLFormulaCheckerC.

Results. We present the verification results in the Tables 1-4 and model generation times in the Figure 5. All times are given in seconds and the timeout for verification was set to 1 hour and 20 hours for model generation. Model generation times are presented using a logarithmic scale. We present the size of the model and the verification times both for the full and the reduced model. We have compared time results for four verification methods, as described in Section 4: the sequential strategy synthesis (Seq.), the parallelized version (Par.) and the fixpoint approximation (Appr.).

Results: Domino DFS and flexible distributed algorithm. We omit the results for the **Domino DFS** algorithm, as it performed much slower than other algorithms. The reason is that in this scenario there is no redundancy in the strategy space, and hence no room for elimination of dominated strategies. In consequence, no gain is achieved compared to the standard DFS strategy synthesis.

The same applies to the flexible distributed algorithm proposed in Section 4.4. Apparently, cloning the model and synchronization between the threads grossly outweighs the benefits of using parallel computation.

Discussion of the results. As the results show, the simple parallel verification performs quite well in most cases. Only for relatively small models the sequential algorithm achieves better performance than the parallel algorithm, which is the result of a overhead associated with the parallelization. As various experiments have shown, the performance of the parallel algorithm is heavily dependent on the structure of the model. Depending on the amount of branching and loops in the model, it can result in a timeout even for small models. The reason for this behavior seems simple: the space of all strategies is too big to manage. As the strategy is generated from top to bottom, less branching means less configurations to check.

The fixpoint approximation algorithm performs well in cases where no strategy can be found, as it quickly reaches the fixpoint. On the other hand, when

the formula is satisfied, multiple loops maybe required before reaching the fixpoint, with each loop adding more states to the computation set.

Challenges and lessons learned. When conducting the experiments, we have encountered two main difficulties: high memory usage and slow model generation. The first one results in a memout for ordinary computers. The second one results in a timeout for more powerful machines with hundreds of gigabytes of RAM. Before moving to better computers, we have tried to overcome the memory usage problem by implementing a communication with an external database engine. The idea was to store parts of the model during the generation in the database. That, in theory, allows to shift some part of the memory requirements to other parts of the system. Unfortunately, this also heavily impacted generation times, resulting in approximately 10 times slower computation, which lead us to abandoning this idea.

To overcome the high model generation times one can also try to parallelize the generation of models. Our experiments have shown that the parallel algorithm, if implemented in a clever way, can greatly reduce the computation times. The model generation procedure works similarly to the DFS algorithm, which suggests that its parallel version can be as effective.

7 Conclusions

Modal logics for MAS, and the related verification problems, have been studied for many years. Unfortunately, they are characterized by high computational complexity, both in theory and in practice. On the other hand, it is necessary to try and verify real-life scenarios, with all their complexity, to make substantial progress in the field.

In this paper, we propose a hands-on case study in verification of a genuine protocol for secure voting. We use the "all out" approach, implementing multiple existing techniques (depth-first strategy search, domination-based strategy synthesis, fixpoint approximation) as well as proposing novel adaptations of others (partial-order reduction, distributed strategy search). Of those, partial order reduction, simple DFS, simple distributed DFS, and fixpoint approximation show very promising performance. Moreover, they produce significant gains in some of the tested configurations. On the other hand, the domination-based synthesis and flexible distributed turn out rather ill-fitted to the model checking task at hand. Overall, the experimental results are promising, and suggest that the time is ripe for the community to engage more in realistic application of the algorithms that we develop.

We also emphasize that the extension of partial order reduction, presented here, is a nontrivial technical result in its own right, as it is the first POR algorithm that handles the nesting of epistemic modalities in the context of strategic ability.

References

[1] Adida, B., *Helios: web-based open-audit voting*, in: *Proceedings of the 17th conference on Security symposium*, SS'08 (2008), pp. 335–348.

[2] Agotnes, T., *A note on syntactic characterization of incomplete information in ATEL*, in: *Procedings of Workshop on Knowledge and Games*, 2004, pp. 34–42.

[3] Agotnes, T., V. Goranko, W. Jamroga and M. Wooldridge, *Knowledge and ability*, in: H. van Ditmarsch, J. Halpern, W. van der Hoek and B. Kooi, editors, *Handbook of Epistemic Logic*, College Publications, 2015 pp. 543–589.

[4] Akintunde, M. E., E. Botoeva, P. Kouvaros and A. Lomuscio, *Formal verification of neural agents in non-deterministic environments*, in: *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems AAMAS*, 2020, pp. 25–33.

[5] Akintunde, M. E., E. Botoeva, P. Kouvaros and A. Lomuscio, *Verifying strategic abilities of neural-symbolic multi-agent systems*, in: *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning KR*, 2020, pp. 22–32.

[6] Alur, R., L. de Alfaro, T. A. Henzinger, S. Krishnan, F. Mang, S. Qadeer, S. Rajamani and S. Tasiran, *MOCHA: Modularity in model checking*, Technical report, University of Berkeley (2000).

[7] Alur, R., T. A. Henzinger and O. Kupferman, *Alternating-time Temporal Logic*, in: *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS)* (1997), pp. 100–109.

[8] Alur, R., T. A. Henzinger and O. Kupferman, *Alternating-time Temporal Logic*, Journal of the ACM **49** (2002), pp. 672–713.

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

[10] Broersen, J., M. Dastani, Z. Huang and L. van der Torre, *The BOID architecture: conflicts between beliefs, obligations, intentions and desires*, in: J. Müller, E. Andre, S. Sen and C. Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents* (2001), pp. 9–16.

[11] Broersen, J., A. Herzig and N. Troquard, *Embedding Alternating-time Temporal Logic in Strategic STIT logic of agency*, Journal of Logic and Computation **16** (2006), pp. 559–578.

[12] Bruni, A., E. Drewsen and C. Schürmann, *Towards a mechanized proof of Selene receipt-freeness and vote-privacy*, in: *Proceedings of E-Vote-ID*, Lecture Notes in Computer Science **10615** (2017), pp. 110–126.

[13] Bulling, N., J. Dix and W. Jamroga, *Model checking logics of strategic ability: Complexity*, in: M. Dastani, K. Hindriks and J.-J. Meyer, editors, *Specification and Verification of Multi-Agent Systems*, Springer, 2010 pp. 125–159.

[14] Bulling, N. and W. Jamroga, *Comparing variants of strategic ability: How uncertainty and memory influence general properties of games*, Journal of Autonomous Agents and Multi-Agent Systems **28** (2014), pp. 474–518.

[15] Chaum, D., R. Carback, J. Clark, A. Essex, S. Popoveniuc, R. Rivest, P. Ryan, E. Shen, A. Sherman and P. Vora, *Scantegrity II: end-to-end verifiability by voters of optical scan elections through confirmation codes*, Trans. Info. For. Sec. **4** (2009), pp. 611–627.

[16] Chaum, D., P. Y. A. Ryan and S. A. Schneider, *A practical voter-verifiable election scheme*, in: *Proceedings of ESORICS*, 2005, pp. 118–139.

[17] Chen, T., V. Forejt, M. Kwiatkowska, D. Parker and A. Simaitis, *PRISM-games: A model checker for stochastic multi-player games*, in: *Proceedings of Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, Lecture Notes in Computer Science **7795** (2013), pp. 185–191.

[18] Clarke, E. M., O. Grumberg and D. A. Peled, "Model Checking," The MIT Press, Cambridge, Massachusetts, 1999.

[19] Cohen, P. and H. Levesque, *Intention is choice with commitment*, Artificial Intelligence **42** (1990), pp. 213–261.

[20] Culnane, C., P. Ryan, S. Schneider and V. Teague, *vvote: A verifiable voting system*, ACM Trans. Inf. Syst. Secur. **18** (2015), pp. 3:1–3:30.
[21] Dastani, M., K. Hindriks and J. Meyer, editors, "Specification and Verification of Multi-Agent Systems," Springer, 2010.
[22] Dembiński, P., A. Janowska, P. Janowski, W. Penczek, A. Półrola, M. Szreter, B. Woźna and A. Zbrzezny, *VerICS: A tool for verifying timed automata and Estelle specifications*, in: *Proceedings of the of the 9th Int. Conf. on Tools and Algorithms for Construction and Analysis of Systems (TACAS'03)*, Lecture Notes in Computer Science **2619**, Springer, 2003 pp. 278–283.
[23] Dima, C., C. Enea and D. Guelev, *Model-checking an alternating-time temporal logic with knowledge, imperfect information, perfect recall and communicating coalitions*, in: *Proceedings of Games, Automata, Logics and Formal Verification (GandALF)*, 2010, pp. 103–117.
[24] Dima, C. and F. Tiplea, *Model-checking ATL under imperfect information and perfect recall semantics is undecidable*, CoRR **abs/1102.4225** (2011).
[25] Emerson, E. A., *Temporal and modal logic*, in: J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, Elsevier and MIT Press, 1990 pp. 995–1072.
[26] Fagin, R., J. Y. Halpern, Y. Moses and M. Y. Vardi, "Reasoning about Knowledge," MIT Press, 1995.
[27] Gardner, R. W., S. Garera and A. D. Rubin, *Coercion resistant end-to-end voting*, in: R. Dingledine and P. Golle, editors, *Financial Cryptography and Data Security*, Lecture Notes in Computer Science **5628**, Springer Berlin Heidelberg, 2009 pp. 344–361.
[28] Gerth, R., R. Kuiper, D. Peled and W. Penczek, *A partial order approach to branching time logic model checking*, Information and Computation **150** (1999), pp. 132–152.
[29] Ghale, M., R. Goré, D. Pattinson and M. Tiwari, *Modular formalisation and verification of STV algorithms*, in: *Proceedings of E-Vote-ID*, Lecture Notes in Computer Science **11143** (2018), pp. 51–66.
[30] Godefroid, P., "Partial-Order Methods for the Verification of Concurrent Systems: An Approach to the State-Explosion Problem," Springer-Verlag, Berlin, Heidelberg, 1996.
[31] Guelev, D. and C. Dima, *Epistemic ATL with perfect recall, past and strategy contexts*, in: *Proceedings of Computational Logic in Multi-Agent Systems (CLIMA)*, Lecture Notes in Computer Science **7486** (2012), pp. 77–93.
[32] Guelev, D., C. Dima and C. Enea, *An alternating-time temporal logic with knowledge, perfect recall and past: axiomatisation and model-checking*, Journal of Applied Non-Classical Logics **21** (2011), pp. 93–131.
[33] Haines, T., R. Goré and M. Tiwari, *Verified verifiers for verifying elections*, in: *Proceedings of CCS* (2019), pp. 685–702.
[34] Jamroga, W., "Logical Methods for Specification and Verification of Multi-Agent Systems," ICS PAS Publishing House, 2015.
[35] Jamroga, W., Y. Kim, D. Kurpiewski and P. Y. A. Ryan, *Towards model checking of voting protocols in uppaal*, in: *Proceedings of E-Vote-ID*, Lecture Notes in Computer Science **12455** (2020), pp. 129–146.
[36] Jamroga, W., M. Knapik and D. Kurpiewski, *Model checking the SELENE e-voting protocol in multi-agent logics*, in: *Proceedings of the 3rd International Joint Conference on Electronic Voting (E-VOTE-ID)*, Lecture Notes in Computer Science **11143** (2018), pp. 100–116.
[37] Jamroga, W., M. Knapik, D. Kurpiewski and Ł. Mikulski, *Approximate verification of strategic abilities under imperfect information*, Artificial Intelligence **277** (2019).
[38] Jamroga, W., D. Kurpiewski and V. Malvone, *Natural strategic abilities in voting protocols*, CoRR **abs/2007.12424** (2020).
[39] Jamroga, W., W. Penczek, P. Dembiński and A. Mazurkiewicz, *Towards partial order reductions for strategic ability*, in: *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (2018), pp. 156–165.
[40] Jamroga, W., W. Penczek and T. Sidoruk, *Strategic abilities of asynchronous agents: Semantic side effects and how to tame them*, in: *Proceedings of KR 2021*, 2021, pp. 368–378.

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

[42] Jamroga, W. and W. van der Hoek, *Agents that know how to play*, Fundamenta Informaticae **63** (2004), pp. 185–219.

[43] Juels, A., D. Catalano and M. Jakobsson, *Coercion-resistant electronic elections*, in: *Proceedings of the 2005 ACM workshop on Privacy in the electronic society*, ACM, 2005, pp. 61–70.

[44] Kacprzak, M., A. Lomuscio and W. Penczek, *From bounded to unbounded model checking for temporal epistemic logic*, Fundamenta Informaticae **63** (2004), pp. 221–240.

[45] Kacprzak, M., W. Nabialek, A. Niewiadomski, W. Penczek, A. Pólrola, M. Szreter, B. Wozna and A. Zbrzezny, *VerICS 2007 - a model checker for knowledge and real-time*, Fundamenta Informaticae **85** (2008), pp. 313–328.

[46] Kacprzak, M. and W. Penczek, *Unbounded model checking for alternating-time temporal logic*, in: *3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004), 19-23 August 2004, New York, NY, USA* (2004), pp. 646–653.

[47] Kacprzak, M. and W. Penczek, *Fully symbolic unbounded model checking for alternating-time temporal logic*, Autonomous Agents and Multi-Agent Systems **11** (2005), pp. 69–89.

[48] Kanski, M., A. Niewiadomski, M. Kacprzak, W. Penczek and W. Nabialek, *SMT-based unbounded model checking for ATL*, in: A. Nouri, W. Wu, K. Barkaoui and Z. Li, editors, *Verification and Evaluation of Computer and Communication Systems - 15th International Conference, VECoS 2021, Virtual Event, November 22-23, 2021, Revised Selected Papers*, Lecture Notes in Computer Science **13187** (2021), pp. 43–58.

[49] Kant, G., A. Laarman, J. Meijer, J. van de Pol, S. Blom and T. van Dijk, *LTSmin: High-performance language-independent model checking*, in: *Tools and Algorithms for the Construction and Analysis of Systems. Proceedings of TACAS*, Lecture Notes in Computer Science **9035** (2015), pp. 692–707.

[50] Kurpiewski, D., W. Jamroga and M. Knapik, *STV: Model checking for strategies under imperfect information*, in: *Proceedings of the 18th International Conference on Autonomous Agents and Multiagent Systems AAMAS 2019* (2019), pp. 2372–2374.

[51] Kurpiewski, D., M. Knapik and W. Jamroga, *On domination and control in strategic ability*, in: *Proceedings of the 18th International Conference on Autonomous Agents and Multiagent Systems AAMAS 2019* (2019), pp. 197–205.

[52] Kurpiewski, D., W. Pazderski, W. Jamroga and Y. Kim, *STV+Reductions: Towards practical verification of strategic ability using model reductions*, in: *Proceedings of AAMAS* (2021), pp. 1770–1772.

[53] Lomuscio, A., W. Penczek and H. Qu, *Partial order reductions for model checking temporal-epistemic logics over interleaved multi-agent systems*, Fundamenta Informaticae **101** (2010), pp. 71–90.

[54] Lomuscio, A., 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** (2017), pp. 9–30.

[55] Malvone, V., A. Murano and L. Sorrentino, *Hiding actions in multi-player games*, in: *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12, 2017*, 2017, pp. 1205–1213.

[56] Martimiano, T., E. D. Santos, M. Olembo and J. Martina, *Ceremony analysis meets verifiable voting: Individual verifiability in Helios*, in: *SECURWARE*, 2015, pp. 195–204.

[57] Peled, D., *All from one, one for all: on model checking using representatives*, in: C. Courcoubetis, editor, *Computer Aided Verification* (1993), pp. 409–423.

[58] Peled, D., *Combining partial order reductions with on-the-fly model-checking*, in: *Proceedings of the 6th International Conference on Computer Aided Verification*, LNCS 818 (1994), pp. 377–390.

[59] Priese, L., *Automata and concurrency*, Theoretical Computer Science **25** (1983), pp. 221–265.

[60] Rao, A. and M. Georgeff, *Modeling rational agents within a BDI-architecture*, in: *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, 1991, pp. 473–484.

[61] Ryan, P., P. Rønne and V. Iovino, *Selene: Voting with transparent verifiability and coercion-mitigation*, in: *Financial Cryptography and Data Security: Proceedings of FC 2016. Revised Selected Papers*, Lecture Notes in Computer Science **9604** (2016), pp. 176–192.

[62] Ryan, P. Y. A., S. A. Schneider and V. Teague, *End-to-end verifiability in voting systems, from theory to practice*, IEEE Security & Privacy **13** (2015), pp. 59–62.

[63] Schobbens, P. Y., *Alternating-time logic with imperfect recall*, Electronic Notes in Theoretical Computer Science **85** (2004), pp. 82–93.

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

[65] Tabatabaei, M., W. Jamroga and P. Y. A. Ryan, *Expressing receipt-freeness and coercion-resistance in logics of strategic ability: Preliminary attempt*, in: *Proceedings of the 1st International Workshop on AI for Privacy and Security, PrAISe@ECAI 2016* (2016), pp. 1:1–1:8.

[66] Valmari, A., *Stubborn sets for reduced state space generation*, in: G. Rozenberg, editor, *Advances in Petri Nets 1990* (1991), pp. 491–515.

[67] van der Hoek, W., A. Lomuscio and M. Wooldridge, *On the complexity of practical ATL model checking*, in: *Proceedings of International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (2006), pp. 201–208.

[68] van der Hoek, W. and M. Wooldridge, *Tractable multiagent planning for epistemic goals*, in: C. Castelfranchi and W. Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-02)* (2002), pp. 1167–1174.

[69] Weiss, G., editor, "Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence," MIT Press: Cambridge, Mass, 1999.

[70] Wooldridge, M., "Reasoning about Rational Agents," MIT Press : Cambridge, Mass, 2000.

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

[72] Zollinger, M., P. Roenne and P. Ryan, *Mechanized proofs of verifiability and privacy in a paper-based e-voting scheme*, in: *Proceedings of 5th Workshop on Advances in Secure Electronic Voting*, 2020.