Verification of the Socio-Technical Aspects of Voting: The Case of the Polish Postal Vote 2020

Yan Kim[1][0000−0001−7523−8783], Wojciech Jamroga[1,2][0000−0001−6340−8845], and Peter Y.A. Ryan[1][0000−0002−1677−9034]

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

Abstract. Voting procedures are designed and implemented by people, for people, and with significant human involvement. Thus, one should take into account the human factors in order to comprehensively analyze properties of an election and detect threats. In particular, it is essential to assess how actions and strategies of the involved agents (voters, municipal office employees, mail clerks) can influence the outcome of other agents' actions as well as the overall outcome of the election. In this paper, we present our first attempt to capture those aspects in a formal multi-agent model of the Polish presidential election 2020. The election marked the first time when postal vote was universally available in Poland. Unfortunately, the voting scheme was prepared under time pressure and political pressure, and without the involvement of experts. This might have opened up possibilities for various kinds of ballot fraud, in-house coercion, etc. We propose a preliminary scalable model of the procedure in the form of a Multi-Agent Graph, and formalize selected integrity and security properties by formulas of agent logics. Then, we transform the models and formulas so that they can be input to the state-of-art model checker Uppaal. The first series of experiments demonstrates that verification scales rather badly due to the state-space explosion. However, we show that a recently developed technique of user-friendly model reduction by variable abstraction allows us to verify more complex scenarios.

1 Introduction

In the last 30 years, the world has become densely connected. This results in a considerable space of potential threats, risks, and conflicting interests, that call for systematic (and preferably machine-assisted) analysis. What is more, IT services are implemented by people, with people, and for people. The intensive human involvement makes them hard to analyse beyond the usual computational complexity obstacles.

Voting procedures. Voting and elections are prime examples of services that are difficult to specify, hard to verify, and extremely important to the society [27]. If democracy is to be effective, it is essential to assess and mitigate the threats of fraud, manipulation, and coercion [44,53]. However, formal analysis of voting procedures must consider both the technological side of elections (i.e., protocols, architectures, and implementations) and the human and social context in which it is embedded [5]. The impact of the social factor has become especially evident during the US presidential elections of 2016 and

2020. In 2016, individual voters were targeted before the election by a combination of technology and social engineering to induce emotional reactions that would change their decisions, and possibly swing the outcome of the vote (the Cambridge Analytica scandal). In 2020, a large group of voters was targeted after the election by unfounded claims that severely undermined the public trust in the outcome. In both cases, it is impossible to understand the nature of what happened, and devise mitigation strategies, without the focus on human incentives and capabilities.

Specification and verification of multi-agent systems. *Multi-agent systems (MAS)* provide models and methodologies for the analysis of systems that feature interaction of multiple autonomous components, be it humans, robots, and/or software agents. The theoretical foundations of MAS are based on mathematical logic and game theory [50]. In particular, logic-based methods can be useful to formally specify and verify the outcomes of multi-agent interaction [22].

Formal analysis with multi-agent logics is typically based on model checking [4]. The system is formalized through a network of graphs (or automata) that define its components, their available actions, and the information flow between them. The properties are usually given as *temporal properties*, expressing that a given temporal pattern must (or may) occur, or *strategic properties* capturing the *strategic abilities* of agents and their groups. Especially the latter kind of properties are relevant for MAS; e.g., one may try to capture voter-verifiability as the ability of the voter to verify her vote, and coercion-resistance as the inability of the coercer to influence the behavior of the voter [53]. There are many available model checking tools, though none of them is perfect. Some admit only temporal properties [8], some focus on the less practical case of perfect information strategies [38], and the others have limited verification capabilities [37]. Moreover, it is often unclear how to formalize an actual real-life scenario, including the "right" model of the system [32] and the formal "transcription" of its desirable properties [34].

Socio-technical aspects of voting. In this paper, we use agent-based methodology to propose and analyze a simple multi-agent model of an actual election, that combines the technological backbone of the voting infrastructure with a model of possible human behaviors. The work is preliminary, in the sense that we do not explore the real breadth of participants' activities that might occur during the vote. Moreover, we mostly look at requirements that can be expressed as trace properties. This is because the computational complexity of the formal analysis turned out prohibitive even for such simple models and properties. We managed to mitigate the complexity by an innovative abstraction technique, but seeing if it scales well enough for realistic models of human interaction remains a subject for future work.

Case study: Polish postal vote of 2020. To focus on a concrete scenario, we consider the Polish presidential election of 2020. That was the first time when postal voting was universally available in Poland. Unfortunately, the voting scheme was prepared under pressure, and without the involvement of experts. This might have opened up possibilities for various kinds of ballot fraud, in-house coercion, etc. We propose a preliminary scalable model of the procedure in the form of a Multi-Agent Graph [8,31], and formalize selected integrity and security properties by formulas of agent logics. Then, we transform the models and formulas so that they can be input to the state-of-art model checker Uppaal [8], chosen because of its flexible model specification language and user-friendly GUI. As expected, the verification of unoptimized models scales rather badly due to the state-space explosion. To improve the performance, we employ a recently developed technique of user-friendly abstraction [31], with more promising results.

Related research. Formal verification of voting protocols has been the subject of research for over a decade. Prominent approaches include theorem proving in first-order, linear or higher order logic [46,14,21,20,25,26], and model checking of temporal, strategic and temporal-epistemic logics [29,32,33]. Most if not all results show that the task is very hard due to the prohibitive computational complexity of the underlying problems. For example, [20,21] conducted a formal analysis of voting protocols using ProVerif, and reported that they had to come up with workarounds for the model in order to the limitations of the tool.

Modelling and analysis of socio-technical systems is even more difficult because of the vast space of possible human behaviors, and problematic nature of the assumptions usually made about how users choose their actions. The theory of socio-technical systems dates back to the work of Trist and Bamforth in 1940s. In security, perhaps the best studied methodology is based on ceremonies [17], in particular the Concertina ceremony [9,10,40]. Some research has been also based on choreographies [15]. Moreover, game-theoretic models and analysis have been used in [16,30,5]. Here, follow up on the strand based on modeling and verification in multi-agent logics [29,32,33], while trying to put more emphasis on the social part of the system outside of the voting infrastructure.

When analyzing systems that involve human agents, it is important to take into account that they behave differently from the machines, and can make *errors*, or more generally, deviate from the prescribed protocol. This can happen due to a variety of reasons: misunderstanding, inattentiveness, malicious intention, or strategic self-interested action. Possible deviations from protocol in user behavior have been studied in [7,6], and we follow up on those ideas. To this end, we use the *skilled-human approach* to capture a variety of users' behaviors in our model of postal voting. That is, we extend the protocol specification of an "honest" behaviour through a hierarchy of *deviation sets*, i.e., sets of actions that deviate from the protocol and expand the repertoire of the participants. As pointed out in [6], there is a trade-off between the breadth of the deviation model and the computational feasibility of the formal analysis. In fact, our experiments in section 4 show that even for the skilled human approach only, the explicit state model checking becomes hard enough, and dedicated techniques must be used to mitigate the complexity.

Related verification tools. We use the Uppaal model checker [8] in our case study, mainly because of its GUI and a flexible system specification language. Other verification tools that we considered when preparing the study are:

- *MCMAS* [39]: a state-of-art OBDD-based symbolic model checker for agent-based systems. The system is described using ISPL (Interpreted Systems Programming Language), and the requirements are specified as formulae of strategic or temporal-epistemic properties;
- *Tamarin-prover* [43]: a tool for security protocol verification and not a model checker per se. The system specification language is based on multiset rewriting theories, and the requirements are specified as first-order temporal properties;
- *STV* [36,37]: an experimental toolbox for explicit-state model checking of strategic properties; at the moment, custom input models are not fully supported;

– *ProVerif* [13]: an automated cryptographic protocol verifier, in the symbolic (Dolev-Yao) model. The protocol representation is based on Horn clauses; it can be used for proving secrecy, authentication and equivalence properties.

Among the above tools, only STV, Uppaal, and (to lesser extent) Tamarin provide a graphical view of the system structure. Of those, only Uppaal allows for *interactive graphical system specification*, which we claim to be crucial in modelling and analysis of voting protocols. Real-life voting procedures include the interaction of numerous participants, each of them with a possibly different agenda and capabilities. Furthermore, the behaviour of most participants is characterized with a mixture of controllable and uncontrollable nondeterminism. In consequence, interactive GUI is crucial if we want to ensure that the model we verify and the one we want to verify are the same thing, cf. [32] for discussion.

Moreover, Uppaal (and, to a smaller extent, STV) allow for parameterized specification of the system, without forcing the designer to program a dedicated model-generator (e.g., as in the verification of SELENE protocol with MCMAS in [29]).

2 Postal Voting Procedure

Postal voting is one of the oldest forms for voting. In its simplest version, it is easy to setup for the authorities and easy to follow for the voters. On the other hand, it can be susceptible to ballot fraud, lacks verifiability, and opens up potential for vote buying and coercion. What is more, only basic mechanisms of recovery are possible (e.g., cancelling the whole elections in case of irregularities). This sometimes leads to controversial ad hoc decisions when dealing with the irregularities [28].

The postal voting procedure employed for the Polish Presidential Election in 2020 is no exception. There was an overall impression that the procedure had been prepared in haste [2,51] and with no proper research on the existing postal voting schemes that were proposed and used during the last two decades, such as [12,35]. For example, voter authentication is based on the assumption that a voter's national identification number (PESEL) is secret, which is hardly the case in real life. Moreover, there are various ways how authorities can delete votes, e.g., by sending invalid ballots to districts with anti-government majority [51,52,24]. In this paper, we make the first step towards systematic modeling and analysis of this kind of threats.

2.1 Postal Voting Procedure for the 2020 Presidential Election

The rules for organizing the election of the President of Poland, with the possibility of postal vote, were published on June 2 and June 3, 2020; the date of the election was set to June 28, 2020. A complete list of legal acts defining the election procedure can be found in [48,45]. For the postal vote, the regulations (mostly concerning the time limits) vary based on the voter's location and her current quarantine status. We focus on the non-expat and non-quarantined voters, but the protocol for the other types of voters is nearly the same.

The protocol consists of several, partly overlapping phases: the setup which involves expression of intention to vote by post, preparation and distribution of election packages (EPs), casting of the vote, validation of votes, and tallying, see Figure 1.

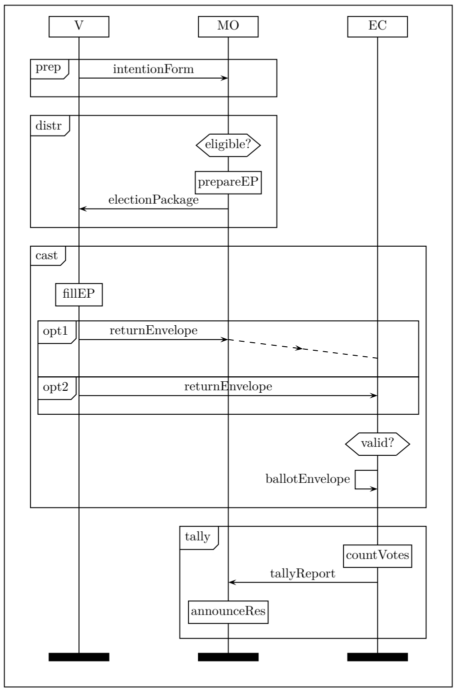

Fig. 1: A simplified diagram of the voting process

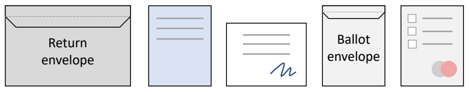

Fig. 2: EP content: return envelope, instruction card, voting card, ballot envelope, stamped ballot

Setup. A voter expresses her intention to vote by post to its local municipal office (MO) at the latest 12 days before the day of election (EDay). This can be done in either oral, written, or electronic form. The intent expression must contain the voter's personal information, such as full name, DOB, id number (PESEL), phone number, email, and residential and postal addresses. If the voter prefers to collect the EP in person, this must be specified instead of a postal address. Additionally, the voter can request to change municipality assigned to her in the voters' register once before the start of the election.

EP preparation. Upon receipt of the intention, a municipal office employee checks the voters' register, and prepares and distributes the EP, provided that the applicant is an eligible voter and no required information is missing. A complete EP (Figure 2) must contain an instruction, a ballot stamped by both the National Electoral Commission (PKW) and the local electoral commission, a voting card, and two envelopes: one for the ballot and one to be returned. The EPs must be delivered or made available for collection to voters no later than five days before the EDay.

Casting. When a complete EP is collected, the voter should put a single 'X' mark against preferred candidate, put the ballot into a ballot envelope, sign a voting card and place it together with ballot envelope into a return envelope. Both ballot and return envelope

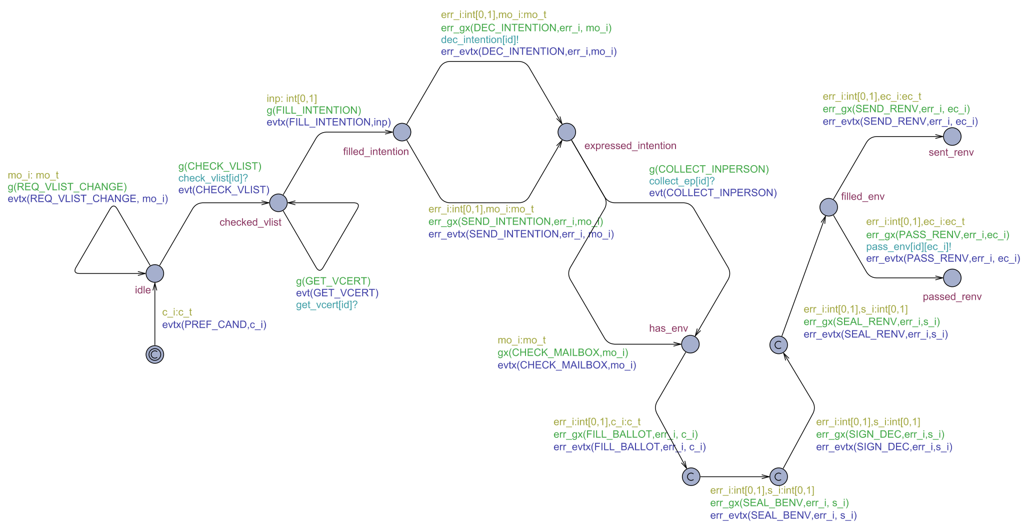
mo_i: mo_t
g(REQ_VLIST_CHANGE)
evtx(REQ_VLIST_CHANGE, mo_i)
idle
c_i:c_t
evtx(PREF_CAND,c_i)
g(CHECK_VLIST)
check_vlist[id]?
evt(CHECK_VLIST)
checked_vlist
g(GET_VCERT)
evt(GET_VCERT)
get_vcert[id]?
inp: int[0,1]
g(FILL_INTENTION)
evtx(FILL_INTENTION,inp)
filled_intention
err_i:int[0,1],mo_i:mo_t
err_gx(DEC_INTENTION,err_i, mo_i)
dec_intention[id]!
err_evtx(DEC_INTENTION,err_i,mo_i)
err_i:int[0,1],mo_i:mo_t
err_gx(SEND_INTENTION,err_i,mo_i)
err_evtx(SEND_INTENTION,err_i, mo_i)
expressed_intention
g(COLLECT_INPERSON)
collect_ep[id]?
evt(COLLECT_INPERSON)
has_env
mo_i:mo_t
gx(CHECK_MAILBOX,mo_i)
evtx(CHECK_MAILBOX,mo_i)
err_i:int[0,1],c_i:c_t
err_gx(FILL_BALLOT,err_i, c_i)
err_evtx(FILL_BALLOT,err_i, c_i)
err_i:int[0,1],s_i:int[0,1]
err_gx(SEAL_BENV,err_i, s_i)
err_evtx(SEAL_BENV,err_i, s_i)
err_i:int[0,1],s_i:int[0,1]
err_gx(SIGN_DEC,err_i,s_i)
err_evtx(SIGN_DEC,err_i,s_i)
err_i:int[0,1],s_i:int[0,1]
err_gx(SEAL_RENV,err_i,s_i)
err_evtx(SEAL_RENV,err_i,s_i)
filled_env
err_i:int[0,1],ec_i:ec_t
err_gx(SEND_RENV,err_i, ec_i)
err_evtx(SEND_RENV,err_i, ec_i)
sent_renv
err_i:int[0,1],ec_i:ec_t
err_gx(PASS_RENV,err_i,ec_i)
pass_env[id][ec_i]!
err_evtx(PASS_RENV,err_i, ec_i)
passed_renv

Fig. 3: Voter template

must be sealed. If there is a deviation in any of the above steps (e.g., the ballot envelope is not sealed), this would invalidate the casting of the vote. Then, the voter must either send the filled REnv to MO, where it will be stored until it is passed to the electoral commission (EC) on the EDay, or turn it in to the assigned electoral commission.

Validation. The EC has to print the voters' list (partitioned according to their municipality) one day before the election at the latest. This will be used to check the validity of the vote and make sure that no person can vote multiple times. If the voter is eligible and the REnv is complete, the BEnv is put into the ballot box.

Tallying. At the end of the EDay, when all REnvs are collected, the commission opens the BEnvs and prepares a voting protocol with information on the number of received REnvs, invalid votes, and the local tally. The protocol is sent to MO which checks it using a proprietary software. If the errors are within the margins allowed by legislation, the MO accepts the protocol. Otherwise, the protocol is rejected and the electoral commission must prepare a new one. When all protocols are accepted and merged, the final tally and the winner are publicly announced.

3 Formal Model of the Procedure

We can now present our preliminary model of the postal voting protocol. The code of the model is available at `https://github.com/polishpostalvote2020/model`.

3.1 Automata Networks, MAS Graphs, and Uppaal Model Checker

We chose the Uppaal model checker as the modeling environment because its user-friendly GUI and a flexible system specification language. The GUI is especially important, as it allows for a preliminary validation of a system specification even at the early stages of modeling.[3] A system in Uppaal is represented as a (parameterized) network of (parameterized) finite automata [8]. The parametrization can be used to define a set of almost identical processes in an easy way. In order to represent occurrence of events, as well as define the available strategies of participants, we use the extension of automata networks to *MAS graphs*, proposed in [31].

A MAS graph allows for the specification of a finite number of agents, possibly interacting via synchronized transitions and/or shared (i.e., global) variables. Formally, it consists of a set of *shared variables* and a number of *agent templates*, each of the templates being a graph with its own set of *locations*, *edges*, and *local variables*. An example agent graph is presented in Figure 3. The locations are depicted by circles; the initial location (one per agent template) is marked by a double circle. Committed locations are marked by a circled 'C'. Whenever an agent is in such a location, the next transition must involve an edge from the committed location. Those are used to create atomic sequences of events, or to encode synchronization, so that there is no interleaving. The edges are annotated by selections (yellow), guards (green), synchronizations (teal) and updates (blue). A selection is a statement of the form `var:type`, which binds the identifier to a value from the given range in a non-deterministic way. These can be used to define a set of parameterized edges in an easy way (both for reading and writing the model). Uppaal allows having function calls for the updates and for the guards if it does not lead to side effects (i.e., without modifying value of any variable).

3.2 MAS Graph for the Postal Voting Procedure

The MAS graph for the procedure consists of the following agent templates: Voter (V, depicted in Figure 3), Electoral Commission (EC, Figure 4), Municipal Office (MO[4]), and Time counter (whose graphs are omitted here due to lack of space). The numbers of agent instances are denoted by NV, NMO, and NEC. Their values, together with the number of candidates NC are specified as part of the model configuration. The sole Time agent is used to impose discrete time constraints on the actions of other agents.

We extend the base model (having infallible agents only) by specifying following mistakes for Voter - may initiate communication and send the forms to a wrong MO or EC, may leave ballot or return envelopes unsealed, misplace the cross mark on the ballot, forget to fill or sign the voting card, attempt to vote by post after obtaining voter's certificate, for Municipal Office (only in some explicitly specified experiments) - may prepare and then distribute ballots without a proper stamp, invalidating those. This yields a hierarchy of the (partially ordered) models, allowing us to study the satisfiability of properties on a finer-grained level, and furthermore get a better understanding of the potential impact of human errors on the system.

The agent templates include a variety of behaviours that can result from human errors as well as purposeful misbehavior. For example, a Voter may not fill the forms properly,

[3] In Uppaal, system components are defined together with their graph-like (local) representation, which could greatly facilitate in reducing the number of bugs/errors, improve understanding and presentation of the model, and provide more confidence that *we know what we are modelling and if it is actually what wanted to model.*

[4] Its template consists of a single location with multiple self-loop edges; the figure was omitted due to space constraints.

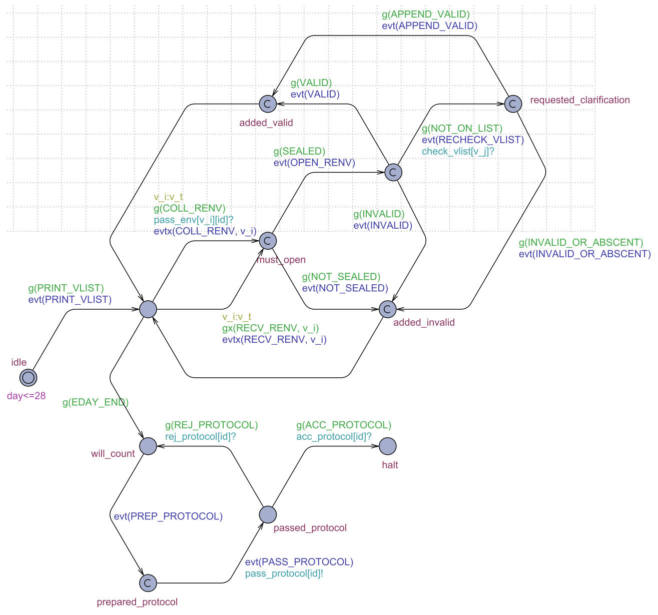

Fig. 4: Electoral Commission template

or attempting to communicate with the wrong municipal office or electoral commission. Furthermore, a Municipal Office can send out an invalid ballot by using a photocopied rather than genuine stamp (which actually happened during the election). In this version, we do not explicitly model the postal services, and thus omit the possible malicious or erroneous actions of postal clerks, or an adversary.[5] Instead, we focus on analysis of human interactions and possible effects of their mistakes (as deviation from the expected behaviour). Similarly, we omit rare events, e.g., those that involve the power of attorney for in-person hand-in of the REnv.

The following data structures are used within the templates:

```
typedef int[1,NC] c_t;              // candidate id
typedef int[1,NV] v_t;              // voter id
typedef int[-NMO,-1] mo_t;          // municipal office id
typedef int[-NMO-NEC,-NMO-1] ec_t;  // election commission id
typedef int[-NMO-NEC,NV] addr_t;    // domain of all agents ids
typedef struct{
    addr_t src;       // sender id
    addr_t dst;       // receiver id
    addr_t addr;      // address for EP delivery
    bool inperson;    // in-person collection preference
    v_tx pesel_of;    // PESEL number
}IntentionForm;
typedef struct{
    bool sealed;        // envelope sealed
    bool pkw_stamp;     // National Electoral Commission stamp
    bool dec_stamp;     // district electoral commission stamp
    int[0,2] cell[c_t]; // number of Xs near candidate cell
                        // (2 for any number greater than 1)
}Benv;                  // ballot envelope
typedef struct{
```

[5] It is worth noting that having a modular representation allows extending the model with other types of agents, including adversary, if needed.

```
    addr_t src;          // sender id
    addr_t dst;          // recipient id
    Benv benv;
    mo_tx stamp;         // envelope stamp
    bool sealed;         // envelope was sealed
    bool dec_signature;  // voter's card signed
    v_tx dec_pesel;      // voter's card PESEL
}Renv;                   // return envelope
typedef struct{
    addr_t src;    // sender id
    addr_t dst;    // recipient id
    Renv renv;
}ElectionPackage;  // election package
typedef struct{
    addr_t mo_addr;  // assigned municipality
    addr_t ec_addr;  // assigned commission
    cmt_t comment;   // nominal field for
    bool changed;    // recently changed
}v_record           // an entry in the voters registry
```

Notice that the space of active agent names (or rather their unique identifiers) is partitioned with respect to the agent type. This allows to model intended parties for a given communication instance in an easy way. All the variables in Uppaal are assigned with either explicit initial or implicit default values. Thus, sometimes aliases which extend the existing data types will be used (e.g., `c_tx` for `int[0,NC]`).

Additionally, to improve readability and keep the graphical view 'clean' we use the following convention for the edge labels:

– all updates are of the form `evt(EVT_CODE)` or `evtx(EVT_CODE,EVT_PARAMS)`;
– all guards are of the form `g(EVT_CODE)` or `gx(EVT_CODE,EVT_PARAMS)`.

Where `EVT_CODE` is a string of characters (i.e., named value from the pre-defined enum of events), representing action name by means of a natural language. The sets of available/specified actions are disjoint among agents of a distinct type. On the level of the model code, functions `evt/evtx` and `g/gx` are declared using switch statement matching passed argument of `EVT_CODE` to its case clause, which should facilitate towards the modularity of a model and higher integrity of its graphical presentation.

4 Verification

As the next step, we specify some relevant properties of the voting system by formulas of multi-agent logics, in particular the branching-time temporal logic $\mathbf{CTL}$ and the strategic logic $\mathbf{ATL}$. Then, we transform them to a form that can be interpreted by Uppaal, and run the model checking. To mitigate the impact of state-space explosion, we use abstraction on variables, proposed in [31].

4.1 Specification of Properties

Following [53,29,32,33], we use the formulas of multi-agent logics to specify some interesting requirements on the election system. In particular, we use the branching-time temporal logic $\mathbf{CTL}$ [23] and strategic logic $\mathbf{ATL}$ [1]. We focus on the following requirements[6]:

[6] Authors would want to stress out that the given properties are just an example, merely for illustration purpose, and they should not be viewed as complete list of requirements.

(***P1***) The number of correctly received ballots cannot exceed the number of sent ballots (a weak variant of resistance against ballot stuffing);

(***P2***) For every voter, cast vote must be properly recorded and reflected by the tally (tally integrity);

(***P3***) The authorities should have no strategy to invalidate certain votes, even when the voters' preferences are known (no strategic ballot removal).

We formalize (*P1*) and (*P2*) by the following **CTL** formulas:

$$\varphi_{P1} \equiv \mathsf{A}\Box(\textstyle\sum_{i=1}^{NB} b\_received_i \leq \sum_{j=1}^{NV}(ep\_sent_j))$$

$$\varphi_{P2} \equiv \mathsf{A}\Box(elec\_end \wedge voted_{i,j} \Rightarrow tallied_{i,j})$$

where NB and NV stand for the maximum number of ballots and voters accordingly, $i \in NV$ is an arbitrary fixed voter, and $j \in NC$ a candidate. The (*P1*) says that, for all possible execution paths (A) and all future time points ($\Box$), the sum of received ballots must not exceed the number of sent election packets. Similarly, (*P2*) says that if election is closed and voter has cast her vote for candidate j, then it must be tallied for j.

Furthermore, we formalize (*P3*) by the following formula of **ATL** (in fact, we formalize the negation of (*P3*) and focus on ballot removal by a Municipal Office, thus expressing that the MO can strategically remove ballots):

$$\varphi_{\neg P3} \equiv \langle\langle MO_k\rangle\rangle\Box \bigwedge_{i\in NV} (vreg_{i,k} \wedge pref_{i,j} \wedge elec\_end \Rightarrow \neg tallied_{i,j})$$

where $i \in NV$, $j \in NC$ and $k \in MO$ is a municipal office. The reading of $\varphi_{\neg P3}$ is: "Municipal office k has a strategy ($\langle\langle MO_k\rangle\rangle$) such that, no matter what the other agents do, at all future time points ($\Box$) for any voter i if registered in this municipality ($vreg_{i,k}$), prefers candidate j ($pref_{i,j}$) and election is already closed (*elec_end*), then her vote will not be tallied correctly ($\neg tallied_{i,j}$).

4.2 From Agent Logics to Uppaal Specifications

Our formalization of resistance to ballot stuffing and tally integrity has a straightforward transcription in Uppaal specification language:

(φ_{P1}) `A[] (b_recv<=ep_sent) ,`

(φ_{P2}) `A[] (Time.end and (Voter(i).sent_renv or Voter(i).passed_renv) imply recorded_link[i]==Voter(i).pref_cand) ,`

where `b_recv`, `ep_sent` and `recorded_link` are auxiliary variables added only for the verification of related property, and representing the number of received ballots, the number of sent election packages, and the mapping from voters to the way their votes have been tallied. Moreover, `pref_cand` is the voter's local variable storing her preferred candidate, `sent_renv` and `passed_renv` are labels of the corresponding locations in the voter's agent graph (see Figure 3).

Unfortunately, Uppaal does not offer the verification of strategic abilities thus does not admit **ATL** operators. To deal with that, we propose to approximate formula $\varphi_{\neg P3}$ by its *under-approximation* $\varphi^{-}_{\neg P3}$ and *over-approximation* $\varphi^{+}_{\neg P3}$, both of which are **CTL** formulas that satisfy the following conditions:

$$M \models_{\mathbf{CTL}} \varphi^{-}_{\neg P3} \quad\Rightarrow\quad M \models_{\mathbf{ATL}} \varphi_{\neg P3}$$

$$M \models_{\mathbf{ATL}} \varphi_{\neg P3} \quad\Rightarrow\quad M \models_{\mathbf{CTL}} \varphi^{+}_{\neg P3}$$

```
Concrete model
typedef struct{
    bool sealed;
    bool pkw_stamp;
    bool dec_stamp;
    int[0,2] cell[c_t];
}Benv;
typedef struct{
    addr_t src;
    addr_t dst;
    Benv benv;
    mo_tx stamp;
    bool sealed;
    bool dec_signature;
    v_tx dec_pesel;
}Renv;
typedef struct{
    addr_t src;
    addr_t dst;
    Renv renv;
}ElectionPackage;
```

```
Abstact model
typedef struct{
    bool invalid;
    c_tx cell;
}Benv;
typedef struct{
    addr_t dst;
    bool invalid;
    Benv benv;
}Renv;
typedef struct{
    bool sent;
    Renv renv;
}ElectionPackage;
```

Fig. 5: Fragment of code resulting from the abstraction, where all evaluations invalidating a vote are merged into a single variable (for REnv and BEnv)

That is, whenever $\varphi^{-}_{\neg P3}$ is true in a model, $\varphi_{\neg P3}$ must also be true there. Moreover, if $\varphi^{+}_{\neg P3}$ is false in a model, $\varphi_{\neg P3}$ must also be false. We use the following approximations:

$$\varphi^{-}_{\neg P3} \equiv \mathsf{A}\Box \bigwedge_{i \in NV} (vreg_{i,k} \wedge pref_{i,j} \wedge elec\_end \Rightarrow \neg tallied_{i,j})$$

$$\varphi^{+}_{\neg P3} \equiv \mathsf{E}\Box \bigwedge_{i \in NV} (vreg_{i,k} \wedge pref_{i,j} \wedge elec\_end \Rightarrow \neg tallied_{i,j})$$

This follows the intuition that, if ψ is guaranteed to always hold on all execution paths ($\mathsf{A}\Box\psi$), then it must also hold when MO plays strategically ($\langle\langle MO \rangle\rangle\Box\psi$). Moreover, if MO has a strategy to maintain ψ ($\langle\langle MO \rangle\rangle\Box\psi$), then ψ must always hold on at least one path ($\mathsf{E}\Box\psi$).

Now, formula $\varphi^{-}_{\neg P3}$ can be fed directly to Uppaal. Unfortunately, this is not the case for the upper approximation $\varphi^{+}_{\neg P3}$, as Uppaal does not interpret the $\mathsf{E}\Box$ combination of **CTL** operators correctly.[7] On the other hand, Uppaal's `E[]` combination is an over-approximation of the $\mathbf{CTL}^{\star}$ $\mathsf{E}\Box$ combination. Thus, we can use it to provide "over-over-approximation" of the original specification, which finally obtains the following list of Uppaal inputs:

($\varphi^{-}_{\neg P3}$) `A[] forall(i:v_t)(Time.end and vlist[i].mo_addr==k and vpref[i]==j imply recorded_link[i]!=j)` ,

($\varphi^{++}_{\neg P3}$) `E[] forall(i:v_t)(Time.end and vlist[i].mo_addr==k and vpref[i]==j imply recorded_link[i]!=j)` ,

where `vlist` refers to the voters' registry.

4.3 Mitigating State Space Explosion by Abstraction of Variables

One of the biggest challenges in the practical application of model checkers is the so called *state-space explosion*. Typically, model checking involves the generation of a

[7] The satisfaction of **CTL** operators in a transition system is interpreted over *maximal runs*, i.e., ones that are either infinite or end in a state with no outgoing transitions. In contrast, Uppaal looks at *all finite runs*. While this does not change the semantics of $\mathsf{A}\Box$ and $\mathsf{E}\Diamond$, the interpretation of both $\mathsf{A}\Diamond$ and $\mathsf{E}\Box$ becomes nonstandard.

huge state/transition graph that includes all the possible states (i.e., configurations) of the system. Clearly, the number of such configurations is exponential in the number of processes and their components – in our case, the number of agent instances and their local variables [18]. This is easy to see in our experimental results for formula φ_{PI}, see Table 1(left), the part under "φ_{PI} (concrete)," with the clear exponential growth of the verification time t and memory use m. As a consequence, the verification of φ_{PI} on the model presented in section 3 scales up to only 2 voters, 1 municipal office, and 3 electoral commissions for an election with 3 candidates.

Mitigating state-space explosion has been an important topic of research for over 30 years. The most important techniques include partial-order reduction [47], symbolic verification [41], bounded and unbounded model checking [42] and state/action abstraction [19]. In particular, abstraction is an intuitive model reduction method, based on the idea of clustering "similar" states of the system (so called *concrete states*) into *abstract states*, hopefully reducing the model to a manageable size. The actual clustering must be carefully crafted. On the one hand, it must only remove information that is irrelevant for the verification of a given property, otherwise the verification results for the abstract model will be inconclusive with respect to the original model (so called "concrete model"). On the other hand, it has to remove sufficiently much of the concrete model, so that the model checking becomes efficient.

In this work, we use an intuitive and easy to use abstraction scheme, based on the removal of variables from agent graphs [31]. The method allows to select a subset of local variables to be removed (possibly with a subset of locations to serve as the scope of the abstraction). For example, the name of the voter's preferred candidate is irrelevant for the verification of resistance to ballot stuffing, hence the corresponding variables can be omitted in the voters' agent graphs. The abstraction generates *two* abstract models. The first one *under-approximates* the concrete model, in the sense that if formula ψ returns `true` on the abstract model, it must be also be true in the concrete model. The second *over-approximates* the concrete model, i.e., if ψ returns `false` on the abstract model, it must be also be false in the concrete model.

Alternatively, the user can define a mapping from the variables to a fresh variable that merge some of the information that used to be stored in the removed variables. For example, we might map the complex representation of all potential ballot faults (unsealed ballot envelope, missing stamps, more than one 'X' on the ballot, for one or more candidates) to a single boolean variable `invalid`, see Figure 5.

In general, these abstraction parameters should be picked firstly to reduce the number of induced states of global model, and secondly to match the property, so that verification is conclusive. Naturally, there are variety of ways to chose fitting parameters for a given property; it is also possible that two distinct properties are matched by the same ones, and therefore same abstract models. We refer to [31] for the formal definitions, correctness proofs, and extensive discussion of the method.

4.4 Verification Experiments

Based on the input prepared in the previous sections, we have conducted a number of model checking experiments. All the results presented here were obtained with Uppaal 4.1.24 (32 bit) on a laptop with Intel i7-8665U 2.11 GHz CPU, running Ubuntu 20.04 on

WSL2 with 4 GB RAM. The outcome of the experiments is shown in Tables 1 and 2. The notation is as follows:

- *conf* denotes the configuration of the experiment, i.e., the number of voters, Municipal Offices, Electoral Commissions, and election candidates;
- *Sat* reports the verification output, i.e., whether the model checker returned `true` or `false`;
- *t* and *m* show the time and memory used in the verification, with `memout` indicating that the model checking process ran out of memory.

Table 1(left) presents the experimental results for our formalization of weak resistance to ballot stuffing. The formula has turned out to be true in all the completed verification runs. However, as already observed, the verification scales rather badly due to state-space explosion. To mitigate that, we reduced the models by abstracting away the identity of candidates, and simplified the data structures representing the intention form and the election package. Moreover, we mapped variables `b_recv` and `ep_sent` to a single fresh variable `ballot_diff = ep_sent-b_recv`, so that the requirement specification became `AG(ballot_diff>=0)`. The results for model checking the under-approximating abstract model are also presented in Table 1(left).

Since the output of under-approximation was `true`, we conclude that φ_{P1} is also true in the original (concrete) model. Note that, for some configurations, the abstraction allowed to run the verification faster by orders of magnitude. Moreover, it allowed for the model checking of scenarios with 3 voters, 1 MO, 1 EC, and 3 candidates, i.e., one more voter than in the concrete case. This might seem slight, but in some cases 3 voters are necessary to demonstrate non-trivial attacks on a voting system [3].

For tally integrity (φ_{P2}), we used formula φ_{P2} proposed in Section 4.2. In this case, we additionally generated the under-approximating abstract models obtained by (a) mapping all the candidate names except j to a fresh value j', (b) for all voters other than i, removing their memory of the choices associated with their intention forms and election packages after they send those. The results in Table 1(right) show that the verification output was not conclusive, i.e., they do not imply whether φ_{P2} is true or false in the original model. However, the verification becomes conclusive under the assumption that voter i makes no errors and strictly follows the protocol, see the rightmost part of the table. In that case, we get `true` as the output, thus concluding the original property holds as well.

Lastly, the experimental results for strategic ballot removal ($\varphi_{\neg P3}$) are presented in Table 2. The table first shows the (inconclusive) output of model checking for under-approximation w.r.t. the formula, under-approximation w.r.t. the formula and the model, and over-approximation w.r.t. the formula. Thus, the original property might (but does not have to) be satisfied. Then, we fix a strategy for MO in the model, so that the municipal office sends a ballot with invalid stamps whenever the voter intends to vote for the "unwelcome" candidate. The results in the rightmost part of Table 2 show now conclusive output: the under-approximation is true, so the original property must be true as well. Thus, the proposed strategy indeed achieves the goal specified by $\varphi_{\neg P3}$. Note that it is essential to couple matching formula- and model-related approximations, otherwise the procedure is not sound. In our case, this meant using the *under-approximating* formula $\varphi^{-}_{\neg P3}$ with the *under-approximating* abstract model of the procedure.

Table 1: Experimental results for model checking of φ_{P1} (left) and φ_{P2} (right)

conf	φ_{P1} (concrete)			φ_{P1} (abstract)		
	Sat	t (s)	m(MB)	Sat	t (s)	m(MB)
1,1,1,1	true	0.07	31	true	0.07	30
1,1,1,2	true	0.10	31	true	0.07	30
1,1,1,3	true	0.12	31	true	0.07	30
1,1,2,1	true	0.12	31	true	0.08	31
1,1,2,2	true	0.20	31	true	0.08	31
1,1,2,3	true	0.27	31	true	0.08	31
1,1,3,1	true	0.29	31	true	0.14	31
1,1,3,2	true	0.54	32	true	0.14	31
1,1,3,3	true	0.80	33	true	0.14	31
1,1,4,1	true	0.92	33	true	0.39	31
1,1,4,2	true	1.8	35	true	0.39	31
1,1,4,3	true	2.7	36	true	0.39	31
1,2,2,1	true	0.27	31	true	0.14	31
1,2,2,2	true	0.48	32	true	0.14	31
1,2,2,3	true	0.71	33	true	0.14	31
1,2,3,1	true	0.75	32	true	0.32	31
1,2,3,2	true	1.4	34	true	0.32	31
1,2,3,3	true	2.2	35	true	0.32	31
1,2,4,1	true	2.5	35	true	0.97	32
1,2,4,2	true	4.9	39	true	0.97	32
1,2,4,3	true	7.5	58	true	0.97	32
2,1,1,1	true	5.0	91	true	1.0	32
2,1,1,2	true	21	271	true	1.0	32
2,1,1,3	true	48	618	true	1.0	32
2,1,2,1	true	15	172	true	2.9	34
2,1,2,2	true	64	581	true	2.9	34
2,1,2,3	true	148	1332	true	2.9	34
2,1,3,1	true	52	330	true	9.5	57
2,1,3,2	true	213	1180	true	9.5	57
2,1,3,3	true	496	2796	true	9.5	57
2,1,4,1	true	190	638	true	35	83
2,1,4,2	true	789	2429	true	35	83
2,1,4,3	memout			true	35	83
2,2,2,1	true	135	990	true	18	76
2,2,2,2	true	558	3901	true	18	76
2,2,2,3	memout			true	18	76
2,2,3,1	true	445	2168	true	59	152
2,2,3,2	memout			true	59	152
2,2,3,3	memout			true	59	152
2,2,4,1	memout			true	203	350
2,2,4,2	memout			true	203	350
2,2,4,3	memout			true	203	350
3,1,1,1	memout			true	80	365
3,1,1,2	memout			true	80	365
3,1,1,3	memout			true	80	365
3,1,2,1	memout			true	241	818
3,1,2,2	memout			true	241	818
3,1,2,3	memout			true	241	818
3,1,3,1	memout			true	793	1882
3,1,3,2	memout			true	793	1882
3,1,3,3	memout			true	793	1882
3,1,4,1	memout			memout		

conf	φ_{P2} (concrete)			φ_{P2} (abstract)			φ_{P2} (honest abstract)		
	Sat	t (s)	m(MB)	Sat	t (s)	m(MB)	Sat	t (s)	m(MB)
1,1,1,1	false	0.07	31	false	0.06	30	true	0.05	30
1,1,1,2	false	0.09	31	false	0.06	31	true	0.06	30
1,1,1,3	false	0.11	31	false	0.07	31	true	0.06	30
1,1,2,1	false	0.12	31	false	0.08	31	true	0.05	31
1,1,2,2	false	0.23	31	false	0.10	31	true	0.07	31
1,1,2,3	false	0.25	31	false	0.13	31	true	0.08	31
1,1,3,1	false	0.31	31	false	0.15	31	true	0.12	31
1,1,3,2	false	0.52	32	false	0.25	31	true	0.13	31
1,1,3,3	false	0.75	33	false	0.35	31	true	0.17	31
1,1,4,1	false	0.90	33	false	0.40	32	true	0.19	31
1,1,4,2	false	1.7	35	false	0.76	32	true	0.33	31
1,1,4,3	false	2.6	36	false	1.1	33	true	0.69	31
1,2,2,1	false	0.26	32	false	0.14	31	true	0.09	31
1,2,2,2	false	0.45	32	false	0.22	31	true	0.13	31
1,2,2,3	false	0.66	33	false	0.31	31	true	0.16	31
1,2,3,1	false	0.70	32	false	0.33	31	true	0.16	31
1,2,3,2	false	1.4	34	false	0.61	32	true	0.26	31
1,2,3,3	false	2.0	35	false	1.3	32	true	0.36	31
1,2,4,1	false	2.3	35	false	1.0	32	true	0.40	31
1,2,4,2	false	4.7	39	false	2.0	33	true	0.97	32
1,2,4,3	false	7.1	42	false	3.0	34	true	1.1	32
2,1,1,1	false	1.8	36	false	0.95	31	true	0.28	31
2,1,1,2	false	6.9	99	false	3.0	34	true	0.72	31
2,1,1,3	false	16	220	false	6.2	54	true	1.3	32
2,1,2,1	false	7.2	79	false	2.8	34	true	0.71	31
2,1,2,2	false	30	235	false	9.2	57	true	2.3	32
2,1,2,3	false	68	535	false	20	100	true	3.4	34
2,1,3,1	false	26	137	false	9.5	41	true	2.0	33
2,1,3,2	false	104	516	false	32	114	true	5.3	36
2,1,3,3	false	239	1180	false	68	200	true	10	57
2,1,4,1	false	95	305	false	35	84	true	6.4	38
2,1,4,2	false	384	1199	false	119	233	true	18	65
2,1,4,3	false	885	2768	false	254	477	true	33	97
2,2,2,1	false	60	377	false	19	77	true	4.0	33
2,2,2,2	false	241	1439	false	61	189	true	10	53
2,2,2,3	false	549	3338	false	126	382	true	19	74
2,2,3,1	false	205	829	false	61	153	true	10	38
2,2,3,2	false	832	3318	false	199	428	true	29	81
2,2,3,3	memout			false	413	881	true	55	130
2,2,4,1	false	729	2026	false	211	338	true	33	70
2,2,4,2	memout			false	696	1075	true	92	174
2,2,4,3	memout			false	1444	2264	true	177	307
3,1,1,1	false	60	683	false	73	340	true	10	54
3,1,1,2	memout			false	384	1855	true	37	153
3,1,1,3	memout			memout			true	88	332
3,1,2,1	false	346	2387	false	224	769	true	28	99
3,1,2,2	memout			memout			true	99	293
3,1,2,3	memout			memout			true	230	667
3,1,3,1	memout			false	775	1827	true	81	201
3,1,3,2	memout			memout			true	283	639
3,1,3,3	memout			memout			true	661	1503
3,1,4,1	memout			memout			true	261	451
3,1,4,2	memout			memout			true	920	1518
3,1,4,3	memout			memout			true	2157	3626
3,2,2,1	memout			memout			true	317	667
3,2,2,2	memout			memout			true	1143	2303
3,2,2,3	memout			memout			memout		
3,2,3,1	memout			memout			true	895	1427
3,2,3,2	memout			memout			memout		
3,2,3,3	memout			memout			memout		
3,2,4,1	memout			memout			true	1065	2433
3,2,4,2	memout			memout			memout		
3,2,4,3	memout			memout			memout		
4,1,1,1	memout			memout			memout		

Table 2: Experimental results for model checking of $\varphi^{-}_{\neg P3}$ and $\varphi^{++}_{\neg P3}$ in a model, where MO may prepare EP without a valid stamp

conf	$\varphi^{-}_{\neg P3}$ (concrete)			$\varphi^{-}_{\neg P3}$ (abstract)			$\varphi^{++}_{\neg P3}$ (concrete)			$\varphi^{-}_{\neg P3}$ (str. abstract)		
	Sat	t (s)	m(MB)	Sat	t (s)	m(MB)	Sat	t (s)	m(MB)	Sat	t (s)	m(MB)
1,1,1,1	false	0.13	31	false	0.058	30	true	0.054	31	true	0.052	30
1,1,1,2	false	0.22	31	false	0.096	31	true	0.049	31	true	0.10	30
1,1,1,3	false	0.40	32	false	0.15	31	true	0.049	31	true	0.17	31
1,1,2,1	false	0.19	31	false	0.13	31	true	0.051	31	true	0.088	31
1,1,2,2	false	0.57	32	false	0.20	31	true	0.064	31	true	0.16	31
1,1,2,3	false	1.1	34	false	0.38	31	true	0.067	31	true	0.35	31
1,1,3,1	false	0.56	32	false	0.20	31	true	0.055	32	true	0.16	31
1,1,3,2	false	1.7	35	false	0.82	31	true	0.060	32	true	0.46	31
1,1,3,3	false	3.8	39	false	1.1	32	true	0.063	32	true	0.98	32
1,1,4,1	false	1.7	34	false	0.85	32	true	0.062	32	true	0.30	31
1,1,4,2	false	6.2	42	false	1.9	34	true	0.066	32	true	1.4	33
1,1,4,3	false	14	71	false	4.0	36	true	0.067	32	true	3.3	35
1,2,2,1	false	0.48	32	false	0.22	31	true	0.061	32	true	0.15	31
1,2,2,2	false	1.6	34	false	0.54	31	true	0.077	32	true	0.40	31
1,2,2,3	false	3.3	54	false	1.0	32	true	0.064	32	true	1.2	31
1,2,3,1	false	1.4	34	false	0.46	31	true	0.087	32	true	0.26	31
1,2,3,2	false	5.1	39	false	1.5	32	true	0.076	32	true	1.1	32
1,2,3,3	false	11	81	false	3.0	34	true	0.082	32	true	2.5	33
1,2,4,1	false	4.9	37	false	1.5	33	true	0.075	32	true	0.74	32
1,2,4,2	false	18	85	false	4.9	36	true	0.075	32	true	3.5	34
1,2,4,3	false	39	142	false	10	41	true	0.075	32	true	8.4	39
2,1,1,1	false	8.3	115	false	2.0	33	true	0.065	31	true	0.62	31
2,1,1,2	false	78	833	false	23	120	true	0.076	31	true	13	77
2,1,1,3	false	323	3496	false	105	495	true	0.062	32	true	75	337
2,1,2,1	false	36	230	false	6.1	53	true	0.070	32	true	1.6	32
2,1,2,2	false	336	2145	false	71	265	true	0.088	32	true	39	152
2,1,2,3		memout		false	321	1140	true	0.079	32	true	224	748
2,1,3,1	false	105	486	false	20	84	true	0.074	32	true	4.8	36
2,1,3,2		memout		false	244	613	true	0.082	32	true	129	326
2,1,3,3		memout		false	1114	2752	true	0.10	32	true	741	1745
2,1,4,1	false	400	1160	false	75	152	true	0.083	32	true	17	48
2,1,4,2		memout		false	896	1502	true	0.094	32	true	457	761
2,1,4,3		memout			memout		true	0.11	32		memout	
2,2,2,1	false	273	1424	false	40	121	true	0.091	32	true	10	53
2,2,2,2		memout		false	443	1156	true	0.11	32	true	241	604
2,2,2,3		memout			memout		true	0.10	32	true	1332	3282
2,2,3,1	false	904	3352		memout		true	0.097	32	true	32	83
2,2,3,2		memout			memout		true	0.096	32	true	756	1388
2,2,3,3		memout			memout		true	0.094	33		memout	
2,2,4,1		memout			memout		true	0.097	33	true	107	163
2,2,4,2		memout			memout		true	0.099	33	true	2575	3482
2,2,4,3		memout			memout		true	0.14	33		memout	
3,1,1,1		memout			memout		true	0.073	32	true	92	289
3,1,1,2		memout			memout		true	0.070	32		memout	
3,1,1,3		memout			memout		true	0.083	32		memout	
3,1,2,1		memout			memout		true	0.11	32	true	167	622
3,1,2,2		memout			memout		true	0.081	32		memout	
3,1,2,3		memout			memout		true	0.089	32		memout	
3,1,3,1		memout			memout		true	0.093	32	true	296	1446
3,1,3,2		memout			memout		true	0.10	33		memout	
3,1,3,3		memout			memout		true	0.093	33		memout	
3,1,4,1		memout			memout		true	0.11	33	true	811	3828
3,1,4,2		memout			memout		true	0.14	33		memout	
3,1,4,3		memout			memout		true	0.11	33		memout	
3,2,2,1		memout			memout		true	0.14	33	true	1056	6175
3,2,2,2		memout			memout		true	0.12	33		memout	
3,2,2,3		memout			memout		true	0.11	33		memout	
3,2,3,1		memout			memout		true	0.12	33		memout	
3,2,3,2		memout			memout		true	0.15	33		memout	

Despite the technical limitations on the number of voters, it was possible to discover and verify attacks violating φ_{P2} and φ_{P3} respectively. For the next step (which remains as a subject of future work), we would want to scale up the model to larger or even unbounded number of voters (the latter is currently beyond technical feasibility of the tool), or to come up with rigorous arguments that certain number of voters is sufficient for certain cases.

5 Conclusions

In this paper, we demonstrate how multi-agent methodology can be used to specify and analyze the impact of human aspects on security and integrity of voting protocols. We also argue that postal voting protocols provide good material for case studies, that will hopefully increase our understanding of the subject, and help to design better protocols.

Speaking in more concrete terms, we propose a preliminary analysis of the Polish postal vote used in the presidential election of 2020. We use Multi-Agent Graphs to represent the participants and their interaction, and formulas of multi-agent logics **CTL** and **ATL** to encode interesting properties. Then, we transform those to match the input of the state-of-art model checker Uppaal. This way, the obtained models are given an intuitive visual representation and a modular structure that allows for easier modifications and detection of errors. We also use a recently proposed method of state abstraction by variable removal to reduce the models and mitigate state-space explosion. The method is guaranteed to preserve the truth of universal **CTL** specifications and thus generates correct-by-design abstract models. No less importantly, it is easy to use, requires almost no technical knowledge from the user, and provides significant savings in terms of the verification performance.

Despite the limitations of Uppaal in the expressive power of its property specification language, we have managed to conclusively verify the selected properties for nontrivial configurations of voters, electoral commissions, and candidates. To this end, we used approximation over formulas (by providing weaker or stronger versions of the original requirements) and approximation w.r.t. models (by generating appropriate abstract models). Choosing the right approximations was by no means obvious and required some skill. We believe, however, that this is inevitable: successful formal analysis of real-life scenarios requires both science and art. With more than a little bit of understanding and domain knowledge.

For the future work we plan to employ alternative verification tool to conduct analysis of a broader scope of interesting properties, and adapt our abstraction methods for that. One of the interesting directions is to adopt a methodology defining a families of possible human mistakes as in [49], where human agent deviates from the protocol through a combination of skipping, modifying, or adding action(s), or in [11], which also advocates using the epistemic modal logic distinguishing between knowledge and possession.

Acknowledgements. The work was supported by NCBR Poland and FNR Luxembourg under the PolLux/FNR-CORE project STV (POLLUX-VII/1/2019).

References

1. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time Temporal Logic. Journal of the ACM **49**, 672–713 (2002). `https://doi.org/10.1145/585265.585270`

2. Andrzej Rzepliński, M.S.: Statement on election irregularities [polish: Oświadczenia w sprawie nieprawidłowości dotyczących wyborów], `https://ow.org.pl/2020/08/05/oswiadczenia-w-sprawie-nieprawidlowosci-dotyczacych-wyborow/`
3. Arapinis, M., Cortier, V., Kremer, S.: When are three voters enough for privacy properties? In: Proceedings of ESORICS. Lecture Notes in Computer Science, vol. 9879, pp. 241–260. Springer (2016). `https://doi.org/10.1007/978-3-319-45741-3\_13`
4. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press (2008)
5. Basin, D.A., Gersbach, H., Mamageishvili, A., Schmid, L., Tejada, O.: Election security and economics: It's all about Eve. In: Proceedings of E-Vote-ID. pp. 1–20 (2017). `https://doi.org/10.1007/978-3-319-68687-5_1`
6. Basin, D.A., Radomirovic, S., Schmid, L.: Modeling human errors in security protocols. In: Computer Security Foundations Symposium, CSF. pp. 325–340. IEEE Computer Society (2016). `https://doi.org/10.1109/CSF.2016.30`
7. Beckert, B., Beuster, G.: A method for formalizing, analyzing, and verifying secure user interfaces. In: International Conference on Formal Engineering Methods. pp. 55–73. Springer (2006)
8. Behrmann, G., David, A., Larsen, K.: A tutorial on UPPAAL. In: Formal Methods for the Design of Real-Time Systems: SFM-RT. pp. 200–236. No. 3185 in LNCS, Springer (2004)
9. Bella, G., Curzon, P., Giustolisi, R., Lenzini, G.: A socio-technical methodology for the security and privacy analysis of services. In: COMPSAC Workshops. pp. 401–406. IEEE Computer Society (2014). `https://doi.org/10.1109/COMPSACW.2014.69`
10. Bella, G., Curzon, P., Lenzini, G.: Service security and privacy as a socio-technical problem. J. Comput. Secur. **23**(5), 563–585 (2015). `https://doi.org/10.3233/JCS-150536`
11. Bella, G., Giustolisi, R., Schürmann, C.: Modelling human threats in security ceremonies. Journal of Computer Security (Preprint), 1–23 (2022)
12. Benaloh, J., Ryan, P.Y., Teague, V.: Verifiable postal voting. In: Cambridge International Workshop on Security Protocols. pp. 54–65. Springer (2013)
13. Blanchet, B., et al.: Modeling and verifying security protocols with the applied pi calculus and proverif. Foundations and Trends® in Privacy and Security **1**(1-2), 1–135 (2016)
14. Bruni, A., Drewsen, E., Schürmann, C.: Towards a mechanized proof of Selene receipt-freeness and vote-privacy. In: Proceedings of E-Vote-ID. Lecture Notes in Computer Science, vol. 10615, pp. 110–126. Springer (2017). `https://doi.org/10.1007/978-3-319-68687-5\_7`
15. Bruni, A., Carbone, M., Giustolisi, R., Mödersheim, S., Schürmann, C.: Security protocols as choreographies. In: Protocols, Strands, and Logic - Essays Dedicated to Joshua Guttman on the Occasion of his 66.66th Birthday. Lecture Notes in Computer Science, vol. 13066, pp. 98–111. Springer (2021). `https://doi.org/10.1007/978-3-030-91631-2\_5`
16. Buldas, A., Mägi, T.: Practical security analysis of e-voting systems. In: Proceedings of IWSEC. Lecture Notes in Computer Science, vol. 4752, pp. 320–335. Springer (2007)
17. Carlos, M.C., Martina, J.E., Price, G., Custódio, R.F.: A proposed framework for analysing security ceremonies. In: SECRYPT. pp. 440–445. SciTePress (2012)
18. Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R., et al.: Handbook of model checking, vol. 10. Springer (2018)
19. Clarke, E., Grumberg, O., Long, D.: Model checking and abstraction. ACM Transactions on Programming Languages and Systems **16**(5), 1512–1542 (1994)
20. Cortier, V., Filipiak, A., Lallemand, J.: Beleniosvs: Secrecy and verifiability against a corrupted voting device. In: 2019 IEEE 32nd Computer Security Foundations Symposium (CSF). pp. 367–36714. IEEE (2019)
21. Cortier, V., Galindo, D., Turuani, M.: A formal analysis of the neuchâtel e-voting protocol. In: 2018 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 430–442. IEEE (2018)

22. Dastani, M., Hindriks, K., Meyer, J. (eds.): Specification and Verification of Multi-Agent Systems. Springer (2010)
23. Emerson, E.: Temporal and modal logic. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, vol. B, pp. 995–1072. Elsevier (1990)
24. Fakt.pl: Ballot papers without stamp were delivered. Will the votes be invalid? [Polish: Dostali karty do głosowania bez pieczęci. Czy głosy będą nieważne?], `https://www.fakt.pl/wydarzenia/polityka/dostali-karty-do-glosowania-bez-pieczeci-czy-glosy-beda-niewazne/6cwhzg4`
25. Haines, T., Goré, R., Tiwari, M.: Verified verifiers for verifying elections. In: Proceedings of CCS. pp. 685–702. ACM (2019). `https://doi.org/10.1145/3319535.3354247`
26. Haines, T., Goré, R., Sharma, B.: Did you mix me? formally verifying verifiable mix nets in electronic voting. In: 42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021. pp. 1748–1765. IEEE (2021). `https://doi.org/10.1109/SP40001.2021.00033`
27. Hao, F., Ryan, P.: Real-World Electronic Voting: Design, Analysis and Deployment. Auerbach Publications (2016)
28. Holroyd, M.: Dutch election: Rule change to accept wrongly sealed mail-in ballots. Euronews (2021), `https://www.euronews.com/2021/03/17/dutch-election-rule-change-to-accept-wrongly-sealed-mail-in-ballots`
29. Jamroga, W., Knapik, M., Kurpiewski, D.: 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, vol. 11143, pp. 100–116. Springer (2018)
30. Jamroga, W., Tabatabaei, M.: Preventing coercion in e-voting: Be open and commit. In: Electronic Voting: Proceedings of E-Vote-ID 2016. Lecture Notes in Computer Science, vol. 10141, pp. 1–17. Springer (2017). `https://doi.org/10.1007/978-3-319-52240-1_1`
31. Jamroga, W., Kim, Y.: Practical abstraction for model checking of multi-agent systems (2022). `https://doi.org/10.48550/ARXIV.2202.12016`
32. Jamroga, W., Kim, Y., Kurpiewski, D., Ryan, P.Y.A.: Towards model checking of voting protocols in uppaal. In: Proceedings of E-Vote-ID. Lecture Notes in Computer Science, vol. 12455, pp. 129–146. Springer (2020). `https://doi.org/10.1007/978-3-030-60347-2\_9`
33. Jamroga, W., Kurpiewski, D., Malvone, V.: Natural strategic abilities in voting protocols. In: Proceedings of STAST 2020 (2021), to appear
34. Jamroga, W., Mestel, D., Roenne, P.B., Ryan, P.Y.A., Skrobot, M.: A survey of requirements for COVID-19 mitigation strategies. Bulletin of The Polish Academy of Sciences: Technical Science **69**(4), e137724 (2021). `https://doi.org/10.24425/bpasts.2021.137724`
35. Killer, C., Stiller, B.: The swiss postal voting process and its system and security analysis. In: International Joint Conference on Electronic Voting. pp. 134–149. Springer (2019)
36. Kurpiewski, D., Jamroga, W., Knapik, M.L.: Stv: Model checking for strategies under imperfect information. In: Proceedings of the 18th International Conference on Autonomous Agents and Multiagent Systems AAMAS 2019. pp. 2372–2374. IFAAMAS (2019)
37. Kurpiewski, D., Pazderski, W., Jamroga, W., Kim, Y.: STV+Reductions: Towards practical verification of strategic ability using model reductions. In: Proceedings of AAMAS. pp. 1770–1772. ACM (2021)
38. Lomuscio, A., Qu, H., Raimondi, F.: MCMAS: An open-source model checker for the verification of multi-agent systems. International Journal on Software Tools for Technology Transfer **19**(1), 9–30 (2017). `https://doi.org/10.1007/s10009-015-0378-x`
39. Lomuscio, A., Qu, H., Raimondi, F.: Mcmas: an open-source model checker for the verification of multi-agent systems. International Journal on Software Tools for Technology Transfer **19**(1), 9–30 (2017)

40. Martimiano, T., Santos, E.D., Olembo, M., Martina, J.: Ceremony analysis meets verifiable voting: Individual verifiability in Helios. In: SECURWARE (2015)
41. McMillan, K.: Symbolic Model Checking: An Approach to the State Explosion Problem. Kluwer Academic Publishers (1993)
42. McMillan, K.: Applying SAT methods in unbounded symbolic model checking. In: Proceedings of Computer Aided Verification (CAV). Lecture Notes in Computer Science, vol. 2404, pp. 250–264 (2002)
43. Meier, S., Schmidt, B., Cremers, C., Basin, D.: The tamarin prover for the symbolic analysis of security protocols. In: International conference on computer aided verification. pp. 696–701. Springer (2013)
44. Meng, B.: A critical review of receipt-freeness and coercion-resistance. Information Technology Journal **8**(7), 934–964 (2009)
45. National Electoral Commission [Polish: Państwowa Komisja Wyborcza]: Presidential election 2020. [Polish: Wybory Prezydenta Rzeczypospolitej Polskiej 2020 r.]. `https://prezydent20200628.pkw.gov.pl/prezydent20200628/pl` (2020)
46. Pattinson, D., Schürmann, C.: Vote counting as mathematical proof. In: Advances in Artificial Intelligence, Proceedings of AI. Lecture Notes in Computer Science, vol. 9457, pp. 464–475. Springer (2015). `https://doi.org/10.1007/978-3-319-26350-2\_41`
47. Peled, D.A.: All from one, one for all: on model checking using representatives. In: Courcoubetis, C. (ed.) Proceedings of CAV. Lecture Notes in Computer Science, vol. 697, pp. 409–423. Springer (1993). `https://doi.org/10.1007/3-540-56922-7\_34`
48. of the Republic of Poland, S.: Internet Legal Acts System [Polish: Internetowy System Aktów Prawnych]. `https://isap.sejm.gov.pl/isap.nsf/search.xsp?status=O&kw=wybory` (2022)
49. Sempreboni, D., Vigano, L.: X-men: A mutation-based approach for the formal analysis of security ceremonies. In: 2020 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 87–104. IEEE (2020)
50. Shoham, Y., Leyton-Brown, K.: Multiagent Systems - Algorithmic, Game-Theoretic, and Logical Foundations. Cambridge University Press (2009)
51. Skubiszewski, M.: Electoral Observatory to the President of the NEC: Incorrectly printed ballots abroad - need to address the problem [Polish: Obserwatorium Wyborcze do Przewodniczącego PKW: Nieprawidłowo wydrukowane karty do głosowania za granicą - konieczność rozwiązania problemu], `https://monitorkonstytucyjny.eu/archiwa/14355`
52. Spotted-Lublin: Election 2020 ballot papers without red DEC seal [Polish: Wybory 2020. Karty do głosowania bez czerwonej pieczęci obwodowej komisji wyborczej]. `https://spottedlublin.pl/wybory-2020-karty-do-glosowania-bez-czerwonej-pieczeci-obwodowej-komisji-wyborczej/`
53. Tabatabaei, M., Jamroga, W., Ryan, P.Y.A.: 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. pp. 1:1–1:8. ACM (2016). `https://doi.org/10.1145/2970030.2970039`