Formally Verified Invariants of Vote Counting Schemes *

Florrie Verity
The Australian National University

Dirk Pattinson
The Australian National University

ABSTRACT

The correctness of ballot counting in electronically held elections is a cornerstone for establishing trust in the final result. Vote counting protocols in particular can be formally specified by as systems of rules, where each rule application represents the effect of a single action in the tallying process that progresses the count. We show that this way of formalising vote counting protocols is also particularly suitable for (formally) establishing properties of tallying schemes. The key notion is that of an invariant: properties that transfer from premiss to conclusion of all vote counting rules. We show that the rule-based formulation of tallying schemes allows us to give transparent formal proofs of properties of the respective scheme with relative ease. As our proofs are based on the specification of vote counting protocols, rather than a program that implements them, we are guaranteed that the property holds for every possible specification-confirming implementation of the respective protocol. This in particular includes the vote counting programs that are automatically extracted from the specification. We demonstrate this point by means of two examples: the monotonicity criterion for majority (first-past-the-post) voting, and the majority criterion for a simple version of single transferable vote.

1. INTRODUCTION

Our trust in traditional paper-based elections largely derives from the fact that all stages of the process are witnessed by scrutineers, variously members of the general public or delegates nominated by stakeholders that ensure, and testify to, proper procedures being followed. In an electronic voting context [14], this is replicated by the concept of *end-to-end verifiability* and *universal verifiability*. End-to-end verifiable systems allow the voters to verify that their vote has been correctly recorded, not been tampered with in (elecctronic) transit to the counting station and has been correctly included in the electronic ballot box. A voting system is universally verifiable if any member of the general public has means to determine whether all ballots cast have been counted correctly. End-to-end verifiability is usually achieved by complex cryptographic methods and generate receipts that voters can use to verify integrity of their ballot, without revealing the ballot content. None of the current voting systems in existence [16, 4, 11, 5] offers universal verifiability of the tallying in elections that rely on complex, preferential voting schemes such as single transferable vote used in the majority of elections in Australia. That is, there is no way for voters or political parties to *verify* the correctness of the count.

This is mediated, for example by the Australian Electoral Commission, by publishing both the ballots cast, and scrutiny sheets that detail the proceeds of the count, on the web so that individuals (or political parties) can run their own vote-counting software on the published data. Our experience indicates that while this is likely to generate the same election result, it is nearly impossible to generate tallies that perfectly match those published on the web. The reason for this is one of under-specification: the data published by the electoral commission is noisy in the sense that some ballots assign the same preference to more than one candidate, or omit certain preferences altogether and the commission does not specify how votes of this kind are to be accounted for. More generally, the specification of the vote counting scheme is usually given in the form of a legal document which is not always unambiguous, specifically concerning rare corner cases.

The correctness of the implementation of vote counting schemes has therefore been subject to a number of approaches based on formal methods. One generally distinguishes between *heavy weight* and *light weight* approaches, where the former involve the use of interactive theorem provers to establish mathematical correctness, and the latter rely on automated or semi-automated techniques. In the domain of electronic voting, it has been recognised that light-weight approaches are not practical for the task at hand [6, 2] and heavy-weight methods, i.e. the use of interactive theorem provers, are necessary to give the desired correctness guarantees. In the context of formal verification, the styles of specifying vote counting schemes vary considerably, and usually some variant of type theory or higher-order logic is used to express the desired properties, see e.g. [7]. In contrast to *directly* encoding vote counting schemes, it has been advocated in [15] that the best way of making the vote counting scheme precise is to give a *formal specification* by means of rules. The rule-based specification of voting protocols

*Coq sources that accompany this paper are available from users.cecs.anu.edu.au/~dpattinson/Software/.

© 2016 ACM. ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

is centred around *states* of the counting process that may be thought of as photographic snapshots of a (n imagined) room where counting takes place. More abstractly, a state records all information that are relevant for the counting process at a given point in time. A *rule* then specifies how this state may be manipulated by an (imagined) official progressing the count. An example of such an action would be "take a ballot from the pile of uncounted votes, update the tally of the candidate named as first preference, and put the ballot paper onto the pile of first-preference votes for this candidate". As such, the description of the state invariably contains the running tally, a list of uncounted ballot papers, and, for every candidate, a list of votes counted in their favour. The count then progresses, in analogy with the paper-based process, by applying rules that (correctly) manipulate the state of the count, until winners can be declared. As a consequence, we can understand a (formal) *specification* of a voting scheme as a set of rules that manipulate the state of the count. It has been argued in [15] that this way of specifying voting schemes is both natural and leads to a receipt (the sequence of counting rules leading to the election outcome) that is both precise and can be checked by correctness, i.e. provides a universally verifiable receipt of the correctness of the count.

In this paper, we argue that the rule-based approach to the specification of voting protocols is also well-suited to establishing *properties* of voting protocols specified in this way, and so provides an orthogonal check on the correctness of the specification. We demonstrate this by re-visiting two protocols, simple plurality voting or first-past-the-post (FPTP) and a simple version of single transferable vote (STV), both described in detail in Sections 2.1 and 2.2, respectively. In plurality voting, every voter votes for precisely one candidate, and the candidate with the highest tally wins. It is known in the literature that this procedure satisfies the monotonicity criterion:

> If a candidate wins an election, they will still win the election if any number of ballot papers have been changed in their favour.

Here, by changing a ballot paper in the favour of a candidate we mean replacing the ballot paper by a vote for this candidate.

Single transferable vote elections, by contrast, typically have more than one winner and are used to elect a set of representatives in a single poll. Ballots are not votes for a single candidate, but are a list of candidates, ordered by voter preference. Votes are counted by successively eliminating candidates, and transferring votes to the next preference on the ballot paper, until candidates have reached a given quota. Here, the property that we formally establish is majority:

> If a candidate is listed as first preference on more than 50% of the ballot papers, (s)he will be declared a winner.

Our technical contribution are formal proofs [10] obtained using an interactive theorem prover (we use Coq [3]) of the monotonicity property of FPTP, and the majority criterion for STV. These proofs are obtained by isolating suitable invariants, that is, mathematical relations that hold at every stage of the count, and will ultimately imply the desired property once counting terminates. We emphasise that we do not prove these properties on the basis of having an *implementation* of a concrete counting scheme (as done e.g. in [9] for plurality voting) but instead are basing our proofs on a (formal) *specification* of the scheme. As a consequence, every implementation that (provably) conforms to the specification, including in particular those synthesised from the specification itself, can be guaranteed to enjoy this property. In other words: the *implementation* of vote counting according to the FPTP and STV schemes that we have synthesised from the specification in [15] are *automatically guaranteed* to satisfy monotonicity (resp. majority) criterion.

Formal proofs that involve properties of formal specifications like the two criteria discussed above also serve a function beyond establishing that the criteria in fact hold: they provide a basic sanity check of the formal specification. Just as for programs, humans invariably introduce errors into specifications, rendering the fact that a program implements a specification virtually useless. Giving formal proofs that the specification satisfies the expected properties thus increases our trust in the correctness of the specification as such. In an area such as electronic voting where we would demand that programs conform to formal specification, we see the process of validating specifications as particularly important.

On a conceptual level, we demonstrate that rule-based specifications can not only capture the specifics of voting protocols in a natural way, but are moreover well-suited to their analysis. By giving *loose* specifications of the protocols under scrutiny we achieve coverage of more than one variant (or implementation) of the scheme. This introduces a very convenient layer of abstraction, as any implementation of a voting protocol would reasonably be expected to satisfy its specification, and our proofs entail that any specification-conforming implementation will satisfy the criteria under consideration. That is, there is no need to re-establish any criteria that have been established on the level of specification, as all implementations (in particular including the one that has been synthesised from the specification) are automatically guaranteed to conform. Moreover, by giving formal proofs that are checked by an interactive prover, we obtain ultimate assurance of the correctness of the properties claimed.

In the medium term, we believe that our work can lead to the adoption of possibly more complex but fairer voting schemes where the complexity is tamed by giving formal proofs, both of desirable properties and correctness of their execution of the voting scheme itself.

Related Work. Properties of vote counting schemes are usually established by means of pen-and-paper proofs, see e.g. [12], and there is very little work on formalising properties of tallying schemes in a theorem prover. One notable exception is [9] where the monotonicity criterion is formally verified for an *implementation* of plurality counting. Our work, in contrast, establishes monotonicity for *all* implementations that conform to the specification that we give. The formalisation of the majority criterion is, to the best of our knowledge, new. We have already mentioned other approaches to the formal verification of vote counting schemes, and our work is based on the precursor paper [15]. Our formulation of Single Transferable Vote is taken from that paper and is the same as that of [8] where linear logic was used to specify STV in the Twelf logical framework [17].

In contrast to our formulation in the Coq theorem prover, the way in which STV is embedded in Twelf does not allow reasoning about meta-properties of the formalisation.

2. VOTE COUNTING PROTOCOLS AS RULE BASED SYSTEMS

We recall the formalisation of voting protocols as rule-based systems [15]. This formalisation tries to mirror hand-counting of votes as closely as possible, and is based on two basic building blocks:

1. *States* of vote counting are the electronic equivalent of the vote counting process at any particularly given point in time. One way of thinking about states is as representing snapshots of the room in which votes are hand-counted, and states variously represent combinations of (piles of) uncounted votes, stacks of votes counted in favour of different candidates, a running tally etc.

2. Vote counting *rules* describe the actions that allow us to progress one state to another according to the counting scheme. The simplest example is counting of a single vote: a ballot paper is removed form the list of uncounted votes, placed on the stack of votes counted in favour of the named candidate, and the tally is updated. That is, every rule describes an action that progresses the count.

For concretely given protocols, we distinguish between *final* states that represent the outcome of vote counting (usually a winner or set of winners) and *intermediate* states, i.e. snapshots of the counting process where rules still need to be applied before a winner can be determined, and *initial* states (in which vote counting commences). A sequence of correctly applied rules that leads from an initial to a final state can then be regarded as evidence of the correct application of the voting protocol. We argue in [15] that this allows for universal verifiability of the vote *counting* process and minimises the gap between a mathematical formalisation and legislative text. Here, we argue that it is also very suitable to formally establish properties of voting protocols in a theorem prover, the validity of which gives additional assurance of the correctness of the specification. We give two main examples of voting rules.

2.1 First-Past-The-Post (Plurality) Voting

Given a set C of candidates, every ballot is a vote for precisely one candidate, and we can therefore identify a ballot with the candidate being voted for. First past the post, or plurality voting takes a list of ballots (candidates) to produce a tally, and a candidate with the highest tally is then declared winner. We do not stipulate how this candidate is determined in case of a tie. Schematically, vote counting proceeds as follows:

1. Given a set b of ballots, compute the tally $t(c)$ to be the number of voters that have voted for candidate c.

2. A candidate c is a winner, if no other candidate has received more votes.

Remark 1. Despite the fact that elections counted according to this scheme typically yield a single winner, our formulation does not specifically mandate this. This is reflected in the second point above where we specifically speak about *a* winner, rather than *the* winner. Any implementation would have to impose tie-breaking rules that we do not want to pre-empty in the specification. As a consequence, the monotonicity property established here will hold for *any* implementation no matter how ties are broken.

For the purpose of our rule-based formulation, the states of FPTP counting are either

- intermediate states that are composed of a list of uncounted ballots $us \in \mathsf{List}(C)$ and a running tally $t : C \to \mathbb{N}$, written as $\mathsf{state}(us, t)$, or
- final states that declare that a candidate c is a winner, written as $\mathsf{winner}(c)$.

The counting process itself is fully described by just two rules. The first rule

$$\frac{\mathsf{state}(u :: us, t)}{\mathsf{state}(us, t[u \mapsto t(u) + 1])}$$

describes that given a list $u :: us$ of uncounted votes that begins with a single (uncounted) vote u and continues with a list us of (more) uncounted votes, together with tally t, we can progress the count by counting the vote u. Recall that we identify a candidate with a vote for this candidate so that $u \in C$ is in fact a candidate. The given state can be progressed to a state where the remaining uncounted votes are us and the tally has been updated: the notation $t[c \mapsto t(c) + 1]$ indicates the function $t[c \mapsto t(c) + 1](d) = t(d)$ for $d \neq c$ and $t[c \mapsto t(c) + 1](c) = t(c) + 1$. The second rule describes that winners are declared once all votes have been counted, that is, the list of uncounted votes has been reduced to the empty list $[]$. If this is the case, and a candidate c has received at least as many votes as all other candidates, (s)he is declared a winner. In or formulation, the rule takes the following form:

$$\frac{\mathsf{state}([], t)}{\mathsf{winner}(c)} \text{ if } t(c) \geq t(d) \text{ for all } d \in C$$

Initial states are states where the list of uncounted ballots is precisely the list of ballots cast, and the tally records 0 for each candidate. To formalise this, we need access to the initial set of ballots cast. This gives rise to the notion of *judgement*, and a judgement (in FPTP counting) is a state, together with the set of initial ballots. We write

$$b \vdash \mathsf{state}(u, t) \qquad b \vdash \mathsf{winner}(c)$$

for the judgement that $\mathsf{state}(u, t)$ (resp. $\mathsf{winner}(c)$) is a valid intermediate (final) state of vote counting where b is the list of ballots cast. Our initial judgement then becomes

$$\frac{}{b \vdash \mathsf{state}(b, t)} \text{ if } t(c) = 0 \text{ for all } c \in C$$

and the two rules above are just extended with the parameter and thus become

$$\frac{b \vdash \mathsf{state}(u :: us, t)}{b \vdash \mathsf{state}(us, t[u \mapsto t(u) + 1])}$$

$$\frac{b \vdash \mathsf{state}([], t)}{b \vdash \mathsf{winner}(c)} \text{ if } t(c) \geq t(d) \text{ for all } d \in C$$

As a consequence, if the judgement $b \vdash \mathsf{winner}(c)$ can be obtained by applying the above rules, candidate c is one of the possible winners of the election where ballots b have been cast.

2.2 Single Transferable Vote

We describe a proof-of-concept formalisation of a system of single transferable vote, and as for FPTP vote counting, our specification is loose in the sense that it may allow for more than one rule to be applied in any given state of vote counting. It is parameterised by a list b of ballots, the number s of seats to be elected, and the quota q (the number of votes needed to elect a candidate) for the election process. Single transferable vote comes in many variants and we recall the description of STV from [15, 8]:

1. if candidate has enough first preference to meet the quota, (s)he is declared elected. Any surplus votes for this candidate are transferred.
2. if all first preference votes are counted, and the number of seats is (strictly) smaller than the number of candidates that are either (still) continuing or elected, a candidate with the least number of first preference votes is eliminated, and her votes are transferred.
3. if a vote is transferred, it is assigned to the next candidate (in preference order) on the ballot.
4. vote counting finishes if either the number of elected candidates is equal to the number of available seats, or if the number of remaining hopeful candidates plus the number of elected candidates is less than or equal to the number of available seats.

Remark 2. Single transferable vote is used in many jurisdictions, in many variants. The choice of the particular variant of STV described here is therefore necessarily somewhat arbitrary. Our selection is guided by the fact that precisely this formulation of STV has been modelled logically before [8]. The formulation of this particular variant is very loose. In particular, no fixed order of counting ballots is prescribed, and the set of ballots transferred to lower-ranked candidates can depend on the order in which ballots are counted. An implementation of this specification of STV therefore has to fix a particular order. By not working with a particular *implementation* of STV and rather establishing the majority criterion for this particular *specification* of STV, we obtain that the monotonicity criterion is satisfied, no matter what ballot order is chosen by any particular implementation. As regards concrete implementations of STV counting, our approach guarantees that the implementation of STV that was synthesised from the specification in [15] satisfies the majority criterion, as it automatically complies with the specification.

We now describe the rule-based specification of STV, given a set C of candidates. Here, ballots are lists of candidates (in preference order) and we write $\mathsf{ballot} = \mathsf{List}(C)$ to denote individual ballots. When a candidate is eliminated during the count, the ballots counted in their favour have to be transferred to the next preference. As a consequence, we need to remember which ballots were counted for each candidate, as those will be the ballots that are transferred when candidates are eliminated. In addition to this, we need to keep track of which candidates are already marked as elected as votes for elected candidates are automatically transferred to the next listed preference. Finally, we need to remember continuing (still hopeful) candidates. The states of STV counting are therefore of one of the following two forms:

- intermediate states that take the form $\mathsf{state}(u, a, t, h, e)$ where u are the uncounted ballots, $a : C \to \mathsf{List}(\mathsf{ballot})$ is an assignment that records, for each candidate $c \in C$, the list $a(c)$ of ballots counted in c's favour. The function $t : C \to \mathbb{N}$ is the current tally of first preferences, and e, h are the lists of elected and continuing (hopeful) candidates.
- final states that we write as $\mathsf{winners}(w)$ where $w \in \mathsf{List}(C)$ is the list of election winners.

As already remarked at the beginning of the section, STV counts are parametrised by the quota $q \in \mathbb{N}$ of first preference votes that a candidate needs to accumulated to be elected, the number s of seats to be filled, and the set b of ballots cast. A judgement in STV counting then takes one of the following forms

$$(b, q, s) \vdash \mathsf{state}(u, a, t, h, e) \qquad (b, q, s) \vdash \mathsf{winners}(w)$$

where $w \in \mathsf{List}(C)$ is the list of election winners. We read the first judgement as "in an election where ballots b have been cast, the quota is q and s seats are to be filled, $\mathsf{state}(u, a, t, h, e)$ is a valid state of vote counting". The second judgement is read in the same way, but declares the winners. Vote counting starts by considering all votes as uncounted, that is, we begin the stepwise application of rules from a judgement of the form

$$(b, q, s) \vdash \mathsf{state}(b, a, t, [c_1, \ldots, c_n], [])$$

where the first component b is the list of all votes cast, i.e. all votes are uncounted initially, $a(c) = []$ is the null assignment that maps every candidate to the empty list of votes as no votes have yet been counted in the candidate's favour. The penultimate component, the list of continuing candidates, is the list of all candidates $[c_1, \ldots, c_n]$ assuming that $C = \{c_1, \ldots, c_n\}$ and the last component, the list of elected candidates is empty. We give an exemplaric description of three vote counting rule (transfer of votes, election of a candidate, and declaring election winners) and refer to [15] for a full description. In the simple STV protocol under consideration here, votes for candidates that are no longer continuing are transferred to the next preference on the ballot paper. As a rule, this takes the form

$$\frac{(b, q, s) \vdash \mathsf{state}(u, a, t, h, e)}{(b, q, s) \vdash \mathsf{state}(u', a, t, h, e)}$$

and the rule may be applied if $u = u_0{+}{+}[f{::}fs]{+}{+}u_1$ is the list of uncounted votes containing a vote for first preference f and (remaining) preferences fs (we write $++$ for the concatenation of lists), the candidate f at first position is no longer continuing ($f \notin h$) and the new list u' of uncounted votes has the form $u_1{+}{+}[fs]{+}{+}u_2$, i.e. the first preference has been removed from the ballot, and all the other data doesn't change.

The rule for electing a candidate takes the form

$$\frac{(b, q, s) \vdash \mathsf{state}(u, a, t, h, e)}{(b, q, s) \vdash \mathsf{state}(u, a, t, h', e')}$$

and the rule may be applied if there is a candidate $c \in h$ so that the following conditions are satisfied:

- $c \in h$, i.e. the candidate c is continuing (still hopeful)

- $t(c) = q$, i.e. c has reached enough votes to be elected
- h' is the list h with c removed
- e' is the list e with c inserted

Note that this just changes the set of elected and continuing candidates, and all other data is unaffected.

The count in single transferable vote terminates as soon as either enough candidates have been elected, or the number of elected and continuing candidates are less than or equal to the number of seats. The first rule takes the form

$$\frac{(b, q, s) \vdash \mathsf{state}(u, a, t, h, e)}{(b, q, s) \vdash \mathsf{winners}(e)} \text{ if } |e| = s$$

i.e. once the number $|e|$ of elected candidates has reached the number of seats, the candidates marked as elected are declared the winners of the election.

Remark 3. There are many different ways to concretely formulate the voting protocols here. While we represent uncounted votes as lists (of ballot papers), a formalisation that represents them as multisets would be equally viable. The choices made here are dictated by ease of implementation in a theorem prover that we describe in the next section.

Both for first past the post and single transferable vote, our specification is *loose* in the sense that the outcome is not fully determined by the specification. For FPTP, this is a consequence of the rule that determines the winner: if there's more than one candidate with maximal number of votes, both can be chosen to be the (only) winner. In STV, the same applies when candidates are eliminated (any candidate with least number of first preferences can be eliminated), and also precisely which votes are transferred to second preferences depends on the order in which votes are counted. While this may be undesirable for a vote counting protocol *per se* it is useful for the purpose of the present paper in that we show that *all* implementations of vote counting (that may impose additional restrictions on how / in which order counting rules may be applied) satisfy the properties we establish here.

3. FORMALISATION

Both voting protocols have been formalised in the Coq theorem prover [3]. The formalisation is a direct representation of the rules using a dependent inductive type. Mathematically, we think of a sequence of correctly applied vote counting rules that ends in declaring a winner as evidence for correct counting. Generalising this idea to also include intermediate steps, this is precisely what we achieve using (depenent) inductive types.

Both formulations are parameterised (by the list of ballots cast, for FPTP, and by the list of ballots, the number of seats and the quota, for STV). Ignoring the parameters that remain fixed, the inductive type takes the form

$$\mathsf{Pf} : \mathsf{Judgement} \to \mathsf{Type}$$

so that, for a judgement j, $\mathsf{Pf}(j)$ is the type whose elements are proofs of the fact that j is a judgement that is derivable by means of the counting rules. That is, elements $\mathsf{Pf}(j)$ represent *evidence* of the fact that j is a judgement that has been derived correctly. This evidence is constructed inductively:

- no evidence is needed to construct an initial state of counting
- If we have evidence of the fact that a judgement j represents a correct state of counting (given by an element of $\mathsf{Pf}(j)$), and j/j' is a counting whose premiss is j and whose conclusion is a judgement j', then we can construct evidence of j' being a correct state of counting by extending the evidence for j's correctness with the rule application that leads from j to j'.

As a consequence, evidence for the correctness of a judgement j is a list of rule applications, starting with an initial judgement, and ending in the judgement j. The inductive nature of the type allows us to prescribe how evidence may be *constructed*. Accordingly, every counting rule corresponds to one constructor of the inductive type that also embodies the conditions that need to be met for the constructor to be applied.

3.1 First-Past-The-Post Voting

The heart of the formalisation of FPTP is an inductive type as explained above. In the syntax of Coq, we obtain the specification given in Table 1. Here, each rule is represented as a constructor of the (dependent) inductive type `Pf`, and the side conditions of a rule are represented as logical constraints that need to be satisfied for the constructor to be applicable.
The Coq-code above uses several auxiliary definitions, the most important one that of a type Node that represents a state of vote counting, and is given by

```
(* intermediate and final states in FPTP counting *)
Inductive Node  :=
  winner : cand -> Node
| state  : (list cand) * (cand -> nat) -> Node.
```

so that a node is either a final state that stipulates the election winner (via the first constructor), or an intermediate state that records the uncounted ballots and the running tally (via the second constructor). Moreover, `nty` is the null tally that records zero votes for every candidate, $[\mathtt{x}_1, \ldots, \mathtt{x}_n]$ is Coq-notation for the list comprising elements $x_1, \ldots, x_n$ (in particular [] is the empty list) and ++ is list concatenation. The main aspect to note is that the rules, presented in the previous section, can be transcribed into Coq's syntax almost verbatim.

3.2 Single Transferable Vote

As for FPTP, we can represent the rule based formulation of the voting protocol directly within a theorem prover. As per the rule-based description of STV in the previous section, intermediate states of vote counting are more complex (more data needs to be represented) and we now have a list of winners rather than a single winner. The type `Node` of (final and intermediate) states of STV counting then takes the following form

```
(* intermediate and final states in stv counting *)
Inductive Node :=
   state:                        (** intermed. states **)
       list ballot               (* uncounted votes *)
     * (cand -> list ballot)     (* assignment *)
     * (cand -> nat)             (* tally *)
     * (list cand)               (* continuing *)
```

```
Inductive Pf (b: list cand) : Node -> Type :=
 ax : forall u t,                              (** start counting **)
 u = b ->                                      (* all ballots are uncounted *)
 t = nty ->                                    (* and the tally is nill *)
 Pf b  (state (u, t))                          (* start counting with null tally *)
| c1 : forall u0 c u1 nu t nt,                 (** count one vote **)
 Pf b  (state  (u0 ++[c]++u1, t)) ->           (* have an uncounted vote for c *)
 inc c t nt ->                                 (* tally increments c's votes by one *)
 nu = u0++u1 ->                                (* vote deleted from uncounted votes *)
 Pf b (state (nu, nt))                         (* continue with new tally *)
| dw : forall c  t,                            (** declare winner **)
 Pf b  (state ([], t)) ->                      (* if all votes have been counted *)
 (forall d : cand, (t d <= t c)) ->            (* and all cands have fewer votes than c *)
 Pf b  (winner c)                              (* then c may be declared the winner *).
```

Table 1: Formal Specification of Plurality Voting

```
  * (list cand)            (* elected cands *)
  -> Node
| winners:                 (** final state **)
  list cand -> Node.       (* election winners *)
```

which is again a direct transcription from the mathematical representation in the previous section. As for FPTP, we formalise the notion of j being a correct state of STV counting by means of a dependent inductive type, where every counting rule becomes a constructor. Figure 2 presents the constructors that correspond to the transfer of a vote to the next preference, and the election of a single candidate (we have elided all other rules).

In the formulation of the transfer rule (`tv`, for transfer vote), `rep (f::fs) fs u nu` is a formula that expresses that the ballot `f::fs`, i.e. a first-preference vote for `f` with remaining preferenes `fs` is replaced by the vote just containing `fs`, so effectively crossing the first preference `f` off the ballot paper. In the formulation of the elect rule (`el`), `eqe x l1 l2` (for equal except) is a predicate that asserts that lists `l1` and `l2` are equal, except that `l1` does not contain `x` but `l2` does. That is, `eqe c nh h` asserts that `c` is a an element of the list `h` of continuing (hopeful) candidates that is no longer present in the update list of (new hopeful) continuing candidates `nh`. The full, formal specification of STV can be obtained from the Coq sources that accompany this paper. As for FPTP, the main point to notice is that the mechanism of inductive types in Coq allows us to represent the rule-based formulation of single transferable vote in a very direct, straight forward way.

4. THE MONOTONICITY CRITERION

The monotonicity criterion [13], usually formulated for preferential voting systems, simplifies greatly when applied to first-past-the-post elections and states that "If a candidate c wins an election given a set b of ballots, and if the set b' of ballots is obtained from b by changing some of the votes to votes for c, then c will (still) win when counting the ballots b'".

It is known that monotonicity holds for FPTP elections, by a mathematically trivial argument. However, a certain level of care must be exercised in a precise mathematical formulation. In case of a tie between candidate c and d resolved in favour of c, changing *zero* votes to votes for c, the tie may be resolved in favour of d. We therefore formulate monotonicity more precisely as follows: If there is a way of counting ballots b that produces c as winner, and if b' are obtained from by changing some (zero or more) votes to votes for c, then ballots b' *can* be counted in such a way that c wins.

Our formulation for monotonicity of FPTP vote counting is based on a notion of *evidence* for the fact that the set b' of ballots arise from b by changing some of the votes into votes for (the winning candidate) c. This notion of evidence is readily formulated as an inductive predicate $\mathsf{better}_c(b, b')$ that captures that the set b' of ballots is "better" than the set b of ballots from the perspective of candidate c, as some of the ballots have been changed into votes for c. This predicate is formulated inductively by means of the rules

$$\frac{}{\mathsf{better}_c([],[])} \qquad \frac{\mathsf{better}_c(xs, ys)}{\mathsf{better}_c(x{::}xs, y{::}ys)}(x = c \text{ or } x = y)$$

where [] is the empty list of ballots and $x{::}xs$ is a list (of ballots) with head x and tail xs. The first rule deals with the empty set of ballots, and says that changing (zero or more) votes in the empty ballot results in a set of ballots (necessarily empty) that is at least as good for winning the election from c's prespective. The second rule assumes that the set of ballots xs is at least as good as ys from c's perspective, and also that the ballot x is either the same as y, or has been changed to c. Under these conditions, we have that the list of ballots consisting of x and xs is at least as good as the ballots comprising y and ys. Again, this idea is readily transcribed into an inductive definition in Coq:

```
Inductive btr (c: cand):
  list cand -> list cand -> Type :=
| both_emp: btr c nil nil
| both_cons: forall x xs y ys,
  btr c xs ys ->
  y = x \/ y = c ->
  btr c (x::xs) (y::ys).
```

Our main theorem, formalised in Coq, then states that if c can win a count that is conducted according to the FPTP rules given ballots b, and b' is a set of ballots that arises by changing some votes to votes for c, then c *can* still win the

```
Inductive Pf (b: list ballot) (q: nat) (s: nat) : Node -> Type :=
   [ ... ]
| tv : forall u nu a t h e f fs,                (** transfer vote **)
  Pf b q s (state (u, a, t, h, e)) ->           (* if we are counting votes *)
  ~(In f h) ->                                  (* and f no longer in the running *)
  rep (f::fs) fs u nu ->                        (* and the uncounted votes are updated *)
                                                (* by deleting first pref f from a vote *)
  Pf b q s (state (nu, a, t, h, e))             (* continue with updated set of votes *)
  [ ... ]
| el : forall u a t h nh e ne c,                (** elect a candidate **)
  Pf b q s (state (u, a, t, h, e)) ->           (* if we are counting votes *)
  In c h ->                                     (* and c is a hopeful *)
  t(c) = q ->                                   (* and c has enough votes *)
  length e < s ->                               (* and there are still unfilled seats *)
  eqe c nh h ->                                 (* and c is no longer continuing *)
  eqe c e ne ->                                 (* and added to the
elected cands *)
  Pf b q s (state (u, a, t, nh, ne))            (* then proceed with updated  data *)
[ ... ]
| ew : forall w u a t h e,                      (** elected win **)
  Pf b q s (state (u, a, t, h, e)) ->           (* if at any time *)
  length e = s ->                               (* we have as many elected candidates as seats *)
  w = e ->                                      (* and winners are precisely the elected *)
  Pf b q s (winners w).                         (* they are declared winners *)
```

Table 2: Part of the formal Specification of STV

count. In Coq, it takes the following form, where `Pf` refers to the specification of plurality voting in Table 1:

```
Theorem mon_T: forall w b nb, btr w b nb ->
  Pf b (winner w) -> Pf nb (winner w).
```

The functional reading of the theorem is that under the condition that some ballots in `nb` have been changed to votes for c, we can construct an FPTP-proof that `w` is the election winner given ballots `nb` from an FPTP-proof that `w` is the election winner given ballots `b`. In other words, from evidence that `w` is the winner given ballots `b`, we can construct evidence that `w` is the winner given ballots `nb` provided that `nb` is better from the perspective of `w`.

The formal proof proceeds by induction on the sequence of rule applications that lead to the judgement `Pf b (winner w)` and maintains an invariant throughout the count. In words, whenever we can reach a state of the count, starting from ballots b with u votes uncounted and tally t, we can reach a state of the count starting from ballots nb with nu ballots uncounted, tally nt such that nu is "better" for c and the (new) tally nt of a candidate other than c is at most as high as the tally t for the (same) candidate.

In Coq, this is represented by the following lemma, where `A * B` is the cartesian product of types `A` and `B` that we use in place of logical conjunction (for technical reasons as this allows us to extract a function that converts evidence):

```
Lemma mon_st: forall c b nb u t,
  btr c b nb ->
  Pf b (state (u, t)) ->
    existsT nu nt,
      (btr c u nu) *
      Pf nb (state (nu, nt)) *
      (t c <= nt c) *
      (forall d, d <> c -> nt d <= t d).
```

which is proved by induction on the given evidence `Pf b (state u t)` using the information that `nb` is a "better" set of ballots from the perspective of candidate `c`. We refer to the Coq sources for a full proof of this lemma, and the main theorem.

5. THE MAJORITY CRITERION

Our second case study concerns a formal proof of the majority criterion [13]. Informally, the majority criterion for single transferable vote states that if at least 50% of the ballots contain a first preference for candidate c then c will be among the winners of the election.

In our – intentionally loose – formulation of single transferable vote, we need to analyse the formulation more carefully, as it doesn't make any assumptions about the quota or the number of seats. It is evident that we need to require that the overall number of seats is strictly positive, as otherwise only the empty set of candidates can possibly be elected. Second, we need to balance the quota against the total number of ballots. For a simple example, consider a quota of 1 with one seat to be filled. Thus, the first candidate d for whom a first preference vote is counted, will be elected – even if all other first preference votes go to c and $d \neq c$. This is a consequence of our intentionally loose formulation of STV that is not intended to specify a single implementation, but instead to subsume as many implementations as possible. We therefore impose the relation

$$s \cdot q \geq \frac{1}{2}|b| \tag{1}$$

where $|b|$ is the number of ballots cast, s is the number of seats and q is the quota. In particular, this restriction is satisfied by the Droop quota ($\lceil \cdot \rceil$ indicates the smallest

integer greater than its argument),

$$q = \lceil \frac{|b|}{s+1} \rceil$$

the most widely used quota in STV elections, and the (smaller) Hare quota.

Our main theorem relies on an auxiliary function that – given a list b of preference-ordered ballots and a candidate c, computes the number $\mathsf{cfp}(b, c)$ of first-preference votes for c (read as "count first preferences"). Formulated in Coq, our main theorem takes the following form

```
Theorem maj: forall b q s w c,
  s >= 1 ->
  2 * s * q >= length b ->
  2 * count_fp c b > (length b) ->
  Pf b q s (winners w) ->
  In c w.
```

and should be read as "if there's at least one seat to be filled, the number of ballots is at most twice the product of seats and quota and c receives more than half first-preference votes, then any set w of election winners contains c". As for FPTP, we interpret `Pf b q s (winners w)` as *evidence* that `w` are the winners of an election with quota q, ballots b and s seats to fill.

As for FPTP, we establish the theorem using an invariant, i.e. a property that holds at all stages of the count, and is strong enough to imply the result that we are about to prove. Here, the invariant involves the current tally, the total number of first preferences amongst the (still) uncounted ballots, and the overall number of votes received by the elected and continuing candidates.

More formally our invariant states that at all stages of the count, either c is already an elected candidate, or all of the following are true:

- c is a continuing candidate ($c \in h$)
- the tally of c, together with first preference votes for c in the uncounted ballots is larger than half of the number of total ballots cast:

$$2 \cdot (t(c) + \mathsf{cfp}(c, u)) > |b| \tag{2}$$

- the sum of the tallies of all continuing and elected candidates, together with the number of uncounted ballots, is below the number of total ballots:

$$\mathsf{ctl}(t, h{+}{+}e)) + |u| \leq |b| \tag{3}$$

Here $\mathsf{ctl}(t, l) = \sum_{c \in l} t(c)$ is the sum of the tallies of candidates in list l, and $++$ is the concatenation of lists as before.

The invariant above guarantees the following: if the candidate c who got 50% of first preference votes isn't elected yet, then at least c's tally, together with yet uncounted first preference votes for c is still larger than 50% of the number of ballots. The second inequality essentially stipulates that we're still in the first round of counting and no votes have been re-distributed yet: re-distributed votes would increase the sum of the tallies beyond the number of votes already counted.

We show, by means of a formal proof in Coq, that this invariant is maintained at all stages of vote counting according to our formalisation of STV. For the rule of transferring a single vote that we have discussed earlier, this can be seen as follows. If the invariant holds immediately before this rule is applied, then either

- c is already elected. In this case, c will also be an elected candidate *after* the rule has been applied and the invariant is valid in the post-state, or
- c is not (yet) elected, but a continuing candidate, and the two inequalities above hold for the state immediately prior to applying the transfer rule. As the rule transfers a preference only in case the candidate in question is not continuing, we know that this does not amount to transferring a first preference for c to the next candidate. As a consequence, both inequalities remain valid.

The situation for the rule that elects a candidate is slightly different, but not more complicated. Again, if the invariant holds immediately prior to applying this rule, then

- candidate c is already elected, and will also be an element of the list of elected candidates *after* the rule has been applied, and the invariant holds in the post-state, or
- candidate c is the new candidate that is elected by applying this rule. In particular, c will be an element of the set of elected candidates in the post-state, and thus the invariant is re-established, or finally
- the rule elects an candidate d distinct from c. Then c is still a continuing candidate (as the invariant holds prior to rule application) and the two numerical inequalities still hold as the data does not change (for the second inequality, note that the union of elected and hopeful candidates remains the same).

In this way, the invariant propagates to the last state before the list of winners is declared. At this point, we know that either c is among the elected candidates (and will hence be one of the winners) or the second alternative of the invariant obtains, which is in contradiction with the side condition of declaring winners as rule applied next so that necessarily, c is one of the elected candidates. We make this explicit for the case where winners are declared using the third rule discussed in Section 2.2, i.e. the number of candidates marked elected equals the number of seats, and the set of elected candidates are declared to be the winners. As the invariant propagates to the last valid state before this rule was applied, we know that we are in one of two cases:

- c is an elected candidate. But this immediately gives that c is an element of the set of winners.
- c is a continuing candidate and Equations (2) and (3) hold. But this is impossible due to the restriction on quota (Equation 1), as we would obtain the following contradiction:

$$\begin{aligned} |b| &\geq \mathsf{ctl}(t, h{+}{+}e) + |u| && (3)\\ &= \mathsf{ctl}(t, h) + q \cdot s + |u| && (\text{quota})\\ &\geq t(c) + q \cdot s + \mathsf{cfp}(c, u) && (\text{cfp})\\ &> |b|/2 + |b|/2 = b && (1), (2) \end{aligned}$$

so that necessarily the first case applies, and c is an element of the set of winners.

In the above, (quota) is the (easily verified) assertion that all candidates marked as elected have received precisely q votes, and (cfp) says that the number of first preferences for c in the uncounted votes is at most as large as the number $|u|$ of uncounted votes.

6. DISCUSSION

There is a great number of criteria against which vote counting schemes are (mathematically) evaluated. They include monotonicity and majority discussed here, but also Independence of Irrelevant Alternatives [1], Consistency, Later-No-Harm, Condorcet, Condorcet-Loser, Independence of Clones, Reversal Symmetry etc. The fact that a concretely given voting system satisfies a given property is usually established using pen-and-paper proofs, see for example Hill *et. al.* [12]. In contrast, the work presented in this paper comprises the first formal proof (in the sense of [10]) of two properties of voting systems (monotonicity and majority) against a formal specification of the respective system.

From the perspective of formal proofs, or interactive theorem proving, our results are certainly not surprising, and from a purely mathematical perspective, they are not new. Rather, our contribution lies in the fact that (a) we demonstrate that the method of vote counting as proof, outlined in [15] presents not only a convenient way of specifying voting protocols, but also allows to verify properties of specifications in this style, (b) that specifications provide an abstraction layer that guarantee that every implementation of the specification will automatically satisfy all properties that have been established on the specification level, and (c) that proofs at specification level provide additional validation of the specification itself. We comment on all three aspects in turn.

Formal Verification of Voting System Properties. Establishing properties of voting systems is essentially a mathematical task, and proving properties of voting systems in a theorem prover takes these mathematical proofs to an entirely new level of rigour: our proofs are machine-checked by an interactive proof assistant (Coq) the kernel of which consists of just a few hundred lines of trusted, well tested code that has been used on a very large number of (formal) proofs. As such, formal proofs greatly increase our confidence in the correctness of the result, in particular if applied to complex voting systems like the many variations of STV that often differ only in minute detail.

While the main technical contribution of this paper are formal proofs of properties of FPTP and STV voting schemes, our conceptual contribution is to demonstrate that the formalisation of voting schemes as *rules* provides us with a powerful proof method that allows us to formally establish properties of voting protocols. We have demonstrated this using two examples, and in both cases, were able to isolate invariants, i.e. properties that hold at all stages of the count, and are strong enough to imply the property under scrutiny.

Formal Specification as Intermediate Layer. We are presented with two choices to verify properties of voting schemes:

1. we start with a concrete algorithmic representation of the voting scheme and then establish the given property for this representation
2. we start with a declarative description of the voting scheme and show that every way of counting consistent with this description satisfies the given property

While the first approach only gives us guarantees for *one particular* implementation of vote counting, the second approach that we follow in this paper will give guarantees for *all* implementations that are consistent with the (rule-based) description of the protocol. In particular, this includes the vote-counting algorithms synthesised from the specification [15]. The aspects of covering *all* realisations of the specification stands or falls with the level of prescriptiveness of the specification: the looser the specification, the more implementations will be covered. This is indeed the motivation behind the specification of FPTP and STV analysed in this paper.

Validation of Specifications. As with software development in general, specifications are rarely error-free. Usually, bugs in specifications are discovered when the verification of a program against an erroneous specification fails. For vote counting protocols, the situation is slightly more specific, as the formal specification prescribes the behaviour of the (vote counting) program at a very great level of detail. In the two specifications considered in this paper, every rule can be directly translated to program statements, and a counting program that conforms with a formal specification that is given at this level of detail is usually developed with the specification and the correctness proof in mind, in other words, will be written to be specification-conformant in the first place. This significantly reduces the opportunity to detect errors in the specification itself: these will only be discovered if the specification itself does not provably entail a criterion that it is expected to satisfy. Our approach does describe one method to achieve this, although clearly a larger number of criteria should be validated to bolster our belief in the correctness of specifications even more.

Conclusion and Future Work. We have presented two case studies that substantiate that a specification of voting protocols in terms of rules not only gives rise to provably correct, universally verifiable vote counting [15] but is also well-suited for a formal mathematical analysis in a theorem prover. Our leading motive is that a *state* of a voting scheme represents a snapshot of the room in which votes are counted and represents all the important information pertaining to the count. A *rule* then represents an action that changes this state in accordance with the voting scheme. Rule-based specifications of voting schemes can be loose in the sense that no strict order of rule applications is required, and/or ties can be resolved in more than one way. It remains to be seen whether our approach scales up to (even) more complex voting systems and voting systems not based on single transferable vote, and to what extent proofs of properties of voting systems can be made modular so that small changes in the scheme do not require to re-do proofs of their properties in their entirety.

7. REFERENCES

[1] K. J. Arrow. A difficulty in the concept of social welfare. *Journal of Political Economy*, 58(4):328–346, 1950.

[2] B. Beckert, T. Börmer, R. Goré, M. Kirsten, and T. Meumann. Reasoning about vote counting schemes using light-weight and heavy-weight methods. 2014.

[3] Y. Bertot, P. Castéran, G. Huet, and C. Paulin-Mohring. *Interactive theorem proving and program development : Coq'Art : the calculus of inductive constructions*. Texts in theoretical computer science. Springer, 2004.

[4] D. Chaum. Secret-ballot receipts: True voter-verifiable elections. *IEEE Security & Privacy*, 2(1):38–47, 2004.

[5] D. Chaum, A. Essex, R. Carback, J. Clark, S. Popoveniuc, A. T. Sherman, and P. L. Vora. Scantegrity: End-to-end voter-verifiable optical-scan voting. *IEEE Security & Privacy*, 6(3):40–46, 2008.

[6] D. Cochran. *Formal Specification and Analysis of Danish and Irish Ballot Counting Algorithms*. PhD thesis, 2012.

[7] J. E. Dawson, R. Goré, and T. Meumann. Machine-checked reasoning about complex voting schemes using higher-order logic. In R. Haenni, R. E. Koenig, and D. Wikström, editors, *Proc. E-Vote-ID 2015*, volume 9269 of *Lecture Notes in Computer Science*, pages 142–158. Springer, 2015.

[8] H. DeYoung and C. Schürmann. Linear logical voting protocols. In A. Kiayias and H. Lipmaa, editors, *Proc. VoteID 2011*, volume 7187 of *Lecture Notes in Computer Science*, pages 53–70. Springer, 2012.

[9] R. Goré and T. Meumann. Proving the monotonicity criterion for a plurality vote-counting program as a step towards verified vote-counting. In R. Krimmer and M. Volkamer, editors, *Proc. EVOTE 2014*, pages 1–7. IEEE, 2014.

[10] T. Hales. Formal proof. *Notices of the AMS*, 55:1370–1380, 2008.

[11] Helios. The helios voting system, 2016. http://heliosvoting.org/, accessed June 25, 2016.

[12] I. D. Hill, B. A. Wichmann, and D. R. Hill:1987:AST. Algorithm 123 : Single transferable vote by meekâĂŹs method. *Computer Journal*, 30:277–âĂŞ281, 1987.

[13] D. R. Hill:1987:AST. Properties of preferential election rules. *Voting matters*, 3:8–15, 1994.

[14] D. W. Jones and B. Simons. *Broken Ballots: Will Your Vote Count?* CSLI Publications, 2012.

[15] D. Pattinson and C. Schürmann. Vote counting as mathematical proof. In B. Pfahringer and J. Renz, editors, *Proc. AI 2015*, volume 9457 of *Lecture Notes in Computer Science*, pages 464–475. Springer, 2015.

[16] Pret-A-Voter. The prêt à voter system, 2016. accessed May 25, 2016.

[17] C. Schürmann. The twelf proof assistant. In S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, editors, *Proc. TPHOL 2009*, volume 5674 of *Lecture Notes in Computer Science*, pages 79–83. Springer, 2009.