Modular Formalisation and Verification of STV Algorithms

Milad K. Ghale Dirk Pattinson

Research School of Computer Science, ANU, Canberra

Abstract. We introduce a framework which offers uniform formalisation and verification of some properties for various Single Transferable Voting (STV) algorithms. These algorithms, although different from one another in some ways, have key properties in common which make them STV instances. By abstracting away common features of particular STV schemes, we obtain minimum requirements to accept a given, arbitrary scheme as a legitimate STV instance. We formally prove that any STV scheme which meets the expectations of the above conditions satisfies some mathematical properties, such as termination. We demonstrate extensibility and concision of our framework by modular formalisation of a divergent range of STV algorithms in the theorem prover Coq. Then, by the built-in mechanisms of Coq, for each of the modules, we extract a certifying executable programme into the Haskell programming language. The certificate produced upon each execution, is a visualisation of the trace of the computation carried out to obtain an end result for an election instance. It provides us with an independently checkable proof of tallying correctness to establish count-as-recorded subproperty of the universal verifiability index. Finally we show effectiveness of our approach by evaluating the executables on some real-size elections.

1 Introduction

2 The Generic STV Machine

There are numerous elections which use STV for their tallying method. Despite the apparent differences observable in the particular version of STV which is invoked, there are fundamental data and algorithmic structure common among most of them. By abstracting these underlying features away, we obtain an abstract data structure and minimal constraints which form a generic STV. We prove some mathematical properties, such as termination, about the generic version. In this section, we elaborate on what the components of the generic version are and how they function as a whole. Shortly afterwards, we demonstrate how the properties are established in the theorem prover Coq and then modularly illustrate how they extend to particular STV cases. For this purpose, we describe our generic STV design by employing a pedagogical language from theoretical computer science, namely automata theory and programming semantics.

Moreover, to make our theoretical approach more sensible, we choose a particular STV algorithm which is used as the counting scheme in the ATC state of Australia for lower house elections. We refer to it throughout to make the discussion concrete to the reader.

2.1 The Lower House Australian Capital Territory STV

The ACT government of Australia employs a version of STV for electing the lower house representatives[]. The protocol's clauses are specified as follows.

Step 1. Count the first preference votes for each candidate.
Step 2. Calculate the quota :

$$\text{(total number of valid votes / (number of seats + 1)) + 1.}$$

Step 3. Any candidate with votes equal to or greater than the quota is declared elected.
- If all vacancies have been filled, the election is completed.
- If all vacancies have not been filled, does any candidate have more votes than the quota?
  1. If yes then go to step 4.
  2. If no then go to step 5.

Step 4. Distribute the successful candidate's surplus votes to continuing candidates according to the further preferences shown on the ballot papers by those voters. Calculate each continuing candidate's new total votes, then go back to step 3.

Step 5. If there are more continuing candidates than there are vacancies remaining unfilled, exclude the candidate with the fewest votes and distribute this candidate's votes to continuing candidates according to the further preferences shown by those voters. Calculate each continuing candidate's new total votes then go back to step 3.
Or, if the number of continuing candidates is equal to the number of vacancies remaining, all of those candidates are declared elected and the election is completed.

In a separate section, the protocol further elaborates on details of some steps such vote transfer:

details of transferring surplus votes. The value of the surplus votes gained by an elected candidate is passed on to other candidates according to the preferences indicated on ballot papers by the voters. If a candidate has received more than a quota of first preference votes, all the ballot papers received by the candidate are distributed at a reduced value called a fractional transfer value. If a candidate has received more votes than the quota following a transfer of votes from another elected candidate or from an excluded candidate, only that "last parcel" of ballot papers that the candidate received are distributed to continuing candidates at a fractional transfer value.

After the surplus votes from an elected candidate have been distributed, the total number of votes which each candidate has received is recalculated. Any further candidates that have votes equal to or greater than the quota are elected. Provided vacancies remain to be filled, the surplus votes of any newly elected candidate are now also distributed one by one.

2.2 The Machine States and Transitions

In tallying process of an election, there are some pieces of information which are necessary to know in order to handle the computation. Moreover, this kind of data invariably appears throughout the tallying process so that tally officers would have access to current state of the procedure. For example, in hand counting methods, officers must know what are the uncounted ballots and what is the current tally amount for each candidate. Since computations are local phenomena, tallying process is divided into stages of counting each of which comprises such vital data. These encapsulated pieces of information, form the sates of our generic STV machine.

There are three types of machine sates; *initial*, *intermediate*, and *final*. An initial state specifies the list of all *formal* ballots cast to be tallied. On the other hand, final states of the machine are accepting stages where winners of an election are announced. Last, each intermediate state consist of six components:

1. A set of uncounted ballots, which must be counted
2. A tally function computing the amount of vote for each candidate
3. A pile function computing which ballots are assigned to which candidate
4. A list of already elected candidate whose votes awaits being transferred
5. A list of elected candidates
6. A list of continuing candidates

One could think of the pile and tally functions as abstraction of the action performed by tally officers when they respectively assign ballots and their values to candidates.

To express machine states in a mathematically enough precise language, we use symbols each of which stands for a notion introduced above. A set of candidates participating in an election is represented by $\mathcal{C}$ and members of this set are illustrated by c, c', and c''. The set of ballots $\mathcal{B}$, is a short hand for $(\mathsf{List}\ \mathcal{C})\times\mathbb{Q}$, where $\mathbb{Q}$ is the set of rational numbers. Therefore a ballot ba is a pair (l, q) where $l \in \mathsf{List}(\mathcal{C})$ and $q \in \mathbb{Q}$. The characters h and nh are reserved for lists of continuing candidates, e and ne for lists of elected candidates, and bl, nbl for backlogs. The quota of election and number of seats are symbolised by qu and st, respectively. Finally, tallies are shown by t, nt and piles by p, np.

Suppose $ba \in \mathcal{B}$, and $bl, h, e, w \in \mathsf{List}(\mathcal{C})$ are given. Also assume t is a function from $\mathcal{C}$ into $\mathbb{Q}$, and p is a function from $\mathcal{C}$ into $\mathsf{List}(\mathcal{B})$. Then we illustrate an initial state of the machine by $\mathsf{initial}(ba)$, an intermediate state by $\mathsf{intermediate}(ba, t, p, bl, e, h)$, and a final one by $\mathsf{final}(w)$. Having established terminology and necessary representations, we can mathematically define the states of the generic STV machine.

Definition 1 (machine states). *Suppose ba is the initial list of ballots cast to be counted, and l is the list of all of candidates competing in the election. Then the set $\mathcal{S}$ of states of the generic STV equals to all of possible intermediate and final states formed based on ba and l, together with the initial state* $\mathsf{initial}(ba)$.

There is also a mechanism devised to advance the counting process by updating the current state of the count with necessary changes. For example, if the counting comes to a stage where some candidate has received enough votes to be elected, a particular rule for electing permits making the transition from this state into a new one where the candidate has been elected.

These steps which are an integral part of each STV and perform updating the information locally, are named counting rules. A specific set of counting rules consistently comes into sight when looking into instances of STV:
start. to determine the *formal* votes and valid initial states.
count. for counting the uncounted ballots,
elect. to elect one or more candidates who have reached or exceeded the quota,
transfer. for transferring surplus votes of already the elected,
eliminate. to eliminate the weakest candidate from the process, and
hopeful win. to finish the counting by declaring the list of elected and continuing candidates as winners.
elected win. to finish the counting by announcing the already elected candidates as winners.

Each of these counting rules accept a machine state as input and output another state. At the moment, we treat them merely as transition labels of the generic STV. However, in the next section, we specify a semantics for each and explain to what kind of state they apply.

Definition 2 (machine transitions). *The set $\mathcal{T}$ consisting of the labels* ***count****,* ***elect****,* ***transfer****,* ***eliminate****,* ***hopeful win****, and* ***elected win****, is the set of transition labels of the generic STV.*

2.3 The Small-step Semantics

STV protocols are composed of clauses which textually describe the expectations of the protocol from each counting rule. Protocol clauses informally specify when and to what kind of state a counting rule applies, and how it must update this state by making a transition to another. From the algorithmic perspective, STV protocols carry two common consistent properties : invariant requirements in order for a counting rule to be applicable, and invariant order of rule applications.

Various STV algorithms differ in details of what conditions must be met before a specific transition step can apply to a given state of the machine. For example, the lower house ACT STV transfers only *the last parcel* of an elected candidate. However, some other STV schemes such as the one used in the upper house Victoria state of Australia that transfer all of the surplus votes rather than merely the last parcel received.

However, there are conditions that appear invariably among different STV schemes. These conditions, each of which correspond to a transition label, comprise the small-step semantics for the generic STV machine. They are also used to check if a given arbitrary STV can legitimately be categorised as an STV instance. We obtain the conditions by singling out the key invariant properties existing in each STV scheme and name them *sanity checks*.

Reducibility. A careful examination of STV protocols illustrates that each rule application at least reduces the length of one the three lists: the list of continuing candidates, the backlog, or the list of uncounted ballots. The astute observer quickly grasps that by the correct choice of ordering among the above quantities, a complexity measure can be imposed on the set of machine states in such a way that each rule application reduces the measure. This measure persists to apply and function across various STV algorithms and to each transition label..

Local Rule Applicability. In order to legally correctly apply a counting rule, the protocol declares some restrictions to be met first. Many of the constraints depend on the particular protocol, however some of them consistently come to attention. For example, as you can also see in step 3 of the lower house ACT STV, all of STV algorithms require three properties to hold in order for elimination rule to apply: there must be empty seats to fill, there must not be any surplus votes awaiting transfer, and no candidate should have reached or exceeded the quota. Each one of the counting rules is constrained to their distinct conditions that constantly apply to the regardless of the specifics of the STV protocol invoked.

To formulate the sanity checks for each transition step, we first define a lexicographic ordering on the set $\mathbb{N}\times\mathbb{N}\times\mathbb{N}\times\mathbb{N}$ and impose it on non-final states of the generic machine.

Definition 3. *The measure* Reduction: $\{s : \mathcal{S} |\ s \text{ not final}\} \longrightarrow \mathbb{N}\times\mathbb{N}\times\mathbb{N}\times\mathbb{N}$ *is defined as:*

$$\textit{Reduction (initial } ba) = (1,0,0,0)$$
$$\textit{Reduction (state } (ba, t, p, bl, e, h)) = (0, \textit{length } h, \textit{length } bl, \textit{length } ba).$$

We are now able to formalise the sanity checks. The first one describes what it means for a start rule to be a legitimate STV start rule.

2.4 The Big-step Semantics

3 Formalisation of The Generic Machine in Coq

4 Modular Formalisation of Some STV Machines

4.1 Victoria STV

4.2 ACT STV

4.3 CADE STV