

Analysing Vote Counting Algorithms via Logic And Its Application to the CADE Election Scheme

Bernhard Beckert¹, Rajeev Goré², and Carsten Schürmann³

¹ Karlsruhe Institute of Technology
beckert@kit.edu

² The Australian National University
Rajeev.Gore@anu.edu.au

³ IT University of Copenhagen
carsten@itu.dk

Abstract. We present a method for using first-order logic to specify the semantics of preferences as used in common vote counting algorithms. We also present a corresponding system that uses Celf linear-logic programs to describe voting algorithms and which generates explicit examples when the algorithm departs from its specification. When we applied our method and system to analyse the vote counting algorithm used for electing the CADE Board of Trustees, we found that it strictly differs from the standard definition of Single Transferable Vote (STV). We therefore argue that “STV” is a misnomer for the CADE algorithm.

1 Introduction

Most research in electronic voting concerns correctness and voter-verifiability of vote-casting. It is also vital that vote counting and tabulation of election results enjoys the same trust as physical counting of paper ballots. Counting votes in a “first past the post” scheme is easy, but there are numerous preferential voting schemes for proportional representation where vote-counting is rather complex.

In social choice theory, the general problem of finding an election result that perfectly reflects the electorate’s collective preferences has no solution. Collective preferences can even be cyclic (Condorcet’s paradox). Various voting schemes (and corresponding algorithms) exist that attempt to provide “good” election results from the preferences expressed in voters’ ballots. They compromise in different ways, for example w.r.t. majority rule vs. minority protection.

In preferential voting, the correct election result for a given set of votes is usually defined algorithmically (in natural language or in pseudo-code) rather than as clear, concise, formal and declarative specifications. A prominent class of such algorithms is the Single Transferable Vote (STV) scheme (see Section 2). Variants of STV are used in many jurisdictions around the world. Testing and verification methods are useful for validating that an implementation refines a high-level algorithmic description, but without good declarative specifications they remain ineffective to gain deeper insight into the democratic nature of the scheme. Unfortunately, the complexity of STV makes it difficult to render its declarative properties succinctly: a possible solution is presented in Section 3.

In Section 4, we then describe an automated reasoning system to check vote counting schemes w.r.t. their declarative properties. Our technique applies to STV but can be generalised to other schemes as well. Building on earlier work [4], we use the concurrent logical framework Celf [8] for (1) representing vote counting algorithms, (2) representing declarative properties, and (3) for building tools such as a bounded model checker for constructing counter examples. The Celf source code can be found at <http://www.demtech.dk/wiki/CADE24-STV>.

We applied our methodology to the “STV” vote-counting algorithm used to elect the CADE Board of Trustees (Section 5) and found that this algorithm does not implement a proportional voting scheme. Thus it is a misnomer to call the CADE algorithm a scheme for Single Transferable Voting (Section 6).

Related work at the Australian National University has analysed a commercial implementation of the Hare-Clark (STV) scheme used in Australian Capital Territory parliamentary elections [6] while Cochran has formally specified and analysed the Danish and Irish vote counting algorithms [3]. Both concentrate on verifying implementations. We concentrate on analysing abstract algorithms.

2 The Single Transferable Vote Scheme

Single transferable vote (STV) is a preferential voting scheme [9] for multi-member constituencies aiming to achieve proportional representation according to the voters’ preferences. Suppose that C candidates, numbered $1, 2, \dots, C$, are competing for $S > 0$ vacant seats in an election. Furthermore, assume that $V \geq 0$ votes have been cast and are collected in a ballot box. It is commonly agreed that for $k \leq C$, a vote $[P_1, P_2, \dots, P_k]$ ranks a subset of the candidates with $P_i \in \{1, 2, \dots, C\}$ in decreasing order of preference. Each vote defines a partial order on candidates. STV first computes a quota necessary to obtain a seat. Different definitions of quotas are used in practice and the most common is the Droop quota $Q = \lfloor V/(S+1) \rfloor + 1$. Then, STV computes the result using an iterative process, which repeats the following two steps until either a winner is found for every seat or no further candidate can be elected:

1. Any candidate with Q or more first-preference votes is declared elected. The Q votes used to elect such a candidate are removed from the ballot box. If the elected candidate has more votes than the quota Q , these surplus votes are transferred to the next candidates according to preference.
2. If no candidate reaches the quota, the candidate with the fewest first-preference votes is eliminated and that candidate’s votes are transferred in the same way as described in step 1.

Example 1. Assume there are four candidates A, B, C, D for two vacant seats, and the votes to be counted are $[A, B, D]$, $[A, B, D]$, $[A, B, D]$, $[D, C]$, $[C, D]$. The Droop quota here is $Q = \lfloor 5/(2+1) \rfloor + 1 = 2$. In the first iteration, we tally first preferences, only A meets the quota and is hence elected. Two votes $[A, B, D]$ are deleted, the third is a surplus vote. It is transferred and transformed into $[B, D]$. In the second iteration, no candidate reaches the quota, thus the weakest of

the remaining candidates B, C, D is eliminated – which one depends on the kind of tie-breaker used as all three have exactly one first-preference vote at that point. (1) If the tie-break eliminates B , the aforementioned transferred vote $[B, D]$ will be transferred again and will become a vote for D , so that D will be elected in the next iteration. (2) If the tie-break eliminates C , the vote $[C, D]$ will be transferred into a vote for D , and thus D will be elected. (3) If the tie-break eliminates D , then C will be elected, analogously, in the next iteration. In summary, the algorithm reports either $[A, D]$ or $[A, C]$ as the election result but not, for example, $[A, B]$ or $[B, D]$. If the number of second-preference votes is used as a tie-breaker, then B is eliminated first (case 1 above).

This illustrates that STV, as given above in informal English, defines an entire family of vote counting algorithms. There are a number of parameters to play with, which we name for later reference: the choice of quota (QUOTA/DROOP, QUOTA/HARE, QUOTA/MAJORITY), the choice of tie-breaker (TIE), the possible resurrection of already eliminated candidates when they receive transferred votes in later rounds (ZOMBIE), the automatic placement of candidates once there are as many vacant seats as remaining candidates (AUTOFILL).

As we argue in Section 5, further options giving schemes that cannot be considered part of the STV family are the persistence of votes used in electing candidates from one iteration to the next (NODEL), or restarting with the original ballot box (RESTART) after a candidate has been elected (with the elected candidate removed). RESTART is a combination of NODEL and ZOMBIE.

3 Semantic Criteria for Judging Voting Schemes

Voting schemes are frequently under-specified via something like “The members of parliament must be elected by a regular, direct, and secret election”, or over-specified via a concrete (pseudo-code) algorithm in the law text that may itself contain bugs. So what is a good specification of STV?

Many semantic criteria have been proposed for preferential vote-counting algorithms including the majority criterion, Condorcet criterion, monotonicity criterion, to name a few [1]. The majority criterion says that *if one candidate is highest ranked by a majority of voters, then that candidate must be elected*. A violation of the majority criterion is clearly undemocratic. But to analyse and distinguish variants of (democratic) voting schemes, stronger criteria are needed that are tailored for the particular scheme. For some voting schemes, axiomatic criteria can fully characterise the correct election result (e.g. the Borda Rule [5]).

However, for STV schemes, writing a declarative specification that fully characterises the election result is hard. For our analysis of STV, we have instead devised several semantic criteria that capture and approximate the essence of STV and are suitable for program verification. We mention only two, but remark that for a more thorough analysis more such criteria will have to be added.

- (1): There must be enough votes for each elected candidate.
- (2): If the preferences of *all* voters w.r.t. two particular candidates are consistent, then that collective preference is not contradicted by the election result.¹

The first criterion only considers number of votes and ignores preferences, while the second criterion only considers preferences and ignores number of votes. This separation of the two dimensions (number of votes and preferences) is the key to finding criteria that can be described declaratively. Due to space restrictions, we do not further consider Criterion 2 in this paper but concentrate on Criterion 1.

The first criterion is only justified for versions of STV that do not use AUT-OFILL. It captures the fact that the ballot box can be partitioned in such a way that each partition justifies the seating of a candidate: Each class contains quota Q ballots, each of which mentions the candidate elect at least once.

Example 2. Returning to Example 1, note that the election result $[A, D]$ satisfies Criterion 1 with the partition $\{[A, B, D], [A, B, D]\}, \{[C, D], [D, C]\}, \{[A, B, D]\}$. The incorrect election result $[B, D]$ also satisfies Criterion 1 choosing the same partition (because the ordering of A and B is not considered), which shows that the criterion is not a complete specification of the election result. But, the incorrect result $[A, B]$ is not supported by this or any other partition.

We use first-order logic over the theories of natural numbers and arrays with the following notation in addition to the notation defined previously:

- b : is the ballot box, where $b[i, j]$ is the number of the candidate that is ranked by voter i in the j th place. If the voter does not rank all candidates, then $b[i, j] = 0$ for the empty places.
- r : is the result, where $r[i]$ is the i th candidate that is elected ($1 \leq i \leq S$). If less than S candidates are elected, then $r[i] = 0$ for the empty seats.

We also use an existentially quantified array a that represents the partition and the assignment of classes in the partition to elected candidates as follows:

- $a[i] = k$ if the i th vote supports the k th elected candidate $r[k]$. If the i th vote does not support any elected candidate, then $a[i] = 0$.

Then, the formula $\phi = \exists a(\phi_1 \wedge \dots \wedge \phi_4)$ is the existentially quantified conjunction:

$$\forall i(1 \leq i \leq V \rightarrow 0 \leq a[i] \leq S) \quad (\phi_1)$$

$$\forall i(1 \leq i \leq V \rightarrow (a[i] \neq 0 \rightarrow r[a[i]] \neq 0)) \quad (\phi_2)$$

¹ This criterion is weaker than what is known as the Condorcet criterion. We assume a preference to be collective if *all* voters agree (or at least not disagree), while the Condorcet criterion assumes a preference to be collective if it is supported by a *majority* of voters. It is well known that the (stronger) Condorcet criterion is not satisfied by standard STV, so it is not suitable for our purposes.

$$\forall i((1 \leq i \leq \mathbf{V} \wedge a[i] \neq 0) \rightarrow \exists j(1 \leq j \leq \mathbf{C} \wedge b[i, j] = r[a[i]])) \quad (\phi_3)$$

$$\begin{aligned} \forall k((1 \leq k \leq \mathbf{S} \wedge r[k] \neq 0) \rightarrow & \quad (\phi_4) \\ \exists count(count[0] = 0 \wedge & \\ \forall i(1 \leq i \leq \mathbf{V} \rightarrow (a[i] = k \rightarrow count[i] = count[i-1] + 1) \wedge & \\ (a[i] \neq k \rightarrow count[i] = count[i-1])) \wedge & \\ count[\mathbf{V}] = \mathbf{Q})) & \end{aligned}$$

Formulas ϕ_1 and ϕ_2 express well-formedness of the partition. Formula ϕ_3 expresses that only votes can support a candidate in which that candidate is somewhere ranked. Formula ϕ_4 expresses that each class supporting a particular elected candidate has exactly $\mathbf{Q}$ elements. To formalise this, we use an array *count* such that *count*[*i*] is the number of supporters among votes $1, \dots, i$ that support the *k*th elected candidate.

4 The System for Analysing Vote Counting Algorithms

Votes are resources that must be counted exactly once, which already suggests that linear logics are well-suited for representing voting-algorithms (see Section 2 and Criterion 1 in Section 3). We therefore used the (linear) logical framework Celf [8]. We considered using model checkers and SMT solvers for the bounded model-checking part, but we stayed with Celf, mainly as a matter of convenience. Because of space restrictions, we only sketch the system here.

4.1 Vote Counting Algorithms as Linear-Logic Programs

The Celf logical framework is based on intuitionistic linear logic. Its operational semantics is proof search, which means that running a vote counting algorithm is tantamount to constructing a *derivation* for “ $\Gamma; \Delta \vdash \text{run } i \text{ C V S } w \text{ } d$ ”. We explain the symbols in turn. The Γ is the unrestricted (intuitionistic) context. Its declarations, like AUTOFILL, NODEL, etc. define precisely the particular STV algorithm to be analysed. During execution, Γ is also populated with assumptions about who was elected and who was defeated.

The Δ to the right of Γ denotes the linear context that contains assumptions that must be used exactly once. It contains, for example, all information about the ballot-box, running totals, etc. The ballot box is represented by a multi-set of assumptions **uncounted-ballot** $A \ L$ (first preference A , remaining preferences L); the running totals as a multi-set of assumptions **hopeful** $A \ N$, summarising that A ’s running total is N .

run is a 6-ary predicate, relating the number of times i that STV may be restarted (see RESTART), the number of candidates $\mathbf{C}$, the number of cast votes $\mathbf{V}$, the number of seats $\mathbf{S}$ that should be filled with each iteration, the list of winners w , and the list of defeated candidates d . The total number of seats filled by the algorithm is hence $i \times \mathbf{S}$.

To save space, we present only one of the rules implementing STV. An introduction on how to represent STV counting algorithms in Celf is given in [4].

This rule elects candidate A after receiving an additional vote that meets the quota.

```
count/2 : count-ballots (S + 2) (H + 1) (U + 1) ⊗
          uncounted-ballot A L ⊗
          hopeful A N ⊗ !quota Q ⊗ !nat-lesseq Q (N + 1) ⊗
          winners W D
          → {counted-ballot A L ⊗ !elected A ⊗
              winners (scons A (N + 1) W) D ⊗
              count-ballots (S + 1) H U}.
```

All uppercase variables are universally bound, we write $\multimap$ for linear implication, $\otimes$ for multiplicative conjunction, $!$ for the bang modality permitting unrestricted assumptions to appear in declarations, and $\{...\}$ for the polarity shift from positive to negative formulas (as Celf implements a focused linear logic). The rule `count/2` can be read as a forward-chaining multi-set rewrite rule. When no candidate reaches the `!quota Q`, the candidate with the fewest votes must be eliminated, and its (already counted) votes redistributed. The bang in front of `quota` indicates that this is an unrestricted assumption that should not be consumed. It is therefore mandatory to keep information about `counted` ballots around, and we do this by replacing an `uncounted-ballot A L` by a `counted-ballot A L`.

Theorem 1 (Standard STV). *Let $\Gamma = \text{QUOTA/DROOP, AUTOFILL, TIE}$, and let $\Delta = \text{ballot box} + \text{initialized running counts}$, then `run 1 C V S w d` is provable if and only if w corresponds to the list of candidates elected by the standard STV algorithm, and d is the list of defeated candidates.*

4.2 Bounded Model Checker

Our bounded model checker is implemented in Celf, taking advantage of the *generate* and *test* behaviour of logic programs. Our system provides an implementation of Criterion 1 as a linear logic program: `sem W D`. Other criteria may be implemented analogously. The model checker generates all possible ballot boxes up to a given bound. The bound comprises the maximal number of permitted RESTARTs max_i , the maximal number of candidates max_c , the maximal size of the ballot box max_v , and the maximal number of seats max_s . Checking the algorithm for a particular input corresponds to running the query:

$$\Gamma; \Delta \vdash (\text{run } i \ c \ v \ s \ W \ D) \ \& \ (\text{sem } W \ D)$$

As above, Γ selects the algorithm for the desired version of STV, Δ, i, c, v, s are inputs for the algorithm that have been generated by the model checker. We use additive conjunction $\&$ in a clever way: it copies the ballot box and allows the box to be used both for running STV and for semantically checking the result.

As inherent in *bounded* model checking, we get a semi-decision procedure. The analysis terminates for any given bound (this is easy to prove by an inspection of the linear logic program). But only in the negative case, where we get a counter

example, the model checker provides a definite answer to the question of whether the algorithm always computes an election result satisfying the given criterion.

In the positive case, where the model checker constructs as many solutions to the above query as there are ballot boxes, we can conclude that the particular STV algorithm selected by I computes valid solutions for all possible elections up to the given bound. Note, that this conclusion requires the vote counting to be deterministic because the current version of the checker does not backtrack over different results for the same input and only validates the first election result found for a particular ballot box. This is not a critical limitation in practice. Although various ways of resolving non-determinism in STV exist, it is important to clearly specify how it is resolved in real-world implementations of TIE. Otherwise, choices by the election officials (or their computers) during counting could influence the result, which is clearly undesirable. Our checker could backtrack over election results but would require greater runtime.

The more interesting case is when the model checker fails to find a solution for one of two reasons. Either, the STV algorithm did not manage to construct an election result for some ballot box, a case that may happen, for example, if AUTOFILL is not selected. Or the model checker was unable to justify an election result w.r.t. the semantic criteria (the really negative case). The Celf tracing model provides enough information to deduce where to find the culprit.

5 Case Study: CADE-STV

The bylaws of the Conference on Automated Deduction (CADE) give an algorithm for counting the ballots for electing the Board of Trustees [2] which is identical to Figure 1 except for typesetting. It has been implemented and used in several elections for the CADE Board of Trustees. It has also been used several times for the election of members to the TABLEAUX Steering Committee.

Note that the specification of CADE-STV is not formal. Although the pseudo code language that is used may be intuitive for programmers, it does not come with a precise operational semantics. Despite being semi-formal, the pseudo code lacks precision in how to break a tie when eliminating or seating candidates.

The CADE-STV algorithm deviates from standard STV in using RESTART, which combines NODEL and ZOMBIE. Moreover, QUOTA/MAJORITY is used instead of QUOTA/DROOP and there is no AUTOFILL, which is unusual. Presumably, QUOTA/MAJORITY was introduced into CADE-STV following criticism of DROOP/QUOTA by David Plaisted [7].

Example 3. Let us run CADE-STV on Example 1. First, we compute the majority quota $Q = 3$. In the first iteration, A has three first preferences, which means that A is the majority winner and is seated. Since CADE-STV uses RESTART, A 's votes are not deleted but are redistributed at the end of the first iteration. Now the ballot box contains $[B, D], [B, D], [B, D], [D, C], [C, D]$. Following the algorithm, we observe that now B is the majority candidate with 3 first preference votes and is seated. The election is over, and the election result is $[A, B]$.

Problem: M voters must elect K of N candidates.

Input: $M \times N$ matrix, Tbl of votes.

$\text{Tbl}(i, j) = p$, $1 \leq p \leq N$, means that voter i gave preference p to candidate j . Every voter can support n ($1 \leq n \leq N$) of the N candidates, and has to give a preference order between those n candidates. This is expressed by assigning a preference between 1 (highest preference) to n (lowest preference) to each of the supported candidates. Each of the values $1 \dots n$ is assigned to exactly one candidate. All candidates not supported receive a preference of $N + 1$.

Weakest candidate: The candidate with the fewest votes of preference 1. Ties are broken by fewest votes of preference 2, then 3, etc.

Equally weak candidates: c is equally weak as w iff c and w have the same number of votes of preference 1, 2, etc.

Output: List of K elected candidates in order of election.

```

Redistribute(k, Tbl):
  for v <-- 1 to M
    p <-- Tbl(v,k)  {* v's preference for candidate k *}
    for c <-- 1 to N
      p' <-- Tbl(v,c)  {* v's preference for candidate c *}
      if p' > p and not p' == N+1 then
        decrement Tbl(v,c) by one
    end for
  end for
  Now remove candidate k from Tbl  {* column k *}

Procedure STV
  Elected <-- empty
  T <-- Tbl          {* Start with the original vote matrix *}
  for E <-- 1 to K
    N' <-- N-E+1  {* Choose a winner among N' candidates *}
    T' <-- T      {* store the current vote matrix *}
    while (no candidate has a majority of 1st preferences)
      w <-- one weakest candidate
      for all candidates c  {* remove all weakest candidates *}
        if c is equally weak as w
          Redistribute(c,T)
      end for
    end while
    win <-- the majority candidate
    Elected <-- append(Elected, [win])
    T <-- T'      {* restore back to N' candidates *}
    Redistribute(win, T)  {* remove winner & redistrib. votes *}
  end for
End STV

```

Fig. 1. Description of the CADE-STV algorithm [2]

Standard STV and CADE-STV produce different results on the same votes. Example 2 has already shown that $[A, B]$ is “incorrect” as it violates Criterion 1.

Theorem 2. *Let $\Gamma = \text{QUOTA/MAJORITY, RESTART}$ and $\Delta = \text{ballot box} + \text{initialized running counts}$, then run $\text{SCV } 1 \text{ w } d$ is provable if and only if w is the list of candidates elected by CADE-STV, and d is the list of defeated candidates.*

Running the bounded model checker confirms that the election results computed by CADE-STV do not always satisfy Criterion 1. Indeed, it finds smaller counter examples than our running example, but these are not as illustrative.

The effect of the differences between standard STV and CADE-STV is clarified by the following theorem and its corollary: in certain cases, there is no proportional representation in the election results computed by CADE-STV.

Theorem 3. *If a majority of voters all vote identically with $[P_1, \dots, P_k]$, then CADE-STV will elect exactly these candidates in exactly this order.*

Proof. Since a majority of voters choose P_1 as their first preference, no other candidate can meet the “majority quota”. Thus P_1 is elected in the first round. When redistributing the ballots, each of the majority of ballots with P_1 as first preference have P_2 as second preference. All become first preferences for P_2 . Thus candidate P_2 is guaranteed to have a majority of first preferences and is elected in round two, and so on until all vacancies are filled. $\square$

Corollary 1. *If the electorate consists of two diametrically opposed camps that vote for their candidates only, in some fixed order, then all candidates from the majority camp are elected and no candidates from the minority camp are elected.*

Standard STV does not use RESTART (nor NODEL), so it elects the first ranked candidate of the majority. It then distributes only the surplus votes, not all votes as done by CADE-STV. Thus the second preference from the majority is not necessarily elected next. Majorities do not rule outright in standard STV.

6 Conclusion

The experiments with our tool show that CADE’s “single transferable vote” voting scheme does not satisfy the intuitive semantic criterion defined in Section 3 and does not achieve proportional representation. We suspect that CADE’s voting scheme was intended to combine the advantages of preferential and majority voting (nothing can happen that the majority does not want). Unfortunately, it also combines their disadvantages (no proportional representation).

Our observations do not imply that CADE voting is undemocratic. But calling the CADE algorithm “Single Transferable Vote” is a misnomer because the goal of proportional representation is inherent to STV. The CADE algorithm is actually closer to what is known as *Majority Preference Voting*.

CADE-STV has been used for many years. It has been implemented, tested, re-implemented, and re-tested by various people. Its properties have been thoroughly discussed at various times by the CADE Trustees. But to our knowledge, the problems outlined in this paper have not been observed before, which clearly indicates that a formal analysis like the one presented here is indispensable.

References

1. Brandt, F., Conitzer, V., Endriss, U.: Computational social choice. In: Weiss, G. (ed.) Multiagent Systems. MIT Press (2012)
2. CADE Inc.: CADE Bylaws (effective November 1, 1996; amended July/August 2000), <http://www.cadeinc.org/Bylaws.html> (accessed January 20, 2013)

3. Cochran, D.: Formal Specification and Analysis of Danish and Irish Ballot Counting Algorithms. Ph.D. thesis, IT University of Copenhagen (2012)
4. DeYoung, H., Schürmann, C.: Linear logical voting protocols. In: Kiayias, A., Lipmaa, H. (eds.) *VoteID 2011*. LNCS, vol. 7187, pp. 53–70. Springer, Heidelberg (2012)
5. Farkas, D., Nitzan, S.: The Borda Rule and Pareto Stability: A comment. *Econometrica* 47(5), 1305–1306 (1979)
6. Logic and Computation Group at ANU: Formal methods applied to electronic voting systems, <http://users.cecs.anu.edu.au/~rpg/EVoting/index.html> (retrieved) (accessed January 20, 2013)
7. Plaisted, D.A.: A consideration of the new CADE bylaws, <http://www.cs.unc.edu/Research/mi/consideration.html> (accessed January 20, 2013)
8. Schack-Nielsen, A., Schürmann, C.: Celf: A logical framework for deductive and concurrent systems (system description). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) *IJCAR 2008*. LNCS (LNAI), vol. 5195, pp. 320–326. Springer, Heidelberg (2008)
9. Wikipedia: Single transferable vote, http://en.wikipedia.org/wiki/Single_transferable_vote (accessed January 20, 2013)