Analysing Vote Counting Algorithms Via Logic
And its Application to the CADE Election System

Bernhard Beckert[1], Rajeev Goré[2], and Carsten Schürmann[3]

[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 our method and system are applied to analyse the vote counting algorithm used to elect CADE Trustees, we find that it strictly differs from the standard definition of Single Transferable Vote (STV). We therefore argue that it is a misnomer to say that the CADE algorithm implements STV.

1 Introduction

Most research in electronic voting is about ensuring correctness and voter-verifiability of the vote-casting process. But it is also vital to imbue the second step, namely vote counting and the computation of election results, with the trust enjoyed by physical counting of paper ballots.

Counting votes in a "first past the post" system is easy, but there are numerous schemes for preferential voting where the counting process is extremely complicated. Schemes that also allow for proportional representation are even more complicated since multiple rather than single candidates can be elected.

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 systems (and corresponding algorithms) exist which attempt to provide "good" election results from the preferences expressed in voters' ballots. They compromise in different ways between giving everything to the majority and giving something to minorities, and also treat inconsistent preferences in different ways.

A prominent class of such algorithms is the Single Transferable Vote (STV) system (see Section 2). Variants of STV are used in many jurisdictions around the world. In this paper, we concentrate on STV systems, but the general ideas apply to other vote counting systems as well.

In preferential voting, the correct election result for a given set of votes is usually described algorithmically (in natural language or in some pseudo-code language). One can use testing and verification methods to validate that an implementation refines that high-level description, but if the high-level algorithmic description is erroneous, the error is elusive unless it leads to obviously undemocratic results. Therefore we need declarative properties for analysing and comparing vote counting algorithms that are vital for finding errors in high-level descriptions. Unfortunately, it is notoriously hard to devise such declarative specifications for STV w.r.t. which the high-level algorithms or their implementation can be validated; a possible solution is presented in Section 3.

We present a system that supports automated reasoning about vote counting algorithms w.r.t. their declarative properties in Section 4. 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`.

As a case study, we have applied our system to the "STV" counting algorithm that is used in the CADE Trustees Elections (Section 5). We found that the CADE algorithm differs from the standard in that it does not implement a proportional voting system. Thus it is a misnomer to say that the CADE algorithm implements a system for Single Transferable Voting (Section 6).

In related work, a group at the Australian National University has formalised and analysed the Hare-Clark system used in the governmental elections of the Australian Capital Territory [6], which is similar to STV. Dermot Cochran has formally specified and analysed the Danish and Irish vote counting algorithms [3]. Both differ from our work in that they are mainly concerned with verifying implementations, while we are mainly interested in analysing abstract algorithms.

2 The Single Transferable Vote System

Single transferable vote (STV) is a preferential voting system [9] for multi-member constituencies aiming to achieve proportional representation according to the voters' preferences. Suppose that $\mathtt{C}$ candidates, numbered $1, 2, \ldots, \mathtt{C}$, are competing for $\mathtt{S} > 0$ vacant seats in an election. Furthermore, assume that $\mathtt{V} \geq 0$ votes have been cast and are collected in a ballot box. It is commonly agreed that for $k \leq \mathtt{C}$, a vote $[P_1, P_2, \ldots, P_k]$ ranks a subset of the candidates with $P_i \in \{1, 2, \ldots, C\}$ in decreasing order of preference. Each vote defines a partial order on candidates.

To determine the election result, 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 $\mathtt{Q} = \lfloor \mathtt{V}/(\mathtt{S}+1) \rfloor + 1$. 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 $\mathtt{Q}$ or more first-preference votes is declared elected. The $\mathtt{Q}$ votes used to elect such a candidate are removed from the ballot box. If the elected candidate has more votes than the quota $\mathtt{Q}$, these surplus votes are transferred to other candidates. To do that, the elected candidate is removed from the ballot, the second-preference candidate becomes the first preference, the third preference becomes the second preference, and so on.
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 $\mathtt{Q} = \lfloor 5/(2+1) \rfloor + 1 = 2$. 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 example illustrates that STV, as given above in informal English, defines not only one but an entire family of vote counting algorithms. There are a number of parameters to play with, which we name here to be able to refer back to them later: 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 free seats as remaining candidates (AUTOFILL).

Further options, which – as we argue in Section 5 – lead to election systems that can no longer be considered part of the STV family, are the persistence of votes used in electing candidates from one iteration to the next (NODEL), or even restarting the STV algorithm with the original ballot box (RESTART) after a candidate has been elected (with only the elected candidate removed). RESTART is a combination of NODEL and ZOMBIE.

3 Semantic Criteria

When voting systems are implemented and deployed at a national level, it seems natural to expect them to be verified against a specification preferably using formal methods. But what is a good specification in case of STV? The law itself is rarely helpful because voting systems are frequently massively underspecified via something like "The members of the parliament are to be elected by a regular, direct, and secret election", or often overspecified by including a concrete algorithm in the law text in the form of pseudocode that may itself contain bugs. Neither of the approaches lends itself to an easy verification exercise.

In the case of preferential voting, many semantic criteria have been proposed for vote counting algorithms, including the majority criterion, Condorcet criterion, monotonicity criterion, to name a few [1]. The majority criterion, for example 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 would clearly be undemocratic. But to analyse and distinguish variants of (democratic) voting systems, stronger criteria are needed that are tailored towards the particular system. For some voting systems, axiomatic criteria can be given that fully characterise the correct election result (an example is the Borda Rule [5]).

However, for STV systems, writing a declarative specification that fully characterises the election result is very hard. For our analysis of STV, we have instead devised several semantic criteria that capture the essence of STV and are suitable for program verification but which do not form a fully unambigous specification. Two of these criteria are mentioned here:

(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.[4]

[4] 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.

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.

As said above, our two criteria are not complete or sufficient in the sense that they fully characterise the correct election result. They cover many possible errors in STV algorithms, but for a more thorough analysis more such criteria will have to be added. Also, due to space restrictions, we do not further consider Criterion 2 in this paper but concentrate on Criterion 1. This criterion captures that the votes can be partitioned with an assignment of exactly one class in the partition to each elected candidate such that, if Q is the quota, then:

1. There are exactly Q votes in each class that supports an elected candidate.
2. For each vote in a class that supports a candidate, that candidate occurs somewhere among the preferences of the supporting vote.

In the second condition above, the actual order of preferences is not taken into consideration. Thus, this is actually a weak property that can be satisfied by a wide variety of STV variants. But it is strict in that each vote counts only 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 jth 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 ith candidate that is elected ($1 \leq i \leq \mathtt{S}$). If less than $\mathtt{S}$ candidates are elected, then $r[i] = 0$ for the empty seats.

Our criterion is formalised by a formula ϕ in which all the above (free) variables occur. 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 ith vote supports the kth elected candidate $r[k]$. If the ith vote does not support any elected candidate, then $a[i] = 0$.

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

$$\forall i\big(1 \leq i \leq \mathtt{V} \rightarrow 0 \leq a[i] \leq \mathtt{S}\big) \tag{ϕ_1}$$

$$\forall i\big(1 \leq i \leq \mathtt{V} \rightarrow (a[i] \neq 0 \rightarrow r[a[i]] \neq 0)\big) \tag{ϕ_2}$$

$$\forall i\big((1 \leq i \leq \mathtt{V} \wedge a[i] \neq 0) \rightarrow \exists j(1 \leq j \leq \mathtt{C} \wedge b[i, j] = r[a[i]])\big) \tag{ϕ_3}$$

$$\begin{aligned}&\forall k\big((1 \leq k \leq \mathtt{S} \wedge r[k] \neq 0) \rightarrow \\ &\quad \exists count\big(count[0] = 0 \wedge \\ &\qquad \forall i(1 \leq i \leq \mathtt{V} \rightarrow (a[i] = k \rightarrow count[i] = count[i-1] + 1) \wedge \\ &\qquad\qquad (a[i] \neq k \rightarrow count[i] = count[i-1])) \wedge \\ &\qquad count[\mathtt{V}] = \mathtt{Q}\big)\big)\end{aligned} \tag{ϕ_4}$$

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 `Q` elements. To formalise this, we use an array *count* such that $count[i]$ is the number of supporters among votes $1, \ldots, i$ that support the kth elected candidate. Note, that this criterion is only justified for versions of STV that do not use AUTOFILL.

4 The System for Analyzing Vote Counting Algorithms

Vote-counting algorithms treat votes as resources that must be counted one time and one time only. This requirement already suggests that linear logics are well-suited for representing voting-algorithms (see Section 2) and Criterion 1 (see Section 3). Our system of choice is therefore the logical framework Celf [8]. For the bounded model-checking part we have considered using model checkers and SMT solvers, at the end, 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

$$\Gamma; \Delta \vdash \texttt{run}\ i\ \texttt{C}\ \texttt{V}\ \texttt{S}\ w\ d$$

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 analyzed. 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` $A\ N$, summarizing 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 `C`, the number of cast votes `V`, the number of seats `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 $i \times$ `S`.

To save space, we present only one of the rules implementing STV. [4] gives an introduction on how to represent STV counting algorithms in Celf. 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 should be considered universally bound, we write —◦ for linear implication, ⊗ 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 is an implementation of a focused linear logic). The rule `count/2` can be read as a forward-chaining multiset rewrite rule. In the case no candidate reaches the !`quota` Q, the candidate with the least amount of 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* Γ = QUOTA/DROOP, AUTOFILL, TIE*, and let* Δ = ballot box + 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* behavior 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 $max_{\mathtt{C}}$, the maximal size of the ballot box $\max_{\mathtt{V}}$, and the maximal number of seats $max_{\mathtt{S}}$. Checking the algorithm for a particular input corresponds to running the query:

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

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 construct as many solutions to the above query as there are ballot boxes, we can conclude that the particular STV algorithm selected by Γ 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.[5]

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] This is not a critical limitation in practice. Although there are various ways of resolving non-determinism in STV, it is important to clearly specify how it is resolved in real-world elections i.e., the particular implementation of TIE. Otherwise, choices by the election officials (or their computers) when counting the ballot could influence the election result, which is clearly undesirable. One could change the checker to backtrack over election results, but that would greatly increase runtime.

5 Case Study: CADE-STV

The bylaws of the Conference on Automated Deduction (CADE) specify an algorithm for counting the ballots cast for the election of members to its Steering Committee [2]. The intention of the bylaws is to design a voting algorithm that takes the voters' preferences into account. The algorithm has been implemented and used by several CADE Presidents and Secretaries in elections for the CADE Board of Trustees. It has also been used by TABLEAUX Steering Committee Presidents, including one of the authors, for the election of members to the TABLEAUX Steering Committee. The CADE-STV specification shown in Appendix A gives an algorithmic specification for how to count ballots.

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.[6]

Example 3. Let us run CADE-STV on Example 1. First, we compute the majority quota $\mathtt{Q} = 3$. 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 $[A, B]$.

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* $\Gamma = \text{QUOTA/MAJORITY}, \text{RESTART}$ *and* $\Delta = \text{ballot box} +$ initialized running counts*, then* `run S C V 1` *w d is provable if and only if w corresponds to the list of candidates elected by the CADE-STV algorithm, 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 vote in exactly the same way* $[P_1, \ldots, P_k]$*, then CADE-STV will elect the candidates preferred by that majority in order of the majority's preference.*

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. □

[6] Presumably, QUOTA/MAJORITY was introduced into CADE STV following criticism of DROOP/QUOTA by David Plaisted [7].

Corollary 1. *If the electorate consists of two diametrically opposed camps that vote for their candidates only, in some fixed order, then the camp with a majority will always get their candidates elected and the camp with a minority will never get their candidate elected.*

Standard STV does not use RESTART (nor NODEL) and so it will elect the first ranked candidate of the majority, but will then distribute only the surplus votes and not all votes as done by CADE-STV. Thus the second preference from the majority is not necessarily the second person elected. Consequently, majorities do not rule outright in standard STV.

6 Conclusion

The experiments with our tool have have shown that the voting system of CADE does not satisfy the intuitive semantic criterion defined in Section 3. We discovered that the bylaws of the CADE community require elections to be decided by a single transferable vote algorithm that does not try to achieve proportional representation. We suspect that CADE's voting system was conceived with the intention 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. F. Brandt, V. Conitzer, and U. Endriss. Computational social choice. In G. Weiss, editor, *Multiagent Systems*. MIT Press, 2012. Forthcoming. Available at `http://www.illc.uva.nl/~ulle/pubs/files/BrandtEtAlMAS2012.pdf`.
2. CADE Inc. CADE Bylaws (effective Nov. 1, 1996; amended July/August 2000). Retrieved from `http://www.cadeinc.org/Bylaws.html`. Accessed 20 Jan 2013.
3. D. Cochran. *Formal Specification and Analysis of Danish and Irish Ballot Counting Algorithms*. PhD thesis, IT University of Copenhagen, 2012.
4. H. DeYoung and C. Schürmann. Linear logical voting protocols. In *Int. Conf. on E-Voting and Identity (VOTE-ID'11)*, LNCS 7187, pages 53–70. Springer, 2011.
5. D. Farkas and S. Nitzan. 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. Retrieved from `http://users.cecs.anu.edu.au/~rpg/EVoting/index.html`. Accessed 20 Jan 2013.
7. D. A. Plaisted. A consideration of the new CADE bylaws. Retrieved from `http://www.cs.unc.edu/Research/mi/consideration.html`. Accessed 20 Jan 2013.
8. A. Schack-Nielsen and C. Schürmann. Celf: A logical framework for deductive and concurrent systems (system description). In *Int. Joint Conf. on Automated Reasoning (IJCAR'08)*, LNCS, pages 320–326. Springer, 2008.
9. Wikipedia. Single transferable vote. Retrieved from `http://en.wikipedia.org/wiki/Single_transferable_vote`. Accessed 20 Jan 2013.

A Appendix: Official CADE-STV Specification

The following is an exact quote from the web page of CADE Inc. [2], where the STV algorithm to be used in CADE elections is described.

```
Problem:  M voters must elect K of N candidates.

Input:    M x N matrix, Tbl, of votes.
    Tbl(i,j) = p, 1<=p<=N,
        means that voter i gave preference p to candidate j.
    Every voter can support n (1<=n<=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...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 & redistrb. votes *}
end for

End STV
```
