Modeling Test Cases for Voting

Using the Alloy Model Finder to Derive Test Cases for PR-STV Elections

Dermot Cochran
Joseph R. Kiniry

IT University Technical Report Series **TR-2011-143**

ISSN 1600–6100 **September 2011**

ISSN 1600–6100

ISBN 978-87-7949-239-4

Copies may be obtained by contacting:

**IT University of Copenhagen
Rued Langgaards Vej 7
DK-2300 Copenhagen S
Denmark**

Telephone:	**+45 72 18 50 00**
Telefax:	**+45 72 18 50 01**
Web	`www.itu.dk`

Modeling Test Cases for Voting

Using the Alloy Model Finder to Derive Test Cases for PR-STV Elections

Dermot Cochran
Joseph R. Kiniry

Abstract

The ballot counting process for Proportional Representation by Single Transferable Vote (PR-STV) elections can be modelled formally using the Alloy model checker so as to cover all possible branches through the ballot counting algorithm.

We use the Alloy model finder to describe the elections in terms of scenarios, consisting of equivalence classes of possible outcomes for each candidate in the election, where each outcome represents one branch through the algorithm.

We show how test data is generated from a first order logic representation of the counting algorithm using the Alloy model finder. This process guarantees that we find the minimal number of ballots needed to test each scenario.

1 Introduction

The electoral process consists of various different stages, from voter registration, through vote casting and tallying, to the final declaration of results.

Some, but perhaps not all, aspects of the election process are apparently suitable for automation. For example, voter registration records can be stored in computer databases, and ballot counting can be done by machine. In Denmark, the final result of the election is calculated by a computer in the Danish Ministry of the Interior.

However, many attempts to introduce electronic counting of ballots have failed, or at least received much criticism, due to software and hardware errors, *including potential counting errors*, many of which are avoided through the appropriate use of formal methods and careful testing.

One of the potential advantages from automation is the *accuracy of vote counting*, so it is important to be able to prove that software can actually count ballots more accurately than the manual labour-intensive process of counting paper ballots by hand, especially for complex voting schemes, otherwise there would simply be no question of using electronic voting.

The security aspects of elections are an important but distinct concern, and are beyond the scope of this paper.

In this paper we will focus mainly on the Irish voting scheme, as a case study.

1.1 Voting Scheme

The Republic of Ireland uses Proportional Representation by Single Transferable Vote (PR-STV) for its national, local and European elections.[1] PR-STV is a multi-seat ranked choice voting system, in which each voter ranks the candidates from first to last preference.

[1] Ireland uses Instant Runoff Voting (IRV) for its presidential elections and for by-elections to fill casual vacancies in Dáil Éireann

Manual recounts are often called for closely contested seats, as the results often vary slightly, indicating small errors in the manual process of counting votes. Paper-based voting with counting by hand is popular in Ireland, and recent attempts at automation were frustrated by subtle logic errors in the ballot counting software [2]. The potential for logic errors exist, in part, due to the complexities and idiosyncrasies with regard to tie breaking, especially involving the rounding up or down of vote transfers.

There has been some desire in Ireland to simplify matters. Referenda to introduce plurality (first past the post) voting were rejected twice by the Irish electorate, once in 1959 and again in 1968 [14]. Since then, there have been no further legislative proposals to change the voting scheme used in Ireland.

The following are selected quotes from the Irish Commission on Electronic Voting (CEV) report on the previous electronic voting system used in Ireland (emphasis added) [4]:

- Design weaknesses, including an error in the implementation of the *count* rules that could compromise the accuracy of an election, have been identified and these have reduced the Commission's confidence in this software.
- The achievement of the full potential of the chosen system in terms of secrecy and accuracy depends upon a number of software and hardware modifications, both major and minor, and more significantly, is dependent on the *reliability of its software being adequately proven*.
- Taking account of the ease and relative cost of making some of these modifications, the potential advantages of the chosen system, once modified in accordance with the Commission's recommendations, can make it a *viable alternative to the existing paper system in terms of secrecy and accuracy*.

Thus, Ireland wishes to keep its current complicated voting scheme, is critical of the existing attempts to implement that scheme in e-voting, but keeps the door slightly ajar for the introduction of e-voting in the future.

1.1.1 Proportional Representation by Single Transferable Vote (PR-STV)

PR-STV achieves proportional representation in multi-winner elections, and reduces to IRV for single-winner elections.

The flowchart in Figure 1 outlines the algorithm used for counting preferences ballots by PR-STV. A quota of preferences is chosen so that at most $N-1$ candidates can reach the quota, where N is the number of seats to be filled. The threshold is always less than the quota. The surplus for a candidate is the number of votes in excess of the quota.

1.2 Vótáil

Vótáil is an open source Java implementation of Irish Proportional Representation by Single Transferable Vote (PR-STV) [9]. Its functional requirements, derived from Irish electoral law, are formally specified using the Business Object Notation (BON) and refined to a Java Modeling Language (JML) specification. Extended Static Checking (ESC) is used to help verify and validate the correctness of the software.

1.3 Related Work

Meagher wrote a Z and B specification for election to the board of Waterford Institute of Technology, which uses a variant of the Irish PR-STV system [13].

Kjölbro used a similar methodology for specification and implementation of the Danish Voting System [10].

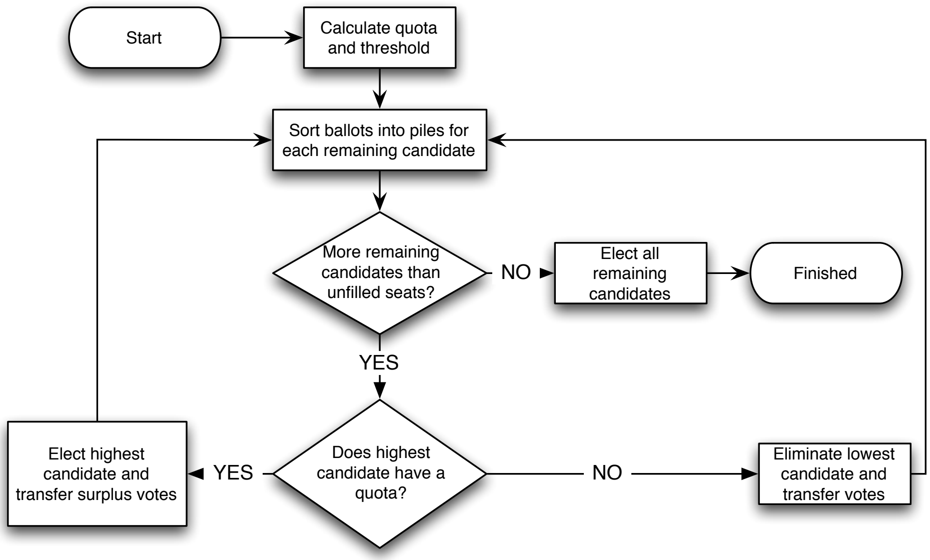

Figure 1: Proportional Representation by Single Transferable Vote

We are also aware of some unpublished or unfinished work relating to previous attempts at formalization of PR-STV, including some Prolog work by Naish and an implementation of the Scottish STV system in CLEAN by researchers at the Radboud University Nijmegen. The only peer-reviewed published related work of interest is a protocol for the tallying of encrypted STV ballots [15] and verifying properties of voting protocols, not software (e.g., several papers by Delaune et al [3]).

There is, of course, a large amount of work in the field of model checking and test generation, but not directly related to voting as a case study and therefore not referenced in this paper.

1.4 Outline of Paper

The next section of the paper describes voting schemes in more detail. The third section describes the system under test using a mathematical theory of ballots and ballot boxes. The fourth section describes the possible configurations of election results under each voting scheme. The fifth outlines the process of deriving test data needed for each election configuration. The final section contains our conclusions and plans for future work.

2 Formalisation

We must represent the input data space in a precise mathematical way to formally reason about its properties with respect to the algorithm.

2.1 Mathematical Models

In this case study, the core concepts of elections must be defined: *ballots*, *ballot boxes*, *candidates*, and *election results*.

Definition 1 (Candidate) Candidates *are individual persons standing for election. They are identified by (distinct) names. The set of all candidates is denoted $\mathcal{C}$. The Alloy encoding includes the following:* [2]

[2] The full definition can be found in the Appendices

```
sig Candidate {
  votes: set Ballot, -- First preference ballots
  transfers: set Ballot, -- Transfers received
  surplus: set Ballot, -- Ballots to be transferred
  wasted: set Ballot, -- Ballots non-transferabe
  outcome: Event -- Election outcome }
```

Definition 2 (Ballot) *An ordinal or preference* Ballot b *is a strict total order on a set of candidates* $\mathcal{C}$. *The length of a ballot,* $|b|$, *is the number of preferences expressed. The minimum number of preferences is one, except in systems like that used in Australia where all preferences must be used. The Alloy encoding is as follows:*

```
sig Ballot {
  assignees: set Candidate, -- Benificiaries of this ballot
  preferences: seq Candidate -- Ranking of candidates
} {
  assignees in preferences.elems
  not preferences.hasDups
  preferences.first in assignees
  Election.method = Plurality implies #preferences <= 1
  0 <= #preferences
  // First preference
  all c: Candidate | preferences.first = c iff this in c.votes
  // Second and subsequent preferences
  all disj donor,receiver: Candidate |
    (donor + receiver in assignees and
    this in receiver.transfers and this in donor.surplus) implies
    (preferences.idxOf[donor] < preferences.idxOf[receiver] and
     receiver in preferences.rest.elems)
  // Last candidate to receive the transfer
  all disj c,d: Candidate | this in c.transfers implies
    c in assignees and
    (d not in assignees or
    preferences.idxOf[d] < preferences.idxOf[c])
  // Transfers to next continuing candidate
  all disj skipped, receiving: Candidate |
    preferences.idxOf[skipped] < preferences.idxOf[receiving] and
    receiving in assignees and (not skipped in assignees) implies
    (skipped in Scenario.eliminated or
    skipped.outcome = SurplusWinner or
    skipped.outcome = AboveQuotaWinner or
    skipped.outcome = WinnerNonTransferable or
    skipped.outcome = QuotaWinnerNonTransferable or
    skipped.outcome = Winner or
    skipped.outcome = QuotaWinner)
}
```

Definition 3 (Ballot Box) *An* unordered ballot box *is a bag (multiset) of ballots; an* ordered ballot box *is a vector of ballots,* $[b_1 b_2 \ldots]$. *Both are* ballot boxes, *denoted* $\mathcal{B}$. *As a bag can be modeled by a vector where order does not matter, we only use the latter formalization in the following.* [3]

The Alloy encoding is as follows:

[3] An ordered ballot box is used to model voting schemes in which surplus ballots are chosen according to the order in which they have been shuffled and mixed

```
one sig BallotBox {
  spoiltBallots: set Ballot, -- empty ballots
  nonTransferables: set Ballot, -- preferences are exhausted
  size: Int -- number of unspolit ballots
}
{
  no b: Ballot | b in spoiltBallots and b in nonTransferables
  size = #Ballot - #spoiltBallots
  all b: Ballot | b in spoiltBallots iff #b.preferences = 0
  all b: Ballot | some c: Candidate | b in nonTransferables
    implies b in c.wasted
}
```

In the Alloy encoding the Ballot Box *contains those* Ballots *not assigned to one of the* Candidate *piles.*

Definition 4 (Outcome) *An* Outcome *represents the path through the algorithm for the pile of ballots initially assigned to that candidate. For example, if the ballots form a surplus or if some of the ballots are* non-transferable *due to exhaustion of preferences.*

```
enum Event {SurplusWinner,
            WinnerNonTransferable,
            Winner,
            AboveQuotaWinner,
            QuotaWinnerNonTransferable,
            QuotaWinner,
            CompromiseWinner,
            TiedWinner,
            TiedLoser,
            Loser,
            EarlyLoser,
            EarlyLoserNonTransferable,
            TiedSoreLoser,
            SoreLoser,
            EarlySoreLoser,
            EarlySoreLoserNonTransferable}
```

Definition 5 (Scenario) *A* Scenario *consists of the overall election results including the* Outcome *for each* Candidate.

```
one sig Scenario {
  losers: set Candidate,
  winners: set Candidate,
  eliminated: set Candidate,
  threshold: Int, -- Minimum number of votes for funding
  quota: Int, -- Maximum number of votes needed for election
  fullQuota: Int -- Quota for a full election
} {
  eliminated in losers
  ...
}
```

Definition 6 (Constituency) *A* Constituency *consists of a number of seats that represent a local area or region.*

```
one sig Election {
  seats: Int, -- number of seats to be filled in this election
  constituencySeats: Int, -- full number of seats in this constituency
  method: Method -- type of election; PR-STV or plurality
} {
 0 < seats and seats <= constituencySeats
 seats < #Candidate
 method = Plurality or method = STV
}
```

2.2 Number of Distinct Ballots

The number of distinct permutations of non-empty preferences is $\sum_{l=1}^{C} (C)_l$, where $C = |\mathcal{C}|$ and partial ballots are allowed, so that the number of preferences used range in length from one to the number of candidates. For a ballot of length l, $(C)_l$ is the number of distinct preferences that can be expressed. [4]

2.2.1 Examples and Encoding Ballots

This distinct ballot count is best understood, particularly for those unexcited by combinatorics, by examining cases for small C and enumerating all possible ballots.

Two Candidates There are four different ways to vote for two candidates (named Alice and Bob): two ballots of length 1, and two ballots of length 2, that is $(2)_1 + (2)_2$:

Ballot	Alice	Bob	Encoding of Ballot
1	1^{st}	-	$\boxed{A}\boxed{-}$
2	-	1^{st}	$\boxed{B}\boxed{-}$
3	1^{st}	2^{nd}	$\boxed{A}\boxed{B}$
4	2^{nd}	1^{st}	$\boxed{B}\boxed{A}$

$\boxed{A}\boxed{-}$ has a different meaning than $\boxed{A}\boxed{B}$. If we had an election with two ballots $\boxed{B}\boxed{-}$ and $\boxed{A}\boxed{B}$, then Bob would be the winner.

Note the symmetry of these four ballots. There are effectively only two different ballots if the candidates cannot be differentiated.

3 Election Outcomes

A naive approach to validating/testing electoral systems (if they are tested at all) is to randomly generate hundreds of thousands (or, indeed, even millions) of ballot boxes and then to compare the results of executing two or more different implementations of the same voting scheme. If different results are found, then the ballots are counted manually to determine which result is correct [2].

This methodology is inadequate because even if one generates billions of ballots in non-trivial election schemes, the fraction of the state space explored is vanishingly small. To make this fact clear, we will analyze the number of distinct ballot boxes in various schemes. [5]

[4] $\sum_{l=0}^{C} C!/(l-C)! = C! \sum_{l=0}^{C} 1/l! < e * C!$ In fact, $\sum 1/l!$ converges quite quickly to e and so the number of distinct ballots is $floor(e * C!)$. You can subtract 1 to get the number of nonempty, distinct ballots.

[5] further examples can be seen in Appendix 1

3.1 Last Two Continuing Candidates

When there are just two continuing candidates and one remaining seat, the algorithm reduces to single winner plurality (first-past-the-post).

In this case there are six possible election results (*candidate outcome events*) for each candidate:

Event	Description
$\mathbb{W}$	The candidate is the poll-topper with the most votes.
$\underline{\mathbb{W}}$	The candidate is joint highest and only wins by tie-breaker.
$\mathbb{L}$	The candidate loses, but receives enough votes to reach the threshold.
$\underline{\mathbb{L}}$	The candidate is joint highest and only loses by tie-breaker.
$\mathbb{S}$	The candidate loses and does not reach the threshold.
$\underline{\mathbb{S}}$	The candidate is joint highest and loses by tie-breaker, but does not reach the threshold.

In our vector representation, an event ϵ in entry i of the election scenario indicates that candidate i obtained outcome ϵ.

3.1.1 Scenarios

In plurality, there is only one winner, who wins either in event $\mathbb{W}$ or $\underline{\mathbb{W}}$.

Two Candidates If there is one loser, the 3 possible outcomes are:

Sub-Scenario	1^{st} Event	2^{nd} Event
1	$\mathbb{W}$	$\mathbb{L}$
2	$\mathbb{W}$	$\mathbb{S}$
3	$\underline{\mathbb{W}}$	$\underline{\mathbb{L}}$

3.2 Filling of Last Seat

When there is one remaining seat, but at least three continuing candidates, then the algorithm reduces to Instant Runoff Voting (IRV):

3.2.1 Events

For each continuing candidate the following event outcomes are possible:

Event	Description
$\mathbb{H}$	The candidate is the poll-topper with a majority of the first preferences and is elected.
$\mathbb{Q}$	The candidate is elected during an intermediate round by receiving transfers.
$\mathbb{W}$	The candidate receives enough transfers to have a majority of the votes and is elected in the last round.
$\underline{\mathbb{W}}$	The candidate is elected by tie-breaker in last round.
$\mathbb{L}$	The candidate is defeated as the lowest candidate in any round but reached the threshold.
$\underline{\mathbb{L}}$	The candidate is defeated by tie-breaker in any round, but reached the threshold.
$\mathbb{S}$	The candidate is excluded as the lowest candidate in any round and did not reach the threshold.

Event	Description	Alloy Encoding
$\mathbb{N}$	The candidate is elected in the first round with a surplus containing at least one non-transferable vote	WinnerNonTransferable
$\mathbb{T}$	The candidate is elected in the first round with at least one surplus vote	SurplusWinner
$\mathbb{H}$	The candidate is elected in the first round without surplus votes	Winner
$\mathbb{X}$	The candidate is elected after receiving vote transfers and then has a surplus with at least one non-transferable vote	QuotaWinnerNonTransferable
$\mathbb{A}$	The candidate is elected during an intermediate round by receiving transfers and has a surplus to distribute	AboveQuotaWinner
$\mathbb{Q}$	The candidate is elected during an intermediate round by receiving transfers, but without a surplus	QuotaWinner
$\mathbb{W}$	The candidate is elected as the highest continuing candidate on last round.	CompromiseWinner
$\underline{\mathbb{W}}$	The candidate is elected by tie-breaker on the last round.	TiedWinner

Figure 2: Winning Outcomes for PR-STV

3.2.2 Sub-Scenarios

Two Candidates If we consider two candidates, the winner and the highest loser (runner-up) than the following combinations of events are possible:

1^{st} Event	2^{nd} Event	Description
$\mathbb{W}$	$\mathbb{L}$	The winner gets a majority and the loser reaches the threshold.
$\mathbb{W}$	$\mathbb{S}$	The winner gets a majority and loser does not reach the threshold.
$\underline{\mathbb{W}}$	$\underline{\mathbb{L}}$	The winner is elected by tie-breaker and the loser reaches the threshold.

3.3 PR-STV

Figure 2 shows the eight winning outcomes and Figure 3 shows the eight losing outcomes.

4 Properties of the Model

The model contains 6 type signatures, 53 appended definitions, 2 enumerated types and 37 lemmas e.g.

Lemma 1 *The events $\mathbb{W}$and $\underline{\mathbb{W}}$ are mutually exclusive.*

Lemma 2 *Every Tied Winner has the same number of votes as every Tied Loser.*

```
assert equalityofTiedWinnersAndLosers {
  all disj w,l: Candidate | w in Scenario.winners and l in Scenario.losers and
    w.votes + w.transfers = l.votes + l.transfers implies
    w.outcome = TiedWinner and
    (l.outcome = TiedLoser or l.outcome = TiedSoreLoser) }
```

Event	Description	Alloy Encoding
$\mathbb{L}$	The candidate is defeated as the lower continuing candidate on the last round.	Loser
$\underline{\mathbb{L}}$	The candidate is defeated by tie-breaker on last round.	TiedLoser
$\mathbb{E}$	The candidate is excluded as the lowest candidate in an earlier round but reached the threshold, all ballots are transferable	EarlyLoser
$\mathbb{D}$	The candidate is excluded in an earlier round and is below the threshold, all ballots are transferable	EarlySoreLoser
$\mathbb{S}$	The candidate is defeated in the last round and is below the threshold.	SoreLoser
$\underline{\mathbb{S}}$	The candidate is excluded by tie-breaker and is below the threshold	TiedSoreLoser
$\mathbb{F}$	The candidate is excluded as the lowest candidate in an earlier round but reached the threshold, with at least one non-transferable ballot	EarlyLoserNonTransferable
$\mathbb{U}$	The candidate is excluded in an earlier round and is below the threshold with at least one non-transferable ballot	EarlySoreLoserNonTransferable

Figure 3: Losing Outcomes for PR-STV

5 Procedure for Automated Test Generation

We used the SAT4J solver with Alloy running concurrently in a thread pool. We suspect that a native solver would be faster, but might not be thread safe. [6]

Ballot counting system tests can be identified and generated in a complete and formal way, complementing existing hand-written unit tests. To accomplish this task, one needs to be able to generate the ballots in each distinct kind of ballot box identified using the results of the earlier sections of this paper. Effectively, the question is one of, "Given the election outcome R, what is a legal set of ballots B that guarantees R holds?"

5.1 Generation of Ballot Boxes

We outline a simple example to show how it is possible to derive test data from the equivalence class of ballot boxes..

Recall that each election outcome $\mathcal{O}$ is described by a single *election scenario*, $\mathcal{S}$, as described by a vector of *candidate outcome events*. We must derive from an outcome $\mathcal{O}$ a vector of ballots $\mathcal{B}$ that guarantee, when counted using the ballot counting algorithm of the election, exactly $\mathcal{O}$, assuming that ties are broken in a deterministic way. We write $\mathcal{B} \vdash_{\mathcal{S}} \mathcal{O}$ to mean counting $\mathcal{B}$ results in outcome $\mathcal{O}$ under scenario $\mathcal{S}$. Such a combination of ballots, outcome, and scenario is called an *election outcome configuration*.

In general, there are a large number of vectors of ballots that guarantee an election outcome. For practical reasons in validation, we wish to find the *smallest* vector that guarantees the outcome; i.e., given $\mathcal{O}$ and $\mathcal{S}$, find $\mathcal{B}$ such that $\forall b . b \vdash_{\mathcal{S}} \mathcal{O} . |\mathcal{B}| \leq |b|$.

For a given outcome $\mathcal{O}$, the conditions that a vector of ballots $\mathcal{B}$ must meet to fulfill scenario $\mathcal{S}$ is described using a first-order logical formula whose validity indicates $\mathcal{B} \vdash_{\mathcal{S}} \mathcal{O}$

[6] See `http://alloy.mit.edu/community/node/1080` for an explanation of why JNI solvers might not be thread safe.

holds. We denote this description Φ. Thus, $\mathcal{B} \vdash_{\mathcal{S}} \mathcal{O} \Leftrightarrow \Phi(\mathcal{B})$, or alternatively, $\Phi(\mathcal{B})\mathcal{B} \vdash_{\mathcal{S}} \mathcal{O}$.

Encoding in Alloy Modeling Language Formally this is achieved using bounded checks in the Alloy Analyser [8].

Informally, to find the minimal sized $\mathcal{B}$, we iteratively describe election configurations $\mathcal{B} \vdash_{\mathcal{S}} \mathcal{O}$ with monotonically increasing numbers of ballots, starting with a ballot box of size one. These descriptions consist of a set of definitions that describe the outcome and a single theorem that states that $\mathcal{O}$ is *not* possible. If the number of ballots is too small to produce the desired outcome, then the formulation of $\mathcal{B} \vdash_{\mathcal{S}} \mathcal{O}$ will be inconsistent, and Alloy will return a satisfiable solution.'

Alternatively, if the ballot box size is just large enough, Alloy will insist that the predicate is *invalid* and provide a counterexample proof context, whose values indicate the necessary values of all of the ballots in $\mathcal{B}$.

Example: Instant Runoff Voting Consider 3 candidate IRV. Two possible outcome classes are QLE and CLE–no candidate has a majority so one is eliminated and then in the next round, one candidate has a majority. These are two distinct cases: firstly a ballot box of 3 ballots for A 2 ballots for B 1 ballot for C

and secondly a ballot box of 2 ballots for A 2 ballots for B 1 ballot with (1st=C 2nd=A).

In both cases, no one has a majority, C is eliminated, and then A wins with a 3 to 2 majority. In both cases the threshold would be one vote. In both cases C is an Early Loser ($\mathbb{E}$) and B is a Loser ($\mathbb{L}$).

5.1.1 An Election Configuration Example

Consider a plurality election with two candidates ($|\mathcal{C}| = 2$). As discussed in Section 3.1.1, there are three scenarios associated with this election configuration: $[\mathbb{W}\mathbb{L}]$, $[\mathbb{W}\mathbb{S}]$, and $[\underline{\mathbb{W}\mathbb{L}}]$.

In the following, let be T be a tiebreaker function that chooses a winner from a set of candidates.

As earlier, let $\mathcal{B}$ denote a ballot box and b a ballot. Let $b[n]$ be the n^{th} preference of ballot b. Finally, as earlier, let τ be the threshold of votes for a given electoral system.

5.1.2 Formalization

Each candidate outcome is described by an definition that expresses the relationship between the number of votes that candidate receives and the outcome. Since most first-order theorem provers do not provide native support for the generalized summation quantifier, we use a generic encoding described by Leino and Monahan [11].

The Scenario Predicate Now, we wish to try to prove a predicate that stipulates that, for a given scenario, an expected outcome is *not* possible for a given number of ballots.

We ask the solver to check the validity of the following predicate (by simply stating the predicate in Alloy that captures the meaning of scenario $[\mathbb{W}\mathbb{L}]$:

$$|\mathcal{B}| = 1 \Rightarrow \neg(\mathbb{W} \wedge \mathbb{L})$$

If the prover responds with "valid," then we know that we need more than one ballot, and we make a new attempt:

$$|\mathcal{B}| = 2 \Rightarrow \neg(\mathbb{W} \wedge \mathbb{L})$$

Consequently, if that attempt also fails, we attempt to prove the theorem with three ballots:

$$|\mathcal{B}| = 3 \Rightarrow \neg(\mathbb{W} \wedge \mathbb{L})$$

at which time the prover returns an "invalid" response with a counterexample. The counterexample for this particular theorem will be of the form

$$b[1][1] = A \land b[2][1] = A \land b[3][1] = B$$

thereby providing a minimal ballot box that guarantees election outcome $[\mathbb{W}\mathbb{L}]$. Note that to check minimality we can attempt to prove the theorem $(\mathbb{W} \land \mathbb{L}) \Rightarrow 3 \leq |\mathcal{B}|$, though such a theorem is quite difficult for automated solvers to prove give the implicit quantification over ballot boxes and is, in general, can only be proven with an interactive theorem prover.

5.2 Open Source Implementation

The source code is open source, under the terms of the MIT open source license, and is available via our Trac server.[7]. The source code is managed using a subversion server hosted on our website[8]

6 Results and Conclusions

We have used our methodology to test Vótáil, achieving full line coverage with only seven candidates in a three seat election, and discovered two errors in its implementation, namely a null pointer exception and possible non-termination of a loop. These were not caught during the original verification of Vótáil, due to under-specification i.e., a missing loop invariant.

Acknowledgments

We generated our test data on a server farm hosted by University College Dublin, Ireland.

The authors would also like to thank Josu Martinez and Daniel M. Zimmerman for their comments.

References

[1] S. Bowler and B. Grofman. *Elections in Australia, Ireland, and Malta under the Single Transferable Vote: Reflections on an embedded institution*. Univ of Michigan Pr, 2000.

[2] L. Coyle, P. Cunnigham, and D. Doyle. Secrecy, accuracy and testing of the chosen electronic voting system: Reliability and accuracy of data inputs and outputs, December 2004.

[3] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Verifying properties of electronic voting protocols. In *Proceedings of IAVoSS Workshop On Trustworthy Elections (WOTE)*, 2006.

[4] Department of Environment and Local Government, Commission on Electronic Voting. Final Report of Commission on Electronic Voting, July 2006.

[5] D.M. Farrell and I. McAllister. *The Australian electoral system: origins, variations, and consequences*. New South Wales University Press, Ltd., 2006.

[6] M. Gallagher. Comparing proportional representation electoral systems: Quotas, thresholds, paradoxes and majorities. *British Journal of Political Science*, 22(04):469–496, 1992.

[7] `https://trac.ucd.ie`

[8] `https://trac.ucd.ie/repos/software/evoting`

[7] J. Gilmour. Detailed description of the STV count in accordance with the rules in the Scottish local government elections order 2007. *Representation*, 43(3):217–229, 2007.

[8] D. Jackson. *Software Abstractions: logic, language and analysis*. The MIT Press, 2006.

[9] J.R. Kiniry, D. Cochran, and P.E. Tierney. Verification-centric realization of electronic vote counting. In *Proceedings of the USENIX/Accurate Electronic Voting Technology on USENIX/Accurate Electronic Voting Technology Workshop*, pages 6–6. USENIX Association Berkeley, CA, USA, 2007.

[10] Olavur Kjölbro. Verifying the Danish Voting System. Master's thesis, IT University of Copenhagen, May 2011.

[11] K. Rustan M. Leino and Rosemary Monahan. Reasoning about comprehensions with first-order SMT solvers. In *Proceedings of the 24th Annual ACM Symposium on Applied Computing (SAC'09)*, 2009.

[12] M. McGaley and J.P. Gibson. Electronic voting: A safety critical system. *Final Year Project Report, NUI Maynooth Department of Computer Science*, 2003.

[13] Mairead Meagher. Towards the development of an electronic count system using formal methods, 2001. MPhil thesis, University of Southampton.

[14] R. Sinnott. *Irish voters decide: Voting behaviour in elections and referendums since 1918*. Manchester Univ Pr, 1995.

[15] Vanessa Teague, Kim Ramchen, and Lee Naish. Coercion-resistant tallying for STV voting. In *Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop*, 2008.

A Appendix: Voting Schemes

A *voting scheme* is an algorithm for counting ballots. A *preference voting scheme* requires the voter to rank two or more candidates ($\mathcal{C}$) in order of preference from first to last. A *plurality voting scheme* requires the voter to pick one candidate, and thus is equivalent to the preference scheme when the ranking list has unitary size.

The *election result* $(\mathcal{W}, \mathcal{L})$ consists of (1) the identification of the winner or winners of the election and (2) the identification of those candidates who achieved a certain *threshold* (denoted τ) of votes, e.g., 5 percent, needed either to qualify for public funding in future elections or to recoup a deposit paid.[9] Note that winners and losers are disjoint.

We denote a ballot box $\mathcal{B}$ as a set of ballots b. Mathematically, a voting scheme $\mathcal{E}$ is a function that takes a ballot box (a set of ballots) as its input, and produces an election result as its output. More formally, $\mathcal{E} : \mathcal{B} \rightarrow (\mathcal{W}, \mathcal{L})$ where $\mathcal{W} \subseteq \mathcal{C}$, $\mathcal{L} \subset \mathcal{C}$, and $\mathcal{W} \cap \mathcal{L} = \emptyset$.

A.0.1 Single Winner Plurality Voting

Plurality voting is one of the simplest possible voting schemes. The candidate with the most votes is the winner. When there is only one remaining seat and just two continuing candidates, then PR-STV reduces to single-winner Plurality.

A.0.2 Instant Runoff Voting (IRV)

IRV allows the voter to rank one or more candidates in order of relative preference, from first to last.

IRV usually has a single winner, but the candidate with the most votes must also have a majority of all votes, otherwise the candidate with least votes is excluded and each ballot for that candidate is transferred to the next candidate in order of preference. This evaluation-and-transfer continues until one of the candidates achieves an overall majority.

When there is just one remaining seat, or a special election to fill a vacancy in one seat, then PR-STV reduces to IRV.

Order of Elimination The candidate with the least number of votes credited to him or her in the curent round is selected for elimination. If there is an equality of votes, then previous rounds are considered. If two or more candidates have equal lowest votes in all rounds, then random selection is used.

Variants of PR-STV To highlight the complexities of election schemes, consider the following variants of PR-STV. As schemes vary, so must testing/validation strategies. For example, Australia, Ireland, Malta, Scotland, and Massachusetts use different variants of PR-STV for their elections [1].

- **Australia** - Australia uses IRV to elect its House of Representatives and an open list system for its Senate, where voters can choose either to vote for individual candidates using PR-STV or to vote "above-the-line" for a party. If voters choose to use PR-STV then all available preferences must be used [5].
- **Ireland** - Ireland uses PR-STV for local, national and European elections. Transfers are rounded to the nearest whole ballot, so the order in which ballots are transferred makes a difference to the result [12]. Not all preferences need to be used, so voters may choose to use only one preference, as in Plurality voting, if desired.

[9] This threshold facet of our election model is not universal, but is a critical component in many electoral systems.

- **Malta** - Malta uses PR-STV for local, national and European elections. For national elections Malta also adds additional members so that the party with the most first preference votes is guaranteed a majority of seats.
- **Scotland, UK** - Scotland uses PR-STV for local elections. Rather than randomly select which ballots to include in the surplus, fractions of each ballot are transferred, that gives a more accurate result but takes much longer to count if counted by hand [7].
- **Massachusetts, USA** - Cambridge in Massachusetts uses PR-STV for city elections. Candidates with less than fifty votes are eliminated in the first round and surplus ballots are chosen randomly.

The fact that a single complex voting scheme like PR-STV has this many variants in use highlights the challenges in reasoning about and validating a given software implementation. This fact makes our work that much more valuable, as each algorithm only need be analyzed *once* to derive a complete validation that may be used again and again over arbitrary implementations of a ballot counting algorithm.

A.0.3 Irish PR-STV

To give context, we now discuss the mechanics of Irish PR-STV in more detail.

Preference Ballots The voter writes the number “1” beside his or her favorite candidate. There can only be one first preference.

The voter then considers which candidate would be his or her next preference if his or her favorite candidate is either excluded from the election or is elected with a surplus of votes.

The second preference is marked with “2” or some equivalent notation. The can be only one second preference; there cannot be a joint second preference. Likewise for third and subsequent preferences. Not all preferences need to be used.

Multi-seat constituencies Each constituency is represented by either three, four or five seats.

The Droop Quota The quota is calculated so that not all winners can reach the quota. The droop quota is $1 + \frac{V}{1+S}$, where V is the total number of valid votes cast and S is the number of vacancies (or seats) to be filled [6]. The quota is chosen so that any candidate reaching the quota is automatically elected, and so that the number of candidates that might reach the quota less than the number of seats.

For example, in a five-seat constituency a candidate needs just over one-sixth of the total vote to be assured of election.

Surplus The surplus for each candidate, is the number of ballots in excess of the quota (if any). The surplus ballots are then available for redistribution to other continuing candidates.

The selection of which ballots belong to the surplus is a complex issue, depending on the round of counting. In the first round of counting, any surplus is divided into sub-piles for each second preference, so that the distribution of the ballots in the surplus is proportional to the second-preferences. In later rounds the surplus is taken from the last parcel of ballots received from other candidates. This surplus is then sorted into sub-piles according to the next available preference.

For example, if the quota is 9,000 votes and candidate A receives 10,000 first preference votes. The surplus is 1,000 votes. Suppose 5,000 ballots had candidate B as next preference, 3,000 had candidate C and 2,000 had candidate D. Then the surplus consists of

500 ballots taken from the 5000 for candidate B, 300 from the 3000 for candidate C and 200 from the 2000 for candidate D. Ideally each subset would also be sorted according to third and subsequent preference, but this does not happen under the current procedure for counting by hand, nor was it mandated in the previous guidelines for electronic voting in Ireland.

Exclusion of weakest candidates When there are more candidates than available seats, and all surplus votes have been distributed, the continuing candidate with least votes is excluded. If two or more candidates have equal lowest votes (at all stages of the count) then one is chosen randomly for exclusion.

All ballots from the pile of the excluded candidate are then transferred to the next preference for a continuing candidate, or to the pile of non-transferable votes.

This continues until another candidate is elected with a surplus or until the number of continuing candidates equals the number of remaining seats.

Filling of Last Seat and Bye-elections When there is only one seat remaining to be filled, i.e., the number of candidates having so far reached the quota is one less than the number of seats, or in a bye-election for a single vacancy, then the algorithm becomes the same as Instant Runoff Voting; no more surplus distributions are possible, and candidates with least votes are excluded until only two remain.

Last Two Continuing Candidates When there are two continuing candidates and one remaining seat, then the algorithm becomes the same as single-seat first-past-the-post plurality; the candidate with more votes than the other is deemed elected to the remaining seat, without needing to reach the quota. If there is a tie then one candidate is chosen randomly.

Axiomatization As a ballot is a vector, a *Ballot Box* is encoded as a matrix, where each column represents a single ballot. In such a representation, the top row of the matrix identifies the first preference candidate for each ballot. Each following row contains either a dash ('-'), meaning no preference, or the identifier of the next preference candidate. [10]

We first need definitions that stipulate the well-formedness of ballots.

$$\forall b \in \mathcal{B} \,.\, b[1] \in |\mathcal{C}|$$

$$(\sum_{\mathcal{B}} b[1] = A) + (\sum_{\mathcal{B}} b[1] = B) = |\mathcal{B}|$$

Definition wf_b describes the well-formedness of ballots, while definition $\mathrm{wf}_{\mathcal{B}}$ describes the well-formedness of the ballot box. If an electoral system permits empty preferences then this latter definition is modified to accommodate such.

Formalizing Scenarios Next, we need to formalize the scenarios of this particular two candidate plurality election as follows, where the label of each formula indicates the semantics of event of the same name e.g., formula $\mathbb{W}$ describes the meaning of event $\mathbb{W}$.

As we commonly quantify over all ballots in $\mathcal{B}$, we write the quantifications over $\mathcal{B}$ rather than the more wordy $b \in \mathcal{B}$. Finally, we encode the set of ballots as the first index in the map b i.e., the second ballot's third preference is $b[2][3]$. Note that these summations are generalized quantifiers: $\sum(b[1] = A)$ means "count the number of ballots whose first preference is candidate A."

[10] Such a representation in our implementation lends itself to nice datatype properties for composition, space usage, novel counting algorithm representations, etc.

$$\sum_{\mathcal{B}}(b[1]=A) > \sum_{\mathcal{B}}(b[1]=B) \tag{$\mathbb{W}$}$$

$$\sum_{\mathcal{B}}(b[1]=A) = \sum_{\mathcal{B}}(b[1]=B) \wedge (T=A) \tag{$\underline{\mathbb{W}}$}$$

$$\tau \leq \sum_{\mathcal{B}}(b[1]=B) \tag{$\mathbb{L}$}$$

$$\sum_{\mathcal{B}}(b[1]=B) < \tau \tag{$\mathbb{S}$}$$

Note that the rightmost clause of formula $\underline{\mathbb{W}}$ states that the coin-flip function picked candidate one as the winner.

B Appendix: Detailed Examples

This appendix contains some more detailed examples for estimation the number of possible outcomes and number of distinct permutations of ballot papers

B.0.4 Number of Distinct Outcomes

$sum_{l=0}^{C} C!/(l-C)! = C! sum_{l=0}^{C} 1/l! < e * C!$

If B is the number of distinct non-empty ballots that can be cast, and $V = |\mathcal{B}|$ is the number of votes cast, then the number of possible combinations of ballots is B^V if the order of ballots is important, and $\frac{B^V}{V!}$ if not.

A typical electoral configuration in Ireland is a five seat constituency with a typical voting population of 100,000 and 24 candidates. Consequently, the number of possible ballot boxes is $(\sum_{l=1}^{24}(24)_l)^{100,000}$, an astronomical number of tests that would be impossible to run.

To avoid this explosion, we partition the set of all possible ballot boxes into equivalence classes with respect to the counting algorithm chosen. We consider the equivalence class of election results for all three counting schemes.

Each election outcome is described by an *election scenario* that is a vector of *candidate outcome events*. Both of these terms are defined in the following.

The key idea is that election scenarios represent an equivalence class of election outcomes, thereby letting us collapse the testing state space due to symmetries in candidates. We will return to this point in detail below in the early examples.

Three Candidates There are 15 legal ways to vote for three candidates called Alice, Bob, and Charlie:

Ballot	Alice	Bob	Charlie	Encoding
1	1^{st}	-	-	$\boxed{A}\boxed{-}\boxed{-}$
2	-	1^{st}	-	$\boxed{B}\boxed{-}\boxed{-}$
3	-	-	1^{st}	$\boxed{C}\boxed{-}\boxed{-}$
4	1^{st}	2^{nd}	-	$\boxed{A}\boxed{B}\boxed{-}$
5	1^{st}	-	2^{nd}	$\boxed{A}\boxed{C}\boxed{-}$
6	2^{nd}	1^{st}	-	$\boxed{B}\boxed{A}\boxed{-}$
7	-	1^{st}	2^{nd}	$\boxed{B}\boxed{C}\boxed{-}$
8	2^{nd}	-	1^{st}	$\boxed{C}\boxed{A}\boxed{-}$
9	-	2^{nd}	1^{st}	$\boxed{C}\boxed{B}\boxed{-}$
10	1^{st}	2^{nd}	3^{rd}	$\boxed{A}\boxed{B}\boxed{C}$
11	1^{st}	3^{rd}	2^{nd}	$\boxed{A}\boxed{C}\boxed{B}$
12	2^{nd}	1^{st}	3^{rd}	$\boxed{B}\boxed{A}\boxed{C}$
13	3^{rd}	1^{st}	2^{nd}	$\boxed{B}\boxed{C}\boxed{A}$
14	2^{nd}	3^{rd}	1^{st}	$\boxed{C}\boxed{A}\boxed{B}$
15	3^{rd}	2^{nd}	1^{st}	$\boxed{C}\boxed{B}\boxed{A}$

There are 3 ballots of length 1, 6 ballots of length 2 and 6 ballots of length 3, that totals $(3)_1 + (3)_2 + (3)_3 = 15$. Again, note the symmetry of these ballots, as there are only three different kinds of ballots in these fifteen ballots.

More than Three Candidates Each additional candidate number n means one extra ballot of length 1, plus another C ballots in which the extra candidate is the last preference, plus every other way in which the candidate could be inserted into the existing set of ballots, in one of n positions along that ballot.

For example, when there are four candidates, the number of single preference ballots increases to 4, the number of length 2 ballots is $4 \times (4-1)$, the number of length 3 ballots is $4 \times (4-1) \times (4-2)$ and the number of full length ballots is $4!$, for a total of 64 ballots, of which there are only three equivalence classes.

C Appendix: Alloy Model

```
enum Event {SurplusWinner,
            WinnerNonTransferable,
            Winner,
            AboveQuotaWinner,
            QuotaWinnerNonTransferable,
            QuotaWinner,
            CompromiseWinner,
            TiedWinner,
            TiedLoser,
            Loser,
            EarlyLoser,
            EarlyLoserNonTransferable,
            TiedSoreLoser,
            SoreLoser,
            EarlySoreLoser,
            EarlySoreLoserNonTransferable}

enum Method {Plurality, STV}
```

```

-- An individual person standing for election
sig Candidate {
  votes:        set Ballot,
  -- First preference ballots received
  transfers:    set Ballot,
  -- Second and subsequent preferences received
  surplus:      set Ballot,
  -- Ballots tranferred to another candidate
  wasted:       set Ballot,
  -- Ballots non-transferable
  outcome:      Event
} {
  0 < #wasted iff (
    outcome = WinnerNonTransferable or
    outcome = QuotaWinnerNonTransferable or
    outcome = EarlyLoserNonTransferable or
    outcome = EarlySoreLoserNonTransferable)

  no b:Ballot | b in votes & transfers

  all b: Ballot | b in votes + transfers implies
    this in b.assignees

  surplus in votes + transfers and
    Election.method = Plurality
    implies #surplus = 0
    and #transfers = 0

  0 < #transfers implies
    Election.method = STV

  -- Losers excluded but above threshold
  (outcome = EarlyLoser or
    outcome = EarlyLoserNonTransferable) iff
    (this in Scenario.eliminated and
    not (#votes + #transfers < Scenario.threshold))

  outcome = TiedLoser implies
    Scenario.threshold <= #votes + #transfers
  outcome = Loser implies
    Scenario.threshold <= #votes + #transfers
  outcome = EarlyLoser implies
    Scenario.threshold <= #votes + #transfers
  outcome = EarlyLoserNonTransferable implies
  Scenario.threshold <= #votes + #transfers

  Election.method = Plurality implies
    (outcome = Loser or
    outcome = SoreLoser or
    outcome = Winner or
    outcome = TiedWinner or
    outcome = TiedLoser or
    outcome = TiedSoreLoser)
```

```
// PR-STV Winner has at least a quota of first preference votes
(Election.method = STV and outcome = Winner) implies
  Scenario.quota = #votes
(outcome = SurplusWinner or outcome = WinnerNonTransferable)
  implies Scenario.quota < #votes

// Quota Winner has a least a quota of votes after transfers
outcome = QuotaWinner implies
  Scenario.quota = #votes + #transfers
(outcome = AboveQuotaWinner or
 outcome = QuotaWinnerNonTransferable)
  implies Scenario.quota < #votes + #transfers

// Quota Winner does not have a quota of first preference votes
    (outcome = QuotaWinner or
     outcome = AboveQuotaWinner or
     outcome = QuotaWinnerNonTransferable) implies
                  not Scenario.quota <= #votes

   // Compromise winners do not have a quota of votes
          outcome = CompromiseWinner  implies
                  not (Scenario.quota <= #votes + #transfers)

  // STV Tied Winners have less than a quota of votes
         (Election.method = STV and outcome = TiedWinner) implies
                  not (Scenario.quota <= #votes + #transfers)

  // Sore Losers have less votes than the threshold
         (outcome = SoreLoser or
   outcome = EarlySoreLoserNonTransferable or
   outcome = EarlySoreLoser or outcome =
   EarlySoreLoserNonTransferable)
   implies #votes + #transfers < Scenario.threshold

  // Tied Sore Losers have less votes than the threshold
         outcome = TiedSoreLoser implies
                 #votes + #transfers < Scenario.threshold

  // Size of surplus for each STV Winner and Quota Winner
         (outcome = SurplusWinner or outcome = WinnerNonTransferable)
    implies ((#surplus = #votes - Scenario.quota) and #transfers = 0)
  (outcome = AboveQuotaWinner or outcome = QuotaWinnerNonTransferable)
    implies (#surplus = #votes + #transfers - Scenario.quota)
  (outcome = Winner and Election.method = STV) implies
    (Scenario.quota + #surplus = #votes) and #transfers = 0
     (outcome = QuotaWinner or outcome = AboveQuotaWinner or
    outcome = QuotaWinnerNonTransferable) implies surplus in transfers
     (outcome = QuotaWinner or outcome = AboveQuotaWinner or
    outcome = QuotaWinnerNonTransferable) implies
                 Scenario.quota + #surplus = #votes + #transfers

   // Existance of surplus ballots
   0 < #surplus implies (outcome = SurplusWinner or
```

```
        outcome = AboveQuotaWinner or
        outcome = WinnerNonTransferable or
        outcome = QuotaWinnerNonTransferable)
}

-- An accurate records of the intentions of the voter
sig Ballot {
  assignees: set Candidate, -- benficiaries of this ballot
  preferences: seq Candidate -- Ranking of candidates
} {
  assignees in preferences.elems
  not preferences.hasDups
  preferences.first in assignees
  Election.method = Plurality implies #preferences <= 1
  0 <= #preferences
  // First preference
  all c: Candidate | preferences.first = c iff this in c.votes
  // Second and subsequent preferences
  all disj donor,receiver: Candidate |
    (donor + receiver in assignees and
    this in receiver.transfers and this in donor.surplus) implies
    (preferences.idxOf[donor] < preferences.idxOf[receiver] and
    receiver in preferences.rest.elems)
  // All ballot transfers are associated with the last candidate
  all disj c,d: Candidate | this in c.transfers implies
    c in assignees and
    (d not in assignees or
    preferences.idxOf[d] < preferences.idxOf[c])
  // Transfers to next continuing candidate
  all disj skipped, receiving: Candidate |
    preferences.idxOf[skipped] < preferences.idxOf[receiving] and
    receiving in assignees and (not skipped in assignees) implies
    (skipped in Scenario.eliminated or
    skipped.outcome = SurplusWinner or
    skipped.outcome = AboveQuotaWinner or
    skipped.outcome = WinnerNonTransferable or
    skipped.outcome = QuotaWinnerNonTransferable or
    skipped.outcome = Winner or
    skipped.outcome = QuotaWinner)
}

-- An election result
one sig Scenario {
  losers: set Candidate,
  winners: set Candidate,
  eliminated: set Candidate, -- Early and Sore Losers under STV rules
  threshold: Int, -- Minimum number of votes for a Loser or Early Loser
  quota: Int, -- Minimum number of votes for a STV Winner or Quota Winner
  fullQuota: Int -- Quota if all constituency seats were vacant
} {
  all c: Candidate | c in winners + losers
  #winners = Election.seats
  no c: Candidate | c in losers & winners
  0 < #losers
```

```
all w: Candidate | all l: Candidate | l in losers and
  w in winners implies
  (#l.votes + #l.transfers <= #w.votes + #w.transfers)
Election.method = STV implies threshold = 1 + fullQuota.div[4]
  eliminated in losers
// All PR-STV losers have less votes than the quota
all c: Candidate | (c in losers and Election.method = STV) implies
  #c.votes + #c.transfers < quota
// Winners have more votes than all non-tied losers
all disj c,d: Candidate | c in winners and
  (d.outcome = SoreLoser or d.outcome = EarlyLoser or
  d.outcome = Loser or
  d.outcome = EarlySoreLoser) implies
  (#d.votes + #d.transfers) < (#c.votes + #c.transfers)
// Losers have less votes than all non-tied winners
all disj c,d: Candidate |
  (c.outcome = CompromiseWinner or
  c.outcome = QuotaWinner or c.outcome = Winner
  or c.outcome = SurplusWinner or
  c.outcome = AboveQuotaWinner or
  c.outcome = WinnerNonTransferable or
  c.outcome = QuotaWinnerNonTransferable) and
  d in losers implies
  #d.votes + #d.transfers < #c.votes + #c.transfers

// Lowest candidate is eliminated first
all disj c,d: Candidate | c in eliminated and
  d not in eliminated implies
  #c.votes + #c.transfers <= #d.votes + #d.transfers

 // A non-sore plurality loser must have received
 //at least five percent of the total vote
 Election.method = Plurality implies
   threshold = 1 + BallotBox.size.div[20]

 // Winning outcomes
 all c: Candidate | c in winners iff
   (c.outcome = Winner or c.outcome = QuotaWinner or
   c.outcome = CompromiseWinner or
   c.outcome = TiedWinner or c.outcome = SurplusWinner or
   c.outcome = AboveQuotaWinner or
   c.outcome = WinnerNonTransferable or
   c.outcome = QuotaWinnerNonTransferable)

// Losing outcomes
all c: Candidate | c in losers iff
  (c.outcome = Loser or c.outcome = EarlyLoser or
  c.outcome = SoreLoser or
  c.outcome = TiedLoser or
  c.outcome = EarlySoreLoser or
  c.outcome = TiedSoreLoser or
  c.outcome = EarlySoreLoserNonTransferable or
  c.outcome = EarlyLoserNonTransferable)
```

```
    // STV election quotas
    Election.method = STV implies
      quota = 1 + BallotBox.size.div[Election.seats+1] and
      fullQuota = 1 + BallotBox.size.div[Election.constituencySeats + 1]
    Election.method = Plurality implies quota = 1 and fullQuota = 1

   // All ties involve equality between at least one winner and at least one loser
    all w: Candidate | some l: Candidate | w.outcome = TiedWinner and
      (l.outcome = TiedLoser or l.outcome = TiedSoreLoser) implies
      (#l.votes + #l.transfers = #w.votes + #w.transfers)
    all s: Candidate | some w: Candidate | w.outcome = TiedWinner and
       (s.outcome = SoreLoser or s.outcome = TiedLoser) implies
       (#s.votes = #w.votes) or
       (#s.votes + #s.transfers = #w.votes + #w.transfers)

   // When there is a tied sore loser then there are no non-sore losers
   no disj a,b: Candidate | a.outcome = TiedSoreLoser and
        (b.outcome = TiedLoser or
         b.outcome=Loser or b.outcome=EarlyLoser or
         b.outcome = EarlyLoserNonTransferable)
    // For each Tied Winner there is a Tied Loser
        all w: Candidate | some l: Candidate | w.outcome = TiedWinner implies
                (l.outcome = TiedLoser or l.outcome = TiedSoreLoser)
    // Tied Winners and Tied Losers have an equal number of votes
        all disj l,w: Candidate |
    ((l.outcome = TiedLoser or l.outcome = TiedSoreLoser) and
    w.outcome = TiedWinner) implies
    #w.votes + #w.transfers = #l.votes + #l.transfers
    // Compromise winner must have more votes than any tied winners
        all disj c,t: Candidate | (c.outcome = CompromiseWinner and
            t.outcome = TiedWinner) implies
            #t.votes + #t.transfers < #c.votes + #c.transfers
    // Winners have more votes than non-tied losers
        all w,l: Candidate | w.outcome = Winner and
      (l.outcome = Loser or l.outcome = EarlyLoser or l.outcome = SoreLoser or
       l.outcome = EarlyLoserNonTransferable or l.outcome = EarlySoreLoser or
       l.outcome = EarlySoreLoserNonTransferable)
       implies
     ((#l.votes < #w.votes) or (#l.votes + #l.transfers < #w.votes + #w.transfers))
    // For each Tied Loser there is at least one Tied Winner
   all c: Candidate | some w: Candidate |
   (c.outcome = TiedLoser or c.outcome = TiedSoreLoser)
                implies w.outcome = TiedWinner
}

-- The Ballot Box
one sig BallotBox {
  spoiltBallots:     set Ballot, -- empty ballots excluded from count
  nonTransferables: set Ballot, -- ballots for which preferences are exhausted
  size:              Int        -- number of unspolit ballots
}
{
  no b: Ballot | b in spoiltBallots and b in nonTransferables
  size = #Ballot - #spoiltBallots
```

```
        all b: Ballot | b in spoiltBallots iff #b.preferences = 0
  // All non-transferable ballots belong to an non-transferable surplus
  all b: Ballot | some c: Candidate | b in nonTransferables implies
    b in c.wasted
}

-- An Electoral Constituency
one sig Election {
  seats:             Int,    -- number of seats to be filled in this election
  constituencySeats: Int,    -- full number of seats in this constituency
  method:            Method  -- type of election; PR-STV or plurality
}
{
  0 < seats and seats <= constituencySeats
  seats < #Candidate
}

-- Basic Lemmas
assert honestCount {
  all c: Candidate | all b: Ballot | b in c.votes + c.transfers
    implies c in b.assignees
}
check honestCount for 15 but 6 int

assert atLeastOneLoser {
  0 < #Scenario.losers
}
check atLeastOneLoser for 15 but 6 int

assert atLeastOneWinner {
  0 < #Scenario.winners
}
check atLeastOneWinner for 14 but 6 int

assert plurality {
        all c: Candidate | all b: Ballot | b in c.votes and
                Election.method = Plurality implies c in b.preferences.first
}
check plurality for 18 but 6 int

assert pluralityNoTransfers {
  all c: Candidate | Election.method = Plurality implies 0 = #c.transfers
}
check pluralityNoTransfers for 13 but 7 int

assert wellFormedTieBreaker {
  some w,l : Candidate | (w in Scenario.winners and
    l in Scenario.losers and
    #w.votes = #l.votes and #w.transfers = #l.transfers) implies
    w.outcome = TiedWinner and
    (l.outcome = TiedLoser or l.outcome = TiedSoreLoser)
}
check wellFormedTieBreaker for 18 but 6 int

```

```
assert validSurplus {
  all c: Candidate | 0 < #c.surplus implies
  (c.outcome = WinnerNonTransferable or
  c.outcome = QuotaWinnerNonTransferable or c.outcome = SurplusWinner or
  c.outcome = AboveQuotaWinner or
  c in Scenario.eliminated)
}
check validSurplus for 16 but 6 int

-- Advanced Lemmas
-- Equal losers are tied or excluded early before last round
assert equalityofTiedWinnersAndLosers {
  all disj w,l: Candidate | w in Scenario.winners and
  l in Scenario.losers and
  #w.votes + #w.transfers = #l.votes + #l.transfers implies
  w.outcome = TiedWinner and
  (l.outcome = TiedLoser or
    l.outcome = TiedSoreLoser or
    l.outcome = EarlyLoserNonTransferable or
    l.outcome = EarlySoreLoserNonTransferable or
    l.outcome = EarlyLoser)
}
check equalityofTiedWinnersAndLosers for 16 but 7 int

-- No lost votes during counting
assert accounting {
  all b: Ballot | some c: Candidate | 0 < #b.preferences implies
  b in c.votes and c in b.assignees
}
check accounting for 16 but 6 int

-- Cannot have tie breaker with both tied sore loser and non-sore loser
assert tiedWinnerLoserTiedSoreLoser {
  no disj c,w,l: Candidate | c.outcome = TiedSoreLoser and
  w.outcome = TiedWinner and
  (l.outcome = Loser or l.outcome = TiedLoser)
}
check tiedWinnerLoserTiedSoreLoser for 6 int

-- Compromise winner must have at least one vote
assert validCompromise {
  all c: Candidate | c.outcome = CompromiseWinner implies
  0 < #c.votes + #c.transfers
}
check validCompromise for 6 int

-- Quota winner needs transfers
assert quotaWinnerNeedsTransfers {
  all c: Candidate | c.outcome = QuotaWinner
  implies 0 < #c.transfers
}
check quotaWinnerNeedsTransfers for 7 int

-- Sore losers below threshold
```

```
assert soreLoserBelowThreshold {
  all c: Candidate | c.outcome = SoreLoser implies not
  (Scenario.threshold <= #c.votes + #c.transfers)
}
check soreLoserBelowThreshold for 10 but 6 int

-- Possible outcomes when under the threshold
assert underThresholdOutcomes {
  all c: Candidate |
     (#c.votes + #c.transfers < Scenario.threshold) implies
     (c.outcome = SoreLoser or c.outcome = TiedSoreLoser or
     c.outcome = TiedWinner or
     c.outcome = EarlySoreLoserNonTransferable or
     c.outcome = EarlySoreLoser or
     c.outcome = CompromiseWinner or
     (Election.method = Plurality and c.outcome = Winner))
}
check underThresholdOutcomes for 10 but 6 int

-- Tied Winners have equality of votes and transfers
assert tiedWinnerEquality {
  all a,b: Candidate | (a.outcome = TiedWinner and
    b.outcome = TiedWinner) implies
    #a.votes + #a.transfers = #b.votes + #b.transfers
}
check tiedWinnerEquality for 10 but 6 int

-- Non-negative threshold and quota
assert nonNegativeThresholdAndQuota {
        0 <= Scenario.threshold and 0 <= Scenario.quota
}
check nonNegativeThresholdAndQuota for 6 but 6 int

-- STV threshold below quota
assert thresholdBelowQuota {
  Election.method = STV and 0 < #Ballot implies
  Scenario.threshold <= Scenario.quota
}
check thresholdBelowQuota for 13 but 7 int

-- Plurality sore loser
assert pluralitySoreLoser {
  all c: Candidate | (c.outcome = SoreLoser and
  Election.method = Plurality) implies
  #c.votes < Scenario.threshold
}
check pluralitySoreLoser for 13 but 7 int

-- Plurality winner for a single seat constituency
assert pluralityWinner {
all disj a, b: Candidate | (Election.method = Plurality and
  Election.seats = 1 and
  a.outcome = Winner) implies #b.votes <= #a.votes
}
```

```
check pluralityWinner for 2 but 7 int

-- Length of PR-STV ballot does not exceed number of candidates
assert lengthOfBallot {
  all b: Ballot | Election.method = STV implies
    #b.preferences <= #Candidate
}
check lengthOfBallot for 7 int

-- Quota for a full election is less than for a by-election
assert  fullQuota {
          Scenario.fullQuota <= Scenario.quota
}
check fullQuota for 7 int

-- All transfers have a source either from a winner with surplus or
-- by early elimination of a loser
assert transfersHaveSource {
  all b: Ballot | some disj donor, receiver : Candidate |
    b in receiver.transfers
    implies b in donor.votes and
    (donor in Scenario.winners or
     donor in Scenario.eliminated)
}
check transfersHaveSource for 7 int

-- No missing candidates
assert noMissingCandidates {
  #Candidate = #Scenario.winners + #Scenario.losers
  }
check noMissingCandidates for 7 int

-- Spoilt votes are not allocated to any candidate
assert handleSpoiltBallots {
    no c : Candidate | some b : Ballot | b in c.votes and
    b in BallotBox.spoiltBallots
}
check handleSpoiltBallots for 7 int
```