Overview

Single Transferable Voting (STV) scheme?
Why is it hard to tally ballots according to STV?
Current computer counting in Australia
Scrutiny and trust?
Light-weight and Heavy-weight Formal Methods
Our work: high level
What do we mean by voting scheme?

A method for setting out, filling in and counting ballots

**STV Ballot Form**

Rank any number of candidates in order of preference.

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
<th>Charlie</th>
<th>Dave</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

Setting out: order of candidates fixed or Robson rotated?

Filling in: write all numbers from 1 to $N$ or only ones you want?

Counting: quota required to be elected; who is weakest candidate; how to break ties; how to transfer a vote; when to stop counting

Nothing to do with electronic voting . . . yet

In particular, nothing to do with security aspects of electronic voting
Single Transferable Vote: “One Version”

Vacancies: number of candidates that we need to elect
Candidates: number of people standing for election
Quota: how many votes are required to elect a candidate
Ballot: is a vote for highest ranked continuing candidate
Transfer Value: current value of ballot (possibly $\leq 1$)
Counting: proceeds in rounds

<table>
<thead>
<tr>
<th>STV Ballot Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rank any number of candidates in order of preference.</td>
</tr>
</tbody>
</table>

Alice 3
Bob
Charlie 1
Dave 2

Rounds: repeat until all seats filled
Tally: all highest preferences
Elected: All candidates with at least “quota” votes are elected
Eliminated: If nobody elected this round then eliminate weakest cand.
Transfer: compute new transfer values of ballots
Autofill: If can seat all remaining cands., do so
Example

Droop Quota: \( Q = \left\lfloor \frac{\text{totalnumberofballots}}{\text{seats}+1} \right\rfloor + 1 \)

Candidates: \( A, B, C, D \)

Seats: 2

Ballots: 5

\[
A > B > D
\]
\[A > B > D\]
\[A > B > D\]
\[D > C\]
\[C > D\]

Assume no fractional transfers and no autofill
Example

Droop Quota: \[ Q = \left\lfloor \frac{\text{total number of ballots}}{\text{seats}+1} \right\rfloor + 1 \]

Candidates: \( A, \ B, \ C, \ D \)

Seats: 2

Ballots: 5

\[ Q = \left\lfloor \frac{5}{2+1} \right\rfloor + 1 = 2 \]

\[ A > B > D \]
\[ A > B > D \]
\[ A > B > D \]
\[ D > C \]
\[ C > D \]

Elected: \( A, \ D \)

Eliminated: \( B \)

Assume no fractional transfers and no autofill
Example

Droop Quota: \[ Q = \left\lfloor \frac{\text{total number of ballots}}{\text{seats} + 1} \right\rfloor + 1 \]

Candidates: A, B, C, D

Seats: 2
Ballots: 5

\( Q = \left\lfloor \frac{5}{2+1} \right\rfloor + 1 = 2 \)

\( A > B > D \) \( \text{votes}(A) = 1 \)
\( A > B > D \) \( \text{votes}(A) = 2 \)
\( A > B > D \) \( \text{votes}(A) = 3 \)
\( D > C \) \( \text{votes}(D) = 1 \)
\( C > D \) \( \text{votes}(C) = 1 \)

Elected: A, D
Eliminated: B
Example

Droop Quota: $Q = \left\lfloor \frac{\text{totalnumberofballots}}{\text{seats}+1} \right\rfloor + 1$

Candidates: $A, B, C, D$

$Q = \left\lfloor \frac{5}{2+1} \right\rfloor + 1 = 2$

Seats: 2
Ballots: 5

$A > B > D$

$D > C \quad \text{votes}(D) = 1$

$C > D \quad \text{votes}(C) = 1$

Elected: $A$
Example

Droop Quota: \( Q = \left\lfloor \frac{\text{totalnumberofballots}}{\text{seats}+1} \right\rfloor + 1 \)

Candidates: \( A, B, C, D \)

Seats: 2

Ballots: 5

\[
Q = \left\lfloor \frac{5}{2+1} \right\rfloor + 1 = 2
\]

\[\begin{align*}
A & \rightarrow B & \rightarrow D \\
A & \rightarrow B & \rightarrow D \\
A & > & B & > & D \\
D & > & C & & \text{votes}(D) = 1 \\
C & > & D & & \text{votes}(C) = 1
\end{align*}\]

Elected: \( A \)
Example

Droop Quota: \[ Q = \left\lfloor \frac{\text{total number of ballots}}{\text{seats}+1} \right\rfloor + 1 \]

Candidates: A, B, C, D

Seats: 2

Ballots: 5

\[
Q = \left\lfloor \frac{5}{2+1} \right\rfloor + 1 = 2
\]

\[
A > B > D \quad \text{votes}(B) = 1
\]

\[
D > C \quad \text{votes}(D) = 1
\]

\[
C > D \quad \text{votes}(C) = 1
\]

Elected: A

Eliminated: B
Example Droop Quota: \[ Q = \left\lfloor \frac{\text{totalnumberofballots}}{\text{seats}+1} \right\rfloor + 1 \]

Candidates: \( A, B, C, D \)

Seats: 2

Ballots: 5

\[ Q = \left\lfloor \frac{5}{2+1} \right\rfloor + 1 = 2 \]

\( A > B > D \)

\( A > B > D \)

\( \Box > \Box > D \) \hspace{1cm} \text{votes}(D) = 2

\( D > C \)

\( C > D \) \hspace{1cm} \text{votes}(C) = 1

Elected: \( A \)

Eliminated: \( B \)
Example Droop Quota: \( Q = \left\lfloor \frac{\text{totalnumberofballots}}{\text{seats}+1} \right\rfloor + 1 \)

Candidates: \( A, \ B, \ C, \ D \)
Seats: 2
Ballots: 5

\[ Q = \left\lfloor \frac{5}{2+1} \right\rfloor + 1 = 2 \]

\[ \begin{align*}
\large\text{A} & \large\rightarrow \text{B} \rightarrow \text{D} \\
\large\text{A} & \large\rightarrow \text{B} \rightarrow \text{D} \\
\large\text{x} & \large\rightarrow \text{x} \rightarrow \text{D} & \text{votes}(D) = 2
\end{align*} \]
\[ D \rightarrow C \]
\[ C \rightarrow D \]

Elected: \( A \)
Eliminated: \( B \)
Example

Droop Quota: \( Q = \left\lfloor \frac{\text{total number of ballots}}{\text{seats}} + 1 \right\rfloor + 1 \)

Candidates: \( A, B, C, D \)

Seats: 2

Ballots: 5

\( Q = \left\lfloor \frac{5}{2+1} \right\rfloor + 1 = 2 \)

\( \begin{array}{l}
A > B > D \\
A > B > D \\
A > B > D \\
A > B > D \\
A > B > D \\
A > B > D \\
A > B > D
\end{array} \)

\( \text{votes}(D) = 2 \)

\( D > C \)

\( C > D \)

Elected: \( A, D \)

Eliminated: \( B \)
Single Transferable Vote: “One Version”

Vacancies: number of candidates that we need to elect
Candidates: number of people standing for election
Quota: how many votes are required to elect a candidate
Ballot: is a vote for highest ranked continuing candidate
Transfer Value: current value of ballot (possibly $\leq 1$)
Counting: proceeds in rounds

Many choices and hence many versions
Each optimisation increases difficulty of hand-counting!
Our thesis ...

- Counting ballots using computers allows us to use voting schemes which really are optimised along multiple dimensions.

- Such optimised schemes may make it impossible to “verify” the result by hand-counting, even when number of ballots is small.

- It is therefore important to imbue these schemes with the trust accorded to existing hand-counted schemes.
Existing Electronic Vote-counting in Australia

Australian Electoral Commission: proprietary code; not available for scrutiny; FOI request to publish code denied on grounds of “security” and “commercial in confidence”

Victorian Electoral Commission: proprietary code; available for scrutiny; no formal scrutiny to my knowledge

Australian Capital Territory: eVACS\textsuperscript{TM}

- developed by Software Improvements Pty Ltd. using C++
- used since 2001 to count four elections
- counting code available from ACTEC website
- full code available if you sign a confidentiality agreement

New South Wales Electoral Commission: detailed functional requirements publicly available; found to comply with legislation by legal expert from QUT; certified by Birlasoft as passing all tests; proprietary code; code not available for scrutiny (Ian?)

\textsuperscript{TM}eVACS is a trademark of Software Improvements Pty Ltd.
eVACS® is the world leader in secure and high integrity election software. eVACS® has been endorsed by the Police Association (Victoria).

eVACS® is:

- Proven system
- Has been extensively tested
- Has been independently audited by software auditing firm, BMM International
- Is transparent with source code publicly available
- Is reliable and secure

eVACS® has the following benefits:

- Allows vision-impaired voters to vote in secret
- Provides instructions in multiple languages
- Eliminates unintentional voter errors
- Reduces the number of informal votes
- Increases the speed and accuracy of election counts
ACTEC and SoftImp Approach

- scrutiny
- artifacts
- trust

- published
- legal text
- functional specs using UML
- published?
- evidence?
- semi-published

ACTEC & SoftImp

- “audited” by BMM: all okay
Would you trust BMM International?
Bugs in eVACS Counting Module

ANU logic group: found three bugs

- programming error: simple for-loop bounds error
- ambiguous legal text: equally weak candidates requires a return to round where "all candidates have an unequal number of votes"
- programming error: un-initialised boolean: different compilers give different results

ACT Elections: acknowledged all as "minor" bugs and fixed them but reiterated his "full confidence" in eVACS

BUT: for each bug, we could find an example election which would be counted erroneously by eVACS

Fortunately: independent implementation by Jeremy Dawson elected same people for the past four elections as did eVACS
NSWEC Approach

- **scrutiny**
- **artifacts**
- **trust**

- published
- **legal text**
- **NSWEC**
- **legal expert (QUT)**
- published 47 pages of functional specs with flow chart
- **Vendor**
- **computer code**

- evidence?

- proprietary

- Birlasoft: passed all tests
ACTEC/NSWEC Approach versus Formal Methods

ACTEC/NSWEC Approach

Legal text: the (English) prose of the Act of Parliament
Functional specs: detailed functional requirements (published)
Testing: extensive tests
Opinion: from legal expert that it complies with legal text;
Certification: from BMM/Birlasoft that code passes their tests.

Formal Methods

Dijkstra on Testing: Program testing can be used to show the presence of bugs, but never to show their absence!

Hoare in retrospection: My basic mistake was to set up proof in opposition to testing, where in fact both of them are valuable and mutually supportive ways of accumulating evidence of the correctness and serviceability of programs.
<table>
<thead>
<tr>
<th>ACTEC/NSWEC Approach</th>
<th>Formal Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Legal text:</strong> the (English) prose of the Act of Parliament</td>
<td><strong>Dijkstra on Testing:</strong> Program testing can be used to show the presence of bugs, but never to show their absence!</td>
</tr>
<tr>
<td><strong>Functional specs:</strong> detailed functional requirements (published)</td>
<td><strong>Hoare in retrospection:</strong> My basic mistake was to set up proof in opposition to testing, where in fact both of them are valuable and mutually supportive ways of accumulating evidence of the correctness and serviceability of programs.</td>
</tr>
<tr>
<td><strong>Testing:</strong> extensive tests</td>
<td></td>
</tr>
<tr>
<td><strong>Opinion:</strong> from legal expert that it complies with legal text;</td>
<td></td>
</tr>
<tr>
<td><strong>Certification:</strong> from BMM/Birlasoft that code passes their tests.</td>
<td></td>
</tr>
</tbody>
</table>
Light-weight Formal Methods (related work)

Scruity artifacts trust

Published legal text

Programmer logic expert

Computer code with functional specs as logical annotations

Computer tool: automated proof

Computer tool: tens of thousands of lines of code

Often fails because proofs cannot be found automatically
Heavy-weight Formal Methods Approach

- **scrumptiny**
- **artifacts**
- **trust**

**Published**

- Legal text

**Published**

- Functional specs as formula SPECHOL of logic

**Published**

- Computer code as formula IMPHOL of logic

**Published**

- Computer code

**Person**

→ (your) logic expert

→ Machine-checked interactive proof that IMPHOL $\Rightarrow$ SPECHOL

→ (your) logic expert
What is a machine-checked proof?

artifacts

Dana Scott’s logic of computable functions

Gordon

HOL4 theorem prover:
4000 lines of SML code

person

proof that IMPHOL \Rightarrow SPECHOL
certified as correct by HOL4

Can take months and requires expertise in formal methods!
Why Should We Trust Machine-checked Proof?

Dana Scott’s logic of computable functions

\[ \text{HOL4 theorem prover: 4000 lines of SML code} \]

\[ \text{proof that IMPHOL } \Rightarrow \text{ SPECHOL} \]
certified as correct by HOL4

peer review
logic community
HOL4 development team
HOL4

person

Gordon

scrutiny

artifacts

trust

published

published

published
Our (Heavy-weight Formal Methods) Approach

- scrutiny artifacts trust
  - published legal text
    - Meumann → (your) logic expert
  - published functional specs as formula SPECHOL of logic
    - Dawson
    - Meumann
    - Meumann → machine-checked interactive proof that IMPHOL ⇒ SPECHOL
  - published computer code as formula IMPHOL of logic
    - Dawson
    - Meumann
    - (your) logic expert
  - published computer code

NB: Dawson scrutineers Meumann and HOL4 scrutineers Dawson
Examples:

Listing 1: Executable function definitions.

(* Returns list of ballots whose first preference is cand *)
val FIRSTS_FOR_DEF = Define ' 
  FIRSTS_FOR cand ballots = 
    FILTER (($= cand) o HD) ballots';

(* Sums the number of ballots with a first preference for 
each of the running candidates. This is needed simply 
because the legislation specifies that this is how the 
quota should be calculated. *)
val SUM_FIRSTS_DEF = Define ' 
  (SUM_FIRSTS [] ballots = 0) 
  \ (SUM_FIRSTS (c::cs) ballots = 
    LENGTH (FIRSTS_FOR c ballots) 
    + SUM_FIRSTS cs ballots)';
Sanity checks

Listing 2: Candidates cannot be introduced partway through the count.

```plaintext
!seats cands ballots state state'.
   (COUNT_HCT seats cands ballots = FINAL_STAGE state)
   \ (COUNT_HCT seats cands ballots = FINAL_STAGE state')
   \ TIME_VAR state' > TIME_VAR state
==> !cand. IS_REM_CAND state' cand
    ==> IS_REM_CAND state cand
```

Listing 3: Candidates cannot be introduced partway through the count.

```plaintext
forall seats cands ballots state state'.
   if (COUNT_HCT seats cands ballots = FINAL_STAGE state)
       and (COUNT_HCT seats cands ballots = FINAL_STAGE state')
       and TIME_VAR state' > TIME_VAR state
   then forall cand. if IS_REM_CAND state' cand
       then IS_REM_CAND state cand
```
Where are we now?

SPECHOL: 600 lines of HOL4 with extensive comments
IMPHOL: 700 lines of HOL4 with extensive comments
SML program: 400 lines of SML (few comments)
Meumann: six months full time
IMPHOL Tested: can actually count small elections
SML program tested: against Dawson’s own ACT implementation
SML program tested: specific examples where they should/do differ
SML program viable: can handle millions of ballots
HOL4Theorem: MERGESORT terminates (trivial)
HOL4Theorem: in all rounds, candidates remain distinct (3 days)
HOL4Theorem: candidates cannot appear or disappear (4 days)
HOL4Theorem: IMPHOL is well-defined and terminates (2 months)
Underway: in all rounds, ballots cannot disappear
Underway: in all rounds, ballots cannot appear
What discrepancies did we find?

Lists: accessing $n^{th}$ element without checking list length first

Distinctness: no check that the candidates are distinct initially

Reachability: to obtain set of all reachable states required cosmetic changes to IMPHOL to obtain “next state” of a count
Conclusions

Heavy-weight Verification: feasible but very time-consuming

Estimate: IMPHOL $\Rightarrow$ SPECHOL proof (months more)