Complexity in Electoral Systems:

Proving Properties Using HOL4

Overview

- Electoral Systems
  - Why complexity is necessary.
- Current e-voting systems
- Our approach
- Small case study: plurality
- Large case study: Hare-Clark Tasmania
- Conclusions
- Discussion...

Electoral Systems

- First-Past-the-Post (Plurality): simple, common, unrepresentative
- Preferential: like multiple run-offs
- Single Transferrable Vote (STV)
  - Multi-member electorates
  - Labour-intensive to count
  - Complex
  - MUCH more representative

Manual Counting of STV

- Slow and labour-intensive
- Error prone:
  - WA debacle: $10-20 million for a re-run and multiple resignations
  - STV ballots often need to be counted over 100 times

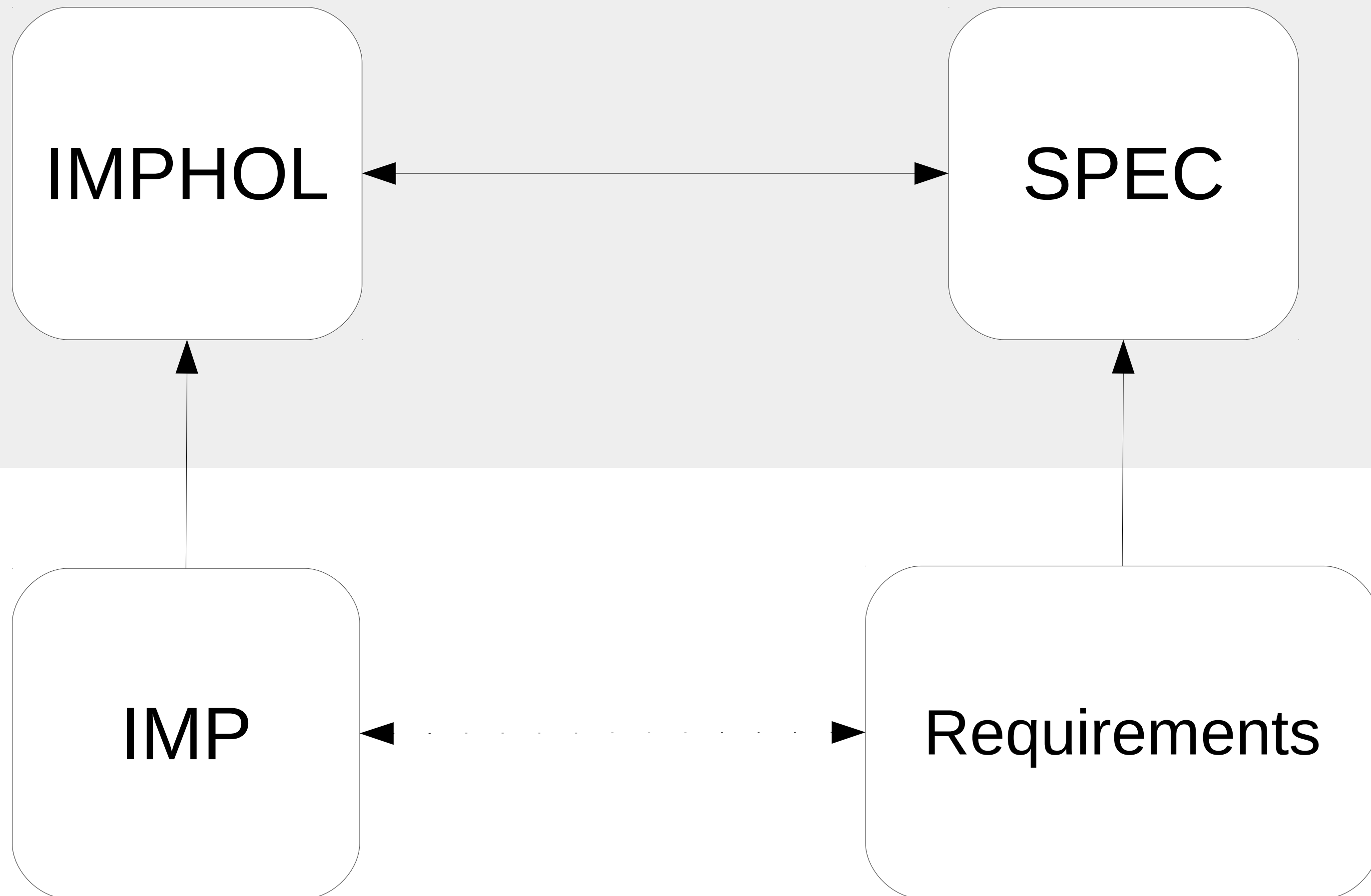
Electronic Counting: State of the Art

- End-to-End Verifiable (eg. Helios & Scantegrity)
  - Cryptographically guarantees votes are collected and ***summed*** correctly (ie. can only handle FPTP)
- Cannot count STV elections
- Shuffle-Sum: counts STV with combo of cryptographic methods
  - Potentially main competitor: needs discussion

…in Australia

- Federal Senate: EasyCount
  - Closed-source
  - FOI request for code denied
    - “trade secret”
- ACT Legislative Assembly: EVACS
  - Unverified C code
  - Several bugs found since being used to count elections
  - But source code available online

Verification Components



Our Approach: Formal Verification

- SML implementation (IMP)
- HOL4 translation of IMP (IMPHOL)
- Specification in HOL4 (SPEC)
- Proof that SPEC holds of IMPHOL

- Why SML?
  - Functional (easy to reason about)
  - Similar type inference system to HOL4

Why HOL4?

- LCF-style (Logic for Computable Functions)
- 8 core logical axioms
  - Assumption introduction, reflexivity, beta-conversion, substitution, abstraction, type instantiation, discharging an assumption and Modus Ponens
  - Complex rules may *only* be constructed from these axioms
  - Results in highly rigorous proofs
- Caveats
  - User must drive the proof process manually
  - Can get bogged down dealing with minutiae

Small-Scale Case Study

- Plurality Voting
  - Correctness: easy with automated tools
  - Monotonicity Criterion: not so easy
    - "If each voter either changes his or her vote to a vote for candidate 'w' or maintains his or her vote unchanged, and 'w' won before any votes changed, then 'w' will still win after the changes."
    - ```
      !C w v v'.
      ((LENGTH v' = LENGTH v) /\
        (!n. (n < LENGTH v) ==> ((EL n v' = w) \/ (EL n v = EL n v'))) /\
                                 (ELECT C v = SOME w))
                                                ==> (ELECT C v' = SOME w)
      ```
      ```

IMP and IMPHOL

```
local
  (* Counts the number of votes in the
     given list for candidate c. *)
  fun COUNTVOTES c []     = 0
    | COUNTVOTES c (h::t) = if h = c
                      then 1 + COUNTVOTES c t
                      else 0 + COUNTVOTES c t;

  (* Finds winner from all candidates
     numbered c or lower. *)
  fun WINNER 0 v = (SOME 0, COUNTVOTES 0 v)
    | WINNER c v =
      let
        val numvotes = COUNTVOTES c v
      in
        let
          val (w, max) = WINNER (c-1) v
        in
          if numvotes > max
            then (SOME c, numvotes)
          else if numvotes = max
            then (NONE, max)
          else (w, max)
        end
      end;
in
  (* C is the number of candidates, v is the
     list of votes *)
  fun ELECT C v = if C <= 0 then NONE
                  else #1 (WINNER (C-1) v)
```

```
val COUNTVOTES_def = Define `
  (COUNTVOTES c []     = 0) /\
  (COUNTVOTES c (h::t) = if (h = c)
                    then 1 + COUNTVOTES c t
                    else 0 + COUNTVOTES c t)`;


val WINNER_def = Define `
  (WINNER 0 v = (SOME 0, COUNTVOTES 0 v)) /\
  (WINNER c v =
    let
      numvotes = COUNTVOTES c v
    in
      let
        (w, max) = WINNER (c-1) v
      in
        if numvotes > max
          then (SOME c, numvotes)
        else if numvotes = max
          then (NONE, max)
        else (w, max))`;


val ELECT_def = Define `
  ELECT C v = if C <= 0 then NONE
              else FST (WINNER (C-1) v)`;
```

Correctness

```
!C v w. w < C ==>
        ((ELECT C v = SOME w) = !c'. c' <> w /\ c' < C
            ==> COUNTVOTES w v > COUNTVOTES c' v)
```

or

```
!C v w. w < C ==>
        ((ELECT C v = SOME w) = !c'. c' <> w /\ c' < C
                    ==> LENGTH (FILTER ($= w) v)
                              > LENGTH (FILTER ($= c') v))
```

Lessons Learnt

- Gap between IMP and IMPHOL
  - Syntactic equivalence ≠ semantic equivalence
    - eg. numerical types
  - Use HOL4's rewriting engine to count the votes
    - Possible for small elections, not for large ones
  - CakeML could solve this
    - Verified HOL4 to CakeML translator
    - Verified CakeML to x86 bytecode compiler

Hare-Clark

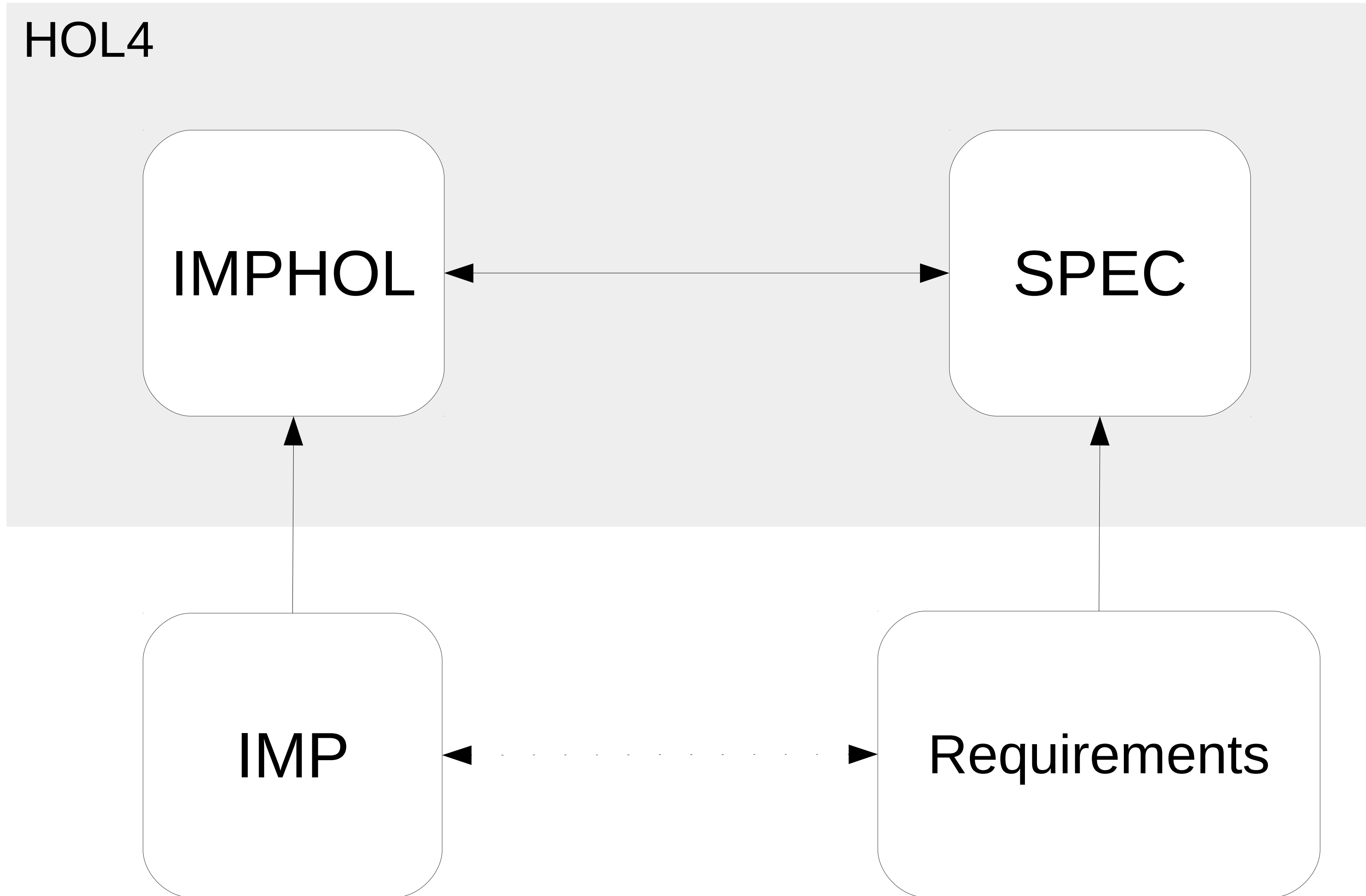
- Variant of PR voting systems
- Used in the ACT and Tasmania
- Gregory Fractional Transfer
- Droop Quota: $\lceil \frac{votes}{seats+1} \rceil$

Updated Process

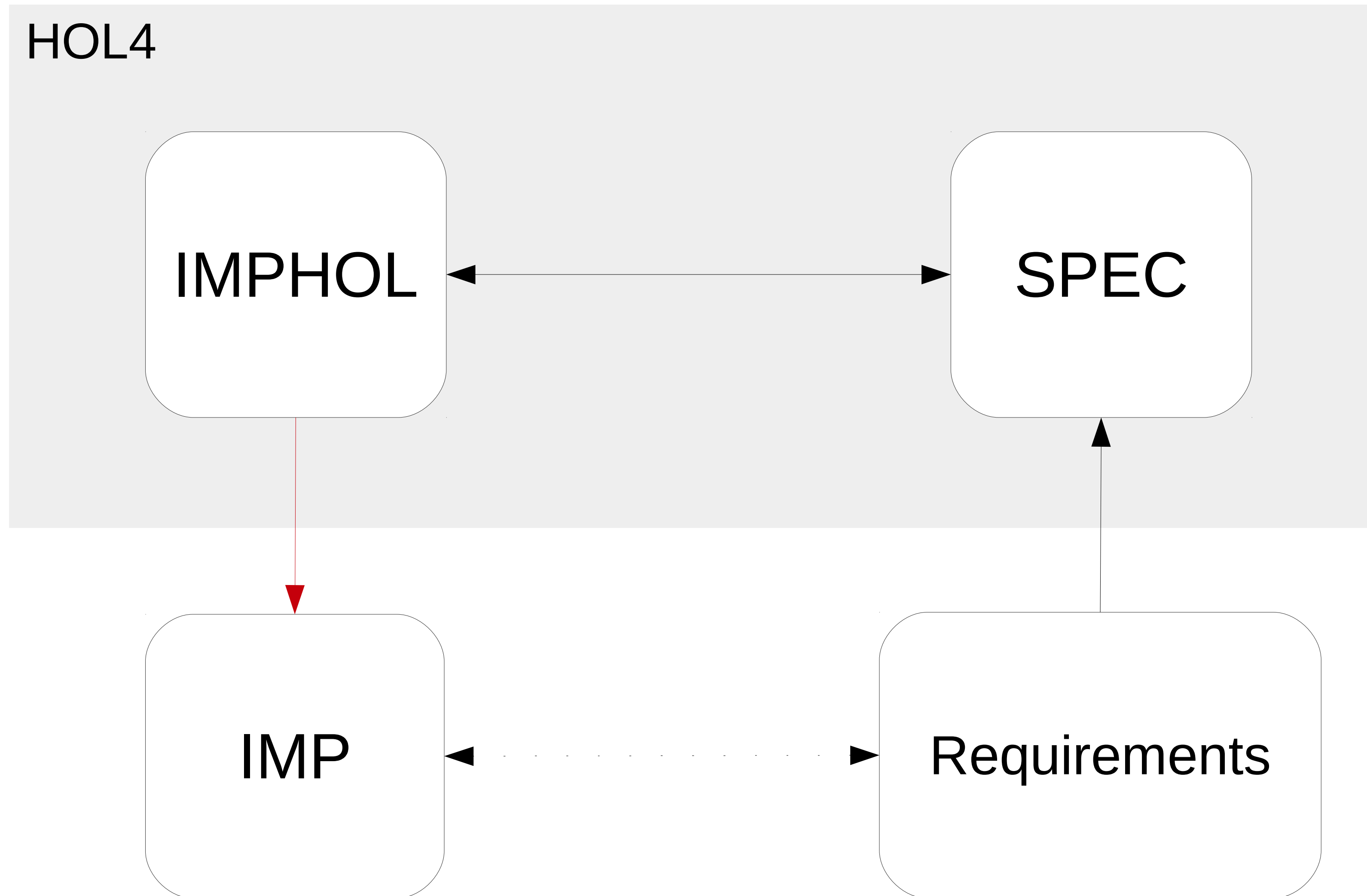
- Changes:
  - IMPHOL $\longrightarrow$ IMP (mirrors CakeML translation)
  - SPEC $\longrightarrow$ IMP/IMPHOL
    - Must make assumptions about form of IMP/IMPHOL
      - Likely to change due to practical considerations

- Currently:
  - Produced SPEC
  - Produced IMP and IMPHOL
  - No proof IMPHOL meets SPEC yet

Updated Process



Updated Process



New Issues!!

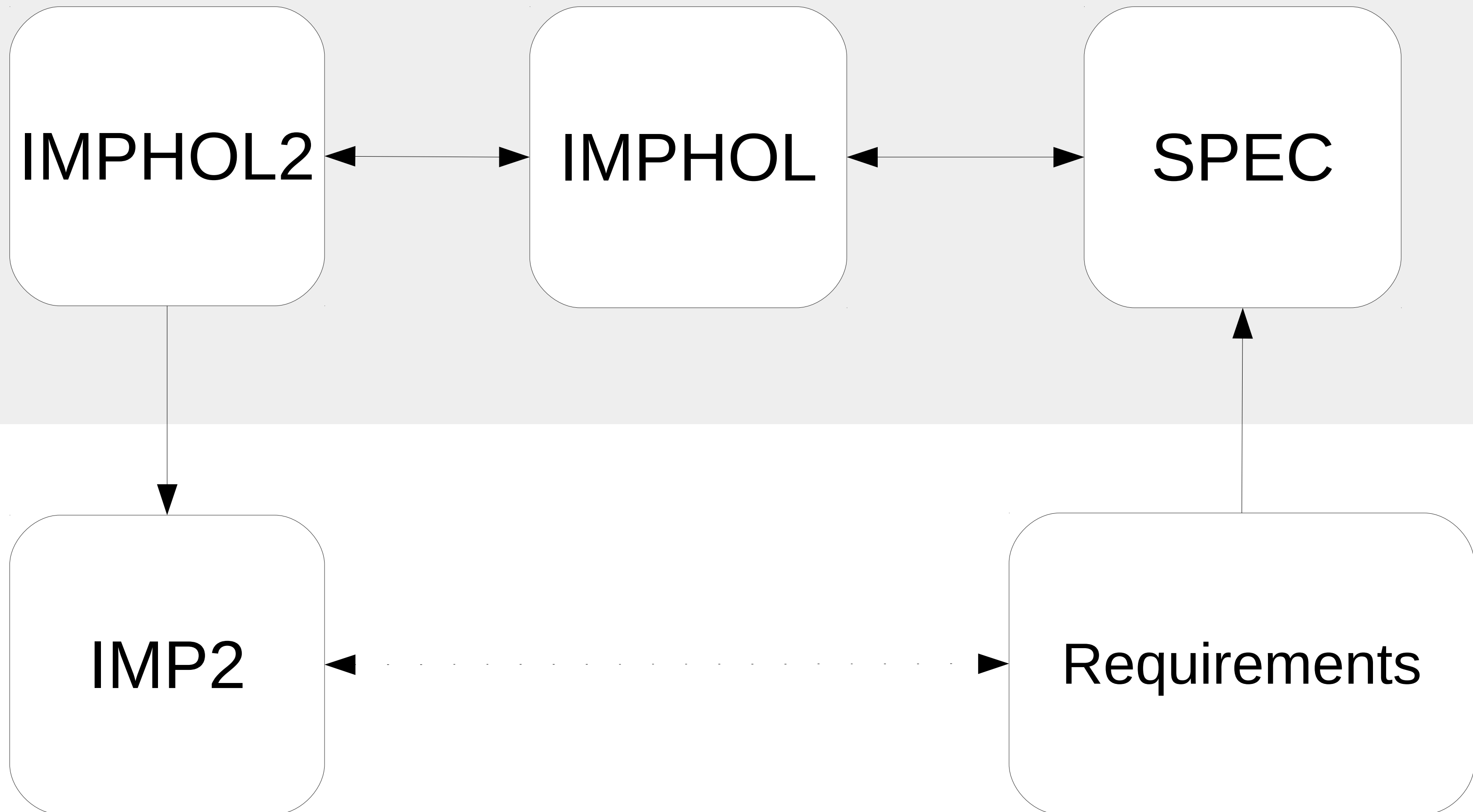
- SPEC and IMPHOL are coupled
  - Add another implementation, prove equivalence
- Legislation is procedural
  - Artificially build state and temporality into program
- Legislation is ambiguous
- SPEC is somewhat dense

- BUT: IMP tested and works efficiently!!

Complexity of SPEC: Clause 10(4)

```
!seats cands ballots state cand_entry COUNT TRANSFER_TO.
    (COUNT_HCT seats cands ballots = NEXT_STAGE state)
    /\ MEM cand_entry (REM_CANDS_VAR state)
    /\ TOTAL_COUNT cand_entry > QUOTA_VAR state
    /\ ((LAST_TRANSFER_CLAUSE cand_entry = clause10_4)
        \/ (LAST_TRANSFER_CLAUSE cand_entry = clause9)
        \/ (LAST_TRANSFER_CLAUSE cand_entry = clause11))
    (* copy of 8 (4) *)
    (* should this be !pre post? *)
    ==> ?pre post.
        (COUNT_HCT seats cands ballots = NEXT_STAGE pre)
        /\ (COUNT_HCT seats cands ballots = NEXT_STAGE post)
        /\ (TIME_VAR post = SUC (TIME_VAR pre))
        /\ TIME_VAR pre >= TIME_VAR state
        /\ !rcvr_pre rcvr_post transval.
            (FST rcvr_pre = FST rcvr_post)
            /\ (FST rcvr_pre <> FST cand_entry)
            /\ MEM rcvr_pre (REM_CANDS_VAR pre)
            /\ ~MEM (FST rcvr_pre) (ELECTED_VAR pre)
/\ (COUNT rcvr_pre = LENGTH (FILTER
    (($= (FST rcvr_pre)) o HD
      o (STRIP_BALLOT (EXCL_VAR pre) (ELECTED_VAR post)))
    (LAST_TRANSFER_BALLOTS cand_entry)))
/\ (transval
        = ((QUOTA_VAR state - TOTAL_COUNT cand_entry)
            , LENGTH (LAST_TRANSFER_BALLOTS cand_entry)))
/\ VALID_TRANS_VAL transval
/\ (TRANSFER_TO rcvr_pre =
    (COUNT rcvr_pre * (FST transval)) DIV (SND transval))
/\ (TRANSFERS rcvr_post = (transval
    , FILTER (($= (FST rcvr_pre)) o HD
       o (STRIP_BALLOT (EXCL_VAR pre) (ELECTED_VAR pre)))
       (FST (SND (HD (SND (SND cand_entry)))))
    , clause8)::(TRANSFERS rcvr_pre))
/\ (TOTAL_COUNT rcvr_post
    = TOTAL_COUNT rcvr_pre + TRANSFER_TO rcvr_pre)
==> MEM rcvr_post (REM_CANDS_VAR post)
    (* ...but...clause 9 or 11... *)
    /\ (((LAST_TRANSFER_CLAUSE cand_entry = clause9)
         \/ (LAST_TRANSFER_CLAUSE cand_entry = clause11))
        ==> (EXCL_TRANS_VAR pre = []))
```

More Refinements



Conclusions

- SML program works for large numbers
- Proofs that IMPHOL meets SPEC are rock-solid
- With CakeML
  - IMPHOL – IMP proof can be rock-solid
  - IMPHOL will be compiled direct to verified bytecode

- **BUT: equivalence b/w legislation & SPEC unclear**
  - Easier to visually verify that the implementation matches the legislation than that the SPEC does…