Hare Clarke Vote Counting Program

Teresa Bradbury

August 10, 2013

If you are interested in looking into the details of this code, the following is a list of all the files and some of their interesting functions. 'The spec' refers to *A HOL model of the Hare-Clark voting system* by Michael Norish, Version 2.04, 10 May 2005, `http://users.cecs.anu.edu.au/~rpg/EVoting/evote_formalisation.html`.

- `EA_implementation_sml.sml`
  This is all the types and functions in the spec, some helper functions required for the conversion and the outer loop which I added. It is as close to the original spec as possible except when the spec was incorrect. The top level functions are:
  - `hareClarkCountImpIO (ballots, numToElect, dead)`
    Count the election and return the associated fullstate. The "`IO`" is because (if the flag `allow_prompts` is true) it will ask for the user to type in the candidates chosen by lot if required.
  - `winnersImp finalState`
    Returns the candidates who are successful (assuming `finalState` actually is the final state).
- `makeTables.sml`
  Post-processing functions to create the distribution of preferences tables (a la the ACT Electoral Commission) in html. This is a hack. There is nothing elegant or simple or efficient or even really understandable about this. It could definitely be better but to really avoid being a hack I would need to change some aspects of the implementation and I would really rather not. The interesting functions are:
  - `printBallotsTable fullState filePrefix`
    Creates the table showing which ballot papers were (re-)distributed where at each count. It is saved in "`filePrefixballotTable.html`" in the current directory.
  - `printVotesTable fullState filePrefix`
    Creates the table showing how many votes each candidate has at each count. It is saved in "`filePrefixvotesTable.html`" in the current directory.
- `parseElectionData.sml`
  Parsing functions for the 2008 election data as given on the ACT Electoral Commission website (the website). The interesting functions are
  - `parseFile filename`
    Parses the ballot paper preference data in `filename`. This will be one of `BrindabellaTotal.txt`, `GininderraTotal.txt` or `MolongloTotal.txt` off the website. The numbering of the candidates will be completely arbitrary. It returns the ballots, the mapping between the way the Electoral Commission represents their candidates and the way I do, and the total number of candidates.
  - `parseCandidates filename`
    Parses `filename` (which should be `Candidates.txt` off the website). The results can be used in conjunction with the mapping returned by `parseFile` to determine the name of the candidate associated with each number.

- `parseFileDict dict filename`
  Parses `filename` as in `ParseFile` but using the given mapping to candidate numbers. The candidates need to be numbered from 0 to (numEntries-1) inclusive. Any candidates found in the file that are not in the dictionary will be given the next available number.
- `candBriOrd`, `candGinOrd`, `candMolOrd`
  Maps that can be given to `parseFileDict` to put the candidates in the same order as on the preference tables on the website.

It's worth noting that these functions expect the data to be in *exactly* the right format and will just die with an obscure error if anything goes wrong. But if you just download the files off the website without changing anything (and the Electoral Commission doesn't change anything) it will work fine.

- `parseBLTFile.sml`
  - `parseBLTFile filename`
    Parses the text file called `filename` representing an election in BLT file format. Again, this is not a robust program so don't give it stupid things. It will also ignore the candidate names and title that are given at the end.
- `random_ballots.sml`
  Generates random elections. It currently generates uniformly distributed ballot papers for the elections which is slightly problematic. When running large elections, you end up just excluding people one by one until you have the same number of candidates as seats. The functions are:
  - `makeRandomBallots gen numCandidates numBallots`
    Generates `numBallots` worth of ballots, where each one is a randomly chosen length of a random permutation. Both the length and the permutation are uniformly distributed (though that doesn't necessarily mean the ballots themselves are). The random number generator is passed in as a parameter so the user can decide to use a specific seed or not as they wish. If the number of ballots is too high (possibly above 150 000) you will get an out of memory exception.
  - `timeCount ballots numToElect`
    Counts the ballot papers `ballots` (assuming no dead candidates) and also produces the distribution tables (with no prefix). It returns the time it took to count the election and the total time for the whole process (as strings).
- `EA_main.sml`
  A basic command line interface for the program. It is fractionally more robust than the election parsing code but not by much. It also currently assumes the 2008 election data is in a particular place in the file tree.
- `Makefile`
  Magic file that means you can type `make` in the terminal get an executable called `EA`. This will run so long as you have the `mosml` run-time on your computer. There is a compiler option to make the executable standalone if that is required.