Machine Checked Properties of the Schulze Method

Mukesh Tiwari
School of Computing and Information Systems
University of Melbourne
Melbourne, Australia
mukesh.tiwari@unimelb.edu.au

Dirk Pattinson
Research School of Computer Science
Australian National University
Canberra, Australia
dirk.pattinson@anu.edu.au

***Abstract*—The correctness of electronic vote-counting software is crucial in establishing the trust in electronic voting. However, most vote-counting programs, used in various jurisdiction for legally binding election, establish correctness by means of testing which is not sufficient, and often fails to identify rare corner cases. We argue that legally binding vote-counting software should be formally verified and their correctness should be evaluated against some well established framework from social choice theory**

In this work, we give machine checked formal proofs that our Coq implementation of the Schulze method, a preferential voting method, follows the Condorcet winner property. We leave the formal machine-checked proof of other properties, e.g., reversal symmetry, Pareto, monotonicity, etc., for future work. This is, to the best of our knowledge, the first machine-checked proof of properties of the Schulze method implementation, and in fact, any preferential vote-counting implementation.

***Index Terms*—formal method, electronic voting, Schulze method, Condorcet winner, Coq theorem prover**

1. Introduction

The Schulze method [1], a preferential voting method, has gained popularity in recent years when it comes to electing the candidates of many open source software communities, Wikipedia, and the Pirate Parties in various countries. At the time of writing, is used by more than 60 organizations with more than 900,000 eligible members in total [2]. One particular reason for this popularity is that it enjoys many desirable properties that have been formulated in social choice theory. Many of these are already established in the Schulze's original paper [1], e.g., Condorcet winner, Pareto, reversal symmetry, monotonicty, etc. Moreover, it fails on independence of irrelevant alternatives (IIA) criterion, a consequence of the impossibility theorem [3] which states that no preferential voting method can have all the properties.

In this work, we establish that our implementation of Schulze method [4] conforms to the Condorcet winning criterion. It is ongoing work and in future, we would like to establish that our implementation follows all the other properties, e.g. reversal symmetry, Pareto, monotonicity, etc. More importantly, it would be interesting to establish that our implementation fails on IIA criterion. The source code for this ongoing work can be accessed from the GitHub repo[1]

2. Schulze Method

In this section, we give a brief overview of the Schulze method and necessary formalisation details to make it self-contained. We invite the curious reader to read our paper [4] for more details.

The method itself rests on the relative margins between two candidates, i.e. the number of voters that prefer one candidate over another. The margin induces an ordering between candidates, where a candidate c is more preferred than d if more voters prefer c over d than vice versa. One can construct simple examples (see e.g. [5]) where this order does not have a maximal element (a so-called Condorcet Winner). Schulze's observation is that this ordering can be made transitive by considering sequences of candidates (called paths). Given candidates c and d, a path between c and d is a sequence of candidates $p = (c, c_1, \dots, c_n, d)$ that joins c and d, and the *strength* of a path is the minimal margin between adjacent nodes. This induces the generalised margin between candidates c and d as the strength of the strongest path that joins c and d. A candidate c then wins a Schulze count if the generalised margin between c and any other candidate d is at least as large as the generalised margin between d and c. It is known that every election has a winner, but the winner may not be uniquely determined (e.g. in the case where no votes have been cast at all). In more detail:

1) Consider an election with a set of t candidates $C = \{c_1, \dots, c_t\}$ and a set of n votes $P = \{b_1, \dots, b_n\}$. A vote is a function $b : C \to \mathbb{N}$ that assigns a natural number (the preference) to each candidate. Note that two candidates may receive the same preference. We recover a strict total preorder $<_b$ on the candidates by setting $c <_b d$ if $b(c) > b(d)$, i.e. candidate c is less preferred over candidate d if the natural number $b(c)$ is greater than the natural number $b(d)$.
2) We construct a margin matrix $\mathrm{marg} : C \times C \to \mathbb{Z}$ as follows: given two candidates $c, d \in C$, the *margin* of c over d is the number of voters that

1. https://github.com/mukeshtiwari/Schulzeproperties/blob/master/Schulzeprop.v

prefer c over d minus the number of voters that prefer d over c. In symbols:

$$marg(c,d) = \sharp\{b \in P \mid c >_b d\} - \sharp\{b \in P \mid d >_b c\}$$

where $\sharp$ denotes cardinality.

3) A directed *path* from candidate c to candidate d is a sequence $p \equiv c_0, \ldots, c_{w+1}$ of candidates with $c_0 = c$ and $c_{w+1} = d$ ($w \geq 0$), and the *strength*, st, of path p is the minimum margin of adjacent nodes, i.e.

$$\mathrm{st}(c_0, \ldots, c_{w+1}) = \min\{\mathrm{marg}(c_i, c_{i+1}) \mid 0 \leq i \leq w\}$$

4) A generalised margin matrix, M, denote the strength of the strongest path between two candidates, i.e.

$$M(c,d) = \max\{\mathrm{st}(p) : p \text{ is path from } c \text{ to } d\}$$

5) The winning set is defined as

$$W = \{c \in C : \forall d \in C \backslash \{c\}, M(c,d) \geq M(d,c)\}$$

Our Coq formalisation models the set of candidates as a type. Moreover, it postulates that the type of candidates is finite and non-empty with decidable equality. For our purpose, the easiest way of stipulating that a type be finite is to require existence of a list containing all inhabitants of this type [6].

```
Parameter cand : Type.
Parameter cand_all : list cand.
Hypothesis cand_fin :
 forall c: cand, In c cand_all.
Hypothesis cand_not_nil :
 cand_all <> nil.
Hypothesis dec_cand :
 forall n m : cand, {n = m} + {n <> m}.
```

One easy way to achieve all of the above is to implement cand as an indictive type with one nullary constructor for each candidate.

We define a boolean function (step 5) that determines (the Schulze) election winners based on the generalised margin matrix (step 4). (M marg (length cand_all) c d in the schulze_winner definition is Coq encoding of the generalised margin matrix, M (c, d), defined in step 4. The syntactic differences between these two notations do not matter for this discussion.)

```
Definition schulze_winner
   (marg : cand * cand -> Z)
   (c : cand) := forallb (fun d =>
   (M marg (length cand_all) d c) <=?
   (M marg (length cand_all) c d))
  cand_all.
```

3. Properties of Schulze Method

We have a boolean function, *schulze_winner*, that elects the winner, but what is the evidence that this winner is indeed the real winner, intended by the voters and not because of (software) bugs in the implementation. Therefore, to address the issue of evidence, we construct a dependent type function, *wins_loses_type_dec*, that every candidate is either winner or loser. This dependent type function produces a proof, in form of data (scrutiny-sheet), that why a candidate is winner or loser, and this scrutiny-sheet can be used election auditors to independently establish that every claim is true [7], [8]. Moreover, we prove that when *schulze_winner* returns true for a candidate, then there is an evidence for this candidate to be a winner, and vice-versa, *schulze_wins_true_type*

```
Lemma wins_loses_type_dec : forall c,
(wins_type c) + (loses_type c).

Lemma schulze_wins_true_type:
 forall c : cand,
 schulze_winner c = true <->
(exists x : wins_type c,
 wins_loses_type_dec c = inl x).
```

The definition *wins_type*, a (computational) definition of sort Type that computes the winner, is fairly complex and not very intuitive, so we provide another definition *wins_prop*, a (logical) definition of sort Prop that postulate the winner, which is simple and easy to understand and proofs, *wins_prop_type* and *wins_type_prop*, that both definitions are equivalent [4]. More elaborately: the reason for having two definitions, one in Type and one in Prop, is our twofold goal: i) extracting an OCaml code from the Coq code and use the OCaml code to count the ballots from real world elections and ii) making the formalisation accessible for everyone to inspect and understand that every claim is true (this is very important in electronic voting that there is nothing in my sleeves). To achieve the first goal, we define a complicated computable definition, *wins_type*, in Type and for the second goal, we define a simple logical definition, *wins_prop*, in Prop. All the reader has to do is inspect the simple logical definition *wins_prop* to ensure that it correctly captures the notion of winner, without understanding the the complicated definition of *wins_type*, and replay the proofs of *wins_prop_type* and *wins_type_prop* in the Coq theorem prover. (During the code extraction, all the (logical) terms in sort Prop are erased while all the (computational) terms in sort Type appears in OCaml code. We can compile this OCaml code to get a machine executable, which can be used to count ballots.)

```
Definition wins_prop (c: cand) : Prop :=
 forall d: cand, exists k: Z,
  Path k c d /\
 (forall l, Path l d c -> l <= k)

Definition wins_type c : Type :=
 forall d : cand, existsT (k : Z),
 ((PathT k c d) *
  (existsT (f : (cand * cand) -> bool),
  f (d, c) = true /\ coclosed (k + 1) f))

Lemma wins_prop_type : forall c,
 wins_prop c -> wins_type c.

Lemma wins_type_prop : forall c,
 wins_type c -> wins_prop c.
```

Path is an inductive datatypes that exactly captures the notion of sequence of nodes between two candidates (step 3). More elaborately, Path k $marg$ c d a path p, sequence of candidates $p = (c, c_1, \ldots, c_w, d)$, that joins c and d and the strength of p, $\mathrm{st}(p) = \min\{\mathrm{marg}(c_i, c_{i+1}) \mid 0 \leq i \leq w\}$, is greater than or equal to k (PathT also captures the same notion as Path. The only difference is sort of Path is Prop and sort of PathT is Type and these two definitions exist for the very same reason we have two definitions of winner, *wins_type* and *wins_prop*). The definition of *coclosed* asserts that paths from every candidate to the winner is not stronger than the vice versa, i.e. if a candidate, say, c is the winner and the strength of the strongest path from c to a candidate, say, d is k, then all the paths from d to c will less than or equal to k.

Now that we have established the correctness of *schulze_winner*, i.e. it always elects the winner which is intended by the voters and not because of software bugs in the implementation. However, we can push this correctness (criterion) boundary further by evaluating our Coq implementation against the various properties, established in the Schulze's original paper [1]. Below, we prove that our implementation follows the Condorcet winner and (not finished) reversal symmetry property.

3.1. Condorcet Winner

A (weak) *Condorcet winner* is a candidate who beats or ties every other candidate in a pairwise comparison, also known as head to head competition. Recall from the previous section that the margin matrix, $marg$, stores exactly this data for every pair of candidates. Therefore, we define the Condorcet winner in Coq:

```
Definition condorcet_winner
 (marg : cand * cand -> Z)
 (c : cand) := forall d,
 marg (c, d) >= 0.
```

Informally, the definition, condorcet_winner states that if a candidate c is the Condorcet winner, then they are ranked equal or higher against every other candidate in more ballots than the vice versa. Our goal is to establish that if there is a Condorcet winner, then the Schulze method elects it. We formally state our intent in Coq as:

```
 Lemma condorcet_winner_implies_winner
    (marg : cand * cand -> Z)
    (c : cand) :
    condorcet_winner marg c ->
    schulze_winner marg c = true.
```

The proof of *condorcet_winner_implies_winner* hinges on the two key observations:

1) If a candidate c is the Condorcet winner, then the generalised margin (matrix) between c and every other candidate, say, d would be greater than or equal to 0, i.e. $M\ (c,\ d) \geq 0$.
2) If a candidate c is the Condorcet winner, then the generalised margin (matrix) between every other candidate, say, d and c would be less than or equal to 0, $M\ (d,\ c) \leq 0$.

These two key observations make the proof of the lemma condorcet_winner_implies_winner trivial because we have $M\ (d,\ c) \leq 0$ and $M\ (c,\ d) \geq 0$, hence $M\ (d,\ c) \leq M\ (c,\ d)$. Intuitively, if a candidate c is the Condorcet winner, then the strongest path between her and every other candidate, say, d would be either a direct path or a more stronger path via some other intermediate candidates (proof by induction on the path length). In both cases, we have the generalised margin between c and the other candidate d is greater than or equal to 0. Similarly, for the second observation. We encode these two key observations in Coq:

```
Lemma first_key_observation :
 forall c d n marg,
 condorcet_winner c marg ->
 M marg n c d >= 0.
```

```
Lemma second_key_observation :
 forall c d n marg,
 condorcet_winner c marg ->
 M marg n d c <= 0.
```

Proof of the last two lemmas is by induction on the path length, i.e. n [9].

3.2. Reversal Symmetry [2]

The *Reversal symmetry* is a voting method criterion which states that if the voting method has produced a unique winner, say, c based on the cast ballots, then c should not be elected if the individual choices were reversed. In context of Schulze method, we first need to define the unique winner[3], and ballot reversal.

```
 Definition unique_winner
 (marg : cand * cand -> Z)
 (c : cand) :=
 schulze_winner marg  c = true /\
 (forall d, d <> c ->
  schulze_winner marg d = false).
```

Informally, the definition of *unique_winner* states that a candidate c is a unique winner if it wins the election and every candidate other than c loses the election. We capture the ballot reversal in terms of margin matrix. For any given ballot set P, the margin between two candidates c and d is:

$$\mathrm{marg}(c, d) = \sharp\{b \in P \mid c >_b d\} - \sharp\{b \in P \mid d >_b c\}$$

If we reverse the individual choices in every ballot, the new margin matrix, denoted as rev_marg, would be:

$$\mathrm{rev\_marg}(c, d) = -\mathrm{marg}(c, d)$$

The connection between rev_marg and marg is very intuitive, but it can be understood by considering a hypothetical single ballot election. Let's assume that we have a single ballot $(A, 1); (B, 2); (C, 3)$ and the interpretation is that A is strictly preferred over B, and B is strictly preferred over C (but we do not need strict preferences to have this property). The margin matrix constructed from this ballot is:

$$\begin{array}{c c} & \begin{array}{ccc} A & B & C \end{array} \\ \begin{array}{c} A \\ B \\ C \end{array} & \begin{pmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{pmatrix} \end{array}$$

2. It is not complete and still ongoing.

3. the Schulze method satisfies the *resolvability criterion*, i.e. it elects a single winner under the assumption that number of voters are much larger than number of candidates, and in case of a tie, a random vote can be selected to declare the winner. However, our formalisation has not taken the randomness into account, so it can produce more than one winner.

After reversing the original ballot, we get $(A, 3); (B, 2); (C, 1)$ and the margin matrix is:

$$\begin{array}{c|ccc} & A & B & C \\ A & 0 & -1 & -1 \\ B & 1 & 0 & -1 \\ C & 1 & 1 & 0 \end{array}$$

It is clearly evident from this example that the connection between rev_marg and marg holds. We capture this connection in Coq as:

```
Definition rev_marg
   (marg : cand -> cand -> Z)
   (c d : cand) := -marg c d.
```

Finally, the reversal symmetry property can be expressed in Coq as:

```
Lemma reversal_symmetry : forall marg c,
  unique_winner marg c ->
  schulze_winner (rev_marg marg) c =
  false.
Proof.
 (* still ongoing *)
```

The lemma reversal_symmetry expresses that if a candidate c is the unique winner, with respect to marg computed from some ballot set P, then she is a loser with respect to rev_marg, computed from reversing all the entries in the ballot set P.

The proof this lemma is fairly straight forward [1], but Schulze's original paper assumes a key property which turns out to be very difficult to prove, at least in our encoding, in Coq. The key property is: if M is the generalised margin matrix (step 4), computed using the margin matrix marg and M_rev is the generalised margin matrix, computed using the margin matrix rev_marg, then $M\_M\_\text{rev} : \forall\ c\ d, M(c, d) = M\_\text{rev}(d, c)$ holds. Currently, we do not have the proof of M_M_rev in Coq, but we managed to prove another property, path_with_rev_marg. It states that if there is a path from c to d of strength k, with respect to marg, then we have a path of same strength k from d to c, with respect to rev_marg. If this path from c to d happens to be the strongest path (step 4), then we can prove M_M_rev [4]. The lemma path_with_rev_marg can be combined with some other properties to complete the proof of M_M_rev.

```
Lemma path_with_rev_marg :
  forall k marg c d,
  Path marg k c d <->
  Path (rev_marg marg) k d c.
```

References

[1] M. Schulze, "A new monotonic, clone-independent, reversal symmetric, and Condorcet-consistent single-winner election method," *Social Choice and Welfare*, vol. 36, no. 2, pp. 267–303, 2011.

[2] ——, "The schulze method of voting," 2020.

[3] K. J. Arrow, "A difficulty in the concept of social welfare," *Journal of political economy*, vol. 58, no. 4, pp. 328–346, 1950.

[4] D. Pattinson and M. Tiwari, "Schulze voting as evidence carrying computation," in *Proc. ITP 2017*, ser. Lecture Notes in Computer Science, M. Ayala-Rincón and C. A. Muñoz, Eds., vol. 10499. Springer, 2017, pp. 410–426.

4. The challenge in our encoding is proving that the strength of the strongest path is $\geq M(c, d)$ and the strength of all paths is $\leq M(c, d)$ to infer the equality, i.e. strength of the strongest path $= M(c, d)$

[5] R. L. Rivest and E. Shen, "An optimal single-winner preferential voting system based on game theory," in *Proc. COMSOC 2010*, V. Conitzer and J. Rothe, Eds. Duesseldorf University Press, 2010.

[6] D. Firsov and T. Uustalu, "Dependently typed programming with finite sets," in *Proceedings of the 11th ACM SIGPLAN Workshop on Generic Programming*, ser. WGP 2015. New York, NY, USA: Association for Computing Machinery, 2015, p. 33–44. [Online]. Available: https://doi.org/10.1145/2808098.2808102

[7] K. Arkoudas and M. C. Rinard, "Deductive runtime certification," *Electr. Notes Theor. Comput. Sci.*, vol. 113, pp. 45–63, 2005.

[8] R. L. Rivest, "On the notion of software independence in voting systems," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 366, no. 1881, pp. 3759–3767, 2008. [Online]. Available: https://royalsocietypublishing.org/doi/10.1098/rsta.2008.0149

[9] B. A. Carré, "An algebra for network routing problems," *IMA Journal of Applied Mathematics*, vol. 7, no. 3, p. 273, 1971.