Borda Election rules

The Borda algorithm is a simple election scheme where voters rank candidates in order of preference by giving 3 points to their first choice, 2 for their 2nd choice and 1 point for the 3rd choice.
Once all votes have been counted the candidate which has the highest number of points declared as the winner.
https://en.wikipedia.org/wiki/Borda_count

A simple loop free Solidity code and correctness rules are located in the git repository
(The correctness rules are written in the Certora Specify language which is a bit low level at the moment)
https://github.com/Certora/CertoraProverSupplementary/tree/master/Examples/Elections/Borda

Definitions:

We use the following attributes in the verification process:

`voted(address x) : bool`

The address x has already voted

`points(address c) : uint`

Returns the current number of points candidate c has.

`winner() : address`

Returns the winner of the election

We use the following state changing operation:

`vote(address x, address first, address second, address third) : bool`

Returns true when `x` has not voted before and successfully updates `points` and `voted` as required for first, second and third choices.

Specification

We use Hyper Linear Temporal logic for specifying desired properties. This is a generalization of linear temporal logic

Correctness Rules

Basic rules

Emptiness:

No user has voted if and only if all candidates have zero votes

Globally $(\forall \text{address } c \ \ \text{points}(c)=0) \Leftrightarrow (\forall \text{address } x \ \neg\text{voted}(x))$

other:

Globally $(\exists \text{address } c \ \ \text{points}(c) > 0) \Leftrightarrow (\exists \text{address } x \ \text{voted}(x))$

Notice that this Globally is the usual case of global invariant. It signals that every step of the program preserves the above invariant.

Persistency voting:

Once a user issues a vote operation, this use is marked as voted globally (for all next states)

Globally $(\text{vote}(x,f,s,t) \Rightarrow$ **Next Globally** $\text{voted}(x))$

Here we use the next operator which denotes the next execution of any API command. It is a special case of [linear temporal logic.](#)

Single vote per user

A user cannot vote if the user has voted before

Globally $(\text{voted}(x) \Rightarrow$ **Globally** $\neg\text{vote}(x,f,s,t))$

Other:

Globally $(\text{vote}(x,f,s,t) \Rightarrow$ **Next Globally** $\neg\text{vote}(x,f,s,t))$

Integrity of points:

The Points data structure is updated as required, this rule also verifies that there are three distinct candidates

$\{ \text{f\_points} = \text{points}(f) \wedge \text{s\_points} = \text{points}(s) \wedge \text{t\_points} = \text{points}(t)\}$

$\text{vote}(x,f,s,t)$

$\{ \text{points}(f) = \text{f\_points}+3 \wedge \text{points}(s) = \text{s\_pointss}+2 \wedge \text{t\_points} = \text{points}(t)+1 \}$

Here we use [Hoare triples](#) of the form $\{p\}\ C\ \{q\}$, which means that if program C executes starting in any state satisfying p, then it will end in a state satisfying q.

No effect on other candidates:

$\{c \neq \{f,s,t\} \wedge \text{c\_points} = \text{points}(c)\} \ \ \text{vote}(x,f,s,t)\ \{\text{points}(c) = \text{c\_points}\}$

No effect on unsuccessful vote operation

$\{\text{c\_points} = \text{points}(c) \wedge \ \ b = \text{voted}(y)\}$

```
        ¬vote(x,f,s,t)
    {points(c) = c_points ∧ voted(y) = b}
     Here ¬vote(x,f,s,t)  is a shorthand for false = vote(x, f, s, t)
```

Commutativity of voting

Order of votes does not change points and voted:

`vote(x,f,s,t)  ; vote(x',f',s',t')  ∼{voted,points}  vote(x',f',s',t') ; vote(x,f,s,t)`

Here we cannot require that the winner is the same in case of a tie. Different orders may result in a different winners in case of tie but the integrity winner rule should hold

This is specified in terms of code equivalence denoted by P1 ∼{voted,points} P2 which means that P1 and P2 are equivalent in terms pf voted and points.

Integrity of winner

The winner has the most points

Globally `∀address c. points(c) ≤  points(winner())`

`(To make Borda count a completely defined function we may want to define lexicographic tie breaking -- I am not sure how that works with "address"s.)`

Wikipedia rules:

[Participation criterion]()

Abstaining from an election can not help a voter's preferred choice
(Voting does not hurt a voter's preferred choice)

`{f=winner()} vote(x,f,s,t) {f=winner()}`

[Resolvability criterion]()

For every possibly tied winner in a result, there must exist a way for **one** added vote to make that winner unique

```
In Dynamic Logic:
c!=winner() ∧  points(c) = points(winner())   ⇒
 <vote(x,f,s,t)>
  ∀address c. points(c) < points(winner())
```

[Later-no-harm criterion]() (expected violation)

a voter giving an additional ranking or positive rating to a less-preferred candidate can not cause a more-preferred candidate to lose

```
exec(s, vote(x,f,s,t)) = s1
And exec(s, vote(x,f,s',t')) = s2 ⇒
        (s1.winner() = f ⇒ s2.winner() = f)
```

HyperLTL specification

Participation criterion

the addition of a ballot, where candidate A is strictly preferred to candidate B, to an existing tally of votes should not change the winner from candidate A to candidate B

$$\forall\ \pi,\pi' .\ \ \forall\ f,s,t\ .\ \text{prefer\_A\_to\_B}(f,s,t)\ \wedge\ \text{add\_ballot}(\pi,\pi',f,s,t)$$
$$\rightarrow \text{no\_change\_of\_winner\_A\_to\_B}(\pi,\pi')$$

Abbreviations:

add_ballot(π,π',f,s,t)

The elections π,π' are the same except that ballot (f,s,t) has been added to π'.

$$(\forall\ f',s',t'\ .\ \ vote_{\pi}(f',s',t') \leftrightarrow vote_{\pi'}(f',s',t'))$$
$$\textbf{Awaits}\ (vote_{\pi'}(f,s,t) \wedge \textbf{Globally}\ (\forall\ f',s',t'\ .\ \ vote_{\pi}(f',s',t') \leftrightarrow \textbf{Next}\ vote_{\pi'}(f',s',t')))$$

prefer_A_to_B(f,s,t):

The ballot (f,s,t) strictly prefers A over B

$$f{=}A \vee (f{\neq}B \wedge s{=}A) \vee (f{\neq}B \wedge s{\neq}B \wedge t{=}A)$$

no_change_of_winner_A_to_B(π,π'):

The winner does not change from A to B

$$\textbf{Globally}\ (\exists\ f',s',t'\ .\ \ \neg(vote_{\pi}(f',s',t') \leftrightarrow vote_{\pi'}(f',s',t'))$$
$$\rightarrow\ \ \textbf{Globally}\ (winner_{\pi} = A \rightarrow \textbf{Next}\ winner_{\pi'} \neq B)$$