Voting Theory in the Lean Theorem Prover

Wesley H. Holliday[1][0000−0001−6054−9052], Chase Norman[1][0000−0001−8954−3770], and Eric Pacuit[2][0000−0002−0751−9011]

[1] University of California, Berkeley
{wesholliday,c_}@berkeley.edu
[2] University of Maryland, College Park
epacuit@umd.edu

Abstract. There is a long tradition of fruitful interaction between logic and social choice theory. In recent years, much of this interaction has focused on computer-aided methods such as SAT solving and interactive theorem proving. In this paper, we report on the development of a framework for formalizing voting theory in the Lean theorem prover, which we have applied to verify all of the claimed properties of a recently proposed voting method. While previous applications of interactive theorem proving to social choice (using Isabelle/HOL and Mizar) have focused on the verification of impossibility theorems, we aim to cover a variety of results ranging from impossibility theorems to the verification of properties of specific voting methods (e.g., Condorcet consistency, independence of clones, etc.). In order to formalize voting theoretic axioms concerning adding or removing candidates and voters, we work in a variable-election setting whose formalization makes use of dependent types in Lean.

Keywords: logic and social choice theory · voting theory · interactive theorem proving · Lean

1 Introduction

There is a long tradition of fruitful interaction between logic and social choice theory. Both Kenneth Arrow [2, p. 154] and Amartya Sen [27, p. 108] have noted the influence of mathematical logic on their thinking about the foundations of social choice theory. Early work using logical methods in social choice theory includes Murakami's [22] application of results about three-valued logic to the analysis of voting rules, Rubinstein's [26] proof of the equivalence between multi-profile and single-profile approaches to social choice, and Parikh's [24] development of a logic of games to study social procedures. There is now a rich literature developing logical systems that can formalize results in social choice theory (see, e.g., [23,28,14,17,10,18]).

In recent years, research on logic and social choice theory has centered around the use of computer-aided methods [16]. Here we focus on the application of interactive theorem provers to verify results in social choice theory. The first of such applications used Isabelle/HOL [23] and Mizar [30] to formalize different proofs of Arrow's Impossibility Theorem [15]. More recently, [7] and [13] used Isabelle to verify theorems from [8] and [6], respectively. These projects demonstrate, as Nipkow [23] notes, that "social choice theory turns out to be perfectly suitable for mechanical theorem proving." In this paper, we provide further evidence of this by developing a framework for formalizing voting theory using an interactive theorem prover. One obvious benefit of such a project is the verification of the correctness of mathematical claims in voting theory. Several published claims, including Arrow's [1] original statement of his impossibility theorem (for more than 3 candidates), Baigent's [3] variation involving "weak IIA" (in the case of 3 candidates), and Routley's [25] claimed generalization of Arrow's theorem to infinite populations, were disproved by counterexamples (see [5], [9], and [4]). Second, formalization allows us to carefully track which assumptions—e.g., about voter preferences, cardinalities, choice of primitive concepts, etc.—are needed for which results, leading to generalizations and perhaps even new avenues for research. Third, formalization may eventually facilitate automated search of the corpus of proved results for use by researchers in proving new results.

For our formalization project we chose to use the Lean theorem prover [12], a framework that supports both interactive and automated theorem proving. Lean's kernel is based on dependent type theory and implements a version of the calculus of inductive constructions [11] and Martin-Löf type theory [21]. There is an extensive and actively maintained library of mathematical results formalized in Lean (see `https://leanprover-community.github.io/mathlib_docs/`). In addition, Lean is the system chosen for the Formal Abstracts project initiated by Thomas Hales (`https://formalabstracts.github.io`).

Our aim was to use Lean to verify results about axioms for voting methods (e.g., Condorcet consistency, independence of clones, etc.). In order to formalize axioms concerning adding or removing candidates and voters, we work in a variable-election setting whose formalization makes use of dependent types, as explained in Section 2. In Section 3, we discuss our formal verification of almost all the results from [19] about a recently proposed voting method, Split Cycle (defined in Example 3 below), illustrating the usefulness of our framework. We conclude in Section 4 with directions for further work. All of the code for our project is available at `https://github.com/chasenorman/Formalized-Voting`.

2 Framework

In this section, we define the basic objects of voting theory: profiles, social choice correspondences, etc. We first give standard set-theoretic definitions and then their type-theoretic counterparts in Lean syntax.

2.1 Profiles

For our set-theoretic definitions, we fix infinite sets $\mathcal{V}$ and $\mathcal{X}$ of voters and candidates, respectively. Given $X \subseteq \mathcal{X}$, let $\mathcal{B}(X)$ be the set of all binary relations on X. Instead of thinking of a binary relation as a set of ordered pairs, it is more convenient to think of a binary on X as a function $S : X \times X \to \{0, 1\}$.

In fact, to better match our Lean formalization, we "curry" all functions with multiple arguments, transforming them into functions with single arguments that output functions. Thus, we regard a binary relation on X as a function $S : X \to \{0,1\}^X$, where $\{0,1\}^X$ is the set of functions from X to $\{0,1\}$. For any $x \in X$, $S(x) : X \to \{0,1\}$, and $S(x)(y) = 1$ means that the binary relation S holds of (x, y). In what follows, we write 'xSy' instead of $S(x)(y) = 1$.

Definition 1. For $V \subseteq \mathcal{V}$ and $X \subseteq \mathcal{X}$, a (V,X)*-profile* is a map $\mathbf{Q} : V \to \mathcal{B}(X)$. We write '$\mathbf{Q}_i$' for the relation $\mathbf{Q}(i)$. Given a (V,X)-profile $\mathbf{Q}$, let $V(\mathbf{Q})$ be V and $X(\mathbf{Q})$ be X. We then define a function Prof that assigns to each pair (V,X) of $V \subseteq \mathcal{V}$ and $X \subseteq \mathcal{X}$ the set $\mathsf{Prof}(V,X)$ of all (V,X)-profiles. Finally, define $\mathsf{PROF} = \bigcup_{V \subseteq \mathcal{V}, X \subseteq \mathcal{X}} \mathsf{Prof}(V,X)$.

Depending on the application, one can interpret $x\mathbf{Q}_i y$ to mean either (i) that voter i strictly prefers x to y or (ii) that voter i strictly prefers x to y or is indifferent between x and y. Under interpretation (i), we use '$\mathbf{P}$' for a profile; under interpretation (ii), we use '$\mathbf{R}$' for a profile.[3] A profile $\mathbf{Q}$ is said to be *asymmetric* (*transitive*, etc.) if for every $i \in V$, $\mathbf{Q}_i$ is asymmetric (transitive, etc.). Of course, asymmetric profiles only make sense under interpretation (i), whereas under interpretation (ii), profiles should be reflexive.

To translate Definition 1 into Lean, we first think of V and X as types, rather than sets, and then represent the function Prof from Definition 1 as follows:[4]

```
def Prof : Type → Type → Type :=
λ (V X : Type), V → X → X → Prop
```

Here `Prop` is a special type that in this definition plays the role of $\{0,1\}$ in the treatment of binary relations mentioned above. The definition states that `Prof` is a function that given two types, `V` and `X`, outputs the type `V → X → X → Prop`. Because `X → X → Prop` is the type of binary relations on `X`, an inhabitant of the type `V → X → X → Prop` can be viewed as a (V,X)-profile. Thus, we may think of `Prof V X` as the type of (V,X)-profiles.

One of the most important kinds of information to read off from a profile is whether one candidate is majority preferred to another.

Definition 2. Given a profile $\mathbf{P}$ and $x, y \in X(\mathbf{P})$, we say that x *is majority preferred to* y *in* $\mathbf{P}$ if more voters rank x above y than rank y above x.

In Lean, we formalize Definition 2 as follows:

```
def majority_preferred {V X: Type}: Prof V X → X → X → Prop
:= λ P x y,
cardinal.mk {v : V // P v x y} > cardinal.mk {v : V // P v y x}
```

[3] Approach (ii) is more general, since it allows one to distinguish between voter i being *indifferent* between x and y, defined as $x\mathbf{R}_i y$ and $y\mathbf{R}_i x$, vs. x and y being *noncomparable* for i, defined as *neither* $x\mathbf{R}_i y$ *nor* $y\mathbf{R}_i x$. When the distinction between voter indifference and noncomparability is not needed, approach (i) can be simpler.

[4] When writing type expressions, arrows associate to the right, so, e.g., the expression '`V → X → X → Prop`' stands for `V → (X → (X → Prop))`.

Here '{V X: Type}' indicates that V and X are implicit arguments to margin of type Type. The majority_preferred function takes in explicit arguments of a (V, X)-profile and two candidates and returns the proposition stating that the cardinality of the set of voters who prefer x to y is greater than the cardinality of the set of voters who prefer y to x.

We often are concerned not only with whether one candidate is majority preferred to another but also, if so, what is the margin of majority preference.

Definition 3. Given a profile $\mathbf{P}$ and $x_1, x_2 \in X(\mathbf{P})$, the *margin of* x_1 *over* x_2 *in* $\mathbf{P}$, denoted $Margin_{\mathbf{P}}(x_1, x_2)$, is $|\{i \in V(\mathbf{P}) \mid x_1\mathbf{P}_i x_2\}| - |\{i \in V(\mathbf{P}) \mid x_2\mathbf{P}_i x_1\}|$.

In Lean, Definition 3 becomes:

```
def margin {V X: Type} [fintype V] : Prof V X → X → X → ℤ :=
λ P x₁ x₂, ↑(finset.univ.filter (λ v, P v x₁ x₂)).card -
↑(finset.univ.filter (λ v, P v x₂ x₁)).card
```

Here '[fintype V]' can roughly be understood as an implicit assumption that V is finite,[5] which we make so that we can perform the subtraction in the definition of margin. The margin function takes in explicit arguments of a (V, X)-profile and two candidates and returns the margin of the first over the second; in particular, 'finset.univ.filter (λ v, P v x₁ x₂)' is syntax for constructing $\{v \in V(\mathbf{P}) \mid x_1\mathbf{P}_v x_2\}$, .card takes the cardinality of the set (a natural number), and ↑ shifts the type from natural number to integer (so we can subtract).

As usual, we can regard the $\text{Margin}_{\mathbf{P}}$ function as an $|X(\mathbf{P})| \times |X(\mathbf{P})|$ matrix. Since $Margin_{\mathbf{P}}(x, y) = -Margin_{\mathbf{P}}(y, x)$, the matrix is skew-symmetric. Treating an integer-valued square matrix as a function from a set X to functions from X to $\mathbb{Z}$, the property of skew-symmetry takes in such a function and outputs the proposition stating that the skew-symmetry equation holds for all pairs:

```
def skew_symmetric {X: Type} : (X → X → ℤ) → Prop :=
λ M, ∀ x y, M x y = - M y x.
```

Even such a basic fact as that $\text{Margin}_{\mathbf{P}}$ is skew-symmetric needs to be proved in Lean, but the verification is straightforward using Lean's automation:

```
lemma margin_skew_symmetric {V X: Type} (P : Prof V X)
[fintype V] : skew_symmetric (margin P) :=
begin
  intro,
  intro,
  unfold margin,
  simp,
end
```

[5] See Section 3.3 of the Lean documentation and the Lean community page on Sets and set-like objects.

As the logical form of the skew-symmetric proposition is `∀ x y, Margin P x y = - Margin P y x`, we prove the claim by two applications of universal introduction using the `intro` tactic. For arbitrary `x` and `y` in `X`, we must prove `Margin P x y = - Margin P y x`. This is done by first unfolding the definition of `margin`, so that our goal is now to prove

```
 ↑(finset.univ.filter (λ v, P v x y)).card -
 ↑(finset.univ.filter (λ v, P v y x)).card
= -(↑(finset.univ.filter (λ v, P v y x)).card -
 ↑(finset.univ.filter (λ v, P v x y)).card)
```

Fortunately, Lean can prove this equation automatically using the `simp` tactic.

Turning to properties of profiles, one of the most important to consider is whether a profile has a so-called Condorcet winner or even a majority winner.

Definition 4. Given a profile $\mathbf{P}$ and $x \in X(\mathbf{P})$, we say that x is a *Condorcet winner in* $\mathbf{P}$ if for all $y \in X(\mathbf{P})$ with $y \neq x$, x is majority preferred to y in $\mathbf{P}$. We say that x is a *majority winner in* $\mathbf{P}$ if the number of voters who rank x (and only x) in first place is greater than the number of voter who do not rank x in first place.

In Lean, Definition 4 becomes:

```
def condorcet_winner {V X: Type} (P : Prof V X) (x : X) :
Prop := ∀ y ≠ x, majority_preferred P x y

def majority_winner {V X: Type} (P : Prof V X) (x : X) :
Prop := cardinal.mk {v : V // ∀ y ≠ x, P v x y} > cardinal.mk
{v : V // ∃ y ≠ x, P v y x}
```

Here the '//' notation indicates that we are identifying the subtype of voters with a certain property.

As an example of a slightly more involved proof than the one above showing that the margin matrix is skew-symmetric, we present a proof in Lean that a majority winner is also a Condorcet winner. For this we use several basic theorems provided by Mathlib, including one formalizing the fact that a subset of a set has cardinality less than or equal to that of its superset:[6]

```
theorem cardinal.mk_subtype_mono {α : Type u} {φ ψ : α → Prop}
(h : ∀ x, φ x → ψ x) :
cardinal.mk {x // φ x} ≤ cardinal.mk {x // ψ x}
```

We explain the following Lean proof that a majority winner is a Condorcet winner in detail below:

[6] We have changed variable names and replaced # with 'cardinal.mk'.

```
    lemma majority_winnner_is_condorcet {V X: Type} (P : Profile V X)
    [fintype V] (x : X) : majority_winner P x → condorcet_winner P x
    :=
    begin
 1.   intros majority z z_ne_x,
 2.   have imp1 : ∀ v, (∀ y ≠ x, P v x y) → P v x z,
 3.     {intros v ranks_x_first,
 4.     exact ranks_x_first z z_ne_x},
 5.   refine lt_of_lt_of_le _ (cardinal.mk_subtype_mono imp1),
 6.   have imp2 : ∀ v, P v z x → (∃ y ≠ x, P v y x),
 7.     {intros v ranks_z_above_x,
 8.     use [z, z_ne_x],
 9.     exact ranks_z_above_x},
10.   apply lt_of_le_of_lt (cardinal.mk_subtype_mono imp2),
11.   exact majority,
    end
```

Since the logical form of what we want to prove, `majority_winner P x` $\to$ `condorcet_winner P x`, is an implication, we use `intros` on line 1 to introduce a name `majority` for a proof of `majority_winner P x`. Then since the consequent, `condorcet_winner P x`, is a universal statement, `∀ y ≠ x, majority_preferred P x y`, we introduce a name `z` for an arbitrary candidate and a name `z_ne_x` for a proof of `z ≠ x`. Our goal is now to prove `majority_preferred P x z`.

The first key move (lines 4-6) is to prove that everyone who ranks `x` first ranks `x` above `z`. On line 5, we introduce a name `v` for an arbitrary voter and a name `ranks_x_first` for a proof of `∀ y ≠ x, P v x y`. Such a proof is a *function* that when given a `w` in the domain of ∀ and a proof of `w ≠ y` outputs a proof of `P v x w`. Thus, when given `z` and our proof `z_ne_x`, `ranks_x_first` outputs a proof of `P v x z`, which is exactly what we want, so we can apply Lean's `exact` tactic. Since `imp1` is of the form `(∀ v,` φ `v →` ψ `v)`, we can apply the Mathlib theorem `cardinal.mk_subtype_mono` to obtain a proof `cardinal.mk_subtype_mono imp1` that the number of voters who rank `x` first is less than or equal to the number of voters who rank `x` above `z`.

Next, on line 7, we use a Mathlib theorem, `lt_of_lt_of_le`, which states that $\mathtt{n} < \mathtt{m} \to \mathtt{m} \leq \mathtt{k} \to \mathtt{n} < \mathtt{k}$ (recall that implication associates to the right). Take `n` to be the number of voters who rank `z` above `x`, `m` to be the number who rank `x` first, and `k` to be the number who rank `x` above `z`. Thus, our goal is to prove $\mathtt{n} < \mathtt{k}$, and above we proved $\mathtt{m} \leq \mathtt{k}$. Now $\mathtt{m} \leq \mathtt{k}$ is not the antecedent of $\mathtt{n} < \mathtt{m} \to \mathtt{m} \leq \mathtt{k} \to \mathtt{n} < \mathtt{k}$, but Lean's `refine` tactic allows us to insert a placeholder `_` for the antecedent, so our goal then becomes proving $\mathtt{n} < \mathtt{m}$.

To prove $\mathtt{n} < \mathtt{m}$, the key move (lines 9-11) is to prove that everyone who ranks `z` over `x` does not rank `x` first. Then we can apply `cardinal.mk_subtype_mono` to obtain a proof `cardinal.mk_subtype_mono imp2` that the number `n` of voters who rank `z` above `x` is less than or equal to the number—call it $\mathtt{m}'$—of voters who do not rank `x` first. Thus, we have a proof of $\mathtt{n} \leq \mathtt{m}'$, so we can apply the implication $\mathtt{n} \leq \mathtt{m}' \to \mathtt{m}' < \mathtt{m} \to \mathtt{n} < \mathtt{m}$ provided by the Mathlib theorem `lt_of_le_of_lt` to obtain a proof of `m′ < m → n < m`. Then since `majority` is exactly a proof of the antecedent of `m′ < m → n < m`, we obtain a proof of our goal `n < m`, so we are done.

2.2 Functions on profiles

Next we define two kinds of functions that take profiles as inputs. The first, called a *social choice correspondence* (SCC), assign to a given profile a set of candidates, who are considered tied for winning the election. It is common to consider "domain restrictions" on the set of profiles for which the SCC is defined. Thus, one may define an SCC as a function on some set $\mathcal{D}$ of profiles such that for all $\mathbf{Q} \in \mathcal{D}$, we have $\emptyset \neq F(\mathbf{Q}) \subseteq X(\mathbf{Q})$. However, for our purposes, it is more convenient to use the following equivalent approach.

Definition 5. For $V \subseteq \mathcal{V}$ and $X \subseteq \mathcal{X}$, a *social choice correspondence for* (V, X), or (V, X)-SCC, is a function $F : \mathsf{Prof}(V, X) \to \wp(X)$. We abuse terminology and call the set $\{\mathbf{Q} \in \mathsf{Prof}(V, X) \mid F(\mathbf{Q}) \neq \emptyset\}$ the *domain* of the F. We say that F satisfies *universal domain* if its domain is $\mathsf{Prof}(V, X)$.

Let SCC be a function that assigns to each pair (V, X) of $V \subseteq \mathcal{V}$ and $X \subseteq \mathcal{X}$ the set of all (V, X)-SCCs.

We represent the function SCC in Lean as follows, where `set X` is the type of subsets of `X`:[7]

```
def SCC := λ (V X : Type), Prof V X → set X
```

The definition states that `SCC` is a function that given two types, `V` and `X`, outputs the type `Prof V X → set X`, which is the type of (V, X)-SCCs.

We formalize universal domain as follows:

```
def universal_domain_SCC {V X: Type} (F : SCC V X) : Prop :=
∀ P : Prof V X, F P ≠ ∅
```

Example 1. For (V, X), consider the Condorcet SCC for (V, X) defined as follows:

$$\mathrm{Cond}_{(V,X)}(\mathbf{P}) = \begin{cases} \{x\} & \text{if } x \text{ is a Condorcet winner in } \mathbf{P} \\ X(\mathbf{P}) & \text{otherwise} \end{cases}.$$

We represent this (V, X)-SCC in Lean as follows:

```
def condorcet_SCC {V X: Type} : SCC V X := λ P,
{x : X | (¬∃ y, condorcet_winner P y) ∨ condorcet_winner P x}
```

The definition states that given a (V, X)-profile $\mathbf{P}$, if there is a Condorcet winner, then output the set of all Condorcet winners (which will be a singleton), and otherwise output all candidates in `X`.

[7] When writing type expressions, function application binds more strongly than arrow, so '`Prof V X → set X`' stands for `(Prof V X) → set X`.

Most voting methods (e.g., Plurality, Borda, Instant Runoff) are defined not only for a fixed set of voters and candidates but for any set of voters and candidates, which motivates the following definition.

Definition 6. A *variable-election social choice correspondence* (VSCC) is a function F that assigns to each pair (V, X) of a $V \subseteq \mathcal{V}$ and $X \subseteq \mathcal{X}$ a (V, X)-SCC. We abuse terminology and call the set $\{\mathbf{Q} \in \mathsf{PROF} \mid F(V(\mathbf{Q}), X(\mathbf{Q}))(\mathbf{Q}) \neq \emptyset\}$ the *domain* of the F. We say that F satisfies (*finite*) *universal domain* if the domain of F includes $\{\mathbf{P} \in \mathsf{PROF} \mid V(\mathbf{P})$ and $X(\mathbf{P})$ nonempty and finite$\}$.[8]

An equivalent but perhaps more intuitive approach would define a VSCC to be a function on PROF (rather than $\wp(\mathcal{V}) \times \wp(\mathcal{X})$) such that for each $\mathbf{Q} \in \mathsf{PROF}$, we have $F(\mathbf{Q}) \subseteq X(\mathbf{Q})$;[9] abusing terminology, we could then call the set $\{\mathbf{Q} \in \mathsf{Prop} \mid F(\mathbf{Q}) \neq \emptyset\}$ the *domain* of the VSCC. However, we have presented Definition 6 above because it nicely connects with our formalization in Lean.

In Lean, we define the type of VSCCs as a *dependent function type*:

```
def VSCC : Type 1 := Π (V X : Type), SCC V X.
```

The definition states that a VSCC is a function that for any types `V` and `X` returns a function of the type `SCC V X`, i.e., a (V, X)-SCC. We formalize (finite) universal domain as follows:

```
def finite_universal_domain_VSCC (F : VSCC) : Prop :=
∀ V X [inhabited V] [inhabited X] [fintype V] [fintype X],
universal_domain_SCC (F V X)
```

Example 2. We define the Condorcet VSCC as follows, taking advantage of our definition for any V and X of the Condorcet (V, X)-SCC in Example 1:

```
def condorcet_VSCC : VSCC := λ V X, condorcet_SCC
```

The second type of function we consider assigns to a given profile a binary relation on the set of candidates in the profile.

Definition 7. For $V \subseteq \mathcal{V}$ and $X \subseteq \mathcal{X}$, a *collective choice rule for* (V, X), or (V, X)-CCR, is a function $f : \mathsf{Prof}(V, X) \to \mathcal{B}(X)$. Let CCR be a function that assigns to each pair (V, X) of $V \subseteq \mathcal{V}$ and $X \subseteq \mathcal{X}$ the set of all (V, X)-CCRs.

Depending on the application, one can interpret the binary relation $f(\mathbf{Q})$ in one of two ways: $x f(\mathbf{Q}) y$ can mean (a) x defeats y socially or (b) x defeats or is tied with y socially.[10] Once again, there is also the issue of "domain restrictions."

[8] Of course, one could also consider the stronger condition that the domain of F contains all profiles even with infinite sets of voters and/or candidates.

[9] This is the definition of a *voting method* used in [19] with the additional stipulations that $F(\mathbf{Q}) \neq \emptyset$ and that $V(\mathbf{Q})$ and $X(\mathbf{Q})$ are nonempty and finite.

[10] As in Footnote 3, approach (b) is more general, since it allows one to distinguish between "social indifference" and "social noncomparability." When notions of social indifference and noncomparability are not needed, approach (a) can be simpler.

Under approach (a), we can mark that the CCR is "undefined" on a profile $\mathbf{Q}$ by setting $f(\mathbf{Q}) = X(\mathbf{Q}) \times X(\mathbf{Q})$. Then we can abuse terminology and call $\{\mathbf{Q} \in \mathsf{Prof}(V, X) \mid f(\mathbf{Q}) \neq X(\mathbf{Q}) \times X(\mathbf{Q})\}$ the domain of f. Under approach (b), we can mark that the CCR is "undefined" on $\mathbf{Q}$ by setting $f(\mathbf{Q}) = \emptyset$. Then we can abuse terminology and call $\{\mathbf{Q} \in \mathsf{Prop}(V, X) \mid f(\mathbf{Q}) \neq \emptyset\}$ the domain of f.

A CCR f is said to be *asymmetric* (resp. *transitive*, etc.), if for all $\mathbf{Q}$ in the domain of f, $f(\mathbf{Q})$ is asymmetric (transitive, etc.). Of course, asymmetric CCRs only make sense under interpretation (a) above, whereas under interpretation (b), CCRs should be reflexive.

In Lean, our representation of the function CCR is similar to that of SCC:

```
def CCR := λ (V X : Type), Prof V X → X → X → Prop
```

Example 3. As an example of a CCR, we consider the Split Cycle CCR studied in [19]. The output of the Split Cycle CCR is an asymmetric relation understood as a relation of "defeat" between candidates. A candidate x defeats a candidate y in $\mathbf{P}$ just in case the margin of x over y is (i) positive and (ii) greater than the weakest margin in each majority cycle containing x and y. To formalize this definition, we first need a definition of a cycle in a binary relation:

```
def cycle {X: Type} := λ (R : X → X → Prop) (c : list X),
∃ (e : c ≠ list.nil), list.chain R (c.last e) c
```

Here the function `cycle` takes in a binary relation `R` and a list `c` of elements of `X` and outputs the proposition stating that (i) there is a proof `e` that `c` is not the empty list, and (ii) `c` is a cycle in `R`. To express (ii), we use the construction `list.chain R a c`, where `R` is a binary relation, `a` is an element of `X`, and `c` is a list of elements of `X`, which means that `a` is `R`-related to the first element of `c` and that every element in the list `c` is related to the next element in `c`. Thus, if we take `a` as the last element of `c`, this implies that `c` is a cycle. Applying `c.last` to the proof `e` that `c` is not the empty list outputs the last element of `c`.

Now we are ready to define the Split Cycle (V, X)-CCR:

```
def split_cycle_CCR {V X: Type} : CCR V X :=
λ (P : Profile V X) (x y : X), ∀ [f: fintype V],
0 < @margin V X f P x y ∧
¬ (∃ (c : list X), x ∈ c ∧ y ∈ c ∧
cycle (λ a b, @margin V X f P x y ≤ @margin V X f P a b) c)
```

Recall that the `margin` function takes as implicit arguments the set `V` of voters, the set `X` of candidates, and a proof that `V` is finite. The `@` symbol is used when explicitly supplying these implicit arguments. Thus, the definition states that given a profile `P` and two candidates `x` and `y`, the binary relation outputted by `split_cycle_CCR` holds of `x`, `y` if for any proof `f` that `V` is finite, the margin of `x` over `y` in `P` (supplying the `margin` function with `V`, `X`, and `f`) is greater than 0 and there is no list `c` of elements containing `x` and `y` such that `c` is a majority cycle for which the margin of `x` over `y` is less than or equal to every margin in the cycle, i.e., `c` is a cycle in the binary relation `R` that holds of `a,b` just in case the margin of `x` over `y` is less than or equal to the margin of `a` over `b`.

Once again, we can consider functions that are not restricted to a fixed set of voters and candidates.

Definition 8. A *variable-election collective choice rule* (VCCR) is a function that assigns to each pair (V, X) of a $V \subseteq \mathcal{V}$ and $X \subseteq \mathcal{X}$ a (V, X)-CCR.

An equivalent but perhaps more intuitive definition takes a VCCR to be a function f on Prof (instead of $\mathcal{V} \times \mathcal{X}$) such that for all $\mathbf{Q} \in \mathsf{Prof}$, $f(\mathbf{Q})$ is a binary relation on $X(\mathbf{Q})$.[11] However, we have presented Definition 8 above because it nicely connects with our formalization in Lean, which as before defines the type of VCCRs to be a dependent function type:

```
def VCCR := Π (V X : Type), CCR V X.
```

The definition states that a VCCR is a function that for any types `V` and `X` returns a function of the type `CCR V X`, i.e., a (V, X)-CCR.

Example 4. We define the Split Cycle VCCR as follows, taking advantage of our definition for any V and X of the Split Cycle (V, X)-SCC in Example 3:

```
def split_cycle_VCCR : VCCR := λ V X, split_cycle_CCR
```

Any VCCR, regarded as outputting a relation of strict social preference or "defeat," can be transformed into a VSCC by assigning to a given profile the set of candidates who are not defeated.

Definition 9. Given an asymmetric VCCR F, we define the *induced* VSCC $F^{\star}$ such that for any V, X, and (V, X)-profile $\mathbf{P}$, we have

$$F^{\star}(V, X)(\mathbf{P}) = \{x \in X(\mathbf{P}) \mid \neg \exists y : (y, x) \in f(V, X)(\mathbf{P})\}.$$

In Lean, we formalize Definition 9 as follows:

```
def induced_VSCC : VCCR → VSCC := λ F V X P,
{x | ∀ y : X, ¬ F V X P y x}
```

Example 5. The Split Cycle voting method [20] is the induced VSCC from the Split Cycle VCCR defined in Example 4:

```
def split_cycle : VSCC := induced_VSCC split_cycle_VCCR
```

As is well known, any acyclic VCCR induces a VSCC satisfying (finite) universal domain:

```
def acyclic {X : Type} : (X → X → Prop) → Prop :=
λ Q, ∀ (c : list X), ¬ cycle Q c

theorem induced_VSCC_universal_domain (F : VCCR)
(a : ∀ V X [inhabited V] [inhabited X] [fintype V] [fintype X]
(P : Prof V X), acyclic (F V X P)) :
finite_universal_domain_VSCC (induced_VSCC F) := ...
```

The proof can be found in our online repository.

[11] This is the definition of a VCCR used in [19] with the aditional stipulation that $V(\mathbf{Q})$ and $X(\mathbf{Q})$ are nonempty and finite.

3 Theorems

As a proof of concept of formalizing theorems in the framework above, we verified most of the results concerning the Split Cycle voting method in [19]. In particular, we verified the equivalence of two definitions of Split Cycle (one quantifying over all cycles containing x and y, the other quantifying over only paths from y to x) and that Split Cycle satisfies the following axioms: (finite) universal domain, Condorcet winner, Condorcet loser, Pareto, monotonicity, independence of clones, (strong) stability for winners, reversal symmetry, positive and negative involvement. Thus, we formalized a mix of intra-profile axioms (Condorcet winner, Condorcet loser, Pareto), inter-profile axioms (monotonicity, reversal symmetry), variable-candidate inter-profile axioms (independence of clones, stability for winners), and variable-voter inter-profile axioms (the involvement axioms).

Before formalizing voting-theoretic proofs, we had to build up basic infrastructure for reasoning about cycles, walks, and paths in graphs, such as rotating and reversing cycles and converting walks to paths, which was not available in Mathlib. For example, to convert walks to paths, we defined an inductive type:

```
noncomputable def to_path : list X → list X
| list.nil := list.nil
| (list.cons u p) :=
  let p′ := to_path p
  in if u ∈ p′
    then (p′.drop (p′.index_of u))
    else (list.cons u p′)
```

The `to_path` operation maps the empty list to the empty list, and given a list constructed by adding `u` to the front of the list `p`, if `u` is an element of `to_path p`, then we output the result of dropping from `to_path p` all elements before `u` in the list, and otherwise we add `u` to the front of `to_path p`. A significant part of the formalization effort was proving needed properties of the `to_path` operation and other operations on lists.

The most involved formalization was of the proof from [19] that Split Cycle satisfies Tideman's [29] axiom of independence of clones. A set C of two or more candidates is a set of *clones* in a profile $\mathbf{P}$ if no voter ranks any candidates outside of C in between two candidates from C. Independence of clones for VSCCs states that (i) removing a clone from a profile should not change which non-clones win and (ii) removing a clone from a profile should not change whether at least one clone is among the winners (though which clone wins is allowed to change upon removing a clone). To formalize this axiom, we need a way of removing a candidate from a profile, which is accomplished as follows:

```
def minus_candidate (P : Prof V X) (b : X) :
Prof V {x : X // x ≠ b} := λ v x y, P v x y
```

Thus, `minus_candidate` takes in a profile `P` for `V` and `X`, as well as a candidate `b` from `X`, and outputs the profile for `V` and `{x : X // x ≠ b}` that agrees with

P on how every voter ranks the candidates other than b. For the sake of space, we do not include the definitions in Lean of clones and independence of clones. The proof that Split Cycle satisfies independence of clones involves manipulating paths in the majority graph of a profile—in particular, replacing all clones in a path by a distinguished clone and then eliminating repetitions of candidates in the resulting sequence using the `to_path` operation.

4 Conclusion

As usual in formalization projects, we caught some omitted assumptions (e.g., of nonemptiness) in definitions needed to prove results about the Split Cycle voting method in [19]. A more striking fact shown by formalizing these results is how little depends on assumptions about properties of voter preferences. While it was assumed in [19] that voter preference relations are linear orders, the full strength of this assumption turned out not to be used in any proofs we formalized. In fact, most results work with no assumptions about voter preferences at all (except the default asymmetry of strict preference). The only exception was the Pareto principle, whose proof used the acyclicity of voter preferences. It would be fascinating to see exactly what properties of voter preferences are needed in formalized proofs of properties of other voting methods.

With its axiomatic approach and discrete mathematical character, voting theory is especially amenable to formal verification. Moreover, given the importance of democratic decision making in society, we find it desirable to formally verify that democratic decision procedures have the desirable properties claimed for them. We have done so for one recently proposed voting method, but we would like to see this done for all methods proposed for use in democratic elections.

References

1. Arrow, K.J.: Social Choice and Individual Values. John Wiley & Sons, Inc., New York, 1st edn. (1951)
2. Arrow, K.J.: Origins of the impossibility theorem. In: Maskin, E., Sen, A. (eds.) The Arrow Impossibility Theorem, pp. 143–148. Columbia University Press, New York (2014)
3. Baigent, N.: Twitching weak dictators. Journal of Economics **47**(4), 407–411 (1987)
4. Blau, J.H.: Review of the article "repairing proofs of arrow's general impossibility theorem and enlarging the scope of the theorem" by R. Routley. Mathematical Reviews **0545435** (1979), https://mathscinet.ams.org/mathscinet-getitem?mr=545435
5. Blau, J.H.: The existence of social welfare functions. Econometrica **25**(2), 302–313 (1957)
6. Brandl, F., Brandt, F., Eberl, M., Geist, C.: Proving the incompatibility of efficiency and strategyproofness via SMT solving. Journal of the ACM **65**(2), 6:1–6:28 (2018)
7. Brandt, F., Eberl, M., Saile, C., Stricker, C.: The incompatibility of fishburn-strategyproofness and pareto-efficiency. Archive of Formal Proofs (2018), https://isa-afp.org/entries/Fishburn_Impossibility.html

8. Brandt, F., Saile, C., Stricker, C.: Voting with ties: Strong impossibilities via SAT solving. In: Dastani, M., Sukthankar, G., André, E., Koenig, S. (eds.) Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018). pp. 1285–1293. International Foundation for Autonomous Agents and Multiagent Systems (2018)
9. Campbell, D.E., Kelly, J.S.: Weak independence and veto power. Economics Letters **66**, 183–189 (2000)
10. Cinà, G., Endriss, U.: Proving classical theorems of social choice theory in modal logic. Autonomous Agents and Multi-Agent Systems **30**(5), 963–989 (2016)
11. Coquand, T., Huet, G.: The calculus of constructions. Information and Computation **76**(2-3), 95–120 (1988)
12. de Moura, L., Kong, S., Avigad, J., van Doorn, F., von Raumer, J.: The lean theorem prover. In: 25th International Conference on Automated Deduction (CADE-25) (2015)
13. Eberl, M.: Verifying randomised social choice. In: International Symposium on Frontiers of Combining Systems, FroCoS 2019. pp. 240–256 (2019)
14. Endriss, U.: Logic and social choice theory. In: Gupta, A., van Benthem, J. (eds.) Logic and Philosophy Today, pp. 333–377. College Publications, London (2011)
15. Geanakoplos, J.: Three brief proofs of Arrow's theorem. Economic Theory **26**(1), 211–215 (2005)
16. Geist, C., Peters, D.: Computer-aided methods for social choice theory. In: Endriss, U. (ed.) Trends in Computational Social Choice. pp. 249–267. AI Access (2017)
17. Grandi, U., Endriss, U.: First-order logic formalisation of impossibility theorems in preference aggregation. Journal of Philosophical Logic **42**(4), 595–618 (2013)
18. Holliday, W.H., Pacuit, E.: Arrow's decisive coalitions. Social Choice and Welfare **54**, 463–505 (2020)
19. Holliday, W.H., Pacuit, E.: Axioms for defeat in democratic elections (2020), arXiv:2008.08451
20. Holliday, W.H., Pacuit, E.: Split Cycle: A new Condorcet consistent voting method independent of clones and immune to spoilers (2020), arXiv:2004.02350
21. Margin-Löf, P.: Intuitionistic type theory. Bibliopolis, Napoli (1984)
22. Murakami, Y.: Logic and Social Choice. Dover Publications, New York (1968)
23. Nipkow, T.: Social choice theory in HOL. Journal of Automated Reasoning **43**(3), 289–304 (2009)
24. Parikh, R.: The logic of games and its applications. In: Karplnski, M., van Leeuwen, J. (eds.) Topics in the Theory of Computation, North-Holland Mathematics Studies, vol. 102, pp. 111–139. North-Holland (1985)
25. Routley, R.: Repairing proofs of arrow's general impossibility theorem and enlarging the scope of the theorem. Notre Dame Journal of Formal Logic **20**(4), 879–890 (1979)
26. Rubinstein, A.: The single profile analogues to multi profile theorems: Mathematical logic's approach. International Economic Review **25**(3), 719–730 (1984)
27. Sen, A.: Collective Choice and Social Welfare: An Expanded Edition. Harvard University Press, Cambridge, Mass. (2017)
28. Tang, P., Lin, F.: Discovering theorems in game theory: Two-person games with unique pure nash equilibrium payoffs. Artificial Intelligence **175**, 2010–2020 (2011)
29. Tideman, T.N.: Independence of clones as a criterion for voting rules. Social Choice and Welfare **4**, 185–206 (1987)
30. Wiedijk, F.: Arrow's impossibility theorem. Formalized Mathematics **15**(4), 171–174 (2007)