Proving the Existence of Stable Assignments in Democratic Forking Using Isabelle/HOL*

Jan-Georg Smaus[1]

[1]IRIT, Université de Toulouse, CNRS, Toulouse INP, UT3, Toulouse, France

Abstract

We consider a recent game theory paper, on democratic forking: suppose you want to go to the opera with a group of 10 friends, with a choice of two operas. Everyone has a preference for one of the operas, but everyone also prefers a large community over a small one. Is there an assignment of each friend to one opera that is stable? The paper answers the question affirmatively giving an algorithm.

We have formalised the main result of that paper in the proof assistant Isabelle. This provides proofs of the results that are much more reliable, since computer-checked, than paper-and-pencil proofs. The exercise has also been useful for improving the paper-and-pencil proof. We also contribute some results concerning the uniqueness of assignments.

Keywords

Higher-order logic, interactive theorem proving, Isabelle, game theory, social choice, democratic forking

1. Introduction

About three years ago, a formalism in the field of social choice (a subfield of game theory), was presented: democratic forking [2]. The scenario and the developed formalism are remarkably simple: We have n agents who have to decide between two alternatives A or B, say *(A)ida* and *(B)ohème*. The agents each have a preference concerning the alternatives, but they also prefer a large group over a small one. The agents might stay all together or they might fork into two separate groups. The authors have political/societal issues in mind, such as adherence to political parties, cryptocurrencies [3] or open-source coding projects [4].

An *assignment* is a function assigning each agent to an alternative. It is *stable* if no agent has an incentive to switch. The main question is: can a stable assignment be found? The authors give an affirmative answer via an algorithm with several variants, and also address issues such as *strategy-proofness*.

This seemingly simple formalism quickly leads to non-trivial considerations. E.g., while it is relatively easy to see that stable assignments exist in the case of two alternatives, they do not always exist for three or more alternatives.

In this paper, we present work on implementing the proposed formalism in the proof assistant Isabelle/HOL [5]. Specifically, we have proven the first and main theorem of the paper, stating

ICTCS'24: Italian Conference on Theoretical Computer Science, September 11–13, 2024, Torino, Italy

* An extended version of this work, containing all the proofs, is available as a technical report [1].

smaus@irit.fr (J. Smaus)
https://www.irit.fr/~Jan-Georg.Smaus/ (J. Smaus)
0000-0003-3938-7081 (J. Smaus)

© 2024 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR-WS.org/Vol-3811/paper250.pdf

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

that a stable assignment exists.

Isabelle/HOL [5] is an *interactive theorem prover* (proof assistant) based on higher-order logic (HOL). You can think of HOL as a combination of a functional programming language with logic. Isabelle/HOL aims at being readable by humans and thus follows the usual mathematical notation. We use standard Isabelle/HOL in Isar proof style [6] without any special libraries.

More generally, *theorem provers* are software for conducting mathematical proofs on a computer to obtain a high degree of confidence [7].

Initially we intended to take the main theorem of [2] and its paper-and-pencil proof and "just" implement a corresponding Isabelle/HOL proof. In the end, we also implemented some additional results concerning uniqueness of assignments and elicitation (see Sec. 4). On the game theory side, our contributions are: we (1) show that for certain preferences, there is a unique stable assignments even for more than two alternatives; (2) spell out a lemma on elicitation (Lemma 2); (3) weaken the assumptions of [2, Prop. 2] on uniqueness; (4) discovered that *totality* of preferences is not required, except possibly for the uniqueness characterisation result of Subsec. 4.4.

We have modelled our development as closely as possible on our source [2], yet in some places the definitions are too imprecise, so we had to come up with reasonable precise definitions ourselves, which we consider to be a contribution of this work on the theory side. Generally speaking, the framework lent itself very well to formalisation in Isabelle.

The rest of this paper is organised as follows: the next section defines the basic concepts. Section 3 presents the algorithm for finding a stable assignment and the proof of this fact. Section 4 gives additional results concerning uniqueness and elicitation. Section 5 concludes. For space reasons, some of the proofs are not presented here; a technical report is available [1]. The Isabelle code can be found at https://www.irit.fr/~Jan-Georg.Smaus/ICTCS2024/Forking.thy.

2. Basic Definitions

In this section we recall the formalism of [2], but we point out some places where we give more explicit concepts or definitions.

Definition 1.

- *Consider a set of* alternatives $\{A, B\}$ *and a set of* voters[1] $V = \{v_1, \dots, v_n\}$.
- *A* community *is an element of* $\mathcal{S} := \{A, B\} \times \mathbb{N}$, *i.e., an alternative paired with the number of voters who have chosen this alternative.*
- *A* preference *is a subset of* $\mathcal{S} \times \mathcal{S}$, *i.e., a relation on communities. We assume that preferences are* monotonic, *i.e.,* $(A, n) \succ (A, n-1) \succ \ldots \succ (A, 1)$ *and* $(B, n) \succ (B, n-1) \succ \ldots \succ (B, 1)$ *(and antisymmetric and transitive).*
- *A* preference profile *is an element of* $(2^{\mathcal{S} \times \mathcal{S}})^n$. *It states the preferences of all voters. We denote by* $\succ_i$ *the preference of voter* v_i.

Example 1. *Consider* $V = \{v_1, v_2\}$, *where the preference profile is given by* $v_1 : (A, 2) \succ_1 (A, 1) \succ_1 (B, 2) \succ_1 (B, 1)$ *and* $v_2 : (B, 2) \succ_2 (B, 1) \succ_2 (A, 2) \succ_2 (A, 1)$. *E.g.,* v_1 *prefers being alone with* A *to being in a group of 2 with* B.

[1] We use this word rather than "agent" to avoid the letter a, to be confused with A.

At this point we state an amazing observation: in [2] it is assumed throughout that the preferences are *total* orders. It seems like a natural assumption to make and we initially added it to our Isabelle code as well. We later observed that this assumption is never used, except for one result stated in Subsec. 4.4.

In Isabelle, we have a so-called *theory file* containing all definitions and theorems, with their proofs, of our development. For modelling the above definitions, the theory file contains the following:

```
type_synonym 'Al community = "'Al * nat"
type_synonym 'Al preference = "('Al community) rel"
definition
  monotonic :: "'Al preference ⇒ bool"
  where "monotonic pref = (∀S j k. k≤j ⟶ ((S,j),(S,k)) ∈ pref)"
```

In contrast to Def. 1, the set of alternatives is not fixed to being $\{A, B\}$ but rather it is modelled by a type parameter `'Al` for the sake of generality.

Note that the above lines *define* monotonicity but do not yet impose that all preferences used should *actually be* monotonic.

Definition 2. *An* assignment *is a function* $f : V \rightarrow \{A, B\}$. *It can be lifted to a* community assignment, *i.e., for any voter* $v \in V$, *the pair* $(f(v), |f^{-1}(f(v))|)$ *gives the community of* v.

We often denote an assignment as $\langle V_A, V_B \rangle$ where V_A and V_B are the sets of voters mapped to A and B respectively.

In Isabelle, these definitions look as follows:

```
type_synonym ('V,'Al) assignment = "'V ⇒ 'Al"
definition
  commAss ::"('V,'Al) assignment ⇒ 'V ⇒ 'Al community"
  where "commAss f v = (f v, card (f -` {f v}))"
```

In Isabelle/HOL, `-`` gives the inverse of a function, so `f -` {f v}` is the set of voters mapped to `f v` by `f`, and `card (f -` {f v})` is the cardinality of that set.

Given an assignment f, a subset of voters may be interested in *deviating*, i.e., there may be an assignment f' such that for all voters in the subset, f' assigns them to another alternative, and they all prefer their "f'-community" over their "f-community". Formally we define:

Definition 3.

- *Given an assignment* $f : V \rightarrow \{A, B\}$, *a* deviation *is an assignment* $f' : V \rightarrow \{A, B\}$ *such that* $f \neq f'$ *and for all voters* v_i *where* $f(v_i) \neq f'(v_i)$,

$$(f'(v_i), |f'^{-1}(f'(v_i))|) \succ_i (f(v_i), |f^{-1}(f(v_i))|). \qquad (1)$$

- *If no deviation exists, then an assignment is* stable.

Example 2. *Consider again Ex. 1. Each voter has an alternative at which she definitely wants to be. Thus, the only stable assignment is $\langle\{v_1\}, \{v_2\}\rangle$, and the community of v_1 is $(A, 1)$ and that of v_2 is $(B, 1)$.*

Before we present the Isabelle versions of these definitions, we introduce the convenient Isabelle feature of *locales*. A locale allows us to fix a certain preference profile and make certain assumptions that are also made in [2], e.g., that the voter set is finite and that the preferences are monotonic:

```
locale GoodPreferences =
  fixes preferenceProfile :: " 'V ⇒ ('Al preference)"
  assumes finite_V : "finite (UNIV::'V set)"
  and monotonic_preferences: "∀v. monotonic (preferenceProfile v)"
  and antisym_preferences: "∀v. antisym (preferenceProfile v)"
  and trans_preferences: "∀v. trans (preferenceProfile v)"
begin
```

UNIV denotes the universal set, which can be enhanced with a type declaration, so that *finite* (*UNIV*:: *'V set*) says: the voter set is finite.

We could have also fixed the alternative set to $\{A, B\}$ here, but we decided to make the assumption of a two-element alternative set only where needed.

There is an Isabelle syntax feature allowing us to write $c \succeq_v c'$ rather than $(c, c') \in$ *preferenceProfile* v.

Now the Isabelle version of Def. 3:

```
definition
  deviation :: "('V,'Al) assignment ⇒ ('V,'Al) assignment ⇒ bool"
  where "deviation f' f = (f'≠f ∧
         (∀v. (f' v) ≠ (f v) ⟶ (commAss f' v) ⪰_v (commAss f v)))"
definition
  stable :: "('V,'Al) assignment ⇒ bool"
  where "stable f = (∄f'. deviation f' f)"
```

Observe that the use of $\succeq_v$ in the definition of *deviation* means that the definition depends on the preference profile, which is not visible in the declared type of *deviation* since an element of the type *('V, 'Al) assignment* does not carry any information about the preferences in it. This is possible thanks to the locale, fixing a preference profile. Around 80% of the lines of our Isabelle code are within the scope of the locale.

3. An Algorithm for Finding a Stable Assignment

Given a (monotonic) preference profile, one may wonder whether a stable assignment always exists and if yes, how it can be found. In [2], the algorithm shown in Fig. 1 is presented to answer both questions affirmatively (it is Thm. 1 there[2]):

[2]Note however that in [2] it is also said that the algorithm is polynomial.

```
V_A = V, V_B = ∅, a ← |V_A|, b ← |V_B|
while true do
  k ← max{j : 0 ≤ j ≤ a, |{v_i ∈ V_A : (B, b + j) ≻_i (A, a)}| ≥ j}
  if k = 0 then
    return {V_A, V_B}
  else
    let X = {v_i ∈ V_A : (B, b + k) ≻_i (A, a)}
    V_B ← V_B ∪ X, V_A ← V_A \ X
    a ← |V_A|, b ← |V_B|
  endif
endwhile
```

Figure 1: Algorithm 1

Theorem 1. *For any monotonic profile, there is a stable assignment.*

Incidentally, the presentation in [2] contains a mistake (clear from the surrounding text) which we corrected here: it says "$v_i \in V$" instead of "$v_i \in V_A$".

To understand the algorithm better, it is useful to refine the notions of deviation and stability by restricting the direction in which voters move:

Definition 4.

- *Given an assignment f, an A-*deviation *is a deviation f' such that for any v_i with $f'(v_i) \neq f(v_i)$, we have $f'(v_i) = A$.*
- *If no A-deviation exists, then an assignment is A-*stable.

In analogy we define these notions for B.

In Isabelle these definitions look as follows:

```
definition
  dev_alt :: "('V,'Al) assignment ⇒ ('V,'Al) assignment ⇒ 'Al⇒bool"
  where "dev_alt f' f S =
          (deviation f' f ∧ (∀v. (f' v) ≠ (f v) ⟶ f' v = S))"
definition
  stable_alt :: "('V,'Al) assignment ⇒ 'Al ⇒ bool"
  where "stable_alt f S = (∄f'. dev_alt f' f S)"
```

The following Isabelle lemma, whose proof has 24 lines, states that for the two-alternative case, A-stability and B-stability together imply stability:

```
lemma stabilityAB :
  assumes "stable_alt f A" and "stable_alt f B"
  and "∀x. (x=A) = (x≠B)"
shows "stable f"
```

The third assumption states that there are exactly two distinct alternatives.

So the algorithm initially places all voters in the A-community, and the while-loop computes a sequence of B-deviations. For any threshold value $j \leq a$, the set $\{v_i \in V_A : (B, b+j) \succ_i (A, a)\}$ is the set of all voters who would switch from A to B under the condition that, well, the size of the B-community grows by at least j voters. If the cardinality of that set is $\geq j$, then a B-deviation exists, and the algorithm actually uses the maximum value of j in each step.

While the algorithm uses the maximal possible B-deviation in each step, its correctness and termination are guaranteed even for arbitrary B-deviations.

3.1. A Paper-and-pencil Version of the Induction Step

The Isabelle/HOL implementation of the formalism of [2] is the main contribution of this paper. However, this endeavour allowed us to spell out in detail the paper-and-pencil version of the "induction step" of the proof of Thm. 1.

Concerning the correctness of the algorithm, the proof of the theorem given in [2] says: "Consider Algorithm 1. Let $a = |V_A|$ and $b = |V_B|$. Initially, we place all agents in V_A [...]. If this assignment is not stable, then there exists a subset X of some $k > 0$ agents that all prefer (B, k) to (A, n). We move all these agents to V_B . **Monotonicity implies that moving additional agents from V_A to V_B will never cause agents in V_B to want to move back to V_A**; [...]"

So at the heart of the correctness proof for the algorithm lies an "induction step" saying that due to monotonicity, there could not possibly occur a situation where voters would want to move (back) to A. This induction step is presented in [2] as if it were obvious, but in our opinion it is not! After all, the argument does not generalise to three alternatives A, B, C (see Ex. 4).

We formalise the induction step as the following lemma:

Lemma 1. *Let f be an A-stable assignment and f' be a B-deviation of f. Then f' is A-stable.*

Developing the paper-and-pencil proof of this induction step helped us implement the Isabelle/HOL proof and vice versa. Before we present the paper-and-pencil proof, we give a brief outline: Our proof assumes, for the purpose of deriving a contradiction, that after some "forward" step (voters moving to B), there are voters moving (back) to A. If this "backward" step only concerns voters just arrived from A, then the fact that they want to go back to A contradicts them wanting to go to B in the "forward" step. If this "backward" step concerns at least one voter previously in B, it turns out that this voter would have wanted to move to A even before the "forward" step, which breaks our induction hypothesis.

Proof. (Lemma 1) Throughout this proof, we frequently use monotonicity of preferences, and for this, knowing the set inclusions between certain sets of voters is crucial. With the help of some diagrams (see Fig. 2 and 3), these inclusions are easy to see.

The symbol $\uplus$ denotes disjoint union.

In this proof we consider a hypothetical "backward step" leading to assignment f''. We use a three-letter notation for voter sets, indicating in what community the voters are according to f,

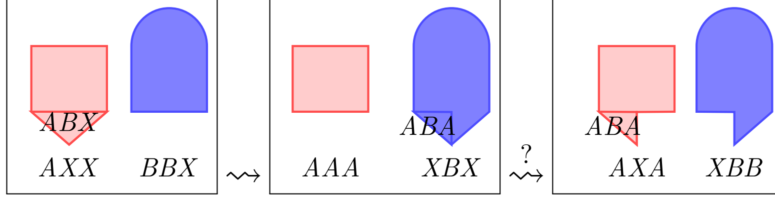

Figure 2: Forward step and backward step, case 1

f', and f'' respectively. E.g., $ABX := \{v \mid f(v) = A \wedge f'(v) = B\}$ is the set of voters moving from A to B in the forward step with nothing specified for the backward step.

Some voters move from A to B as illustrated by the boxes linked by $\rightsquigarrow$ in Fig. 2, where $AXX := f^{-1}(A)$, $BBX := f^{-1}(B)$, $AAA = f'^{-1}(A)$[3], $XBX = f'^{-1}(B)$, $ABX := \{v \mid f(v) = A \wedge f'(v) = B\}$. For every voter $u \in ABX$, since she moved, we have

$$(B, |XBX|) \succ_u (A, |AXX|) \tag{2}$$

(we suggest to look at Fig. 2 again).

For the purpose of deriving a contradiction, assume there was a "backward step" from B (possibly back) to A. We denote by f'' the assignment after this backward step.

Case 1: The backward step only concerns voters originally in A, i.e., $\{v \mid f'(v) = B \wedge f''(v) = A\} \subseteq ABX$, or equivalently, $\{v \mid f'(v) = B \wedge f''(v) = A\} = \{v \mid f(v) = A \wedge f'(v) = B \wedge f''(v) = A\} =: ABA$. This case is illustrated in Fig. 2 and we write $AXA := f''^{-1}(A)$, $XBB := f''^{-1}(B)$.

For every voter $u \in ABA$, since she moved, we have

$$(A, |AXA|) \succ_u (B, |XBX|). \tag{3}$$

By monotonicity

$$(A, |AXX|) \succ_u (A, |AXA|). \tag{4}$$

By transitivity, (2) and (4) imply that

$$(B, |XBX|) \succ_u (A, |AXA|).$$

This gives a contradiction to (3).

Case 2: The "backward step" concerns at least one voter originally in B, i.e., $\{v \mid f'(v) = B \wedge f''(v) = A\} \setminus ABX \neq \emptyset$. This case is illustrated in Fig. 3 and we write $XXA := f''^{-1}(A)$, $XBB := f''^{-1}(B)$, $BBA := \{v \mid f(v) = f'(v) = B \wedge f''(v) = A\}$.

[3] Note that by construction $f'^{-1}(A) \subseteq f^{-1}(A)$ and $f'^{-1}(A) \subseteq f''^{-1}(A)$, so we write AAA and not XAX.

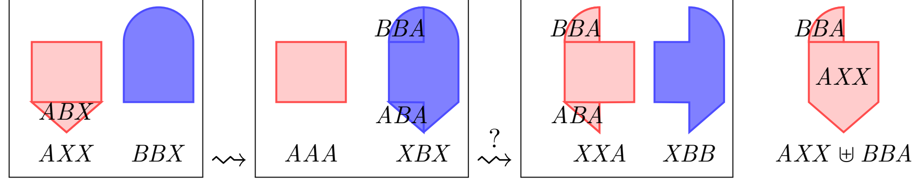

Figure 3: Forward and backward step, case 2

For every voter $u \in BBA$, since she moved, we have

$$(A, |XXA|) \succ_u (B, |XBX|). \tag{5}$$

We now consider a further set, namely $AXX \uplus BBA$, i.e., the voters originally in A, plus the voters originally in B that move to A in the backward step. It is also illustrated in Fig 3 all to the right. Observe that $XXA \subseteq AXX \uplus BBA$, and so by monotonicity we have that for every voter, in particular any $u \in BBA$,

$$(A, |AXX \uplus BBA|) \succ_u (A, |XXA|), \tag{6}$$

and again by monotonicity,

$$(B, |XBX|) \succ_u (B, |BBX|). \tag{7}$$

By transitivity, (5)-(7) imply that

$$(A, |AXX \uplus BBA|) \succ_u (B, |BBX|)$$

which contradicts the assumption that f is A-stable. In short, if there was such a voter u, she would have wanted to move to A even before the “forward step” took place.

So in any case, the assumption that there was a “backward step” leads to a contradiction. □

3.2. Induction Step in Isabelle

In Isabelle, Lemma 1 looks as follows:

```
lemma staysStable :
  assumes "stable_alt f A"
  and "∀x. (x=A) = (x≠B)"
  and "dev_alt f' f B"
  shows "stable_alt f' A"
```

The proof of this lemma has around 120 lines of Isabelle proofscript but it relies on numerous auxiliary lemmas. It is the main contribution of this work. Some of the additional overhead in the Isabelle proof is related to the subset relations among the various sets of voters and the arities of these sets. Such set-theoretic reasoning is considered trivial in paper-and-pencil proofs but must be spelt out in the smallest detail in Isabelle.

3.3. The Entire Algorithm

The previous two subsections talk about deviations (Def. 3) and B-deviations (Def. 4). These definitions do not tell us how to *construct* an f', given f. In contrast, Algorithm 1 calculates a number k and then a set X based on k, which translates into an assignment. But is this assignment really a B-deviation? On the paper-and-pencil level this might be considered "obvious by construction" but in Isabelle, it requires some overhead that we outline here.

Given an assignment f, the[4] alternative B and a natural number j, we define the set of voters who would switch to B provided the size of the B-community grows by at least j. We also define the maximum j such that the cardinality of that set is greater or equal to j, exactly as this is done in the algorithm:

```
definition
  deviators :: "('V,'Al) assignment ⇒ 'Al ⇒ nat ⇒ 'V set"
  where "deviators f B j = { v. f v ≠ B ∧
         (B, card (f -` {B}) + j)  ⪰v  (f v, card (f -` {f v}))}"
definition
  dev_max :: "('V,'Al) assignment ⇒ 'Al ⇒ nat"
  where "dev_max f B = Max {j. card (deviators f B j) ≥ j}"
```

We continue our Isabelle implementation of the algorithm, which is naturally in a functional programming style, as Isabelle/HOL is based on the typed λ-calculus [8]. We define the assignment making all voters in the maximal deviators set switch towards B, and letting all other voters stay where they are:

```
definition
  max_dev_alt :: "('V,'Al) assignment ⇒ 'Al ⇒ ('V,'Al) assignment"
  where "max_dev_alt f B =
         (λv. if v∈deviators f B (dev_max f B) then B else f v)"
```

If the deviators set is empty, then this assignment will be identical to f and we have reached a fixpoint, which is the break condition of the algorithm loop. In this case, the assignment reached is B-stable:

```
lemma max_dev_alt_is_stable :
  assumes "max_dev_alt f B = f"
  shows "stable_alt f B"
```

which takes 20 lines to prove using some auxiliary lemmas.

The function `algo` iterates `max_deviat_alt`, starting with the initial assignment $\lambda v.A$, i.e., the assignment assigning all voters to A:

[4] The lemmas are formulated for Algorithm 1, which moves voters from A to B, but technically, A and B are just variables, so the lemmas also hold for A and B swapped.

```
fun algo
  where
  "algo 0 A B = (λv. A)"
| "algo (Suc j) A B = max_dev_alt (algo j A B) B"
```

This function is written using the *function package* of Isabelle/HOL which provides excellent support for proving termination and other things [9, 10].

The following lemma completes the induction proof based on Lemma 1 (*staysStable*); the computed assignment is A-stable at all times:

```
lemma staysStableInduct :
  assumes "∀x. (x=A) = (x≠B)"
  shows "stable_alt (algo n A B) A"
```

The following lemma states that after at most a number of iterations corresponding to the number of voters, a fixpoint is reached:

```
lemma is_fixpoint :
  shows "algo (Suc (Suc (card (UNIV::'V set)))) A B =
         algo (Suc (card (UNIV::'V set))) A B"
```

For space reasons, this paper does not present any Isabelle/HOL *proofs*, except the proof of the Isabelle version of Thm. 1. This proof has just five lines, and is based on some comprehensible lemmas stated above, so we consider it the occasion to show how lemmas are tied together in an Isabelle/HOL proof.

The theorem states that after at most a number of iterations corresponding to the number of voters, the algorithm has computed a stable assignment. In the proof, the successor of the cardinality of the voter set is abbreviated by the macro *?n1* for readability:

```
lemma algo_finds_stable_assignment :
  assumes "∀ x. (x=A) = (x≠B)"
  shows "stable (algo (Suc (card (UNIV::'V set))) A B)"
    (is "stable (algo ?n1 A B)")
proof -
  have  "stable_alt (algo ?n1 A B) A"
    using assms by (rule staysStableInduct)
  moreover have "algo (Suc ?n1) A B = algo ?n1 A B"
    by (rule is_fixpoint)
  hence "max_dev_alt (algo ?n1 A B) B = algo ?n1 A B" by simp
  hence "stable_alt (algo ?n1 A B) B" by (rule max_dev_alt_is_stable)
  ultimately show ?thesis using assms by (rule stabilityAB)
qed
```

Note that the third line of the proof effectively uses the recursive definition of *algo* given above.

This completes the presentation of the Isabelle/HOL development concerning Algorithm 1 and thus the main result of [2]. It consists of just under 1000 lines.

Unlike [2], we have not assumed that preferences are total. In the extreme case, we could have preferences where any community $(A, _)$ would be unrelated to $(B, _)$. Then there could be no deviations (it can never be better to switch to another alternative) and thus any assignment is trivially stable. So to make our results *interesting*, at least some $(A, _)$, $(B, _)$ should be related.

4. Additional Results: Uniqueness and Elicitation

4.1. Clear Preferences and Unique Assignments

In [2], a *non-interleaving* preference is defined as a preference $\succ$ such that either $(A, 1) \succ (B, n)$ or $(B, 1) \succ (A, n)$. It is stated as "Observation 1" with three lines of proof that if all preferences are non-interleaving, then the unique stable assignment is the one that assigns a voter v_i to A if and only if $(A, 1) \succ_i (B, n)$.

We found that this result can be generalised to an arbitrary, even infinite, number of alternatives, so we make this assumption, in this subsection only.

We say that a preference is *clear* if there exists an alternative that is preferred to any other alternative even for the smallest, i.e., one-member, community. Note that this is a generalisation of "non-interleaving" to an arbitrary number of alternatives. It is defined in Isabelle as follows:

```
definition
  clear :: "'Al preference ⇒ bool"
  where "clear p = (∃S. ∀T. S≠T ⟶ (∀n. ((S,1),(T,n)) ∈ p))"
```

E.g., for three alternatives, $(A, 2) \succ (A, 1) \succ (B, 2) \succ (C, 2) \succ (B, 1) \succ (C, 1)$ is a clear preference (for A). The example also illustrates that the name "non-interleaving" would have been inappropriate for more than two alternatives.

For space reasons we only present the final result, stating that any stable assignment is necessarily the assignment mapping each voter to her clearly preferred alternative. This additional development consists of about 130 lines.

```
lemma genericUniqueStable :
  assumes "∀v. clear (preferenceProfile v)" and "stable f"
shows "f = (λv. THE S. ∀T. S≠T ⟶ (∀n. (S,1) ⪰v (T,n)))"
```

4.2. Non-critically-interleaving Preferences for Elicitation

A somewhat weaker condition than "non-interleaving" is also proposed by [2]:

Definition 5. *A preference is* non-critically-interleaving *if* $(A, t) \succ (B, n) \succ (B, n - t) \succ (A, t - 1)$ *or* $(B, t) \succ (A, n) \succ (A, n - t) \succ (B, t - 1)$ *for some* $t \in [1, n]$. *We say in this case that* (A, t)*, resp.* (B, t)*, is the* threshold *of* $\succ$.[5] *A profile is non-critically-interleaving if it contains only such preferences.*

[5]This sentence has been added by us.

The essential point is this: for a voter, there is a threshold value t at which she would definitely want to be at her preferred alternative. Note that for $t = 1$, the definition stipulates $\cdots \succ (A, 0)$; this spurious part simply disappears.

Non-critically-interleaving preferences facilitate elicitation. The threshold of each voter is sufficient for deciding the condition $(B, b + j) \succ_i (A, a)$ of Algorithm 1. We contribute the following lemma.

Lemma 2. *Consider $a, b, i \in [1, n]$ such that $a + b = n$ and $0 < i \leq a$ and a non-critically-interleaving preference $\succ$.*

If the threshold of $\succ$ is (B, t) for some $t \leq b + i$, or (A, t) for some $t > a$, then $(B, b + i) \succ (A, a)$. Otherwise, $(A, a) \succ (B, b + i)$.

In Isabelle the lemma looks as follows. This development has around 80 lines.

```
lemma nci_elicitation :
  assumes "a+b= card (UNIV::'V set)" (is "a+b=?n") and "j ≤ a"
  and "monotonic pref" and "trans pref"
  and "threshold pref B t A ∨ threshold pref A t B"
shows "if ((threshold pref B t A ∧ t ≤ b+j) ∨
          (threshold pref A t B ∧ t > a))
      then ((B,b+j),(A,a)) ∈ pref else ((A,a),(B,b+j)) ∈ pref"
```

4.3. Strong Preferences for Uniqueness

[2, Prop. 2] states conditions for uniqueness of assignments, assuming non-critically-interleaving preferences. We make the assumptions both *weaker* (we do not require non-critically-interleaving preferences) and *more explicit*.

Below, V'_A (V'_B) denotes the set of voters with preferred alternative A (B) who will definitely be with A (B) in any stable assignment. Our definition of these sets is equivalent to that of [2] but makes explicit that these are the voters who move in the *first step* of Algorithm 1.

Proposition 1. *Let*

$$\begin{aligned} k &:= \max\{j : 0 \leq j \leq n,\ |\{v_i \in V : (B, j) \succ_i (A, n)\}| \geq j\} \\ V'_B &:= \{v_i \in V : (B, k) \succ_i (A, n)\} \end{aligned}$$

and V'_A be defined analogously. If for each voter $v_i \in V \setminus (V'_A \cup V'_B)$, either $(B, |V'_B| + 1) \succ_i (A, n - |V'_B|)$ or $(A, |V'_A| + 1) \succ_i (B, n - |V'_A|)$, then there is a unique stable assignment.

Example 3. *Consider $n = 9$ and suppose that*

$$\begin{aligned} &(A,1) \succ_1 (B,9),\ \succ_2 = \succ_1,\ (A,2) \succ_3 (B,9),\ (A,5) \succ_4 (B,9),\ \succ_5 = \succ_4, \\ &(B,2) \succ_6 (A,9),\ \succ_7 = \succ_6, \\ &(B,3) \succ_8 (A,7), \qquad (A,6) \succ_9 (B,4) \end{aligned}$$

Then $V'_A = \{v_1, v_2, v_3, v_4, v_5\}$ and $V'_B = \{v_6, v_7\}$. Moreover v_8 will necessarily join V'_B and v_9 will join V'_A, so there is a unique stable assignment.

Here is the Isabelle version of Prop. 1:

```
lemma argmaxUniqueStable :
  assumes "stable f"
  and "∀x. (x=A) = (x≠B)"
  and "(∀v. v∉ sureFirstAlt B A ∪ sureFirstAlt A B ⟶
       ((B,card (sureFirstAlt B A) + 1) ⪰v
        (A,card (UNIV::'V set)-card (sureFirstAlt B A))) ∨
       ((A,card (sureFirstAlt A B) + 1) ⪰v
        (B,card (UNIV::'V set)-card (sureFirstAlt A B))))"
shows "f = (λv. if v ∈ (sureFirstAlt A B) ∨
                   (A,card (sureFirstAlt A B) + 1) ⪰v
                   (B,card (UNIV::'V set)-card (sureFirstAlt A B))
                then A else B)"
```

The Isabelle development consists of approximately 250 lines of code.

4.4. A Characterisation of Uniqueness

[2, Thm. 2] states that there is a unique stable assignment if and only if Algorithm 1 returns the same assignment as its symmetric version starting from $\langle \emptyset, V \rangle$. The proof of this theorem refers to "the properties of Algorithm 1" which we find too vague. In fact, we were very surprised when we discovered that we needed preferences to be total, but for all previous results we do not! And even here, we are not sure that the result really needs total preferences.

To prove Thm. 2, the following lemma is useful. It says that there cannot be two distinct stable assignments such that one is obtained from the other by switching some voters.

Lemma 3. *Let $V = V_A \uplus V_B \uplus C \uplus D$ where $C \neq \emptyset$ and $D \neq \emptyset$. Let $f_1 = \langle V_A \uplus C, V_B \uplus D \rangle$ and $f_2 = \langle V_A \uplus D, V_B \uplus C \rangle$. Assume all preferences are total.*
Then f_1 and f_2 cannot both be stable assignments.

Proof. For the purpose of deriving a contradiction, let us assume that f_1 and f_2 are both stable assignments.

Since f_1 is stable, and preferences are total, there is at least one voter $v_a \in C$ such that $(A, |V_A \uplus C|) \succ_a (B, |V_B \uplus D \uplus C|)$ (otherwise the group C would move to B). By monotonicity $(A, |V_A \uplus C|) \succ_a (B, |V_B \uplus C|)$. If $|D| + 1 \geq |C|$, then $(A, |V_A \uplus D| + 1) \succ_a (A, |V_A \uplus C|)$ and thus $(A, |V_A \uplus D| + 1) \succ_a (B, |V_B \uplus C|)$, so v_a would want to deviate from f_2, contradicting the stability of f_2. Therefore $|D| + 1 < |C|$.

In the previous paragraph, by switching the roles of C and D, and A and B, we can show that $|C| + 1 < |D|$.

This is a contradiction, so our assumption that f_1 and f_2 are both stable is false. □

Its Isabelle version looks as follows:

```
lemma noSwapStable :
  assumes "∀x. (x=A) = (x≠B)"
  and "(∀v x y. x≠y ⟶(x ≿v y) ∨ y ≿v x)"
  and "VA ∩ C = {} ∧ VA ∩ D = {} ∧ D ∩ C = {}"
  and "C≠{}" and "D≠{}"
shows "¬ (stable (λu. if u∈VA∪C then A else B) ∧
          stable (λu. if u∈VA∪D then A else B))"
```

So if a profile admits (at least) two distinct stable assignments, then one must be obtained from the other by shuffling some voters from A to B or vice versa, but not both ways.

Theorem 2. *Assume that all preferences are total.*
Algorithm 1 and the reverse Algorithm 1 return the same assignment if and only if the profile admits a unique stable assignment.

Proof. One direction is trivial and already clearly stated in [2].

Now assume that Algorithm 1 as well as the *reverse* Algorithm 1 both return the assignment f_1, and for the purpose of deriving a contradiction, that another stable assignment f_2 exists. Without loss of generality, using Lemma 3, assume that f_1 has the form $\langle V_A, V_B \uplus C\rangle$ and f_2 has the form $\langle V_A \uplus C, V_B\rangle$. As Algorithm 1 starts from the assignment $\langle V, \emptyset\rangle$ and terminates in $\langle V_A, V_B \uplus C\rangle$, at some iteration it must have "skipped over" $\langle V_A \uplus C, V_B\rangle$, i.e., it must be possible to write $C = C' \uplus C''$ with $C'' \neq \emptyset$, $V_B = V_B' \uplus V_B'' \uplus V_B'''$ with $V_B' \neq \emptyset$, such that some iteration of Algorithm 1 goes from $\langle V_A \uplus C' \uplus C'' \uplus V_B' \uplus V_B'', V_B'''\rangle$ to $\langle V_A \uplus C' \uplus V_B', V_B''' \uplus C'' \uplus V_B''\rangle$. This means that for each voter $v_i \in C''$, we have

$$(B, |V_B''' \uplus C'' \uplus V_B''|) \succ_i (A, |V_A \uplus C' \uplus C'' \uplus V_B' \uplus V_B''|).$$

By monotonicity,

$$(B, |V_B' \uplus V_B''' \uplus C'' \uplus V_B''|) \succ_i (A, |V_A \uplus C' \uplus C''|),$$

which means that the voters in C'' have a reason to deviate from

$$f_2 = \langle V_A \uplus C' \uplus C'', V_B' \uplus V_B'' \uplus V_B'''\rangle$$

This contradicts the assumption that f_2 is a stable assignment. □

The proof of Lemma 3 is the only place where we use the assumption of total preferences; Thm. 2 inherits this assumption. We strongly suspect that the assumption of totality could be replaced by an assumption along the lines "only assignments arising during the run of Algorithm 1 are considered", and so totality would not really be needed. But given that Algorithm 1 is run in two directions and that we rely on contradiction proofs, a-priori excluding some assignments from the reasoning seems complicated.

The Isabelle version looks as follows. The development has around 550 lines of code.

```
lemma twoDirectionsImplyUniqueness :
  assumes "∀x. (x=A) = (x≠B)"
  and "(∀v x y. x≠y ⟶(x ≿v y) ∨ y ≿v x)"
  and "algo (Suc (card (UNIV::'V set))) A B =
       algo (Suc (card (UNIV::'V set))) B A"
  and "stable f2"
shows "algo (Suc (card (UNIV::'V set))) A B = f2"
```

Theorem 2 is not obvious and while trying to prove it we had doubts whether it actually holds. The Isabelle code for this result corresponds to 25% of our entire Isabelle development.

5. Discussion and Future Work

Stable assignments do not always exist for more than two alternatives:

Example 4. *Olivier Létoffé, a student doing an internship at at Toulouse University in 2022, has found this example:*

$$\begin{aligned}
v_1: \quad & (A,3) \succ_1 (A,2) \succ_1 (B,3) \succ_1 (B,2) \succ_1 (C,3) \succ_1 (C,2) \\
& \succ_1 (A,1) \succ_1 (B,1) \succ_1 (C,1) \\
v_2: \quad & (B,3) \succ_2 (B,2) \succ_2 (C,3) \succ_2 (C,2) \succ_2 (A,3) \succ_2 (A,2) \\
& \succ_2 (B,1) \succ_2 (C,1) \succ_2 (A,1) \\
v_3: \quad & (C,3) \succ_3 (C,2) \succ_3 (A,3) \succ_3 (A,2) \succ_3 (B,3) \succ_3 (B,2) \\
& \succ_3 (C,1) \succ_3 (A,1) \succ_3 (B,1)
\end{aligned}$$

Starting in $(A,3)$, we turn in circles. No stable assignment exists. Independently, [2] also contains such an example while the earlier unpublished version [11] (from which we started our work) does not discuss this point at all.

Generalising the work to more than two alternatives is the obvious direction for future work: under which conditions does a stable assignment exist, and which algorithm can find it?

In [2] complexity issues are also discussed, such as how hard it is to access a preference relation. This is also an issue for future work.

Another interesting issue for future work is strategy-proofness, i.e., immunity to false reporting of preferences.

During the above-mentioned internship, some implementation work was done using Coq rather than Isabelle/HOL. We found this implementation less natural, i.e., less close to the source [2], than the one presented here; for example, the Coq implementation did not have the notion of preferences between communities. Instead, the communities were mapped to natural numbers that were then compared by the usual $>$ relation. We are unable to judge whether a better design would have been possible in Coq. In any case we found no such discrepancies for Isabelle.

Relatively little work has been done on applications of proof assistants to game theory. To the best of our knowledge, no work on formalising the framework of [2] has been published. We believe that our work is a contribution to making the formalism proposed by [2] more precise and the results more reliable, apart from the improvements of the theory shown in Section 4.

Acknowledgments

I would like to thank Umberto Grandi for pointing me to the article formalised in this work. I would like to thank him as well as Ben Abramowitz, Davide Grossi and Nimrod Talmon for fruitful discussions.

References

[1] J.-G. Smaus, Proving the Existence of Stable Assignments in Democratic Forking Using Isabelle/HOL (extended version), Technical Report 2024–03–FR, Institut de Recherche en Informatique de Toulouse, 2024.

[2] B. Abramowitz, E. Elkind, D. Grossi, E. Shapiro, N. Talmon, Democratic forking: Choosing sides with social choice, in: D. Fotakis, D. R. Insua (Eds.), Algorithmic Decision Theory - 7th International Conference, ADT 2021, Toulouse, France, November 3-5, 2021, Proceedings, volume 13023 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 341–356. doi:10.1007/978-3-030-87756-9_22.

[3] A. Phillip, J. S. Chan, S. Peiris, A new look at Cryptocurrencies, Economics Letters 163 (2018) 6–9. URL: https://ideas.repec.org/a/eee/ecolet/v163y2018icp6-9.html. doi:10.1016/j.econlet.2017.11.

[4] S. Zhou, B. Vasilescu, C. Kästner, How has forking changed in the last 20 years?: a study of hard forks on github, in: G. Rothermel, D. Bae (Eds.), ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020, ACM, 2020, pp. 445–456. doi:10.1145/3377811.3380412.

[5] T. Nipkow, L. C. Paulson, M. Wenzel, Isabelle/HOL - A Proof Assistant for Higher-Order Logic, volume 2283 of *Lecture Notes in Computer Science*, Springer, 2002.

[6] M. Wenzel, The Isabelle/Isar Reference Manual, 2022. https://isabelle.in.tum.de/doc/isar-ref.pdf.

[7] R. Pollack, How to Believe a Machine-Checked Proof, Technical Report RS-97-18, Basic Research in Computer Science, University of Aarhus, 1997.

[8] H. Barendregt, S. Abramsky, D. M. Gabbay, T. S. E. Maibaum, Lambda Calculi with Types, Oxford University Press, 1992, pp. 117–309. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.26.4391.

[9] A. Krauss, Defining recursive functions in Isabelle/HOL, 2006. Tutorial available via https://isabelle.in.tum.de/doc/functions.pdf.

[10] A. Krauss, Partial recursive functions in higher-order logic, in: U. Furbach, N. Shankar (Eds.), Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings, volume 4130 of *Lecture Notes in Computer Science*, Springer, 2006, pp. 589–603. doi:10.1007/11814771_48.

[11] B. Abramowitz, E. Elkind, D. Grossi, E. Shapiro, N. Talmon, Democratic forking: Choosing sides with social choice, CoRR abs/2103.03652 (2021). URL: https://arxiv.org/abs/2103.03652. arXiv:2103.03652.