Arrow’s General Possibility Theorem

Peter Gammie
peteg42 at gmail.com
August 16, 2018

Contents

1 Overview 2

2 General Lemmas 2

2.1 Extra Finite-Set Lemmas . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2

2.2 Extra bijection lemmas . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

2.3 Collections of witnesses: hasw, has . . . . . . . . . . . . . . . . . . . . . . . . 5

3 Preliminaries 8

3.1 Rational Preference Relations (RPRs) . . . . . . . . . . . . . . . . . . . . . . 9

3.2 Profiles . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

3.3 Choice Sets, Choice Functions . . . . . . . . . . . . . . . . . . . . . . . . . . 11

3.4 Social Choice Functions (SCFs) . . . . . . . . . . . . . . . . . . . . . . . . . . 13

3.5 Social Welfare Functions (SWFs) . . . . . . . . . . . . . . . . . . . . . . . . . 13

3.6 General Properties of an SCF . . . . . . . . . . . . . . . . . . . . . . . . . . . 14

3.7 Decisiveness and Semi-decisiveness . . . . . . . . . . . . . . . . . . . . . . . 15

4 Arrow’s General Possibility Theorem 16

4.1 Semi-decisiveness Implies Decisiveness . . . . . . . . . . . . . . . . . . . . . 16

4.2 The Existence of a Semi-decisive Individual . . . . . . . . . . . . . . . . . . . 23

4.3 Arrow’s General Possibility Theorem . . . . . . . . . . . . . . . . . . . . . . 27

5 Sen’s Liberal Paradox 27

5.1 Social Decision Functions (SDFs) . . . . . . . . . . . . . . . . . . . . . . . . . 27

5.2 Sen’s Liberal Paradox . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30

6 May’s Theorem 35

6.1 May’s Conditions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35

6.2 The Method of Majority Decision satisfies May’s conditions . . . . . . . . . . 37

6.3 Everything satisfying May’s conditions is the Method of Majority Decision . . 39

6.4 The Plurality Rule . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 45

7 Bibliography 46
1 Overview

This is a fairly literal encoding of some of Armatya Sen’s proofs [Sen70] in Isabelle/HOL. The author initially wrote it while learning to use the proof assistant, and some locutions remain naïve. This work is somewhat complementary to the mechanisation of more recent proofs of Arrow’s Theorem and the Gibbard-Satterthwaite Theorem by Tobias Nipkow [Nip08].

I strongly recommend Sen’s book to anyone interested in social choice theory; his proofs are quite lucid and accessible, and he situates the theory quite well within the broader economic tradition.

2 General Lemmas

2.1 Extra Finite-Set Lemmas

Small variant of Finite-Set :finite-subset-induct: also assume \( F \subseteq A \) in the induction hypothesis.

**lemma** :finite-subset-induct’ [consumes 2, case-names empty insert]:

**assumes** finite \( F \) and \( F \subseteq A \)

and empty: \( P \{\} \)

and insert: \( \forall a. \left( \text{finite } F; a \in A; F \subseteq A; a \notin F; P F \right) \implies P (\text{insert } a F) \)

**shows** \( P F \)

**proof** –

**from** \( \text{(finite } F) \)

**have** \( F \subseteq A \implies ?thesis \)

**proof** induction

**show** \( P \{\} \) by fact

next

**fix** \( x \) \( F \)

**assume** finite \( F \) and \( x \notin F \) and

\( P; F \subseteq A \implies P F \) and \( i: \text{insert } x F \subseteq A \)

**show** \( P (\text{insert } x F) \)

**proof** (rule insert)

**from** \( i \) **show** \( x \in A \) by blast

**from** \( i \) **have** \( F \subseteq A \) by blast

**with** \( P \) **show** \( P F . \)

**show** finite \( F \) by fact

**show** \( x \notin F \) by fact

**show** \( F \subseteq A \) by fact

**qed**

**with** \( F \subseteq A \) **show** ?thesis by blast

**qed**

A slight improvement on List :finite-list - add distinct.

**lemma** :finite-list: finite \( A \implies \exists l. \text{set } l = A \land \text{distinct } l \)

**proof** (induct rule: finite-induct)

**case** (insert \( x F \))

then obtain \( l \) where \( \text{set } l = F \land \text{distinct } l \) by auto

with insert **have** \( \text{set } (x\#l) = \text{insert } x F \land \text{distinct } (x\#l) \) by auto
thus case by blast
qed auto

2.2 Extra bijection lemmas

lemma bij-betw-onto: bij-betw f A B \implies f : A = B unfolding bij-betw-def by simp

lemma inj-on-UnI: \[ \begin{align*}
\text{inj-on } f \ A; \ \text{inj-on } f \ B; \ f \ (A - B) \cap f \ (B - A) = \{\} \end{align*} \] \implies \text{inj-on } f \ (A \cup B)
by (auto iff: inj-on-Un)

lemma card-compose-bij:
assumes bijf: bij-betw f A A
shows \( \card \{ a \in A. \ P (f a) \} = \card \{ a \in A. \ P a \} \)
proof
  from bijf have T: \( f \ (\{ a \in A. \ P (f a) \}) = \{ a \in A. \ P a \} \)
  unfolding bij-betw-def by auto
  from bijf have \( \card \{ a \in A. \ P (f a) \} = \card (f \ (\{ a \in A. \ P (f a) \})) \)
  unfolding bij-betw-def by (auto intro: subset-inj-on card-image [symmetric])
  with T show thesis by simp
qed

lemma card-eq-bij:
assumes cardAB: \( \card A = \card B \) and finiteA: finite A and finiteB: finite B
obtains f where bij-betw f A B
proof
  let \( f = \lambda x. \ \text{if } x \in A \ \text{then } f x \ \text{else } g x \)
  have inj-on \( f : A \cup (B - A) \)
  by (rule inj-on-UnI)
  from bij show inj-on \( f : A \)
  by (rule inj-on-UnI, auto dest: inj-onD bij-betw-imp-inj-on)

  thus \( \\) thesis ..
qed

lemma bij-combine:
assumes ABCD: \( A \subseteq B \ C \subseteq D \) and bijf: bij-betw f A C and bijg: bij-betw g (B - A) (D - C)
obtains h
where bij-betw h B D
and \( \forall x. \ x \in A \implies h x = f x \)
and \( \forall x. \ x \in B - A \implies h x = g x \)
proof
  let \( h = \lambda x. \ \text{if } x \in A \ \text{then } f x \ \text{else } g x \)
  have inj-on \( h : (A \cup (B - A)) \)
  proof (rule inj-on-UnI)
    from bij show inj-on \( h : A \)
    by (rule inj-on-UnI, auto dest: inj-onD bij-betw-imp-inj-on)
  qed
from bijg show inj-on ?h (B - A)
  by (rule inj-onI, auto dest: inj-onD bij-betw-imp-inj-on)
from bij big show ?h '(A - (B - A)) \cap ?h ' (B - A - A) = {}
  by (simp, blast dest: bij-betw-onto)
qed

with ABCD have inj-on ?h B by (auto iff: Un-absorb1)
moreover
have ?h ' = D
proof
  from ABCD have ?h ' = f ' A \cup g ' (B - A)
    by (auto iff: image-Un Un-absorb1)
also from ABCD bijf bijg have \ldots = D by (blast dest: bij-betw-onto)
finally show ?thesis.
qed

ultimately have bij-betw ?h B D
  and \forall x. x \in A \implies ?h x = f x
  and \forall x. x \in B - A \implies ?h x = g x
unfolding bij-betw-def by auto
thus thesis ..
qed

lemma bij-complete:
assumes finiteC: finite C
  and ABC: A \subseteq C B \subseteq C
  and bijf: bij-betw f A B
obtains f' where bij-betw f' C C
  and \forall x. x \in A \implies f' x = f x
  and \forall x. x \in C - A \implies f' x \in C - B
proof
  from finiteC ABC bijf have card B = card A
    unfolding bij-betw-def
    by (auto iff: inj-on-iff-eq-card [symmetric] intro: finite-subset)
with finiteC ABC bijf have card (C - A) = card (C - B)
    by (auto iff: finite-subset card-Diff-subset)
with finiteC obtain g where bijg: bij-betw g (C - A) (C - B)
    by - (drule card-eq-bij, auto)
from ABC bijf bijg
obtain f' where bijf': bij-betw f' C C
  and f'f: \forall x. x \in A \implies f' f x = f x
  and f'g: \forall x. x \in C - A \implies f' x = g x
    by - (drule bij-combine, auto)
from f'g bijg have \forall x. x \in C - A \implies f' x \in C - B
    by (blast dest: bij-betw-onto)
with bijf' f'f show thesis ..
qed

lemma card-greater:
assumes finiteA: finite A
  and c: card { x \in A. P x } > card { x \in A. Q x }
obtains C
  where card (\{ x \in A. P x \} - C) = card { x \in A. Q x }
    and C \neq {}
    and C \subseteq \{ x \in A. P x \}
proof
let ?PA = \{ x \in A . P x \}
let ?QA = \{ x \in A . Q x \}

from finiteA obtain p where P: bij-betw p \{ 0..<\text{card } ?PA \} ?PA
  using ex-bij-betw-nat-finite[where M=?PA]
  by (blast intro: finite-subset)

let ?CN = \{ card ?QA..<\text{card } ?PA \}
let ?C = p ' ?CN

have card (\{ x \in A. P x \} - ?C) = card ?QA
proof
  have nat-add-sub-shuffle: \( \forall x y z. \[ (x::nat) > y; x - y = z \] \implies x - z = y \)
    by simp
  from P have T: p ' \{ card ?QA..<\text{card } ?PA \} \subseteq ?PA
    unfolding bij-betw-def by auto
  from P have card ?PA - card ?QA = card ?C
    unfolding bij-betw-def
    by (auto iff: card-image subset-inj-on[where A=?CN])
  with c have card ?PA - card ?C = card ?QA by (rule nat-add-sub-shuffle)
  with finiteA P T have card (?PA - ?C) = card ?QA
    unfolding bij-betw-def
    by (auto iff: finite-subset card-Diff-subset)
  thus ?thesis .
qed

moreover
  from P c have ?C \neq \{\}
    unfolding bij-betw-def by auto
moreover
  from P have ?C \subseteq \{ x \in A. P x \}
    unfolding bij-betw-def by auto
ultimately show thesis ..
qed

2.3 Collections of witnesses: hasw, has

Given a set of cardinality at least \( n \), we can find up to \( n \) distinct witnesses. The built-in \text{card} function unfortunately satisfies:

\[
\text{Finite-Set.card-infinite}: \text{infinite } A \implies \text{card } A = 0
\]

These lemmas handle the infinite case uniformly.

Thanks to Gerwin Klein suggesting this approach.

definition hasw :: 'a list \Rightarrow 'a set \Rightarrow bool where
  hasw xs S \equiv set xs \subseteq S \land distinct xs

definition has :: nat \Rightarrow 'a set \Rightarrow bool where
  has n S \equiv \exists xs. hasw xs S \land length xs = n

declare hasw-def[simp]

lemma hasI[intro]: hasw xs S \implies has (length xs) S by (unfold has-def, auto)

lemma card-has:
  assumes cardS: \text{card } S = n
  shows has n S
proof(cases n = 0)
case True thus ?thesis by (simp add: has-def)
next
case False
with cardS card-eq-0-iff[where A=S] have finiteS: finite S by simp
show ?thesis
proof (rule ccontr)
  assume nhas: ¬ has n S
  with distinct-card[symmetric]
  have nxs: ¬ (∃ xs. set xs ⊆ S ∧ distinct xs ∧ card (set xs) = n)
    by (auto simp add: has-def)
  from finite-list finiteS
  obtain xs where S = set xs by blast
  with cardS nxs show False by auto
qed
qed

lemma card-has-rev:
  assumes finiteS: finite S
  shows has n S =⇒ card S ≥ n (is ?lhs =⇒ ?rhs)
proof
  assume ?lhs
  then obtain xs where set xs ⊆ S ∧ n = length xs
    and dxs: distinct xs by (unfold has-def hasw-def, blast)
  with card-mono[OF finiteS] distinct-card[OF dxs, symmetric]
  show ?rhs by simp
qed

lemma has-0: has 0 S by (simp add: has-def)

lemma has-suc-notempty: has (Suc n) S =⇒ {} ≠ S
  by (clarsimp simp add: has-def)

lemma has-suc-subset: has (Suc n) S =⇒ {} ⊂ S
  by (rule psubsetI, (simp add: has-suc-notempty)+)

lemma has-notempty-1:
  assumes Sne: S ≠ {}
  shows has 1 S
proof
  from Sne obtain x where x ∈ S by blast
  hence set [x] ⊆ S ∧ distinct [x] ∧ length [x] = 1 by auto
  thus ?thesis by (unfold has-def hasw-def, blast)
qed

lemma has-le-has:
  assumes h: has n S
    and nn': n' ≤ n
  shows has n' S
proof
  from h obtain xs where hasw xs S length xs = n by (unfold has-def, blast)
  with nn' set-take-subset[where n=n' and xs=xs]
  have hasw (take n' xs) S length (take n' xs) = n'
  thus ?thesis by (simp add: hasw_def hasw_w_def, blast)
qed
by (simp-all add: min-def, blast+)
thus thesis by (unfold has-def, blast)
qed

lemma has-ge-has-not:
assumes h: ¬has n S
and nn': n ≤ n'
shows ¬has n' S
using h nn' by (blast dest: has-le-has)

lemma has-eq:
assumes h: has n S
and hn': ¬has (Suc n) S
shows card S = n
proof
from h obtain xs
  where xs: hasw xs S and lenxs: length xs = n by (unfold has-def, blast)
have set xs = S
proof
  from xs show set xs ⊆ S by simp
next
  show S ⊆ set xs
proof(rule ccontr)
    assume ¬ S ⊆ set xs
    then obtain x where x ∈ S x ∉ set xs by blast
    with lenxs xs have hasw (x # xs) S length (x # xs) = Suc n by simp-all
    with hn' show False by (unfold has-def, blast)
  qed
  qed
  with xs lenxs distinct-card show card S = n by auto
qed

lemma has-extend-witness:
assumes h: has n S
shows [ set xs ⊆ S; length xs < n ] ⇒ set xs ⊆ S
proof(induct xs)
case Nil
  with h has-suc-notempty show ?case by (cases n, auto)
next
case (Cons x xs)
  have set (x ≠ xs) ≠ S
  proof
    assume S(xs): set (x ≠ xs) = S
    hence finiteS: finite S by auto
    from h obtain xs'
      where S(xs)': set xs' ⊆ S
      and dlxs': distinct xs' ∧ length xs' = n
      by (unfold has-def hasw-def, blast)
      with distinct-card have card (set xs') = n by auto
      with finiteS S(xs)' card-mono have card S ≥ n by auto
      moreover
      from S(xs) Cons card-length[where xs=x ≠ xs]
      have card S < n by auto
  qed

7
ultimately show False by simp
qed
with Cons show ?case by auto
qed

lemma has-extend-witness':
[ has n S; hasw xs S; length xs < n ] \implies \exists x. hasw (x \# xs) S
by (simp, blast dest: has-extend-witness)

lemma has-witness-two:
assumes hasnS: has n S
and nn': 2 \leq n
shows \exists x y. hasw [x,y] S
proof -
have has2S: has 2 S by (rule has-le-has[OF hasnS nn'])
from has-extend-witness'[OF has2S, where xs=[]]
obtain x where x \in S by auto
with has-extend-witness'[OF has2S, where xs=[x]]
show ?thesis by auto
qed

lemma has-witness-three:
assumes hasnS: has n S
and nn': 3 \leq n
shows \exists x y z. hasw [x,y,z] S
proof -
from nn' obtain x y where hasw [x,y] S
using has-witness-two[OF hasnS] by auto
with nn' show ?thesis
using has-extend-witness'[OF hasnS, where xs=[x,y]] by auto
qed

lemma finite-set-singleton-contra:
assumes finiteS: finite S
and Sne: S \neq {} 
and cardS: card S > 1 \implies False
shows \exists j. S = \{ j \}
proof -
from cardS Sne card-0-eq[OF finiteS] have Scard: card S = 1 by auto
from has-extend-witness[where xs=[], OF card-has[OF this]]
obtain j where \{ j \} \subseteq S by auto
from card-seteq[OF finiteS this] Scard show ?thesis by auto
qed

3 Preliminaries

The auxiliary concepts defined here are standard [Rou79, Sen70, Tay05]. Throughout we make use of a fixed set A of alternatives, drawn from some arbitrary type 'a of suitable size. Taylor [Tay05] terms this set an agenda. Similarly we have a type 'i of individuals and a
3.1 Rational Preference Relations (RPRs)

Definitions for rational preference relations (RPRs), which represent indifference or strict preference amongst some set of alternatives. These are also called weak orders or (ambiguously) ballots.

Unfortunately Isabelle’s standard ordering operators and lemmas are typeclass-based, and as introducing new types is painful and we need several orders per type, we need to repeat some things.

type-synonym `a RPR = `a set

abbreviation rpr-eq-syntax :: `a ⇒ `a RPR ⇒ `a ⇒ bool (- _ ≤ - [50, 1000, 51] 50) where
  x r≤ y ≡ (x, y) ∈ r

definition indifferent-pref :: `a ⇒ `a RPR ⇒ `a ⇒ bool (- _ ≈ - [50, 1000, 51] 50) where
  x r≈ y ≡ (x r≤ y ∧ y r≤ x)

lemma indifferent-prefI[intro]: [ x r≤ y; y r≤ x ] ⇒ x r≈ y
  unfolding indifferent-pref-def by simp

lemma indifferent-prefD[dest]: x r≈ y ⇒ x r≤ y ∧ y r≤ x
  unfolding indifferent-pref-def by simp

definition strict-pref :: `a ⇒ `a RPR ⇒ `a ⇒ bool (- _ < - [50, 1000, 51] 50) where
  x r≺ y ≡ (x r≤ y ∧ ¬(y r≤ x))

lemma strict-pref-def-irrefl[simp]: ¬ (x r≺ x) unfolding strict-pref-def by blast

lemma strict-prefI[intro]: [ x r≤ y; ¬(y r≤ x) ] ⇒ x r≺ y
  unfolding strict-pref-def by simp

Traditionally, x r≤ y would be written x R y, x r≈ y as x I y and x r≺ y as x P y, where the relation r is implicit, and profiles are indexed by subscripting.

*Complete* means that every pair of distinct alternatives is ranked. The ”distinct” part is a matter of taste, as it makes sense to regard an alternative as as good as itself. Here I take reflexivity separately.

definition complete :: `a set ⇒ `a RPR ⇒ bool where
  complete A r ≡ (∀ x ∈ A. ∀ y ∈ A - {x}. x r≤ y ∨ y r≤ x)

lemma completeI[intro]:
  (∀ x. [ x ∈ A; y ∈ A; x ≠ y ] ⇒ x r≤ y ∨ y r≤ x) ⇒ complete A r
  unfolding complete-def by auto

lemma completeD[dest]:
  [ complete A r; x ∈ A; y ∈ A; x ≠ y ] ⇒ x r≤ y ∨ y r≤ x
  unfolding complete-def by auto

lemma complete-less-not: [ complete A r; hasw [x,y] A; ¬ x r≺ y ] ⇒ y r≤ x
  unfolding complete-def strict-pref-def by auto
lemma complete-indiff-not: [ complete A r; hasw [x,y] A; ¬ x r≈ y ] ⇒ x r≺ y ∨ y r≺ x
unfolding complete-def indifferent-pref-def strict-pref-def by auto

lemma complete-exh:
  assumes complete A r and hasw [x,y] A
  obtains (xPy) y r≺ x
  | (yPx) y r≺ x
  | (x[y] y r≡ y)
using assms unfolding complete-def strict-pref-def indifferent-pref-def by auto

Use the standard refl. Also define irreflexivity analogously to how refl is defined in the standard library.

declare refl-onI[intro] refl-onD[dest]

lemma complete-refl-on:
  [ complete A r; refl-on A r; x ∈ A; y ∈ A ] ⇒ x r≤ y ∨ y r≤ x
unfolding complete-def by auto

Rational preference relations, also known as weak orders and (I guess) complete pre-orders.

definition irrefl :: 'a set ⇒ 'a RPR ⇒ bool where
  irrefl A r ≡ r ⊆ A × A ∧ (∀ x ∈ A. ¬ x r≤ x)

lemma irreflI[intro]: [ r ⊆ A × A; ∃x. x ∈ A ⇒ ¬ x r≤ x ] ⇒ irrefl A r
unfolding irrefl-def by simp

lemma irreflD[dest]: [ irrefl A r; (x, y) ∈ r ] ⇒ hasw [x,y] A
unfolding irrefl-def by auto

lemma irreflD[dest]:
  [ irrefl A r; r ≠ {} ] ⇒ ∃x y. hasw [x,y] A ∧ (x, y) ∈ r
unfolding irrefl-def by auto

lemma rpr-refl[Intro]: [ complete A r; refl-on A r; trans r ] ⇒ rpr A r
unfolding rpr-def by simp

lemma rprD: rpr A r ⇒ complete A r ∧ refl-on A r ∧ trans r
unfolding rpr-def by simp

lemma rpr-in-set[dest]: [ rpr A r; x r≤ y ] ⇒ {x,y} ⊆ A
unfolding rpr-def refl-on-def by auto

lemma rpr-refl[dest]: [ rpr A r; x ∈ A ] ⇒ x r≤ x
unfolding rpr-def by blast

lemma rpr-less-not: [ rpr A r; hasw [x,y] A; ¬ x r≺ y ] ⇒ y r≤ x
unfolding rpr-def by (auto simp add: complete-less-not)

lemma rpr-less-imp-le[simp]: [ x r≺ y ] ⇒ x r≤ y
unfolding strict-pref-def by simp

lemma rpr-less-imp-neq[simp]: \[ x \prec y \] \implies x \neq y
unfolding strict-pref-def by blast

lemma rpr-less-trans[trans]: \[ x \prec y; \ y \prec z; \ rpr A r \] \implies x \prec z
unfolding rpr-def strict-pref-def trans-def by blast

lemma rpr-le-trans[trans]: \[ x \preceq y;\ y \preceq z;\ rpr A r \] \implies x \preceq z
unfolding rpr-def trans-def by blast

lemma rpr-le-less-trans[trans]: \[ x \preceq y;\ y \prec z;\ rpr A r \] \implies x \prec z
unfolding rpr-def strict-pref-def trans-def by blast

lemma rpr-less-le-trans[trans]: \[ x \prec y;\ y \preceq z;\ rpr A r \] \implies x \prec z
unfolding rpr-def strict-pref-def trans-def by blast

lemma rpr-complete: \[ rpr A r;\ x \in A;\ y \in A \] \implies x \preceq y \lor y \preceq x
unfolding rpr-def by (blast dest: complete-refl-on)

3.2 Profiles

A profile (also termed a collection of ballots) maps each individual to an RPR for that individual.

type-synonym \('a, 'i) Profile = 'i \Rightarrow 'a RPR

definition profile :: \('a set \Rightarrow 'i set \Rightarrow ('a, 'i) Profile \Rightarrow bool where
profile A Is P \equiv Is \neq \{} \land (\forall i \in Is. \ rpr A (P i))

lemma profileI[intro]: \[ \forall i. i \in Is \implies rpr A (P i); \ Is \neq \{} \] \implies profile A Is P
unfolding profile-def by simp

lemma profile-rprD[dest]: \[ profile A Is P; \ i \in Is \] \implies rpr A (P i)
unfolding profile-def by simp

lemma profile-non-empty: profile A Is P \implies Is \neq \{}
unfolding profile-def by simp

3.3 Choice Sets, Choice Functions

A choice set is the subset of A where every element of that subset is (weakly) preferred to every other element of A with respect to a given RPR. A choice function yields a non-empty choice set whenever A is non-empty.

definition choiceSet :: \('a set \Rightarrow 'a RPR \Rightarrow 'a set where
choiceSet A r \equiv \{ x \in A. \forall y \in A. x \preceq y \}

definition choiceFn :: \('a set \Rightarrow 'a RPR \Rightarrow bool where
choiceFn A r \equiv \forall A' \subseteq A. \ A' \neq \{} \longrightarrow choiceSet A' r \neq \{}
lemma choiceSetI[intro]:
\[ x \in A; \forall y. y \in A \Rightarrow x \preceq y \] \Rightarrow x \in \text{choiceSet } A r
unfolding choiceSet-def by simp

lemma choiceFnI[intro]:
\[ (\forall A'. A' \subseteq A; A' \neq \emptyset) \Rightarrow \text{choiceSet } A' r \neq \emptyset \] \Rightarrow \text{choiceFn } A r
unfolding choiceFn-def by simp

If a complete and reflexive relation is also quasi-transitive it will yield a choice function.

definition quasi-trans :: 'a RPR \Rightarrow bool where
quasi-trans r \equiv \forall x y z. x \preceq y \land y \preceq z \rightarrow x \preceq z

lemma quasi-transI[intro]:
\[ (\forall x y z. x \preceq y; y \preceq z \Rightarrow x \preceq z) \Rightarrow \text{quasi-trans } r \]
unfolding quasi-trans-def by blast

lemma quasi-transD: \[ x \preceq y; y \preceq z; \text{quasi-trans } r \] \Rightarrow x \preceq z
unfolding quasi-trans-def by blast

lemma trans-imp-quasi-trans: \[ \text{trans } r \Rightarrow \text{quasi-trans } r \]
by (rule quasi-transI, unfold strict-pref-def trans-def, blast)

lemma r-c-qt-imp-cf:
assumes finiteA: finite A
and c: complete A r
and qt: quasi-trans r
and r: refl-on A r
shows \text{choiceFn } A r
proof
fix B assume B: B \subseteq A B \neq \emptyset
with finite-subset finiteA have finiteB: finite B by auto
from finiteB B show \text{choiceSet } B r \neq \emptyset by auto
proof(induct rule: finite-subset-induct')
case empty with B show ?case by auto
next
case (insert a B)
hence finiteB: finite B
and aA: a \in A
and and AB: B \subseteq A
and aB: a \notin B
and and cF: B \neq \emptyset \Rightarrow \text{choiceSet } B r \neq \emptyset by blast
show ?case
proof(cases B = \{}
case True with aA r show ?thesis
  unfolding choiceSet-def by blast
next
case False with cF obtain b where bCF: b \in \text{choiceSet } B r by blast
from AB aA bCF complete-refl-on[OF c r]
have a \preceq b \lor b \preceq a unfolding choiceSet-def strict-pref-def by blast
thus ?thesis
proof
  assume ab: b \preceq a

12
with \( b \text{CF} \) show \(?\text{thesis}\) unfolding choiceSet-def by auto

next

assume \( ab : a \prec b \)

have \( a \in \text{choiceSet} \) (insert \( a \) \( B \)) \( r \)

proof (rule ccontr)

assume \( a \text{CF} : a \notin \text{choiceSet} \) (insert \( a \) \( B \)) \( r \)

from \( ab \) have \( \bigwedge b. \ b \in B \implies a \neq b \) by auto

with \( a \text{CF} \) \( a \text{A} \) \( \text{AB} \) \( r \)

from \( a \text{B} \) have \( \forall b. \ b \in B \implies b \neq a \) by auto

unfolding choiceSet-def complete-def strict-pref-def by blast

with \( b \text{CF} \) \( B \)

show \( \text{False} \) unfolding choiceSet-def strict-pref-def by blast

qed

thus \(?\text{thesis}\) by auto

qed

qed

qed

lemma \( \text{rpr-choiceFn}: \{ \{ \text{finite} \ A; \ \text{rpr} \ A \ r \ \} \implies \text{choiceFn} \ A \ r \} \)

unfolding \( \text{rpr-def} \) by (blast dest: trans-imp-quasi-trans r-c-qt-imp-cf)

3.4 Social Choice Functions (SCFs)

A social choice function (SCF), also called a collective choice rule by Sen [Sen70, p28], is a function that somehow aggregates society’s opinions, expressed as a profile, into a preference relation.

The least we require of an SCF is that it be complete and some function of the profile. The latter condition is usually implied by other conditions, such as iia.

definition \( \text{SCF} :: (\text{'}a, \text{'}i) \text{SCF} \Rightarrow \text{'}a \text{ set} \Rightarrow \text{'}i \text{ set} \Rightarrow (\text{'}a \text{ set} \Rightarrow \text{'}i \text{ set} \Rightarrow (\text{'}a, \text{'}i) \text{ Profile} \Rightarrow \text{bool}) \Rightarrow \text{bool} \)

where

\( \text{SCF scf} \ A \ Is \ Pcond \equiv (\forall P. \ Pcond \ A \ Is \ P \implies \text{complete} \ A \ (\text{scf} \ P)) \)

lemma \( \text{SCFI}[\text{intro}]: \)

assumes \( c : \ \bigwedge P. \ Pcond \ A \ Is \ P \implies \text{complete} \ A \ (\text{scf} \ P) \)

shows \( \text{SCF scf} \ A \ Is \ Pcond \)

unfolding \( \text{SCF-def} \) using \( \text{assms} \) by blast

lemma \( \text{SCF-completeD}[\text{dest}]: [ \text{SCF scf} \ A \ Is \ Pcond; \ Pcond \ A \ Is \ P ] \implies \text{complete} \ A \ (\text{scf} \ P) \)

unfolding \( \text{SCF-def} \) by blast

3.5 Social Welfare Functions (SWFs)

A social welfare function (SWF) is an SCF that expresses the society’s opinion as a single RPR.

In some situations it might make sense to restrict the allowable profiles.

definition \( \text{SWF} :: (\text{'}a, \text{'}i) \text{SCF} \Rightarrow \text{'}a \text{ set} \Rightarrow \text{'}i \text{ set} \Rightarrow (\text{'}a \text{ set} \Rightarrow \text{'}i \text{ set} \Rightarrow (\text{'}a, \text{'}i) \text{ Profile} \Rightarrow \text{bool}) \Rightarrow \text{bool} \)

where
3.6 General Properties of an SCF

An SCF has a \textit{universal domain} if it works for all profiles.

\textbf{definition} \textit{universal-domain} :: \textit{'a set} \Rightarrow \textit{'i set} \Rightarrow (\textit{'a, 'i}) \textit{Profile} \Rightarrow \text{bool} \textit{ where}

universal-domain \textit{A Is P} \equiv \textit{profile A Is P}

\textbf{declare} universal-domain-def [simp]

\begin{itemize}
  \item An SCF is \textit{weakly Pareto-optimal} if, whenever everyone strictly prefers \textit{x} to \textit{y}, the SCF does too.
  \item \textbf{definition} \textit{weak-pareto} :: (\textit{'a, 'i}) \textit{SCF} \Rightarrow \textit{'a set} \Rightarrow \textit{'i set} \Rightarrow (\textit{'a set} \Rightarrow \textit{'i set} \Rightarrow (\textit{'a, 'i}) \textit{Profile} \Rightarrow \text{bool}) \Rightarrow \text{bool}

  \textit{ weak-pareto scf A Is Pcond} \equiv
  \forall P x y. \textit{Pcond A Is P} \land x \in A \land y \in A \land (\forall i \in \textit{Is}. \textit{x} (P_i) \prec y) \quad \Rightarrow \quad \textit{x (scf P) \prec y} \\

  \textbf{lemma} weak-paretoD:
  \[
  \forall P x y. \ [\textit{scf P} \Rightarrow \textit{Pcond A Is P} : x \in A; y \in A; \ [\forall i \in \textit{Is}. \textit{x (P_i) \prec y}] \quad \Rightarrow \quad \textit{x (scf P) \prec y} \\
  \]

  \textbf{lemma} weak-paretoI[intro]:
  \[
  \forall P x y. \ [\textit{weak-pareto scf A Is Pcond} : x \in A; y \in A; \ [\forall i \in \textit{Is}. \textit{x (P_i) \prec y}] \quad \Rightarrow \quad \textit{x (scf P) \prec y} \\
  \]

  \textbf{lemma} weak-pareto-intro:
  \[
  \forall P x y. \ [\textit{weak-pareto scf A Is Pcond} : x \in A; y \in A; \ [\forall i \in \textit{Is}. \textit{x (P_i) \prec y}] \quad \Rightarrow \quad \textit{x (scf P) \prec y} \\
  \]

  \textbf{lemma} weak-pareto-def [simp]

  \begin{itemize}
  \item An SCF satisfies \textit{independence of irrelevant alternatives} if, for two preference profiles \textit{P} and \textit{P'} where for all individuals \textit{i}, alternatives \textit{x} and \textit{y} drawn from set \textit{S} have the same order in \textit{P i} and \textit{P' i}, then alternatives \textit{x} and \textit{y} have the same order in \textit{scf P} and \textit{scf P'}.
  \item \textbf{definition} \textit{iia} :: (\textit{'a, 'i}) \textit{SCF} \Rightarrow \textit{'a set} \Rightarrow \textit{'i set} \Rightarrow \text{bool} \textit{ where}

    \textit{iia scf S Is} \equiv
    \forall P P' x y. \textit{profile S Is P} \land \textit{profile S Is P'}
    \land (\forall i \in \textit{Is}. ((x (P_i) \preceq y) \quad \leftrightarrow \quad (x (P'_i) \preceq y)) \land ((y (P_i) \preceq x) \quad \leftrightarrow \quad (y (P'_i) \preceq x)))
    \quad \rightarrow \quad ((x (scf P) \preceq y) \quad \leftrightarrow \quad (x (scf P') \preceq y)))

  \textbf{lemma} iia-intro:
  \[
  \forall P P' x y. \ [\textit{iia scf S Is} : x \in S; y \in S; \ [\forall i \in \textit{Is}. ((x (P_i) \preceq y) \quad \leftrightarrow \quad (x (P'_i) \preceq y)) \land ((y (P_i) \preceq x) \quad \leftrightarrow \quad (y (P'_i) \preceq x))] \quad \rightarrow \quad ((x (scf P) \preceq y) \quad \leftrightarrow \quad (x (scf P') \preceq y))) \\
  \]

  \textbf{lemma} iiaE:

  \[
  \forall P P' x y. \ [\textit{iia scf S Is} : x \in S; y \in S; \ [\forall i \in \textit{Is}. ((x (P_i) \preceq y) \quad \leftrightarrow \quad (x (P'_i) \preceq y)) \land ((y (P_i) \preceq x) \quad \leftrightarrow \quad (y (P'_i) \preceq x))] \quad \rightarrow \quad ((x (scf P) \preceq y) \quad \leftrightarrow \quad (x (scf P') \preceq y))) \\
  \]

  \textbf{lemma} iiaE-def [simp]

\end{itemize}


\[ iia \; scf \; S \; Is; \]
\[
\{ x, y \} \subseteq S; \]
\[
a \in \{ x, y \}; \; b \in \{ x, y \}; \]
\[
\bigwedge i \; a \; b. \; [ a \in \{ x, y \}; \; b \in \{ x, y \}; \; i \in Is ] \implies (a \; (P \; i) \leq b) \iff (a \; (P \; i) \leq b); \]
\[
\text{profile} \; S \; Is \; P; \; \text{profile} \; S \; Is \; P' \]
\[
\implies (a \; (\text{swf} \; P) \leq b) \iff (a \; (\text{swf} \; P') \leq b) \]
\[
\text{unfolding} \; iia-\text{def} \; \text{by} \; (\text{simp}, \; \text{blast}) \]

3.7 Decisiveness and Semi-decisiveness

This notion is the key to Arrow’s Theorem, and hinges on the use of strict preference [Sen70, p42].

A coalition \( C \) of agents is semi-decisive for \( x \) over \( y \) if, whenever the coalition prefers \( x \) to \( y \) and all other agents prefer the converse, the coalition prevails.

**definition** semidecisive :: \( ('a, i') \; SCF \Rightarrow ('a \; set) \Rightarrow i \; set \Rightarrow i \; set \Rightarrow 'a \Rightarrow 'a \Rightarrow \text{bool} \) where

\[
\text{semidecisive scf} \; A \; Is \; C \; x \; y \equiv\]
\[
C \subseteq Is \land (\forall P. \; \text{profile} \; A \; Is \; P \land (\forall i \in C. \; x \; (P \; i) \prec y) \land (\forall i \in Is - C. \; y \; (P \; i) \prec x) \implies x \; (\text{scf} \; P) \prec y) \]

**lemma** semidecisiveI[intro]:

\[
[ C \subseteq Is; \]
\[
\bigwedge P. \; [ \text{profile} \; A \; Is \; P; \bigwedge i. \; i \in C \implies x \; (P \; i) \prec y; \bigwedge i. \; i \in Is - C \implies y \; (P \; i) \prec x ] \implies x \; (\text{scf} \; P) \prec y \]
\[
\text{unfolding} \; \text{semidecisive-def} \; \text{by} \; \text{simp} \]

**lemma** semidecisive-coalitionD[dest]: semidecisive scf A Is C x y \implies C \subseteq Is
\[
\text{unfolding} \; \text{semidecisive-def} \; \text{by} \; \text{simp} \]

**lemma** sd-refl: [ \( C \subseteq Is; \; C \neq \{ \} \) ] \implies semidecisive scf A Is C x x
\[
\text{unfolding} \; \text{semidecisive-def} \; \text{strict-pref-def} \; \text{by} \; \text{blast} \]

A coalition \( C \) is decisive for \( x \) over \( y \) if, whenever the coalition prefers \( x \) to \( y \), the coalition prevails.

**definition** decisive :: \( ('a, i') \; SCF \Rightarrow ('a \; set) \Rightarrow i \; set \Rightarrow i \; set \Rightarrow 'a \Rightarrow 'a \Rightarrow \text{bool} \) where

\[
\text{decisive scf} \; A \; Is \; C \; x \; y \equiv\]
\[
C \subseteq Is \land (\forall P. \; \text{profile} \; A \; Is \; P \land (\forall i \in C. \; x \; (P \; i) \prec y) \implies x \; (\text{scf} \; P) \prec y) \]

**lemma** decisiveI[intro]:

\[
[ C \subseteq Is; \; \bigwedge P. \; [ \text{profile} \; A \; Is \; P; \bigwedge i. \; i \in C \implies x \; (P \; i) \prec y ] \implies x \; (\text{scf} \; P) \prec y \]
\[
\text{unfolding} \; \text{decisive-def} \; \text{by} \; \text{simp} \]

**lemma** d-imp-sd: decisive scf A Is C x y \implies semidecisive scf A Is C x y
\[
\text{unfolding} \; \text{decisive-def} \; \text{by} \; (\text{rule semidecisiveI}, \; \text{blast}+) \]

**lemma** decisive-coalitionD[dest]: decisive scf A Is C x y \implies C \subseteq Is
\[
\text{unfolding} \; \text{decisive-def} \; \text{by} \; \text{simp} \]

Anyone is trivially decisive for \( x \) against \( x \).

**lemma** d-refl: [ \( C \subseteq Is; \; C \neq \{ \} \) ] \implies decisive scf A Is C x x

15
Agent $j$ is a dictator if her preferences always prevail. This is the same as saying that she is decisive for all $x$ and $y$. 

**Definition** dictator :: (∀ x y. ∀ y ∈ A. ∀ x ∈ A. decisive scf A Is {j} x y) 

**Lemma** dictator I: [ intro ]: [ j ∈ Is; ⋀ x y. [ x ∈ A; y ∈ A ] ] =⇒ decisive scf A Is {j} x y ] =⇒ dictator scf A Is j 

**Lemma** dictator-individual [dest]: dictator scf A Is j =⇒ j ∈ Is 

**4 Arrow’s General Possibility Theorem**

The proof falls into two parts: showing that a semi-decisive individual is in fact a dictator, and that a semi-decisive individual exists. I take them in that order.

It might be good to do some of this in a locale. The complication is untangling where various witnesses need to be quantified over.

**4.1 Semi-decisiveness Implies Decisiveness**

I follow [Sen70, Chapter 3] quite closely here. Formalising his appeal to the iia assumption is the main complication here.

The witness for the first lemma: in the profile $P'$, special agent $j$ strictly prefers $x$ to $y$ to $z$, and doesn’t care about the other alternatives. Everyone else strictly prefers $y$ to each of $x$ to $z$, and inherits the relative preferences between $x$ and $z$ from profile $P$.

The model has to be specific about ordering all the other alternatives, but these are immaterial in the proof that uses this witness. Note also that the following lemma is used with different instantiations of $x$, $y$ and $z$, so we need to quantify over them here. This happens implicitly, but in a locale we would have to be more explicit.

This is just tedious.

**Lemma** decisive1-witness: 
**Assumes** has3A: hasw [x,y,z] A 
and profileP: profile A Is P 
and jIs: j ∈ Is 
**Obtains** $P'$ 
**Where** profile A Is $P'$ 
and $x (P' j) ≺ y ∧ y (P' j) ≺ z$ 
and $⋀ i. i ≠ j =⇒ y (P' i) ≺ x ∧ y (P' i) ≺ z ∧ ((x (P' i) ≼ z) = (x (P i) ≼ z)) ∧ ((z (P' i) ≼ x) = (z (P i) ≼ x))$

**Proof** 
**Let** $P' = \lambda i. (if i = j then \{(x, u) | u. u ∈ A\} \cup \{(y, u) | u. u ∈ A - \{x\}\}$
\[ \cup \{ (z, u) \mid u \in A \setminus \{x, y\} \} \]

else

\[ \cup \{ (y, u) \mid u \in A \} \]

\[ \cup \{ (x, u) \mid u \in A \setminus \{y, z\} \} \]

\[ \cup \{ (z, u) \mid u \in A \setminus \{x, y\} \} \]

\[ \cup \{ (z, u) \mid u \in A \setminus \{x, y\} \} \]

\[ \cup \{ (y, u) \mid u \in A \setminus \{y, z\} \} \]

\[ \cup \{ (x, u) \mid u \in A \setminus \{y, z\} \} \]

\[ \cup \{ (z, u) \mid u \in A \setminus \{x, y\} \} \]

\[ \cup \{ \text{if } x \, (P_i) \preceq z \text{ then } \{(x, z)\} \text{ else } \{\} \} \]

\[ \cup \{ \text{if } z \, (P_i) \preceq x \text{ then } \{(z, x)\} \text{ else } \{\} \} \]
where \( \text{profileP': profile } A \text{ Is } P' \)
and \( \text{ixyzP': } x \ (P', j) \prec y \ y \ (P', j) \prec z \)
and \( \text{ixyzP': } \bigwedge i. \ i \neq j \rightarrow y \ (P', i) \prec x \ \wedge \ y \ (P', i) \prec z \ \wedge \ ((x \ (P', i) \leq z) = (x \ (P', i) \leq z)) \ \wedge \ ((z \ (P', i) \leq z)) \)

\((P', i) \leq z) = (z \ (P', i) \leq x)) \)

by \( - \) (rule decisive1-witness, blast+)

from \( \text{iaa have } \bigwedge a. \ b. \ a \in \{x, z\}; \ b \in \{x, z\} \implies (a \ (\text{swf } P') \leq b) = (a \ (\text{swf } P') \leq b) \)

proof (rule iiaE)
from \( \text{has3A show } \{x, z\} \subseteq A \) by simp

next
fix \( i \) assume \( iIs: i \in Is \)
fix \( a \) \( b \) assume \( ab: a \in \{x, z\} \) \( b \in \{x, z\} \)
show \( (a \ (P', i) \leq b) = (a \ (P', i) \leq b) \)

proof (cases \( i = j \))
  case False
  with \( ab \) \( iIs \) \( \text{ixyzP' profileP' profileP' has3A} \)
  show \( \vdash \text{thesis unfolding profile-def by auto} \)

next
  case True
  from \( \text{profileP' jIs } \text{ixyzP'} \) \( \text{have } x \ (P', j) \prec z \)
  by (auto dest: rpr-less-trans)
  with \( \text{True ab iIs profileP' profileP' has3A} \)
  show \( \vdash \text{thesis unfolding profile-def strict-pref-def by auto} \)

qed

qd (simp-all add: profileP profileP')

moreover have \( x \ (\text{swf } P') \prec z \)

proof
  from \( \text{profileP' sd } \text{ixyzP' } \text{ixyzP'} \) \( \text{have } x \ (\text{swf } P') \prec y \) by (simp add: semidecisive-def)

moreover
from \( \text{ixyzP' } \text{ixyzP'} \) \( \text{have } \bigwedge i. \ i \in Is \implies y \ (P', i) \prec z \) by (case-tac \( i=j \), auto)

with \( wp \) \( \text{profileP' has3A have } y \ (\text{swf } P') \prec z \) by (auto dest: weak-paretoD)

moreover note \( \text{SWF-rpr[OF swf] profileP'} \)

ultimately show \( x \ (\text{swf } P') \prec z \)

unfolding universal-domain-def by (blast dest: rpr-less-trans)

qd

ultimately show \( x \ (\text{swf } P') \prec z \) unfolding strict-pref-def by blast

qd

The witness for the second lemma: special agent \( j \) strictly prefers \( z \) to \( x \) to \( y \), and everyone else strictly prefers \( z \) to \( x \) and \( y \) to \( x \). (In some sense the last part is upside-down with respect to the first witness.)

lemma decisive2-witness:
assumes \( \text{has3A: hasw } [x, y, z] \) \( A \)
and \( \text{profileP: profile } A \text{ Is } P \)
and \( jIs; j \in Is \)

obtains \( P' \)
where \( \text{profile } A \text{ Is } P' \)
and \( z \ (P', j) \prec x \ \wedge \ x \ (P', j) \prec y \)
and \( \bigwedge i. \ i \neq j \implies z \ (P', i) \prec x \ \wedge \ y \ (P', i) \prec x \ \wedge \ ((y \ (P', i) \leq z) = (y \ (P', i) \leq z)) \ \wedge \ ((z \ (P', i) \leq y)) \)

proof
let $?P' = \lambda i. (\text{if } i = j \text{ then } \{(z, u) \mid u \in A\} \cup \{(x, u) \mid u \in A - \{z\}\}) \cup \{(y, u) \mid u \in A - \{x, z\}\})$
else $(\{(z, u) \mid u \in A - \{y\}\} \cup \{(x, u) \mid u \in A - \{z\}\}) \cup \{(y, u) \mid u \in A - \{x, y, z\}\}) \cup (A - \{x, y, z\}) \times (A - \{x, y, z\})$

show $\text{profile } A \text{ Is } ?P'$
proof
fix $i$ assume $i\epsilon A$
show $\text{rpr } A (\{?P' i\})$
proof (cases $i = j$)
case $\text{True}$ with $\text{has3A}$ show $?\text{thesis}$ by $- (\text{rule rprI, simp-all add: trans-def, blast +})$
next
case $\text{False}$ hence $ij: i \neq j$.
show $\text{thesis}$
proof
from $\text{iIs}$ $\text{profileP}$ have $\text{complete } A (P i)$ by (auto simp add: $\text{rpr-def}$)
with $ij$ show $\text{complete } A (\{?P' i\})$ by (simp add: complete-def, blast)
from $\text{iIs}$ $\text{profileP}$ have $\text{refl-on } A (P i)$ by (auto simp add: $\text{rpr-def}$)
with $\text{has3A}$ $ij$ show $\text{refl-on } A (\{?P' i\})$ by (simp, blast)
from $\text{ij has3A}$ show $\text{trans } (\{?P' i\})$ by (clarsimp simp add: trans-def)
qed
qed
next
show $\text{Is } \neq \{\}$ by (rule profile-non-empty[OF profileP])
qed
from $\text{has3A}$
show $z (\{?P' i\}) \leq x \land x (\{?P' i\}) \leq y$
and $\{\forall i. i \neq j \Rightarrow z (\{?P' i\}) \leq x \land y (\{?P' i\}) \leq x \land (y (\{?P' i\}) \leq z) = (y (P i) \leq z) \land ((z (\{?P' i\}) \leq y) = (z (P i) \leq y))$
unfolding $\text{strict-pref-def}$ by auto
qed

lemma $\text{decisive2}$:
assumes $\text{has3A}: \text{hasw } [x, y, z] A$
and $\text{iia}: \text{iia suf } A \text{ Is}$
and $\text{swf}: \text{SWF suf } A \text{ Is universal-domain}$
and $\text{wp}: \text{weak-pareto suf } A \text{ Is universal-domain}$
and $\text{sd}: \text{semidecisive suf } A \text{ Is } \{j\} x y$
shows $\text{decisive suf } A \text{ Is } \{j\} z y$
proof
from $\text{sd}$ show $\text{jIs: } \{j\} \subseteq \text{Is}$ by blast
fix $P$
assume $\text{profileP: profile } A \text{ Is } P$
and $\text{jyzP: } \{\forall i. i \in \{j\} \Rightarrow z (P i) \leq y$
from $\text{has3A profileP jIs}$
obtain $?P'$
where $\text{profileP': profile } A \text{ Is } P'$


lemma decisive3:
assumes has3A: hasw \([x,y,z]\) \(A\)
and iia: iia \(sf\) \(A\) \(Is\)
and \(sf\): \(SWF\) \(sf\) \(A\) \(Is\) \(universal\)-\(domain\)
and \(wp\): \(weak\)-\(pareto\) \(sf\) \(A\) \(Is\) \(universal\)-\(domain\)
and \(sd\): \(semidecisive\) \(sf\) \(A\) \(Is\) \{\(j\)\} \(x\) \(z\)
shows decisive \(sf\) \(A\) \(Is\) \{\(j\)\} \(y\) \(z\)
using has3A decisive2[\(OF\) - iia \(sf\) \(wp\) \(sd\)] by (simp, blast)

lemma decisive4:
assumes has3A: hasw \([x,y,z]\) \(A\)
and iia: iia \(sf\) \(A\) \(Is\)
and \(sf\): \(SWF\) \(sf\) \(A\) \(Is\) \(universal\)-\(domain\)
and wp: weak-pareto swf A Is universal-domain
and sd: semidecisive swf A Is \{j\} y z
shows decisive swf A Is \{j\} y x
using has3A decisive1[OF - iia swf wp sd] by (simp, blast)

lemma decisive5:
assumes has3A: hasw [x,y,z] A
and iia: iia swf A Is
and swf: SWF swf A Is universal-domain
and wp: weak-pareto swf A Is universal-domain
and sd: semidecisive swf A Is \{j\} x y
shows decisive swf A Is \{j\} y x
proof –
from sd
have decisive swf A Is \{j\} x z by (rule decisive1[OF has3A iia swf wp])
hence semidecisive swf A Is \{j\} x z by (rule d-imp-sd)
hence decisive swf A Is \{j\} y z by (rule decisive3[OF has3A iia swf wp])
hence semidecisive swf A Is \{j\} y z by (rule d-imp-sd)
thus decisive swf A Is \{j\} y x by (rule decisive4[OF has3A iia swf wp])
qed

lemma decisive6:
assumes has3A: hasw [x,y,z] A
and iia: iia swf A Is
and swf: SWF swf A Is universal-domain
and wp: weak-pareto swf A Is universal-domain
and sd: semidecisive swf A Is \{j\} y x
shows decisive swf A Is \{j\} y z decisive swf A Is \{j\} z x decisive swf A Is \{j\} x y
proof –
from has3A have has3A': hasw [y,x,z] A by auto
show decisive swf A Is \{j\} y z by (rule decisive1[OF has3A' iia swf wp sd])
show decisive swf A Is \{j\} z x by (rule decisive2[OF has3A' iia swf wp sd])
show decisive swf A Is \{j\} y x by (rule decisive5[OF has3A' iia swf wp sd])
qed

lemma decisive7:
assumes has3A: hasw [x,y,z] A
and iia: iia swf A Is
and swf: SWF swf A Is universal-domain
and wp: weak-pareto swf A Is universal-domain
and sd: semidecisive swf A Is \{j\} x y
shows decisive swf A Is \{j\} y z decisive swf A Is \{j\} z x decisive swf A Is \{j\} x y
proof –
from sd
have decisive swf A Is \{j\} y x by (rule decisive5[OF has3A iia swf wp])
hence semidecisive swf A Is \{j\} y x by (rule d-imp-sd)
thus decisive swf A Is \{j\} y z decisive swf A Is \{j\} z x decisive swf A Is \{j\} x y
by (rule decisive6[OF has3A iia swf wp])
qed

lemma j-decisive-xy:
assumes has3A: hasw [x,y,z] A
and iia: iia swf A Is
and swf: SWF swf A Is universal-domain
and wp: weak-pareto swf A Is universal-domain
and sd: semidecisive swf A Is \{j\} x y
and uv: hasw [u,v] \{x,y,z\} shows decisive swf A Is \{j\} u v
using uv decisive1[OF has3A iia swf wp sd]
decisive2[OF has3A iia swf wp sd]
decisive5[OF has3A iia swf wp sd]
decisive7[OF has3A iia swf wp sd]
by (simp, blast)

lemma j-decisive:
assumes has3A: has 3 A
and iia: iia swf A Is
and swf: SWF swf A Is universal-domain
and wp: weak-pareto swf A Is universal-domain
and xyA: hasw [x,y] A
and sd: semidecisive swf A Is \{j\} x y
and uv: hasw [u,v] A
shows decisive swf A Is \{j\} u v
proof –
from has-extend-witness"[OF has3A xyA]
obtain z where xyzA: hasw [x,y,z] A by auto
{ assume ux: u = x and vy: v = y
  with xyzA iia swf wp sd have ?thesis by (auto intro: j-decisive-xy)
}
moreover
{ assume ux: u = x and vNEy: v \neq y
  with xyA iia swf wp sd have ?thesis by(auto intro: j-decisive-xy[of x y])
}
moreover
{ assume uy: u = y and vx: v = x
  with xyzA iia swf wp sd have ?thesis by (auto intro: j-decisive-xy)
}
moreover
{ assume uy: u = y and vNEx: v \neq x
  with uv xyA iia swf wp sd have ?thesis by (auto intro: j-decisive-xy)
}
moreover
{ assume uNExy: u \notin \{x,y\} and ex: v = x
  with uv xyA iia swf wp sd have ?thesis by (auto intro: j-decisive-xy[of x y])
}
moreover
{ assume uNExy: u \notin \{x,y\} and vy: v = y
  with uv xyA iia swf wp sd have ?thesis by (auto intro: j-decisive-xy[of x y])
}
moreover
\begin{verbatim}
{  assume u \notin \{x,y\} and v \notin \{x,y\}  
  with \( uv \in A \) and \( \text{swf} \) wp sd  
  have decisive \( A \) Is \( \{j\} \) x \( x \) u \( y \) v \( \) by (auto intro: j-decisive-xy[where \( x=x \) and \( z=u \)])  
  hence sd: semidecisive \( A \) Is \( \{j\} \) u \( x \) v \( \) by (rule d-imp-sd)  
  with \( u \in A \) v \( \notin A \) and \( \text{swf} \) wp have \( \) thesis \( \) by (auto intro: j-decisive-xy[of \( x \)])  
}  
ultimately show \( \) thesis by blast  
}
\end{verbatim}

The first result: if \( j \) is semidecisive for some alternatives \( u \) and \( v \), then they are actually a dictator.

**lemma sd-imp-dictator:**

**assumes** \( \text{has3A} \): \( \text{hasw} \) \( [x,y,z] \) \( A \)

and \( \text{iia} \): \( \text{iia swf} \) \( A \) Is

and \( \text{swf} \): \( \text{SWF} \) \( \text{swf} \) \( A \) Is universal-domain

and \( \text{wp} \): \( \text{weak-pareto} \) \( \text{swf} \) \( A \) Is universal-domain

and \( \text{uv} \): \( \text{hasw} \) \([u,v]\) \( A \)

and \( \text{sd} \): \( \text{semidecisive} \) \( \text{swf} \) \( A \) Is \( \{j\} \) \( u \) \( v \)

**shows** dictator \( \text{swf} \) \( A \) Is \( j \)

**proof**

fix \( x \) \( y \) assume \( x \in A \) and \( y \in A \)

**show** decisive \( \text{swf} \) \( A \) Is \( \{j\} \) \( x \) \( y \)

**proof**(cases \( x = y \))

case \( \text{True} \) with \( \text{sd} \) show decisive \( \text{swf} \) \( A \) Is \( \{j\} \) \( x \) \( y \) \( y \)

by (blast intro: d-refl)

next
case \( \text{False} \) with \( x \) \( y \) \( \text{iia swf} \) \( \text{wp} \) \( \text{has3A} \) \( u \) \( v \) \( \text{sd} \) show decisive \( \text{swf} \) \( A \) Is \( \{j\} \) \( x \) \( y \)

by (auto intro: j-decisive)

qed

next

from \( \text{sd} \) show \( j \in Is \) by blast

qed

4.2 The Existence of a Semi-decisive Individual

The second half of the proof establishes the existence of a semi-decisive individual. The required witness is essentially an encoding of the Condorcet paradox (aka "the paradox of voting" that shows we get tied up in knots if a certain agent didn’t have dictatorial powers.

**lemma sd-exists-witness:**

**assumes** \( \text{has3A} \): \( \text{hasw} \) \( [x,y,z] \) \( A \)

and \( \text{V1} \): \( \text{Is} = V1 \cup V2 \cup V3 \) \( \land \) \( V1 \cap V2 = \{\} \land V1 \cap V3 = \{\} \land V2 \cap V3 = \{\} \land Is \neq \{\} \)

**obtains** \( P \)

where \( \text{profile} \) \( A \) \( P \)

and \( \forall i \in V1. \ x \ (P i) \prec y \land y \ (P i) \prec z \)

and \( \forall i \in V2. \ z \ (P i) \prec x \land x \ (P i) \prec y \)

and \( \forall i \in V3. \ y \ (P i) \prec z \land z \ (P i) \prec x \)

**proof**

let \( ?P = \)

\lambda i. (if \( i \in V1 \) then \( \{ (x,u) \mid u, u \in A \} \))
Given alternatives.

Proof

A lemma obtains a minimally-sized semi-decisive set.

Hence, by splitting it up.

QED

Next, show that this set is decisive.

Proof

Assume an individual in the set.

Proof

Obtain a witness two.

And

W

Unfolding

QED

This proof is unfortunately long. Many of the statements rely on a lot of context, making it difficult to split it up.

Lemma sd-exists:

Assumes

And

TwoIs:

TwoIs.

And

S

And

Weak-pareto swf A Is universal-domain

And

wp:

Weak-pareto swf A Is universal-domain

Shows

∀ i u v. isw [u,v] A ∧ semiDecisive swf A Is {j} u v

Proof

Let

Obtain

Where

Using

Hence

Hence

With

Hence

Have

— Obtain a minimally-sized semi-decisive set.

From

Ex-has-least-nat[where P=?P and m=card, OF this]

24
obtain $V x y$ where $V Is: V \subseteq Is$
and $Vnotempty: V \neq \{}$
and $xyA: hasw [x,y] A$
and $Vsd$: semidecisive $swf A Is V x y$
and $Vmin: \bigwedge V'. \forall P V' \implies \text{card } V \leq \text{card } V'$
by blast
from $VIs \text{ finite}Is$ have $Vfinite: \text{ finite } V$ by (rule finite-subset)
— Show that minimal set contains a single individual.
from $Vfinite Vnotempty$ have $\exists j. V = \{j\}$
proof (rule finite-set-singleton contra)
assume $Vcard: 1 < \text{card } V$
then obtain $j$ where $jV: \{j\} \subseteq V$
using $has\text{-extend-witness}\{where \text{ xs=}[,}, \text{ OF card-has}\{where n=\text{card } V\}$ by auto
— Split an individual from the ”minimal” set.
let $?V1 = \{j\}$
let $?V2 = V - $?V1
let $?V3 = Is - V$
from $jV$ card-Diff-singleton[\text{of } Vfinite] $Vcard$
have $?V2card: \text{card } $?V2 > 0 \text{ card } $?V2 < \text{card } V$ by auto
hence $V2notempty: \{\} \neq $?V2$ by auto
from $jV$ $VIs$
by auto
— Show that that individual is semi-decisive for $x$ over $z$.
from $\text{has\text{-extend-witness}\{where } x,y,z] A$ by auto
obtain $z$ where $\text{threeDist: hasw } [x,y,z] A$ by auto
from $\text{sd-exists-witness}\{\text{of } \text{threeDist } ?V2V3\} VIs Vnotempty$
obtain $P$ where $\text{profileP: profile } A Is P$
and $V1xyzP: x (P j) \prec y \land y (P j) \prec z$
and $V2xyzP: \forall i \in $?V2. z (P i) \prec x \land x (P i) \prec y$
and $V3xyzP: \forall i \in $?V3. y (P i) \prec z \land z (P i) \prec x$
by (simp, blast)
have $xPz: x (swf P) \prec z$
proof (rule rpr-less-le-trans\{where y=y\})
from $\text{profileP swf show } rpr A (swf P)$ by auto
next
— $V2$ is semi-decisive, and everyone else opposes their choice. Ergo they prevail.
show $x (swf P) \prec y$
proof
from $\text{profileP } V3xyzP$
have $\forall i \in $?V3. y (P i) \prec x$ by (blast dest: rpr-less-trans)
with $\text{profileP } V1xyzP V2xyzP Vsd$
show $?\text{thesis unfolding } \text{semidecisive-def}$ by auto
qed
next
— This result is unfortunately quite tortuous.
from $\text{SWF-rpr}\{\text{of } swf\}$ show $y (swf P) \preceq z$
proof (rule rpr-less-not\{\text{of } - - \text{notI}\})
from $\text{threeDist show hasw } [z, y] A$ by auto
next
assume $zPy: z (swf P) \prec y$
have semidecisive swf A Is \( ?V2 \ z \ y \)
proof
from VIs show V \( \{j\} \subseteq Is \) by blast
next
fix \( P' \)
assume profile\( P' \); profile A Is \( P' \)
and \( V2yz' \); \( \forall i. i \in \ ?V2 \implies z (P'\ i) \prec y \)
and \( nV2yz' \); \( \forall i. i \in Is - \ ?V2 \implies y (P'\ i) \prec z \)
from iia have \( \forall a. b. \[ a \in \{y, z\}; \ b \in \{y, z\} \] \implies (a (swf P) \preceq b) = (a (swf P') \preceq b) \)
proof(rule iiaE)
from threeDist show yzA; \( \{y, z\} \subseteq A \) by simp
next
fix i assume iIs; i \in Is
fix a b assume ab; a \in \{y, z\} b \in \{y, z\}
with VIs profile\( P \) \( V2yzP \)
have \( V2yzP; \forall i \in \ ?V2. \ z (P\ i) \prec y \) by (blast dest: rpr-less-trans)
show \( (a (P'\ i) \preceq b) = (a (P\ i) \preceq b) \)
proof(cases i \in \ ?V2)
case True
with VIs profile\( P' \) profile\( P' \) ab V2yzP threeDist
show \( \{y, z\} \subseteq A \) by simp
next
case False
from V1yzP \( V2yzP \)
have \( \forall i \in Is - \ ?V2. \ y (P\ i) \prec z \) by auto
with iIs False VIs jV profile\( P \) profile\( P' \) ab nV2yz' threeDist
show \( \{y, z\} \subseteq A \) by simp
qed
qed (simp-all add: profile\( P \) profile\( P' \))
with \( zPy \) show \( z (swf P) \preceq y \) unfolding strict-pref-def by blast
qed
with VIs Vsd Vmin[where V=\( ?V2 \)] V2card V2notempty threeDist show False
by auto
qed (simp add: profile\( P \) threeDist)
qed
have semidecisive swf A Is \( ?V1 \ x \ z \)
proof
from jV VIs show \( \{j\} \subseteq Is \) by blast
next
— Use iia to show the SWF must allow the individual to prevail.
fix \( P' \)
assume profile\( P' \); profile A Is \( P' \)
and \( V1yz' \); \( \forall i. i \in \ ?V1 \implies x (P'\ i) \prec z \)
and \( nV1yz' \); \( \forall i. i \in Is - \ ?V1 \implies z (P'\ i) \prec x \)
from iia have \( \forall a. b. \[ a \in \{x, z\}; \ b \in \{x, z\} \] \implies (a (swf P) \preceq b) = (a (swf P') \preceq b) \)
proof(rule iiaE)
from threeDist show xzA; \( \{x, z\} \subseteq A \) by simp
next
fix i assume iIs; i \in Is
fix a b assume ab; a \in \{x, z\} b \in \{x, z\}
show \( (a (P'\ i) \preceq b) = (a (P\ i) \preceq b) \)
proof (cases \( i \in ?V1 \))

case True

with \( jV \) \( VIs \) profileP \( V1xyzP \)

have \( \forall i \in ?V1. \: x (P i) \prec z \) by (blast dest: rpr-less-trans)

with True \( jV \) \( VIs \) profileP profileP' ab \( V1yz' \) threeDist

show ?thesis unfolding strict-pref-def profile-def by auto

next

case False

from \( V2xyzP \) \( V3xyzP \)

have \( \forall i \in Is \setminus ?V1. \: z (P i) \prec x \) by auto

with \( iIs \) False \( VIs \) profileP profileP' ab \( nV1yz' \) threeDist

show ?thesis unfolding strict-pref-def profile-def by auto

qed

qed (simp-all add: profileP profileP')

with \( xPz \) show \( x (swf P') \prec z \) unfolding strict-pref-def by blast

qed

with \( xyz \) \( Vsd \) show \( ?thesis \) by blast

qed

4.3 Arrow’s General Possibility Theorem

Finally we conclude with the celebrated “possibility” result. Note that we assume the set of individuals is finite; [Rou79] relaxes this with some fancier set theory. Having an infinite set of alternatives doesn’t matter, though the result is a bit more plausible if we assume finiteness [Sen70, p54].

theorem ArrowGeneralPossibility:

assumes has3A: has 3 A

and finiteIs: finite Is

and has2Is: has 2 Is

and iia: iia swf A Is

and swf: SWF swf A Is universal-domain

and wp: weak-pareto swf A Is universal-domain

obtains j where dictator swf A Is j

using sd-imp-dictator \( \{OF has3A iia swf wp\} \)

sd-exists \( \{OF has3A finiteIs has2Is iia swf wp\} \)

by blast

5 Sen’s Liberal Paradox

5.1 Social Decision Functions (SDFs)

To make progress in the face of Arrow’s Theorem, the demands placed on the social choice function need to be weakened. One approach is to only require that the set of alternatives that society ranks highest (and is otherwise indifferent about) be non-empty.
Following [Sen70, Chapter 4*], a *Social Decision Function* (SDF) yields a choice function for every profile.

**definition**

\[ SDF : (\{ a, i \} SCF \Rightarrow \{ a \ set \Rightarrow \{ i \ set \Rightarrow (\{ a, i \} Profile \Rightarrow \text{bool}) \Rightarrow \text{bool} \}) \]

**lemma** *SDFI*\[ intro]:

\((\forall P. \text{Pcond} A Is P \Rightarrow \text{choiceFn} A (\text{sdf} P)) \Rightarrow \text{SDF sdf} A \text{ Is Pcond} \)

**unfolding** *SDF-def* by simp

**lemma** *SWF-SDF*:

assumes *finiteA*: \(\text{finite} A\)

shows *SWF scf A Is universal-domain* = \(\Rightarrow \text{SDF scf} A \text{ Is universal-domain} \)

**unfolding** *SDF-def* *SWF-def* by (blast dest: rpr-choiceFn[OF *finiteA*])

In contrast to SWFs, there are SDFs satisfying Arrow’s (relevant) requirements. The lemma uses a witness to show the absence of a dictatorship.

**lemma** *SDF-nodictator-witness*:

assumes *has2A*: \(\text{has 2} A\)

and *has2Is*: \(\text{has 2} Is\)

obtains \( P \)

where \(\text{profile} A \text{ Is} P\)

and \( x (P i) \prec y\)

and \( y (P j) \prec x\)

**proof**

let \( ?P = \lambda k. (\text{if} k = i \text{ then} \{ (x, u) \mid u. u \in A \} \)

\( \cup \{ (y, u) \mid u. u \in A - \{x\} \}\)

\( \text{else} \ (\{ (y, u) \mid u. u \in A \}

\( \cup \{ (x, u) \mid u. u \in A - \{y\} \} ))\)

\( \cup (A - \{x,y\}) \times (A - \{x,y\}) \)

show \(\text{profile} A \text{ Is} \ ?P\)

**proof**

fix \( i \) assume *iis*: \( i \in Is\)

from *has2A* show \(\text{rpr} A (\text{?P} i)\)

by -(rule rprI, simp-all add: trans-def, blast+)

next

from *has2Is* show \( Is \neq \{ \} \) by auto

qed

from *has2A* *has2Is*

show \( x (\text{?P} i) \prec y\)

and \( y (\text{?P} j) \prec x\)

**unfolding** *strict-pref-def* by auto

qed

**lemma** *SDF-possibility*:

assumes *finiteA*: \(\text{finite} A\)

and *has2A*: \(\text{has 2} A\)

and *has2Is*: \(\text{has 2} Is\)

obtains \(\text{sdf}\)

where \(\text{weak-pareto sdf} A \text{ Is universal-domain}\)

and \(\text{iia sdf} A \text{ Is}\)

28
and \( \neg(\exists j. \text{dictator } \text{sdf } A \text{ Is } j) \)
and SDF \( \text{sdf } A \text{ Is universal-domain} \)

**proof** –

let \( \text{sdf} = \lambda P. \{ (x, y) . x \in A \land y \in A \land \neg ((\forall i \in \text{Is}. \ y (P i) \preceq x) \land (\exists i \in \text{Is}. \ y (P i) < x)) \} \)

have weak-pareto \( \text{sdf } A \text{ Is universal-domain} \)
by (rule, unfold strict-pref-def, auto dest: profile-non-empty)

moreover
have \( \text{ia } \text{sdf } A \text{ Is unfolding strict-pref-def by auto} \)

moreover
have \( \neg(\exists j. \text{dictator } \text{sdf } A \text{ Is } j) \)

**proof**

assume \( \exists j. \text{dictator } \text{sdf } A \text{ Is } j \)
then obtain \( j \) where \( j\text{Is} \): \( j \in \text{Is} \)
and \( j\text{D} \): \( \forall x \in A. \forall y \in A. \text{decisive } \text{sdf } A \text{ Is } \{j\} \ x \ y \)

unfolding decisive-def decisive-def by auto

from \( j\text{Is} \) has-witness-two[\( \text{OF has2Is} \)] obtain \( i\text{Is} \) where \( i\text{Is} \text{}[i,j] \) Is
by auto

from has-witness-two[\( \text{OF has2A} \)] obtain \( x \ y \) where \( x\text{A} \text{}[x,y] \) A by auto

from \( x\text{A} i\text{Is} \) obtain \( P \)
where \( \text{profileP} \): \( \text{profile } A \text{ Is } P \)
and \( y\text{Pix} \): \( y (P j) \preceq x \)

by (rule SDF-nodictator-witness)

from \( \text{profileP} \ j\text{D} j\text{Is} x\text{A} y\text{Pix} \) have \( y (\text{sdf } P) \preceq x \)

unfolding decisive-def by simp

moreover
from \( i\text{Is} x\text{A} y\text{Pix} y\text{Pix} \) have \( x (\text{sdf } P) \preceq y \)

unfolding strict-pref-def by auto

ultimately show False

unfolding strict-pref-def by blast

qed

moreover
have SDF \( \text{sdf } A \text{ Is universal-domain} \)

**proof**

fix \( P \) assume \( \text{ud} \): \( \text{universal-domain } A \text{ Is } P \)

show choiceFn \( A \) (\( \text{sdf } P \) )

proof (rule r-c-qt-imp-cf[\( \text{OF finiteA} \)])

show complete \( A \) (\( \text{sdf } P \) ) and refl-on \( A \) (\( \text{sdf } P \) )

unfolding strict-pref-def by auto

show quasi-trans (\( \text{sdf } P \) )

proof

fix \( x \ y \ z \) assume \( xy \): \( x (\text{sdf } P) \preceq y \) and \( yz \): \( y (\text{sdf } P) \preceq z \)

from \( xy \ yz \) have \( x\text{yzA} \): \( x \in A \ y \in A \ z \in A \)

unfolding strict-pref-def by auto

from \( xy \ yz \) have \( Ax\text{Ry} \): \( \forall i \in \text{Is}. \ x (P i) \preceq y \)
and \( Ex\text{Py} \): \( \exists i \in \text{Is}. \ x (P i) < y \)
and \( Ay\text{Rz} \): \( \forall i \in \text{Is}. \ y (P i) \preceq z \)

unfolding strict-pref-def by auto

from \( Ax\text{Ry} \ Ay\text{Rz} \) \( \text{ud} \) have \( Ax\text{Rz} \): \( \forall i \in \text{Is}. \ x (P i) \preceq z \)
by \(-\) (unfold universal-domain-def, blast dest: rpr-le-trans)
from ExPy A y Rz ad have ExPz: \(\exists i \in Is. \ x (\ p \ i) \prec \ z\)
by \(-\) (unfold universal-domain-def, blast dest: rpr-less-le-trans)
from xyz A x Rz ExPz show \(x (\ ?sdf \ p) \prec \ z\) unfolding strict-pref-def by auto
qed
qed
qed
ultimately show thesis ..
qed

Sen makes several other stronger statements about SDFs later in the chapter. I leave these for future work.

5.2 Sen’s Liberal Paradox

Having side-stepped Arrow’s Theorem, Sen proceeds to other conditions one may ask of an SCF. His analysis of liberalism, mechanised in this section, has attracted much criticism over the years [AK96].

Following [Sen70, Chapter 6*], a liberal social choice rule is one that, for each individual, there is a pair of alternatives that she is decisive over.

**definition** liberal :: \('a, 'i) SCF => 'a set => 'i set => bool where
liberal scf A Is \equiv
\(\forall i \in Is. \exists x \in A. \exists y \in A. \ x \neq y\)
\& decisive scf A Is \{i\} x y \& decisive scf A Is \{i\} y x

**lemma** liberalE:
[ liberal scf A Is; i \in Is ]
\implies \exists x \in A. \exists y \in A. \ x \neq y
\& decisive scf A Is \{i\} x y \& decisive scf A Is \{i\} y x
by (simp add: liberal-def)

This condition can be weakened to require just two such decisive individuals; if we required just one, we would allow dictatorships, which are clearly not liberal.

**definition** minimally-liberal :: \('a, 'i) SCF => 'a set => 'i set => bool where
minimally-liberal scf A Is \equiv
\(\exists i \in Is. \exists j \in Is. \ i \neq j\)
\& \(\exists x \in A. \exists y \in A. \ x \neq y\)
\& decisive scf A Is \{i\} x y \& decisive scf A Is \{i\} y x
\& \(\exists x \in A. \exists y \in A. \ x \neq y\)
\& decisive scf A Is \{j\} x y \& decisive scf A Is \{j\} y x
)

**lemma** liberal-imp-minimally-liberal:
assumes has2Is: has 2 Is
and L: liberal scf A Is
shows minimally-liberal scf A Is
proof –
from has-extend-witness[where xs=[], OF has2Is]
obtain i where i: i \in Is by auto
with has-extend-witness[where xs=[i], OF has2Is]
obtain j where j: j \in Is i \neq j by auto
from L i j show \$thesis
The key observation is that once we have at least two decisive individuals we can complete
the Condorcet (paradox of voting) cycle using the weak Pareto assumption. The details of
the proof don’t give more insight.

Firstly we need three types of profile witnesses (one of which we saw previously). The
main proof proceeds by case distinctions on which alternatives the two liberal agents are
decisive for.

lemma liberal-witness-two = SDF-nodictator-witness

lemma liberal-witness-three:
assumes threeA: hasw [x,y,v] A
and twoIs: hasw [i,j] Is
obtains P
where profile A Is P
and x (P i)≺ y
and ∃ (P j)≺ x
and ∀ i ∈ Is. y (P i)≺ v
proof –
let ?P =
λa. if a = i then \{(x, u) | u ∈ A \} ∪ \{(y, u) | u ∈ A - \{x\}\} ∪ (A - \{x,y\})
else \{(y, u) | u ∈ A \} ∪ \{(v, u) | u ∈ A - \{y\}\} ∪ (A - \{v,y\}) × (A - \{v,y\})
have profile A Is ?P
proof
fix i assume iis: i ∈ Is
show rpr A (?P i)
proof
show complete A (?P i) by (simp blast)
from threeA iis show refl-on A (?P i) by (simp blast)
from threeA iis show trans (?P i) by (clarsimp simp add: trans-def)
qed
next
from twoIs show Is ≠ {} by auto
qed
moreover
from threeA twoIs have x (?P i)≺ y v (?P j)≺ x ∃ i ∈ Is. y (?P i)≺ v
  unfolding strict-pref-def by auto
ultimately show ?thesis ..
qed

lemma liberal-witness-four:
assumes fourA: hasw [x,y,u,v] A
and twoIs: hasw [i,j] Is
obtains P
where profile A Is P
and x (P i)≺ y
and v (P j)≺ v
proof –
let ?P =
λa. if a = i then \{(x, u) | u ∈ A \} ∪ \{(y, u) | u ∈ A - \{x\}\} ∪ (A - \{x,y\})
else \{(y, u) | u ∈ A \} ∪ \{(v, u) | u ∈ A - \{y\}\} ∪ (A - \{v,y\}) × (A - \{v,y\})
have profile A Is ?P
proof
fix i assume iis: i ∈ Is
show rpr A (?P i)
proof
show complete A (?P i) by (simp blast)
from threeA iis show refl-on A (?P i) by (simp blast)
from threeA iis show trans (?P i) by (clarsimp simp add: trans-def)
qed
next
from twoIs show Is ≠ {} by auto
qed
moreover
from threeA twoIs have x (?P i)≺ y v (?P j)≺ x ∃ i ∈ Is. y (?P i)≺ v
  unfolding strict-pref-def by auto
ultimately show ?thesis ..
qed
\[ \forall i \in I_s. v(P_i) \prec x \land y(P_i) \prec u \]

proof

- let \(?P = \lambda a. \text{if } a = i \text{ then } \{(v, w) \mid w, w \in A\} \cup \{(x, w) \mid w, w \in A - \{v\}\} \cup \{(y, w) \mid w, w \in A - \{v, x\}\} \cup \{A - \{v, x, y\}\} \times (A - \{v, x, y\}) \text{ else } \{(y, w) \mid w, w \in A\} \cup \{(x, w) \mid w, w \in A - \{y\}\} \cup \{(y, w) \mid w, w \in A - \{u, y\}\} \cup \{A - \{u, v, y\}\} \times (A - \{u, v, y\})\]

have profile \(A I_s \ ?P \)

proof

fix i assume iis: \(i \in I_s\)

show \(\text{rpr } A (\ ?P i)\)

proof

show complete \(A (\ ?P i)\) by (simp, blast)
from fourA iis show refl-on \(A (\ ?P i)\) by (simp, blast)
from fourA iis show trans \(\ ?P i\) by (clarsimp simp add: trans-def)

qed

next

from twoIs show \(\text{Is} \neq \{\}\) by auto

qed

moreover

from fourA twoIs have \(x (\ ?P i) \prec y u (\ ?P j) \prec v \forall i \in I_s. v (\ ?P i) \prec x \land y (\ ?P i) \prec u \)
by (unfold strict-pref-def, auto)

ultimately show thesis..

qed

The Liberal Paradox: having two decisive individuals, an SDF and the weak pareto assumption is inconsistent.

corollary LiberalParadox:

assumes SDF: \(\text{SDF sdf } A \ I_s \text{ universal-domain}\)
and ml: \(\text{minimally-liberal sdf } A \ I_s\)
and wp: \(\text{weak-pareto sdf } A \ I_s \text{ universal-domain}\)

shows False

proof

from ml obtain \(i \ j \ x \ y \ u \ v\)

where \(i: i \in I_s \land j: j \in I_s \land ij: i \neq j\)
and \(x: x \in A \land y: y \in A \land u: u \in A \land v: v \in A\)
and xy: \(x \neq y\)
and dixy: \(\text{decisive sdf } A \ I_s \{i\} x y\)
and dixy: \(\text{decisive sdf } A \ I_s \{i\} y x\)
and uw: \(u \neq v\)
and djuv: \(\text{decisive sdf } A \ I_s \{j\} u v\)
and djuv: \(\text{decisive sdf } A \ I_s \{j\} v u\)
by (unfold minimally-liberal-def, auto)

from \(i \ j \ ij\) have twoIs: \(\text{hasw }[i,j] \ I_s \) by simp

assume \(xu: x = u \land yv: y = v\)
from \(xy \ x y\) have twoA: \(\text{hasw }[x, y] \ A \) by simp

obtain \(P\)
where \( \text{profile } A \ Is P \ x \ (P_i) \prec y \ y \ (P_j) \prec x \)

using liberal-witness-two\([\text{OF twoA twoIs}]\) by blast

with \( i \ j \ \text{dizy djvu xu yv have False} \)
by (unfold decisive-def strict-pref-def, blast)

moreover

\{ 
assume \( xw : x = u \ and \ yw : y \neq v \)
with \( xy \ wv \ xu \ x \ y \ v \ have \ \text{threeA: hasw } [x,y,v] \ A \ by \ \text{simp} \)

obtain \( P \)
where \( \text{profileP: profile } A \ Is P \)
and \( xPiy : x \ (P_i) \prec y \)
and \( vPjx : v \ (P_j) \prec x \)
and \( AyPv : \forall i \ in \ Is. \ y \ (P_i) \prec v \)

using liberal-witness-three\([\text{OF threeA twoIs}]\) by blast

from \( vPjx \ j \ djvu \ xu \ profileP \ have \ vPx : v \ (sdf \ p) \prec x \)
by (unfold decisive-def strict-pref-def, auto)

from \( xPiy \ i \ \text{dizy profileP have xPy : x \ (sdf \ p) \prec y} \)
by (unfold decisive-def strict-pref-def, auto)

from \( AyPv \ weak-paretoD(\text{OF wp - y v]} \ profileP \ have \ yPv : y \ (sdf \ p) \prec v \)
by auto

from \( \text{threeA profileP SDF have choiceSet } \{x,y,v\} \ (sdf \ p) \neq \{\} \)
by (simp add: SDF-def choiceFn-def)

with \( vP \ xPy \ yPv \ have \ False \)
by (unfold choiceSet-def strict-pref-def, blast)

\}

moreover

\{ 
assume \( xv : x = v \ and \ yu : y = u \)

from \( xy \ x \ y \ have \ \text{twoA: hasw } [x,y] \ A \ by \ auto \)

obtain \( P \)
where \( \text{profileP: profile } A \ Is P \ x \ (P_i) \prec y \ y \ (P_j) \prec x \)

using liberal-witness-two\([\text{OF twoA twoIs}]\) by blast

with \( i \ j \ \text{dizy djvu xu yv have False} \)
by (unfold decisive-def strict-pref-def, blast)

\}

moreover

\{ 
assume \( xv : x = v \ and \ yu : y \neq u \)
with \( xy \ wv \ u \ x \ y \ have \ \text{threeA: hasw } [x,y,u] \ A \ by \ simp \)

obtain \( P \)
where \( \text{profileP: profile } A \ Is P \)
and \( xPiy : x \ (P_i) \prec y \)
and \( uPjx : u \ (P_j) \prec x \)
and \( AyPv : \forall i \ in \ Is. \ y \ (P_i) \prec u \)

using liberal-witness-three\([\text{OF threeA twoIs}]\) by blast

from \( uPjx \ j \ djvu \ xu \ profileP \ have \ uPx : u \ (sdf \ p) \prec x \)
by (unfold decisive-def strict-pref-def, auto)

from \( xPiy \ i \ \text{dizy profileP have xPy : x \ (sdf \ p) \prec y} \)
by (unfold decisive-def strict-pref-def, auto)

from \( AyPv \ weak-paretoD(\text{OF wp - y u]} \ profileP \ have \ yPv : y \ (sdf \ p) \prec u \)
by auto
from threeA profileP SDF have choiceSet \{x,y,u\} (sdf P) \neq \{\}
  by (simp add: SDF-def choiceFn-def)
with uPx xPy yPu have False
  by (unfold choiceSet-def strict-pref-def, blast)
}
moreover
{
  assume xu: x \neq u and xv: x \neq v and yu: y = u
  with v x y xy uw xu have threeA: hasw \{y,x,v\} A by simp
  obtain P
    where profileP: profile A Is P
    and yPiz: y (P i) \not\prec x
    and uPyj: v (P j) \not\prec y
    and AxPv: \forall i \in Is. x (P i) \not\prec v
    using liberal-witness-three[OF threeA twoIs] by blast
  from yPiz i digy profileP have yPz: y (sdf P) \not\prec x
    by (unfold decisive-def strict-pref-def, auto)
  from uPyj j digy profileP have uPy: v (sdf P) \not\prec y
    by (unfold decisive-def strict-pref-def, auto)
  from AxPv weak-paretoD[OF wp - x v] profileP have xPv: x (sdf P) \not\prec v
    by auto
from threeA profileP SDF have choiceSet \{x,y,v\} (sdf P) \neq \{\}
  by (simp add: SDF-def choiceFn-def)
with yPx vPy xPv have False
  by (unfold choiceSet-def strict-pref-def, blast)
}
moreover
{
  assume xu: x \neq u and xv: x \neq v and yw: y = v
  with u x y xy uw xu have threeA: hasw \{y,x,u\} A by simp
  obtain P
    where profileP: profile A Is P
    and yPiz: y (P i) \not\prec x
    and uPyj: v (P j) \not\prec u
    and AxPv: \forall i \in Is. x (P i) \not\prec u
    using liberal-witness-three[OF threeA twoIs] by blast
  from yPiz i digy profileP have yPz: y (sdf P) \not\prec x
    by (unfold decisive-def strict-pref-def, auto)
  from uPyj j digw yw profileP have uPy: u (sdf P) \not\prec y
    by (unfold decisive-def strict-pref-def, auto)
  from AxPv weak-paretoD[OF wp - x u] profileP have xPu: x (sdf P) \not\prec u
    by auto
from threeA profileP SDF have choiceSet \{x,y,u\} (sdf P) \neq \{\}
  by (simp add: SDF-def choiceFn-def)
with yPx uPy xPu have False
  by (unfold choiceSet-def strict-pref-def, blast)
}
moreover
{
  assume xu: x \neq u and xv: x \neq v and yu: y \neq u and yv: y \neq v
with \( u \ v \ x \ y \ xy \ uv \ xu \) have fourA: \([x,y,u,v] \) A by simp

obtain \( P \)
where profileP: profile A Is P
and xPy: \( x (P \ y) \prec y \)
and uPjv: \( u (P \ j) \prec v \)
and AvPxAyPu: \( \forall i \in Is, \ v (P \ i) \prec x \land y (P \ i) \prec u \)

using liberal-witness-four[OF fourA twoIs] by blast

from xPy i dixz profileP have xPy: \( x (sdf P) \prec y \)
by (unfold decisive-def strict-pref-def, simp)

from uPjv j djuv profileP have uPv: \( u (sdf P) \prec v \)
by (unfold decisive-def strict-pref-def, simp)

from AvPxAyPu weak-paretoD[OF wp] profileP x y u v
have vPx: \( v (sdf P) \prec x \) and yPu: \( y (sdf P) \prec u \) by auto

from fourA profileP SDF have choiceSet \( \{x,y,u,v\} (sdf P) \neq \{\} \)
by (simp add: SDF-def choiceFn-def)

with \( xPy \ uPv \ vPx \ yPu \) have False
by (unfold choiceSet-def strict-pref-def, blast)

ultimately show False by blast

qed

6 May’s Theorem

May’s Theorem [May52] provides a characterisation of majority voting in terms of four conditions that appear quite natural for a priori unbiased social choice scenarios. It can be seen as a refinement of some earlier work by Arrow [Arr63, Chapter V.1].

The following is a mechanisation of Sen’s generalisation [Sen70, Chapter 5*]: originally Arrow and May consider only two alternatives, whereas Sen’s model maps profiles of full RPRs to a possibly intransitive relation that does at least generate a choice set that satisfies May’s conditions.

6.1 May’s Conditions

The condition of anonymity asserts that the individuals’ identities are not considered by the choice rule. Rather than talk about permutations we just assert the result of the SCF is the same when the profile is composed with an arbitrary bijection on the set of individuals.

definition anonymous :: \((a, 'i) SCF \Rightarrow 'a set \Rightarrow 'i set \Rightarrow bool\)
where
anonymous scf A Is \( \equiv \)
\((\forall P f x y. \ profile A Is P \land bij-betw f Is Is \land x \in A \land y \in A \rightarrow (x (scf P) \preceq y) = (x (scf (P \circ f)) \preceq y))\)

lemma anonymousI[intro]:
\((\forall P f x y. \ profile A Is P; \ bij-betw f Is Is;\)
\( x \in A; y \in A \rightarrow (x (scf P) \preceq y) = (x (scf (P \circ f)) \preceq y))\)

\( \Rightarrow \) anonymous scf A Is

unfolding anonymous-def by simp
lemmas anonymousD:
\[ \text{anonymous scf } A \text{ Is} \equiv \begin{cases} \text{profile } A \text{ Is } P; & \text{bij-betw } f \text{ Is}; & x \in A; & y \in A \end{cases} \]
\[ \Rightarrow (x \ (\text{scf } P) \preceq y) = (x \ (\text{scf } (P \circ f)) \preceq y) \]
unfolding anonymous-def by simp

Similarly, an SCF is neutral if it is insensitive to the identity of the alternatives. This is Sen’s characterisation [Sen70, p72].

definition neutral :: ('a, 'i) SCF ⇒ 'a set ⇒ 'i set ⇒ bool where
neutral scf A Is ≡
(∀ P P’ x y z w. profile A Is P \land profile A Is P’ \land x ∈ A \land y ∈ A \land z ∈ A \land w ∈ A
\land (∀ i ∈ Is, x \ (P i) \preceq y \iff z \ (P’ i) \preceq w) \land (∀ i ∈ Is, y \ (P i) \preceq x \iff w \ (P’ i) \preceq z)
\Rightarrow ((x \ (\text{scf } P) \preceq y \iff z \ (\text{scf } P’) \preceq w) \land (y \ (\text{scf } P) \preceq x \iff w \ (\text{scf } P’) \preceq z)))

lemma neutral[intro]:
\( \land P P’ x y z w. \)
\[ \begin{cases} \text{profile } A \text{ Is } P; & \text{profile } A \text{ Is } P’; & \{x,y,z,w\} \subseteq A; & \end{cases} \]
\[ \forall i. \ i \in Is \Rightarrow x \ (P i) \preceq y \iff z \ (P’ i) \preceq w; \]
\[ \forall i. \ i \in Is \Rightarrow y \ (P i) \preceq x \iff w \ (P’ i) \preceq z \]
\[ \Rightarrow ((x \ (\text{scf } P) \preceq y \iff z \ (\text{scf } P’) \preceq w) \land (y \ (\text{scf } P) \preceq x \iff w \ (\text{scf } P’) \preceq z)) \]
unfolding neutral-def by simp

lemma neutralD:
\[ \land P P’ x y z w. \]
\[ \begin{cases} \text{profile } A \text{ Is } P; & \text{profile } A \text{ Is } P’; & \{x,y,z,w\} \subseteq A; & \end{cases} \]
\[ \forall i. \ i \in Is \Rightarrow x \ (P i) \preceq y \iff z \ (P’ i) \preceq w; \]
\[ \forall i. \ i \in Is \Rightarrow y \ (P i) \preceq x \iff w \ (P’ i) \preceq z \]
\[ \Rightarrow (x \ (\text{scf } P) \preceq y \iff z \ (\text{scf } P’) \preceq w) \land (y \ (\text{scf } P) \preceq x \iff w \ (\text{scf } P’) \preceq z) \]
unfolding neutral-def by simp

Neutrality implies independence of irrelevant alternatives.

lemma neutral-iiA: neutral scf A Is ⇒ iia scf A Is
unfolding neutral-def by (rule, auto)

Positive responsiveness is a bit like non-manipulability: if one individual improves their opinion of x, then the result should shift in favour of x.

definition positively-responsive :: ('a, 'i) SCF ⇒ 'a set ⇒ 'i set ⇒ bool where
positively-responsive scf A Is ≡
(∀ P P’ x y. profile A Is P \land profile A Is P’ \land x ∈ A \land y ∈ A
\land (∀ i ∈ Is, x \ (P i) \prec y \iff x \ (P’ i) \prec y) \land (x \ (P i) \approx y \iff x \ (P’ i) \approx y)
\land (\exists k \in Is, \ (x \ (P k) \approx y \land x \ (P’ k) \approx y) \lor (y \ (P k) \prec x \land x \ (P’ k) \prec y))
\Rightarrow x \ (\text{scf } P) \preceq y \iff x \ (\text{scf } P’) \preceq y)

lemma positively-responsive[intro]:
assumes I: \( \land P P’ x y. \)
\[ \begin{cases} \text{profile } A \text{ Is } P; & \text{profile } A \text{ Is } P’; & x \in A; & y \in A; & \end{cases} \]
\[ \forall i. \ i \in Is; \ (P i) \prec y \Rightarrow (P’ i) \prec y; \]
\[ \forall i. \ i \in Is; \ (P i) \approx y \Rightarrow (P’ i) \approx y; \]
\[ \exists k \in Is. \ (x \ (P k) \approx y \land x \ (P’ k) \approx y) \lor (y \ (P k) \prec x \land x \ (P’ k) \prec y); \]
\[ x \text{ (scf } P \text{)} \preceq y \] 
\[ \implies x \text{ (scf } P' \text{)} \prec y \]

shows positively-responsive scf A Is

unfolding positively-responsive-def

by (blast intro: I)

lemma positively-responsiveD:
\[
\begin{align*}
&\text{positively-responsive scf A Is;} \\
&\quad \text{profile A Is P; profile A Is P'; } x \in A; y \in A; \\
&\quad \text{\[ } i \in \text{Is); } x (P i) \prec y \implies x (P' i) \preceq y; \\
&\quad \text{\[ } i \in \text{Is); } x (P i) \approx y \implies x (P' i) \preceq y; \\
&\quad \exists k \in \text{Is}. \ (x (P k) \approx y \land x (P' k) \prec y) \lor (y (P k) \prec x \land x (P' k) \preceq y); \\
&\quad x \text{ (scf } P \text{)} \preceq y \] \\
&\implies x \text{ (scf } P' \text{)} \prec y
\end{align*}
\]

unfolding positively-responsive-def

apply clasimp

apply (erule allE[where x=P])

apply (erule allE[where x=P'])

apply (erule allE[where x=x])

apply (erule allE[where x=y])

by auto

6.2 The Method of Majority Decision satisfies May’s conditions

The method of majority decision (MMD) says that if the number of individuals who strictly prefer \( x \) to \( y \) is larger than or equal to those who strictly prefer the converse, then \( x R y \). Note that this definition only makes sense for a finite population.

definition MMD :: 'i set ⇒ ('a, 'i) SCF where
\[ MMD \text{ Is } P \equiv \{ (x, y). \text{ card } \{ i \in \text{Is). } x (P i) \prec y \} \geq \text{ card } \{ i \in \text{Is). } y (P i) \prec x \} \} \]

The first part of May’s Theorem establishes that the conditions are consistent, by showing that they are satisfied by MMD.

lemma MMD-l2r:
\[
\begin{align*}
&\text{fixes A :: 'a set} \\
&\quad \text{and Is :: 'i set} \\
&\quad \text{assumes finiteIs: finite Is} \\
&\quad \text{shows SCF (MMD Is) A Is universal-domain} \\
&\quad \quad \text{and anonymous (MMD Is) A Is} \\
&\quad \quad \text{and neutral (MMD Is) A Is} \\
&\quad \quad \text{and positively-responsive (MMD Is) A Is}
\end{align*}
\]

proof –

show SCF (MMD Is) A Is universal-domain

proof

fix P show complete A (MMD Is P)
by (rule completeI, unfold MMD-def, simp, arith)
qed

show anonymous (MMD Is) A Is

proof

fix P

fix x y :: 'a

fix f assume bijf: bij-betw f Is Is
show \((x (\text{MMD Is } P) \preceq y) = (x (\text{MMD Is } (P \circ f)) \preceq y)\)

using card-compose-bij[OF bijf, where \(P=\lambda i. x \ (P_i) < y\)]

card-compose-bij[OF bijf, where \(P=\lambda i. y \ (P_i) < x\)]

unfolding MMD-def by simp

qed

next

show neutral \((\text{MMD Is } A) \text{ Is}\)

proof

fix \(P \ P'\)

fix \(x \ y \ z \ w\) assume \(xyzwA\ \{x,y,z,w\} \subseteq A\)

assume \(xyzw\ \land i. \ i \in Is \implies (x \ (P_i) \prec y) = (z \ (P'_i) \prec w)\)

and \(yvwz\ \land i. \ i \in Is \implies (y \ (P_i) \preceq x) = (w \ (P'_i) \preceq z)\)

from \(xyzwA \ xzwv \ yvwz\)

have \(\{ i \in Is. \ x \ (P_i) \prec y \} = \{ i \in Is. \ z \ (P'_i) \prec w \}\)

and \(\{ i \in Is. \ y \ (P_i) \preceq x \} = \{ i \in Is. \ w \ (P'_i) \preceq z \}\)

unfolding strict-pref-def by auto

thus \((x (\text{MMD Is } P) \preceq y) = (z (\text{MMD Is } P') \preceq w) \land (y (\text{MMD Is } P) \preceq x) = (w (\text{MMD Is } P') \preceq z)\)

unfolding MMD-def by simp

qed

next

show positively-responsive \((\text{MMD Is } A) \text{ Is}\)

proof

fix \(P \ P'\)

fix \(x \ y\) assume \(xyA\ \{x, y\} \subseteq A\)

assume \(xPy: \ \land i. \ [i \in Is; \ x \ (P_i) \prec y] \implies x \ (P'_i) \prec y\)

and \(xly: \ \land i. \ [i \in Is; \ x \ (P_i) \approx y] \implies x \ (P'_i) \preceq y\)

and \(k: \ \exists k \in Is. \ x \ (P_k) \approx y \land x \ (P'_k) \prec y \lor y \ (P_k) \prec x \land x \ (P'_k) \preceq y\)

and \(xRSCFy: \ x \ (\text{MMD Is } P) \preceq y\)

from \(k \) obtain \(k\)

where \(kls: \ k \in Is\)

and \(kcond: \ (x \ (P_k) \approx y \land x \ (P'_k) \prec y) \lor (y \ (P_k) \prec x \land x \ (P'_k) \preceq y)\)

by blast

let \(?xPy = \{ i \in Is. \ x \ (P_i) \prec y \}\)

let \(?xP'y = \{ i \in Is. \ x \ (P'_i) \prec y \}\)

let \(?yPx = \{ i \in Is. \ y \ (P_i) \prec x \}\)

let \(?yP'x = \{ i \in Is. \ y \ (P'_i) \prec x \}\)

from profileP \(xyA \ xly \) have \(yP'x \preceq yPx\)

unfolding strict-pref-def indifferent-pref-def

by (blast dest: rpr-complete)

with finiteIs have \(yP'x \preceq xP'x\)

by (blast intro: card-mono finite-subset)

from finiteIs \(xPy \) have \(xP'x \preceq yP'x\)

by (blast intro: card-mono finite-subset)

show \(x \ (\text{MMD Is } P') \prec y\)

proof

from \(xRSCFy \ xPy \ xP'y \) show \(x \ (\text{MMD Is } P') \preceq y\)

unfolding MMD-def by auto

next
\begin{verbatim}
{
 assume xIky: x (P k) \approx y and xP'ky: x (P' k) \preceq y
 have card ?xPy < card ?xP'y
 proof -
  from xIky have knP: k \notin ?xPy
   unfolding indifferent-pref-def strict-pref-def by blast
  from kIs xP'ky have kP': k \in ?xP'y by simp
  from finiteIs xPy knP kP' show ?thesis
   by (blast intro: psubset-card-mono finite-subset)
 qed
 with xRSCFy yP'xyPxC have card ?yP'x < card ?xP'y
 unfolding MMD-def by auto
}

moreover
{
 assume yPkx: y (P k) \prec x and xR'ky: x (P' k) \preceq y
 have card ?yP'x < card ?yPx
 proof -
  from kIs yPkx have kP: k \in ?yPx by simp
  from kIs xR'ky have knP': k \notin ?yP'x
   unfolding strict-pref-def by blast
  from yP'xyP\preceq yP'x kP knP' have ?yP'x \subset ?yPx by blast
  with finiteIs show ?thesis
   by (blast intro: psubset-card-mono finite-subset)
 qed
 with xRSCFy yP'xyPxC have card ?yP'x < card ?xP'y
 unfolding MMD-def by auto
}

moreover note kcond
 ultimately show \neg (y (MMD Is P') \preceq x)
 unfolding MMD-def by auto
 qed
 qed
 qed

6.3 Everything satisfying May’s conditions is the Method of Majority Decision

Now show that MMD is the only SCF that satisfies these conditions.

Firstly develop some theory about exchanging alternatives x and y in profile P.

definition swapAlts :: 'a \\Rightarrow 'a \\Rightarrow 'a \\Rightarrow 'a where
 swapAlts a b u \equiv if u = a then b else if u = b then a else u

lemma swapAlts-in-set-iff: \{ a, b \} \subseteq A \implies swapAlts a b u \in A \iff u \in A
 unfolding swapAlts-def by (simp split: if-split)

definition swapAltsP :: ('a, 'i) Profile \\Rightarrow 'a \\Rightarrow 'a \\Rightarrow ('a, 'i) Profile where
 swapAltsP P a b \equiv (\lambda i. \{ (u, v) . (swapAlts a b u, swapAlts a b v) \in P i \})

lemma swapAltsP-ab: a (P i) \preceq b \iff b (swapAltsP P a b i) \preceq a (swapAltsP P a b i) \preceq

}\end{verbatim}
lemma profile-swapAltsP:
  assumes profileP: profile A Is P
  and abA: \{a,b\} \subseteq A
  shows profile A Is \(\text{swapAltsP} P a b\)
proof (rule profileI)
  from profileP show Is \(\neq\) \{\} by (rule profile-non-empty)
next
  fix i assume ils: i \in Is
  show rpr A (swapAltsP P a b i)
  proof (rule rprI)
    show refl-on A (swapAltsP P a b i)
    proof (rule refl-onI)
      from profileP ils abA show swapAltsP P a b i \subseteq A \times A
      unfolding swapAltsP-def by (blast dest: swapAlts-in-set-iff)
      from profileP ils abA show \(\forall x. x \in A \Rightarrow (\text{swapAltsP} P a b i) \preceq x\)
      unfolding swapAltsP-def swapAlts-def by auto
    qed
  next
  from profileP ils abA show complete A (swapAltsP P a b i)
  unfolding swapAltsP-def
  by (rule completeI, simp, rule rpr-complete[where \(A=A\)], auto iff: swapAlts-in-set-iff)
  next
  from profileP ils show trans (swapAltsP P a b i)
  unfolding swapAltsP-def by (blast dest: rpr-le-trans intro: transI)
  qed
next

lemma profile-bij-profile:
  assumes profileP: profile A Is P
  and bijf: bij-betw f Is Is
  shows profile A Is \((P \circ f)\)
  using bij-betw-onto[OF bijf] profileP
  by (rule, auto dest: profile-non-empty)

The locale keeps the conditions in scope for the next few lemmas. Note how weak the constraints on the sets of alternatives and individuals are; clearly there needs to be at least two alternatives and two individuals for conflict to occur, but it is pleasant that the proof uniformly handles the degenerate cases.

locale May =
fixes A :: 'a set

fixes Is :: 'i set
assumes finiteIs: finite Is

fixes scf :: ('a, 'i) SCF
assumes SCF: SCF scf A Is universal-domain
  and anonymous: anonymous scf A Is
  and neutral: neutral scf A Is
  and positively-responsive: positively-responsive scf A Is
Anonymity implies that, for any pair of alternatives, the social choice rule can only depend on the number of individuals who express any given preference between them. Note we also need $iia$, implied by neutrality, to restrict attention to alternatives $x$ and $y$.

**lemma** anonymous-card:

**assumes** profile$P$: profile $A$ Is $P$

and profile$P'$: profile $A$ Is $P'$

and $xyA$: hasw $[x,y]$ $A$

and $\text{ytally}$: card $\{i \in Is. x (P i) \prec y\} = \text{card } \{i \in Is. x (P' i) \prec y\}$

and $\text{ytally}$: card $\{i \in Is. x (P i) \prec x\} = \text{card } \{i \in Is. y (P' i) \prec x\}$

**shows** $x (\text{scf } P) \preceq y \iff x (\text{scf } P') \preceq y$

**proof**

\[
\begin{align*}
\text{let } & \ ?xPy = \{i \in Is. x (P i) \prec y\} \\
\text{let } & \ ?xP'y = \{i \in Is. x (P' i) \prec y\} \\
\text{let } & \ ?yPx = \{i \in Is. y (P i) \prec x\} \\
\text{let } & \ ?yP'x = \{i \in Is. y (P' i) \prec x\} \\
\text{have } \text{disjPxy}: (\#xPy \cup \#yPx) \neq \#xPy = \#yPx \\
\text{unfolding } \text{strict-pref-def by blast} \\
\text{have } \text{disjP'xy}: (\#xP'y \cup \#yP'x) \neq \#xP'y = \#yP'x \\
\text{unfolding } \text{strict-pref-def by blast} \\
\text{from } \text{finiteIs yxtally} \\
\text{obtain } f \text{ where bijf: bij-betw } f \?xPy \?xP'y \\
\text{by } (\text{drule card-eq-bij, auto}) \\
\text{from } \text{finiteIs yxtally} \\
\text{obtain } g \text{ where bijg: bij-betw } g \?yPx \?yP'x \\
\text{by } (\text{drule card-eq-bij, auto}) \\
\text{from } bijf bijg \\text{ disjPxy disjP'xy} \\
\text{obtain } h \\
\text{ where } bijh: \\text{bij-betw } h (\#xPy \cup \#yPx) (\#xP'y \cup \#yP'x) \\
\quad \text{and } hf: \forall j. j \in \#xPy \implies h j = f j \\
\quad \text{and } hq: \forall j. j \in (\#xPy \cup \#yPx) \implies \#xPy \implies h j = g j \\
\text{using } \text{bij-combine}[\text{where } f=j \quad \text{and } g=g \quad \text{and } A=\#xPy \quad \text{and } B=\#xPy \cup \#yPx \quad \text{and } C=\#yP'x \quad \text{and } D=\#xP'y \cup \#yP'x] \\
\quad \text{by auto} \\
\text{from } bijh \text{ finiteIs} \\
\text{obtain } h' \text{ where } bijh': \\text{bij-betw } h' \text{ Is } Is \\
\quad \text{and } hh': \forall j. j \in (\#xPy \cup \#yPx) \implies h' j = h j \\
\quad \text{and } hrest: \forall j. j \in Is - (\#xPy \cup \#yPx) \implies h' j \in Is - (\#xP'y \cup \#yP'x) \\
\quad \text{by } (\text{drule bij-complete, auto}) \\
\text{from } \text{neutral-iia}[\text{OF neutral}] \\
\text{have } x (\text{scf } (P' \circ h')) \preceq y \iff x (\text{scf } P) \preceq y \\
\text{proof } \text{(rule iiaE)} \\
\quad \text{from } \text{finite-is } [x, y] \subseteq A \text{ by simp} \\
\text{next} \\
\text{fix } i \text{ assume } iIs: i \in Is \\
\text{fix } a \ b \text{ assume } ab: a \in \{x, y\} \ b \in \{x, y\} \\
\text{from } profileP \ iIs \text{ have } \text{completePi}: \text{complete } A (P i) \text{ by } (\text{auto dest: rprD}) \\
\text{from } \text{completePi } xyA \\
\text{show } (a (P i) \preceq b) \iff (a ((P' \circ h') i) \preceq b) \\
\text{proof } (\text{cases rule: complete-exh})
\end{align*}
\]
case $xPy$ with profile $P$ profile $P'$ $xyA$ iIs ab $hh'$ $hf$ bijf show ?thesis

unfolding strict-pref-def bij-betw-def by (simp, blast)

next
case $yPx$ with profile $P$ profile $P'$ $xyA$ iIs ab $hh'$ $hg$ bijf show ?thesis

unfolding strict-pref-def bij-betw-def by (simp, blast)

next
case $xly$ with profile $P$ profile $P'$ $xyA$ iIs ab $hrst$ where $j=i$ show ?thesis

unfolding indifferent-pref-def strict-pref-def bij-betw-def

by (simp, blast dest: rpr-complete)

qed

qed (simp-all add: profile $P$ profile-bij-profile[OF profile $P'$ bijh'])

moreover

from anonymousD[OF anonymous profile $P'$ bijh'] $xyA$

have $x$ $(\text{scf } P') \preceq y \leftrightarrow x$ $(\text{scf } (P' \circ h')) \preceq y$ by simp

ultimately show ?thesis by simp

qed

Using the previous result and neutrality, it must be the case that if the tallies are tied for alternatives $x$ and $y$ then the social choice function is indifferent between those two alternatives.

lemma anonymous-neutral-indifference:

assumes profile $P$: profile $A$ Is $P$

and $xyA$: basw $[x,y] \ A$

and tally $P$: card $\{i \in \text{Is. } x (P_i) < y \} =$ card $\{i \in \text{Is. } y (P_i) < x \}$

shows $x$ (scf $P$) $\approx y$

proof |

— Neutrality insists the results for $P$ are symmetrical to those for swapAlts $P$.

from $xyA$

have sym $P'$: $(x$ (scf $P') \preceq y \leftrightarrow x$ (scf (swapAlts $P$ $x$ $y$)) $\preceq x)$

$\land (y$ (scf $P') \preceq x \leftrightarrow x$ (scf (swapAlts $P$ $x$ $y$)) $\preceq y)$

by — (rule neutralD[OF neutral profile $P$ profile-swapAlts[OF profile $P'$]],

simp-all, (rule swapAlts-ab)+)

— Anonymity and neutrality insist the results for $P$ are identical to those for swapAlts $P$.

from $xyA$ tally $P$ have card $\{i \in \text{Is. } x (P_i) < y \} =$ card $\{i \in \text{Is. } x (\text{swapAlts } P$ $x$ $y$) $\approx y \}$

and card $\{i \in \text{Is. } y (P_i) < x \} =$ card $\{i \in \text{Is. } y (\text{swapAlts } P$ $x$ $y$) $\approx x \}$

unfolding swapAlts-def swapAlts-def strict-pref-def by simp-all

with profile $P$ $xyA$ have id $P'$: $x$ (scf $P') \preceq y \leftrightarrow x$ (scf (swapAlts $P$ $x$ $y$)) $\preceq y$

and $y$ (scf $P') \preceq x \leftrightarrow y$ (scf (swapAlts $P$ $x$ $y$)) $\preceq x$

by — (rule anonymous-card[OF profile $P$ profile-swapAlts[OF profile $P'$], clarsimp]+)

from $xyA$ SCF-completeD[OF SCF] profile $P$ sym $P'$ id $P'$ show $x$ (scf $P$) $\approx y$ by (simp, blast)

qed

Finally, if the tallies are not equal then the social choice function must lean towards the one with the higher count due to positive responsiveness.

lemma positively-responsive-prefer-witness:

assumes profile $P$: profile $A$ Is $P$

and $xyA$: basw $[x,y] \ A$

and tally $P$: card $\{i \in \text{Is. } x (P_i) < y \} =$ card $\{i \in \text{Is. } y (P_i) < x \}$

obtains $P'$ $k$

where profile $A$ Is $P'$

and $\exists i. [i \in \text{Is. } x (P_i) < y] \Rightarrow x (P_i) < y$

42
and \(\forall i. \{ i \in Is; \ x \ (p \ i) \leq y \} \implies x \ (p \ i) \leq y\)

and \(k \in Is \land x \ (p \ k) \approx y \land x \ (p \ k) < y\)

and \(\text{card} \ \{ i \in Is; \ x \ (p \ i) < y \} = \text{card} \ \{ i \in Is; \ y \ (p \ i)< x \}\)

proof –
from \(\text{tallyP}\) obtain \(C\)
where \(\text{tallyP'}\): \(\text{card} \ \{ i \in Is; \ x \ (p \ i) < y \} = \text{card} \ \{ i \in Is; \ y \ (p \ i)< x \}\)

and \(C: \ C \neq \emptyset \subseteq Is\)
and \(C x P y: \ C \subseteq \{ i \in Is; \ x \ (p \ i) < y \}\)

by – (drule \(\text{card-greater}[OF finiteIs]\), \(auto\))
— \(\text{Add} \ (b, a)\) and close under transitivity.
let \(?P' = \lambda i. \text{if } i \in C \text{ then } P i \cup \{ (y, x) \} \text{ else } P i\)
have \(\text{profile A Is } ?P'\)
proof
fix \(i\) assume \(iIs: i \in Is\)
show \(\text{rpr A } (?P' \ i)\)
proof
from \(\text{profileP iIs}\) show \(\text{complete A } (?P' \ i)\)

unfolding \(\text{complete-def}\) by \((\text{simp, blast dest: rpr-complete})\)

from \(\text{profileP iIs x y A}\) show \(\text{refl-on A } (?P' \ i)\)

by – (rule \(\text{refl-on I, auto}\))
show \(\text{trans } (?P' \ i)\)
proof(cases \(i \in C\))
case \(\text{False}\) with \(\text{profileP iIs}\) show \(\text{thesis}\)
by \((\text{simp, blast dest: rpr-le-trans intro: trans I})\)
next
case \(\text{True}\) with \(\text{profileP iIs C x y A}\) show \(\text{thesis}\)

unfolding \(\text{strict-pref-def}\)
by – (rule \(\text{trans I, simp, blast dest: rpr-le-trans rpr-complete}\))

qed

next
from \(C\) show \(\text{Is } \neq \{\}\) by \(\text{blast}\)

qed

moreover
have \(\forall i. \{ i \in Is; \ x \ (\bar{p} \ i) \approx y \} \implies x \ (p \ i) < y\)

unfolding \(\text{strict-pref-def}\) by \((\text{simp split: if-split-asm})\)

moreover
from \(\text{profileP C x y A}\) have \(\forall i. \{ i \in Is; \ x \ (\bar{p} \ i) \approx y \} \implies x \ (p \ i) \leq y\)

unfolding \(\text{indifferent-pref-def}\) by \((\text{simp split: if-split-asm})\)

moreover
from \(C x P y\) obtain \(k\) where \(kC: k \in C\) and \(x P y: x \ (p \ k) < y\) by \(\text{blast}\)
hence \(x \ (\bar{p} \ k) \approx y\) by \(\text{auto}\)
with \(C k C x P y\) have \(k \in Is \land x \ (\bar{p} \ k) \approx y \land x \ (p \ k) < y\) by \(\text{blast}\)
moreover
have \(\text{card} \ \{ i \in Is; \ x \ (\bar{p} \ i) < y \} = \text{card} \ \{ i \in Is; \ y \ (\bar{p} \ i)< x \}\)
proof
  have \{ i \in Is . x (\trianglerighteq_{P' i}) y \} = \{ i \in Is . x (\trianglerighteq_{P' i}) y \} - C
proof
  from C have \{ i . i \in Is ; x (\trianglerighteq_{P' i}) y \} \implies i \in Is - C
  unfolding indifferent-pref-def strict-pref-def by auto
thus \thesis by blast
qed
also have \ldots = \{ i \in Is . x (P i) y \} - C by auto
finally have card \{ i \in Is . x (\trianglerighteq_{P' i}) y \} = card \{ i \in Is . x (P i) y \} - C
by simp
with tallyP' have card \{ i \in Is . x (\trianglerighteq_{P' i}) y \} = card \{ i \in Is . y (P i) x \}
by simp
also have \ldots = card \{ i \in Is . y (\trianglerighteq_{P' i}) x \} (is card ?lhs = card ?rhs)
proof
  from profileP xyA have \{ i . i \in Is ; y (\trianglerighteq_{P' i}) x \} \implies y (P i) x
  unfolding strict-pref-def by (simp split: if-split-asm, blast dest: rpr-complete)
hence ?rhs \subseteq ?lhs by blast
moreover
from profileP xyA have \{ i . i \in Is ; y (\trianglerighteq_{P' i}) x \} \implies y (\trianglerighteq_{P' i}) x
  unfolding strict-pref-def by simp
hence ?lhs \subseteq ?rhs by blast
ultimately show \thesis by simp
qed
finally show \thesis .
qed
ultimately show \thesis ..
qed

lemma positively-responsive-prefer:
  assumes profileP: profile A Is P
    and xyA: basw [x,y] A
    and tallyP: card \{ i . i \in Is . x (P i) y \} > card \{ i . i \in Is . y (P i) x \}
shows x (scf P) y
proof
  from assms obtain P' k
    where profileP': profile A Is P'
      and F: \{ i . i \in Is ; x (P' i) y \} \implies x (P i) y
      and G: \{ i . i \in Is ; x (P' i) y \} \implies x (P i) y
      and pivot: k \in Is \land x (P' k) y \land x (P k) y
      and cardP': card \{ i . i \in Is . x (P' i) y \} = card \{ i . i \in Is . y (P' i) x \}
    by - (drule positively-responsive-prefer-witness, auto)
from profileP' xyA cardP' have x (scf P') y
    by - (rule anonymous-neutral-indifference, auto)
with xyA F G pivot show \thesis
  by - (rule positively-responsiveD[OF positively-responsive profileP' profileP], auto)
qed

lemma MMD-r2l:
  assumes profileP: profile A Is P
    and xyA: basw [x,y] A
shows \( x (\text{scf} P)\preceq y \iff (\text{MMD Is} P)\preceq y \)

**proof** (cases rule: linorder-cases)

**assume** \( \text{card} \{ i \in \text{Is}. \ (P i)\prec y \} = \text{card} \{ i \in \text{Is}. \ (P i)\prec x \} \)

**with** profile\( P \text{ xyA} \) show \( ?\text{thesis} \)

**using** anonymous-neutral-indifference

**unfolding** indifferent-pref-def MMD-def by simp

**next**

**assume** \( \text{card} \{ i \in \text{Is}. \ (P i)\prec y \} > \text{card} \{ i \in \text{Is}. \ (P i)\prec x \} \)

**with** profile\( P \text{ xyA} \) show \( ?\text{thesis} \)

**using** positively-responsive-prefer

**unfolding** strict-pref-def MMD-def by simp

**next**

**assume** \( \text{card} \{ i \in \text{Is}. \ (P i)\prec y \} < \text{card} \{ i \in \text{Is}. \ (P i)\prec x \} \)

**with** profile\( P \text{ xyA} \) show \( ?\text{thesis} \)

**using** positively-responsive-prefer

**unfolding** strict-pref-def MMD-def by clarsimp

qed end

May’s original paper [May52] goes on to show that the conditions are independent by exhibiting choice rules that differ from MMD and satisfy the conditions remaining after any particular one is removed. I leave this to future work.

May also wrote a later article [May53] where he shows that the conditions are completely independent, i.e. for every partition of the conditions into two sets, there is a voting rule that satisfies one and not the other.

There are many later papers that characterise MMD with different sets of conditions.

### 6.4 The Plurality Rule

Goodin and List [GL06] show that May’s original result can be generalised to characterise plurality voting. The following shows that this result is a short step from Sen’s much earlier generalisation.

**Plurality voting** is a choice function that returns the alternative that receives the most votes, or the set of such alternatives in the case of a tie. Profiles are restricted to those where each individual casts a vote in favour of a single alternative.

**type-synonym** (‘a, ‘i) SVProfile = ‘i ⇒ ‘a

**definition** s征程ile :: ‘a set ⇒ ‘i set ⇒ (‘a, ‘i) SVProfile ⇒ bool where s征程ile A Is F ∋ Is ≠ {} ∧ F ‘Is ⊆ A

**definition** plurality-rule :: ‘a set ⇒ ‘i set ⇒ (‘a, ‘i) SVProfile ⇒ ‘a set where plurality-rule A Is F

\[ \equiv \{ \ x \in A . \ \forall y \in A. \ \text{card} \{ i \in \text{Is}. \ F i = x \} \geq \text{card} \{ i \in \text{Is}. \ F i = y \} \} \]

By translating single-vote profiles into RPRs in the obvious way, the choice function arising from MMD coincides with traditional plurality voting.

**definition** MMD-plurality-rule :: ‘a set ⇒ ‘i set ⇒ (‘a, ‘i) Profile ⇒ ‘a set where

MMD-plurality-rule A Is P ∋ choiceSet A (MMD Is P)

45
definition `single-vote-to-RPR :: 'a set ⇒ 'a ⇒ 'a RPR where`
`single-vote-to-RPR A a ≡ { (a, x) | x ∈ A } ∪ (A − {a}) × (A − {a})`  

lemma `single-vote-to-RPR-iff`:
`[ a ∈ A; x ∈ A; a ≠ x ] ⇒ (a (single-vote-to-RPR A b) ≺ x) ⇔ (b = a)`  

unfolding `single-vote-to-RPR-def strict-pref-def` by auto  

lemma `plurality-rule-equiv`:
`plurality-rule A Is F = MMD-plurality-rule A Is (single-vote-to-RPR A ◦ F)`  

proof `{  
  fix x y  
  have [ x ∈ A; y ∈ A ] ⇒  
    (card { i ∈ Is. F i = y } ≤ card { i ∈ Is. F i = x }) =  
    (card { i ∈ Is. y (single-vote-to-RPR A (F i)) ≺ x })  
    ≤ card { i ∈ Is. x (single-vote-to-RPR A (F i)) ≺ y })  
    by (cases x=y, auto iff: single-vote-to-RPR-iff)  
  }  
  thus `thesis`  
  unfolding `plurality-rule-def MMD-plurality-rule-def choiceSet-def MMD-def`  
  by auto  
qed  

Thus it is clear that Sen’s generalisation of May’s result applies to this case as well.

Their paper goes on to show how strengthening the anonymity condition gives rise to a characterisation of approval voting that strictly generalises May’s original theorem. As this requires some rearrangement of the proof I leave it to future work.

7 Bibliography

References


