Arrow's General Possibility Theorem

Peter Gammie
`peteg42 at gmail.com`

June 16, 2019

Contents

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 naive. 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*: $\bigwedge a\ F.\ [\![\mathit{finite}\ F;\ a \in A;\ F \subseteq A;\ a \notin F;\ P\ F\ ]\!] \Longrightarrow P\ (\mathit{insert}\ a\ F)$
  shows *P F*
proof −
  from ‹*finite F*›
  have $F \subseteq A \Longrightarrow$ *?thesis*
  proof *induct*
    show *P* {} **by** *fact*
  next
    fix *x F*
    assume *finite F* **and** $x \notin F$ **and**
      P: $F \subseteq A \Longrightarrow P\ F$ **and** *i*: *insert x F* $\subseteq A$
    show *P* (*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
  qed
  with ‹$F \subseteq A$› **show** *?thesis* **by** *blast*
qed

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

lemma *finite-list*: *finite A* $\Longrightarrow \exists\, l.$ *set l* $= A \wedge$ *distinct l*
proof(*induct rule*: *finite-induct*)
  case (*insert x F*)
  then obtain *l* **where** *set l* $= F \wedge$ *distinct l* **by** *auto*
  with *insert* **have** *set* $(x\#l) =$ *insert x F* $\wedge$ *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* $\Longrightarrow$ *f* ‘ $A = B$ **unfolding** *bij-betw-def* **by** *simp*

lemma *inj-on-UnI*: $[\![$ *inj-on f A*; *inj-on f B*; f ‘ $(A - B) \cap f$ ‘ $(B - A) = \{\}$ $]\!]$ $\Longrightarrow$ *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 $-$
from *finiteA* **obtain** *g* **where** G: *bij-betw g A* $\{0..<card\ A\}$
by (*blast dest*: *ex-bij-betw-finite-nat*)
from *finiteB* **obtain** *h* **where** H: *bij-betw h* $\{0..<card\ B\}$ *B*
by (*blast dest*: *ex-bij-betw-nat-finite*)
from *G H cardAB* **have** I: *inj-on* $(h \circ g)$ *A*
unfolding *bij-betw-def* **by** $-$ (*rule comp-inj-on*, *simp-all*)
from *G H cardAB* **have** $(h \circ g)$ ‘ $A = B$
unfolding *bij-betw-def* **by** *auto* (*metis image-cong image-image*)
with I **have** *bij-betw* $(h \circ g)$ *A B*
unfolding *bij-betw-def* **by** *blast*
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 $\bigwedge x.\ x \in A \Longrightarrow h\ x = f\ x$
and $\bigwedge x.\ x \in B - A \Longrightarrow h\ x = g\ x$
proof $-$
let *?h* = $\lambda x.$ *if* $x \in A$ *then* $f\ x$ *else* $g\ x$
have *inj-on ?h* $(A \cup (B - A))$
proof(*rule inj-on-UnI*)
from *bijf* **show** *inj-on ?h A*
by $-$ (*rule inj-onI*, *auto dest*: *inj-onD bij-betw-imp-inj-on*)

from *bijg* **show** *inj-on ?h* $(B - A)$
by $-$ (*rule inj-onI*, *auto dest*: *inj-onD bij-betw-imp-inj-on*)
from *bijf bijg* **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* ‘ $B = D$
proof $-$
from *ABCD* **have** *?h* ‘ $B = f$ ‘ $A \cup g$ ‘ $(B - A)$ **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 $\bigwedge x.\ x \in A \Longrightarrow$ *?h* $x = f\ x$
and $\bigwedge x.\ x \in B - A \Longrightarrow$ *?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 $\bigwedge x.\ x \in A \Longrightarrow f'\ x = f\ x$
and $\bigwedge x.\ x \in C - A \Longrightarrow f'\ x \in C - B$
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*: $\bigwedge x.\ x \in A \Longrightarrow f'\ x = f\ x$
and *f'g*: $\bigwedge x.\ x \in C - A \Longrightarrow f'\ x = g\ x$
by $-$ (*drule bij-combine*, *auto*)
from *f'g bijg* **have** $\bigwedge x.\ x \in C - A \Longrightarrow f'\ x \in C - B$
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* ∈ *A* . *P x* }
let *?QA* = { *x* ∈ *A* . *Q x* }
from *finiteA* **obtain** *p* **where** *P*: *bij-betw p* {*0*..<*card ?PA*} *?PA*
using *ex-bij-betw-nat-finite*[**where** *M*=*?PA*]
by (*blast intro*: *finite-subset*)
let *?CN* = {*card ?QA*..<*card ?PA*}
let *?C* = *p* ' *?CN*
have *card* ({ *x* ∈ *A*. *P x* } − *?C*) = *card ?QA*
proof −
have *nat-add-sub-shuffle*: ⋀*x y z*. ⟦ (*x*::*nat*) > *y*; *x* − *y* = *z* ⟧ ⟹ *x* − *z* = *y* **by** *simp*
from *P* **have** *T*: *p* ' {*card ?QA*..<*card ?PA*} ⊆ *?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* ≠ {}
unfolding *bij-betw-def* **by** *auto*
moreover
from *P* **have** *?C* ⊆ { *x* ∈ *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 *card* function unfortunately satisfies:

Finite-Set.card-infinite: *infinite A* ⟹ *card A* = *0*

These lemmas handle the infinite case uniformly.

Thanks to Gerwin Klein suggesting this approach.

definition *hasw* :: *'a list* ⇒ *'a set* ⇒ *bool* **where**
hasw xs S ≡ *set xs* ⊆ *S* ∧ *distinct xs*

definition *has* :: *nat* ⇒ *'a set* ⇒ *bool* **where**
has n S ≡ ∃ *xs*. *hasw xs S* ∧ *length xs* = *n*

declare *hasw-def*[*simp*]

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

lemma *card-has*:
assumes *cardS*: *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'
```

```
    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 Sxxs: set (x # xs) = S
    hence finiteS: finite S by auto
    from h obtain xs′
      where Sxs′: 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 Sxs′ card-mono have card S ≥ n by auto
    moreover
    from Sxxs Cons card-length[where xs=x # xs]
    have card S < n by auto
```

```
    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 ⟧ ⟹ ∃ x. hasw (x # xs) S
  by (simp, blast dest: has-extend-witness)

lemma has-witness-two:
  assumes hasnS: has n S
      and nn': 2 ≤ n
  shows ∃ 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 ∈ 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 ≤ n
  shows ∃ 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 ≠ {}
      and cardS: card S > 1 ⟹ False
  shows ∃ 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} ⊆ 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 population *Is*.

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 $'a\ RPR = ('a * 'a)\ set$

abbreviation *rpr-eq-syntax* :: $'a \Rightarrow 'a\ RPR \Rightarrow 'a \Rightarrow bool$ (- -$\preceq$ - [50, 1000, 51] 50) **where**
$x\ {}_r\preceq\ y == (x, y) \in r$

definition *indifferent-pref* :: $'a \Rightarrow 'a\ RPR \Rightarrow 'a \Rightarrow bool$ (- -$\approx$ - [50, 1000, 51] 50) **where**
$x\ {}_r\approx\ y \equiv (x\ {}_r\preceq\ y \wedge y\ {}_r\preceq\ x)$

lemma *indifferent-prefI*[*intro*]: $[\![\ x\ {}_r\preceq\ y;\ y\ {}_r\preceq\ x\ ]\!] \Longrightarrow x\ {}_r\approx\ y$
unfolding *indifferent-pref-def* **by** *simp*

lemma *indifferent-prefD*[*dest*]: $x\ {}_r\approx\ y \Longrightarrow x\ {}_r\preceq\ y \wedge y\ {}_r\preceq\ x$
unfolding *indifferent-pref-def* **by** *simp*

definition *strict-pref* :: $'a \Rightarrow 'a\ RPR \Rightarrow 'a \Rightarrow bool$ (- -$\prec$ - [50, 1000, 51] 50) **where**
$x\ {}_r\prec\ y \equiv (x\ {}_r\preceq\ y \wedge \neg(y\ {}_r\preceq\ x))$

lemma *strict-pref-def-irrefl*[*simp*]: $\neg\ (x\ {}_r\prec\ x)$ **unfolding** *strict-pref-def* **by** *blast*

lemma *strict-prefI*[*intro*]: $[\![\ x\ {}_r\preceq\ y;\ \neg(y\ {}_r\preceq\ x)\ ]\!] \Longrightarrow x\ {}_r\prec\ y$
unfolding *strict-pref-def* **by** *simp*

Traditionally, $x\ {}_r\preceq\ y$ would be written $x\ R\ y$, $x\ {}_r\approx\ y$ as $x\ I\ y$ and $x\ {}_r\prec\ y$ 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* :: $'a\ set \Rightarrow 'a\ RPR \Rightarrow bool$ **where**
$complete\ A\ r \equiv (\forall x \in A.\ \forall y \in A - \{x\}.\ x\ {}_r\preceq\ y \vee y\ {}_r\preceq\ x)$

lemma *completeI*[*intro*]:
$(\bigwedge x\ y.\ [\![\ x \in A;\ y \in A;\ x \neq y\ ]\!] \Longrightarrow x\ {}_r\preceq\ y \vee y\ {}_r\preceq\ x) \Longrightarrow complete\ A\ r$
unfolding *complete-def* **by** *auto*

lemma *completeD*[*dest*]:
$[\![\ complete\ A\ r;\ x \in A;\ y \in A;\ x \neq y\ ]\!] \Longrightarrow x\ {}_r\preceq\ y \vee y\ {}_r\preceq\ x$
unfolding *complete-def* **by** *auto*

lemma *complete-less-not*: $[\![\ complete\ A\ r;\ hasw\ [x,y]\ A;\ \neg\ x\ {}_r\prec\ y\ ]\!] \Longrightarrow y\ {}_r\preceq\ x$
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*) *x* $_r$≺ *y*
| (*yPx*) *y* $_r$≺ *x*
| (*xIy*) *x* $_r$≈ *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*

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*

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

definition *rpr* :: ′*a set* ⇒ ′*a RPR* ⇒ *bool* **where**
rpr A r ≡ *complete A r* ∧ *refl-on A r* ∧ *trans r*

lemma *rprI*[*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*]: $[\![x\ {}_r{\prec}\ y]\!] \Longrightarrow x \neq y$
unfolding *strict-pref-def* **by** *blast*

lemma *rpr-less-trans*[*trans*]: $[\![x\ {}_r{\prec}\ y;\ y\ {}_r{\prec}\ z;\ rpr\ A\ r]\!] \Longrightarrow x\ {}_r{\prec}\ z$
unfolding *rpr-def strict-pref-def trans-def* **by** *blast*

lemma *rpr-le-trans*[*trans*]: $[\![x\ {}_r{\preceq}\ y;\ y\ {}_r{\preceq}\ z;\ rpr\ A\ r]\!] \Longrightarrow x\ {}_r{\preceq}\ z$
unfolding *rpr-def trans-def* **by** *blast*

lemma *rpr-le-less-trans*[*trans*]: $[\![x\ {}_r{\preceq}\ y;\ y\ {}_r{\prec}\ z;\ rpr\ A\ r]\!] \Longrightarrow x\ {}_r{\prec}\ z$
unfolding *rpr-def strict-pref-def trans-def* **by** *blast*

lemma *rpr-less-le-trans*[*trans*]: $[\![x\ {}_r{\prec}\ y;\ y\ {}_r{\preceq}\ z;\ rpr\ A\ r]\!] \Longrightarrow x\ {}_r{\prec}\ z$
unfolding *rpr-def strict-pref-def trans-def* **by** *blast*

lemma *rpr-complete*: $[\![rpr\ A\ r;\ x \in A;\ y \in A]\!] \Longrightarrow x\ {}_r{\preceq}\ y \lor y\ {}_r{\preceq}\ x$
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*]: $[\![\bigwedge i.\ i \in Is \Longrightarrow rpr\ A\ (P\ i);\ Is \neq \{\}]\!] \Longrightarrow profile\ A\ Is\ P$
unfolding *profile-def* **by** *simp*

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

lemma *profile-non-empty*: $profile\ A\ Is\ P \Longrightarrow Is \neq \{\}$
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**
$choiceSet\ A\ r \equiv \{\ x \in A\ .\ \forall\, y \in A.\ x\ {}_r{\preceq}\ y\ \}$

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*]:
$[\![\ x \in A;\ \bigwedge y.\ y \in A \Longrightarrow x\ {}_{r}\!\preceq\ y\ ]\!] \Longrightarrow x \in$ *choiceSet A r*
unfolding *choiceSet-def* **by** *simp*

lemma *choiceFnI*[*intro*]:
$(\bigwedge A'.\ [\![\ A' \subseteq A;\ A' \neq \{\}\ ]\!] \Longrightarrow$ *choiceSet* $A'\ r \neq \{\}) \Longrightarrow$ *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* ⇒ *bool* **where**
quasi-trans r $\equiv \forall x\ y\ z.\ x\ {}_{r}\!\prec\ y \wedge y\ {}_{r}\!\prec\ z \longrightarrow x\ {}_{r}\!\prec\ z$

lemma *quasi-transI*[*intro*]:
$(\bigwedge x\ y\ z.\ [\![\ x\ {}_{r}\!\prec\ y;\ y\ {}_{r}\!\prec\ z\ ]\!] \Longrightarrow x\ {}_{r}\!\prec\ z) \Longrightarrow$ *quasi-trans r*
unfolding *quasi-trans-def* **by** *blast*

lemma *quasi-transD*: $[\![\ x\ {}_{r}\!\prec\ y;\ y\ {}_{r}\!\prec\ z;$ *quasi-trans r* $]\!] \Longrightarrow x\ {}_{r}\!\prec\ z$
unfolding *quasi-trans-def* **by** *blast*

lemma *trans-imp-quasi-trans*: *trans r* ⟹ *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 *choiceFn A r*
proof
fix *B* **assume** *B*: $B \subseteq A\ B \neq \{\}$
with *finite-subset finiteA* **have** *finiteB*: *finite B* **by** *auto*
from *finiteB B* **show** *choiceSet B r* $\neq \{\}$
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 *AB*: $B \subseteq A$
and *aB*: $a \notin B$
and *cF*: $B \neq \{\} \Longrightarrow$ *choiceSet B r* $\neq \{\}$ **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*: $b \in$ *choiceSet B r* **by** *blast*
from *AB aA bCF complete-refl-on*[*OF c r*]
have $a\ {}_{r}\!\prec\ b \vee b\ {}_{r}\!\preceq\ a$ **unfolding** *choiceSet-def strict-pref-def* **by** *blast*
thus *?thesis*
proof
assume *ab*: $b\ {}_{r}\!\preceq\ a$

```
      with bCF show ?thesis unfolding choiceSet-def by auto
    next
      assume ab: a r≺ b
      have a ∈ choiceSet (insert a B) r
      proof(rule ccontr)
        assume aCF: a ∉ choiceSet (insert a B) r
        from aB have ⋀b. b ∈ B ⟹ a ≠ b by auto
        with aCF aA AB c r obtain b' where B: b' ∈ B b' r≺ a
          unfolding choiceSet-def complete-def strict-pref-def by blast
        with ab qt have b' r≺ b by (blast dest: quasi-transD)
        with bCF B show False unfolding choiceSet-def strict-pref-def by blast
      qed
      thus ?thesis by auto
    qed
   qed
  qed
qed
```

```
lemma rpr-choiceFn: ⟦ finite A; rpr A r ⟧ ⟹ choiceFn A r
  unfolding 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.

```
type-synonym ('a, 'i) SCF = ('a, 'i) Profile ⇒ 'a RPR
```

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
  SCF :: ('a, 'i) SCF ⇒ 'a set ⇒ 'i set ⇒ ('a set ⇒ 'i set ⇒ ('a, 'i) Profile ⇒ bool) ⇒ bool
where
  SCF scf A Is Pcond ≡ (∀ P. Pcond A Is P ⟶ (complete A (scf P)))
```

```
lemma SCFI[intro]:
  assumes c: ⋀P. Pcond A Is P ⟹ complete A (scf P)
  shows SCF scf A Is Pcond
  unfolding SCF-def using assms by blast
```

```
lemma SCF-completeD[dest]: ⟦ SCF scf A Is Pcond; Pcond A Is P ⟧ ⟹ complete A (scf P)
  unfolding 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
  SWF :: ('a, 'i) SCF ⇒ 'a set ⇒ 'i set ⇒ ('a set ⇒ 'i set ⇒ ('a, 'i) Profile ⇒ bool) ⇒ bool
where
```

SWF swf A Is Pcond ≡ (∀ *P*. *Pcond A Is P* ⟶ *rpr A* (*swf P*))

lemma *SWF-rpr*[*dest*]: ⟦ *SWF swf A Is Pcond*; *Pcond A Is P* ⟧ ⟹ *rpr A* (*swf P*)
unfolding *SWF-def* **by** *simp*

3.6 General Properties of an SCF

An SCF has a *universal domain* if it works for all profiles.

definition *universal-domain* :: *'a set* ⇒ *'i set* ⇒ (*'a*, *'i*) *Profile* ⇒ *bool* **where**
universal-domain A Is P ≡ *profile A Is P*

declare *universal-domain-def*[*simp*]

An SCF is *weakly Pareto-optimal* if, whenever everyone strictly prefers x to y, the SCF does too.

definition
weak-pareto :: (*'a*, *'i*) *SCF* ⇒ *'a set* ⇒ *'i set* ⇒ (*'a set* ⇒ *'i set* ⇒ (*'a*, *'i*) *Profile* ⇒ *bool*) ⇒ *bool*
where
weak-pareto scf A Is Pcond ≡
(∀ *P x y*. *Pcond A Is P* ∧ *x* ∈ *A* ∧ *y* ∈ *A* ∧ (∀ *i* ∈ *Is*. $x\ _{(P\ i)}{\prec}\ y$) ⟶ $x\ _{(scf\ P)}{\prec}\ y$)

lemma *weak-paretoI*[*intro*]:
(⋀*P x y*. ⟦*Pcond A Is P*; *x* ∈ *A*; *y* ∈ *A*; ⋀*i*. *i*∈*Is* ⟹ $x\ _{(P\ i)}{\prec}\ y$⟧ ⟹ $x\ _{(scf\ P)}{\prec}\ y$)
⟹ *weak-pareto scf A Is Pcond*
unfolding *weak-pareto-def* **by** *simp*

lemma *weak-paretoD*:
⟦ *weak-pareto scf A Is Pcond*; *Pcond A Is P*; *x* ∈ *A*; *y* ∈ *A*;
(⋀*i*. *i* ∈ *Is* ⟹ $x\ _{(P\ i)}{\prec}\ y$) ⟧ ⟹ $x\ _{(scf\ P)}{\prec}\ y$
unfolding *weak-pareto-def* **by** *simp*

An SCF satisfies *independence of irrelevant alternatives* if, for two preference profiles P and P' where for all individuals i, alternatives x and y drawn from set S have the same order in $P\ i$ and $P'\ i$, then alternatives x and y have the same order in *scf P* and *scf P'*.

definition *iia* :: (*'a*, *'i*) *SCF* ⇒ *'a set* ⇒ *'i set* ⇒ *bool* **where**
iia scf S Is ≡
(∀ *P P' x y*. *profile S Is P* ∧ *profile S Is P'*
∧ *x* ∈ *S* ∧ *y* ∈ *S*
∧ (∀ *i* ∈ *Is*. (($x\ _{(P\ i)}{\preceq}\ y$) ⟷ ($x\ _{(P'\ i)}{\preceq}\ y$)) ∧ (($y\ _{(P\ i)}{\preceq}\ x$) ⟷ ($y\ _{(P'\ i)}{\preceq}\ x$)))
⟶ (($x\ _{(scf\ P)}{\preceq}\ y$) ⟷ ($x\ _{(scf\ P')}{\preceq}\ y$)))

lemma *iiaI*[*intro*]:
(⋀*P P' x y*.
⟦ *profile S Is P*; *profile S Is P'*;
x ∈ *S*; *y* ∈ *S*;
⋀*i*. *i* ∈ *Is* ⟹ (($x\ _{(P\ i)}{\preceq}\ y$) ⟷ ($x\ _{(P'\ i)}{\preceq}\ y$)) ∧ (($y\ _{(P\ i)}{\preceq}\ x$) ⟷ ($y\ _{(P'\ i)}{\preceq}\ x$))
⟧ ⟹ (($x\ _{(swf\ P)}{\preceq}\ y$) ⟷ ($x\ _{(swf\ P')}{\preceq}\ y$)))
⟹ *iia swf S Is*
unfolding *iia-def* **by** *simp*

lemma *iiaE*:

$\llbracket$ *iia swf S Is*;
$\{x,y\} \subseteq S$;
$a \in \{x, y\}$; $b \in \{x, y\}$;
$\bigwedge i\ a\ b.\ \llbracket\ a \in \{x, y\};\ b \in \{x, y\};\ i \in Is\ \rrbracket \Longrightarrow (a\ _{(P'\ i)}\preceq\ b) \longleftrightarrow (a\ _{(P\ i)}\preceq\ b)$;
profile S Is P; *profile S Is P′* $\rrbracket$
$\Longrightarrow (a\ _{(swf\ P)}\preceq\ b) \longleftrightarrow (a\ _{(swf\ P')}\preceq\ b)$
unfolding *iia-def* **by** (*simp*, *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* :: $('a, 'i)\ SCF \Rightarrow 'a\ set \Rightarrow 'i\ set \Rightarrow 'i\ set \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$ **where**
semidecisive scf A Is C x y $\equiv$
$C \subseteq Is \wedge (\forall P.\ profile\ A\ Is\ P \wedge (\forall i \in C.\ x\ _{(P\ i)}\prec\ y) \wedge (\forall i \in Is - C.\ y\ _{(P\ i)}\prec\ x)$
$\longrightarrow x\ _{(scf\ P)}\prec\ y)$

lemma *semidecisiveI*[*intro*]:
$\llbracket\ C \subseteq Is$;
$\bigwedge P.\ \llbracket\ profile\ A\ Is\ P;\ \bigwedge i.\ i \in C \Longrightarrow x\ _{(P\ i)}\prec\ y;\ \bigwedge i.\ i \in Is - C \Longrightarrow y\ _{(P\ i)}\prec\ x\ \rrbracket$
$\Longrightarrow x\ _{(scf\ P)}\prec\ y\ \rrbracket \Longrightarrow$ *semidecisive scf A Is C x y*
unfolding *semidecisive-def* **by** *simp*

lemma *semidecisive-coalitionD*[*dest*]: *semidecisive scf A Is C x y* $\Longrightarrow C \subseteq Is$
unfolding *semidecisive-def* **by** *simp*

lemma *sd-refl*: $\llbracket\ C \subseteq Is;\ C \neq \{\}\ \rrbracket \Longrightarrow$ *semidecisive scf A Is C x x*
unfolding *semidecisive-def strict-pref-def* **by** *blast*

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

definition *decisive* :: $('a, 'i)\ SCF \Rightarrow 'a\ set \Rightarrow 'i\ set \Rightarrow 'i\ set \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$ **where**
decisive scf A Is C x y $\equiv$
$C \subseteq Is \wedge (\forall P.\ profile\ A\ Is\ P \wedge (\forall i \in C.\ x\ _{(P\ i)}\prec\ y) \longrightarrow x\ _{(scf\ P)}\prec\ y)$

lemma *decisiveI*[*intro*]:
$\llbracket\ C \subseteq Is;\ \bigwedge P.\ \llbracket\ profile\ A\ Is\ P;\ \bigwedge i.\ i \in C \Longrightarrow x\ _{(P\ i)}\prec\ y\ \rrbracket \Longrightarrow x\ _{(scf\ P)}\prec\ y\ \rrbracket$
$\Longrightarrow$ *decisive scf A Is C x y*
unfolding *decisive-def* **by** *simp*

lemma *d-imp-sd*: *decisive scf A Is C x y* $\Longrightarrow$ *semidecisive scf A Is C x y*
unfolding *decisive-def* **by** (*rule semidecisiveI*, *blast+*)

lemma *decisive-coalitionD*[*dest*]: *decisive scf A Is C x y* $\Longrightarrow C \subseteq Is$
unfolding *decisive-def* **by** *simp*

Anyone is trivially decisive for x against x.

lemma *d-refl*: $\llbracket\ C \subseteq Is;\ C \neq \{\}\ \rrbracket \Longrightarrow$ *decisive scf A Is C x x*

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

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 $dictator :: ('a, 'i)\ SCF \Rightarrow 'a\ set \Rightarrow 'i\ set \Rightarrow 'i \Rightarrow bool$ **where**
$dictator\ scf\ A\ Is\ j \equiv j \in Is \wedge (\forall x \in A.\ \forall y \in A.\ decisive\ scf\ A\ Is\ \{j\}\ x\ y)$

lemma *dictatorI*[*intro*]:
$[\![j \in Is; \bigwedge x\ y.\ [\![x \in A; y \in A]\!] \Longrightarrow decisive\ scf\ A\ Is\ \{j\}\ x\ y]\!] \Longrightarrow dictator\ scf\ A\ Is\ j$
unfolding *dictator-def* **by** *simp*

lemma *dictator-individual*[*dest*]: $dictator\ scf\ A\ Is\ j \Longrightarrow j \in Is$
unfolding *dictator-def* **by** *simp*

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*: $j \in Is$
obtains P'
where $profile\ A\ Is\ P'$
and $x\ _{(P'\ j)}{\prec}\ y \wedge y\ _{(P'\ j)}{\prec}\ z$
and $\bigwedge i.\ i \neq j \Longrightarrow y\ _{(P'\ i)}{\prec}\ x \wedge y\ _{(P'\ i)}{\prec}\ z \wedge ((x\ _{(P'\ i)}{\preceq}\ z) = (x\ _{(P\ i)}{\preceq}\ z)) \wedge ((z\ _{(P'\ i)}{\preceq}\ x) = (z\ _{(P\ i)}{\preceq}\ x))$
proof
let $?P' = \lambda i.\ (if\ i = j\ then\ (\{\ (x,\ u)\ |\ u.\ u \in A\ \}$
$\cup \{\ (y,\ u)\ |\ u.\ u \in A - \{x\}\ \}$

```
                     ∪ { (z, u) | u. u ∈ A − {x,y} })
                  else ({ (y, u) | u. u ∈ A }
                     ∪ { (x, u) | u. u ∈ A − {y,z} }
                     ∪ { (z, u) | u. u ∈ A − {x,y} }
                     ∪ (if x (P i)≼ z then {(x,z)} else {})
                     ∪ (if z (P i)≼ x then {(z,x)} else {})))
              ∪ (A − {x,y,z}) × (A − {x,y,z})
  show profile A Is ?P′
  proof
    fix i assume iIs: i ∈ Is
    show rpr A (?P′ i)
    proof(cases i = j)
      case True with has3A show ?thesis
        by − (rule rprI, simp-all add: trans-def, blast+)
    next
      case False hence ij: i ≠ j .
      show ?thesis
      proof
        from iIs profileP have complete A (P i) by (blast dest: rpr-complete)
        with ij show complete A (?P′ i) by (simp add: complete-def, blast)
        from iIs profileP have refl-on A (P i) by (auto simp add: rpr-def)
        with has3A ij show refl-on A (?P′ i) by (simp, blast)
        from ij has3A show trans (?P′ i) by (clarsimp simp add: trans-def)
      qed
    qed
  next
    from profileP show Is ≠ {} by (rule profile-non-empty)
  qed
  from has3A
  show 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))
    unfolding strict-pref-def by auto
qed
```

The key lemma: in the presence of Arrow's assumptions, an individual who is semi-decisive for x and y is actually decisive for x over any other alternative z. (This is where the quantification becomes important.)

```
lemma decisive1:
  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} x z
proof
  from sd show jIs: {j} ⊆ Is by blast
  fix P
  assume profileP: profile A Is P
   and jxzP: ⋀i. i ∈ {j} ⟹ x (P i)≺ z
  from has3A profileP jIs
  obtain P′
```

where *profileP′*: *profile A Is P′*
and *jxyzP′*: $x\ _{(P'\ j)}\!\prec\ y\ \ y\ _{(P'\ j)}\!\prec\ z$
and *ixyzP′*: $\bigwedge i.\ i \neq j \longrightarrow y\ _{(P'\ i)}\!\prec\ x \wedge y\ _{(P'\ i)}\!\prec\ z \wedge ((x\ _{(P'\ i)}\!\preceq\ z) = (x\ _{(P\ i)}\!\preceq\ z)) \wedge ((z\ _{(P'\ i)}\!\preceq\ x) = (z\ _{(P\ i)}\!\preceq\ x))$
by − (*rule decisive1-witness*, *blast*+)
from *iia* **have** $\bigwedge a\ b.\ [\![\ a \in \{x,\ z\};\ b \in \{x,\ z\}\ ]\!] \Longrightarrow (a\ _{(swf\ P)}\!\preceq\ b) = (a\ _{(swf\ P')}\!\preceq\ b)$
proof(*rule iiaE*)
from *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 $(a\ _{(P'\ i)}\!\preceq\ b) = (a\ _{(P\ i)}\!\preceq\ b)$
proof(*cases* $i = j$)
case *False*
with *ab iIs ixyzP′ profileP profileP′ has3A*
show *?thesis* **unfolding** *profile-def* **by** *auto*
next
case *True*
from *profileP′ jIs jxyzP′* **have** $x\ _{(P'\ j)}\!\prec\ z$
by (*auto dest*: *rpr-less-trans*)
with *True ab iIs jxzP profileP profileP′ has3A*
show *?thesis* **unfolding** *profile-def strict-pref-def* **by** *auto*
qed
qed (*simp-all add*: *profileP profileP′*)
moreover have $x\ _{(swf\ P')}\!\prec\ z$
proof −
from *profileP′ sd jxyzP′ ixyzP′* **have** $x\ _{(swf\ P')}\!\prec\ y$ **by** (*simp add*: *semidecisive-def*)
moreover
from *jxyzP′ ixyzP′* **have** $\bigwedge i.\ i \in Is \Longrightarrow y\ _{(P'\ i)}\!\prec\ z$ **by** (*case-tac i=j*, *auto*)
with *wp profileP′ has3A* **have** $y\ _{(swf\ P')}\!\prec\ z$ **by** (*auto dest*: *weak-paretoD*)
moreover note *SWF-rpr*[*OF swf*] *profileP′*
ultimately show $x\ _{(swf\ P')}\!\prec\ z$
unfolding *universal-domain-def* **by** (*blast dest*: *rpr-less-trans*)
qed
ultimately show $x\ _{(swf\ P)}\!\prec\ z$ **unfolding** *strict-pref-def* **by** *blast*
qed

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 *has3A*: *hasw* [x,y,z] A
and *profileP*: *profile A Is P*
and *jIs*: $j \in Is$
obtains P'
where *profile A Is P′*
and $z\ _{(P'\ j)}\!\prec\ x \wedge x\ _{(P'\ j)}\!\prec\ y$
and $\bigwedge i.\ i \neq j \Longrightarrow z\ _{(P'\ i)}\!\prec\ x \wedge y\ _{(P'\ i)}\!\prec\ x \wedge ((y\ _{(P'\ i)}\!\preceq\ z) = (y\ _{(P\ i)}\!\preceq\ z)) \wedge ((z\ _{(P'\ i)}\!\preceq\ y) = (z\ _{(P\ i)}\!\preceq\ y))$
proof

```
  let ?P′ = λi. (if i = j then ({ (z, u) | u. u ∈ A }
                          ∪ { (x, u) | u. u ∈ A − {z} }
                          ∪ { (y, u) | u. u ∈ A − {x,z} })
                       else ({ (z, u) | u. u ∈ A − {y} }
                          ∪ { (y, u) | u. u ∈ A − {z} }
                          ∪ { (x, u) | u. u ∈ A − {y,z} }
                          ∪ (if y (P i)≼ z then {(y,z)} else {})
                          ∪ (if z (P i)≼ y then {(z,y)} else {})))
                   ∪ (A − {x,y,z}) × (A − {x,y,z})
  show profile A Is ?P′
  proof
    fix i assume iIs: i ∈ Is
    show rpr A (?P′ i)
    proof(cases i = j)
      case True with has3A show ?thesis
        by − (rule rprI, simp-all add: trans-def, blast+)
    next
      case False hence ij: i ≠ j .
      show ?thesis
      proof
        from iIs profileP have complete A (P i) by (auto simp add: rpr-def)
        with ij show complete A (?P′ i) by (simp add: complete-def, blast)
        from iIs profileP have refl-on A (P i) by (auto simp add: rpr-def)
        with has3A ij show refl-on A (?P′ i) by (simp, blast)
        from ij has3A show trans (?P′ i) by (clarsimp simp add: trans-def)
      qed
    qed
  next
    show Is ≠ {} by (rule profile-non-empty[OF profileP])
  qed
  from has3A
  show z (?P′ j)≺ x ∧ x (?P′ j)≺ y
   and ⋀i. i ≠ j ⟹ z (?P′ i)≺ x ∧ y (?P′ i)≺ x ∧ ((y (?P′ i)≼ z) = (y (P i)≼ z)) ∧ ((z (?P′ i)≼
y) = (z (P i)≼ y))
    unfolding strict-pref-def by auto
qed

lemma decisive2:
  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} z y
proof
  from sd show jIs: {j} ⊆ Is by blast
  fix P
  assume profileP: profile A Is P
     and jyzP: ⋀i. i ∈ {j} ⟹ z (P i)≺ y
  from has3A profileP jIs
  obtain P′
    where profileP′: profile A Is P′
```

and *jxyzP'*: $z\ {}_{(P'\ j)}\!\prec x\ x\ {}_{(P'\ j)}\!\prec y$
and *ixyzP'*: $\bigwedge i.\ i \neq j \longrightarrow z\ {}_{(P'\ i)}\!\prec x \wedge y\ {}_{(P'\ i)}\!\prec x \wedge ((y\ {}_{(P'\ i)}\!\preceq z) = (y\ {}_{(P\ i)}\!\preceq z)) \wedge ((z\ {}_{(P'\ i)}\!\preceq y) = (z\ {}_{(P\ i)}\!\preceq y))$
by − (*rule decisive2-witness*, *blast*+)
from *iia* **have** $\bigwedge a\ b.\ [\![\ a \in \{y,\ z\};\ b \in \{y,\ z\}\ ]\!] \Longrightarrow (a\ {}_{(swf\ P)}\!\preceq b) = (a\ {}_{(swf\ P')}\!\preceq b)$
proof(*rule iiaE*)
from *has3A* **show** $\{y,z\} \subseteq A$ **by** *simp*
next
fix *i* **assume** *iIs*: $i \in Is$
fix *a b* **assume** *ab*: $a \in \{y,\ z\}\ b \in \{y,\ z\}$
show $(a\ {}_{(P'\ i)}\!\preceq b) = (a\ {}_{(P\ i)}\!\preceq b)$
proof(*cases* $i = j$)
case *False*
with *ab iIs ixyzP' profileP profileP' has3A*
show *?thesis* **unfolding** *profile-def* **by** *auto*
next
case *True*
from *profileP' jIs jxyzP'* **have** $z\ {}_{(P'\ j)}\!\prec y$
by (*auto dest*: *rpr-less-trans*)
with *True ab iIs jyzP profileP profileP' has3A*
show *?thesis* **unfolding** *profile-def strict-pref-def* **by** *auto*
qed
qed (*simp-all add*: *profileP profileP'*)
moreover have $z\ {}_{(swf\ P')}\!\prec y$
proof −
from *profileP' sd jxyzP' ixyzP'* **have** $x\ {}_{(swf\ P')}\!\prec y$ **by** (*simp add*: *semidecisive-def*)
moreover
from *jxyzP' ixyzP'* **have** $\bigwedge i.\ i \in Is \Longrightarrow z\ {}_{(P'\ i)}\!\prec x$ **by** (*case-tac* $i{=}j$, *auto*)
with *wp profileP' has3A* **have** $z\ {}_{(swf\ P')}\!\prec x$ **by** (*auto dest*: *weak-paretoD*)
moreover note *SWF-rpr*[*OF swf*] *profileP'*
ultimately show $z\ {}_{(swf\ P')}\!\prec y$
unfolding *universal-domain-def* **by** (*blast dest*: *rpr-less-trans*)
qed
ultimately show $z\ {}_{(swf\ P)}\!\prec y$ **unfolding** *strict-pref-def* **by** *blast*
qed

The following results permute x, y and z to show how decisiveness can be obtained from semi-decisiveness in all cases. Again, quite tedious.

lemma *decisive3*:
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 z*
shows *decisive swf A Is* $\{j\}$ *y z*
using *has3A decisive2*[*OF* - *iia swf wp sd*] **by** (*simp*, *blast*)

lemma *decisive4*:
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 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} x y 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 ≠ y
    with uv 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 ≠ x
    with uv xyA iia swf wp sd have ?thesis by (auto intro: j-decisive-xy)
  }
  moreover
  {
    assume uNExy: u ∉ {x,y} and vx: v = x
    with uv xyA iia swf wp sd have ?thesis by (auto intro: j-decisive-xy[of x y])
  }
  moreover
  {
    assume uNExy: u ∉ {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
```

```
  {
    assume uNExy: u ∉ {x,y} and vNExy: v ∉ {x,y}
    with uv xyA iia swf wp sd
    have decisive swf A Is {j} x u by (auto intro: j-decisive-xy[where x=x and z=u])
    hence sdxu: semidecisive swf A Is {j} x u by (rule d-imp-sd)
    with uNExy vNExy uv xyA iia swf wp have ?thesis by (auto intro: j-decisive-xy[of x])
  }
  ultimately show ?thesis by blast
qed
```

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

```
lemma sd-imp-dictator:
  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 uv: hasw [u,v] A
      and sd: semidecisive swf A Is {j} u v
  shows dictator swf A Is j
proof
  fix x y assume x: x ∈ A and y: y ∈ A
  show decisive swf A Is {j} x y
  proof(cases x = y)
    case True with sd show decisive swf A Is {j} x y by (blast intro: d-refl)
  next
    case False
    with x y iia swf wp has3A uv sd show decisive swf A Is {j} x y
      by (auto intro: j-decisive)
  qed
next
  from sd show j ∈ 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 pardox (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 has3A: hasw [x,y,z] A
      and Vs: Is = V1 ∪ V2 ∪ V3
              ∧ V1 ∩ V2 = {} ∧ V1 ∩ V3 = {} ∧ V2 ∩ V3 = {}
      and Is: Is ≠ {}
  obtains P
    where profile A Is P
      and ∀ i ∈ V1. x (P i)≺ y ∧ y (P i)≺ z
      and ∀ i ∈ V2. z (P i)≺ x ∧ x (P i)≺ y
      and ∀ i ∈ V3. y (P i)≺ z ∧ z (P i)≺ x
proof
  let ?P =
    λi. (if i ∈ V1 then ({ (x, u) | u. u ∈ A }
```

```
                        ∪ { (y, u) | u. u ∈ A ∧ u ≠ x }
                        ∪ { (z, u) | u. u ∈ A ∧ u ≠ x ∧ u ≠ y })
                  else
          if i ∈ V2 then ({ (z, u) | u. u ∈ A }
                        ∪ { (x, u) | u. u ∈ A ∧ u ≠ z }
                        ∪ { (y, u) | u. u ∈ A ∧ u ≠ x ∧ u ≠ z })
                  else  ({ (y, u) | u. u ∈ A }
                        ∪ { (z, u) | u. u ∈ A ∧ u ≠ y }
                        ∪ { (x, u) | u. u ∈ A ∧ u ≠ y ∧ u ≠ z }))
                      ∪ { (u, v) | u v. u ∈ A − {x,y,z} ∧ v ∈ A − {x,y,z}}
  show profile A Is ?P
  proof
    fix i assume iIs: i ∈ Is
    show rpr A (?P i)
    proof
      show complete A (?P i) by (simp add: complete-def, blast)
      from has3A iIs show refl-on A (?P i) by − (simp, blast)
      from has3A iIs show trans (?P i) by (clarsimp simp add: trans-def)
    qed
  next
    from Is show Is ≠ {} .
  qed
  from has3A Vs
  show ∀ i ∈ V1. x (?P i)≺ y ∧ y (?P i)≺ z
   and ∀ i ∈ V2. z (?P i)≺ x ∧ x (?P i)≺ y
   and ∀ i ∈ V3. y (?P i)≺ z ∧ z (?P i)≺ x
    unfolding strict-pref-def by auto
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 has3A: has 3 A
      and finiteIs: finite Is
      and twoIs: 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
  shows ∃ j u v. hasw [u,v] A ∧ semidecisive swf A Is {j} u v
proof −
  let ?P = λS. S ⊆ Is ∧ S ≠ {} ∧ (∃ u v. hasw [u,v] A ∧ semidecisive swf A Is S u v)
  obtain u v where uvA: hasw [u,v] A
    using has-witness-two[OF has3A] by auto
      — The weak pareto requirement implies that the set of all individuals is decisive between any given alternatives.
  hence decisive swf A Is Is u v
    by − (rule, auto intro: weak-paretoD[OF wp])
  hence semidecisive swf A Is Is u v by (rule d-imp-sd)
  with uvA twoIs has-suc-notempty[where n=1] nat-2[symmetric]
  have ?P Is by auto
      — Obtain a minimally-sized semi-decisive set.
  from ex-has-least-nat[where P=?P and m=card, OF this]
```

```
obtain V x y where VIs: V ⊆ Is
  and Vnotempty: V ≠ {}
  and xyA: hasw [x,y] A
  and Vsd: semidecisive swf A Is V x y
  and Vmin: ⋀V'. ?P V' ⟹ card V ≤ card V'
  by blast
from VIs finiteIs have Vfinite: finite V by (rule finite-subset)
    — Show that minimal set contains a single individual.
from Vfinite Vnotempty have ∃j. V = {j}
proof(rule finite-set-singleton-contra)
  assume Vcard: 1 < card V
  then obtain j where jV: {j} ⊆ V
    using has-extend-witness[where xs=[], OF card-has[where n=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[OF Vfinite] Vcard
  have V2card: card ?V2 > 0 card ?V2 < card V by auto
  hence V2notempty: {} ≠ ?V2 by auto
  from jV VIs
  have jV2V3: Is = ?V1 ∪ ?V2 ∪ ?V3 ∧ ?V1 ∩ ?V2 = {} ∧ ?V1 ∩ ?V3 = {} ∧ ?V2 ∩ ?V3 =
{}
    by auto
      — Show that that individual is semi-decisive for x over z.
  from has-extend-witness'[OF has3A xyA]
  obtain z where threeDist: hasw [x,y,z] A by auto
  from sd-exists-witness[OF threeDist jV2V3] VIs Vnotempty
  obtain P where profileP: profile A Is P
           and V1xyzP: x (P j)≺ y ∧ y (P j)≺ z
           and V2xyzP: ∀i ∈ ?V2. z (P i)≺ x ∧ x (P i)≺ y
           and V3xyzP: ∀i ∈ ?V3. y (P i)≺ z ∧ z (P i)≺ x
    by (simp, blast)
  have xPz: x (swf P)≺ z
  proof(rule rpr-less-le-trans[where y=y])
    from 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)≺ y
    proof −
      from profileP V3xyzP
      have ∀i ∈ ?V3. y (P i)≺ x by (blast dest: rpr-less-trans)
      with profileP V1xyzP V2xyzP Vsd
      show ?thesis unfolding semidecisive-def by auto
    qed
  next
    — This result is unfortunately quite tortuous.
    from SWF-rpr[OF swf] show y (swf P)≼ z
    proof(rule rpr-less-not[OF - - notI])
      from threeDist show hasw [z, y] A by auto
    next
      assume zPy: z (swf P)≺ y
```

```
    have semidecisive swf A Is ?V2 z y
    proof
      from VIs show V − {j} ⊆ Is by blast
    next
      fix P'
      assume profileP': profile A Is P'
        and V2yz': ⋀i. i ∈ ?V2 ⟹ z (P' i)≺ y
        and nV2yz': ⋀i. i ∈ Is − ?V2 ⟹ y (P' i)≺ z
      from iia have ⋀a b. ⟦ a ∈ {y, z}; b ∈ {y, z} ⟧ ⟹ (a (swf P)≼ b) = (a (swf P')≼ b)
      proof(rule iiaE)
        from threeDist show yzA: {y,z} ⊆ A by simp
      next
        fix i assume iIs: i ∈ Is
        fix a b assume ab: a ∈ {y, z} b ∈ {y, z}
        with VIs profileP V2xyzP
        have V2yzP: ∀ i ∈ ?V2. z (P i)≺ y by (blast dest: rpr-less-trans)
        show (a (P' i)≼ b) = (a (P i)≼ b)
        proof(cases i ∈ ?V2)
          case True
          with VIs profileP profileP' ab V2yz' V2yzP threeDist
          show ?thesis unfolding strict-pref-def profile-def by auto
        next
          case False
          from V1xyzP V3xyzP
          have ∀ i ∈ Is − ?V2. y (P i)≺ z by auto
          with iIs False VIs jV profileP profileP' ab nV2yz' threeDist
          show ?thesis unfolding profile-def strict-pref-def by auto
        qed
      qed (simp-all add: profileP profileP')
      with zPy show z (swf P')≺ 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: profileP threeDist)
qed
have semidecisive swf A Is ?V1 x z
proof
  from jV VIs show {j} ⊆ Is by blast
next
  — Use iia to show the SWF must allow the individual to prevail.
  fix P'
  assume profileP': profile A Is P'
    and V1yz': ⋀i. i ∈ ?V1 ⟹ x (P' i)≺ z
    and nV1yz': ⋀i. i ∈ Is − ?V1 ⟹ z (P' i)≺ x
  from iia have ⋀a b. ⟦ a ∈ {x, z}; b ∈ {x, z} ⟧ ⟹ (a (swf P)≼ b) = (a (swf P')≼ b)
  proof(rule iiaE)
    from threeDist show xzA: {x,z} ⊆ A by simp
  next
    fix i assume iIs: i ∈ Is
    fix a b assume ab: a ∈ {x, z} b ∈ {x, z}
    show (a (P' i)≼ b) = (a (P i)≼ b)
```

```
      proof(cases i ∈ ?V1)
        case True
        with jV VIs profileP V1xyzP
        have ∀ i ∈ ?V1. x (P i)≺ 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 ∀ i ∈ Is − ?V1. z (P i)≺ x by auto
        with iIs False VIs jV 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')≺ z unfolding strict-pref-def by blast
  qed
  with jV VIs Vsd Vmin[where V'=?V1] V2card threeDist show False
    by auto
 qed
 with xyA 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* ⇒ '*a set* ⇒ '*i set* ⇒ ('*a set* ⇒ '*i set* ⇒ ('*a*, '*i*) *Profile* ⇒ *bool*) ⇒ *bool*
where
  SDF sdf A Is Pcond ≡ (∀ *P*. *Pcond A Is P* ⟶ *choiceFn A* (*sdf P*))

lemma *SDFI*[*intro*]:
  (⋀*P*. *Pcond A Is P* ⟹ *choiceFn A* (*sdf P*)) ⟹ *SDF sdf A Is Pcond*
  unfolding *SDF-def* **by** *simp*

lemma *SWF-SDF*:
  assumes *finiteA*: *finite A*
  shows *SWF scf A Is universal-domain* ⟹ *SDF scf A 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*: *hasw* [*x*,*y*] *A*
      and *has2Is*: *hasw* [*i*,*j*] *Is*
  obtains *P*
  where *profile A Is P*
    and $x \ _{(P\ i)}\!\prec\ y$
    and $y \ _{(P\ j)}\!\prec\ x$
proof
  let *?P* = λ*k*. (*if k* = *i then* ({ (*x*, *u*) | *u*. *u* ∈ *A* }
                              ∪ { (*y*, *u*) | *u*. *u* ∈ *A* − {*x*} })
                        else ({ (*y*, *u*) | *u*. *u* ∈ *A* }
                              ∪ { (*x*, *u*) | *u*. *u* ∈ *A* − {*y*} }))
                    ∪ (*A* − {*x*,*y*}) × (*A* − {*x*,*y*})
  show *profile A Is ?P*
  proof
    fix *i* **assume** *iis*: *i* ∈ *Is*
    from *has2A* **show** *rpr A* (*?P i*)
      by − (*rule rprI*, *simp-all add*: *trans-def*, *blast*+)
  next
    from *has2Is* **show** *Is* ≠ {} **by** *auto*
  qed
  from *has2A has2Is*
  show $x \ _{(?P\ i)}\!\prec\ y$
   and $y \ _{(?P\ j)}\!\prec\ x$
    unfolding *strict-pref-def* **by** *auto*
qed

lemma *SDF-possibility*:
  assumes *finiteA*: *finite A*
      and *has2A*: *has 2 A*
      and *has2Is*: *has 2 Is*
  obtains *sdf*
  where *weak-pareto sdf A Is universal-domain*
    and *iia sdf A Is*

```
    and ¬(∃ j. dictator sdf A Is j)
    and SDF sdf A Is universal-domain
proof −
  let ?sdf = λP. { (x, y) . x ∈ A ∧ y ∈ A
                          ∧ ¬ ((∀ i ∈ Is. y (P i)≼ x)
                            ∧ (∃ i ∈ Is. y (P i)≺ x)) }
  have weak-pareto ?sdf A Is universal-domain
    by (rule, unfold strict-pref-def, auto dest: profile-non-empty)
  moreover
  have iia ?sdf A Is unfolding strict-pref-def by auto
  moreover
  have ¬(∃ j. dictator ?sdf A Is j)
  proof
    assume ∃ j. dictator ?sdf A Is j
    then obtain j where jIs: j ∈ Is
                and jD: ∀ x ∈ A. ∀ y ∈ A. decisive ?sdf A Is {j} x y
      unfolding dictator-def decisive-def by auto
    from jIs has-witness-two[OF has2Is] obtain i where ijIs: hasw [i,j] Is
      by auto
    from has-witness-two[OF has2A] obtain x y where xyA: hasw [x,y] A by auto
    from xyA ijIs obtain P
      where profileP: profile A Is P
        and yPix: x (P i)≺ y
        and yPjx: y (P j)≺ x
      by (rule SDF-nodictator-witness)
    from profileP jD jIs xyA yPjx have y (?sdf P)≺ x
      unfolding decisive-def by simp
    moreover
    from ijIs xyA yPjx yPix have x (?sdf P)≼ y
      unfolding strict-pref-def by auto
    ultimately show False
      unfolding strict-pref-def by blast
  qed
  moreover
  have SDF ?sdf A Is universal-domain
  proof
    fix P assume ud: universal-domain A Is P
    show choiceFn A (?sdf P)
    proof(rule r-c-qt-imp-cf[OF finiteA])
      show complete A (?sdf P) and refl-on A (?sdf P)
        unfolding strict-pref-def by auto
      show quasi-trans (?sdf P)
      proof
        fix x y z assume xy: x (?sdf P)≺ y and yz: y (?sdf P)≺ z
        from xy yz have xyzA: x ∈ A y ∈ A z ∈ A
          unfolding strict-pref-def by auto
        from xy yz have AxRy: ∀ i ∈ Is. x (P i)≼ y
                    and ExPy: ∃ i ∈ Is. x (P i)≺ y
                    and AyRz: ∀ i ∈ Is. y (P i)≼ z
          unfolding strict-pref-def by auto
        from AxRy AyRz ud have AxRz: ∀ i ∈ Is. x (P i)≼ z
```

```
      by − (unfold universal-domain-def, blast dest: rpr-le-trans)
     from ExPy AyRz ud have ExPz: ∃ i ∈ Is. x (P i)≺ z
      by − (unfold universal-domain-def, blast dest: rpr-less-le-trans)
     from xyzA AxRz ExPz show x (?sdf P)≺ 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 ≡
   (∀ i ∈ Is. ∃ x ∈ A. ∃ y ∈ A. x ≠ y
    ∧ decisive scf A Is {i} x y ∧ decisive scf A Is {i} y x)

lemma liberalE:
  ⟦ liberal scf A Is; i ∈ Is ⟧
   ⟹ ∃ x ∈ A. ∃ y ∈ A. x ≠ 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 ≡
   (∃ i ∈ Is. ∃ j ∈ Is. i ≠ j
    ∧ (∃ x ∈ A. ∃ y ∈ A. x ≠ y
       ∧ decisive scf A Is {i} x y ∧ decisive scf A Is {i} y x)
    ∧ (∃ x ∈ A. ∃ y ∈ A. x ≠ 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 ∈ Is by auto
  with has-extend-witness[where xs=[i], OF has2Is]
  obtain j where j: j ∈ Is i ≠ j by auto
  from L i j show ?thesis
```

unfolding *minimally-liberal-def* **by** (*blast intro*: *liberalE*)
qed

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.

```
lemmas 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 v (P j)≺ x
      and ∀ i ∈ Is. y (P i)≺ v
proof −
  let ?P =
    λa. if a = i then { (x, u) | u. u ∈ A }
                    ∪ { (y, u) | u. u ∈ A − {x} }
                    ∪ (A − {x,y}) × (A − {x,y})
                 else { (y, u) | u. u ∈ A }
                    ∪ { (v, u) | 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 u (P j)≺ v
```

and $\forall\, i \in Is.\ v\ {}_{(P\ i)}{\prec}\ x \wedge y\ {}_{(P\ i)}{\prec}\ u$
proof −
  let *?P* =
    $\lambda a.$ *if* $a = i$ *then* $\{\ (v,\ w) \mid w.\ w \in A\ \}$
      $\cup\ \{\ (x,\ w) \mid w.\ w \in A - \{v\}\ \}$
      $\cup\ \{\ (y,\ w) \mid w.\ w \in A - \{v,x\}\ \}$
      $\cup\ (A - \{v,x,y\}) \times (A - \{v,x,y\})$
    else $\{\ (y,\ w) \mid w.\ w \in A\ \}$
      $\cup\ \{\ (u,\ w) \mid w.\ w \in A - \{y\}\ \}$
      $\cup\ \{\ (v,\ w) \mid w.\ w \in A - \{u,y\}\ \}$
      $\cup\ (A - \{u,v,y\}) \times (A - \{u,v,y\})$
  have *profile A Is ?P*
  proof
    fix *i* **assume** *iis*: $i \in Is$
    show *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** $Is \neq \{\}$ **by** *auto*
  qed
  moreover
  from *fourA twoIs* **have** $x\ {}_{(?P\ i)}{\prec}\ y\ \ u\ {}_{(?P\ j)}{\prec}\ v\ \ \forall\, i \in Is.\ v\ {}_{(?P\ i)}{\prec}\ x \wedge y\ {}_{(?P\ i)}{\prec}\ u$
    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.

theorem *LiberalParadox*:
  assumes *SDF*: *SDF sdf A Is universal-domain*
    and *ml*: *minimally-liberal sdf A Is*
    and *wp*: *weak-pareto sdf A Is universal-domain*
  shows *False*
proof −
  from *ml* **obtain** *i j x y u v*
    where *i*: $i \in Is$ **and** *j*: $j \in Is$ **and** *ij*: $i \neq j$
      and *x*: $x \in A$ **and** *y*: $y \in A$ **and** *u*: $u \in A$ **and** *v*: $v \in A$
      and *xy*: $x \neq y$
      and *dixy*: *decisive sdf A Is* $\{i\}$ *x y*
      and *diyx*: *decisive sdf A Is* $\{i\}$ *y x*
      and *uv*: $u \neq v$
      and *djuv*: *decisive sdf A Is* $\{j\}$ *u v*
      and *djvu*: *decisive sdf A Is* $\{j\}$ *v u*
    by (*unfold minimally-liberal-def*, *auto*)
  from *i j ij* **have** *twoIs*: *hasw* [*i*,*j*] *Is* **by** *simp*
  {
    assume *xu*: $x = u$ **and** *yv*: $y = v$
    from *xy x y* **have** *twoA*: *hasw* [*x*,*y*] *A* **by** *simp*
    obtain *P*

where *profile A Is P x* $_{(P\ i)}\prec$ *y y* $_{(P\ j)}\prec$ *x*
using *liberal-witness-two*[*OF twoA twoIs*] **by** *blast*
with *i j dixy djvu xu yv* **have** *False*
by (*unfold decisive-def strict-pref-def*, *blast*)
}
moreover
{
assume *xu*: $x = u$ **and** *yv*: $y \neq v$
with *xy uv xu x y v* **have** *threeA*: *hasw* [*x*,*y*,*v*] *A* **by** *simp*
obtain *P*
where *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*[*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 dixy profileP* **have** *xPy*: *x* $_{(sdf\ P)}\prec$ *y*
by (*unfold decisive-def strict-pref-def*, *auto*)
from *AyPv weak-paretoD*[*OF wp - y v*] *profileP* **have** *yPv*: *y* $_{(sdf\ P)}\prec$ *v*
by *auto*
from *threeA profileP SDF* **have** *choiceSet* {*x*,*y*,*v*} (*sdf P*) $\neq$ {}
by (*simp add*: *SDF-def choiceFn-def*)
with *vPx 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** *twoA*: *hasw* [*x*,*y*] *A* **by** *auto*
obtain *P*
where *profile A Is P x* $_{(P\ i)}\prec$ *y y* $_{(P\ j)}\prec$ *x*
using *liberal-witness-two*[*OF twoA twoIs*] **by** *blast*
with *i j dixy djuv xv yu* **have** *False*
by (*unfold decisive-def strict-pref-def*, *blast*)
}
moreover
{
assume *xv*: $x = v$ **and** *yu*: $y \neq u$
with *xy uv u x y* **have** *threeA*: *hasw* [*x*,*y*,*u*] *A* **by** *simp*
obtain *P*
where *profileP*: *profile A Is P*
and *xPiy*: *x* $_{(P\ i)}\prec$ *y*
and *uPjx*: *u* $_{(P\ j)}\prec$ *x*
and *AyPu*: $\forall\, i \in Is.$ *y* $_{(P\ i)}\prec$ *u*
using *liberal-witness-three*[*OF threeA twoIs*] **by** *blast*
from *uPjx j djuv xv profileP* **have** *uPx*: *u* $_{(sdf\ P)}\prec$ *x*
by (*unfold decisive-def strict-pref-def*, *auto*)
from *xPiy i dixy profileP* **have** *xPy*: *x* $_{(sdf\ P)}\prec$ *y*
by (*unfold decisive-def strict-pref-def*, *auto*)
from *AyPu weak-paretoD*[*OF wp - y u*] *profileP* **have** *yPu*: *y* $_{(sdf\ P)}\prec$ *u*

```
    by auto
  from threeA profileP SDF have choiceSet {x,y,u} (sdf P) ≠ {}
    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 ≠ u and xv: x ≠ v and yu: y = u
  with v x y xy uv xu have threeA: hasw [y,x,v] A by simp
  obtain P
    where profileP: profile A Is P
      and yPix: y (P i)≺ x
      and vPjy: v (P j)≺ y
      and AxPv: ∀ i ∈ Is. x (P i)≺ v
    using liberal-witness-three[OF threeA twoIs] by blast
  from yPix i diyx profileP have yPx: y (sdf P)≺ x
    by (unfold decisive-def strict-pref-def, auto)
  from vPjy j djvu yu profileP have vPy: v (sdf P)≺ y
    by (unfold decisive-def strict-pref-def, auto)
  from AxPv weak-paretoD[OF wp - x v] profileP have xPv: x (sdf P)≺ v
    by auto
  from threeA profileP SDF have choiceSet {x,y,v} (sdf P) ≠ {}
    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 ≠ u and xv: x ≠ v and yv: y = v
  with u x y xy uv xu have threeA: hasw [y,x,u] A by simp
  obtain P
    where profileP: profile A Is P
      and yPix: y (P i)≺ x
      and uPjy: u (P j)≺ y
      and AxPu: ∀ i ∈ Is. x (P i)≺ u
    using liberal-witness-three[OF threeA twoIs] by blast
  from yPix i diyx profileP have yPx: y (sdf P)≺ x
    by (unfold decisive-def strict-pref-def, auto)
  from uPjy j djuv yv profileP have uPy: u (sdf P)≺ y
    by (unfold decisive-def strict-pref-def, auto)
  from AxPu weak-paretoD[OF wp - x u] profileP have xPu: x (sdf P)≺ u
    by auto
  from threeA profileP SDF have choiceSet {x,y,u} (sdf P) ≠ {}
    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 ≠ u and xv: x ≠ v and yu: y ≠ u and yv: y ≠ v
```

with *u v x y xy uv xu* **have** *fourA*: *hasw* [*x*,*y*,*u*,*v*] *A* **by** *simp*
obtain *P*
where *profileP*: *profile A Is P*
and *xPiy*: $x\ _{(P\ i)}\!\prec y$
and *uPjv*: $u\ _{(P\ j)}\!\prec v$
and *AvPxAyPu*: $\forall i \in Is.\ v\ _{(P\ i)}\!\prec x \wedge y\ _{(P\ i)}\!\prec u$
using *liberal-witness-four*[*OF fourA twoIs*] **by** *blast*
from *xPiy i dixy profileP* **have** *xPy*: $x\ _{(sdf\ P)}\!\prec y$
by (*unfold decisive-def strict-pref-def*, *auto*)
from *uPjv j djuv profileP* **have** *uPv*: $u\ _{(sdf\ P)}\!\prec v$
by (*unfold decisive-def strict-pref-def*, *auto*)
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* ⇒ '*a set* ⇒ '*i set* ⇒ *bool* **where**
anonymous scf A Is ≡
$(\forall P\ f\ x\ y.\ profile\ A\ Is\ P \wedge bij\text{-}betw\ f\ Is\ Is \wedge x \in A \wedge y \in A$
$\longrightarrow (x\ _{(scf\ P)}\!\preceq y) = (x\ _{(scf\ (P \circ f))}\!\preceq y))$

lemma *anonymousI*[*intro*]:
$(\bigwedge P\ f\ x\ y.\ [\![\ profile\ A\ Is\ P;\ bij\text{-}betw\ f\ Is\ Is;$
$x \in A;\ y \in A\ ]\!] \Longrightarrow (x\ _{(scf\ P)}\!\preceq y) = (x\ _{(scf\ (P \circ f))}\!\preceq y))$
⟹ *anonymous scf A Is*
unfolding *anonymous-def* **by** *simp*

lemma *anonymousD*:
$\llbracket$ *anonymous scf A Is*; *profile A Is P*; *bij-betw f Is Is*; $x \in A$; $y \in A$ $\rrbracket$
$\Longrightarrow (x\ {}_{(scf\ P)}{\preceq}\ y) = (x\ {}_{(scf\ (P \circ f))}{\preceq}\ y)$
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* :: $('a,\ 'i)\ SCF \Rightarrow\ 'a\ set \Rightarrow\ 'i\ set \Rightarrow bool$ **where**
neutral scf A Is $\equiv$
$(\forall P\ P'\ x\ y\ z\ w.\ profile\ A\ Is\ P \wedge profile\ A\ Is\ P' \wedge x \in A \wedge y \in A \wedge z \in A \wedge w \in A$
$\wedge (\forall i \in Is.\ x\ {}_{(P\ i)}{\preceq}\ y \longleftrightarrow z\ {}_{(P'\ i)}{\preceq}\ w) \wedge (\forall i \in Is.\ y\ {}_{(P\ i)}{\preceq}\ x \longleftrightarrow w\ {}_{(P'\ i)}{\preceq}\ z)$
$\longrightarrow ((x\ {}_{(scf\ P)}{\preceq}\ y \longleftrightarrow z\ {}_{(scf\ P')}{\preceq}\ w) \wedge (y\ {}_{(scf\ P)}{\preceq}\ x \longleftrightarrow w\ {}_{(scf\ P')}{\preceq}\ z)))$

lemma *neutralI*[*intro*]:
$(\bigwedge P\ P'\ x\ y\ z\ w.$
$\llbracket$ *profile A Is P*; *profile A Is P'*; $\{x,y,z,w\} \subseteq A$;
$\bigwedge i.\ i \in Is \Longrightarrow x\ {}_{(P\ i)}{\preceq}\ y \longleftrightarrow z\ {}_{(P'\ i)}{\preceq}\ w$;
$\bigwedge i.\ i \in Is \Longrightarrow y\ {}_{(P\ i)}{\preceq}\ x \longleftrightarrow w\ {}_{(P'\ i)}{\preceq}\ z$ $\rrbracket$
$\Longrightarrow ((x\ {}_{(scf\ P)}{\preceq}\ y \longleftrightarrow z\ {}_{(scf\ P')}{\preceq}\ w) \wedge (y\ {}_{(scf\ P)}{\preceq}\ x \longleftrightarrow w\ {}_{(scf\ P')}{\preceq}\ z)))$
$\Longrightarrow$ *neutral scf A Is*
unfolding *neutral-def* **by** *simp*

lemma *neutralD*:
$\llbracket$ *neutral scf A Is*;
profile A Is P; *profile A Is P'*; $\{x,y,z,w\} \subseteq A$;
$\bigwedge i.\ i \in Is \Longrightarrow x\ {}_{(P\ i)}{\preceq}\ y \longleftrightarrow z\ {}_{(P'\ i)}{\preceq}\ w$;
$\bigwedge i.\ i \in Is \Longrightarrow y\ {}_{(P\ i)}{\preceq}\ x \longleftrightarrow w\ {}_{(P'\ i)}{\preceq}\ z$ $\rrbracket$
$\Longrightarrow (x\ {}_{(scf\ P)}{\preceq}\ y \longleftrightarrow z\ {}_{(scf\ P')}{\preceq}\ w) \wedge (y\ {}_{(scf\ P)}{\preceq}\ x \longleftrightarrow w\ {}_{(scf\ P')}{\preceq}\ z)$
unfolding *neutral-def* **by** *simp*

Neutrality implies independence of irrelevant alternatives.

lemma *neutral-iia*: *neutral scf A Is* $\Longrightarrow$ *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* :: $('a,\ 'i)\ SCF \Rightarrow\ 'a\ set \Rightarrow\ 'i\ set \Rightarrow bool$ **where**
positively-responsive scf A Is $\equiv$
$(\forall P\ P'\ x\ y.\ profile\ A\ Is\ P \wedge profile\ A\ Is\ P' \wedge x \in A \wedge y \in A$
$\wedge (\forall i \in Is.\ (x\ {}_{(P\ i)}{\prec}\ y \longrightarrow x\ {}_{(P'\ i)}{\prec}\ y) \wedge (x\ {}_{(P\ i)}{\approx}\ y \longrightarrow x\ {}_{(P'\ i)}{\preceq}\ y))$
$\wedge (\exists k \in Is.\ (x\ {}_{(P\ k)}{\approx}\ y \wedge x\ {}_{(P'\ k)}{\prec}\ y) \vee (y\ {}_{(P\ k)}{\prec}\ x \wedge x\ {}_{(P'\ k)}{\preceq}\ y))$
$\longrightarrow x\ {}_{(scf\ P)}{\preceq}\ y \longrightarrow x\ {}_{(scf\ P')}{\prec}\ y)$

lemma *positively-responsiveI*[*intro*]:
assumes I: $\bigwedge P\ P'\ x\ y.$
$\llbracket$ *profile A Is P*; *profile A Is P'*; $x \in A$; $y \in A$;
$\bigwedge i.\ \llbracket\ i \in Is;\ x\ {}_{(P\ i)}{\prec}\ y\ \rrbracket \Longrightarrow x\ {}_{(P'\ i)}{\prec}\ y$;
$\bigwedge i.\ \llbracket\ i \in Is;\ x\ {}_{(P\ i)}{\approx}\ y\ \rrbracket \Longrightarrow x\ {}_{(P'\ i)}{\preceq}\ y$;
$\exists k \in Is.\ (x\ {}_{(P\ k)}{\approx}\ y \wedge x\ {}_{(P'\ k)}{\prec}\ y) \vee (y\ {}_{(P\ k)}{\prec}\ x \wedge x\ {}_{(P'\ k)}{\preceq}\ y)$;

x $_{(scf\ P)}\preceq$ y ⟧
$\Longrightarrow x$ $_{(scf\ P')}\prec$ y
shows *positively-responsive scf A Is*
unfolding *positively-responsive-def*
by (*blast intro*: *I*)

lemma *positively-responsiveD*:
⟦ *positively-responsive scf A Is*;
profile A Is P; *profile A Is P'*; $x \in A$; $y \in A$;
$\bigwedge i.$ ⟦ $i \in Is$; x $_{(P\ i)}\prec$ y ⟧ $\Longrightarrow x$ $_{(P'\ i)}\prec$ y;
$\bigwedge i.$ ⟦ $i \in Is$; x $_{(P\ i)}\approx$ y ⟧ $\Longrightarrow x$ $_{(P'\ i)}\preceq$ y;
$\exists\, k \in Is.\ (x$ $_{(P\ k)}\approx$ $y \wedge x$ $_{(P'\ k)}\prec$ $y) \vee (y$ $_{(P\ k)}\prec$ $x \wedge x$ $_{(P'\ k)}\preceq$ $y)$;
x $_{(scf\ P)}\preceq$ y ⟧
$\Longrightarrow x$ $_{(scf\ P')}\prec$ y
unfolding *positively-responsive-def*
apply *clarsimp*
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 Is P ≡ { (*x*, *y*) . *card* { $i \in Is.\ x$ $_{(P\ i)}\prec$ y } ≥ *card* { $i \in Is.\ y$ $_{(P\ i)}\prec$ 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*:
fixes *A* :: *'a set*
and *Is* :: *'i set*
assumes *finiteIs*: *finite Is*
shows *SCF* (*MMD Is*) *A Is universal-domain*
and *anonymous* (*MMD Is*) *A Is*
and *neutral* (*MMD Is*) *A Is*
and *positively-responsive* (*MMD Is*) *A Is*
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 $(x\ _{(MMD\ Is\ P)}\preceq\ y) = (x\ _{(MMD\ Is\ (P \circ f))}\preceq\ y)$
using *card-compose-bij*[*OF bijf*, **where** $P{=}\lambda i.\ x\ _{(P\ i)}\prec\ y$]
card-compose-bij[*OF bijf*, **where** $P{=}\lambda i.\ y\ _{(P\ i)}\prec\ x$]
unfolding *MMD-def* **by** *simp*
qed
next
show *neutral* (*MMD Is*) *A Is*
proof
fix $P\ P'$
fix $x\ y\ z\ w$ **assume** *xyzwA*: $\{x,y,z,w\} \subseteq A$
assume *xyzw*: $\bigwedge i.\ i \in Is \Longrightarrow (x\ _{(P\ i)}\preceq\ y) = (z\ _{(P'\ i)}\preceq\ w)$
and *yxwz*: $\bigwedge i.\ i \in Is \Longrightarrow (y\ _{(P\ i)}\preceq\ x) = (w\ _{(P'\ i)}\preceq\ z)$
from *xyzwA xyzw yxwz*
have $\{\ i \in Is.\ x\ _{(P\ i)}\prec\ y\ \} = \{\ i \in Is.\ z\ _{(P'\ i)}\prec\ w\ \}$
and $\{\ i \in Is.\ y\ _{(P\ i)}\prec\ x\ \} = \{\ i \in Is.\ w\ _{(P'\ i)}\prec\ z\ \}$
unfolding *strict-pref-def* **by** *auto*
thus $(x\ _{(MMD\ Is\ P)}\preceq\ y) = (z\ _{(MMD\ Is\ P')}\preceq\ w) \wedge$
$(y\ _{(MMD\ Is\ P)}\preceq\ x) = (w\ _{(MMD\ Is\ P')}\preceq\ z)$
unfolding *MMD-def* **by** *simp*
qed
next
show *positively-responsive* (*MMD Is*) *A Is*
proof
fix $P\ P'$ **assume** *profileP*: *profile A Is P*
fix $x\ y$ **assume** *xyA*: $x \in A\ y \in A$
assume *xPy*: $\bigwedge i.\ [\![i \in Is;\ x\ _{(P\ i)}\prec\ y]\!] \Longrightarrow x\ _{(P'\ i)}\prec\ y$
and *xIy*: $\bigwedge i.\ [\![i \in Is;\ x\ _{(P\ i)}\approx\ y]\!] \Longrightarrow x\ _{(P'\ i)}\preceq\ y$
and k: $\exists\, k{\in}Is.\ x\ _{(P\ k)}\approx\ y \wedge x\ _{(P'\ k)}\prec\ y \vee y\ _{(P\ k)}\prec\ x \wedge x\ _{(P'\ k)}\preceq\ y$
and *xRSCFy*: $x\ _{(MMD\ Is\ P)}\preceq\ y$
from k **obtain** k
where *kIs*: $k \in Is$
and *kcond*: $(x\ _{(P\ k)}\approx\ y \wedge x\ _{(P'\ k)}\prec\ y) \vee (y\ _{(P\ k)}\prec\ x \wedge x\ _{(P'\ k)}\preceq\ y)$
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 xPy xIy* **have** *yP'xyPx*: $?yP'x \subseteq ?yPx$
unfolding *strict-pref-def indifferent-pref-def*
by (*blast dest*: *rpr-complete*)
with *finiteIs* **have** *yP'xyPxC*: $card\ ?yP'x \leq card\ ?yPx$
by (*blast intro*: *card-mono finite-subset*)
from *finiteIs xPy* **have** *xPyxP'yC*: $card\ ?xPy \leq card\ ?xP'y$
by (*blast intro*: *card-mono finite-subset*)
show $x\ _{(MMD\ Is\ P')}\prec\ y$
proof
from *xRSCFy xPyxP'yC yP'xyPxC* **show** $x\ _{(MMD\ Is\ P')}\preceq\ y$
unfolding *MMD-def* **by** *auto*
next

```
      {
        assume xIky: x (P k)≈ y and xP'ky: x (P' k)≺ y
        have card ?xPy < card ?xP'y
        proof −
          from xIky have knP: k ∉ ?xPy
            unfolding indifferent-pref-def strict-pref-def by blast
          from kIs xP'ky have kP': k ∈ ?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)≺ x and xR'ky: x (P' k)≼ y
        have card ?yP'x < card ?yPx
        proof −
          from kIs yPkx have kP: k ∈ ?yPx by simp
          from kIs xR'ky have knP': k ∉ ?yP'x
            unfolding strict-pref-def by blast
          from yP'xyPx kP knP' have ?yP'x ⊂ ?yPx by blast
          with finiteIs show ?thesis
            by (blast intro: psubset-card-mono finite-subset)
        qed
        with xRSCFy xPyxP'yC have card ?yP'x < card ?xP'y
          unfolding MMD-def by auto
      }
      moreover note kcond
      ultimately show ¬(y (MMD Is P')≼ 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 ⇒ 'a ⇒ 'a ⇒ 'a where
  swapAlts a b u ≡ if u = a then b else if u = b then a else u

lemma swapAlts-in-set-iff: {a, b} ⊆ A ⟹ swapAlts a b u ∈ A ⟷ u ∈ A
  unfolding swapAlts-def by (simp split: if-split)

definition swapAltsP :: ('a, 'i) Profile ⇒ 'a ⇒ 'a ⇒ ('a, 'i) Profile where
  swapAltsP P a b ≡ (λi. { (u, v) . (swapAlts a b u, swapAlts a b v) ∈ P i })

lemma swapAltsP-ab: a (P i)≼ b ⟷ b (swapAltsP P a b i)≼ a b (P i)≼ a ⟷ a (swapAltsP P a b i)≼
b
```

```
  unfolding swapAltsP-def swapAlts-def by simp-all

lemma profile-swapAltsP:
  assumes profileP: profile A Is P
      and abA: {a,b} ⊆ A
  shows profile A Is (swapAltsP P a b)
proof(rule profileI)
  from profileP show Is ≠ {} by (rule profile-non-empty)
next
  fix i assume iIs: i ∈ 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 iIs abA show swapAltsP P a b i ⊆ A × A
        unfolding swapAltsP-def by (blast dest: swapAlts-in-set-iff)
      from profileP iIs abA show ⋀x. x ∈ A ⟹ x (swapAltsP P a b i)≼ x
        unfolding swapAltsP-def swapAlts-def by auto
    qed
  next
    from profileP iIs 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 iIs show trans (swapAltsP P a b i)
      unfolding swapAltsP-def by (blast dest: rpr-le-trans intro: transI)
  qed
qed

lemma profile-bij-profile:
  assumes profileP: profile A Is P
      and bijf: bij-betw f Is Is
  shows profile A Is (P ∘ 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
```

begin

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 *profileP*: $\mathit{profile}\ A\ \mathit{Is}\ P$
    and *profileP'*: $\mathit{profile}\ A\ \mathit{Is}\ P'$
    and *xyA*: $\mathit{hasw}\ [x,y]\ A$
    and *xytally*: $\mathit{card}\ \{\ i \in \mathit{Is}.\ x\ _{(P\ i)}{\prec}\ y\ \} = \mathit{card}\ \{\ i \in \mathit{Is}.\ x\ _{(P'\ i)}{\prec}\ y\ \}$
    and *yxtally*: $\mathit{card}\ \{\ i \in \mathit{Is}.\ y\ _{(P\ i)}{\prec}\ x\ \} = \mathit{card}\ \{\ i \in \mathit{Is}.\ y\ _{(P'\ i)}{\prec}\ x\ \}$
  shows $x\ _{(\mathit{scf}\ P)}{\preceq}\ y \longleftrightarrow x\ _{(\mathit{scf}\ P')}{\preceq}\ y$
proof −
  let $?xPy = \{\ i \in \mathit{Is}.\ x\ _{(P\ i)}{\prec}\ y\ \}$
  let $?xP'y = \{\ i \in \mathit{Is}.\ x\ _{(P'\ i)}{\prec}\ y\ \}$
  let $?yPx = \{\ i \in \mathit{Is}.\ y\ _{(P\ i)}{\prec}\ x\ \}$
  let $?yP'x = \{\ i \in \mathit{Is}.\ y\ _{(P'\ i)}{\prec}\ x\ \}$
  have *disjPxy*: $(?xPy \cup ?yPx) - ?xPy = ?yPx$
    unfolding *strict-pref-def* **by** *blast*
  have *disjP'xy*: $(?xP'y \cup ?yP'x) - ?xP'y = ?yP'x$
    unfolding *strict-pref-def* **by** *blast*
  from *finiteIs xytally*
  obtain f **where** *bijf*: *bij-betw f ?xPy ?xP'y*
    by − (*drule card-eq-bij*, *auto*)
  from *finiteIs yxtally*
  obtain g **where** *bijg*: *bij-betw g ?yPx ?yP'x*
    by − (*drule card-eq-bij*, *auto*)
  from *bijf bijg disjPxy disjP'xy*
  obtain h
    where *bijh*: *bij-betw* $h\ (?xPy \cup ?yPx)\ (?xP'y \cup ?yP'x)$
    and *hf*: $\bigwedge j.\ j \in ?xPy \Longrightarrow h\ j = f\ j$
    and *hg*: $\bigwedge j.\ j \in (?xPy \cup ?yPx) - ?xPy \Longrightarrow h\ j = g\ j$
   using *bij-combine*[**where** $f{=}f$ **and** $g{=}g$ **and** $A{=}?xPy$ **and** $B = ?xPy \cup ?yPx$ **and** $C{=}?xP'y$ **and** $D{=}?xP'y \cup ?yP'x$]
    by *auto*
  from *bijh finiteIs*
  obtain h' **where** *bijh'*: *bij-betw* h' *Is Is*
        and *hh'*: $\bigwedge j.\ j \in (?xPy \cup ?yPx) \Longrightarrow h'\ j = h\ j$
        and *hrest*: $\bigwedge j.\ j \in \mathit{Is} - (?xPy \cup ?yPx) \Longrightarrow h'\ j \in \mathit{Is} - (?xP'y \cup ?yP'x)$
    by − (*drule bij-complete*, *auto*)
  from *neutral-iia*[*OF neutral*]
  have $x\ _{(\mathit{scf}\ (P' \circ h'))}{\preceq}\ y \longleftrightarrow x\ _{(\mathit{scf}\ P)}{\preceq}\ y$
  proof(*rule iiaE*)
    from *xyA* **show** $\{x,\ y\} \subseteq A$ **by** *simp*
  next
    fix i **assume** *iIs*: $i \in \mathit{Is}$
    fix $a\ b$ **assume** *ab*: $a \in \{x,\ y\}\ b \in \{x,\ y\}$
    from *profileP iIs* **have** *completePi*: $\mathit{complete}\ A\ (P\ i)$ **by** (*auto dest*: *rprD*)
    from *completePi xyA*
    show $(a\ _{(P\ i)}{\preceq}\ b) \longleftrightarrow (a\ _{((P' \circ h')\ i)}{\preceq}\ b)$
    proof(*cases rule*: *complete-exh*)

case *xPy* **with** *profileP profileP′ xyA iIs ab hh′ hf bijf* **show** *?thesis*
unfolding *strict-pref-def bij-betw-def* **by** (*simp*, *blast*)
next
case *yPx* **with** *profileP profileP′ xyA iIs ab hh′ hg bijg* **show** *?thesis*
unfolding *strict-pref-def bij-betw-def* **by** (*simp*, *blast*)
next
case *xIy* **with** *profileP profileP′ xyA iIs ab hrest*[**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*: *profileP profile-bij-profile*[*OF profileP′ bijh′*])
moreover
from *anonymousD*[*OF anonymous profileP′ bijh′*] *xyA*
have $x\ {}_{(scf\ P')}{\preceq}\ y \longleftrightarrow x\ {}_{(scf\ (P' \circ h'))}{\preceq}\ y$ **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 *profileP*: *profile A Is P*
and *xyA*: *hasw* [*x*,*y*] *A*
and *tallyP*: $card\ \{\ i \in Is.\ x\ {}_{(P\ i)}{\prec}\ y\ \} = card\ \{\ i \in Is.\ y\ {}_{(P\ i)}{\prec}\ x\ \}$
shows $x\ {}_{(scf\ P)}{\approx}\ y$
proof −
— Neutrality insists the results for P are symmetrical to those for *swapAltsP P*.
from *xyA*
have *symPP′*: $(x\ {}_{(scf\ P)}{\preceq}\ y \longleftrightarrow y\ {}_{(scf\ (swapAltsP\ P\ x\ y))}{\preceq}\ x)$
$\wedge\ (y\ {}_{(scf\ P)}{\preceq}\ x \longleftrightarrow x\ {}_{(scf\ (swapAltsP\ P\ x\ y))}{\preceq}\ y)$
by − (*rule neutralD*[*OF neutral profileP profile-swapAltsP*[*OF profileP*]],
simp-all, (*rule swapAltsP-ab*)+)
— Anonymity and neutrality insist the results for P are identical to those for *swapAltsP P*.
from *xyA tallyP* **have** $card\ \{i \in Is.\ x\ {}_{(P\ i)}{\prec}\ y\} = card\ \{\ i \in Is.\ x\ {}_{(swapAltsP\ P\ x\ y\ i)}{\prec}\ y\ \}$
and $card\ \{i \in Is.\ y\ {}_{(P\ i)}{\prec}\ x\} = card\ \{\ i \in Is.\ y\ {}_{(swapAltsP\ P\ x\ y\ i)}{\prec}\ x\ \}$
unfolding *swapAltsP-def swapAlts-def strict-pref-def* **by** *simp-all*
with *profileP xyA* **have** *idPP′*: $x\ {}_{(scf\ P)}{\preceq}\ y \longleftrightarrow x\ {}_{(scf\ (swapAltsP\ P\ x\ y))}{\preceq}\ y$
and $y\ {}_{(scf\ P)}{\preceq}\ x \longleftrightarrow y\ {}_{(scf\ (swapAltsP\ P\ x\ y))}{\preceq}\ x$
by − (*rule anonymous-card*[*OF profileP profile-swapAltsP*], *clarsimp*+)+
from *xyA SCF-completeD*[*OF SCF*] *profileP symPP′ idPP′* **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 *profileP*: *profile A Is P*
and *xyA*: *hasw* [*x*,*y*] *A*
and *tallyP*: $card\ \{\ i \in Is.\ x\ {}_{(P\ i)}{\prec}\ y\ \} > card\ \{\ i \in Is.\ y\ {}_{(P\ i)}{\prec}\ x\ \}$
obtains $P'\ k$
where *profile A Is P′*
and $\bigwedge i.\ [\![i \in Is;\ x\ {}_{(P'\ i)}{\prec}\ y]\!] \Longrightarrow x\ {}_{(P\ i)}{\prec}\ y$

and $\bigwedge i.\ [\![i \in Is;\ x\ _{(P'\ i)}{\approx}\ y]\!] \Longrightarrow x\ _{(P\ i)}{\preceq}\ y$
and $k \in Is \wedge x\ _{(P'\ k)}{\approx}\ y \wedge x\ _{(P\ k)}{\prec}\ y$
and $card\ \{\ i \in Is.\ x\ _{(P'\ i)}{\prec}\ y\ \} = card\ \{\ i \in Is.\ y\ _{(P'\ i)}{\prec}\ x\ \}$
proof $-$
from *tallyP* **obtain** C
where $tallyP'$: $card\ (\{\ i \in Is.\ x\ _{(P\ i)}{\prec}\ y\ \} - C) = card\ \{\ i \in Is.\ y\ _{(P\ i)}{\prec}\ x\ \}$
and C: $C \neq \{\}\ C \subseteq Is$
and *CxPy*: $C \subseteq \{\ i \in Is.\ x\ _{(P\ i)}{\prec}\ y\ \}$
by $-$ (*drule card-greater*[*OF finiteIs*], *auto*)
— Add (b, a) and close under transitivity.
let $?P' = \lambda i.\ if\ i \in C$
$then\ P\ i \cup \{\ (y,\ x)\ \}$
$\cup \{\ (y,\ u)\ |u.\ x\ _{(P\ i)}{\preceq}\ u\ \}$
$\cup \{\ (u,\ x)\ |u.\ u\ _{(P\ i)}{\preceq}\ y\ \}$
$\cup \{\ (v,\ u)\ |u\ v.\ x\ _{(P\ i)}{\preceq}\ u \wedge v\ _{(P\ i)}{\preceq}\ y\ \}$
$else\ P\ i$
have *profile A Is ?P′*
proof
fix i **assume** *iIs*: $i \in Is$
show *rpr A* (*?P′ i*)
proof
from *profileP iIs* **show** *complete A* (*?P′ i*)
unfolding *complete-def* **by** (*simp*, *blast dest*: *rpr-complete*)
from *profileP iIs xyA* **show** *refl-on A* (*?P′ i*)
by $-$ (*rule refl-onI*, *auto*)
show *trans* (*?P′ i*)
proof(*cases* $i \in C$)
case *False* **with** *profileP iIs* **show** *?thesis*
by (*simp*, *blast dest*: *rpr-le-trans intro*: *transI*)
next
case *True* **with** *profileP iIs C CxPy xyA* **show** *?thesis*
unfolding *strict-pref-def*
by $-$ (*rule transI*, *simp*, *blast dest*: *rpr-le-trans rpr-complete*)
qed
qed
next
from C **show** $Is \neq \{\}$ **by** *blast*
qed
moreover
have $\bigwedge i.\ [\![\ i \in Is;\ x\ _{(?P'\ i)}{\prec}\ y\ ]\!] \Longrightarrow x\ _{(P\ i)}{\prec}\ y$
unfolding *strict-pref-def* **by** (*simp split*: *if-split-asm*)
moreover
from *profileP C xyA*
have $\bigwedge i.\ [\![i \in Is;\ x\ _{(?P'\ i)}{\approx}\ y]\!] \Longrightarrow x\ _{(P\ i)}{\preceq}\ y$
unfolding *indifferent-pref-def* **by** (*simp split*: *if-split-asm*)
moreover
from *C CxPy* **obtain** k **where** *kC*: $k \in C$ **and** *xPky*: $x\ _{(P\ k)}{\prec}\ y$ **by** *blast*
hence $x\ _{(?P'\ k)}{\approx}\ y$ **by** *auto*
with *C kC xPky* **have** $k \in Is \wedge x\ _{(?P'\ k)}{\approx}\ y \wedge x\ _{(P\ k)}{\prec}\ y$ **by** *blast*
moreover
have $card\ \{\ i \in Is.\ x\ _{(?P'\ i)}{\prec}\ y\ \} = card\ \{\ i \in Is.\ y\ _{(?P'\ i)}{\prec}\ x\ \}$

proof –
have $\{ i \in Is.\ x\ _{(?P'\ i)}{\prec}\ y \} = \{ i \in Is.\ x\ _{(?P'\ i)}{\prec}\ y \} - C$
proof –
from C **have** $\bigwedge i.\ [\![i \in Is;\ x\ _{(?P'\ i)}{\prec}\ y]\!] \Longrightarrow i \in Is - C$
unfolding *indifferent-pref-def strict-pref-def* **by** *auto*
thus *?thesis* **by** *blast*
qed
also have $\ldots = \{ i \in Is.\ x\ _{(P\ i)}{\prec}\ y \} - C$ **by** *auto*
finally have $card\ \{ i \in Is.\ x\ _{(?P'\ i)}{\prec}\ y \} = card\ (\{ i \in Is.\ x\ _{(P\ i)}{\prec}\ y \} - C)$
by *simp*
with *tallyP'* **have** $card\ \{ i \in Is.\ x\ _{(?P'\ i)}{\prec}\ y \} = card\ \{ i \in Is.\ y\ _{(P\ i)}{\prec}\ x \}$
by *simp*
also have $\ldots = card\ \{ i \in Is.\ y\ _{(?P'\ i)}{\prec}\ x \}$ (**is** *card ?lhs = card ?rhs*)
proof –
from *profileP xyA* **have** $\bigwedge i.\ [\![i \in Is;\ y\ _{(?P'\ i)}{\prec}\ x]\!] \Longrightarrow y\ _{(P\ i)}{\prec}\ x$
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** $\bigwedge i.\ [\![i \in Is;\ y\ _{(P\ i)}{\prec}\ x]\!] \Longrightarrow y\ _{(?P'\ i)}{\prec}\ x$
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*: *hasw* [*x*,*y*] *A*
and *tallyP*: $card\ \{ i \in Is.\ x\ _{(P\ i)}{\prec}\ y \} > card\ \{ i \in Is.\ y\ _{(P\ i)}{\prec}\ x \}$
shows $x\ _{(scf\ P)}{\prec}\ y$
proof –
from *assms* **obtain** $P'\ k$
where *profileP'*: *profile A Is P'*
and F: $\bigwedge i.\ [\![i \in Is;\ x\ _{(P'\ i)}{\prec}\ y]\!] \Longrightarrow x\ _{(P\ i)}{\prec}\ y$
and G: $\bigwedge i.\ [\![i \in Is;\ x\ _{(P'\ i)}{\approx}\ y]\!] \Longrightarrow x\ _{(P\ i)}{\preceq}\ y$
and *pivot*: $k \in Is \wedge x\ _{(P'\ k)}{\approx}\ y \wedge x\ _{(P\ k)}{\prec}\ y$
and *cardP'*: $card\ \{ i \in Is.\ x\ _{(P'\ i)}{\prec}\ y \} = card\ \{ i \in Is.\ y\ _{(P'\ i)}{\prec}\ x \}$
by – (*drule positively-responsive-prefer-witness*, *auto*)
from *profileP' xyA cardP'* **have** $x\ _{(scf\ P')}{\approx}\ y$
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*: *hasw* [*x*,*y*] *A*

shows $x\ {}_{(scf\ P)}{\preceq}\ y \longleftrightarrow x\ {}_{(MMD\ Is\ P)}{\preceq}\ y$
proof(*cases rule*: *linorder-cases*)
assume $card\ \{\ i \in Is.\ x\ {}_{(P\ i)}{\prec}\ y\ \} = card\ \{\ i \in Is.\ y\ {}_{(P\ i)}{\prec}\ x\ \}$
with *profileP xyA* **show** *?thesis*
using *anonymous-neutral-indifference*
unfolding *indifferent-pref-def MMD-def* **by** *simp*
next
assume $card\ \{\ i \in Is.\ x\ {}_{(P\ i)}{\prec}\ y\ \} > card\ \{\ i \in Is.\ y\ {}_{(P\ i)}{\prec}\ x\ \}$
with *profileP xyA* **show** *?thesis*
using *positively-responsive-prefer*
unfolding *strict-pref-def MMD-def* **by** *simp*
next
assume $card\ \{\ i \in Is.\ x\ {}_{(P\ i)}{\prec}\ y\ \} < card\ \{\ i \in Is.\ y\ {}_{(P\ i)}{\prec}\ x\ \}$
with *profileP xyA* **show** *?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 $('a,\ 'i)\ SVProfile = 'i \Rightarrow 'a$

definition $svprofile :: 'a\ set \Rightarrow 'i\ set \Rightarrow ('a,\ 'i)\ SVProfile \Rightarrow bool$ **where**
$svprofile\ A\ Is\ F \equiv Is \neq \{\} \wedge F\ `\ Is \subseteq A$

definition $plurality\text{-}rule :: 'a\ set \Rightarrow 'i\ set \Rightarrow ('a,\ 'i)\ SVProfile \Rightarrow 'a\ set$ **where**
$plurality\text{-}rule\ A\ Is\ F$
$\equiv \{\ x \in A\ .\ \forall y \in A.\ card\ \{\ i \in Is\ .\ F\ i = x\ \} \geq card\ \{\ i \in Is\ .\ F\ i = y\ \}\ \}$

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

definition $MMD\text{-}plurality\text{-}rule :: 'a\ set \Rightarrow 'i\ set \Rightarrow ('a,\ 'i)\ Profile \Rightarrow 'a\ set$ **where**
$MMD\text{-}plurality\text{-}rule\ A\ Is\ P \equiv choiceSet\ A\ (MMD\ Is\ P)$

definition *single-vote-to-RPR* :: $'a$ *set* $\Rightarrow$ $'a$ $\Rightarrow$ $'a$ *RPR* **where**
single-vote-to-RPR A a $\equiv$ $\{ (a, x) \mid x.\ x \in A \} \cup (A - \{a\}) \times (A - \{a\})$

lemma *single-vote-to-RPR-iff*:
$[\![a \in A;\ x \in A;\ a \neq x]\!] \Longrightarrow (a\ {}_{(single\text{-}vote\text{-}to\text{-}RPR\ A\ b)}{\prec}\ x) \longleftrightarrow (b = a)$
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* $\circ$ *F*)
proof –
{
fix *x y*
have $[\![x \in A;\ y \in A]\!] \Longrightarrow$
$(card\ \{i \in Is.\ F\ i = y\} \leq card\ \{i \in Is.\ F\ i = x\}) =$
$(card\ \{i \in Is.\ y\ {}_{(single\text{-}vote\text{-}to\text{-}RPR\ A\ (F\ i))}{\prec}\ x\}$
$\leq card\ \{i \in Is.\ x\ {}_{(single\text{-}vote\text{-}to\text{-}RPR\ A\ (F\ i))}{\prec}\ y\})$
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

[AK96] *Analyse & Kritik*, volume 18(1). 1996.

[Arr63] K. J. Arrow. *Social Choice and Individual Values*. John Wiley and Sons, second edition, 1963.

[GL06] R. E. Goodin and C. List. A conditional defense of plurality rule: Generalizing May's Theorem in a restricted informational environment. *American Journal of Political Science*, 50(4), 2006.

[May52] K. O. May. A set of independent, necessary and sufficient conditions for simple majority decision. *Econometrica*, 20(4), 1952.

[May53] K. O. May. A note on the complete independence of the conditions for simple majority decision. *Econometrica*, 21(1), 1953.

[Nip08] Tobias Nipkow. Arrow and gibbard-satterthwaite. *Archive of Formal Proofs*, September 2008. http://isa-afp.org/entries/ArrowImpossibilityGS.shtml, Formal proof development.

[Rou79] R. Routley. Repairing proofs of Arrow's General Impossibility Theorem and enlarging the scope of the theorem. *Notre Dame Journal of Formal Logic*, XX(4), 1979.

[Sen70] Amartya Sen. *Collective Choice and Social Welfare*. Holden Day, 1970.

[Tay05] A. D. Taylor. *Social Choice and the Mathematics of Manipulation*. Outlooks. Cambridge University Press, 2005.