Randomised Social Choice

Manuel Eberl

June 11, 2019

Abstract

This work contains a formalisation of basic Randomised Social Choice, including Stochastic Dominance and Social Decision Schemes (SDSs) along with some of their most important properties (Anonymity, Neutrality, SD-Efficiency, SD-Strategy-Proofness) and two particular SDSs – Random Dictatorship and Random Serial Dictatorship (with proofs of the properties that they satisfy). Many important properties of these concepts are also proven such as the two equivalent characterisations of Stochastic Dominance and the fact that SD-efficiency of a lottery only depends on the support.

The entry also provides convenient commands to define Preference Profiles, prove their well-formedness, and automatically derive restrictions that sufficiently nice SDSs need to satisfy on the defined profiles. (cf. [1])

Currently, the formalisation focuses on weak preferences and Stochastic Dominance (SD), but it should be easy to extend it to other domains – such as strict preferences – or other lottery extensions – such as Bilinear Dominance or Pairwise Comparison.

Contents

1 Order Relations as Binary Predicates 4
  1.1 Basic Operations on Relations ................................. 4
  1.2 Preorders ...................................................... 4
  1.3 Total preorders ............................................... 5
  1.4 Orders .......................................................... 6
  1.5 Maximal elements ............................................. 6
  1.6 Weak rankings ................................................. 8
  1.7 Rankings ........................................................ 16

2 Preference Profiles ................................................. 17
  2.1 Pareto dominance ............................................. 19
  2.2 Preferred alternatives ...................................... 20
  2.3 Favourite alternatives ...................................... 20
  2.4 Anonymous profiles ........................................ 21
1 Order Relations as Binary Predicates

theory Order-Predicates
imports
  Main
  HOL−Library.Disjoint-Sets
  HOL−Library.Permutations
  List−Index.List-Index
begin

1.1 Basic Operations on Relations

The type of binary relations

\textbf{type-synonym} 'a relation = 'a ⇒ 'a ⇒ bool

\textbf{definition} map-relation :: ('a ⇒ 'b) ⇒ 'b relation ⇒ 'a relation where
  \textbf{map-relation} f R = (λx y. R (f x) (f y))

\textbf{definition} restrict-relation :: 'a set ⇒ 'a relation ⇒ 'a relation where
  \textbf{restrict-relation} A R = (λx y. x ∈ A ∧ y ∈ A ∧ R x y)

\textbf{lemma} restrict-relation-restrict-relation [simp]:
  \textbf{restrict-relation} A (\textbf{restrict-relation} B R) = \textbf{restrict-relation} (A ∩ B) R
  \langle \text{proof} \rangle

\textbf{lemma} restrict-relation-empty [simp]: \textbf{restrict-relation} {} R = (λ- -. False)
  \langle \text{proof} \rangle

\textbf{lemma} restrict-relation-UNIV [simp]: \textbf{restrict-relation} UNIV R = R
  \langle \text{proof} \rangle

1.2 Preorders

Preorders are reflexive and transitive binary relations.

\textbf{locale} preorder-on =
  \textbf{fixes} carrier :: 'a set
  \textbf{fixes} le :: 'a relation
  \textbf{assumes} not-outside: le x y ⇒ x ∈ carrier le x y ⇒ y ∈ carrier
  \textbf{assumes} refl: x ∈ carrier ⇒ le x x
  \textbf{assumes} trans: le x y ⇒ le y z ⇒ le x z
begin

\textbf{lemma} carrier-eq: carrier = {x. le x x}
  \langle \text{proof} \rangle

\textbf{lemma} preorder-on-map:
  preorder-on (f −′ carrier) (\textbf{map-relation} f le)
  \langle \text{proof} \rangle
lemma preorder-on-restrict:
preorder-on (carrier ∩ A) (restrict-relation A le)
⟨proof⟩

lemma preorder-on-restrict-subset:
A ⊆ carrier × preorder-on A (restrict-relation A le)
⟨proof⟩

lemma restrict-relation-carrier [simp]:
restrict-relation carrier le = le
⟨proof⟩
end

1.3 Total preorders

Total preorders are preorders where any two elements are comparable.

locale total-preorder-on = preorder-on +
  assumes total: x ∈ carrier ⇒ y ∈ carrier ⇒ le x y ∨ le y x
begin

lemma total': ¬le x y ⇒ x ∈ carrier ⇒ y ∈ carrier ⇒ le y x
⟨proof⟩

lemma total-preorder-on-map:
  total-preorder-on (f − carrier) (map-relation f le)
⟨proof⟩

lemma total-preorder-on-restrict:
  total-preorder-on (carrier ∩ A) (restrict-relation A le)
⟨proof⟩

lemma total-preorder-on-restrict-subset:
  A ⊆ carrier ⇒ total-preorder-on A (restrict-relation A le)
⟨proof⟩
end

Some fancy notation for order relations

abbreviation (input) weakly-preferred :: 'a ⇒ 'a relation ⇒ 'a ⇒ bool
  (- ⪯[-] - [51,10,51] 60) where
  a ⪯[R] b ≡ R a b

definition strongly-preferred (- ⪰[-] - [51,10,51] 60) where
  a ⪰[R] b ≡ (a ⪯[R] b) ∧ ¬(b ⪯[R] a)

definition indifferent (- ∼[-] - [51,10,51] 60) where
  a ∼[R] b ≡ (a ⪯[R] b) ∧ (b ⪯[R] a)
abbreviation (input) weakly-not-preferred (- ⪰ [-] - [51,10,51] 60) where
\[ a \geq [R] b \equiv b \preceq [R] a \]
\[ \text{term } a \geq [R] b \iff b \preceq [R] a \]

abbreviation (input) strongly-not-preferred (- ≻ [-] - [51,10,51] 60) where
\[ a \succ [R] b \equiv b \prec [R] a \]

context preorder-on
begin

lemma strict-trans: \( a \prec [le] b \implies b \prec [le] c \implies a \prec [le] c \)
⟨proof⟩

lemma weak-strict-trans: \( a \preceq [le] b \implies b \prec [le] c \implies a \prec [le] c \)
⟨proof⟩

lemma strict-weak-trans: \( a \prec [le] b \implies b \preceq [le] c \implies a \prec [le] c \)
⟨proof⟩

end

lemma (in total-preorder-on) not-weakly-preferred-iff:
\[ a \in \text{carrier } \implies b \in \text{carrier } \implies \neg a \preceq [le] b \iff b \preceq [le] a \]
⟨proof⟩

lemma (in total-preorder-on) not-strongly-preferred-iff:
\[ a \in \text{carrier } \implies b \in \text{carrier } \implies \neg a \prec [le] b \iff b \preceq [le] a \]
⟨proof⟩

1.4 Orders
locale order-on = preorder-on +
\[ \text{assumes } \text{antisymmetric: } le x y \implies le y x \implies x = y \]

locale linorder-on = order-on carrier le + total-preorder-on carrier le for carrier le

1.5 Maximal elements

Maximal elements are elements in a preorder for which there exists no strictly greater element.

definition Max-wrt-among :: \( \text{\'a relation } \Rightarrow \text{\'a set } \Rightarrow \text{\'a set where} \)
Max-wrt-among \( R A = \{ x \in A. x R x \land (\forall y \in A. R x y \Rightarrow R y x) \} \)

lemma Max-wrt-among-cong:
\[ \text{\assumes restrict-relation } A R = \text{restrict-relation } A R' \]
\[ \text{shows } \text{Max-wrt-among } R A = \text{Max-wrt-among } R' A \]
⟨proof⟩
definition Max-wrt :: 'a relation ⇒ 'a set where
Max-wrt R = Max-wrt-among R UNIV

lemma Max-wrt-altdef: Max-wrt R = \{ x. R x x ∧ (∀ y. R x y → R y x)\}
⟨proof⟩

context preorder-on
begin

lemma Max-wrt-among-preorder:
Max-wrt-among le A = \{ x ∈ carrier ∩ A. ∀ y ∈ carrier ∩ A. le x y → le y x\}
⟨proof⟩

lemma Max-wrt-preorder:
Max-wrt le = \{ x ∈ carrier. ∀ y ∈ carrier. le x y → le y x\}
⟨proof⟩

lemma Max-wrt-among-subset:
Max-wrt-among le A ⊆ carrier Max-wrt-among le A ⊆ A
⟨proof⟩

lemma Max-wrt-subset:
Max-wrt le ⊆ carrier
⟨proof⟩

lemma Max-wrt-among-nonempty:
assumes B ∩ carrier ≠ \{\} finite (B ∩ carrier)
shows Max-wrt-among le B ≠ \{\}
⟨proof⟩

lemma Max-wrt-nonempty:
carrier ≠ \{\} ⇒ finite carrier ⇒ Max-wrt le ≠ \{\}
⟨proof⟩

lemma Max-wrt-among-map-relation-vimage:
f −¹ Max-wrt-among le A ⊆ Max-wrt-among (map-relation f le) (f −¹ A)
⟨proof⟩

lemma Max-wrt-map-relation-vimage:
f −¹ Max-wrt le ⊆ Max-wrt (map-relation f le)
⟨proof⟩

lemma image-subset-vimage-the-inv-into:
assumes inj-on f A B ⊆ A
shows f −¹ B ⊆ the-inv-into A f −¹ B
⟨proof⟩

lemma Max-wrt-among-map-relation-bij-subset:
assumes bij (f :: 'a ⇒ 'b)
shows f ` Max-wrt-among le A ⊆ Max-wrt-among (map-relation (inv f) le) (f ` A)
⟨proof⟩

lemma Max-wrt-among-map-relation-bij:
  assumes bij f
  shows f ` Max-wrt-among le A = Max-wrt-among (map-relation (inv f) le) (f ` A)
  ⟨proof⟩

lemma Max-wrt-map-relation-bij:
  bij f ⇒ f ` Max-wrt le = Max-wrt (map-relation (inv f) le)
  ⟨proof⟩

lemma Max-wrt-among-mono:
  le x y ⇒ x ∈ Max-wrt-among le A ⇒ y ∈ A ⇒ y ∈ Max-wrt-among le A
  ⟨proof⟩

lemma Max-wrt-mono:
  le x y ⇒ x ∈ Max-wrt le ⇒ y ∈ Max-wrt le
  ⟨proof⟩

end

context total-preorder-on
begin

lemma Max-wrt-among-total-preorder:
  Max-wrt-among le A = {x ∈ carrier ∩ A. ∀ y ∈ carrier ∩ A. le y x}
  ⟨proof⟩

lemma Max-wrt-total-preorder:
  Max-wrt le = {x ∈ carrier. ∀ y ∈ carrier. le x y}
  ⟨proof⟩

lemma decompose-Max:
  assumes A: A ⊆ carrier
  defines M ≡ Max-wrt-among le A
  shows restrict-relation A le = (λx y. x ∈ A ∧ y ∈ M ∨ (y ∉ M ∧ restrict-relation (A − M) le x y))
  ⟨proof⟩

end

1.6 Weak rankings

inductive of-weak-ranking :: 'alt set list ⇒ 'alt relation where
\[ i \leq j \Rightarrow i < \text{length } xs \Rightarrow j < \text{length } xs \Rightarrow x \in xs ! i \Rightarrow y \in xs ! j \Rightarrow x \succeq [\text{of-weak-ranking } xs] y \]

**Lemma** of-weak-ranking-Nil [simp]: of-weak-ranking \([\ ] = (\lambda - \cdot. \text{False})

**Lemma** of-weak-ranking-Nil’ [code]: of-weak-ranking \([\ ] \ x \ y = \text{False}

**Lemma** of-weak-ranking-Cons [code]:
\[ x \succeq [\text{of-weak-ranking } (z \# zs)] y \leftrightarrow x \in z \land y \in \bigcup (\text{set} \ (z \# zs)) \lor x \succeq [\text{of-weak-ranking } zs] y \]

**Lemma** of-weak-ranking-indifference:
\[ \text{assumes } A \in \text{set } xs \ x \in A \ y \in A \]
\[ \text{shows } x \preceq [\text{of-weak-ranking } xs] y \]

**Lemma** of-weak-ranking-map:
\[ \text{map-relation } f \ (\text{of-weak-ranking } xs) = \text{of-weak-ranking } (\text{map } ((\sim) \ f) \ xs) \]

**Lemma** of-weak-ranking-permute:\[\text{\ 'alt set list}\]
\[ \text{assumes } f \text{ permutes } (\bigcup (\text{set } xs)) \]
\[ \text{shows } \text{map-relation } f \ (\text{of-weak-ranking } xs) = \text{of-weak-ranking } (\text{map } ((\sim) \ f) \ xs) \]

**Lemma** of-weak-ranking-permute:
\[ \text{assumes } f \text{ permutes } (\bigcup (\text{set } xs)) \]
\[ \text{shows } \text{of-weak-ranking } (\text{map } ((\sim) \ f) \ xs) = \text{map-relation } (\text{inv } f) \ (\text{of-weak-ranking } xs) \]

**Definition** is-weak-ranking where
\[ \text{is-weak-ranking } xs \leftrightarrow (\{\} \not\in \text{set } xs) \land \]
\[ (\forall i \ j. \ i < \text{length } xs \land j < \text{length } xs \land i \neq j \rightarrow xs ! i \cap xs ! j = \{\}) \]

**Definition** is-finite-weak-ranking where
\[ \text{is-finite-weak-ranking } xs \leftrightarrow \text{is-weak-ranking } xs \land (\forall x \in \text{set } xs. \ \text{finite } x) \]

**Definition** weak-ranking :: `'alt relation ⇒ 'alt set list where
\[ \text{weak-ranking } R = (\text{SOME } xs. \ \text{is-weak-ranking } xs \land R = \text{of-weak-ranking } xs) \]

**Lemma** is-weak-rankingI [intro?]:
\[ \text{assumes } \{\} \not\in \text{set } xs \land i \ j. \ i < \text{length } xs \Rightarrow j < \text{length } xs \Rightarrow i \neq j \Rightarrow xs ! i \]
\( \cap \) \( \text{xs} \) ! \( j = \{ \} \)

**shows** is-weak-ranking \( \text{xs} \)

**lemma** is-weak-ranking-nonempty: is-weak-ranking \( \text{xs} \) \( \implies \) \{ \} \( \notin \) set \( \text{xs} \)

**lemma** is-weak-rankingD:

**assumes** is-weak-ranking \( \text{xs} \) \( i < \) length \( \text{xs} \) \( j < \) length \( \text{xs} \) \( i \neq j \)

**shows** \( \text{xs} \) ! \( i \cap \text{xs} \) ! \( j = \{ \} \)

**lemma** is-weak-ranking-iff:

is-weak-ranking \( \text{xs} \) \iff distinct \( \text{xs} \) \( \land \) disjoint (set \( \text{xs} \)) \( \land \) \{ \} \( \notin \) set \( \text{xs} \)

**lemma** is-weak-ranking-rev [simp]: is-weak-ranking (rev \( \text{xs} \)) \( \longleftrightarrow \) is-weak-ranking \( \text{xs} \)

**lemma** is-weak-ranking-map-inj:

**assumes** is-weak-ranking \( \text{xs} \) inj-on \( f \) \( \bigcup \) (set \( \text{xs} \))

**shows** is-weak-ranking (map (\( 'f \)) \( \text{xs} \))

**lemma** of-weak-ranking-rev [simp]:

of-weak-ranking (rev \( \text{xs} \)) \( x::'a \) \( y \longleftrightarrow \) of-weak-ranking \( \text{xs} \) \( y \) \( x \)

**lemma** is-weak-ranking-Nil [simp, code]: is-weak-ranking \( [] \)

**lemma** is-finite-weak-ranking-Nil [simp, code]: is-finite-weak-ranking \( [] \)

**lemma** is-weak-ranking-Cons-empty [simp]:

\( \neg \) is-weak-ranking \( \{ \} \ # \) \( \text{xs} \)

**lemma** is-finite-weak-ranking-Cons-empty [simp]:

\( \neg \) is-finite-weak-ranking \( \{ \} \ # \) \( \text{xs} \)

**lemma** is-weak-ranking-singleton [simp]:

is-weak-ranking \( [x] \) \( \longleftrightarrow \) \( x \neq \{ \} \)

**lemma** is-finite-weak-ranking-singleton [simp]:

is-finite-weak-ranking \( [x] \) \( \longleftrightarrow \) \( x \neq \{ \} \land \) finite \( x \)
lemma is-weak-ranking-append:

\[
\text{is-weak-ranking} \ (xs @ ys) \iff \\
\text{is-weak-ranking} \ xs \land \text{is-weak-ranking} \ ys \land \\
(set \ xs \cap set \ ys = \{\}) \land (\bigcup (set \ xs) \cap \bigcup (set \ ys) = \{\})
\]

⟨proof⟩

lemma is-weak-ranking-Cons [code]:

\[
\text{is-weak-ranking} \ (x \# xs) \iff \\
x \neq {} \land \text{is-weak-ranking} \ xs \land x \cap \bigcup (set \ xs) = {} 
\]

⟨proof⟩

lemma is-finite-weak-ranking-Cons [code]:

\[
\text{is-finite-weak-ranking} \ (x \# xs) \iff \\
x \neq {} \land \text{finite} \ x \land \text{is-finite-weak-ranking} \ xs \land x \cap \bigcup (set \ xs) = {} 
\]

⟨proof⟩

primrec is-weak-ranking-aux where

\[
\text{is-weak-ranking-aux} \ A [] \iff \text{True} \\
| \text{is-weak-ranking-aux} \ A \ (x \# xs) \iff x \neq {} \land \\
A \cap x = {} \land \text{is-weak-ranking-aux} \ (A \cup x) \ xs
\]

lemma is-weak-ranking-aux:

\[
\text{is-weak-ranking-aux} \ A \ xs \iff A \cap \bigcup (set \ xs) = {} \land \text{is-weak-ranking} \ xs
\]

⟨proof⟩

lemma is-weak-ranking-code [code]:

\[
\text{is-weak-ranking} \ xs \iff \text{is-weak-ranking-aux} \ \{} \ xs
\]

⟨proof⟩

lemma of-weak-ranking-altdef:

\[
\text{assumes} \ \text{is-weak-ranking} \ xs \ x \in \bigcup (set \ xs) \ y \in \bigcup (set \ xs) \\
\text{shows} \ \text{of-weak-ranking} \ xs \ x \ y \iff \\
\text{find-index} \ ((\in) \ x) \ xs \geq \text{find-index} \ ((\in) \ y) \ xs
\]

⟨proof⟩

lemma total-preorder-of-weak-ranking:

\[
\text{assumes} \ \bigcup (set \ xs) = A \\
\text{assumes} \ \text{is-weak-ranking} \ xs \\
\text{shows} \ \text{total-preorder-on} \ A \ (\text{of-weak-ranking} \ xs)
\]

⟨proof⟩

lemma restrict-relation-of-weak-ranking-Cons:

\[
\text{assumes} \ \text{is-weak-ranking} \ (A \# As) \\
\text{shows} \ \text{restrict-relation} \ (\bigcup (set \ As)) \ (\text{of-weak-ranking} \ (A \# As)) = \text{of-weak-ranking} \ As
\]

⟨proof⟩

11
lemmas of-weak-ranking-wf =
  total-preorder-of-weak-ranking is-weak-ranking-code insert-commute

lemma total-preorder-on \{1,2,3,4::nat\} (of-weak-ranking \{\{1,3\},\{2\},\{4\}\})
⟨proof⟩

context
  fixes x :: 'alt set and xs :: 'alt set list
  assumes of: is-weak-ranking (x#xs)
begin
interpretation R: total-preorder-on \bigcup (set (x#xs)) of-weak-ranking (x#xs)
⟨proof⟩

lemma of-weak-ranking-imp-in-set:
  assumes of-weak-ranking xs a b
  shows a \in \bigcup (set xs) b \in \bigcup (set xs)
⟨proof⟩

lemma of-weak-ranking-Cons':
  assumes a \in \bigcup (set (x#xs)) b \in \bigcup (set (x#xs))
  shows of-weak-ranking (x#xs) a b \iff b \in x \lor (a \notin x \land of-weak-ranking xs a b)
⟨proof⟩

lemma Max-wrt-among-of-weak-ranking-Cons1:
  assumes x \cap A = {} 
  shows Max-wrt-among (of-weak-ranking (x#xs)) A = Max-wrt-among (of-weak-ranking xs) A 
⟨proof⟩

lemma Max-wrt-among-of-weak-ranking-Cons2:
  assumes x \cap A \neq {} 
  shows Max-wrt-among (of-weak-ranking (x#xs)) A = x \cap A 
⟨proof⟩

lemma Max-wrt-among-of-weak-ranking-Cons:
  Max-wrt-among (of-weak-ranking (x#xs)) A =
  (if x \cap A = {} then Max-wrt-among (of-weak-ranking xs) A else x \cap A) 
⟨proof⟩

lemma Max-wrt-of-weak-ranking-Cons:
\[\text{Max-wrt} \ (\text{of-weak-ranking} \ (x \# xs)) = x\]
\[\langle \text{proof} \rangle\]

**end**

**lemma** Max-wrt-of-weak-ranking:
- **assumes** is-weak-ranking \(xs\)
- **shows** \(\text{Max-wrt} \ (\text{of-weak-ranking} \ xs) = (\text{if} \ xs = [] \text{then} \{\} \text{else} \text{hd} \ xs)\)
\[\langle \text{proof} \rangle\]

**locale** finite-total-preorder-on = total-preorder-on +
- **assumes** finite-carrier [intro]: finite carrier

**begin**

**lemma** finite-total-preorder-on-map:
- **assumes** finite \((f - \cdot \ carrier)\)
- **shows** finite-total-preorder-on \((f - \cdot \ carrier) \ (\text{map-relation} \ f \ le)\)
\[\langle \text{proof} \rangle\]

**function** weak-ranking-aux :: 'a set ⇒ 'a set list

\[
\text{weak-ranking-aux} \{\} = []
\]
\[
| A \neq \{\} \Rightarrow A \subseteq \text{carrier} \Rightarrow \text{weak-ranking-aux} \ A =
\]
\[
\text{Max-wrt-among} \ le \ A \neq \text{weak-ranking-aux} \ (A - \text{Max-wrt-among} \ le \ A)
\]
\[
\neg(A \subseteq \text{carrier}) \Rightarrow \text{weak-ranking-aux} \ A = \text{undefined}
\]
\[\langle \text{proof} \rangle\]

**termination** \[\langle \text{proof} \rangle\]

**lemma** weak-ranking-aux-Union:
- \(A \subseteq \text{carrier} \Rightarrow \bigcup(\text{set} \ (\text{weak-ranking-aux} \ A)) = A\)
\[\langle \text{proof} \rangle\]

**lemma** weak-ranking-aux-wf:
- \(A \subseteq \text{carrier} \Rightarrow \text{is-weak-ranking} \ (\text{weak-ranking-aux} \ A)\)
\[\langle \text{proof} \rangle\]

**lemma** of-weak-ranking-weak-ranking-aux'::
- **assumes** \(A \subseteq \text{carrier} \ x \in A \ y \in A\)
- **shows** \(\text{of-weak-ranking} \ (\text{weak-ranking-aux} \ A) \ x \ y \leftrightarrow \text{restrict-relation} \ A \ le \ x \ y\)
\[\langle \text{proof} \rangle\]

**lemma** of-weak-ranking-weak-ranking-aux:
- \(\text{of-weak-ranking} \ (\text{weak-ranking-aux} \ \text{carrier}) = \text{le}\)
\[\langle \text{proof} \rangle\]

**lemma** weak-ranking-aux-unique'::
- **assumes** \(\bigcup(\text{set} \ As) \subseteq \text{carrier} \ \text{is-weak-ranking} \ As\)
- \(\text{of-weak-ranking} \ As \ = \text{restrict-relation} \ (\bigcup(\text{set} \ As)) \ le\)
- **shows** \(As = \text{weak-ranking-aux} \ (\bigcup(\text{set} \ As))\)
lemma weak-ranking-aux-unique:
  assumes is-weak-ranking As of-weak-ranking As = le
  shows As = weak-ranking-aux carrier
  ⟨proof⟩

lemma weak-ranking-total-preorder:
  is-weak-ranking (weak-ranking le) of-weak-ranking (weak-ranking le) = le
  ⟨proof⟩

lemma weak-ranking-altdf:
  weak-ranking le = weak-ranking-aux carrier
  ⟨proof⟩

lemma weak-ranking-Union: \( \bigcup (\text{set (weak-ranking le)}) \) = carrier
  ⟨proof⟩

lemma weak-ranking-unique:
  assumes is-weak-ranking As of-weak-ranking As = le
  shows As = weak-ranking le
  ⟨proof⟩

lemma weak-ranking-permute:
  assumes f permutes carrier
  shows weak-ranking (map-relation (inv f) le) = map ((' f)) (weak-ranking le)
  ⟨proof⟩

lemma weak-ranking-index-unique:
  assumes is-weak-ranking xs i < length xs j < length xs x \( i \in xs \wedge j \in xs \)
  shows i = j
  ⟨proof⟩

lemma weak-ranking-index-unique':
  assumes is-weak-ranking xs i < length xs \( i \in xs \wedge i \)
  shows i = find-index ((\in) x) xs
  ⟨proof⟩

lemma weak-ranking-eqclass1:
  assumes A \( \in \) set (weak-ranking le) \( x \in A \wedge y \in A \)
  shows le x y
  ⟨proof⟩

lemma weak-ranking-eqclass2:
  assumes A: A \( \in \) set (weak-ranking le) \( x \in A \) and le: le x y le y x
  shows y \( \in \) A
  ⟨proof⟩

lemma hd-weak-ranking:
assumes $x \in \mathsf{hd}(\text{weak-ranking } \preceq) \ y \in \text{carrier}$
shows $\preceq y x$
(proof)

lemma last-weak-ranking:
assumes $x \in \text{last}(\text{weak-ranking } \preceq) \ y \in \text{carrier}$
shows $\preceq x y$
(proof)

The index in weak ranking of a given alternative. An element with index 0 is first-ranked; larger indices correspond to less-preferred alternatives.

definition weak-ranking-index :: 'a ⇒ nat
where
weak-ranking-index $x = \text{find-index}(\lambda A. \ x \in A) (\text{weak-ranking } \preceq)$

lemma nth-weak-ranking-index:
assumes $x \in \text{carrier}$
shows $\text{weak-ranking-index } x < \text{length}(\text{weak-ranking } \preceq) x \in \text{weak-ranking } \preceq \backslash \text{weak-ranking-index } x$
(proof)

lemma ranking-index-eqI:
i < \text{length}(\text{weak-ranking } \preceq) \implies x \in \text{weak-ranking } \preceq \backslash i = \implies \text{weak-ranking-index } x$
(proof)

lemma ranking-index-le-iff [simp]:
assumes $x, y \in \text{carrier}$
shows $\text{weak-ranking-index } x \geq \text{weak-ranking-index } y \iff \preceq x y$
(proof)

end

lemma weak-ranking-False [simp]: weak-ranking (\lambda _. False) = []
(proof)

lemmas of-weak-ranking-weak-ranking =
finite-total-preorder-on.weak-ranking-total-preorder(2)

lemma finite-total-preorder-on-iff:
finite-total-preorder-on $A \ R \iff \text{total-preorder-on } A \ R \land \text{finite } A$
(proof)

lemma finite-total-preorder-of-weak-ranking:
assumes $\bigcup (\text{set } xs) = A \ \text{is-finite-weak-ranking } xs$
shows $\text{finite-total-preorder-on } A (\text{of-weak-ranking } xs)$
(proof)

lemma weak-ranking-of-weak-ranking:
assumes $\text{is-finite-weak-ranking } xs$

15
shows  weak-ranking (of-weak-ranking xs) = xs
(proof)

lemma weak-ranking-eqD:
assumes finite-total-preorder-on alts R1
assumes finite-total-preorder-on alts R2
assumes weak-ranking R1 = weak-ranking R2
shows R1 = R2
(proof)

lemma weak-ranking-eq-iff:
assumes finite-total-preorder-on alts R1
assumes finite-total-preorder-on alts R2
shows weak-ranking R1 = weak-ranking R2 ⊣→ R1 = R2
(proof)

definition preferred-alts :: 'alt relation ⇒ 'alt ⇒ 'alt set
preferred-alts R x = {y. y ⪰ [R] x}

lemma (in preorder-on) preferred-alts-refl [simp]; x ∈ carrier ⊢ x ∈ preferred-alts le x
(proof)

lemma (in preorder-on) preferred-alts-altdef:
preferred-alts le x = {y ∈ carrier. y ⪰ [le] x}
(proof)

lemma (in preorder-on) preferred-alts-subset: preferred-alts le x ⊆ carrier
(proof)

1.7 Rankings

definition ranking :: 'a relation ⇒ 'a list
ranking R = map the-elem (weak-ranking R)

locale finite-linorder-on = linorder-on +
assumes finite-carrier [intro]: finite carrier
begin

sublocale finite-total-preorder-on carrier le
(proof)

lemma singleton-weak-ranking:
assumes A ∈ set (weak-ranking le)
sows is-singleton A
(proof)
lemma weak-ranking-ranking: weak-ranking le = map (λx. {x}) (ranking le)

⟨proof⟩

end

end

2 Preference Profiles

theory Preference-Profiles
imports
  Main
  Order-Predicates
  HOL-Library.Multiset
  HOL-Library.Disjoint-Sets
begin

The type of preference profiles

type-synonym ('agent, 'alt) pref-profile = 'agent ⇒ 'alt relation

locale preorder-family =
  fixes dom :: 'a set and carrier :: 'b set and R :: 'a ⇒ 'b relation
  assumes nonempty-dom: dom ≠ {}
  assumes in-dom [simp]: i ∈ dom ⇒ preorder-on carrier (R i)
  assumes not-in-dom [simp]: i /∈ dom ⇒ ¬R i x y
begin

lemma not-in-dom': i /∈ dom ⇒ R i = (λ-. False)

⟨proof⟩

end

locale pref-profile-wf =
  fixes agents :: 'agent set and alts :: 'alt set and R :: ('agent, 'alt) pref-profile
  assumes nonempty-agents [simp]: agents ≠ {} and nonempty-alts [simp]: alts ≠ {}
  assumes prefs-wf [simp]: i ∈ agents ⇒ finite-total-preorder-on alts (R i)
  assumes prefs-undefined [simp]: i /∈ agents ⇒ ¬R i x y
begin

lemma finite-alts [simp]: finite alts

⟨proof⟩

lemma prefs-wf' [simp]:
  i ∈ agents ⇒ total-preorder-on alts (R i) i ∈ agents ⇒ preorder-on alts (R i)

⟨proof⟩

lemma not-outside:
assumes $x \preceq_i y$
shows $i \in \text{agents} \land x \in \text{alts} \land y \in \text{alts}$
(proof)

sublocale preorder-family agents alts $R$
(proof)

lemmas $\text{prefs-undefined}' = \text{not-in-dom}'$

lemma $\text{wf-update}$:
assumes $i \in \text{agents} \land \text{total-preorder-on alts} \Rightarrow R_i'$
shows $\text{pref-profile-wf agents alts} \ (R(i := R_i'))$
(proof)

lemma $\text{wf-permute-agents}$:
assumes $\sigma$ permutes agents
shows $\text{pref-profile-wf agents alts} \ (R \circ \sigma)$
(proof)

lemma $\text{(in ~)} \text{pref-profile-eqI}$:
assumes $\text{pref-profile-wf agents alts} \ R_1 \text{ pref-profile-wf agents alts} \ R_2$
assumes $\forall x. \ x \in \text{agents} \Rightarrow R_1 x = R_2 x$
shows $R_1 = R_2$
(proof)

end

Permutes a preference profile w.r.t. alternatives in the way described in the paper. This is needed for the definition of neutrality.

definition permute-profile where
permute-profile $\sigma$ $R = (\lambda i \ x \ . \ R \ i \ (\text{inv} \ \sigma \ x) \ (\text{inv} \ \sigma \ y))$

lemma permute-profile-map-relation:
permute-profile $\sigma$ $R = (\lambda i . \ \text{map-relation} \ (\text{inv} \ \sigma) \ (R \ i))$
(proof)

lemma permute-profile-compose [simp]:
permute-profile $\sigma \ (R \circ \pi) = \text{permute-profile} \ \sigma \ R \circ \pi$
(proof)

lemma permute-profile-id [simp]: permute-profile $\text{id} \ R = R$
(proof)

lemma permute-profile-o:
assumes bij $f$ bij $g$
shows permute-profile $f \ (\text{permute-profile} \ g \ R) = \text{permute-profile} \ (f \circ g) \ R$
(proof)

lemma (in $\text{pref-profile-wf}$) $\text{wf-permute-alts}$:
assumes $\sigma$ permutes alts
shows pref-profile-of agents alts (permute-profile $\sigma R$)
⟨proof⟩

This shows that the above definition is equivalent to that in the paper.

lemma permute-profile-iff [simp]:
fixes $R :: \langle\text{'agent, 'alt} \rangle$ pref-profile
assumes $\sigma$ permutes alts $x \in \text{alts}$ $y \in \text{alts}$
defines $R' \equiv \text{permute-profile } \sigma R$
shows $\sigma x \preceq [R' i] \sigma y \iff x \preceq [R i] y$
⟨proof⟩

2.1 Pareto dominance
definition Pareto :: \langle\text{'agent $\Rightarrow$ 'alt \Rightarrow} \\text{alt relation} \rangle \Rightarrow \text{alt relation} \\text{where}
\begin{align*}
x \preceq [\text{Pareto}(R)] y & \iff (\exists j. x \preceq [R j] x) \land (\forall i. x \preceq [R i] x \rightarrow x \preceq [R i] y)
\end{align*}
A Pareto loser is an alternative that is Pareto-dominated by some other alternative.
definition pareto-losers :: \langle\text{'agent, 'alt} \rangle pref-profile $\Rightarrow$ 'alt set \\text{where}
pareto-losers $R = \{ x. \exists y. y \succ [\text{Pareto}(R)] x \}$

lemma pareto-losersI [intro?, simp]: $y \succ [\text{Pareto}(R)] x \implies x \in \text{pareto-losers } R$
⟨proof⟩

context preorder-family

begin

lemma Pareto-iff:
\begin{align*}
x \preceq [\text{Pareto}(R)] y & \iff (\forall i \in \text{dom}. x \succeq [R i] y)
\end{align*}
⟨proof⟩

lemma Pareto-strict-iff:
\begin{align*}
x \prec [\text{Pareto}(R)] y & \iff (\forall i \in \text{dom}. x \succeq [R i] y) \land (\exists i \in \text{dom}. x \prec [R i] y)
\end{align*}
⟨proof⟩

lemma Pareto-strictI:
assumes $\land i. i \in \text{dom} \implies x \succeq [R i] y$
shows $x \prec [\text{Pareto}(R)] y$
⟨proof⟩

lemma Pareto-strictI':
assumes $\land i. i \in \text{dom} \implies x \succeq [R i] y$
shows $x \prec [\text{Pareto}(R)] y$
⟨proof⟩

sublocale Pareto: preorder-on carrier Pareto(R)
⟨proof⟩

19
lemma pareto-loser-in-alts:
  assumes \( x \in \text{pareto-losers } R \)
  shows \( x \in \text{carrier} \)
⟨proof⟩

lemma pareto-losersE:
  assumes \( x \in \text{pareto-losers } R \)
  obtains \( y \) where \( y \in \text{carrier } y \succ [\text{Pareto}(R)] x \)
⟨proof⟩

end

2.2 Preferred alternatives

context pref-profile-wf
begin

lemma preferred-alts-subset-alts: preferred-alts (R i) x ⊆ alts (is ?A)
  and finite-preferred-alts [simp,intro!]: finite (preferred-alts (R i) x) (is ?B)
⟨proof⟩

lemma preferred-alts-altdef:
  \( i \in \text{agents} \implies \text{preferred-alts } (R i) x = \{ y \in \text{alts}. \ y \geq [R i] x \} \)
⟨proof⟩

end

2.3 Favourite alternatives

definition favorites :: ('agent, 'alt) pref-profile ⇒ 'agent ⇒ 'alt set where
  favorites R i = Max-wrt (R i)

definition favorite :: ('agent, 'alt) pref-profile ⇒ 'agent ⇒ 'alt where
  favorite R i = the-elem (favorites R i)

definition has-unique-favorites :: ('agent, 'alt) pref-profile ⇒ bool where
  has-unique-favorites R ←→ (∀ i. favorites R i = \{ \} ∨ is-singleton (favorites R i))

context pref-profile-wf
begin

lemma favorites-altdef:
  favorites R i = Max-wrt-among (R i) alts
⟨proof⟩

lemma favorites-no-agent [simp]: \( i \notin \text{agents} \implies \text{favorites } R i = \{ \} \)
⟨proof⟩
lemma favorites-altdef':
  \begin{align*}
  \text{favorites} R i &= \{ x \in \text{alts}. \forall y \in \text{alts}. x \succeq [R i] y \}\end{align*}
\langle proof \rangle

lemma favorites-subset-alts: favorites R i \subseteq alts
\langle proof \rangle

lemma finite-favorites [simp, intro]: finite (favorites R i)
\langle proof \rangle

lemma favorites-nonempty: i \in \text{agents} \implies \text{favorites} R i \neq \{\}
\langle proof \rangle

lemma favorites-permute:
  \begin{align*}
  \text{assumes } i &\in \text{agents and } \sigma \text{ permutes alts} \\
  \text{shows } \text{favorites} (\text{permute-profile } \sigma R) i &= \sigma ' \text{favorites} R i \\
\end{align*}
\langle proof \rangle

lemma has-unique-favorites-altdef:
  \begin{align*}
  \text{has-unique-favorites} R &\iff (\forall i \in \text{agents. is-singleton } (\text{favorites} R i)) \\
\end{align*}
\langle proof \rangle

end

locale pref-profile-unique-favorites = pref-profile-wf agents alts R
  for agents :: 'agent set and alts :: 'alt set and R +
  assumes unique-favorites': has-unique-favorites R
begin
  lemma unique-favorites: i \in \text{agents} \implies \text{favorites} R i = \{\text{favorite} R i\}
\langle proof \rangle

  lemma favorite-in-alts: i \in \text{agents} \implies \text{favorite} R i \in \text{alts}
\langle proof \rangle

end

2.4 Anonymous profiles

type-synonym ('agent, 'alt) apref-profile = 'alt set list multiset

definition anonymous-profile :: ('agent, 'alt) pref-profile \Rightarrow ('agent, 'alt) apref-profile

  where anonymous-profile-altdef:
    anonymous-profile R = image-mset (\text{weak-ranking } \circ R) (mset-set \{ i. R i \neq (\lambda -. False)\})

lemma (in pref-profile-wf) agents-eq:
agents = \{ i. R i \neq (\lambda - \cdot . False)\}

(\textit{proof})

\textbf{lemma (in pref-profile-wf) anonymous-profile-def:}
\hspace{1em} anonymous-profile R = image-mset (weak-ranking \circ R) (mset-set agents)

(\textit{proof})

\textbf{lemma (in pref-profile-wf) anonymous-profile-permute:}
\hspace{1em} \texttt{\textbf{assumes} } \sigma \texttt{\textbf{permutes} } \texttt{\textbf{alts} } \texttt{\textbf{finite} } \texttt{\textbf{agents}}
\hspace{1em} \texttt{\textbf{shows} } anonymous-profile (\texttt{permute-profile} \sigma R) = image-mset (map ((\cdot ) \sigma)) (anonymous-profile R)

(\textit{proof})

\textbf{lemma (in pref-profile-wf) anonymous-profile-update:}
\hspace{1em} \texttt{\textbf{assumes} } i. i \in agents \texttt{\textbf{and} } \texttt{\textbf{fin} } [\texttt{simp}]: \texttt{\textbf{finite} } \texttt{\textbf{agents}} \texttt{\textbf{and} } \texttt{\textbf{total-preorder-on} } \texttt{\textbf{alts} } R_i
\hspace{1em} \texttt{\textbf{shows} } anonymous-profile (R(i := R_i')) = anonymous-profile R - \{\#weak-ranking (R i)\} + \{\#weak-ranking R_i'\}

(\textit{proof})

\textbf{2.5 Preference profiles from lists}

\textbf{definition pref\text{-}from\text{-}table :: \texttt{('agent } \times \texttt{ 'alt set list) list } \Rightarrow \texttt{('agent } \times \texttt{ 'alt) pref-profile}}
\hspace{1em} \texttt{where}
\hspace{1em} \texttt{pref\text{-}from\text{-}table xss = (\lambda i. \texttt{case-option (\lambda - \cdot . False) of-weak-ranking (map-of xss i)}}

\textbf{definition pref\text{-}from\text{-}table-wf where}
\hspace{1em} \texttt{pref\text{-}from\text{-}table-wf agents alts xss } \leftrightarrow \texttt{\{ \texttt{agents} } \neq \texttt{\{} \texttt{\}} \texttt{\} } \texttt{\land \texttt{\}} \texttt{distinct (map \texttt{\textbf{fst} xss) } \texttt{\land}
\hspace{1em} \texttt{set (map \texttt{\textbf{fst} xss) = agents } \texttt{\land (\forall x s \in set (map \texttt{\textbf{snd} xss). } \texttt{\bigcup (set x s) = alts } \texttt{\land}
\hspace{1em} \texttt{is-finite-weak-ranking x s)\}

\textbf{lemma pref\text{-}from\text{-}table-wfI:}
\hspace{1em} \texttt{\textbf{assumes} } \texttt{agents } \neq \texttt{\{} \texttt{\}} \texttt{\} } \texttt{\land \texttt{\}} \texttt{distinct (map \texttt{\textbf{fst} xss) }
\hspace{1em} \texttt{\textbf{assumes} } \texttt{set (map \texttt{\textbf{fst} xss) = agents}
\hspace{1em} \texttt{\textbf{assumes} } \texttt{\bigwedge x s \in set (map \texttt{\textbf{snd} xss) } \texttt{\bigcup (set x s) = alts}
\hspace{1em} \texttt{\textbf{assumes} } \texttt{\bigwedge x s \in set (map \texttt{\textbf{snd} xss) } \texttt{\bigcup (set x s) = is-finite-weak-ranking x s}
\hspace{1em} \texttt{\textbf{shows} } \texttt{pref\text{-}from\text{-}table-wf agents alts xss (\textit{proof})}

\textbf{lemma pref\text{-}from\text{-}table-wfD:}
\hspace{1em} \texttt{\textbf{assumes} } \texttt{pref\text{-}from\text{-}table-wf agents alts xss}
\hspace{1em} \texttt{\textbf{shows} } \texttt{agents } \neq \texttt{\{} \texttt{\} } \texttt{\} } \texttt{\land \texttt{\}} \texttt{distinct (map \texttt{\textbf{fst} xss) }
\hspace{1em} \texttt{\textbf{and} } \texttt{set (map \texttt{\textbf{fst} xss) = agents}
\hspace{1em} \texttt{\textbf{and} } \texttt{\bigwedge x s \in set (map \texttt{\textbf{snd} xss) } \texttt{\bigcup (set x s) = alts}
\hspace{1em} \texttt{\textbf{and} } \texttt{\bigwedge x s \in set (map \texttt{\textbf{snd} xss) } \texttt{\bigcup (set x s) = is-finite-weak-ranking x s (\textit{proof})

22
lemma pref-profile-from-tableI:
  \( \text{prefs-from-table-wf agents alts xss} \implies \text{pref-profile-wf agents alts (prefs-from-table xss)} \)
⟨proof⟩

lemma prefs-from-table-eqI:
  assumes distinct \((\text{map fst xs})\) distinct \((\text{map fst ys})\) set xs = set ys
  shows \( \text{prefs-from-table xs} = \text{prefs-from-table ys} \)
⟨proof⟩

lemma prefs-from-table-undef:
  assumes \( \text{prefs-from-table-wf agents alts xss i} \notin \text{agents} \)
  shows \( \text{prefs-from-table xss i} = (\lambda - _. \text{False}) \)
⟨proof⟩

lemma prefs-from-table-map-of:
  assumes \( \text{prefs-from-table-wf agents alts xss i} \in \text{agents} \)
  shows \( \text{prefs-from-table xss i} = \text{of-weak-ranking (the (map-of xss i))} \)
⟨proof⟩

lemma prefs-from-table-update:
  fixes \( x \) \( x's \)
  assumes \( i \in \text{set (map fst xs)} \)
  defines \( x's' \equiv \text{map (\( \lambda (j, y). \text{if } j = i \text{ then } (j, x) \text{ else } (j, y) \)) xs} \)
  shows \( (\text{prefs-from-table xs}) (i := \text{of-weak-ranking } x)\)
  = \( \text{prefs-from-table xs'} \) (is ?lhs = ?rhs)
⟨proof⟩

lemma prefs-from-table-swap:
  \( x \neq y \implies \text{prefs-from-table ((x,x')#(y,y')#xs)} = \text{prefs-from-table ((y,y')#(x,x')#xs)} \)
⟨proof⟩

lemma permute-prefs-from-table:
  assumes \( \sigma \text{ permutes fst ' set xs} \)
  shows \( \text{prefs-from-table xs} \circ \sigma = \text{prefs-from-table (map (\( \lambda (x,y). (\text{inv } \sigma x, y) \)) xs)} \)
⟨proof⟩

lemma permute-profile-from-table:
  assumes \( \text{wf: prefs-from-table-wf agents alts xss} \)
  assumes \( \text{perm: } \sigma \text{ permutes alts} \)
  shows \( \text{permute-profile } \sigma (\text{prefs-from-table xss}) = \)
  \( \text{prefs-from-table (map (\( \lambda (x,y). (x, \text{map (\( \sigma \)) y) \)) xs)} \) (is ?f = ?g)
⟨proof⟩

2.6 Automatic evaluation of preference profiles
lemma eval-prefs-from-table [simp]:
\[
\text{prefs-from-table} \quad \| i = (\lambda - \cdot. \text{False}) \\
\text{prefs-from-table} \quad ((i, y) \# xs) \quad i = \text{of-weak-ranking} \quad y \\
i \neq j \implies \text{prefs-from-table} \quad ((j, y) \# xs) \quad i = \text{prefs-from-table} \quad xs \quad i
\]

**Lemma** eval-of-weak-ranking [simp]:
\[
a \notin \bigcup (\text{set} \quad xs) \implies \neg \text{of-weak-ranking} \quad xs \quad a \quad b \\
b \in x \implies a \in \bigcup (\text{set} \quad (x \# xs)) \implies \text{of-weak-ranking} \quad (x \# xs) \quad a \quad b \\
b \notin x \implies \text{of-weak-ranking} \quad (x \# xs) \quad a \quad b \iff \text{of-weak-ranking} \quad xs \quad a \quad b
\]

**Lemma** prefs-from-table-cong [cong]:
\[
\text{assumes} \quad \text{prefs-from-table} \quad xs = \text{prefs-from-table} \quad ys \\
\text{shows} \quad \text{prefs-from-table} \quad (x \# xs) = \text{prefs-from-table} \quad (x \# ys)
\]

**Definition** of-weak-ranking-Collect-ge where
\[
\text{of-weak-ranking-Collect-ge} \quad xs \quad x \quad = \quad \{ y. \text{of-weak-ranking} \quad xs \quad y \quad x \}
\]

**Lemma** eval-Collect-of-weak-ranking:
\[
\text{Collect} \quad (\text{of-weak-ranking} \quad xs \quad x) = \text{of-weak-ranking-Collect-ge} \quad (\text{rev} \quad xs) \quad x
\]

**Lemma** of-weak-ranking-Collect-ge-empty [simp]:
\[
\text{of-weak-ranking-Collect-ge} \quad \| \quad x \quad = \quad \{ \}
\]

**Lemma** of-weak-ranking-Collect-ge-Cons [simp]:
\[
y \in x \implies \text{of-weak-ranking-Collect-ge} \quad (x \# xs) \quad y = \bigcup (\text{set} \quad (x \# xs)) \\
y \notin x \implies \text{of-weak-ranking-Collect-ge} \quad (x \# xs) \quad y = \text{of-weak-ranking-Collect-ge} \quad xs \quad y
\]

**Lemma** of-weak-ranking-Collect-ge-Cons':
\[
\text{of-weak-ranking-Collect-ge} \quad (x \# xs) = (\lambda y. \quad (\text{if} \quad y \in x \quad \text{then} \bigcup (\text{set} \quad (x \# xs)) \quad \text{else} \quad \text{of-weak-ranking-Collect-ge} \quad xs \quad y))
\]

**Lemma** anonymise-prefs-from-table:
\[
\text{assumes} \quad \text{prefs-from-table-wf} \quad \text{agents} \quad \text{alts} \quad xs \\
\text{shows} \quad \text{anonymous-profile} \quad (\text{prefs-from-table} \quad xs) = \text{mset} \quad (\text{map} \quad \text{snd} \quad xs)
\]

**Lemma** prefs-from-table-agent-permutation:
\[
\text{assumes} \quad \text{wf}: \quad \text{prefs-from-table-wf} \quad \text{agents} \quad \text{alts} \quad xs \quad \text{prefs-from-table-wf} \quad \text{agents} \quad \text{alts} \quad ys \\
\text{assumes} \quad \text{mset-eq}: \quad \text{mset} \quad (\text{map} \quad \text{snd} \quad xs) = \text{mset} \quad (\text{map} \quad \text{snd} \quad ys) \\
\text{obtains} \quad \pi \quad \text{where} \quad \pi \quad \text{permutes} \quad \text{agents} \quad \text{prefs-from-table} \quad xs \quad \circ \quad \pi = \text{prefs-from-table} \quad ys
\]
lemma permute-list-distinct:
  assumes f : {..<length xs} ⊆ {..<length xs} distinct xs
  shows permute-list f xs = map (λx. xs ! f (index xs x)) xs
⟨proof⟩

lemma image-mset-eq-permutation:
  assumes {#f x. x ∈ #mset-set A#} = {#g x. x ∈ #mset-set A#} finite A
  obtains π where π permutes A ⋀ x ∈ A ⇒ g (π x) = f x
⟨proof⟩

lemma anonymous-profile-agent-permutation:
  assumes eq: anonymous-profile R1 = anonymous-profile R2
  assumes wf: pref-profile-wf agents alts R1 pref-profile-wf agents alts R2
  assumes fin: finite agents
  obtains π where π permutes agents R2 ◦ π = R1
⟨proof⟩

end

theory Elections

imports Preference-Profiles

begin

An election consists of a finite set of agents and a finite non-empty set of alternatives.

locale election =
  fixes agents :: 'agent set and alts :: 'alt set
  assumes finite-agents [simp, intro]: finite agents
  assumes finite-alts [simp, intro]: finite alts
  assumes nonempty-agents [simp]: agents ≠ {}
  assumes nonempty-alts [simp]: alts ≠ {}

begin

abbreviation is-pref-profile ≡ pref-profile-wf agents alts

lemma finite-total-preorder-on-iff' [simp]:
  finite-total-preorder-on alts R ←→ total-preorder-on alts R
⟨proof⟩

lemma pref-profile-wfI' [intro?):
  (⎨i. i ∈ agents ⇒ total-preorder-on alts (R i)) ⇒
  (⎨i. i /∈ agents ⇒ R i = (λ-. False)) ⇒ is-pref-profile R
⟨proof⟩

lemma is-pref-profile-update [simp,intro]:
  assumes is-pref-profile R total-preorder-on alts Ri' i ∈ agents
  shows is-pref-profile (R(i := Ri'))
⟨proof⟩

end
lemma election [simp,intro]: election agents alts
⟨proof⟩

context
  fixes R assumes R: total-preorder-on alts R
begin
interpretation R: total-preorder-on alts R ⟨proof⟩

lemma Max-wrt-prefs-finite: finite (Max-wrt R) ⟨proof⟩

lemma Max-wrt-prefs-nonempty: Max-wrt R ≠ {} ⟨proof⟩

lemma maximal-imp-preferred:
  x ∈ alts ⇒ Max-wrt R ⊆ preferred-alts R x ⟨proof⟩
end
end
end

3 Auxiliary facts about PMFs

theory Lotteries imports Complex-Main HOL−Probability.Probability begin

The type of lotteries (a probability mass function)
type-synonym 'alt lottery = 'alt pmf
definition lotteries-on :: 'a set ⇒ 'a lottery set where
  lotteries-on A = {p. set-pmf p ⊆ A}

lemma pmf-of-set-lottery:
  A ≠ {} ⇒ finite A ⇒ A ⊆ B ⇒ pmf-of-set A ∈ lotteries-on B ⟨proof⟩

lemma pmf-of-list-lottery:
  pmf-of-list-wf xs ⇒ set (map fst xs) ⊆ A ⇒ pmf-of-list xs ∈ lotteries-on A ⟨proof⟩

lemma return-pmf-in-lotteries-on [simp,intro]:
  x ∈ A ⇒ return-pmf x ∈ lotteries-on A ⟨proof⟩
theory Utility-Functions
imports
  Complex-Main
  HOL-Probability.Probability
  Lotteries
  Preference-Profiles
begin

3.1 Definition of von Neumann–Morgenstern utility functions

locale vnm-utility = finite-total-preorder-on +
  fixes u :: 'a ⇒ real
  assumes utility-le-iff: x ∈ carrier ⇒ y ∈ carrier ⇒ u x ≤ u y ↔ x ≤[le] y
begin

lemma utility-le: x ≤[le] y ⇒ u x ≤ u y
⟨proof⟩

lemma utility-less-iff:
  x ∈ carrier ⇒ y ∈ carrier ⇒ u x < u y ↔ x <[le] y
⟨proof⟩

lemma utility-less: x <[le] y ⇒ u x < u y
⟨proof⟩

The following lemma allows us to compute the expected utility by summing over all indifference classes, using the fact that alternatives in the same indifference class must have the same utility.

lemma expected-utility-weak-ranking:
  assumes p ∈ lotteries-on carrier
  shows measure-pmf.expectation p u =
  (∑ A←weak-ranking le. u (SOME x. x ∈ A) * measure-pmf.prob p A)
⟨proof⟩

lemma scaled: c > 0 ⇒ vnm-utility carrier le (λx. c * u x)
⟨proof⟩

lemma add-right:
  assumes ∃ x y. le x y ⇒ f x ≤ f y
  shows vnm-utility carrier le (λx. u x + f x)
⟨proof⟩

lemma add-left:
  (∃ x y. le x y ⇒ f x ≤ f y) ⇒ vnm-utility carrier le (λx. f x + u x)
⟨proof⟩

end
Given a consistent utility function, any function that assigns equal values to equivalent alternatives can be added to it (scaled with a sufficiently small \( \varepsilon \)), again yielding a consistent utility function.

**Lemma add-epsilon:**

**Assumes**

\[ \forall x, y. \text{le} \ x \ y \Rightarrow \text{le} \ y \ x \Rightarrow \text{f} \ x = \text{f} \ y \]

**Shows**

\[ \exists \varepsilon > 0. \text{vnm-utility carrier le} \ (\lambda x. \ u \ x + \varepsilon \ast \text{f} \ x) \]

**Proof**

**Lemma diff-epsilon:**

**Assumes**

\[ \forall x, y. \text{le} \ x \ y \Rightarrow \text{le} \ y \ x \Rightarrow \text{f} \ x = \text{f} \ y \]

**Shows**

\[ \exists \varepsilon > 0. \text{vnm-utility carrier le} \ (\lambda x. \ u \ x - \varepsilon \ast \text{f} \ x) \]

**Proof**

4 Stochastic Dominance

**Theory** Stochastic-Dominance

**Imports**

Complex-Main
HOL-Probability.Probability
Lotteries
Preference-Profiles
Utility-Functions

**Begin**

4.1 Definition of Stochastic Dominance

This is the definition of stochastic dominance. It lifts a preference relation on alternatives to the stochastic dominance ordering on lotteries.

**Definition** SD :: 'alt relation ⇒ 'alt lottery relation where

\[ p \geq_{[SD(R) \!]} q \iff p \in \text{lotteries-on} \{x. \ R \ x \ x\} \land q \in \text{lotteries-on} \{x. \ R \ x \ x\} \land \]

\[ (\forall x. \ R \ x \ x \Rightarrow \text{measure-pmf.prob} \ p \ \{y. \ y \geq_{[R]} x\} \geq \]

\[ \text{measure-pmf.prob} \ q \ \{y. \ y \geq_{[R]} x\}) \]

**Lemma** SD-empty [simp]: SD (λ- -. False) = (λ- -. False)

**Proof**

Stochastic dominance over any relation is a preorder.

**Lemma** SD-refl: p ≤_{[SD(R) \!]} p \iff p \in \text{lotteries-on} \{x. \ R \ x \ x\}

**Proof**

**Lemma** SD-trans [simp, trans]: p ≤_{[SD(R) \!]} q \Rightarrow q ≤_{[SD(R) \!]} r \Rightarrow p ≤_{[SD(R) \!]} r

**Proof**

28
lemma SD-is-preorder: preorder-on (lotteries-on \{x. R x x\}) (SD R)
(proof)

context preorder-on
begin

lemma SD-preorder:
\begin{align*}
p \succeq & [SD(le)] q \iff p \in \text{lotteries-on carrier} \land q \in \text{lotteries-on carrier} \land \\
& (\forall x \in \text{carrier}. \text{measure-pmf}.\text{prob} \ p \ (\text{preferred-alts le} \ x) \geq \\
& \text{measure-pmf}.\text{prob} \ q \ (\text{preferred-alts le} \ x) )
\end{align*}
(proof)

lemma SD-preorderI [intro?]:
assumes \( p \in \text{lotteries-on carrier} \ q \in \text{lotteries-on carrier} \)
assumes \( \forall x. x \in \text{carrier} \implies \\
\text{measure-pmf}.\text{prob} \ p \ (\text{preferred-alts le} \ x) \geq \text{measure-pmf}.\text{prob} \ q \)
shows \( p \succeq [SD(le)] q \)
(proof)

lemma SD-preorderD:
assumes \( p \succeq [SD(le)] q \)
shows \( p \in \text{lotteries-on carrier} \ q \in \text{lotteries-on carrier} \)
and \( \forall x. x \in \text{carrier} \implies \\
\text{measure-pmf}.\text{prob} \ p \ (\text{preferred-alts le} \ x) \geq \text{measure-pmf}.\text{prob} \ q \)
(proof)

lemma SD-refl' [simp]: \( p \preceq [SD(le)] p \iff p \in \text{lotteries-on carrier} \)
(proof)

lemma SD-is-preorder': preorder-on (lotteries-on carrier) (SD(le))
(proof)

lemma SD-singleton-left:
assumes \( x \in \text{carrier} \ q \in \text{lotteries-on carrier} \)
sshows \( \text{return-pmf} \ x \preceq [SD(le)] q \iff (\forall y \in \text{set-pmf} q. \ x \preceq [le] y) \)
(proof)

lemma SD-singleton-right:
assumes \( x: x \in \text{carrier} \) and \( q: q \in \text{lotteries-on carrier} \)
sshows \( q \preceq [SD(le)] \text{return-pmf} \ x \iff (\forall y \in \text{set-pmf} q. \ y \preceq [le] x) \)
(proof)

lemma SD-strict-singleton-left:
assumes \( x \in \text{carrier} \ q \in \text{lotteries-on carrier} \)
sshows \( \text{return-pmf} \ x \prec [SD(le)] q \iff (\forall y \in \text{set-pmf} q. \ x \preceq [le] y) \land (\exists y \in \text{set-pmf} \ q. \ (x \prec [le] y)) \)
proof

lemma SD-strict-singleton-right:
assumes \( x \in \text{carrier} \) \( q \in \text{lotteries-on carrier} \)
shows \( q \prec ([SD(le)]) \text{return-pmf} x \leftrightarrow (\forall y \in \text{set-pmf} q. \ y \preceq [le] x) \land (\exists y \in \text{set-pmf} q. \ (y \prec [le] x)) \)
(proof)

lemma SD-singleton [simp]:
\( x \in \text{carrier} \implies y \in \text{carrier} \implies \text{return-pmf} x \preceq [SD(le)] \text{return-pmf} y \leftrightarrow x \preceq [le] y \)
(proof)

lemma SD-strict-singleton [simp]:
\( x \in \text{carrier} \implies y \in \text{carrier} \implies \text{return-pmf} x \prec [SD(le)] \text{return-pmf} y \leftrightarrow x \prec [le] y \)
(proof)

end

context pref-profile-wf
begin

context
fixes \( i \) assumes \( i \in \text{agents} \)
begin
interpretation Ri: preorder-on alts \( R i \) (proof)

lemmas SD-singleton-left = Ri.SD-singleton-left
lemmas SD-singleton-right = Ri.SD-singleton-right
lemmas SD-strict-singleton-left = Ri.SD-strict-singleton-left
lemmas SD-strict-singleton-right = Ri.SD-strict-singleton-right
lemmas SD-singleton = Ri.SD-singleton
lemmas SD-strict-singleton = Ri.SD-strict-singleton
end

end

lemmas (in pref-profile-wf) [simp] = SD-singleton SD-strict-singleton

4.2 Stochastic Dominance for preference profiles

context pref-profile-wf
begin

lemma SD-pref-profile:
assumes \( i \in \text{agents} \)
shows \( p \succeq [SD(R i)] q \leftrightarrow p \in \text{lotteries-on alts} \land q \in \text{lotteries-on alts} \land \)

30
\[
(\forall x \in \text{alts} \cdot \text{measure-pmf}\ p (\text{preferred-alts} \ (R \ i) \ x) \geq \\
\text{measure-pmf}\ q (\text{preferred-alts} \ (R \ i) \ x))
\]

\(\langle\text{proof}\rangle\)

**Lemma SD-pref-profileI** [intro?]:
\[\text{assumes } i \in \text{agents } p \in \text{lotteries-on alts} q \in \text{lotteries-on alts} \]
\[\text{assumes } \forall x. \ x \in \text{alts} \implies \\
\text{measure-pmf}\ p (\text{preferred-alts} \ (R \ i) \ x) \geq \\
\text{measure-pmf}\ q (\text{preferred-alts} \ (R \ i) \ x) \]
\[\text{shows } p \succeq_{[SD(R \ i)]} q \]
\(\langle\text{proof}\rangle\)

**Lemma SD-pref-profileD:**
\[\text{assumes } i \in \text{agents } p \succeq_{[SD(R \ i)]} q \]
\[\text{shows } p \in \text{lotteries-on alts} q \in \text{lotteries-on alts} \]
\[\text{and } \forall x. \ x \in \text{alts} \implies \\
\text{measure-pmf}\ p (\text{preferred-alts} \ (R \ i) \ x) \geq \\
\text{measure-pmf}\ q (\text{preferred-alts} \ (R \ i) \ x) \]
\(\langle\text{proof}\rangle\)

**4.3 SD efficient lotteries**

**Definition SD-efficient** :: ('agent', 'alt') pref-profile \(\Rightarrow\) 'alt lottery \(\Rightarrow\) bool where
\[SD\text{-efficient-auxdef}: \]
\[SD\text{-efficient } R \ p \iff \neg(\exists q \in \text{lotteries-on alts} \{x. \exists i. \ R i x x\cdot q \succ [\text{Pareto } (SD \circ R)] p) \]

**Context pref-profile-wf**

**Begin**

**Sublocale** \(SD\): preorder-family agents lotteries-on alts \(SD \circ R\) \(\langle\text{proof}\rangle\)

A lottery is considered SD-efficient if there is no other lottery such that all
agents weakly prefer the other lottery (w.r.t. stochastic dominance) and at
least one agent strongly prefers the other lottery.

**Lemma SD-efficient-def:**
\[SD\text{-efficient } R \ p \iff \neg(\exists q \in \text{lotteries-on alts} \ q \succ [\text{Pareto } (SD \circ R)] p) \]
\(\langle\text{proof}\rangle\)

**Lemma SD-efficient-def':**
\[SD\text{-efficient } R \ p \iff \\
\neg(\exists q \in \text{lotteries-on alts} \ (\forall i \in \text{agents} \ q \succeq [SD(R \ i)] p) \land (\exists i \in \text{agents} \ q \succ [SD(R \ i)] p)) \]
\(\langle\text{proof}\rangle\)

**Lemma SD-inefficientI:**
assumes $q \in \text{lotteries-on alts} \land \forall i. i \in \text{agents} \implies q \succeq [\text{SD}(R i)] p$
shows $\neg \text{SD-efficient } R p$
(proof)

lemma $\text{SD-inefficient}'$:
assumes $q \in \text{lotteries-on alts} \land \forall i. i \in \text{agents} \implies q \succeq [\text{SD}(R i)] p$
shows $\neg \text{SD-efficient } R p$
(proof)

lemma $\text{SD-inefficientE}$:
assumes $\neg \text{SD-efficient } R p$
obtains $q$ where
$q \in \text{lotteries-on alts} \land \forall i. i \in \text{agents} \implies q \succeq [\text{SD}(R i)] p$
and $\exists i \in \text{agents}. q \succ [\text{SD}(R i)] p$
(proof)

lemma $\text{SD-efficientD}$:
assumes $\text{SD-efficient } R p$ $q \in \text{lotteries-on alts}$ and $\forall i. i \in \text{agents} \implies q \succeq [\text{SD}(R i)] p$
shows $\exists i \in \text{agents}. \neg (q \preceq [\text{SD}(R i)] p)$
(proof)

lemma $\text{SD-efficient-singleton-iff}$:
assumes $\text{[simp]}: x \in \text{alts}$
shows $\text{SD-efficient } R (\text{return-pmf } x) \iff x \notin \text{pareto-losers } R$
(proof)

end

4.4 Equivalence proof

We now show that a lottery is preferred w.r.t. Stochastic Dominance iff it yields more expected utility for all compatible utility functions.

context $\text{finite-total-preorder-on}$
begin
abbreviation $\text{is-vnm-utility} \equiv \text{vnm-utility } \text{carrier } \text{le}$

lemma $\text{utility-weak-ranking-index}$:
$\text{is-vnm-utility} (\lambda x. \text{real} (\text{length } (\text{weak-ranking } \text{le}) - \text{weak-ranking-index } x))$
(proof)

lemma $\text{SD-iff-expected-utilities-le}$:
assumes $p \in \text{lotteries-on carrier } q \in \text{lotteries-on carrier}$
shows $p \succeq [\text{SD}(\text{le})] q \iff$
$(\forall u. \text{is-vnm-utility } u \rightarrow \text{measure-pmf.expectation } p u \leq \text{measure-pmf.expectation } q u)$
lemma not-strict-SD-iff:
  assumes \( p \in \text{lotteries-on carrier} \ q \in \text{lotteries-on carrier} \)
  shows \( \neg(p \prec [\text{SD(le)}] q) \iff \)
    \((\exists \ u. \ \text{is-vnm-utility} \ u \land \text{measure-pmf.expectation} \ q \ u \leq \text{measure-pmf.expectation} \ p \ u)\)
  \(\langle \text{proof} \rangle\)

lemma strict-SD-iff:
  assumes \( p \in \text{lotteries-on carrier} \ q \in \text{lotteries-on carrier} \)
  shows \( (p \prec [\text{SD(le)}] q) \iff \)
    \((\forall \ u. \ \text{is-vnm-utility} \ u \rightarrow \text{measure-pmf.expectation} \ p \ u < \text{measure-pmf.expectation} \ q \ u)\)
  \(\langle \text{proof} \rangle\)

end
end

theory SD-Efficiency
imports Complex-Main Preference-Profiles Lotteries Stochastic-Dominance
begin

context pref-profile-wf
begin

lemma SD-inefficient-support-subset:
  assumes \( \text{inefficient} : \neg\text{SD-efficient} \ R \ p' \)
  assumes \( \text{support} : \text{set-pmf} \ p' \subseteq \text{set-pmf} \ p \)
  assumes \( \text{lotteries} : \ p \in \text{lotteries-on alts} \)
  shows \( \neg\text{SD-efficient} \ R \ p \)
  \(\langle \text{proof} \rangle\)

lemma SD-efficient-support-subset:
  assumes \( \text{SD-efficient} \ R \ p \text{ set-pmf} \ p' \subseteq \text{set-pmf} \ p \ p \in \text{lotteries-on alts} \)
  shows \( \text{SD-efficient} \ R \ p' \)
  \(\langle \text{proof} \rangle\)

lemma SD-efficient-same-support:
  assumes \( \text{set-pmf} \ p = \text{set-pmf} \ p' \ p \in \text{lotteries-on alts} \)
  shows \( \text{SD-efficient} \ R \ p \iff \text{SD-efficient} \ R \ p' \)
  \(\langle \text{proof} \rangle\)

lemma SD-efficient-iff:
assumes \( p \in \text{lotteries-on alts} \)
shows \( SD\text{-efficient } R \ p \leftrightarrow SD\text{-efficient } R \ (\text{pmf-of-set } (\text{set-pmf } p)) \)

(\textbf{proof})

\textbf{lemma} \( SD\text{-efficient-no-pareto-loser:} \)
\hspace{1em} assumes \( \text{efficient: } SD\text{-efficient } R \ p \text{ and } p\text{-wf: } p \in \text{lotteries-on alts} \)
\hspace{1em} shows \( \text{set-pmf } p \cap \text{pareto-losers } R = \{\} \)

(\textbf{proof})

Given two lotteries with the same support where one is strictly Pareto-SD-preferred to the other, one can construct a third lottery that is weakly Pareto-SD-preferred to the better lottery (and therefore strictly Pareto-SD-preferred to the worse lottery) and whose support is a strict subset of the original supports.

\textbf{lemma} \( \text{improve-lottery-support-subset:} \)
\hspace{1em} assumes \( p \in \text{lotteries-on alts} \ q \in \text{lotteries-on alts} \ q \succ [\text{Pareto}(SD \circ R)] \ p \)
\hspace{1em} set-pmf \( p = \text{set-pmf } q \)
\hspace{1em} obtains \( r \) where \( r \in \text{lotteries-on alts} \ r \succeq [\text{Pareto}(SD \circ R)] \ q \text{ set-pmf } r \subset \text{set-pmf } p \)

(\textbf{proof})

4.5 \textbf{Existence of SD-efficient lotteries}

In this section, we will show that any lottery can be ‘improved’ to an SD-efficient lottery, i.e. for any lottery, there exists an SD-efficient lottery that is weakly SD-preferred to the original one by all agents.

context
\hspace{1em} fixes \( p :: \text{alt lottery} \)
\hspace{1em} assumes \( lott: p \in \text{lotteries-on alts} \)

begin

private definition \( \text{improve-lottery } :: \text{alt lottery } \Rightarrow \text{alt lottery } \) where
\( \text{improve-lottery } q = \) (let \( A = \{r \in \text{lotteries-on alts}. \ r \succ [\text{Pareto}(SD \circ R)] \ q\} \) in
\( \text{SOME } r. \ r \in A \land \lnot (\exists r' \in A. \text{set-pmf } r' \subset \text{set-pmf } r)) \)

private lemma \( \text{improve-lottery:} \)
\hspace{1em} assumes \( \lnot SD\text{-efficient } R \ q \)
\hspace{1em} defines \( r \equiv \text{improve-lottery } q \)
\hspace{1em} shows \( r \in \text{lotteries-on alts} \ q \succ [\text{Pareto}(SD \circ R)] \ q \)
\hspace{1em} \( \bigwedge r', r' \in \text{lotteries-on alts} \ r' \succ [\text{Pareto}(SD \circ R)] \ q \implies \neg (\text{set-pmf } r' \subset \text{set-pmf } r) \)

(\textbf{proof}) \textbf{fun} \( \text{sd-chain } :: \text{nat } \Rightarrow \text{alt lottery option } \) where
\( \text{sd-chain } 0 = \text{Some } p \)
\( \mid \text{sd-chain } (\text{Suc } n) = \)
\( \text{case sd-chain } n \text{ of} \)
\hspace{1em} \( \text{None} \Rightarrow \text{None} \)
\hspace{1em} \( \mid \text{Some } p \Rightarrow \text{if } SD\text{-efficient } R \ p \text{ then } \text{None else Some } \) (improve-lottery \( p)) \)

34
private lemma sd-chain-None-propagate:
\( m \geq n \Rightarrow \text{sd-chain} \ n = \text{None} \Rightarrow \text{sd-chain} \ m = \text{None} \)
⟨proof⟩ lemma sd-chain-Some-propagate:
\( m \geq n \Rightarrow \text{sd-chain} \ m = \text{Some} \ q \Rightarrow \exists q'. \text{sd-chain} \ n = \text{Some} \ q' \)
⟨proof⟩ lemma sd-chain-NoneD:
\( \text{sd-chain} \ n = \text{None} \Rightarrow \exists n \ p. \text{sd-chain} \ n = \text{Some} \ p \land \text{SD-efficient} \ R \ p \)
⟨proof⟩ lemma sd-chain-lottery: sd-chain n = Some q \( \Rightarrow \exists q' \in \text{lotteries-on alts} \)
⟨proof⟩ lemma sd-chain-Suc:
assumes sd-chain m = Some q
assumes sd-chain (Suc m) = Some r
shows \( q \prec \text{Pareto} (\text{SD} \circ R) \) r
⟨proof⟩ lemma sd-chain-strictly-preferred:
assumes \( m < n \)
assumes sd-chain m = Some q
assumes sd-chain n = Some s
shows \( q \prec \text{Pareto} (\text{SD} \circ R) \) s
⟨proof⟩ lemma sd-chain-preferred:
assumes \( m \leq n \)
assumes sd-chain m = Some q
assumes sd-chain n = Some s
shows \( q \succeq \text{Pareto} (\text{SD} \circ R) \) s
⟨proof⟩

lemma SD-efficient-lottery-exists:
\( \text{obtains} \ q \ \text{where} \ q \in \text{lotteries-on alts} \ q \succeq \text{Pareto} (\text{SD} \circ R) \) r \ Q SD-efficient R \ q
⟨proof⟩

end

lemma
assumes \( p \in \text{lotteries-on alts} \)
shows \( \exists q \in \text{lotteries-on alts}. \ q \succeq \text{Pareto} (\text{SD} \circ R) \) p \ Q SD-efficient R \ q
⟨proof⟩

end

end

5 Social Decision Schemes

theory Social-Decision-Schemes
imports
  Complex-Main
  HOL-Probability
  Probability
  Preference-Profiles
  Elections
  Order-Predicates
  Stochastic-Dominance
  SD-Efficiency
5.1 Basic Social Choice definitions

context election
begin

The set of lotteries, i.e. the probability mass functions on the type \( \text{'alt} \) whose support is a subset of the alternative set.

abbreviation lotteries where
lotteries \( \equiv \) lotteries-on alts

The probability that a lottery returns an alternative that is in the given set

abbreviation lottery-prob \( \vdash \text{'alt lottery} \Rightarrow \text{'alt set} \Rightarrow \) real where
lottery-prob \( \equiv \) measure-pmf.prob

lemma lottery-prob-alts-superset:
assumes \( p \in \text{lotteries alts} \subseteq A \)
shows \( \text{lottery-prob} \ p \ A = 1 \)
⟨proof⟩

lemma lottery-prob-alts: \( p \in \text{lotteries} \implies \text{lottery-prob} \ p \ \text{alts} = 1 \)
⟨proof⟩

end

In the context of an election, a preference profile is a function that assigns to each agent her preference relation (which is a total preorder)

5.2 Social Decision Schemes

In the context of an election, a Social Decision Scheme (SDS) is a function that maps preference profiles to lotteries on the alternatives.

locale social-decision-scheme = election agents alts
for agents :: \text{'agent set} and alts :: \text{'alt set} +
fixes sds :: (\text{'agent, \text{'alt}) pref-profile \Rightarrow \text{'alt lottery}
assumes sds-uf: is-pref-profile R \Rightarrow sds R \in \text{lotteries}

5.3 Anonymity

An SDS is anonymous if permuting the agents in the input does not change the result.

locale anonymous-sds = social-decision-scheme agents alts sds
for agents :: \text{'agent set} and alts :: \text{'alt set} and sds +
assumes anonymous: \( \pi \) permutes agents \( \Rightarrow \) is-pref-profile \( R \Rightarrow sds \ (R \circ \pi) = sds R \)
begin


lemma anonymity-prefs-from-table:
  assumes prefs-from-table-wf agents alts xs prefs-from-table-wf agents alts ys
  assumes mset (map snd xs) = mset (map snd ys)
  shows  sds (prefs-from-table xs) = sds (prefs-from-table ys)
⟨proof⟩
context
begin
qualified lemma anonymity-prefs-from-table-aux:
  assumes R1 = prefs-from-table xs prefs-from-table-wf agents alts xs
  assumes R2 = prefs-from-table ys prefs-from-table-wf agents alts ys
  assumes mset (map snd xs) = mset (map snd ys)
  shows  sds R1 = sds R2 ⟨proof⟩
end

5.4 Neutrality

An SDS is neutral if permuting the alternatives in the input does not change the result, modulo the equivalent permutation in the output lottery.

locale neutral-sds =
  social-decision-scheme agents alts sds
  for agents :: 'agent set and alts :: 'alt set and sds +
  assumes neutral: σ permutes alts → is-pref-profile R →
  sds (permute-profile σ R) = map-pmf σ (sds R)
begin
Alternative formulation of neutrality that shows that our definition is equivalent to that in the paper.
lemma neutral':
  assumes σ permutes alts
  assumes is-pref-profile R
  assumes a ∈ alts
  shows  pmf (sds (permute-profile σ R)) (σ a) = pmf (sds R) a
⟨proof⟩
end

locale an-sds =
  anonymous-sds agents alts sds + neutral-sds agents alts sds
  for agents :: 'agent set and alts :: 'alt set and sds
begin
lemma sds-anonymous-neutral:
  assumes perm: σ permutes alts and wf: is-pref-profile R1 is-pref-profile R2
  assumes eq: anonymous-profile R1 =
lemma \textsf{sds-anonymous-neutral'}:
\begin{itemize}
  \item \textbf{assumes} \textit{perm: }\sigma \text{ permutes alts and } \textit{wf: }\text{is-pref-profile }R_1 \text{ is-pref-profile }R_2
  \item \textbf{assumes} \textit{eq: }\text{anonymous-profile }R_1 = \text{\textit{image-mset \textit{(map \textit{((\textit{\Delta}) \sigma)) \text{ (anonymous-profile }R_2)}}}}
\end{itemize}
\begin{itemize}
  \item \textbf{shows} \textit{pmf \textit{(sds }R_1\text{)} (\sigma x) = pmf \textit{(sds }R_2\text{)} x}
\end{itemize}
\langle \textit{proof} \rangle

\begin{itemize}
\end{itemize}

lemma \textsf{sds-automorphism}:
\begin{itemize}
  \item \textbf{assumes} \textit{perm: }\sigma \text{ permutes alts and } \textit{wf: }\text{is-pref-profile }R
  \item \textbf{assumes} \textit{eq: }\text{\textit{image-mset \textit{(map \textit{((\textit{\Delta}) \sigma)) \text{ (anonymous-profile }R)}}}} = \text{anonymous-profile }R
\end{itemize}
\begin{itemize}
  \item \textbf{shows} \textit{pmf \textit{(sds }R\text{)} x = pmf \textit{(sds }R\text{)} y}
\end{itemize}
\langle \textit{proof} \rangle

5.5 Ex-post efficiency

locale \textit{ex-post-efficient-sds} = \textit{social-decision-scheme} \textit{agents alts sds}
\begin{itemize}
  \item for \textit{agents :: 'agent set and alts :: 'alt set and sds +}
  \item \textbf{assumes} \textit{ex-post-efficient:}
    \begin{itemize}
      \item \textit{is-pref-profile }R \Rightarrow \textit{set-pmf \textit{(sds }R\text{)} \cap \text{pareto-losers }R = \{\}}
    \end{itemize}
\end{itemize}
\begin{itemize}
\end{itemize}

begin

lemma \textit{ex-post-efficient'}:
\begin{itemize}
  \item \textbf{assumes} \textit{is-pref-profile }R \text{ }y \succ [\text{Pareto}(R)] x
  \item \textbf{shows} \textit{pmf \textit{(sds }R\text{)} x = 0}
\end{itemize}
\langle \textit{proof} \rangle

\begin{itemize}
\end{itemize}

lemma \textit{ex-post-efficient''}:
\begin{itemize}
  \item \textbf{assumes} \textit{is-pref-profile }R \text{ }i \in \text{agents } \forall i \in \text{agents. } y \succ [R i] x \dashv y \preceq [R i] x
  \item \textbf{shows} \textit{pmf \textit{(sds }R\text{)} x = 0}
\end{itemize}
\langle \textit{proof} \rangle
5.6 SD efficiency

An SDS is SD-efficient if it returns an SD-efficient lottery for every preference profile, i.e. if the SDS outputs a lottery, it is never the case that there is another lottery that is weakly preferred by all agents and strictly preferred by at least one agent.

locale sd-efficient-sds = social-decision-scheme agents alts sds
for agents :: 'agent set and alts :: 'alt set and sds +
asumes SD-efficient: is-pref-profile R \(\implies\) SD-efficient R (sds R)
begin

An alternative formulation of SD-efficiency that is somewhat more convenient to use.

lemma SD-efficient':
  assumes is-pref-profile R q ∈ lotteries
  assumes \(\forall i. i ∈ agents \implies q ⪰ [SD(R i)] sds R i ∈ agents q ⪰ [SD(R i)] sds R\)
sows \(P\)
⟨proof⟩

Any SD-efficient SDS is also ex-post efficient.

sublocale ex-post-efficient-sds
⟨proof⟩

The following rule can be used to derive facts from inefficient supports: If a set of alternatives is an inefficient support, at least one of the alternatives in it must receive probability 0.

lemma SD-inefficient-support:
  assumes A: A ≠ {} A ⊆ alts and inefficient: ¬SD-efficient R (pmf-of-set A)
  assumes wf: is-pref-profile R
  shows \(\exists x ∈ A. pmf (sds R) x = 0\)
⟨proof⟩

lemma SD-inefficient-support':
  assumes wf: is-pref-profile R
  assumes A: A ≠ {} A ⊆ alts and
  wit: \(p ∈ lotteries \forall i ∈ agents. p ⪰ [SD(R i)] pmf-of-set A i ∈ agents\)
  shows \(\exists x ∈ A. pmf (sds R) x = 0\)
⟨proof⟩
end

5.7 Weak strategyproofness

context social-decision-scheme
The SDS is said to be manipulable for a particular preference profile, a particular agent, and a particular alternative preference ordering for that agent if the lottery obtained if the agent submits the alternative preferences strictly SD-dominates that obtained if the original preferences are submitted. (SD-dominated w.r.t. the original preferences)

**definition** manipulable-profile

:: ('agent', 'alt') pref-profile ⇒ 'agent ⇒ 'alt relation ⇒ bool

where

manipulable-profile R i Ri’ ←→ sds (R(i := Ri’)) ≻[SD (R i)] sds R

An SDS is weakly strategyproof (or just strategyproof) if it is not manipulable for any combination of preference profiles, agents, and alternative preference relations.

**locale** strategyproof-sds = social-decision-scheme agents alts sds

for agents :: 'agent set and alts :: 'alt set and sds +

assumes strategyproof:

is-pref-profile R ⇒ i ∈ agents ⇒ total-preorder-on alts Ri' ⇒ ¬manipulable-profile R i Ri'

### 5.8 Strong strategyproofness

**context** social-decision-scheme

**begin**

The SDS is said to be strongly strategyproof for a particular preference profile, a particular agent, and a particular alternative preference ordering for that agent if the lottery obtained if the agent submits the alternative preferences is SD-dominated by the one obtained if the original preferences are submitted. (SD-dominated w.r.t. the original preferences)

In other words: the SDS is strategyproof w.r.t the preference profile R and the agent i and the alternative preference relation R’i if the lottery for obtained for R is at least as good for i as the lottery obtained when i misrepresents her preferences as R’i.

**definition** strongly-strategyproof-profile

:: ('agent', 'alt') pref-profile ⇒ 'agent ⇒ 'alt relation ⇒ bool

where

strongly-strategyproof-profile R i Ri’ ←→ sds R ≳[SD (R i)] sds (R(i := Ri'))

**lemma** strongly-strategyproof-profileI [intro]:

assumes is-pref-profile R total-preorder-on alts Ri’ i ∈ agents

assumes \( \forall x. x ∈ alts ⇒ lottery-prob (sds (R(i := Ri'))) (preferred-alts (R i) x) \) \( \leq lottery-prob (sds R) (preferred-alts (R i) x) \)

shows strongly-strategyproof-profile R i Ri’

⟨proof⟩
lemma strongly-strategyproof-imp-not-manipulable:
assumes strongly-strategyproof-profile R i Ri'
shows ¬manipulable-profile R i Ri'
(proof)
end

An SDS is strongly strategyproof if it is strongly strategyproof for all combinations of preference profiles, agents, and alternative preference relations.

locale strongly-strategyproof-sds = social-decision-scheme agents alts sds
for agents :: 'agent set and alts :: 'alt set and sds +
assumes strongly-strategyproof:
  is-pref-profile R ⇒ i ∈ agents ⇒ total-preorder-on alts Ri' ⇒
  strongly-strategyproof-profile R i Ri'
begin
Any SDS that is strongly strategyproof is also weakly strategyproof.

sublocale strategyproof-sds
(proof)
end

locale strategyproof-an-sds =
  strategyproof-sds agents alts sds + an-sds agents alts sds
for agents :: 'agent set and alts :: 'alt set and sds
end

6 Lowering Social Decision Schemes

theory SDS-Lowering
imports Social-Decision-Schemes
begin

definition lift-pref-profile ::
  'agent set ⇒ 'alt set ⇒ 'agent set ⇒ 'alt set ⇒
  ('agent, 'alt) pref-profile ⇒ ('agent, 'alt) pref-profile
where
lift-pref-profile agents alts agents' alts' R = (λi x y.
  x ∈ alts' ∧ y ∈ alts' ∧ i ∈ agents' ∧
  (x = y ∨ x /∈ alts ∨ i /∈ agents ∨ (y ∈ alts ∧ R i x y)))

lemma lift-pref-profile-wf:
assumes pref-profile-wf agents alts R
assumes agents ⊆ agents' alts ⊆ alts' finite alts'
defines R' ≡ lift-pref-profile agents alts agents' alts' R
shows pref-profile-wf agents' alts' R'
lemma lift-pref-profile-permute-agents:
  assumes $\pi$ permutes agents agents $\subseteq$ agents'
  shows lift-pref-profile agents alts agents' alts' $(R \circ \pi) =$
  lift-pref-profile agents alts agents' alts' $R \circ \pi$
⟨proof⟩

lemma lift-pref-profile-permute-alts:
  assumes $\sigma$ permutes alts alts $\subseteq$ alts'
  shows lift-pref-profile agents alts agents' alts' $(permute-profile \sigma R) =$
  permute-profile $\sigma$ (lift-pref-profile agents alts agents' alts' $R$)
⟨proof⟩

lemma lotteries-on-subset: $A \subseteq B \implies p \in \text{lotteries-on } A \implies p \in \text{lotteries-on } B$
⟨proof⟩

lemma lottery-prob-carrier: $p \in \text{lotteries-on } A \implies \text{measure-pmf.prob } p \ A = 1$
⟨proof⟩

context
  fixes agents alts $R$ agents' alts' $R'$
  assumes $R$-uf: pref-profile-uf agents alts $R$
  assumes election: agents $\subseteq$ agents' alts $\subseteq$ alts' alts $\neq \{\}$ agents $\neq \{\}$ finite alts'
  defines $R' \equiv \text{lift-pref-profile agents alts agents' alts' } R$
begin
interpretation $R$: pref-profile-uf agents alts $R$ (proof)
interpretation $R'$: pref-profile-uf agents' alts' $R'$ (proof)

lemma lift-pref-profile-strict-iff:
  $x \prec ([Pareto(R')]) y \iff i \in \text{agents } \land (y \in \text{alts } \land x \in \text{alts'} \setminus \text{alts}) \lor x \prec [R i] y$
⟨proof⟩

lemma preferred-alts-lift-pref-profile:
  assumes $i: i \in \text{agents'} \land x: x \in \text{alts'}$
  shows preferred-alts $(R' i) x =$
  (if $i \in \text{agents } \land x \in \text{alts}$ then preferred-alts $(R i) x$ else alts')
⟨proof⟩

lemma lift-pref-profile-Pareto-iff:
  $x \preceq [\text{Pareto}(R')] y \iff x \in \text{alts'} \land y \in \text{alts'} \land (x \not\in \text{alts } \lor x \preceq [\text{Pareto}(R)] y)$
⟨proof⟩

lemma lift-pref-profile-Pareto-strict-iff:
  $x \prec [\text{Pareto}(R')] y \iff x \in \text{alts'} \land y \in \text{alts'} \land (x \not\in \text{alts } \land y \in \text{alts } \lor x$
lemma pareto-losers-lift-pref-profile:
  shows pareto-losers $R' = \text{pareto-losers } R \cup (\text{alts}' - \text{alts})$

(lemma)

context begin
private lemma lift-SD-iff-agent:
  assumes $p \in \text{lotteries-on alts } q \in \text{lotteries-on alts } \text{and } i: i \in \text{agents}$
  shows $p \preceq [\text{SD}(R' i)] q \iff p \preceq [\text{SD}(R i)] q$
(lemma)
lemma lift-SD-iff-nonagent:
  assumes $p \in \text{lotteries-on alts } q \in \text{lotteries-on alts } \text{and } i: i \in \text{agents'} - \text{agents}$
  shows $p \preceq [\text{SD}(R' i)] q$
(lemma)
lemmas lift-SD-iff = lift-SD-iff-agent lift-SD-iff-nonagent

lemma lift-SD-iff':
  $p \in \text{lotteries-on alts } q \in \text{lotteries-on alts } \implies i \in \text{agents'} \implies$
  $p \preceq [\text{SD}(R' i)] q \iff i \notin \text{agents } \lor p \preceq [\text{SD}(R i)] q$
(lemma)

end

lemma lift-Pareto-SD-iff:
  assumes $p \in \text{lotteries-on alts } q \in \text{lotteries-on alts}$
  shows $p \preceq [\text{Pareto}(\text{SD} \circ R')] q \iff p \preceq [\text{Pareto}(\text{SD} \circ R)] q$
(lemma)

lemma lift-Pareto-SD-strict-iff:
  assumes $p \in \text{lotteries-on alts } q \in \text{lotteries-on alts}$
  shows $p \prec [\text{Pareto}(\text{SD} \circ R')] q \iff p \prec [\text{Pareto}(\text{SD} \circ R)] q$
(lemma)

lemma lift-SD-efficient-iff:
  assumes $p: p \in \text{lotteries-on alts}$
  shows $\text{SD-efficient } R' p \iff \text{SD-efficient } R p$
(lemma)

end
locale sds-lowering = 
ex-post-efficient-sds agents alts sds
for agents :: 'agent set and alts :: 'alt set and sds +
fixes agents' alts'
assumes agents'-subset: agents' ⊆ agents and alts'-subset: alts' ⊆ alts
and agents'-nonempty [simp]: agents' ≠ {} and alts'-nonempty [simp]: alts'
≠ {}
begin

lemma finite-agents' [simp]: finite agents'
⟨proof⟩

lemma finite-alts' [simp]: finite alts'
⟨proof⟩

abbreviation lift :: ('agent, 'alt) pref-profile ⇒ ('agent, 'alt) pref-profile where
lift ≡ lift-pref-profile agents' alts' agents alts

definition lowered :: ('agent, 'alt) pref-profile ⇒ 'alt lottery where
lowered = sds ◦ lift

lemma lift-wf [simp, intro]:
pref-profile-wf agents' alts' R ⇒ is-pref-profile (lift R)
⟨proof⟩

sublocale lowered: election agents' alts'
⟨proof⟩

lemma preferred-alts-lift:
lowered.is-pref-profile R ⇒ i ∈ agents ⇒ x ∈ alts ⇒
preferred-alts (lift R i) x =
(if i ∈ agents' ∧ x ∈ alts' then preferred-alts (R i) x else alts)
⟨proof⟩

lemma pareto-losers-lift:
lowered.is-pref-profile R ⇒ pareto-losers (lift R) = pareto-losers R ∪ (alts − alts')
⟨proof⟩

lemma lowered-lotteries: lowered.lotteries ⊆ lotteries
⟨proof⟩

sublocale lowered: social-decision-scheme agents' alts' lowered
⟨proof⟩

sublocale ex-post-efficient-sds agents' alts' lowered
⟨proof⟩

lemma lowered-in-lotteries [simp]: lowered.is-pref-profile R ⇒ lowered R ∈ lotteries

locale sds-lowering-anonymous =
  anonymous-sds agents alts sds +
  sds-lowering agents alts sds agents' alts'
  for agents :: 'agent set and alts :: 'alt set and sds agents' alts'
begin

sublocale lowered: anonymous-sds agents' alts' lowered
⟨proof⟩
end

locale sds-lowering-neutral =
  neutral-sds agents alts sds +
  sds-lowering agents alts sds agents' alts'
  for agents :: 'agent set and alts :: 'alt set and sds agents' alts'
begin

sublocale lowered: neutral-sds agents' alts' lowered
⟨proof⟩
end

locale sds-lowering-sd-efficient =
  sd-efficient-sds agents alts sds +
  sds-lowering agents alts sds agents' alts'
  for agents :: 'agent set and alts :: 'alt set and sds agents' alts'
begin

sublocale sd-efficient-sds agents' alts' lowered
⟨proof⟩
end

locale sds-lowering-strategyproof =
  strategyproof-sds agents alts sds +
  sds-lowering agents alts sds agents' alts'
  for agents :: 'agent set and alts :: 'alt set and sds agents' alts'
begin

sublocale strategyproof-sds agents' alts' lowered
⟨proof⟩

45
locale sds-lowering-anonymous-neutral-sd-efficient-strategyproof =
  sds-lowering-anonymous + sds-lowering-neutral +
  sds-lowering-sd-efficient + sds-lowering-strategyproof
end

7 Random Dictatorship

theory Random-Dictatorship
imports
  Complex-Main
  Social-Decision-Schemes
begin
We define Random Dictatorship as a social decision scheme on total pre-
orders (i.e. agents are allowed to have ties in their rankings) by first se-
lecting an agent uniformly at random and then selecting one of that agents’
most preferred alternatives uniformly at random. Note that this definition
also works for weak preferences.
definition random-dictatorship :: 'agent set ⇒ 'alt set ⇒ ('agent, 'alt) pref-profile ⇒ 'alt lottery where
  random-dictatorship-auxdef:
  random-dictatorship agents alts R =
  do {
    i ← pmf-of-set agents;
    pmf-of-set (Max-wrt-among (R i) alts)
  }
context election
begin
abbreviation RD :: ('agent, 'alt) pref-profile ⇒ 'alt lottery where
  RD ≡ random-dictatorship agents alts
lemma random-dictatorship-def:
  assumes is-pref-profile R
  shows RD R =
  do {
    i ← pmf-of-set agents;
    pmf-of-set (favorites R i)
  }
⟨proof⟩
lemma random-dictatorship-unique-favorites:
  assumes is-pref-profile R has-unique-favorites R
shows $RD R = \text{map-pmf} (\text{favorite } R) (\text{pmf-of-set } \text{agents})$

⟨proof⟩

lemma random-dictatorship-unique-favorites':
  assumes is-pref-profile $R$ has-unique-favorites $R$
  shows $RD R = \text{pmf-of-multiset} (\text{image-mset} (\text{favorite } R) (\text{mset-set } \text{agents}))$

⟨proof⟩

lemma pmf-random-dictatorship:
  assumes is-pref-profile $R$
  shows $\text{pmf} (RD R) x = \frac{\left(\sum_{i\in \text{agents}.} \text{indicator} (\text{favorites } R i) x \right)}{\text{real} (\text{card} (\text{favorites } R i))} / \text{real} (\text{card } \text{agents})$

⟨proof⟩

sublocale RD: social-decision-scheme agents alts RD
⟨proof⟩

We now show that Random Dictatorship fulfils anonymity, neutrality, and strong strategyproofness. At the very least, this shows that the definitions of these notions are consistent.

7.1 Anonymity

The following proof is essentially the following: In Random Dictatorship, permuting the agents in the preference profile is the same as applying the permutation to the agent that was picked uniformly at random in the first step. However, uniform distributions are invariant under permutation, therefore the outcome is totally unchanged.

sublocale RD: anonymous-sds agents alts RD
⟨proof⟩

7.2 Neutrality

The proof of neutrality is similar to that of anonymity. We have proven elsewhere that the most preferred alternatives of an agent in a profile with permuted alternatives are simply the image of the originally preferred alternatives. Since we pick one alternative from the most preferred alternatives of the selected agent uniformly at random, this means that we effectively pick an agent, then pick one of her most preferred alternatives, and then apply the permutation to that alternative, which is simply Random Dictatorship transformed with the permutation.

sublocale RD: neutral-sds agents alts RD
⟨proof⟩
7.3 Strong strategyproofness

The argument for strategyproofness is quite simple: Since the preferences submitted by an agent $i$ only influence the outcome when that agent is picked in the first process, it suffices to focus on this case. When the agent $i$ submits her true preferences, the probability of obtaining a result at least as good as $x$ (for any alternative $x$) is 1, since the outcome will always be one of her most-preferred alternatives. Obviously, the probability of obtaining such a result cannot exceed 1 no matter what preferences she submits instead, and thus, RD is strategyproof.

sublocale RD: strongly-strategyproof-sds agents alts RD
⟨proof⟩
end
end

8 Random Serial Dictatorship

theory Random-Serial-Dictatorship
imports
  Complex-Main
  Social-Decision-Schemes
  Random-Dictatorship
begin

Random Serial Dictatorship is an anonymous, neutral, strongly strategy-proof, and ex-post efficient Social Decision Scheme that extends Random Dictatorship to the domain of weak preferences.

We define RSD using a fold over a random permutation. Effectively, we choose a random order of the agents (in the form of a list) and then traverse that list from left to right, where each agent in turn removes all the alternatives that are not top-ranked among the remaining ones.

definition random-serial-dictatorship ::
  'agent set ⇒ 'alt set ⇒ ('agent, 'alt) pref-profile ⇒ 'alt lottery where
random-serial-dictatorship agents alts $R =$
  fold-bind-random-permutation ($\lambda i$ alts. Max-wrt-among ($R i$) alts) pmf-of-set
alts agents

The following two facts correspond give an alternative recursive definition to the above definition, which uses random permutations and list folding.

lemma random-serial-dictatorship-empty [simp]:
  random-serial-dictatorship {} alts $R =$ pmf-of-set alts
⟨proof⟩
We define the RSD winners w.r.t. a given set of alternatives and a fixed permutation (i.e. list) of agents. In contrast to the above definition, the RSD winners are determined by traversing the list of agents from right to left. This may seem strange, but it makes induction much easier, since induction over \texttt{foldr} does not require generalisation over the set of alternatives and is therefore much easier than over \texttt{foldl}.

**Definition** \texttt{rsd-winners} where
\[
\texttt{rsd-winners } R \texttt{ alts agents} = \texttt{foldr} \left( \lambda i \texttt{ alts}. \texttt{Max-wrt-among} \ (R \ i \ \texttt{alts}) \ R \right) \texttt{agents alts}
\]

**Lemma** \texttt{rsd-winners-empty} [simp]: \texttt{rsd-winners } R \texttt{ alts } \texttt{[]} = \texttt{alts}

**Lemma** \texttt{rsd-winners-Cons} [simp]:
\[
\texttt{rsd-winners } R \texttt{ alts } (i \neq \texttt{agents}) = \texttt{Max-wrt-among} \ (R \ i \ \texttt{alts}) \ (\texttt{rsd-winners } R \texttt{ alts agents})
\]

**Lemma** \texttt{rsd-winners-map} [simp]:
\[
\texttt{rsd-winners } R \texttt{ alts } (\texttt{map } f \texttt{ agents}) = \texttt{rsd-winners} \ (R \circ f) \texttt{ alts agents}
\]

There is now another alternative definition of RSD in terms of the RSD winners. This will mostly be used for induction.

**Lemma** \texttt{random-serial-dictatorship-altdef}:
\[
\text{assumes finite agents}
\]
\[
\text{shows random-serial-dictatorship } \texttt{agents alts } R = \texttt{do } \{ \texttt{agents’ } \leftarrow \texttt{pmf-of-set} \ (\texttt{permutations-of-set} \texttt{agents}); \texttt{pmf-of-set} \ (\texttt{rsd-winners } R \texttt{ alts agents’}) \}
\]

The following lemma shows that folding from left to right yields the same distribution. This is probably the most commonly used definition in the literature, along with the recursive one.

**Lemma** \texttt{random-serial-dictatorship-foldl}:
\[
\text{assumes finite agents}
\]
\[
\text{shows random-serial-dictatorship } \texttt{agents alts } R = \texttt{do } \{ \texttt{random-serial-dictatorship alts R = } \}
\]

49
agents' ← pmf-of-set (permutations-of-set agents);
    pmf-of-set (foldl (λalts i. Max-wrt-among (R i) alts) alts agents')

⟨proof⟩

8.1 Auxiliary facts about RSD

8.1.1 Pareto-equivalence classes

First of all, we introduce the auxiliary notion of a Pareto-equivalence class. A non-empty set of alternatives is a Pareto equivalence class if all agents are indifferent between all alternatives in it, and if some alternative \( x \) is contained in the set, any other alternative \( y \) is contained in it if and only if, to all agents, \( y \) is at least as good as \( x \). The importance of this notion lies in the fact that the set of RSD winners is always a Pareto-equivalence class, which we will later use to show ex-post efficiency and strategy-proofness.

definition RSD-pareto-eqclass where
    RSD-pareto-eqclass agents alts R A ←→
        A ≠ {} ∧ A ⊆ alts ∧ (∀ x∈A. ∀ y∈alts. y ∈ A ←→ (∀ i∈agents. R i x y))

lemma RSD-pareto-eqclassI:
    assumes
        A ≠ {} A ⊆ alts \( ∧ \) (∀ x∈A. y ∈ alts y ∈ A ←→ (∀ i∈agents. R i x y))
    shows
        RSD-pareto-eqclass agents alts R A
    ⟨proof⟩

lemma RSD-pareto-eqclassD:
    assumes
        RSD-pareto-eqclass agents alts R A
    shows
        A ≠ {} A ⊆ alts \( ∧ \) (∀ x∈A. y ∈ alts y ∈ A ←→ (∀ i∈agents. R i x y))
    ⟨proof⟩

lemma RSD-pareto-eqclass-indiff-set:
    assumes
        RSD-pareto-eqclass agents alts R A i ∈ agents x ∈ A y ∈ A
    shows
        R i x y
    ⟨proof⟩

lemma RSD-pareto-eqclass-empty [simp, intro!]:
    alts ≠ {} \( → \) RSD-pareto-eqclass {} alts R alts
    ⟨proof⟩

lemma (in pref-profile-wf) RSD-pareto-eqclass-insert:
    assumes
        RSD-pareto-eqclass agents' alts R finite alts
        i ∈ agents agents' ⊆ agents
    shows
        RSD-pareto-eqclass (insert i agents') alts R (Max-wrt-among (R i) A)
    ⟨proof⟩

50
8.1.2 Facts about RSD winners

context pref-profile-wf

begin

Any RSD winner is a valid alternative.

lemma rsd-winners-subset:
  assumes set agents' ⊆ agents
  shows rsd-winners R alts' agents' ⊆ alts'
  ⟨proof⟩

There is always at least one RSD winner.

lemma rsd-winners-nonempty:
  assumes finite: finite alts and alts' ≠ {} set agents' ⊆ agents alts' ⊆ alts
  shows rsd-winners R alts' agents' ≠ {}
  ⟨proof⟩

Obviously, the set of RSD winners is always finite.

lemma rsd-winners-finite:
  assumes set agents' ⊆ agents finite alts alts' ⊆ alts
  shows finite (rsd-winners R alts' agents')
  ⟨proof⟩

lemmas rsd-winners-wf =
  rsd-winners-subset rsd-winners-nonempty rsd-winners-finite

The set of RSD winners is a Pareto-equivalence class.

lemma RSD-pareto-eqclass-rsd-winners-aux:
  assumes finite: finite alts and alts' ≠ {} and set agents' ⊆ agents
  shows RSD-pareto-eqclass (set agents') alts R (rsd-winners R alts agents')
  ⟨proof⟩

lemma RSD-pareto-eqclass-rsd-winners:
  assumes finite: finite alts and alts' ≠ {} and set agents' = agents
  shows RSD-pareto-eqclass agents alts R (rsd-winners R alts agents')
  ⟨proof⟩

For the proof of strategy-proofness, we need to define indifference sets and
lift preference relations to sets in a specific way.

context
begin

An indifference set for a given preference relation is a non-empty set of
alternatives such that the agent is indifferent over all of them.

private definition indiff-set where
  indiff-set S A ←→ A ≠ {} ∧ (∀ x ∈ A. ∀ y ∈ A. S x y)
private lemma indiff-set-mono: \( \text{indiff } S \ A \implies B \subseteq A \implies B \neq \emptyset \implies \text{indiff } S \ B \) 

(\text{proof})

Given an arbitrary set of alternatives \( A \) and an indifference set \( B \), we say that \( B \) is set-preferred over \( A \) w.r.t. the preference relation \( R \) if all (or, equivalently, any) of the alternatives in \( B \) are preferred over all alternatives in \( A \).

private definition RSD-set-rel where
\[ \text{RSD-set-rel } S \ A \ B \iff \text{indiff-set } S \ B \land (\forall x \in A. \ \forall y \in B. \ \text{S } x y) \]

The most-preferred alternatives (w.r.t. \( R \)) among any non-empty set of alternatives form an indifference set w.r.t. \( R \).

private lemma indiff-set-Max-wrt-among:
assumes finite carrier \( A \subseteq \text{carrier } A \neq \emptyset \) total-preorder-on carrier \( S \)
shows \( \text{indiff-set } S (\text{Max-wrt-among } S \ A) \) 

(\text{proof})

We now consider the set of RSD winners in the setting of a preference profile \( R \) and a manipulated profile \( R(i := R_i') \). This theorem shows that the set of RSD winners in the outcome is either the same in both cases or the outcome for the truthful profile is an indifference set that is set-preferred over the outcome for the manipulated profile.

lemma rsd-winners-manipulation-aux:
assumes \( \text{wf: total-preorder-on alts } R_i' \)
and \( i: i \in \text{agents } \text{and set agents'} \subseteq \text{agents } \text{finite agents} \)
and \( \text{finite: finite alts } \text{and alts } \neq \emptyset \)
defines [simp]: \( w' \equiv \text{rsd-winners } (R(i := R_i')) \text{ alts} \) and [simp]: \( w \equiv \text{rsd-winners } R \text{ alts} \)
shows \( w' \text{ agents'} = w \text{ agents'} \lor \text{RSD-set-rel } (R \ i) (w' \text{ agents'}) (w \text{ agents'}) \) 

(\text{proof})

The following variant of the previous theorem is slightly easier to use. We eliminate the case where the two outcomes are the same by observing that the original outcome is then also set-preferred to the manipulated one. In essence, this means that no matter what manipulation is done, the original outcome is always set-preferred to the manipulated one.

lemma rsd-winners-manipulation:
assumes \( \text{wf: total-preorder-on alts } R_i' \)
and \( i: i \in \text{agents } \text{and set agents'} = \text{agents } \text{finite agents} \)
and \( \text{finite: finite alts } \text{and alts } \neq \emptyset \)
defines [simp]: \( w' \equiv \text{rsd-winners } (R(i := R_i')) \text{ alts} \) and [simp]: \( w \equiv \text{rsd-winners } R \text{ alts} \)
shows \( \forall x \in w' \text{ agents'}. \ \forall y \in w \text{ agents'}. \ x \preceq_{R \ i} y \) 

(\text{proof})

end
The lottery that RSD yields is well-defined.

**Lemma** random-serial-dictatorship-support:

- **Assumes** finite agents finite alts agents' ⊆ agents alts' ≠ {} alts' ⊆ alts
- **Shows** \( \text{set-pmf} \left( \text{random-serial-dictatorship agents' alts' } R \right) \subseteq \text{alts'} \)

⟨proof⟩

Permutation of alternatives commutes with RSD winners.

**Lemma** rsd-winners-permute-profile:

- **Assumes** \( \sigma \) permutes alts and \( \text{set agents}' \subseteq \text{agents} \)
- **Shows** \( \text{rsd-winners} \left( \text{permute-profile } \sigma \right) \text{alts agents}' = \sigma ' \text{rsd-winners } R \text{alts agents}' \)

⟨proof⟩

**Lemma** random-serial-dictatorship-singleton:

- **Assumes** finite agents finite alts agents' ⊆ agents \( x \in \text{alts} \)
- **Shows** \( \text{random-serial-dictatorship agents'} \{x\} R = \text{return-pmf } x \) (is ?d = -)

⟨proof⟩

end

8.2 Proofs of properties

With all the facts that we have proven about the RSD winners, the hard work is mostly done. We can now simply fix some arbitrary order of the agents, apply the theorems about the RSD winners, and show the properties we want to show without doing much reasoning about probabilities.

context election

begin

abbreviation RSD ≡ random-serial-dictatorship agents alts

8.2.1 Well-definedness

**Sublocale** RSD: social-decision-scheme agents alts RSD

⟨proof⟩

8.2.2 RD extension

**Lemma** RSD-extends-RD:

- **Assumes** \( \text{is-pref-profile } R \) and unique: \( \text{has-unique-favorites } R \)
- ** Shows** \( \text{RSD } R = \text{RD } R \)

⟨proof⟩

8.2.3 Anonymity

Anonymity is a direct consequence of the fact that we randomise over all permutations in a uniform way.
### 8.2.4 Neutrality
Neutrality follows from the fact that the RSD winners of a permuted profile are simply the image of the original RSD winners under the permutation.

### 8.2.5 Ex-post efficiency
Ex-post efficiency follows from the fact that the set of RSD winners is a Pareto-equivalence class.

### 8.2.6 Strong strategy-proofness
Strong strategy-proofness is slightly more difficult to show. We have already shown that the set of RSD winners for the truthful profile is always set-preferred (by the manipulating agent) to the RSD winners for the manipulated profile. This can now be used to show strategy-proofness: We recall that the set of RSD winners is always an indifference class. Therefore, given any fixed alternative $x$ and considering a fixed order of the agents, either all of the RSD winners in the original profile are at least as good as $x$ or none of them are, and, since the original RSD winners are set-preferred to the manipulated ones, none of the RSD winners in the manipulated case are at least as good as $x$ either in that case. This means that for a fixed order of agents, either the probability that the original outcome is at least as good as $x$ is 1 or the probability that the manipulated outcome is at least as good as $x$ is 0. Therefore, the original lottery is clearly SD-preferred to the manipulated one.
9 Automatic definition of Preference Profiles

theory Preference-Profile-Cmd

imports
  Complex-Main
  ../Elections

keywords
  preference-profile :: thy-goal

begin

⟨ML⟩

context election

begin

lemma preferred-alts-prefs-from-table:
  assumes prefs-from-table-wf agents alts xs i ∈ set (map fst xs)
  shows preferred-alts (prefs-from-table xs i) x =
    of-weak-ranking-Collect-ge (rev (the (map-of xs i))) x
⟨proof⟩

lemma favorites-prefs-from-table:
  assumes wf prefs-from-table-wf agents alts xs and i: i ∈ agents
  shows favorites (prefs-from-table xs) i = hd (the (map-of xs i))
⟨proof⟩

lemma has-unique-favorites-prefs-from-table:
  assumes wf prefs-from-table-wf agents alts xs
  shows has-unique-favorites (prefs-from-table xs) =
    list-all (λz. is-singleton (hd (snd z))) xs
⟨proof⟩

end

9.1 Automatic definition of preference profiles from tables

function favorites-prefs-from-table where
  i = j ⇒ favorites-prefs-from-table ((j,x)⧵xs) i = hd x
  | i ≠ j ⇒ favorites-prefs-from-table ((j,x)⧵xs) i =
    favorites-prefs-from-table xs i
  | favorites-prefs-from-table [] i = {}
⟨proof⟩

termination ⟨proof⟩

lemma (in election) eval-favorites-prefs-from-table:
assumes \( \text{prefs-from-table-wf} \) agents alts \( \mathcal{X} \)
shows \( \text{favorites-prefs-from-table} \) \( \mathcal{X} \) \( i = \) favorites (prefs-from-table \( \mathcal{X} \)) \( i \)

\( \langle \text{proof} \rangle \)

function weak-ranking-prefs-from-table where 
\( i \neq j \implies \text{weak-ranking-prefs-from-table} \) \( ((i, x)\#) \mathcal{X} j = \text{weak-ranking-prefs-from-table} \) \( \mathcal{X} j \)
\( i = j \implies \text{weak-ranking-prefs-from-table} \) \( ((i, x)\#) \mathcal{X} j = x \)
\( \text{weak-ranking-prefs-from-table} \) \( [] \) \( j = [] \)

\( \langle \text{proof} \rangle \)

lemma eval-weak-ranking-prefs-from-table:
assumes \( \text{prefs-from-table-wf} \) agents alts \( \mathcal{X} \)
shows \( \text{weak-ranking-prefs-from-table} \) \( \mathcal{X} i = \text{weak-ranking} \) (prefs-from-table \( \mathcal{X} \)) \( i \)

\( \langle \text{proof} \rangle \)

lemma eval-prefs-from-table-aux:
assumes \( \mathcal{R} \equiv \text{prefs-from-table} \) \( \mathcal{X} \) \( \text{prefs-from-table-wf} \) agents alts \( \mathcal{X} \)
shows \( \mathcal{R} \) \( i \) \( a \) \( b \) \( \iff \) prefs-from-table \( \mathcal{X} i \) \( a \) \( \prec \) [\( \mathcal{R} \) \( i \)] \( b \) \( \iff \) prefs-from-table \( \mathcal{X} i \) \( a \) \( \prec \) \( \mathcal{R} \) \( i \) \( b \) \( \neg \) prefs-from-table \( \mathcal{X} i \) \( b \) \( \prec \) \( \mathcal{R} \) \( i \) 
anonymous-profile \( \mathcal{R} \equiv \text{mset} \) (map snd \( \mathcal{X} \))
election agents alts \( i \in \text{set} \) (map fst \( \mathcal{X} \)) \( \implies \) preferred-alts (\( \mathcal{R} \) \( i \)) \( x \) \( \equiv \) of-weak-ranking-Collect-ge (rev (the (map-of \( \mathcal{X} i \)))) \( x \)
election agents alts \( i \in \text{set} \) (map fst \( \mathcal{X} \)) \( \implies \) favorites \( \mathcal{R} \) \( i \) \( = \) favorites-prefs-from-table \( \mathcal{X} i \)
election agents alts \( i \in \text{set} \) (map fst \( \mathcal{X} \)) \( \implies \) weak-ranking (\( \mathcal{R} \) \( i \)) \( = \) weak-ranking-prefs-from-table \( \mathcal{X} i \)
election agents alts \( i \in \text{set} \) (map fst \( \mathcal{X} \)) \( \implies \) favorite \( \mathcal{R} \) \( i \) \( = \) the-elem (favorites-prefs-from-table \( \mathcal{X} i \))
election agents alts \( \implies \) has-unique-favorites \( \mathcal{R} \) \( \iff \) list-all (\( \lambda z \). is-singleton (hd (snd \( z \)))) \( \mathcal{X} \)

\( \langle \text{proof} \rangle \)

lemma pref-profile-from-tableI:\ :
assumes \( \mathcal{R} \equiv \text{prefs-from-table} \) \( \mathcal{X} \) \( \text{prefs-from-table-wf} \) agents alts \( \mathcal{X} \)
shows \( \text{pref-profile-wf} \) agents alts \( \mathcal{R} \)

\( \langle \text{proof} \rangle \)

\langle \text{ML} \rangle

end
theory QSOpt-Exact
imports Complex-Main
begin
end

10 Automatic Fact Gathering for Social Decision Schemes

theory SDS-Automation
imports Preferences-Profile-Cmd QSOpt-Exact ../Social-Decision-Schemes
keywords derive-orbit-equations derive-support-conditions derive-ex-post-conditions find-inefficient-supports prove-inefficient-supports derive-strategyproofness-conditions :: thy-goal
begin

We now provide the following commands to automatically derive restrictions on the results of Social Decision Schemes satisfying Anonymity, Neutrality, Efficiency, or Strategy-Proofness:

- **derive-orbit-equations** to derive equalities arising from automorphisms of the given profiles due to Anonymity and Neutrality
- **derive-ex-post-conditions** to find all Pareto losers and the given profiles and derive the facts that they must be assigned probability 0 by any ex-post-efficient SDS
- **find-inefficient-supports** to use Linear Programming to find all minimal SD-inefficient (but not ex-post-inefficient) supports in the given profiles and output a corresponding witness lottery for each of them
- **prove-inefficient-supports** to prove a specified set of support conditions arising from ex-post- or SD-Efficiency. For conditions arising from SD-Efficiency, a witness lottery must be specified (e.g. as computed by derive-orbit-equations).
- **derive-support-conditions** to automatically find and prove all support conditions arising from ex-post- and SD-Efficiency
**derive-strategyproofness-conditions** to automatically derive all conditions arising from weak Strategy-Proofness and any manipulations between the given preference profiles. An optional maximum manipulation size can be specified.

All commands except **find-inefficient-supports** open a proof state and leave behind proof obligations for the user to discharge. This should always be possible using the Simplifier, possibly with a few additional rules, depending on the context.

**lemma** disj-False-right: \( P \lor \text{False} \longleftrightarrow P \) <proof> \end{proof}

**lemmas** multiset-add-ac = add-ac[where \( ?a = \text{a multiset} \)]

**lemma** less-or-eq-real:
\[
(x :: \text{real}) < y \lor x = y \longleftrightarrow x \leq y \lor y = x \longleftrightarrow x \leq y \quad \text{⟨proof⟩}
\]

**lemma** multiset-Diff-single-normalize:
\[
\text{fixes } a \ c \quad \text{assumes } a \neq c \\
\text{shows } 
\{ a \} + \{ a \} - \{ c \} = \{ a \} + ( B - \{ c \})
\text{⟨proof⟩}
\]

**lemma** ex-post-efficient-aux:
\[
\text{assumes } \text{pref-from-table-wf } \text{agents} \ \text{alts} \ \text{xss} \ \text{R} \equiv \text{pref-from-table } \text{xss} \\
\text{assumes } i \in \text{agents} \ \forall i \in \text{agents}. y \succeq [\text{pref-from-table } \text{xss} i] x \sim y \succeq [\text{pref-from-table } \text{xss} i] x \\
\text{shows } \text{ex-post-efficient-sds } \text{agents} \ \text{alts} \ \text{sds} \longrightarrow \text{pmf } (\text{sds} \ R) x = 0
\text{⟨proof⟩}
\]

**lemma** SD-inefficient-support-aux:
\[
\text{assumes } \text{R} : \text{pref-from-table-wf } \text{agents} \ \text{alts} \ \text{xss} \ \text{R} \equiv \text{pref-from-table } \text{xss} \\
\text{assumes } \text{as} : \text{as} \neq [] \text{ set as } \subseteq \text{alts distinct as } A = \text{set as} \\
\text{assumes } \text{ys} : \forall x \in \text{set } (\text{map snd } \text{ys}). 0 \leq x \ \text{sum-list } (\text{map snd } \text{ys}) = 1 \text{ set } (\text{map } \text{fst } \text{ys}) \subseteq \text{alts} \\
\text{assumes } i : i \in \text{agents} \\
\text{assumes } \text{SD1} : \forall i \in \text{agents}. \forall x \in \text{alts}. \\
\text{sum-list } (\text{map snd } (\text{filter } (\lambda y. \text{pref-from-table } \text{xss} i x (\text{fst } y)) \ \text{ys})) \geq \text{real } (\text{length } (\text{filter } (\text{pref-from-table } \text{xss} i x) \ \text{as})) / \text{real } (\text{length } \text{as}) \\
\text{assumes } \text{SD2} : \exists x \in \text{alts}. \text{sum-list } (\text{map snd } (\text{filter } (\lambda y. \text{pref-from-table } \text{xss} i x (\text{fst } y)) \ \text{ys})) > \text{real } (\text{length } (\text{filter } (\text{pref-from-table } \text{xss} i x) \ \text{as})) / \text{real } (\text{length } \text{as}) \\
\text{shows } \text{sd-efficient-sds } \text{agents} \ \text{alts} \ \text{sds} \longrightarrow (\exists x \in A. \text{pmf } (\text{sds} \ R) x = 0)
\text{⟨proof⟩}
\]

**definition** pref-classes where
\[
\text{pref-classes } \text{alts} \ \text{le} = \text{preferred-alts } \text{le} \ • \text{alts} - \{ \text{alts} \}
\]

58
primrec pref-classes-lists where
  pref-classes-lists [] = {}
| pref-classes-lists (xs # xss) = insert (Union (set (xs # xss))) (pref-classes-lists xss)

fun pref-classes-lists-aux where
  pref-classes-lists-aux acc [] = {}
| pref-classes-lists-aux acc (xs # xss) = insert acc (Union (set (xs # xss))) (pref-classes-lists-aux (acc Union xs) xss)

lemma pref-classes-lists-append:
  pref-classes-lists (xs @ ys) = (Union ((Union (set ys)) (pref-classes-lists xs Union pref-classes-lists ys))
⟨proof⟩
lemma pref-classes-lists-aux:
  assumes is-weak-ranking xss acc (Union (set xss)) = {}
  shows pref-classes-lists-aux acc xss =
    (insert acc (Union ((lambda A. A Union acc) (pref-classes-lists (rev xss))) Union {acc Union (Union (set xss)}))
⟨proof⟩
lemma pref-classes-list-aux-hd-tl:
  assumes is-weak-ranking xss xss (neq [])
  shows pref-classes-lists-aux (hd xss) (tl xss) = pref-classes-lists (rev xss) - {Union (set xss)}
⟨proof⟩
lemma pref-classes-of-weak-ranking-aux:
  assumes is-weak-ranking xss
  shows of-weak-ranking-Collect-ge xss (Union (set xss)) = pref-classes-lists xss
⟨proof⟩
lemma eval-pref-classes-of-weak-ranking:
  assumes Union (set xss) = alts is-weak-ranking xss alts (neq {})
  shows pref-classes alts (of-weak-ranking xss) = pref-classes-lists-aux (hd xss) (tl xss)
⟨proof⟩

context preorder-on
begin
lemma SD-iff-pref-classes:
  assumes p ∈ lotteries-on carrier q ∈ lotteries-on carrier
  shows p ⪯ SD (le) q ←→
    (∀ A∈ pref-classes carrier le. measure-pmf.prob p A ≤ measure-pmf.prob q A)
⟨proof⟩
lemma (in strategyproof-an-sds) strategyproof':
assumes wf: is-pref-profile R total-preorder-on alts Ri' and i: i ∈ agents
shows (∃ A∈pref-classes alts (R i). lottery-prob (sds (R(i := Ri'))) A <
lottery-prob (sds R) A) ∨
(∀ A∈pref-classes alts (R i). lottery-prob (sds (R(i := Ri'))) A =
lottery-prob (sds R) A)
⟨proof⟩
lemma pref-classes-lists-aux-finite:
A ∈ pref-classes-lists-aux acc xss ⇒ finite acc ⇒ (∀ A ∈ set xss ⇒ finite A)
⟨proof⟩
lemma strategyproof-aux:
assumes wf: prefs-from-table-wf agents alts xss1 R1 ≡ prefs-from-table xss1
prefs-from-table-wf agents alts xss2 R2 ≡ prefs-from-table xss2
assumes sds: strategyproof-an-sds agents alts sds and i: i ∈ agents and j: j ∈ agents
assumes eq: R1(i := R2 j) = R2 the (map-of xss1 i) = xs
pref-classes-lists-aux (hd xs) (tl xs) = ps
shows (∃ A∈ps. (∑ x∈A. pmf (sds R2) x) < (∑ x∈A. pmf (sds R1) x)) ∨
(∀ A∈ps. (∑ x∈A. pmf (sds R2) x) = (∑ x∈A. pmf (sds R1) x))
⟨proof⟩
lemma strategyproof-aux':
assumes wf: prefs-from-table-wf agents alts xss1 R1 ≡ prefs-from-table xss1
prefs-from-table-wf agents alts xss2 R2 ≡ prefs-from-table xss2
assumes sds: strategyproof-an-sds agents alts sds and i: i ∈ agents and j: j ∈ agents
assumes perm: list-permutates ys alts
defines σ ≡ permutation-of-list ys and σ' ≡ inverse-permutation-of-list ys
defines xs ≡ the (map-of xss1 i)
defines xs': xs' ≡ map ((i) σ) (the (map-of xss2 j))
defines R'i ≡ of-weak-ranking xs'
assumes distinct-ps: ∀ A∈ps. distinct A
assumes eq: mset (map snd xss1) - {#the (map-of xss1 i)\#} + {#xs'\#} =
mset (map (map ((i) σ) ○ snd) xss2)
pref-classes-lists-aux (hd xs) (tl xs) = set ' ps
shows list-permutates ys alts ∧
((∃ A∈ps. (∑ x←A. pmf (sds R2) (σ' x)) < (∑ x←A. pmf (sds R1) x))) ∨
((∀ A∈ps. (∑ x←A. pmf (sds R2) (σ' x)) = (∑ x←A. pmf (sds R1) x)))
(is - ∧ ?th)
⟨proof⟩
References