Randomised Social Choice

Manuel Eberl

April 20, 2020

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
2.5 Preference profiles from lists .................................. 22
2.6 Automatic evaluation of preference profiles ................. 23

3 Auxiliary facts about PMFs ........................................ 26
3.1 Definition of von Neumann–Morgenstern utility functions .. 27

4 Stochastic Dominance ............................................ 28
4.1 Definition of Stochastic Dominance .......................... 28
4.2 Stochastic Dominance for preference profiles ............... 30
4.3 SD efficient lotteries ............................................. 31
4.4 Equivalence proof ............................................. 32
4.5 Existence of SD-efficient lotteries .............................. 34

5 Social Decision Schemes ......................................... 35
5.1 Basic Social Choice definitions ................................. 36
5.2 Social Decision Schemes ....................................... 36
5.3 Anonymity ...................................................... 36
5.4 Neutrality ...................................................... 37
5.5 Ex-post efficiency ............................................ 38
5.6 SD efficiency ................................................. 39
5.7 Weak strategyproofness ..................................... 39
5.8 Strong strategyproofness ................................... 40

6 Lowering Social Decision Schemes ............................. 41

7 Random Dictatorship ............................................. 46
7.1 Anonymity ...................................................... 47
7.2 Neutrality ...................................................... 47
7.3 Strong strategyproofness ................................... 48

8 Random Serial Dictatorship .................................... 48
8.1 Auxiliary facts about RSD ..................................... 50
8.1.1 Pareto-equivalence classes ................................. 50
8.1.2 Facts about RSD winners .................................. 51
8.2 Proofs of properties .......................................... 53
8.2.1 Well-definedness ......................................... 53
8.2.2 RD extension ............................................. 53
8.2.3 Anonymity .............................................. 53
8.2.4 Neutrality ............................................... 54
8.2.5 Ex-post efficiency ....................................... 54
8.2.6 Strong strategy-proofness .............................. 54

9 Automatic definition of Preference Profiles ................. 55
9.1 Automatic definition of preference profiles from tables .... 55
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

type-synonym 'a relation = 'a ⇒ 'a ⇒ bool

definition map-relation :: ('a ⇒ 'b) ⇒ 'b relation ⇒ 'a relation where
  map-relation f R = (λx y. R (f x) (f y))
definition restrict-relation :: 'a set ⇒ 'a relation ⇒ 'a relation where
  restrict-relation A R = (λx y. x ∈ A ∧ y ∈ A ∧ R x y)

lemma restrict-relation-restrict-relation [simp]:
  restrict-relation A (restrict-relation B R) = restrict-relation (A ∩ B) R
  (proof)

lemma restrict-relation-empty [simp]: restrict-relation {} R = (λ-. False)
  (proof)

lemma restrict-relation-UNIV [simp]: restrict-relation UNIV R = R
  (proof)

1.2 Preorders

Preorders are reflexive and transitive binary relations.

locale preorder-on =
  fixes carrier :: 'a set
  fixes le :: 'a relation
  assumes not-outside: le x y ⟹ x ∈ carrier le x y ⟹ y ∈ carrier
  assumes refl: x ∈ carrier ⟹ le x x
  assumes trans: le x y ⟹ le y z ⟹ le x z
begin

lemma carrier-eq: carrier = {x. le x x}
  (proof)

lemma preorder-on-on-map:
  preorder-on (f − carrier) (map-relation f le)
  (proof)
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
\( a \geq[R] b \equiv b \preceq[R] a \)
term \( a \geq[R] b \leftrightarrow b \preceq[R] a \)

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

1.4 Orders
locale order-on = preorder-on +
  assumes 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 :: ’a relation ⇒ ’a set ⇒ ’a set
Max-wrt-among R A = \{ x \in A. R x x \land (\forall y \in A. R x y \rightarrow R y x) \}\nlemma Max-wrt-among-cong:
  assumes restrict-relation A R = restrict-relation A R’
  shows Max-wrt-among R A = 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:
\textbf{assumes} bij \((f :: 'a \Rightarrow 'b)\)
\textbf{shows} \(f \cdot \text{Max-wrt-among le} \ A \subseteq\)
\(\text{Max-wrt-among} \ (\text{map-relation} \ (\text{inv f}) \ \text{le}) \ (f \cdot \ A)\)
\(\langle\text{proof}\rangle\)

\textbf{lemma} \(\text{Max-wrt-among-map-relation-bij}:\)
\textbf{assumes} bij \(f\)
\textbf{shows} \(f \cdot \text{Max-wrt-among le} \ A = \text{Max-wrt-among} \ (\text{map-relation} \ (\text{inv f}) \ \text{le}) \ (f \cdot \ A)\)
\(\langle\text{proof}\rangle\)

\textbf{lemma} \(\text{Max-wrt-map-relation-bij}:\)
\(\text{bij f} \Rightarrow f \cdot \text{Max-wrt le} = \text{Max-wrt} \ (\text{map-relation} \ (\text{inv f}) \ \text{le})\)
\(\langle\text{proof}\rangle\)

\textbf{lemma} \(\text{Max-wrt-among-mono}:\)
\(\text{le} \ x \ y \Rightarrow x \in \text{Max-wrt-among le} \ A \Rightarrow y \in A \Rightarrow y \in \text{Max-wrt-among le} \ A\)
\(\langle\text{proof}\rangle\)

\textbf{lemma} \(\text{Max-wrt-mono}:\)
\(\text{le} \ x \ y \Rightarrow x \in \text{Max-wrt le} \Rightarrow y \in \text{Max-wrt le}\)
\(\langle\text{proof}\rangle\)

\textbf{end}

\textbf{context} total-preorder-on
\textbf{begin}

\textbf{lemma} \(\text{Max-wrt-among-total-preorder}:\)
\(\text{Max-wrt-among le} \ A = \{x \in \text{carrier} \ \cap \ A. \ \forall y \in \text{carrier} \ \cap A. \ \text{le} y x\}\)
\(\langle\text{proof}\rangle\)

\textbf{lemma} \(\text{Max-wrt-total-preorder}:\)
\(\text{Max-wrt le} = \{x \in \text{carrier}. \ \forall y \in \text{carrier}. \ \text{le} y x\}\)
\(\langle\text{proof}\rangle\)

\textbf{lemma} \(\text{decompose-Max}:\)
\textbf{assumes} \(A: A \subseteq \text{carrier}\)
\textbf{defines} \(M \equiv \text{Max-wrt-among le} \ A\)
\textbf{shows} \(\text{restrict-relation} \ A \ le = (\lambda x y. \ x \in A \ \land \ y \in M \ \lor \ y \notin M \ \land \ \text{restrict-relation} \ (A - M) \ \text{le} \ x y))\)
\(\langle\text{proof}\rangle\)

\textbf{end}

\textbf{1.6 Weak rankings}

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

**lemma** of-weak-ranking-Nil [simp]: of-weak-ranking [] = (\lambda -. False)

(\proof)

**lemma** of-weak-ranking-Nil' [code]: of-weak-ranking [] x y = False

(\proof)

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

(is ?lhs \leftrightarrow ?rhs)

(\proof)

**lemma** of-weak-ranking-indifference:

assumes \( A \in \text{set } xs \), \( x \in A \), \( y \in A \)

shows \( x \geq [\text{of-weak-ranking } xs] y \)

(\proof)

**lemma** of-weak-ranking-map:

map-relation \( f \) (of-weak-ranking \( xs \)) = of-weak-ranking (map ((-') \( f \)) \( xs \))

(\proof)

**lemma** of-weak-ranking-permute':

assumes \( f \) permutes (\bigcup (\text{set } xs))

shows \( \text{map-relation } f \) (of-weak-ranking \( xs \)) = of-weak-ranking (map ((') (inv \( f \))) \( xs \))

(\proof)

**lemma** of-weak-ranking-permute:

assumes \( f \) permutes (\bigcup (\text{set } xs))

shows \( \text{of-weak-ranking } (\text{map } ((') \( f \)) \( xs \)) = \text{map-relation } (\text{inv } f) \) (of-weak-ranking \( xs \))

(\proof)

definition is-weak-ranking where
\[
is-weak-ranking \; xs \leftarrow ((\{\} \notin \text{set } xs) \land \\
(\forall i j. i < \text{length } xs \land j < \text{length } zs \land i \neq j \rightarrow xs ! i \cap xs ! j = \{\}))
\]

definition is-finite-weak-ranking where
\[
is-finite-weak-ranking \; xs \leftarrow is-weak-ranking \; xs \land (\forall x \in \text{set } xs. \text{finite } x)
\]

definition weak-ranking :: 'alt relation \Rightarrow '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?]:

assumes \( \{\} \notin \text{set } xs \land i j. i < \text{length } xs \rightarrow j < \text{length } xs \rightarrow i \neq j \rightarrow xs ! i

\[ \cap \text{xs } \vdash j = \{ \} \]
shows is-weak-ranking \text{xs}
\langle proof \rangle

lemma is-weak-ranking-nonempty: is-weak-ranking \text{xs} \implies \{ \} \notin \text{set xs}
\langle proof \rangle

lemma is-weak-rankingD:
assumes is-weak-ranking \text{xs } i < \text{length } \text{xs } j < \text{length } \text{xs } i \neq j
shows \text{xs } ! i \cap \text{xs } ! j = \{ \} 
\langle proof \rangle

lemma is-weak-ranking-iff:
is-weak-ranking \text{xs} \iff \text{distinct } \text{xs} \land \text{disjoint } (\text{set } \text{xs}) \land \{ \} \notin \text{set xs}
\langle proof \rangle

lemma is-weak-ranking-rev [simp]: is-weak-ranking (rev \text{xs}) \iff is-weak-ranking \text{xs}
\langle proof \rangle

lemma is-weak-ranking-map-inj:
assumes is-weak-ranking \text{xs } \text{inj-on } f (\bigcup (\text{set } \text{xs}))
shows is-weak-ranking (map ((\text{'f}) f) \text{xs})
\langle proof \rangle

lemma of-weak-ranking-rev [simp]:
of-weak-ranking (rev \text{xs}) (x::'a) y \iff of-weak-ranking \text{xs } y x 
\langle proof \rangle

lemma is-weak-ranking-Nil [simp, code]: is-weak-ranking []
\langle proof \rangle

lemma is-finite-weak-ranking-Nil [simp, code]: is-finite-weak-ranking []
\langle proof \rangle

lemma is-weak-ranking-Cons-empty [simp]:
\neg is-weak-ranking (\{ \} # \text{xs}) \langle proof \rangle

lemma is-finite-weak-ranking-Cons-empty [simp]:
\neg is-finite-weak-ranking (\{ \} # \text{xs}) \langle proof \rangle

lemma is-weak-ranking-singleton [simp]:
is-weak-ranking [x] \iff x \neq \{ \}
\langle proof \rangle

lemma is-finite-weak-ranking-singleton [simp]:
is-finite-weak-ranking [x] \iff x \neq \{ \} \land \text{finite } x 
\langle proof \rangle
lemma is-weak-ranking-append:
is-weak-ranking (xs @ ys) ←→
is-weak-ranking xs ∧ is-weak-ranking ys ∧
(set xs ∩ set ys = {}) ∧ ∪(set xs) ∩ ∪(set ys) = {}
⟨proof⟩

lemma is-weak-ranking-Cons [code]:
is-weak-ranking (x # xs) ←→
x ≠ {} ∧ is-weak-ranking xs ∧ x ∩ ∪(set xs) = {}
⟨proof⟩

lemma is-finite-weak-ranking-Cons [code]:
is-finite-weak-ranking (x # xs) ←→
x ≠ {} ∧ finite x ∧ is-finite-weak-ranking xs ∧ x ∩ ∪(set xs) = {}
⟨proof⟩

primrec is-weak-ranking-aux where
is-weak-ranking-aux A [] ←→ True
| is-weak-ranking-aux A (x # xs) ←→ x ≠ {} ∧
A ∩ x = {} ∧ is-weak-ranking-aux (A ∪ x) xs

lemma is-weak-ranking-aux:
is-weak-ranking-aux A xs ←→ A ∪ (set xs) = {} ∧ is-weak-ranking xs
⟨proof⟩

lemma is-weak-ranking-code [code]:
is-weak-ranking xs ←→ is-weak-ranking-aux {} xs
⟨proof⟩

lemma of-weak-ranking-altdef:
assumes is-weak-ranking xs x ∈ ∪(set xs) y ∈ ∪(set xs)
shows of-weak-ranking xs x y ←→
find-index ((∈) x) xs ≥ find-index ((∈) y) xs
⟨proof⟩

lemma total-preorder-of-weak-ranking:
assumes ∪(set xs) = A
assumes is-weak-ranking xs
shows total-preorder-on A (of-weak-ranking xs)
⟨proof⟩

lemma restrict-relation-of-weak-ranking-Cons:
assumes is-weak-ranking (A # As)
shows restrict-relation (∪(set As)) (of-weak-ranking (A # As)) = of-weak-ranking As
⟨proof⟩
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 wf: 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 ∈ \bigcup(\set{xs}) b ∈ \bigcup(\set{xs})
⟨proof⟩

lemma of-weak-ranking-Cons:
  assumes a ∈ \bigcup(\set{x \# xs}) b ∈ \bigcup(\set{x \# xs})
  shows of-weak-ranking (x#xs) a b ←→ b ∈ x ∨ (a ∉ x ∧ of-weak-ranking xs a b)
⟨proof⟩

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

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

lemma Max-wrt-of-weak-ranking-Cons:
Max-wrt (of-weak-ranking (x\#xs)) = x
⟨proof⟩

end

lemma Max-wrt-of-weak-ranking:
  assumes is-weak-ranking xs
  shows Max-wrt (of-weak-ranking xs) = (if xs = [] then {} else hd xs)
⟨proof⟩

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

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

function weak-ranking-aux :: ’a set → ’a set list where
  weak-ranking-aux {} = []
  | A ≠ {} ⇒ A ⊆ carrier ⇒ weak-ranking-aux A =
    Max-wrt-among le A ≠ weak-ranking-aux (A − Max-wrt-among le A)
  | ¬(A ⊆ carrier) ⇒ weak-ranking-aux A = undefined
⟨proof⟩

termination ⟨proof⟩

lemma weak-ranking-aux-Union:
  A ⊆ carrier ⇒ ∪(set (weak-ranking-aux A)) = A
⟨proof⟩

lemma weak-ranking-aux-wf:
  A ⊆ carrier ⇒ is-weak-ranking (weak-ranking-aux A)
⟨proof⟩

lemma of-weak-ranking-weak-ranking-aux’:
  assumes A ⊆ carrier x ∈ A y ∈ A
  shows of-weak-ranking (weak-ranking-aux A) x y ↔ restrict-relation A le x y
⟨proof⟩

lemma of-weak-ranking-weak-ranking-aux:
  of-weak-ranking (weak-ranking-aux carrier) = le
⟨proof⟩

lemma weak-ranking-aux-unique’:
  assumes ∪(set As) ⊆ carrier is-weak-ranking As
  of-weak-ranking As = restrict-relation (∪(set As)) le
  shows As = weak-ranking-aux (∪(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-altdef:
  weak-ranking le = weak-ranking-aux carrier
⟨proof⟩

lemma weak-ranking-Union: \( \bigcup \{\text{weak-ranking le}\} = \text{carrier} \)
⟨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 \( \in \) xs ! i x \( \in \) xs ! j
  shows i = j
⟨proof⟩

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

lemma weak-ranking-eqclass1:
  assumes A \( \in \) set (weak-ranking le) x \( \in \) A 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:

14
assumes \( x \in \text{hd}(\text{weak-ranking le}) \) \( y \in \text{carrier} \)
shows \( \text{le} \ x \ x \)
(proof)

lemma last-weak-ranking:
assumes \( x \in \text{last}(\text{weak-ranking le}) \) \( y \in \text{carrier} \)
shows \( \text{le} \ x \ y \)
(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 \text{weak-ranking-index} :: 'a \Rightarrow \text{nat} \)
where
\( \text{weak-ranking-index} \ x = \text{find-index} (\lambda A. x \in A) (\text{weak-ranking le}) \)

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

lemma ranking-index-eqI:
\( i < \text{length} (\text{weak-ranking le}) \Rightarrow x \in \text{weak-ranking le} ! i \mathrel{\implies} \text{weak-ranking-index} \ x = i \)
(proof)

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

end

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

lemmas of-weak-ranking-weak-ranking =
\text{finite-total-preorder-on.weak-ranking-total-preorder}(2)

lemma finite-total-preorder-on-iff:
\text{finite-total-preorder-on} A R \leftarrow\rightarrow \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 \( \text{is-finite-weak-ranking} \ xs \)
shows \( \text{weak-ranking} \ (\text{of-weak-ranking} \ \text{xs}) = \text{xs} \) (proof)

**lemma** weak-ranking-eqD:

- assumes finite-total-preorder-on alts \( R1 \)
- assumes finite-total-preorder-on alts \( R2 \)
- assumes weak-ranking \( R1 = \text{weak-ranking} \ R2 \)
- shows \( R1 = R2 \) (proof)

**lemma** weak-ranking-eq-iff:

- assumes finite-total-preorder-on alts \( R1 \)
- assumes finite-total-preorder-on alts \( R2 \)
- shows \( \text{weak-ranking} \ R1 = \text{weak-ranking} \ R2 \iff R1 = R2 \) (proof)

**definition** preferred-alts :: 'alt relation ⇒ 'alt ⇒ 'alt set where

\[ \text{preferred-alts} \ R \ x = \{ y. \ y \geq [R] \ x \}\]

**lemma** (in preorder-on) preferred-alts-refl [simp]: \( x \in \text{carrier} \implies x \in \text{preferred-alts} \) le \( x \) (proof)

**lemma** (in preorder-on) preferred-alts-altdef:

\[ \text{preferred-alts} \ \text{le} \ x = \{ y \in \text{carrier}. \ y \geq [\text{le}] \ x \}\]

(proof)

**lemma** (in preorder-on) preferred-alts-subset: preferred-alts \( \leq x \subseteq \text{carrier} \) (proof)

### 1.7 Rankings

**definition** ranking :: 'a relation ⇒ 'a list where

\[ \text{ranking} \ R = \text{map} \ \text{the-elem} \ (\text{weak-ranking} \ R) \]

**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 \( A \in \text{set} \) (weak-ranking \( \leq \))
- shows \( \text{is-singleton} A \) (proof)

16
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:

17
assumes \( x \leq [R \ i] \ y \)
shows \( i \in \text{agents} \ x \in \text{alts} \ y \in \text{alts} \)
⟨proof⟩

**sublocale** preorder-family agents alts \( R \)
⟨proof⟩

**lemmas** prefs-undefined' = not-in-dom'

**lemma** wf-update:
assumes \( i \in \text{agents} \ \text{total-preorder-on} \ \text{alts} \ Ri' \)
shows \( \text{pref-profile-wf} \ \text{agents} \ \text{alts} \ (R(i := Ri')) \)
⟨proof⟩

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

**lemma** (in −) pref-profile-eq1:
assumes \( \text{pref-profile-wf} \ \text{agents} \ \text{alts} \ R1 \ \text{pref-profile-wf} \ \text{agents} \ \text{alts} \ R2 \)
assumes \( \forall x, x \in \text{agents} \Rightarrow R1 x = R2 x \)
shows \( R1 = R2 \)
⟨proof⟩

end

Permutates 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 \( \sigma \) \( R = (\lambda i \ x \ y. \ R \ i \ (\text{inv} \ \sigma \ x) \ (\text{inv} \ \sigma \ y)) \)

**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 \( \text{bij} \ f \ \text{bij} \ g \)
shows \( \text{permute-profile} \ f \ (\text{permute-profile} \ g \ R) = \text{permute-profile} \ (f \circ g) \ R \)
⟨proof⟩

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

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

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

### 2.1 Pareto dominance

**definition** Pareto :: ($\text{agent} \Rightarrow \text{'alt relation} \Rightarrow \text{'alt relation}$ where
\[
x \preceq [\text{Pareto}(R)] \ y \iff (\exists j. \ x \preceq [R\ j] \ x) \land (\forall i. \ x \preceq [R \ i] \ x \rightarrow x \preceq [R \ i] \ y)
\]

A Pareto loser is an alternative that is Pareto-dominated by some other alternative.

**definition** pareto-losers :: ($\text{agent, 'alt}$ pref-profile $\Rightarrow$ 'alt set where
\[
\text{pareto-losers } R = \{ x. \ \exists y. \ y > [\text{Pareto}(R)] \ x \}
\]

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

context preorder-family
begin

**lemma** Pareto-iff:
\[
x \preceq [\text{Pareto}(R)] \ y \iff (\forall i \in \text{dom}\ x \preceq [R\ i] \ y)
\]

**lemma** Pareto-strict-iff:
\[
x \prec [\text{Pareto}(R)] \ y \iff (\forall i \in \text{dom}\ x \preceq [R\ i] \ y) \land (\exists i \in \text{dom}\ x \prec [R\ i] \ y)
\]

**lemma** Pareto-strictI:
assumes $\bigwedge i. \ i \in \text{dom} \implies x \preceq [R\ i] \ y\ i \in \text{dom} \ x \prec [R\ i] \ y$
shows $x \prec [\text{Pareto}(R)] \ y$

**lemma** Pareto-strictI':
assumes $\bigwedge i. \ i \in \text{dom} \implies x \preceq [R\ i] \ y\ i \in \text{dom} \ \neg x \succeq [R\ i] \ y$
shows $x \prec [\text{Pareto}(R)] \ y$

sublocale Pareto: preorder-on carrier Pareto$(R)$

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

class context pref-profile-wf
begin

lemma preferred-alts-subset-alts: preferred-alts $(R i) x \subseteq \text{alts}$ (is ?A)
    and finite-preferred-alts [simp,intro!]: finite (preferred-alts $(R i) x$) (is ?B)
⟨proof⟩

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

end

2.3 Favourite alternatives

definition favorites :: (agent, alt) pref-profile => agent => alt set where
    favorites $R i = \text{Max-wrt } (R i)$

definition favorite :: (agent, alt) pref-profile => agent => alt where
    favorite $R i = \text{the-elem } (\text{favorites } R i)$

definition has-unique-favorites :: (agent, alt) pref-profile => bool where
    has-unique-favorites $R \longleftrightarrow (\forall i. \; \text{favorites } R i = \{\} \lor \text{is-singleton } (\text{favorites } R i))$

class context pref-profile-wf
begin

lemma favorites-altdef:
    favorites $R i = \text{Max-wrt-among } (R i) \text{ alts}$
⟨proof⟩

lemma favorites-no-agent [simp]: $i \notin \text{agents} \implies \text{favorites } R i = \{\}$
⟨proof⟩

end
**Lemma** favorites-altdef:
\[
\text{favorites } R i = \{ x \in \text{alts. } \forall y \in \text{alts. } x \succeq [R i] y \}
\]

**Lemma** favorites-subset-alts: favorites \( R i \subseteq \text{alts} \)

**Lemma** finite-favorites [simp, intro]: finite (favorites \( R i \))

**Lemma** favorites-nonempty: \( i \in \text{agents} \implies \text{favorites } R i \neq \{ \} \)

**Lemma** favorites-permute:
\[
\text{assumes } i : i \in \text{agents and perm: } \sigma \text{ permutes alts}
\text{shows } \text{favorites } (\text{permute-profile } \sigma \ R) i = \sigma ' \text{ favorites } R i
\]

**Lemma** has-unique-favorites-altdef:
\[
\text{has-unique-favorites } R \longleftrightarrow (\forall i \in \text{agents. } \text{is-singleton } (\text{favorites } R i))
\]

end

**Locale** pref-profile-unique-favorites = pref-profile-wf agents alts \( R \)

**Lemma** unique-favorites: \( i \in \text{agents} \implies \text{favorites } R i = \{ \text{favorite } R i \} \)

**Lemma** favorite-in-alts: \( i \in \text{agents} \implies \text{favorite } R i \in \text{alts} \)

end

2.4 Anonymous profiles

**Type-synonym** ('agent, 'alt) apref-profile = 'alt list multiset

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

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

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

\langle proof \rangle

lemma (in pref-profile-wf) anonymous-profile-def:
  anonymous-profile \ R = \text{image-mset (weak-ranking} \ \circ \ \ R \ \text{)} \ \text{(mset-set agents)}
\langle proof \rangle

lemma (in pref-profile-wf) anonymous-profile-permute:
  assumes \ \sigma \ \text{permutes alts} \ \text{finite agents}
  shows \ \text{anonymous-profile} \ \text{(permute-profile} \ \sigma \ \text{R)} =
  \text{image-mset (map} \ ((\') \ \sigma) \ \text{)} \ \text{(anonymous-profile} \ \text{R)}
\langle proof \rangle

lemma (in pref-profile-wf) anonymous-profile-update:
  assumes \ i: \ i \in \text{agents} \ \text{and fin [simp]: finite agents and total-preorder-on alts} \ \text{Ri'}
  shows \ \text{anonymous-profile} \ \text{(R(i := Ri')} \ \text{)} =
  \text{anonymous-profile} \ \text{R} - \ \text{\{#weak-ranking} \ \text{(R i)}\}\ + \ \text{\{#weak-ranking} \ \text{Ri'}\}\$
\langle proof \rangle

2.5 Preference profiles from lists

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

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

lemma prefs-from-table-wfI:
  assumes \ \text{agents} \ \neq \ \text{} \ \land \ \text{alts} \ \neq \ \text{} \ \land \ \text{distinct (map} \ \text{fst} \ \text{xss})
  assumes \ \text{set (map} \ \text{fst} \ \text{xss}) = \ \text{agents}
  assumes \ \bigwedge x. \ \text{x} \ \in \ \text{set (map} \ \text{snd} \ \text{xss}) \ \Rightarrow \ \bigcup \text{(set} \ \text{x}) = \text{alts}
  assumes \ \bigwedge x. \ \text{x} \ \in \ \text{set (map} \ \text{snd} \ \text{xss}) \ \Rightarrow \ \text{is-finite-weak-ranking} \ \text{x}
  shows \ \text{prefs-from-table-wf agents} \ \text{alts} \ \text{xss}
\langle proof \rangle

lemma prefs-from-table-wfD:
  assumes \ \text{prefs-from-table-wf agents} \ \text{alts} \ \text{xss}
  shows \ \text{agents} \ \neq \ \text{} \ \land \ \text{alts} \ \neq \ \text{} \ \land \ \text{distinct (map} \ \text{fst} \ \text{xss})
  and \ \text{set (map} \ \text{fst} \ \text{xss}) = \ \text{agents}
  and \ \bigwedge x. \ \text{x} \ \in \ \text{set (map} \ \text{snd} \ \text{xss}) \ \Rightarrow \ \bigcup \text{(set} \ \text{x}) = \text{alts}
  and \ \bigwedge x. \ \text{x} \ \in \ \text{set (map} \ \text{snd} \ \text{xss}) \ \Rightarrow \ \text{is-finite-weak-ranking} \ \text{x}
\langle proof \rangle

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

lemma prefs-from-table-eqI:
  assumes distinct (map fst xs) distinct (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 \( \text{prefs-from-table-wf agents alts xss i} \not\in \text{agents} \)
  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 \( x \) \( x' \)
  assumes \( i \in \text{set (map fst xs)} \)
  defines \( x'' \equiv \text{map (λ(j, y). if j = i then (j, x) else (j, y)) xs} \)
  shows \( \text{(prefs-from-table xs)}(i := \text{of-weak-ranking x} = \text{prefs-from-table x's} \text{?lhs = ?rhs}) \)
  ⟨proof⟩

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

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

2.6 Automatic evaluation of preference profiles

lemma eval-prefs-from-table [simp]:
\[
\text{pref}s\text{-from-table} \ ((i, y) \ # \ xs) \ i = \text{of-weak-ranking} \ y \\
i \neq j \implies \text{pref}s\text{-from-table} \ ((j, y) \ # \ xs) \ i = \text{pref}s\text{-from-table} \ xs \ i
\]

**Lemma** eval-of-weak-ranking [simp]:

\[
\begin{align*}
a \notin \bigcup (\text{set} \ xs) & \implies \neg \text{of-weak-ranking} \ xs \ a \ b \\
b \in x & \implies a \in \bigcup (\text{set} \ (x \# \ xs)) \implies \text{of-weak-ranking} \ (x \ # \ xs) \ a \ b \\
b \notin x & \implies \text{of-weak-ranking} \ (x \ # \ xs) \ a \ b \iff \text{of-weak-ranking} \ xs \ a \ b
\end{align*}
\]

**Definition** of-weak-ranking-Collect-ge where

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

**Lemma** eval-Collect-of-weak-ranking:

\[
\text{Collect} \ (\text{of-weak-ranking} \ xs \ x) = \text{of-weak-ranking-Collect-ge} \ (\text{rev} \ xs) \ x
\]

**Lemma** of-weak-ranking-Collect-ge-empty [simp]:

\[
\text{of-weak-ranking-Collect-ge} \ [] \ x = \{
\]

**Lemma** of-weak-ranking-Collect-ge-Cons [simp]:

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

**Lemma** of-weak-ranking-Collect-ge-Cons’:

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

**Lemma** anonymise-prefs-from-table:

\[
\begin{align*}
\text{assumes} & \ \text{pref}s\text{-from-table-wf} \ \text{agents} \ \text{alts} \ xs \\
\text{shows} & \ \text{anonymous-profile} \ (\text{pref}s\text{-from-table} \ xs) = \text{mset} \ (\text{map} \ \text{snd} \ xs)
\end{align*}
\]

**Lemma** prefs-from-table-agent-permutation:

\[
\begin{align*}
\text{assumes} \ & \ \text{wf}: \ \text{pref}s\text{-from-table-wf} \ \text{agents} \ \text{alts} \ xs \ \text{pref}s\text{-from-table-wf} \ \text{agents} \ \text{alts} \ ys \\
\text{assumes} \ & \ \text{mset-eq}: \ \text{mset} \ (\text{map} \ \text{snd} \ xs) = \text{mset} \ (\text{map} \ \text{snd} \ ys) \\
\text{obtains} \ & \ \pi \ \text{where} \ \pi \ \text{permutes} \ \text{agents} \ \text{pref}s\text{-from-table} \ xs \circ \ \pi = \ \text{pref}s\text{-from-table} \ ys
\end{align*}
\]

24
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. 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)) ⇒
  (λi. 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⟩
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

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)
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⟩
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:** $A \colon \forall x \ y. \ le \ x \ y \ \Rightarrow \ le \ y \ x \ \Rightarrow \ f \ x = f \ y$
- **Shows:** $\exists \varepsilon > 0. \ vnm\text{-utility carrier} \ le \ (\lambda x. \ u \ x + \varepsilon \ * \ f \ x)$

**Lemma diff-epsilon:**
- **Assumes:** $A \colon \forall x \ y. \ le \ x \ y \ \Rightarrow \ le \ y \ x \ \Rightarrow \ f \ x = f \ y$
- **Shows:** $\exists \varepsilon > 0. \ vnm\text{-utility carrier} \ le \ (\lambda x. \ u \ x - \varepsilon \ * \ f \ x)$

**4 Stochastic Dominance**

**Theory Stochastic-Dominance**

**Imports**
- Complex-Main
- $HOL\text{-Probability}.Probability$
- 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 $\Rightarrow$ '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 (\lambda- -. False) = (\lambda- -. False)**

**End**
lemma SD-is-preorder: preorder-on (lotteries-on { x. R x x }) (SD R)
⟨proof⟩

context preorder-on
begin

lemma SD-preorder:
  p ≥[SD(le)] q ⟷ p ∈ lotteries-on carrier ∧ q ∈ lotteries-on carrier ∧
  (∀ x∈carrier. measure-pmf.prob p (preferred-alts le x) ≥
  measure-pmf.prob q (preferred-alts le x))
⟨proof⟩

lemma SD-preorderI [intro?]:
  assumes p ∈ lotteries-on carrier q ∈ lotteries-on carrier
  assumes ∀ x. x ∈ carrier ⇒
  measure-pmf.prob p (preferred-alts le x) ≥ measure-pmf.prob q
  shows p ≥[SD(le)] q
⟨proof⟩

lemma SD-preorderD:
  assumes p ≥[SD(le)] q
  shows p ∈ lotteries-on carrier q ∈ lotteries-on carrier
  and ∀ x. x ∈ carrier ⇒
  measure-pmf.prob p (preferred-alts le x) ≥ measure-pmf.prob q
⟨proof⟩

lemma SD-refl' [simp]: p ≤[SD(le)] p ⟷ p ∈ lotteries-on carrier
⟨proof⟩

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

lemma SD-singleton-left:
  assumes x ∈ carrier q ∈ lotteries-on carrier
  shows return-pmf x ≤[SD(le)] q ⟷ (∀ y∈set-pmf q. x ≤[le] y)
⟨proof⟩

lemma SD-singleton-right:
  assumes x: x ∈ carrier and q: q ∈ lotteries-on carrier
  shows q ≤[SD(le)] return-pmf x ⟷ (∀ y∈set-pmf q. y ≤[le] x)
⟨proof⟩

lemma SD-strict-singleton-left:
  assumes x ∈ carrier q ∈ lotteries-on carrier
  shows return-pmf x ≤(SD(le)] q ⟷ (∀ y∈set-pmf q. x ≤[le] y) ∧ (∃ y∈set-pmf q. (x ∼[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 $i: i \in \text{agents}$

begin

interpretation $R_i: \text{preorder-on alts} R i$ (proof)

lemmas SD-singleton-left = $R_i.SD-singleton-left$
lemmas SD-singleton-right = $R_i.SD-singleton-right$
lemmas SD-strict-singleton-left = $R_i.SD-strict-singleton-left$
lemmas SD-strict-singleton-right = $R_i.SD-strict-singleton-right$
lemmas SD-singleton = $R_i.SD-singleton$
lemmas SD-strict-singleton = $R_i.SD-strict-singleton$

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 $p \succeq [SD(R i)] q \iff p \in \text{lotteries-on alts} \land q \in \text{lotteries-on alts}$

30
\( \forall x \in \text{alts}. \, \text{measure-pmf}. \text{prob} \ p (\text{preferred-alts} \ (R \ i) \ x) \geq \text{measure-pmf}. \text{prob} \ q (\text{preferred-alts} \ (R \ i) \ x) \)

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

**lemma SD-pref-profileI [intro?]:**

**assumes** \( i \in \text{agents} \, \ p \in \text{lotteries-on alts} \, \ q \in \text{lotteries-on alts} \)

**assumes** \( \bigwedge x. \, x \in \text{alts} \implies \text{measure-pmf}. \text{prob} \ p (\text{preferred-alts} \ (R \ i) \ x) \geq \text{measure-pmf}. \text{prob} \ q (\text{preferred-alts} \ (R \ i) \ x) \)

**shows** \( p \geq [\text{SD}(R \ i)] \, q \)

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

**lemma SD-pref-profileD:**

**assumes** \( i \in \text{agents} \, \ p \geq [\text{SD}(R \ i)] \, q \)

**shows** \( p \in \text{lotteries-on alts} \, q \in \text{lotteries-on alts} \)

and \( \bigwedge x. \, x \in \text{alts} \implies \text{measure-pmf}. \text{prob} \ p (\text{preferred-alts} \ (R \ i) \ x) \geq \text{measure-pmf}. \text{prob} \ q (\text{preferred-alts} \ (R \ i) \ x) \)

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

**end**

### 4.3 SD efficient lotteries

**definition** SD-efficient :: ('agent, 'alt) pref-profile ⇒ 'alt lottery ⇒ bool where

**SD-efficient-auxdef:**

SD-efficient \( R \, p \leftrightarrow \neg(\exists q \in \text{lotteries-on \ {x.\ E. \ R \ i \ x \ x}. \, q \succ [\text{Pareto} \ (SD \circ R)] \, p) \)

**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-efficient \( R \, p \leftrightarrow \neg(\exists q \in \text{lotteries-on alts}. \, q \succ [\text{Pareto} \ (SD \circ R)] \, p) \)

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

**lemma SD-efficient-def':**

SD-efficient \( R \, p \leftrightarrow \neg(\exists q \in \text{lotteries-on alts}. \, (\forall i \in \text{agents}. \, q \succeq [\text{SD}(R \ i)] \, p) \land (\exists i \in \text{agents}. \, q \succ [\text{SD}(R \ i)] \, p)) \)

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

**lemma SD-inefficientI:**

31
assumes $q \in \text{lotteries-on alts } \land i. \ i \in \text{agents } \Longrightarrow q \succeq [\text{SD}(R \ i)] \ p
$

shows $\neg \text{SD-efficient } R \ p$

(proof)

lemma SD-inefficientI':

assumes $q \in \text{lotteries-on alts } \land i. \ i \in \text{agents } \Longrightarrow q \succeq [\text{SD}(R \ i)] \ p
$

shows $\neg \text{SD-efficient } R \ p$

(proof)

lemma SD-inefficientE:

assumes $\neg \text{SD-efficient } R \ p
$

obtains $q \ i$ where

$q \in \text{lotteries-on alts } \land i. \ i \in \text{agents } \Longrightarrow q \succeq [\text{SD}(R \ i)] \ p
$

(i. \ i \in \text{agents } q \succeq [\text{SD}(R \ i)] \ p
$

(proof)

lemma SD-efficientD:

assumes $\text{SD-efficient } R \ p \ q \in \text{lotteries-on alts }
\land \land i. \ i \in \text{agents } \Longrightarrow q \succeq [\text{SD}(R \ i)] \ p
$

and $\exists i \in \text{agents}. \ q \succeq [\text{SD}(R \ i)] \ p
$

shows $\exists (q \succeq [\text{SD}(R \ i)] \ p
$

(proof)

lemma SD-efficient-singleton-iff:

assumes [simp]: $x \in \text{alts}$

shows $\text{SD-efficient } R \ (\text{return-pmf } x) \IF x \notin \text{pareto-losers } R$

(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.

category finite-total-preorder-on

begin

abbreviation is-vnm-utility $\equiv \text{vnm-utility carrier le }$

lemma utility-weak-ranking-index:

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

(proof)

lemma SD-iff-expected-utilities-le:

assumes $p \in \text{lotteries-on carrier } q \in \text{lotteries-on carrier}$

shows $p \succeq [\text{SD}(le)] \ q \IF$

$(\forall u. \ \text{is-vnm-utility } u \IF \text{measure-pmf}.\text{expectation } p \ u \leq \text{measure-pmf}.\text{expectation}$
\( q u \)
\( \langle \text{proof} \rangle \)

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

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

\textbf{end}

\textbf{end}

\textbf{theory} SD-Efficiency
\textbf{imports} Complex-Main Preference-Profiles Lotteries Stochastic-Dominance
\textbf{begin}

\textbf{context} pref-profile-wf
\textbf{begin}

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

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

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

\textbf{lemma} SD-efficient-iff:

33
\textbf{lemma SD-efficient-no-pareto-loser:}
\begin{itemize}
  \item \textbf{assumes} efficient: SD-efficient \( R \) \( p \) and \( p \)-wf: \( p \in \text{lotteries-on alts} \)
  \item \textbf{shows} set-pmf \( p \cap \text{pareto-losers} \) \( R \) = \{\}
\end{itemize}
\begin{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.
\end{proof}

4.5 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.

\textbf{private definition improve-lottery :: 'alt lottery \Rightarrow 'alt lottery where}
\begin{itemize}
  \item \textbf{assumes} \neg SD-efficient \( R \) \( q \)
  \item \textbf{defines} \( r \equiv \text{improve-lottery} \ q \)
  \item \textbf{shows} \( r \in \text{lotteries-on alts} \) \( r >\)\[\text{Pareto}(SD \circ R)\] \( q \)
  \( \land \forall r' \in \text{lotteries-on alts} \) \( r' >\)\[\text{Pareto}(SD \circ R)\] \( q \)
  \( \land \neg (\exists r' \in A. \text{set-pmf} \ r' \subset \text{set-pmf} \ r) \)
\end{itemize}
\begin{proof}
  \begin{itemize}
    \item fun sd-chain :: nat \Rightarrow 'alt lottery option where
    \item sd-chain 0 = Some \( p \)
    \item sd-chain (Suc n) =
      \begin{cases}
        \text{None} & \text{if SD-efficient} \ R \ p \ \text{then None else Some (improve-lottery} \ p)\\
        \text{Some} \ p & \text{if SD-efficient} \ R \ p
      \end{cases}
  \end{itemize}
\end{proof}
private lemma sd-chain-None-propagate:
  \( m \geq n \implies \text{sd-chain } n = \text{None} \implies \text{sd-chain } m = \text{None} \)

⟨proof⟩

lemma sd-chain-Some-propagate:
  \( m \geq n \implies \text{sd-chain } m = \text{Some } q \implies \exists q'. \text{sd-chain } n = \text{Some } q' \)

⟨proof⟩

lemma sd-chain-NoneD:
  \( \text{sd-chain } n = \text{None} \implies \exists n \ p. \text{sd-chain } n = \text{Some } p \land \text{SD-efficient } R \ p \)

⟨proof⟩

lemma sd-chain-lottery: \( \text{sd-chain } n = \text{Some } q \implies q \in \text{lotteries-on alts} \)

⟨proof⟩

lemma sd-chain-Suc:
  assumes \( \text{sd-chain } m = \text{Some } q \)
  assumes \( \text{sd-chain } (\text{Suc } m) = \text{Some } r \)
  shows \( q \prec [\text{Pareto}(SD \circ R)] r \)

⟨proof⟩

lemma sd-chain-strictly-preferred:
  assumes \( m < n \)
  assumes \( \text{sd-chain } m = \text{Some } q \)
  assumes \( \text{sd-chain } n = \text{Some } s \)
  shows \( q \prec [\text{Pareto}(SD \circ R)] s \)

⟨proof⟩

lemma sd-chain-preferred:
  assumes \( m \leq n \)
  assumes \( \text{sd-chain } m = \text{Some } q \)
  assumes \( \text{sd-chain } n = \text{Some } s \)
  shows \( q \preceq [\text{Pareto}(SD \circ R)] s \)

⟨proof⟩

lemma SD-efficient-lottery-exists:
  obtains \( q \) where \( q \in \text{lotteries-on alts } q \succeq [\text{Pareto}(SD \circ R)] p \land \text{SD-efficient } R \ p \)

⟨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

35
begin

5.1 Basic Social Choice definitions

context election
begin

The set of lotteries, i.e. the probability mass functions on the type 'alt whose support is a subset of the alternative set.

abbreviation lotteries where
lotteries ≡ lotteries-on alts

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

abbreviation lottery-prob :: 'alt lottery ⇒ 'alt set ⇒ real where
lottery-prob ≡ measure-pmf.prob

lemma lottery-prob-alts-superset:
assumes p ∈ lotteries alts ⊆ A
shows lottery-prob p A = 1
⟨proof⟩

lemma lottery-prob-alts: p ∈ lotteries ⇒ lottery-prob p 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 :: 'agent set and alts :: 'alt set +
fixes sds :: ('agent, 'alt) pref-profile ⇒ 'alt lottery
assumes sds-uf: is-pref-profile R ⇒ sds R ∈ 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 :: 'agent set and alts :: 'alt set and sds +
assumes anonymous: π permutes agents ⇒ is-pref-profile R ⇒ sds (R ∘ π) = 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
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 =
\( \text{image-mset} \ (\text{map} \ ((\cdot) \ \sigma)) \ \text{(anonymous-profile} \ R2) \)
\( \text{shows} \ sds \ R1 = \text{map-pmf} \ \sigma \ (sds \ R2) \)
\( \langle \text{proof} \rangle \)

**Lemma sds-anonymous-neutral’**:
- **Assumes** \( \text{perm: } \sigma \ \text{permutes alts} \ \text{and} \ \text{wf: } \text{is-pref-profile} \ R1 \ \text{is-pref-profile} \ R2 \)
- **Assumes** \( \text{eq: } \text{anonymous-profile} \ R1 = \text{image-mset} \ (\text{map} \ ((\cdot) \ \sigma)) \ \text{(anonymous-profile} \ R2) \)
- **Shows** \( \text{pmf} \ (sds \ R1) \ (\sigma \ x) = \text{pmf} \ (sds \ R2) \ x \)
\( \langle \text{proof} \rangle \)

**Lemma sds-automorphism**:
- **Assumes** \( \text{perm: } \sigma \ \text{permutes alts} \ \text{and} \ \text{wf: } \text{is-pref-profile} \ R \)
- **Assumes** \( \text{eq: } \text{image-mset} \ (\text{map} \ ((\cdot) \ \sigma)) \ \text{(anonymous-profile} \ R) = \text{anonymous-profile} \ R \)
- **Shows** \( \text{map-pmf} \ \sigma \ (sds \ R) = sds \ R \)
\( \langle \text{proof} \rangle \)

**Lemma an-sds-automorphism-aux**:
- **Assumes** \( \text{wf: } \text{prefs-from-table-wf} \ \text{agents} \ \text{alts} \ \text{yss} \ R \equiv \text{prefs-from-table} \ \text{yss} \)
- **Assumes** \( \text{an: } \text{an-sds} \ \text{agents} \ \text{alts} \ \text{sds} \)
- **Assumes** \( \text{eq: } \text{mset} \ (\text{map} \ ((\cdot) \ (\text{permutation-of-list} \ \text{xs}))) \ \circ \ \text{snd} \ \text{yss} = \text{mset} \ (\text{map} \ \text{snd} \ \text{yss}) \)
- **Assumes** \( \text{perm: } \text{set} \ (\text{map} \ \text{fst} \ \text{xs}) \ \subseteq \ \text{alts} \ \text{set} \ (\text{map} \ \text{snd} \ \text{xs}) = \text{set} \ (\text{map} \ \text{fst} \ \text{xs}) \)
  - **And** \( x: \ x \in \text{alts} \ y = \text{permutation-of-list} \ \text{xs} \ x \)
- **Shows** \( \text{pmf} \ (sds \ R) \ x = \text{pmf} \ (sds \ R) \ y \)
\( \langle \text{proof} \rangle \)

### 5.5 Ex-post efficiency

**Locale ex-post-efficient-sds** = \( \text{social-decision-scheme} \ \text{agents} \ \text{alts} \ \text{sds} \)
- **For** \( \text{agents} :: \ 'agent \ \text{set} \ \text{and} \ \text{alts} :: \ 'alt \ \text{set} \ \text{and} \ \text{sds} + \)
- **Assumes** \( \text{ex-post-efficient: } \Rightarrow \ \text{set-pmf} \ (sds \ R) \cap \text{pareto-losers} \ R = \{\} \)

**Begin**

**Lemma ex-post-efficient’**:
- **Assumes** \( \text{is-pref-profile} \ R \ y \succ [\text{Pareto}(R)] \ x \)
- **Shows** \( \text{pmf} \ (sds \ R) \ x = 0 \)
\( \langle \text{proof} \rangle \)

**Lemma ex-post-efficient”’**:
- **Assumes** \( \text{is-pref-profile} \ R \ i \in \text{agents} \ \forall i \in \text{agents}. \ y \succeq [R \ i] \ x \ \neg y \preceq [R \ i] \ x \)
- **Shows** \( \text{pmf} \ (sds \ R) \ x = 0 \)
\( \langle \text{proof} \rangle \)

38
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.

```plaintext
locale sd-efficient-sds = social-decision-scheme agents alts sds
  for agents :: agent set and alts :: alt set and sds +
  assumes SD-efficient: is-pref-profile R → 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 ∀ i. i ∈ agents → q ⪰ [SD(R i)] sds R i ∈ agents q ⪰ [SD(R i)] sds R
  shows 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 ∃ 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 ∀ i∈agents. p ⪰ [SD(R i)] pmf-of-set A i ∈ agents
  ¬p ⪰ [SD(R i)] pmf-of-set A
  shows ∃ 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 Ri' if the lottery for obtained for R is at least as good for i as the lottery obtained when i misrepresents her preferences as Ri'.

**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** \(∀x. x ∈ \text{alts} \implies \text{lottery-prob} (\text{sds} (R(i := Ri'))) (\text{preferred-alts} (R i) x) \leq \text{lottery-prob} (\text{sds} R) (\text{preferred-alts} (R i) x)\)

**shows** strongly-strategyproof-profile R i Ri'
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 π permutes agents agents ′ ⊆ agents ′
  shows lift-pref-profile agents alts agents ′ alts ′ (R ∘ π) = lift-pref-profile agents alts agents ′ alts ′ R ∘ π
⟨proof⟩

lemma lift-pref-profile-permute-alts:
  assumes σ permutes alts alts ′ ⊆ alts ′
  shows lift-pref-profile agents alts agents ′ alts ′ (permute-profile σ R) = permute-profile σ (lift-pref-profile agents alts agents ′ alts ′ R)
⟨proof⟩

lemma lotteries-on-subset: A ⊆ B ⇒ p ∈ lotteries-on A ⇒ p ∈ lotteries-on B
⟨proof⟩

lemma lottery-prob-carrier: p ∈ lotteries-on A ⇒ measure-pmf.prob p A = 1
⟨proof⟩

context
  fixes agents alts R agents ′ alts ′ R ′
  assumes R-wf: pref-profile-wf agents alts R
  assumes election: agents ⊆ agents ′ alts ⊆ alts ′ alts ′ ≠ {} agents ≠ {} finite alts ′
  defines R′ ≡ lift-pref-profile agents alts agents ′ alts ′ R
begin
interpretation R: pref-profile-wf agents alts R
⟨proof⟩
interpretation R ′: pref-profile-wf agents ′ alts ′ R ′
⟨proof⟩

lemma lift-pref-profile-strict-iff:
x ≺[lift-pref-profile agents alts agents ′ alts ′ R i] y ←→ i ∈ agents ∧ (y ∈ alts ∧ x ∈ alts ′ − alts) ∨ x ≺[R i] y
⟨proof⟩

lemma preferred-alts-lift-pref-profile:
  assumes i: i ∈ agents ′ and x: x ∈ alts ′
  shows preferred-alts (R ′ i) x = (if i ∈ agents ∧ x ∈ alts then preferred-alts (R i) x else alts ′)
⟨proof⟩

lemma lift-pref-profile-Pareto-iff:
x ≳[Pareto(R ′)] y ←→ x ∈ alts ′ ∧ y ∈ alts ′ ∧ (x ∉ alts ∨ x ≤[Pareto(R)] y)
⟨proof⟩

lemma lift-pref-profile-Pareto-strict-iff:
x ≪[Pareto(R ′)] y ←→ x ∈ alts ′ ∧ y ∈ alts ′ ∧ (x ∉ alts ∧ y ∈ alts ∨ x
lemma pareto-losers-lift-pref-profile:
  shows pareto-losers $R' = \text{pareto-losers } R \cup (\text{alts} - \text{alts})$
⟨proof⟩

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

lemmas lift-SD-iff = lift-SD-iff-agent lift-SD-iff-nonagent

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

end

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

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

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

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'

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
proof

end

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⟩
We define Random Dictatorship as a social decision scheme on total preorders (i.e., agents are allowed to have ties in their rankings) by first selecting an agent uniformly at random and then selecting one of that agent’s 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:

```plaintext
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 =

```plaintext
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 \( RD R = \text{map-pmf} \ (\text{favorite } R) \ (\text{pmf-of-set agents}) \)

\langle \text{proof} \rangle

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

\textbf{lemma pmf-random-dictatorship}:
\textbf{assumes} is-pref-profile \( R \)
\textbf{shows} pmf \( (RD R) \ x = \sum_{i \in \text{agents}} \text{indicator} \ (\text{favorites } R \ i) \ x / \text{real} \ (\text{card} \ (\text{favorites } R \ i)) / \text{real} \ (\text{card} \ \text{agents}) \)
\langle \text{proof} \rangle

\textbf{sublocale} RD: social-decision-scheme agents alts RD
\langle \text{proof} \rangle

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.

\section*{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.

\textbf{sublocale} RD: anonymous-sds agents alts RD
\langle \text{proof} \rangle

\section*{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.

\textbf{sublocale} RD: neutral-sds agents alts RD
\langle \text{proof} \rangle
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.

\textit{sublocale} RD: strongly-strategyproof-sds agents alts RD
\langle proof \rangle
\end

8 Random Serial Dictatorship

\textit{theory} Random-Serial-Dictatorship
\textit{imports} Complex-Main Social-Decision-Schemes Random-Dictatorship
\textit{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.

\textit{definition} random-serial-dictatorship ::
  \textquote{agent set} \Rightarrow \textquote{alt set} \Rightarrow \textquote{(agent, alt) pref-profile} \Rightarrow \textquote{alt lottery} \textit{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.

\textit{lemma} random-serial-dictatorship-empty [simp];
  random-serial-dictatorship \{\} alts R = pmf-of-set alts
\langle proof \rangle
lemma random-serial-dictatorship-nonempty:
finite agents \(\implies\) agents \(\neq\) \(\{}\) \(\implies\)
random-serial-dictatorship agents alts R =
do { i \leftarrow \text{pmf-of-set agents}; random-serial-dictatorship (agents - \{i\}) \text{ (Max-wrt-among} (R i) \text{ alts)} R }

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 \textit{foldr} does not require generalisation over the set of alternatives and is therefore much easier than over \textit{foldl}.

definition rsd-winners where
rsd-winners R alts agents = \text{foldr} (\lambda i \text{ alts. Max-wrt-among} (R i) \text{ alts}) agents alts

lemma rsd-winners-empty [simp]: rsd-winners R alts \([]\) = alts

lemma rsd-winners-Cons [simp]:
rsd-winners R alts (i \# agents) = Max-wrt-among (R i) (rsd-winners R alts agents)

lemma rsd-winners-map [simp]:
rsd-winners R alts (map f agents) = rsd-winners (R \circ f) alts agents

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

lemma random-serial-dictatorship-altdef:
assumes finite agents
shows random-serial-dictatorship agents alts R =
do { agents' \leftarrow \text{pmf-of-set} \text{ (permutations-of-set agents)}; \text{pmf-of-set} \text{ (rsd-winners R 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 random-serial-dictatorship-foldl:
assumes finite agents
shows random-serial-dictatorship agents alts R =
do {
agents' ← pmf-of-set (permutations-of-set agents);

pmf-of-set \( \langle \text{foldl} \ (\lambda \text{alts} \ i. \ \text{Max-wrt-among} \ (R \ i) \ \text{alts}) \ \text{alts} \ \text{agents}' \rangle \)

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

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
  \( \text{RSD-pareto-eqclass} \ \text{agents} \ \text{alts} \ R \ A \leftarrow \)
  \( A \neq \{} \ \land \ A \subseteq \text{alts} \ \land \ (\forall x \in A. \ \forall y \in \text{alts}. \ y \in A \leftarrow (\forall i \in \text{agents}. \ R \ i \ x \ y)) \)

lemma RSD-pareto-eqclassI:
  assumes \( A \neq \{} \ \land \ A \subseteq \text{alts} \ \land \ (\forall x \in A. \ \forall y \in \text{alts}. \ y \in A \leftarrow (\forall i \in \text{agents}. \ R \ i \ x \ y)) \)
  shows \( \text{RSD-pareto-eqclass} \ \text{agents} \ \text{alts} \ R \ A \)
  \( \langle \text{proof} \rangle \)

lemma RSD-pareto-eqclassD:
  assumes \( \text{RSD-pareto-eqclass} \ \text{agents} \ \text{alts} \ R \ A \)
  shows \( A \neq \{} \ \land \ A \subseteq \text{alts} \ \land \ (\forall x \in A. \ \forall y \in \text{alts}. \ y \in A \leftarrow (\forall i \in \text{agents}. \ R \ i \ x \ y)) \)
  \( \langle \text{proof} \rangle \)

lemma RSD-pareto-eqclass-indiff-set:
  assumes \( \text{RSD-pareto-eqclass} \ \text{agents} \ \text{alts} \ R \ A \) \( \text{i} \in \text{agents} \ x \in A \ y \in A \)
  shows \( R \ i \ x \ y \)
  \( \langle \text{proof} \rangle \)

lemma RSD-pareto-eqclass-empty [simp, intro!]:
  \( \text{alts} \neq \{} \ \implies \ \text{RSD-pareto-eqclass} \ \{\} \ \text{alts} \ R \ \text{alts} \)
  \( \langle \text{proof} \rangle \)

lemma (in pref-profile-wf) RSD-pareto-eqclass-insert:
  assumes \( \text{RSD-pareto-eqclass} \ \text{agents}' \ \text{alts} \ R \ A \) \( \text{finite} \ \text{alts} \)
  \( \text{i} \in \text{agents} \ \text{agents}' \subseteq \text{agents} \)
  shows \( \text{RSD-pareto-eqclass} \ (\text{insert} \ i \ \text{agents}') \ \text{alts} \ (\text{Max-wrt-among} \ (R \ i) \ A) \)
  \( \langle \text{proof} \rangle \)

50
8.1.2 Facts about RSD winners

context pref-profile-wf

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' \neq \{\}
  set agents' ⊆ agents
  alts' ⊆ alts
  shows rsd-winners R alts' agents' \neq \{\}
(\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' \neq \{\}
  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' \neq \{\}
  set agents' = agents
  shows RSD-pareto-eqclass 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 \neq \{\} \land (\forall x \in A. \forall y \in A. S x y)
private lemma indiff-set-mono: \( \text{indiff } S A \Rightarrow B \subseteq A \Rightarrow B \neq \{\} \Rightarrow \text{indiff } S B \)  
(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  
\( \text{RSD-set-rel } S A B \leftarrow\rightarrow \text{indiff-set } S B \land (\forall x \in A. \forall y \in B. S x y) \)  

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 \( \text{finite carrier } A \subseteq \text{carrier } A \neq \{\} \) \( \text{total-preorder-on } \text{carrier } S \)  
shows \( \text{indiff-set } S (\text{Max-wrt-among } S A) \)  
(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 \( \text{wf: total-preorder-on } \text{alts } R_i' \)  
and \( i: i \in \text{agents and set } \text{agents}' \subseteq \text{agents } \) finite agents  
and finite: \( \text{finite } \text{alts and } \text{alts } \neq \{\} \)  
defines [simp]: \( w' \equiv \text{rsd-winners } (R(i := R_i')) \) \( \text{alts } \) and  
[simp]; \( w \equiv \text{rsd-winners } R \) \( \text{alts } \)  
sshows \( w' \) \( \text{agents}' = w \) \( \text{agents}' \lor \text{RSD-set-rel } (R i) (w' \text{agents}') (w \text{agents}') \)  
(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 \( \text{wf: total-preorder-on } \text{alts } R_i' \)  
and \( i: i \in \text{agents and set } \text{agents}' = \text{agents } \) finite agents  
and finite: \( \text{finite } \text{alts and } \text{alts } \neq \{\} \)  
defines [simp]: \( w' \equiv \text{rsd-winners } (R(i := R_i')) \) \( \text{alts } \) and  
[simp]; \( w \equiv \text{rsd-winners } R \) \( \text{alts } \)  
sshows \( \forall x \in w' \) \( \text{agents}' \). \( \forall y \in w \) \( \text{agents}' \). \( x \preceq_{[R \ i]} y \)  
(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** set-pmf (random-serial-dictatorship agents' alts' R) ⊆ alts'

 ⟨proof⟩

Permutation of alternatives commutes with RSD winners.

**Lemma** rsd-winners-permute-profile:
- **Assumes** perm: σ permutes alts and set agents' ⊆ agents
- **Shows** rsd-winners (permute-profile σ) R alts agents' = σ rsd-winners R alts agents'

 ⟨proof⟩

**Lemma** random-serial-dictatorship-singleton:
- **Assumes** finite agents finite alts agents' ⊆ agents x ∈ alts
- **Shows** random-serial-dictatorship agents' {x} R = 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** af: is-pref-profile R and unique: has-unique-favorites R
- **Shows** RSD R = 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.
sublocale RSD: anonymous-sds agents alts RSD
(proof)

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.

sublocale RSD: neutral-sds agents alts RSD
(proof)

8.2.5 Ex-post efficiency

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

sublocale RSD: ex-post-efficient-sds agents alts RSD
(proof)

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.

sublocale RSD: strongly-strategyproof-sds agents alts RSD
(proof)

end

theory Randomised-Social-Choice

imports
  Complex-Main
  SDS-Lowering
  Random-Dictatorship
  Random-Serial-Dictatorship

begin
9 Automatic definition of Preference Profiles

theory Preference-Profile-Cmd
imports
  Complex-Main
  ../Elections
keywords
  preference-profile :: thy-goal
begin

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 af: 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 af: 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 \texttt{prefs-from-table-wf agents alts xs} \\
shows \texttt{favorites-prefs-from-table xs i = favorites (prefs-from-table xs) i} \\
(proof)

\textbf{function} \texttt{weak-ranking-prefs-from-table where} \\
\texttt{i ≠ j ↦ weak-ranking-prefs-from-table ((i,x)#xs) j = weak-ranking-prefs-from-table xs j} \\
\texttt{i = j ↦ weak-ranking-prefs-from-table ((i,x)#xs) j = x} \\
\texttt{weak-ranking-prefs-from-table [] j = []} \\
(proof)

\textbf{termination} (proof)

\textbf{lemma} \texttt{eval-weak-ranking-prefs-from-table:} \\
\textbf{assumes} \texttt{prefs-from-table-wf agents alts xs} \\
\textbf{shows} \texttt{weak-ranking-prefs-from-table xs i = weak-ranking (prefs-from-table xs i)} \\
(proof)

\textbf{lemma} \texttt{eval-prefs-from-table-aux:} \\
\textbf{assumes} \texttt{R ≡ prefs-from-table xs prefs-from-table-wf agents alts xs} \\
\textbf{shows} \texttt{R i a b ↔ prefs-from-table xs i a b ∧ ¬prefs-from-table xs i b a} \\
\texttt{anonymous-profile R = mset (map snd xs)} \\
\texttt{election agents alts ↦ i ∈ set (map fst xs) ↦ preferred-alts (R i) x =} \\
\texttt{of-weak-ranking-Collect-ge (rev (the (map-of xs i))) x} \\
\texttt{election agents alts ↦ i ∈ set (map fst xs) ↦ favorites R i = favorites-prefs-from-table xs i} \\
\texttt{election agents alts ↦ i ∈ set (map fst xs) ↦ weak-ranking (R i) = weak-ranking-prefs-from-table xs i} \\
\texttt{election agents alts ↦ i ∈ set (map fst xs) ↦ favorite R i = the-elem (favorites-prefs-from-table xs i)} \\
\texttt{election agents alts ↦ has-unique-favorites R ↔ list-all (λz. is-singleton (hd (snd z))) xs} \\
(proof)

\textbf{lemma} \texttt{pref-profile-from-tableI′:} \\
\textbf{assumes} \texttt{R1 ≡ prefs-from-table xss prefs-from-table-wf agents alts xss} \\
\textbf{shows} \texttt{pref-profile-wf agents alts R1} \\
(proof)

\langle ML \rangle

end

theory \texttt{QSOpt-Exact} \\
imports \texttt{Complex-Main} \\
begin
10 Automatic Fact Gathering for Social Decision Schemes

declare theory SDS-Automation
import Preference-Profile-Cmd
QSOpt-Exact
../Social-Decision-Schemes

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: $P ∨ False \leftrightarrow P$ \(\langle\text{proof}\rangle\)

**lemmas** multiset-add-ac = add-ac[where \(?a = 'a\) multiset]

**lemma** less-or-eq-real:

\((x::real) < y ∨ x = y \leftrightarrow x \leq y ∨ y = x \leftrightarrow x \leq y\) \(\langle\text{proof}\rangle\)

**lemma** multiset-Diff-single-normalize:

*fixes* \(a\) \(<\) \(c\) \n*assumes* \(a \neq c\) \n*shows* \((\{\#a\#\} + B) - \{\#c\#\} = \{\#a\#\} + (B - \{\#c\#\})\) \(\langle\text{proof}\rangle\)

**lemma** ex-post-efficient-aux:

*assumes* \(\text{prefs-from-table-wf agents alts xss R} \equiv \text{prefs-from-table xss}\) \n*assumes* \(i\) \(\in\) agents \(\forall i\) \(\in\) agents. \(y \geq \text{[prefs-from-table xss i]} \) \(x \not\leq y \leq \text{[prefs-from-table xss i]} \(x\) \n*shows* ex-post-efficient-sds agents alts sds \(\rightarrow\) pmf \((sds R) x = 0\) \(\langle\text{proof}\rangle\)

**lemma** SD-inefficient-support-aux:

*assumes* \(R: \text{prefs-from-table-wf agents alts xss R} \equiv \text{prefs-from-table xss}\) \n*assumes* \(\text{as} = \[]\) \(\subseteq\) alts \(\text{distinct as A = set as}\) \n*assumes* \(ys: \forall x\in\text{set (map snd ys)}, 0 \leq x \) \(\text{sum-list (map snd ys)} = 1 \text{ set (map fst ys)} \subseteq\) alts \n*assumes* \(i: i\) \(\in\) agents \n*assumes* \(\text{SD1: } \forall i\in\text{agents. } \forall x\in\text{alts.}\) \n\(\text{sum-list (map snd (filter (λy. \text{prefs-from-table xss i x (fst y)) ys)))} \geq\) \(\text{real (length (filter (prefs-from-table xss i x) as))} / \text{real (length as)}\) \n*assumes* \(\text{SD2: } \exists x\in\text{alts. sum-list (map snd (filter (λy. \text{prefs-from-table xss i x (fst y)) ys)))} >\) \(\text{real (length (filter (prefs-from-table xss i x) as))} / \text{real (length as)}\) \n*shows* sd-efficient-sds agents alts sds \(\rightarrow\) \((\exists x\in A. \text{pmf (sds R) x = 0})\) \(\langle\text{proof}\rangle\)

**definition** pref-classes where

\(\text{pref-classes alts le = preferred-alts le ' alts - \{alts\}}\)
primrec pref-classes-lists where
pref-classes-lists [] = {}
| pref-classes-lists (xs#xss) = insert (\( \bigcup \) (set (xs#xss))) (pref-classes-lists xss)

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

lemma pref-classes-lists-append:
pref-classes-lists (xs @ ys) = (\( \bigcup \)) (\( \bigcup \) (set ys)) ' pref-classes-lists xs \cup pref-classes-lists ys
⟨proof⟩

lemma pref-classes-lists-aux:
assumes is-weak-ranking xss acc \( \cap \) (\( \bigcup \) (set xss)) = {}
shows pref-classes-lists-aux acc xss =
(insert acc ((\( \lambda \)A. A \cup acc) ' pref-classes-lists (rev xss)) \- {acc \cup \bigcup (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) \- {
\bigcup (set xss)}
⟨proof⟩

lemma pref-classes-of-weak-ranking-aux:
assumes is-weak-ranking xss
shows of-weak-ranking-Collect-ge xss ' (\( \bigcup \) (set xss)) = pref-classes-lists xss
⟨proof⟩

lemma eval-pref-classes-of-weak-ranking:
assumes \( \bigcup \) (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 \( \in \) lotteries-on carrier q \( \in \) lotteries-on carrier
shows p \( \preceq \) [SD(le)] q \( \iff \)
(\( \forall \) A\( \in \) pref-classes carrier le. measure-pmf.prob p A \leq \) measure-pmf.prob q A)
⟨proof⟩

end
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. 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) o 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