Randomised Social Choice

Manuel Eberl

February 23, 2021

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
10 Automatic Fact Gathering for Social Decision Schemes
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-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 ($\preceq$ - $[51,10,51]$ 60) where
$a \succeq [R] b \equiv b \preceq [R] a$

term $a \succeq [R] b \leftrightarrow b \preceq [R] a$

abbreviation (input) strongly-not-preferred ($\succ$ - $[51,10,51]$ 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: $a \preceq [le] b \implies b \preceq [le] c \implies a \preceq [le] c$
⟨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:
$a \in \text{carrier} \implies b \in \text{carrier} \implies \neg a \preceq [le] b \implies a \prec [le] b$
⟨proof⟩

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

1.4 Orders

locale order-on = preorder-on +
  assumes antisymmetric: $\text{le } x y \implies \text{le } y x \implies x = y$

locale linorder-on = order-on carrier $\text{le}$ + total-preorder-on carrier $\text{le}$ for carrier $\text{le}$

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 $\Rightarrow$ 'a set $\Rightarrow$ 'a set where
  Max-wrt-among $R$ $A$ = \{ $x \in A$. $R$ $x$ $x$ $\land$ ($\forall y \in A$. $R$ $x$ $y$ $\rightarrow$ $R$ $y$ $x$)\}

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

lemma Max-wrt-map-relation-vimage:
f −’ 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 −’ B ⊆ the-inv-into A f −’ B
⟨proof⟩

lemma Max-wrt-among-map-relation-bij-subset:
assumes bij \( f :: 'a \Rightarrow 'b \)
shows \( f \cdot \text{Max-wrt-among le } A \subseteq \text{Max-wrt-among } (\text{map-relation } (\text{inv f}) \text{ le }) (f \cdot A) \)

⟨proof⟩

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

⟨proof⟩

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

⟨proof⟩

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

⟨proof⟩

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

⟨proof⟩

end

context total-preorder-on
begin

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

⟨proof⟩

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

⟨proof⟩

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

⟨proof⟩

end

1.6 Weak rankings

inductive of-weak-ranking :: 'alt set list ⇒ 'alt relation where

8
i \leq j \implies i < \text{length } \textit{xs} \implies j < \text{length } \textit{xs} \implies x \in \textit{xs} ! i \implies y \in \textit{xs} ! j \implies x \geq \text{of-weak-ranking } \textit{xs} \ y

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

\textbf{proof}

\textbf{lemma} \textit{of-weak-ranking-Nil}' [code]: \text{of-weak-ranking } [] x y = \text{False}

\textbf{proof}

\textbf{lemma} \textit{of-weak-ranking-Cons} [code]:

\[
x \geq \text{of-weak-ranking} (\textit{z#zs}) \ y \iff x \in \textit{z} \land y \in \bigcup (\text{set} (\textit{z#zs})) \lor x \geq \text{of-weak-ranking} \textit{zs} \ y\\
\text{(is } ?\text{lhs }\iff ?\text{rhs})
\]

\textbf{proof}

\textbf{lemma} \textit{of-weak-ranking-indifference}:

\textbf{ assumes } A \in \text{set } \textit{xs} \ x \in A \ y \in A

\textbf{shows} \ x \geq \text{of-weak-ranking } \textit{xs} \ y

\textbf{proof}

\textbf{lemma} \textit{of-weak-ranking-map}:

\text{map-relation } f \ (\text{of-weak-ranking } \textit{xs}) = \text{of-weak-ranking} \ (\text{map } ((- ) f) \textit{xs})

\textbf{proof}

\textbf{lemma} \textit{of-weak-ranking-permute}':

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

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

\textbf{proof}

\textbf{lemma} \textit{of-weak-ranking-permute}:

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

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

\textbf{proof}

\textbf{definition} \textit{is-weak-ranking} where

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

(\forall i. j. i < \text{length } \textit{xs} \land j < \text{length } \textit{xs} \land i \neq j \implies \textit{xs} ! i \cap \textit{zs} ! j = \{\})

\textbf{definition} \textit{is-finite-weak-ranking} where

\textit{is-finite-weak-ranking} \textit{xs} \iff \textit{is-weak-ranking} \textit{xs} \land (\forall x \in \text{set } \textit{xs}. \text{finite } x)

\textbf{definition} \textit{weak-ranking} :: 'alt relation => 'alt set list where

\textit{weak-ranking} R = (\text{SOME } \textit{xs}. \textit{is-weak-ranking} \textit{xs} \land R = \textit{of-weak-ranking} \textit{xs})

\textbf{lemma} \textit{is-weak-rankingI} [intro]:

\textbf{ assumes } (\{\} \notin \text{set } \textit{xs} \land i. j. i < \text{length } \textit{xs} \implies j < \text{length } \textit{xs} \implies i \neq j \implies \textit{xs} ! i
\[ \cap \text{xs} \not\in j = \{\} \]

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

(\textit{proof})

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

(\textit{proof})

**lemma** \( \text{is-weak-rankingD:} \)

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

**shows** \( \text{xs} \not\in i \cap \text{xs} \not\in j = \{\} \)

(\textit{proof})

**lemma** \( \text{is-weak-ranking-iff:} \)

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

(\textit{proof})

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

(\textit{proof})

**lemma** \( \text{is-weak-ranking-map-inj:} \)

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

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

(\textit{proof})

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

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

(\textit{proof})

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

(\textit{proof})

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

(\textit{proof})

**lemma** \( \text{is-weak-ranking-Cons-empty [simp]:} \)

\( \neg \text{is-weak-ranking \, (\{\} \not\# \,xs) \} \)

(\textit{proof})

**lemma** \( \text{is-finite-weak-ranking-Cons-empty [simp]:} \)

\( \neg \text{is-finite-weak-ranking \, (\{\} \not\# \,xs) \} \)

(\textit{proof})

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

\( \text{is-weak-ranking \,[x]} \iff x \neq \{\} \)

(\textit{proof})

**lemma** \( \text{is-finite-weak-ranking-singleton [simp]:} \)

\( \text{is-finite-weak-ranking \,[x]} \iff x \neq \{\} \land \text{finite \,x} \)

(\textit{proof})
lemma is-weak-ranking-append:

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

(\text{proof})

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

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

(\text{proof})

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

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

(\text{proof})

primrec is-weak-ranking-aux where

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

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

lemma is-weak-ranking-aux:

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

(\text{proof})

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

\[ \text{is-weak-ranking} \left( xs \right) \iff \text{is-weak-ranking-aux} \ \{\} \ xs \]

(\text{proof})

lemma of-weak-ranking-altdef:

\[ \text{assumes} \ \text{is-weak-ranking} \left( xs \right) \ x \in \bigcup(\text{set} \, xs) \ y \in \bigcup(\text{set} \, xs) \]

\[ \text{shows} \ \text{of-weak-ranking} \left( xs \right) \ x \ y \iff \text{find-index} \ ((\in) \ x) \ xs \geq \text{find-index} \ ((\in) \ y) \ xs \]

(\text{proof})

lemma total-preorder-of-weak-ranking:

\[ \text{assumes} \ \bigcup(\text{set} \, xs) = A \]

\[ \text{assumes} \ \text{is-weak-ranking} \left( xs \right) \]

\[ \text{shows} \ \text{total-preorder-on} \ A \ (\text{of-weak-ranking} \ \left( xs \right)) \]

(\text{proof})

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

\[ \text{assumes} \ \text{is-weak-ranking} \ (A \# As) \]

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

(\text{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 \( \bigcup (\text{set} (x\#xs)) \) of-weak-ranking (x#xs)
  ⟨proof⟩

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

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

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

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

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

lemma Max-wrt-of-weak-ranking-Cons:
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))

\langle proof \rangle

\textbf{lemma} \textit{weak-ranking-aux-unique:}
\begin{itemize}
    \item \textbf{assumes} \textit{is-weak-ranking As of-weak-ranking As} = \textit{le}
    \item \textbf{shows} \textit{As} = \textit{weak-ranking-aux carrier}
\end{itemize}
\langle proof \rangle

\textbf{lemma} \textit{weak-ranking-total-preorder:}
\begin{itemize}
    \item \textit{is-weak-ranking (weak-ranking le) of-weak-ranking (weak-ranking le)} = \textit{le}
\end{itemize}
\langle proof \rangle

\textbf{lemma} \textit{weak-ranking-altdef:}
\begin{itemize}
    \item \textit{weak-ranking le} = \textit{weak-ranking-aux carrier}
\end{itemize}
\langle proof \rangle

\textbf{lemma} \textit{weak-ranking-Union:} \bigcup (\textit{set (weak-ranking le)}) = \textit{carrier}
\langle proof \rangle

\textbf{lemma} \textit{weak-ranking-unqiue:}
\begin{itemize}
    \item \textbf{assumes} \textit{is-weak-ranking As of-weak-ranking As} = \textit{le}
    \item \textbf{shows} \textit{As} = \textit{weak-ranking le}
\end{itemize}
\langle proof \rangle

\textbf{lemma} \textit{weak-ranking-permute:}
\begin{itemize}
    \item \textbf{assumes} \textit{f permutes carrier}
    \item \textbf{shows} \textit{weak-ranking (map-relation (inv f) le)} = \textit{map ((\rangle f) (weak-ranking le)}
\end{itemize}
\langle proof \rangle

\textbf{lemma} \textit{weak-ranking-index-unique:}
\begin{itemize}
    \item \textbf{assumes} \textit{is-weak-ranking xs i < length xs} \textit{j} < \textit{length xs} \textit{x} \in \textit{xs} ! \textit{i} \textit{x} \in \textit{xs} ! \textit{j}
    \item \textbf{shows} \textit{i} = \textit{j}
\end{itemize}
\langle proof \rangle

\textbf{lemma} \textit{weak-ranking-index-unique':}
\begin{itemize}
    \item \textbf{assumes} \textit{is-weak-ranking xs i < length xs} \textit{x} \in \textit{xs} ! \textit{i}
    \item \textbf{shows} \textit{i} = \textit{find-index ((\in) x) xs}
\end{itemize}
\langle proof \rangle

\textbf{lemma} \textit{weak-ranking-eqclass1:}
\begin{itemize}
    \item \textbf{assumes} \textit{A} \in \textit{set (weak-ranking le)} \textit{x} \in \textit{A} \textit{y} \in \textit{A}
    \item \textbf{shows} \textit{le x y}
\end{itemize}
\langle proof \rangle

\textbf{lemma} \textit{weak-ranking-eqclass2:}
\begin{itemize}
    \item \textbf{assumes} \textit{A: A} \in \textit{set (weak-ranking le)} \textit{x} \in \textit{A} \textit{and le: le x y le y x}
    \item \textbf{shows} \textit{y} \in \textit{A}
\end{itemize}
\langle proof \rangle

\textbf{lemma} \textit{hd-weak-ranking:}
assumes \( x \in \text{hd} (\text{weak-ranking le}) \) \( y \in \text{carrier} \)
shows \( \text{le} \ y \ x \)
(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** weak-ranking-index :: \( 'a \Rightarrow \text{nat} \)
where
weak-ranking-index \( x \) = find-index (\( \lambda A. \ x \in A \)) (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:
\( i < \text{length} (\text{weak-ranking le}) \implies x \in \text{weak-ranking le} ! \) \( i \implies \text{weak-ranking-index} \ x = i \)
(proof)

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

end

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

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

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

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

**lemma** weak-ranking-of-weak-ranking:
assumes \( \text{is-finite-weak-ranking} \ \text{xs} \)
shows  weak-ranking (of-weak-ranking xs) = xs
(proof)

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

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

definition preferred-alts :: 'alt relation \Rightarrow 'alt \Rightarrow 'alt set where
preferred-alts R x = \{y. y \geq [R] x\}

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

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

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

1.7 Rankings

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

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

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

lemma singleton-weak-ranking:
assumes A \in set (weak-ranking le)
shows  is-singleton A
(proof)
lemma  weak-ranking-ranking: weak-ranking le = map (λx. {x}) (ranking le)
(proof)
end
end

2 Preference Profiles

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

The type of preference profiles

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

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

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

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

lemma  finite-alts [simp]: finite alts
(proof)

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

lemma  not-outside:
assumes $x \preceq [R \ i] \ y$
shows $i \in \text{agents} \ x \in \text{alts} \ y \in \text{alts}$
(proof)

sublocale preorder-family agents alts R
(proof)

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

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

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

lemma $(\text{in } - ) \ \text{pref-profile-eqI}$:
assumes $\text{pref-profile-wf} \ \text{agents} \ \text{alts} \ R_1 \ \text{pref-profile-wf} \ \text{agents} \ \text{alts} \ R_2$
assumes $\forall x, x \in \text{agents} \implies R_1 x = R_2 x$
shows $R_1 = R_2$
(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 $\sigma$ $(R \circ \pi)$ = permute-profile $\sigma$ $R \circ \pi$
(proof)

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

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

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

(proof)

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

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

2.1 Pareto dominance

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

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

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

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

context preorder-family
begin

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

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

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

lemma Pareto-strictI’:
assumes $\forall i. \ i \in \text{dom} \Rightarrow x \preceq[R \ i \ y] \ i \in \text{dom} \ \neg x \preceq[R \ i \ y]$
shows $x \prec[Pareto(R)] \ y$
(proof)

sublocale Pareto: preorder-on carrier Pareto(R)
(proof)
lemma pareto-loser-in-alts:
  assumes $x \in \text{pareto-losers } R$
  shows $x \in \text{carrier}$
  ⟨proof⟩

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

end

2.2 Preferred alternatives

context pref-profile-wf
begin

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

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

end

2.3 Favourite alternatives

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

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

definition has-unique-favorites :: ('agent,'alt) pref-profile ⇒ bool where 
has-unique-favorites $R$ $\iff$ ($\forall i$. favorites $R \, i$ = $\{\}$ $\vee$ is-singleton (favorites $R \, i$))

context pref-profile-wf
begin

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

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

lemma favorites-altdef':
favorites $R i = \{x \in \text{alts. } \forall y \in \text{alts. } x \succeq_{[R i]} y\}$

(\text{proof})

\text{lemma} favorites-subset-alts: favorites $R i \subseteq \text{alts}$
(\text{proof})

\text{lemma} finite-favorites [simp, intro]: finite (favorites $R i$)
(\text{proof})

\text{lemma} favorites-nonempty: $i \in \text{agents} \implies \text{favorites } R i \neq \{\}$
(\text{proof})

\text{lemma} favorites-permute:
  \text{assumes } i: i \in \text{agents and perm: } \sigma \text{ permutes alts}
  \text{shows } \text{favorites (permute-profile } \sigma R i = \sigma : \text{favorites } R i$
(\text{proof})

\text{lemma} has-unique-favorites-altdef:
  has-unique-favorites $R \iff (\forall i \in \text{agents. is-singleton (favorites } R i))$
(\text{proof})

end

locale pref-profile-unique-favorites = pref-profile-wf agents alts $R$
  for agents :: 'agent set and alts :: 'alt set and $R+$
  assumes unique-favorites': has-unique-favorites $R$
begin

\text{lemma} unique-favorites: $i \in \text{agents} \implies \text{favorites } R i = \{\text{favorite } R i\}$
(\text{proof})

\text{lemma} favorite-in-alts: $i \in \text{agents} \implies \text{favorite } R i \in \text{alts}$
(\text{proof})

end

2.4 Anonymous profiles

\text{type-synonym} ('agent, 'alt) apref-profile = 'alt set list multiset

\text{definition} anonymous-profile :: ('agent, 'alt) pref-profile $\Rightarrow$ ('agent, 'alt) apref-profile

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

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

**lemma (in pref-profile-wf) anonymous-profile-def:**

\[
\text{anonymous-profile } R = \text{image-mset (weak-ranking } \circ R \text{)} (\text{mset-set agents})
\]

⟨proof⟩

**lemma (in pref-profile-wf) anonymous-profile-permute:**

**assumes** \(\sigma\) permutes alts finite agents

**shows** anonymous-profile (permute-profile \(\sigma\) R) = image-mset (map ((\') \(\sigma\))) (anonymous-profile R)

⟨proof⟩

**lemma (in pref-profile-wf) anonymous-profile-update:**

**assumes** \(i\): \(i \in\) agents and \(\text{fin [simp]: finite agents and total-preorder-on alts } Ri'\)

**shows** anonymous-profile (R(i := Ri')) = anonymous-profile R - \{\#weak-ranking (R i)#\} + \{\#weak-ranking Ri'\#\}

⟨proof⟩

2.5 Preference profiles from lists

definition prefs-from-table :: ('agent × 'alt set list) list ⇒ ('agent, 'alt) pref-profile

where

\[
prefs-from-table \text{ xss } = (\lambda i. \text{case-option (λ- -. False) of-weak-ranking (map-of xss} \ i))
\]

definition prefs-from-table-wf where

\[
prefs-from-table-wf \text{ agents alts xss } \iff \text{agents } \neq \{\} \land \text{alts } \neq \{\} \land \text{distinct (map fst xss)} \land \text{set (map fst xss) } = \text{agents} \land (\forall \text{xs } \in \text{set (map snd xss)}). \bigcup \{\text{set xs} = \text{alts} \land \text{is-finite-weak-ranking xs}\}
\]

**lemma prefs-from-table-wfI:**

**assumes** agents \(\neq\) \{\} \land alts \(\neq\) \{\} \land \text{distinct (map fst xss)}

**assumes** map fst xss = agents

**assumes** \(\forall \text{xs } \in \text{set (map snd xss)} \implies \bigcup \{\text{set xs} = \text{alts}\} \land \text{is-finite-weak-ranking xs}\)

**shows** prefs-from-table-wf agents alts xss

⟨proof⟩

**lemma prefs-from-table-wfD:**

**assumes** prefs-from-table-wf agents alts xss

**shows** agents \(\neq\) \{\} \land alts \(\neq\) \{\} \land \text{distinct (map fst xss)}

**and** map fst xss = agents

**and** \(\forall \text{xs } \in \text{set (map snd xss)} \implies \bigcup \{\text{set xs} = \text{alts}\} \land \text{is-finite-weak-ranking xs}\)

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

(\langle \text{proof} \rangle)

lemma prefs-from-table-eqI:
  \( \begin{align*}
    \text{assumes} & \quad \text{distinct (map fst xs)} \quad \text{distinct (map fst ys)} \quad \text{set xs} = \text{set ys} \\
    \text{shows} & \quad \text{prefs-from-table xs} = \text{prefs-from-table ys}
  \end{align*} \)

(\langle \text{proof} \rangle)

lemma prefs-from-table-undef:
  \( \begin{align*}
    \text{assumes} & \quad \text{prefs-from-table-wf agents alts xss i} \not\in \text{agents} \\
    \text{shows} & \quad \text{prefs-from-table xss i} = (\lambda - -. \text{False})
  \end{align*} \)

(\langle \text{proof} \rangle)

lemma prefs-from-table-map-of:
  \( \begin{align*}
    \text{assumes} & \quad \text{prefs-from-table-wf agents alts xss i} \in \text{agents} \\
    \text{shows} & \quad \text{prefs-from-table xss i} = \text{of-weak-ranking (the (map-of xss i))}
  \end{align*} \)

(\langle \text{proof} \rangle)

lemma prefs-from-table-update:
  \( \begin{align*}
    \text{fixes} & \quad x \text{ xs} \\
    \text{assumes} & \quad i \in \text{set (map fst xs)} \\
    \text{defines} & \quad \text{xs' } \equiv \text{map (\lambda (j, y). if } j = i \text{ then } (j, x) \text{ else } (j, y)) \text{ xs} \\
    \text{shows} & \quad \text{prefs-from-table xs'}(\text{is lhs } = \text{?rhs}) = \text{prefs-from-table xs'}(i) \text{ (is } \text{?lhs } = \text{?rhs})
  \end{align*} \)

(\langle \text{proof} \rangle)

lemma prefs-from-table-swap:
  \( \begin{align*}
    x \neq y & \Rightarrow \text{prefs-from-table}((x,x')\#(y,y')\#xs) = \text{prefs-from-table}((y,y')\#(x,x')\#xs)
  \end{align*} \)

(\langle \text{proof} \rangle)

lemma permute-prefs-from-table:
  \( \begin{align*}
    \text{assumes} & \quad \sigma \text{ permutes fst \ ' set xs} \\
    \text{shows} & \quad \text{prefs-from-table xs} \circ \sigma = \text{prefs-from-table (map (\lambda (x,y). (\text{inv } \sigma x, y)) xs)}
  \end{align*} \)

(\langle \text{proof} \rangle)

lemma permute-profile-from-table:
  \( \begin{align*}
    \text{assumes} & \quad \sigma \text{ permutes alts} \\
    \text{assumes} & \quad \text{wf: prefs-from-table-wf agents alts xss} \\
    \text{shows} & \quad \text{permute-profile } \sigma (\text{prefs-from-table xss}) = \text{prefs-from-table (map (\lambda (x,y). (\text{map ((\text{inv } \sigma y) } xss)) xs)} (\text{is } \text{?f } = \text{?g})
  \end{align*} \)

(\langle \text{proof} \rangle)

2.6 Automatic evaluation of preference profiles

lemma eval-prefs-from-table [simp]:
  \[ \text{prefs-from-table [i] } = (\lambda - -. \text{False}) \]
\[ \text{prefs-from-table}((i, y) \# xs) i = \text{of-weak-ranking} y \]
\[ i \neq j \Rightarrow \text{prefs-from-table}((j, y) \# xs) i = \text{prefs-from-table} xs i \]

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

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

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

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

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

**Lemma prefs-from-table-agent-permutation:**
assumes \( \text{wf} \): \( \text{prefs-from-table-wf} \) agents alts xs \( \text{prefs-from-table-wf} \) agents alts ys
assumes \( \text{mset-eq} \): \( \text{mset} \; (\text{map} \; \text{snd} \; xs) = \text{mset} \; (\text{map} \; \text{snd} \; ys) \)
obtains \( \pi \) where \( \pi \) permutes agents \( \text{prefs-from-table} xs \circ \pi = \text{prefs-from-table} ys \)

**Lemma permute-list-distinct:**
assumes \( f \cdot \{<\text{length } xs\} \subseteq \{<\text{length } xs\} \) distinct \( xs \)
shows \( \text{permute-list } f \; xs = \text{map} \; (\lambda \; x. \; xs ! f \; (\text{index } xs \; x)) \; xs \)
(proof)

lemma image-mset-eq-permutation:
assumes \( \{#f \; x. \; x \in \# \; \text{mset-set } A\#\} = \{#g \; x. \; x \in \# \; \text{mset-set } A\#\} \) finite \( A \)
obtains \( \pi \) where \( \pi \) permutes \( A \) \( \land x. \; x \in A \implies g (\pi \; x) = f \; x \)
(proof)

lemma anonymous-profile-agent-permutation:
assumes \( \text{eq} \cdot \text{anonymous-profile } R1 = \text{anonymous-profile } R2 \)
assumes \( \text{wf} \cdot \text{pref-profile-wf} \text{ agents } alts \; R1 \; \text{pref-profile-wf} \text{ agents } alts \; R2 \)
assumes \( \text{fin} \cdot \text{finite agents} \)
obtains \( \pi \) where \( \pi \) permutes \( R2 \circ \pi = R1 \)
(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 \( \neq \) \{\}
assumes nonempty-alts [simp]: alts \( \neq \) \{}
begin
abbreviation is-pref-profile \equiv \text{pref-profile-wf} \text{ agents } alts

lemma finite-total-preorder-on-iff' [simp]:
\text{finite-total-preorder-on } alts \; R \iff \text{total-preorder-on } alts \; R
(proof)

lemma pref-profile-wf' [intro?]:
(\(\forall i. \; i \in \text{agents} \implies \text{total-preorder-on } alts \; (R \; i)\)) \implies
(\(\forall i. \; i \notin \text{agents} \implies R \; i = (\lambda- \cdot \text{False})\)) \implies \text{is-pref-profile } R
(proof)

lemma is-pref-profile-update [simp,intro]:
assumes \( \text{is-pref-profile } R \; \text{total-preorder-on } alts \; Ri' \; i \in \text{agents} \)
shows \( \text{is-pref-profile } (R(i := Ri')) \)
(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 $R \neq \{}$ (proof)

lemma maximal-imp-preferred: 
\[ x \in \text{alts} \implies \text{Max-wrt } R \subseteq \text{preferred-alts } R \ x \] (proof)

end

end

end

3 Auxiliary facts about PMFs

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

The type of lotteries (a probability mass function)
type-synonym 'alt lottery = 'alt pmf

definition lotteries-on :: 'a set \Rightarrow 'a lottery set where
lotteries-on $A$ = \{p. set-pmf $p$ \subseteq $A$\}

lemma pmf-of-set-lottery:
$A \neq \{\} \implies$ finite $A \implies A \subseteq B \implies$ pmf-of-set $A \in$ lotteries-on $B$
(proof)

lemma pmf-of-list-lottery:
\[ \text{pmf-of-list-wf } xs \Rightarrow \text{set (map fst xs) } \subseteq A \implies \text{pmf-of-list } xs \in \text{lotteries-on } A \] (proof)

lemma return-pmf-in-lotteries-on [simp,intro]:
\[ x \in A \implies \text{return-pmf } x \in \text{lotteries-on } A \] (proof)

end
theory Utility-Functions
imports
  Complex-Main
  HOL-Probability.Probability
  Lotteries
  Preference-Profiles
begin

3.1 Definition of von Neumann–Morgenstern utility functions

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

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

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

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

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

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

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

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

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

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

lemma add-epsilon:
  assumes A: \( \forall x y. \ le x y \implies \ le y x \implies f x = f y \)
  shows \( \exists \varepsilon > 0. \) vnm-utility carrier le (\( \lambda x. u x + \varepsilon * f x \))
⟨proof⟩

lemma diff-epsilon:
  assumes \( \forall x y. \ le x y \implies \ le y x \implies f x = f y \)
  shows \( \exists \varepsilon > 0. \) vnm-utility carrier le (\( \lambda x. u x - \varepsilon * f x \))
⟨proof⟩

end

end

4 Stochastic Dominance

theory Stochastic-Dominance

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

begin

4.1 Definition of Stochastic Dominance

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

definition SD :: 'alt relation ⇒ 'alt lottery relation where
  \( p \succeq [SD(R)] q \longleftrightarrow p \in \text{lotteries-on } \{ x. R x x \} \land q \in \text{lotteries-on } \{ x. R x x \} \land \)
  \( \forall x. R x x \rightarrow \text{measure-pmf.prob } p \{ y. y \succeq [R] x \} \geq \text{measure-pmf.prob } q \{ y. y \succeq [R] x \} \)

lemma SD-empty [simp]: SD (\( \lambda - -. \) False) = (\( \lambda - -. \) False)
⟨proof⟩

Stochastic dominance over any relation is a preorder.

lemma SD-refl: \( p \succeq [SD(R)] p \longleftrightarrow p \in \text{lotteries-on } \{ x. R x x \} \)
⟨proof⟩

lemma SD-trans [simp, trans]: \( p \succeq [SD(R)] q \implies q \succeq [SD(R)] r \implies p \succeq [SD(R)] r \)
⟨proof⟩

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

28
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 $q \prec [SD(le)] \text{ return-pmf } x \iff (\forall y \in \text{set-pmf } q. \ y \succeq [le] x) \land (\exists y \in \text{set-pmf } q. \ (y \prec [le] x))$

(\text{proof})

\begin{enumerate}
\item \textbf{lemma SD-singleton [simp]:}
\item $x \in \text{carrier} \implies y \in \text{carrier} \implies \text{return-pmf } x \preceq [SD(le)] \text{ return-pmf } y \iff x \preceq [le] y$
\item \text{proof}
\end{enumerate}

\begin{enumerate}
\item \textbf{lemma SD-strict-singleton [simp]:}
\item $x \in \text{carrier} \implies y \in \text{carrier} \implies \text{return-pmf } x \prec [SD(le)] \text{ return-pmf } y \iff x \prec [le] y$
\item \text{proof}
\end{enumerate}

end
context \textit{pref-profile-wf}
begin

\begin{enumerate}
\item \textbf{context \textit{pref-profile-wf}}
\item \textbf{begin}
\item \textbf{context}
\item \textbf{fixes} $i$ \textbf{assumes} $i: i \in \text{agents}$
\item \textbf{begin}
\item \textbf{interpretation} $R_i: \text{preorder-on alts } R i$
\item \textbf{proof}
\end{enumerate}

\begin{enumerate}
\item \textbf{lemmas} $SD-singleton-left = R_i.\text{SD-singleton-left}$
\item \textbf{lemmas} $SD-singleton-right = R_i.\text{SD-singleton-right}$
\item \textbf{lemmas} $SD-strict-singleton-left = R_i.\text{SD-strict-singleton-left}$
\item \textbf{lemmas} $SD-strict-singleton-right = R_i.\text{SD-strict-singleton-right}$
\item \textbf{lemmas} $SD-singleton = R_i.\text{SD-singleton}$
\item \textbf{lemmas} $SD-strict-singleton = R_i.\text{SD-strict-singleton}$
\end{enumerate}

end

\begin{enumerate}
\item \textbf{lemmas} (in \textit{pref-profile-wf}) [simp] = $\text{SD-singleton } \text{SD-strict-singleton}$
\end{enumerate}

4.2 Stochastic Dominance for preference profiles

\begin{enumerate}
\item \textbf{context \textit{pref-profile-wf}}
\item \textbf{begin}
\item \textbf{lemma SD-pref-profile:}
\item \textbf{assumes} $i \in \text{agents}$
\item \textbf{shows} $p \succeq [SD(R i)] q \iff p \in \text{lotteries-on alts } R i \land q \in \text{lotteries-on alts } R i \land$
\item $(\forall x \in \text{alts. measure-pmf.p } p \text{ (preferred-alts } (R i) \ x) \geq$
\item $\text{measure-pmf.p } q \text{ (preferred-alts } (R i) \ x))$
\item \text{proof}
\end{enumerate}
lemma SD-pref-profileI [intro?]:
assumes \( i \in \text{agents} \) \( p \in \text{lotteries-on alts} \) \( q \in \text{lotteries-on alts} \)
assumes \( \forall x. x \in \text{alts} \implies \text{measure-pmf.prob} \ p \ (\text{preferred-alts} \ (R i) \ x) \geq \text{measure-pmf.prob} \ q \ (\text{preferred-alts} \ (R i) \ x) \)
shows \( p \succeq [\text{SD}(R i)] q \)
⟨proof⟩

lemma SD-pref-profileD:
assumes \( i \in \text{agents} \)
shows \( p \succeq [\text{SD}(R i)] q \)
⟨proof⟩
end

4.3 SD efficient lotteries

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

context pref-profile-wf
begin

sublocale SD: preorder-family agents lotteries-on alts SD \( R \) ⟨proof⟩

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 \( R \ p \longleftrightarrow \neg (\exists q \in \text{lotteries-on alts}. \ q \succ [\text{Pareto}(\text{SD} \circ R)] p) \)
⟨proof⟩

lemma SD-efficient-def':
SD-efficient \( R \ p \longleftrightarrow \neg (\exists q \in \text{lotteries-on alts}. (\forall i \in \text{agents}. \ q \geq [\text{SD}(R i)] p) \land (\exists i \in \text{agents}. \ q \succ [\text{SD}(R i)] p)) \)
⟨proof⟩

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

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

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

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

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

context finite-total-preorder-on
begin

abbreviation is-vnm-utility ≡ 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 \( p \in \text{lotteries-on carrier} \ q \in \text{lotteries-on carrier} \)
sshows \( p \succeq [\text{SD}(le)] q \iff 
(\forall u. \text{is-vnm-utility } u \implies \text{measure-pmf.expectation } p \ u \leq \text{measure-pmf.expectation } q \ u) \)
⟨proof⟩
lemma not-strict-SD-iff:
assumes $p \in \text{lotteries-on carrier}$ $q \in \text{lotteries-on carrier}$
shows $\neg (p \sim [\text{SD}(\leq)] q) \iff (\exists u. \text{is-vnm-utility} u \land \text{measure-pmf.expectation} q u \leq \text{measure-pmf.expectation} p u)$
⟨proof⟩

lemma strict-SD-iff:
assumes $p \in \text{lotteries-on carrier}$ $q \in \text{lotteries-on carrier}$
shows $p \sim [\text{SD}(\leq)] q \iff (\forall u. \text{is-vnm-utility} u \rightarrow \text{measure-pmf.expectation} p u < \text{measure-pmf.expectation} q u)$
⟨proof⟩
end
end

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

context pref-profile-wf
begin

lemma SD-inefficient-support-subset:
assumes inefficient: $\neg \text{SD-efficient} R p'$
assumes support: $\text{set-pmf} p' \subseteq \text{set-pmf} p$
assumes lotteries: $p \in \text{lotteries-on alts}$
shows $\neg \text{SD-efficient} R p$
⟨proof⟩

lemma SD-efficient-support-subset:
assumes $\text{SD-efficient} R p \text{ set-pmf} p' \subseteq \text{set-pmf} p$ $p \in \text{lotteries-on alts}$
shows $\text{SD-efficient} R p'$
⟨proof⟩

lemma SD-efficient-same-support:
assumes $\text{set-pmf} p = \text{set-pmf} p'$ $p \in \text{lotteries-on alts}$
shows $\text{SD-efficient} R p \iff \text{SD-efficient} R p'$
⟨proof⟩

lemma SD-efficient-iff:
assumes $p \in \text{lotteries-on alts}$
shows $\text{SD-efficient} R p \iff \text{SD-efficient} R (\text{pmf-of-set} (\text{set-pmf} p))$
⟨proof⟩
end
Lemma SD-efficient-no-pareto-loser:
assumes efficient: SD-efficient \( R \) \( p \) and \( p \)-af: \( p \in \text{lotteries-on alts} \)
shows \( \text{set-pmf} \; p \cap \text{pareto-losers} \; R = \{\} \)
⟨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.

Lemma improve-lottery-support-subset:
assumes \( p \in \text{lotteries-on alts} \) \( q \in \text{lotteries-on alts} \) \( q \succ [\text{Pareto}(SD \circ R)] \) \( p \)

obtains \( r \) where \( r \in \text{lotteries-on alts} \) \( r \succeq [\text{Pareto}(SD \circ R)] \) \( q \) \( \text{set-pmf} \; r \subset \text{set-pmf} \; p \)
⟨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.

Context
fixes \( p :: \text{'}alt lottery \) 
assumes lott: \( p \in \text{lotteries-on alts} \)
begin

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

private lemma improve-lottery:
assumes \( \neg \text{SD-efficient} \; R \) \( q \)
defines \( r \equiv \text{improve-lottery} \) \( q \)
shows \( r \in \text{lotteries-on alts} \) \( r \succ [\text{Pareto}(SD \circ R)] \) \( q \)
\( \wedge r'. \; r' \in \text{lotteries-on alts} \Rightarrow r' \succ [\text{Pareto}(SD \circ R)] \) \( q \Rightarrow \neg (\text{set-pmf} \; r' \subset \text{set-pmf} \; r) \)
⟨proof⟩

fun sd-chain :: nat \Rightarrow \text{'}alt lottery option where
sd-chain 0 = Some \( p \) 
| sd-chain \( (\text{Suc} \; n) \) =
  (case sd-chain \( n \) of
    None \Rightarrow None 
  | Some \( p \) \Rightarrow if SD-efficient \( R \) \( p \) then None else Some (improve-lottery \( p \))

private lemma sd-chain-None-propagate:
\( m \geq n \Rightarrow \text{sd-chain} \; n = \text{None} \Rightarrow \text{sd-chain} \; m = \text{None} \)
⟨proof⟩

Lemma sd-chain-Some-propagate:
\[ m \geq n \implies \text{sd-chain } m = \text{Some } q \implies \exists q'. \text{sd-chain } n = \text{Some } q' \]

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

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

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

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

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

(\text{proof})

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

(\text{proof})

\textbf{end}

\textbf{lemma}
\[ \text{assumes } p \in \text{lotteries-on alts} \]
\[ \text{shows } \exists q \in \text{lotteries-on alts}. q \succeq [\text{Pareto}(\text{SD} \circ R)] p \land \text{SD-efficient } R q \]

(\text{proof})

\textbf{end}

\textbf{end}

5 Social Decision Schemes

theory Social-Decision-Schemes

imports
\text{Complex-Main}
\text{HOL-Probability,Probability}
\text{Preference-Profiles}
\text{Elections}
\text{Order-Predicates}
\text{Stochastic-Dominance}
\text{SD-Efficiency}

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 $\text{prefs-from-table-wf agents alts xs prefs-from-table-wf agents alts ys}$
assumes $\text{mset (map snd xs) = mset (map snd ys)}$
shows $\text{sds (prefs-from-table xs) = sds (prefs-from-table ys)}$

(proof)

context
begin
qualified lemma anonymity-prefs-from-table-aux:
assumes $R_1 = \text{prefs-from-table xs prefs-from-table-wf agents alts xs}$
assumes $R_2 = \text{prefs-from-table ys prefs-from-table-wf agents alts ys}$
assumes $\text{mset (map snd xs) = mset (map snd ys)}$
shows $\text{sds R}_1 = \text{sds R}_2$ (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: $\sigma$ permutes alts $\Rightarrow$ is-pref-profile $R$ $\Rightarrow$
$sds (\text{permute-profile } \sigma R) = \text{map-pmf } \sigma (sds R)$

begin
Alternative formulation of neutrality that shows that our definition is equivalent to that in the paper.
lemma neutral':
assumes $\sigma$ permutes alts
assumes is-pref-profile $R$
assumes $a \in$ alts
shows $\text{pmf (sds (permute-profile } \sigma R)) (\sigma a) = \text{pmf (sds R) a}$
(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: $\sigma$ permutes alts and $\text{wf: is-pref-profile } R_1 \text{ is-pref-profile } R_2$
assumes eq: anonymous-profile $R_1 =$
image-$\text{mset (map ((\sigma) \sigma)) (anonymous-profile } R_2)$
shows $\text{sds R}_1 = \text{map-pmf } \sigma (\text{sds R}_2)$

37
lemma sds-anonymous-neutral':
assumes perm: σ permutes alts and wf: is-pref-profile R1 is-pref-profile R2
assumes eq: anonymous-profile R1 =
  image-mset (map (λ σ)) (anonymous-profile R2)
shows pmf (sds R1) (σ x) = pmf (sds R2) x
⟨proof⟩

lemma sds-automorphism:
assumes perm: σ permutes alts and wf: is-pref-profile R
assumes eq: image-mset (map (λ σ)) (anonymous-profile R) = anonymous-profile R
shows map-pmf σ (sds R) = sds R
⟨proof⟩
end

lemma an-sds-automorphism-aux:
assumes wf: prefs-from-table-wf agents alts yss R ≡ prefs-from-table yss
assumes an: an-sds agents alts sds
assumes eq: mset (map (map (map (λ x) (permutation-of-list xs)) ◦ snd) yss) = mset (map snd yss)
assumes perm: set (map fst xs) ⊆ alts set (map snd xs) = set (map fst xs)
distinct (map fst xs)
and x: x ∈ alts y = permutation-of-list xs x
shows pmf (sds R) x = pmf (sds R) y
⟨proof⟩

5.5 Ex-post efficiency

locale ex-post-efficient-sds =
  social-decision-scheme agents alts sds
for agents :: 'agent set and alts :: 'alt set and sds +
assumes ex-post-efficient:
  is-pref-profile R ⇒ set-pmf (sds R) ∩ pareto-losers R = {}
begin

lemma ex-post-efficient':
assumes is-pref-profile R y >[Pareto(R)] x
shows pmf (sds R) x = 0
⟨proof⟩

lemma ex-post-efficient'':
assumes is-pref-profile R i ∈ agents ∀ i∈agents. y ≥[R i] x ¬y ≤[R i] x
shows pmf (sds R) x = 0
⟨proof⟩
end

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.

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 \( \bigwedge_{i \in \text{agents}} q \succeq [\text{SD}(R \ i)] \text{ sds } i \in \text{agents} \ \ q > [\text{SD}(R \ i)] \text{ sds } R \)

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)

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

shows ∃x∈A. pmf (sds R) x = 0
⟨proof⟩

end

5.7 Weak strategyproofness

context social-decision-scheme

begin

The SDS is said to be manipulable for a particular preference profile, a particular agent, and a particular alternative preference ordering for that

39
agent if the lottery obtained if the agent submits the alternative preferences strictly SD-dominates that obtained if the original preferences are submit-
ted. (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

**end**

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

**locale** strategyproof-sds = social-decision-scheme agents alts sds
for agents :: 'agent set and alts :: 'alt set and sds +
assumes strategyproof:
  is-pref-profile R ⇒ i ∈ agents ⇒ total-preorder-on alts Ri' ⇒
  ¬manipulable-profile R i Ri'

5.8 Strong strategyproofness

**context** social-decision-scheme
begin

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

In other words: the SDS is strategyproof w.r.t the preference profile
R and the agent i and the alternative preference relation R'i if the lottery for ob-
tained for R is at least as good for i as the lottery obtained when i misrep-
resents her preferences as R'.

**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 := R'))

**lemma** strongly-strategyproof-profileI [intro]:
assumes is-pref-profile R total-preorder-on alts Ri' i ∈ agents
assumes \( \land x. x \in \text{alts} \implies \text{lottery-prob} \ (\text{sds} \ (R(i := R'))) \ (\text{preferred-alts} \ (R i) \ x) \)
≤ lottery-prob (sds R) (preferred-alts (R i) x)
shows strongly-strategyproof-profile R i Ri' (proof)

**lemma** strongly-strategyproof-imp-not-manipulable:
assumes strongly-strategyproof-profile R i Ri'
sshows ¬manipulable-profile R i Ri'
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))))

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'
  (proof)

lemma lift-pref-profile-permute-agents:
  assumes π permutes agents agents'
shows \( \text{lift-pref-profile agents alts agents’ alts’ } (R \circ \pi) = \text{lift-pref-profile agents alts agents’ alts’ } R \circ \pi \) 

\langle proof \rangle

\textbf{lemma lift-pref-profile-permute-alts:}
\begin{itemize}
  \item \textbf{assumes} \( \pi \) permutes \( \text{alts alts’} \subseteq \text{alts’} \)
  \item \textbf{shows} \( \text{lift-pref-profile agents alts agents’ alts’ } (\text{permute-profile } \pi R) = \text{permute-profile } \pi (\text{lift-pref-profile agents alts agents’ alts’ } R) \)
\end{itemize}
\langle proof \rangle

\textbf{lemma lotteries-on-subset:} \( \text{A} \subseteq \text{B} \implies p \in \text{lotteries-on } \text{A} \implies p \in \text{lotteries-on } \text{B} \) 
\langle proof \rangle

\textbf{lemma lottery-prob-carrier:} \( p \in \text{lotteries-on } \text{A} \implies \text{measure-\text{pmf}.prob } p \text{ } \text{A} = 1 \) 
\langle proof \rangle

\textbf{context}
\begin{itemize}
  \item \textbf{fixes} \( \text{agents alts } R \text{ agents’ alts’ } R’ \)
  \item \textbf{assumes} \( R\text{-uf: } \text{pref-profile-uf agents alts } R \)
  \item \textbf{assumes} \( \text{election: agents } \subseteq \text{agents’ alts } \subseteq \text{alts’ alts } \neq \{} \text{ agents } \neq \{} \text{ finite alts’ } \)
  \item \textbf{defines} \( R’ \equiv \text{lift-pref-profile agents alts agents’ alts’ } R \)
\end{itemize}

\textbf{begin}

\textbf{interpretation} \( R: \text{pref-profile-wf agents alts } R \) 
\langle proof \rangle

\textbf{interpretation} \( R’: \text{pref-profile-wf agents’ alts’ } R’ \) 
\langle proof \rangle

\textbf{lemma lift-pref-profile-strict-iff:}
\begin{itemize}
  \item \( x \prec [\text{lift-pref-profile agents alts agents’ alts’ } R \ i ] y \iff i \in \text{agents } \land ((y \in \text{alts } \land x \in \text{alts’ } - \text{alts}) \lor x \prec [R \ i ] y) \)
\end{itemize}
\langle proof \rangle

\textbf{lemma preferred-alts-lift-pref-profile:}
\begin{itemize}
  \item \textbf{assumes} \( i: i \in \text{agents’} \text{ and } x: x \in \text{alts’} \)
  \item \textbf{shows} \( \text{preferred-alts } (R’ \ i ) x = (i \in \text{agents } \land x \in \text{alts then preferred-alts } (R \ i ) x \text{ else alts’}) \)
\end{itemize}
\langle proof \rangle

\textbf{lemma lift-pref-profile-Pareto-iff:}
\begin{itemize}
  \item \( x \succeq [\text{Pareto}(R’)] y \iff x \in \text{alts’ } \land y \in \text{alts’ } \land (x \notin \text{alts } \lor x \succeq [\text{Pareto}(R)] y) \)
\end{itemize}
\langle proof \rangle

\textbf{lemma lift-pref-profile-Pareto-strict-iff:}
\begin{itemize}
  \item \( x \prec [\text{Pareto}(R’)] y \iff x \in \text{alts’ } \land y \in \text{alts’ } \land (x \notin \text{alts } \land y \in \text{alts } \lor x \prec [\text{Pareto}(R)] y) \)
\end{itemize}
\langle proof \rangle

\textbf{lemma pareto-losers-lift-pref-profile:}
shows pareto-losers \( R' = \text{pareto-losers } R \cup (\text{alts}' \setminus \text{alts}) \)
(proof)

context

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 \( i : i \in \text{agents}' \setminus \text{agents} \)
  shows \( p \preceq \SD(R' i) q \)
(proof)

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

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

end

lemma lift-SD-strict-iff:
  assumes \( p \in \text{lotteries-on alts} \) \( q \in \text{lotteries-on alts} \) and \( i : i \in \text{agents} \)
  shows \( p \prec \SD(R' i) q \iff p \prec \SD(R i) q \)
(proof)

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

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

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

and $\text{agents}'$-nonempty [simp]: $\text{agents}' \neq \{\}$ and $\text{alts}'$-nonempty [simp]: $\text{alts}' \neq \{\}$

begin

lemma finite-agents' [simp]: finite $\text{agents}'$
\begin{proof}
\end{proof}

lemma finite-alts' [simp]: finite $\text{alts}'$
\begin{proof}
\end{proof}

abbreviation $\text{lift} :: \text{agents}' \times \text{alts}' \Rightarrow \text{agents} \times \text{alts}$

where $\text{lift} \equiv \text{lift-pref-profile} \text{agents}' \text{alts}'$

definition lowered :: $\text{agents}' \times \text{alts}' \Rightarrow \text{alts}$

where $\text{lowered} = \text{sds} \circ \text{lift}$

lemma lift-wf [simp, intro]:

\begin{proof}
\end{proof}

sublocale lowered: election $\text{agents}' \times \text{alts}'$
\begin{proof}
\end{proof}

lemma preferred-alts-lift:

\begin{proof}
\end{proof}

lemma pareto-losers-lift:

\begin{proof}
\end{proof}

lemma lowered-lotteries: lowered $\text{lotteries} \subseteq \text{lotteries}$
\begin{proof}
\end{proof}

sublocale lowered: social-decision-scheme $\text{agents}' \times \text{alts}'$
\begin{proof}
\end{proof}

sublocale ex-post-efficient-sds $\text{agents}' \times \text{alts}'$
\begin{proof}
\end{proof}

lemma lowered-in-lotteries [simp]: lowered is-pref-profile $\text{agents}'$ $\text{alts}'$
\begin{proof}
\end{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
(\text{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
(\text{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
(\text{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
(\text{proof})
end
locale sds-lowering-anonymous-neutral-sdeff-stratproof =
  sds-lowering-anonymous + sds-lowering-neutral +
  sds-lowering-sd-efficient + sds-lowering-strategyproof
end

7 Random Dictatorship

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

We define Random Dictatorship as a social decision scheme on total 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:
random-dictatorship agents alts R =
  do {i ← pmf-of-set agents;
       pmf-of-set (Max-wrt-among (R i) alts)}
context election
begin

abbreviation RD :: ('agent, 'alt) pref-profile ⇒ 'alt lottery where
RD ≡ random-dictatorship agents alts

lemma random-dictatorship-def:
  assumes is-pref-profile R
  shows RD R =
    do {i ← pmf-of-set agents;
         pmf-of-set (favorites R i)}
  ⟨proof⟩

lemma random-dictatorship-unique-favorites:
  assumes is-pref-profile R has-unique-favorites R
  shows RD R = map-pmf (favorite R) (pmf-of-set agents)
  ⟨proof⟩
**Lemma** random-dictatorship-unique-favorites':

** Assumes** is-pref-profile R has-unique-favorites R

** Shows**  \( RD R = \text{pmf-of-multiset} (\text{image-mset} (\text{favorite} R) (\text{mset-set} \text{ agents})) \)

\begin{proof}
\end{proof}

**Lemma** pmf-random-dictatorship:

** Assumes** is-pref-profile R

** Shows**  \( \text{pmf} (RD R) x = \left( \sum_{i \in \text{agents}} \text{indicator} (\text{favorites} R i) x / \text{real} (\text{card} (\text{favorites} R i))) / \text{real} (\text{card} \text{ agents}) \right) \)

\begin{proof}
\end{proof}

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

\begin{proof}
\end{proof}

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

### 7.1 Anonymity

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

**Sublocale** RD: anonymous-sds agents alts RD

\begin{proof}
\end{proof}

### 7.2 Neutrality

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

**Sublocale** RD: neutral-sds agents alts RD

\begin{proof}
\end{proof}
7.3 Strong strategyproofness

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

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

end

end

8 Random Serial Dictatorship

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

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

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

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

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

lemma random-serial-dictatorship-empty [simp]:  
  random-serial-dictatorship {} alts R = pmf-of-set alts  
⟨proof⟩
**lemma** random-serial-dictatorship-nonempty:
finite agents \(\Rightarrow\) agents \(\neq\) {} \(\Rightarrow\)
random-serial-dictatorship agents alts R =
do {
i \leftarrow \text{pmf-of-set agents};
random-serial-dictatorship (agents − {i}) (Max-wrt-among (R i) 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 foldr does not require generalisation over the set of alternatives and is therefore much easier than over foldl.

**definition** rsd-winners where
rsd-winners R alts agents =
foldr (λi alts. Max-wrt-among (R i) 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 o 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 (permutations-of-set agents)};
\text{pmf-of-set (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 {

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**

\[
\text{RSD-pareto-eqclass agents alts R A} \leftarrow \text{if } A \neq \emptyset \text{ and } A \subseteq \text{alts} \text{ and } (\forall x \in A. \forall y \in \text{alts}. y \in A \leftrightarrow (\forall i \in \text{agents}. R i x y))
\]

**lemma RSD-pareto-eqclassI:**

**assumes** \(A \neq \emptyset\) \(A \subseteq \text{alts}\) \((\forall x \in A. \forall y \in \text{alts}. y \in A \leftrightarrow (\forall i \in \text{agents}. R i x y))\)

**shows** \(\text{RSD-pareto-eqclass agents alts R A}\)

(\text{proof})

**lemma RSD-pareto-eqclassD:**

**assumes** \(\text{RSD-pareto-eqclass agents alts R A}\)

**shows** \(A \neq \emptyset\) \(A \subseteq \text{alts}\) \((\forall x \in A. \forall y \in \text{alts}. y \in A \leftrightarrow (\forall i \in \text{agents}. R i x y))\)

(\text{proof})

**lemma RSD-pareto-eqclass-indiff-set:**

**assumes** \(\text{RSD-pareto-eqclass agents alts R A}\)

**shows** \(R i x y\)

(\text{proof})

**lemma RSD-pareto-eqclass-empty [simp, intro!]:**

\(\text{alts}\neq\emptyset\) \(\implies\) \(\text{RSD-pareto-eqclass}\) \(\emptyset\) \(\text{alts}\) \(R\) \(\text{alts}\)

(\text{proof})

**lemma (in pref-profile-wf) RSD-pareto-eqclass-insert:**

**assumes** \(\text{RSD-pareto-eqclass agents}\) \(\text{alts R finite alts}\)

\(i \in \text{agents}\) \(\text{agents}' \subseteq \text{agents}\)

**shows** \(\text{RSD-pareto-eqclass} (\text{insert i agents}') \text{ alts R} (\text{Max-wrt-among} (R i) A)\)

(\text{proof})
8.1.2 Facts about RSD winners

context pref-profile-wf
begin

Any RSD winner is a valid alternative.

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

There is always at least one RSD winner.

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

Obviously, the set of RSD winners is always finite.

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

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

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

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

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

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

context
begin

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

private definition indiff-set where
  indiff-set S A ≜ A ≠ {} ∧ (∀ x ∈ A. ∀ y ∈ A. S x y)
private lemma indiff-set-mono: indiff S A ⇒ B ⊆ A ⇒ B ≠ {} ⇒ 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
RSD-set-rel S A B ⇐⇒ indiff-set S B ∧ (∀ x∈A. ∀ y∈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 finite carrier A ⊆ carrier A ≠ {} total-preorder-on carrier S
shows indiff-set S (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 := Ri'). 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 wf: total-preorder-on alts Ri'
   and i: i ∈ agents and set agents' ⊆ agents finite agents
   and finite: finite alts and alts ≠ {}
defines [simp]: w' ≡ rsd-winners (R(i := Ri')) alts and [simp]: w ≡ rsd-winners R alts
shows w' agents' = w agents' ∨ RSD-set-rel (R i) (w' agents') (w 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 wf: total-preorder-on alts Ri'
   and i: i ∈ agents and set agents' = agents finite agents
   and finite: finite alts and alts ≠ {}
defines [simp]: w' ≡ rsd-winners (R(i := Ri')) alts and [simp]: w ≡ rsd-winners R alts
shows ∀ x∈w' agents'. ∀ y∈w agents'. x ≤[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)

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.
8.2.4 Neutrality

Neutrality follows from the fact that the RSD winners of a permuted profile are simply the image of the original RSD winners under the permutation.

8.2.5 Ex-post efficiency

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

8.2.6 Strong strategy-proofness

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

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

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 sf: 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 sf: 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 prefs-from-table-wf agents alts xs
shows favorites-prefs-from-table xs i = favorites (prefs-from-table xs) i
(proof)

function weak-ranking-prefs-from-table where
i ≠ j \implies weak-ranking-prefs-from-table ((i,x)#xs) j = weak-ranking-prefs-from-table xs j
| i = j \implies weak-ranking-prefs-from-table ((i,x)#xs) j = x
| weak-ranking-prefs-from-table [] j = []
(proof)

termination (proof)

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

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

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

⟨ML⟩
end
theory QSOpt-Exact
imports Complex-Main
begin
10 Automatic Fact Gathering for Social Decision Schemes

theory SDS-Automation
imports
  Preference-Profile-Cmd
  QSOpt-Exact

../Social-Decision-Schemes

keywords
derive-orbit-equations
derive-support-conditions
derive-ex-post-conditions
find-inefficient-supports
prove-inefficient-supports
derive-strategyproofness-conditions :: thy-goal

begin

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

derive-orbit-equations to derive equalities arising from automorphisms of the given profiles due to Anonymity and Neutrality

derive-ex-post-conditions to find all Pareto losers and the given profiles and derive the facts that they must be assigned probability 0 by any ex-post-efficient SDS

find-inefficient-supports to use Linear Programming to find all minimal SD-inefficient (but not ex-post-inefficient) supports in the given profiles and output a corresponding witness lottery for each of them

prove-inefficient-supports to prove a specified set of support conditions arising from ex-post- or SD-Efficiency. For conditions arising from SD-Efficiency, a witness lottery must be specified (e.g. as computed by derive-orbit-equations).

derive-support-conditions to automatically find and prove all support conditions arising from ex-post- and SD-Efficiency
derive-strategyproofness-conditions to automatically derive all conditions arising from weak Strategy-Proofness and any manipulations between the given preference profiles. An optional maximum manipulation size can be specified.

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

lemma disj-False-right: \( P \lor False \longleftrightarrow P \) (proof)

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

lemma less-or-eq-real: \[(x::real) < y \lor x = y \longleftrightarrow x \leq y \lor y = x \longleftrightarrow x \leq y \) (proof)\\

lemma multiset-Diff-single-normalize: \[\langle proof \rangle\]

lemma ex-post-efficient-aux: \[\langle proof \rangle\]

lemma SD-inefficient-support-aux: \[\langle proof \rangle\]

definition pref-classes where \[\langle proof \rangle\]

58
primrec pref-classes-lists where
\[ \text{pref-classes-lists} \, [] = \{\} \]
\[ | \text{pref-classes-lists} \, (xs \# xss) = \text{insert} \left( \bigcup (\text{set} \, (xs \# xss)) \right) \, (\text{pref-classes-lists} \, xss) \]

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

lemma pref-classes-lists-append:
\[ \text{pref-classes-lists} \, (xs @ ys) = \left( \bigcup (\text{set} \, ys) \right) \cdot \text{pref-classes-lists} \, xs \cup \text{pref-classes-lists} \, ys \]
(\text{proof})

lemma pref-classes-lists-aux:
\[ \text{assumes} \, \text{is-weak-ranking} \, xss \, \text{acc} \cap \left( \bigcup (\text{set} \, xss) \right) = \{\} \]
\[ \text{shows} \, \text{pref-classes-lists-aux} \, \text{acc} \, xss = \]
\[ \left( \text{insert} \, \text{acc} \left( (\lambda A. \, A \cup \text{acc}) \cdot \text{pref-classes-lists} \, (\text{rev} \, xss) \right) \right) \setminus \{\text{acc} \cup \left( \bigcup (\text{set} \, xss) \right)\} \]
(\text{proof})

lemma pref-classes-list-aux-hd-tl:
\[ \text{assumes} \, \text{is-weak-ranking} \, xss \, xss \neq [] \]
\[ \text{shows} \, \text{pref-classes-lists-aux} \, (\text{hd} \, xss) \, (\text{tl} \, xss) = \text{pref-classes-lists} \, (\text{rev} \, xss) \setminus \{\bigcup (\text{set} \, xss)\} \]
(\text{proof})

lemma pref-classes-of-weak-ranking-aux:
\[ \text{assumes} \, \text{is-weak-ranking} \, xss \]
\[ \text{shows} \, \text{of-weak-ranking-Collect-ge} \, xss \cdot \left( \bigcup (\text{set} \, xss) \right) = \text{pref-classes-lists} \, xss \]
(\text{proof})

lemma eval-pref-classes-of-weak-ranking:
\[ \text{assumes} \, \bigcup (\text{set} \, xss) = \text{alts} \, \text{is-weak-ranking} \, xss \, \text{alts} \neq {} \]
\[ \text{shows} \, \text{pref-classes} \, \text{alts} \, \text{(of-weak-ranking} \, xss) = \text{pref-classes-lists-aux} \, (\text{hd} \, xss) \]
(\text{proof})

context preorder-on
begin

lemma SD-iff-pref-classes:
\[ \text{assumes} \, p \in \text{lotteries-on} \, \text{carrier} \, q \in \text{lotteries-on} \, \text{carrier} \]
\[ \text{shows} \, p \preceq_{[SD,le]} q \iff \]
\[ (\forall A \in \text{pref-classes} \, \text{carrier} \, \text{le} \cdot \text{measure-pmf.prob} \, p \, A \leq \text{measure-pmf.prob} \, q \, A) \]
(\text{proof})

59
lemma (in strategyproof-an-sds) strategyproof'\textsuperscript{1}:

assumes \( \text{wf}: \text{is-pref-profile } R \text{ total-preorder-on } \text{alts } R' \text{ and } i : i \in \text{agents} \)

shows \[ (\exists A \in \text{pref-classes } \text{alts } (R i) : \text{lottery-prob } (sds (R(i := R'))) A < \text{lottery-prob } (sds R) A) \lor \]
\[ (\forall A \in \text{pref-classes } \text{alts } (R i) : \text{lottery-prob } (sds (R(i := R'))) A = \text{lottery-prob } (sds (R) A)) \]

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

lemma pref-classes-lists-aux-finite:

\[ A \in \text{pref-classes-lists-aux } \text{acc } xss \longrightarrow \text{finite } \text{acc} \longrightarrow (\forall A. A \in \text{set } xss \longrightarrow \text{finite } A) \]

\[ \longrightarrow \text{finite } A \]

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

lemma strategyproof-aux:\n
assumes \( \text{wf}: \text{prefs-from-table-wf } \text{agents } \text{alts } xss1 \text{R1} = \text{prefs-from-table } xss1 \)

\[ \text{prefs-from-table-wf } \text{agents } \text{alts } xss2 \text{R2} = \text{prefs-from-table } xss2 \]

assumes \( \text{sds}: \text{strategyproof-an-sds } \text{agents } \text{alts } \text{sds} \text{ and } i : i \in \text{agents} \text{ and } j : j \in \text{agents} \)

assumes \( \text{eq}: \text{R1} (i := R2 j) = R2 \text{ the } (\text{map-of } xss1 i) = xs \)

\[ \text{pref-classes-lists-aux } (\text{hd } xss) (\text{tl } xss) = \text{ps} \]

shows \[ (\exists A \in \text{ps}. (\sum x \in A. \text{pmf } (sds R2) x) < (\sum x \in A. \text{pmf } (sds R1) x)) \lor \]
\[ (\forall A \in \text{ps}. (\sum x \in A. \text{pmf } (sds R2) x) = (\sum x \in A. \text{pmf } (sds R1) x)) \]

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

lemma strategyproof-aux'\textsuperscript{1}:

assumes \( \text{wf}: \text{prefs-from-table-wf } \text{agents } \text{alts } xss1 \text{R1} = \text{prefs-from-table } xss1 \)

\[ \text{prefs-from-table-wf } \text{agents } \text{alts } xss2 \text{R2} = \text{prefs-from-table } xss2 \]

assumes \( \text{sds}: \text{strategyproof-an-sds } \text{agents } \text{alts } \text{sds} \text{ and } i : i \in \text{agents} \text{ and } j : j \in \text{agents} \)

assumes \( \text{perm}: \text{list-permutes } ys \text{ alts} \)

defines \( \sigma \equiv \text{permutation-of-list } ys \text{ and } \sigma' \equiv \text{inverse-permutation-of-list } ys \)

defines \( xs \equiv \text{the } (\text{map-of } xss1 i) \)

defines \( xs': xss' \equiv \text{map } (\text{the } (\text{map-of } xss2 j)) \)

defines \( R'i \equiv \text{of-weak-ranking } xs' \)

assumes \( \text{distinct-ps}: \forall A \in \text{ps}. \text{distinct } A \)

assumes \( \text{eq}: \text{mset } (\text{map} \text{ snd } xss1) \setminus \{#\text{the } (\text{map-of } xss1 i)\#\} + \{#xs'\#\} = \text{mset } (\text{map} \text{ (the } (\text{map-of } xss2 j) \circ \text{snd} ) xss2) \)

\[ \text{pref-classes-lists-aux } (\text{hd } xss) (\text{tl } xss) = \text{set } ' ps \]

shows \[ \text{list-permutes } ys \text{ alts } \land \]
\[ (\exists A \in \text{ps}. (\sum x \leftarrow A. \text{pmf } (sds R2) (\sigma' x)) < (\sum x \leftarrow A. \text{pmf } (sds R1) x)) \]
\[ \lor \]
\[ (\forall A \in \text{ps}. (\sum x \leftarrow A. \text{pmf } (sds R2) (\sigma' x)) = (\sum x \leftarrow A. \text{pmf } (sds R1) x)) \]
\[ \text{(is } - \land \text{?th}) \]

\[ \langle \text{proof} \rangle \]
References