Randomised Social Choice

Manuel Eberl

March 19, 2025

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

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

1.1 Basic Operations on Relations

The type of binary relations

type-synonym $'a\ relation = 'a \Rightarrow 'a \Rightarrow bool$

definition $map\text{-}relation :: ('a \Rightarrow 'b) \Rightarrow 'b\ relation \Rightarrow 'a\ relation$ **where**
$map\text{-}relation\ f\ R = (\lambda x\ y.\ R\ (f\ x)\ (f\ y))$

definition $restrict\text{-}relation :: 'a\ set \Rightarrow 'a\ relation \Rightarrow 'a\ relation$ **where**
$restrict\text{-}relation\ A\ R = (\lambda x\ y.\ x \in A \wedge y \in A \wedge R\ x\ y)$

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

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

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

1.2 Preorders

Preorders are reflexive and transitive binary relations.

locale *preorder-on* =
fixes $carrier :: 'a\ set$
fixes $le :: 'a\ relation$
assumes *not-outside*: $le\ x\ y \Longrightarrow x \in carrier\ le\ x\ y \Longrightarrow y \in carrier$
assumes *refl*: $x \in carrier \Longrightarrow le\ x\ x$
assumes *trans*: $le\ x\ y \Longrightarrow le\ y\ z \Longrightarrow le\ x\ z$
begin

lemma *carrier-eq*: $carrier = \{x.\ le\ x\ x\}$
$\langle proof \rangle$

lemma *preorder-on-map*:
$preorder\text{-}on\ (f\ -`\ carrier)\ (map\text{-}relation\ f\ le)$
$\langle proof \rangle$

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

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

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

end

1.3 Total preorders

Total preorders are preorders where any two elements are comparable.

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

lemma *total′*: ¬*le x y* ⟹ *x* ∈ *carrier* ⟹ *y* ∈ *carrier* ⟹ *le y x*
⟨*proof*⟩

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

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

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

end

Some fancy notation for order relations

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

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

definition *indifferent* (‹- ∼[-] -› [*51*,*10*,*51*] *60*) **where**
a ∼[*R*] *b* ≡ (*a* ⪯[*R*] *b*) ∧ (*b* ⪯[*R*] *a*)

abbreviation (*input*) *weakly-not-preferred* (‹- $\succeq$[-] -› [*51*,*10*,*51*] *60*) **where**
$a \succeq[R]\ b \equiv b \preceq[R]\ a$
term $a \succeq[R]\ b \longleftrightarrow b \preceq[R]\ a$

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*: $a \prec[le]\ b \Longrightarrow b \prec[le]\ c \Longrightarrow a \prec[le]\ c$
⟨*proof*⟩

lemma *weak-strict-trans*: $a \preceq[le]\ b \Longrightarrow b \prec[le]\ c \Longrightarrow a \prec[le]\ c$
⟨*proof*⟩

lemma *strict-weak-trans*: $a \prec[le]\ b \Longrightarrow b \preceq[le]\ c \Longrightarrow a \prec[le]\ c$
⟨*proof*⟩

end

lemma (**in** *total-preorder-on*) *not-weakly-preferred-iff*:
$a \in carrier \Longrightarrow b \in carrier \Longrightarrow \neg a \preceq[le]\ b \longleftrightarrow b \prec[le]\ a$
⟨*proof*⟩

lemma (**in** *total-preorder-on*) *not-strongly-preferred-iff*:
$a \in carrier \Longrightarrow b \in carrier \Longrightarrow \neg a \prec[le]\ b \longleftrightarrow b \preceq[le]\ a$
⟨*proof*⟩

1.4 Orders

locale *order-on* = *preorder-on* +
assumes *antisymmetric*: $le\ x\ y \Longrightarrow le\ y\ x \Longrightarrow x = y$

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

1.5 Maximal elements

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

definition *Max-wrt-among* :: *'a relation* ⇒ *'a set* ⇒ *'a set* **where**
$\textit{Max-wrt-among}\ R\ A = \{x{\in}A.\ R\ x\ x \wedge (\forall y{\in}A.\ R\ x\ y \longrightarrow R\ y\ x)\}$

lemma *Max-wrt-among-cong*:
assumes $\textit{restrict-relation}\ A\ R = \textit{restrict-relation}\ A\ R'$
shows $\textit{Max-wrt-among}\ R\ A = \textit{Max-wrt-among}\ R'\ A$
⟨*proof*⟩

definition *Max-wrt* :: $'a$ *relation* $\Rightarrow$ $'a$ *set* **where**
Max-wrt R = Max-wrt-among R UNIV

lemma *Max-wrt-altdef*: *Max-wrt R* = $\{x.\ R\ x\ x \wedge (\forall y.\ R\ x\ y \longrightarrow R\ y\ x)\}$
⟨*proof*⟩

context *preorder-on*
begin

lemma *Max-wrt-among-preorder*:
Max-wrt-among le A = $\{x{\in}carrier \cap A.\ \forall y{\in}carrier \cap A.\ le\ x\ y \longrightarrow le\ y\ x\}$
⟨*proof*⟩

lemma *Max-wrt-preorder*:
Max-wrt le = $\{x{\in}carrier.\ \forall y{\in}carrier.\ le\ x\ y \longrightarrow le\ y\ x\}$
⟨*proof*⟩

lemma *Max-wrt-among-subset*:
Max-wrt-among le A $\subseteq$ *carrier Max-wrt-among le A* $\subseteq$ *A*
⟨*proof*⟩

lemma *Max-wrt-subset*:
Max-wrt le $\subseteq$ *carrier*
⟨*proof*⟩

lemma *Max-wrt-among-nonempty*:
assumes *B* $\cap$ *carrier* $\neq \{\}$ *finite* (*B* $\cap$ *carrier*)
shows *Max-wrt-among le B* $\neq \{\}$
⟨*proof*⟩

lemma *Max-wrt-nonempty*:
carrier $\neq \{\} \Longrightarrow$ *finite carrier* $\Longrightarrow$ *Max-wrt le* $\neq \{\}$
⟨*proof*⟩

lemma *Max-wrt-among-map-relation-vimage*:
f −‘ *Max-wrt-among le A* $\subseteq$ *Max-wrt-among* (*map-relation f le*) (*f* −‘ *A*)
⟨*proof*⟩

lemma *Max-wrt-map-relation-vimage*:
f −‘ *Max-wrt le* $\subseteq$ *Max-wrt* (*map-relation f le*)
⟨*proof*⟩

lemma *image-subset-vimage-the-inv-into*:
assumes *inj-on f A B* $\subseteq$ *A*
shows *f* ‘ *B* $\subseteq$ *the-inv-into A f* −‘ *B*
⟨*proof*⟩

lemma *Max-wrt-among-map-relation-bij-subset*:

assumes *bij* $(f :: 'a \Rightarrow 'b)$
shows f ‘ *Max-wrt-among le A* $\subseteq$
Max-wrt-among (*map-relation* (*inv f*) *le*) (f ‘ A)
⟨*proof*⟩

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

lemma *Max-wrt-map-relation-bij*:
bij f $\Longrightarrow$ f ‘ *Max-wrt le* = *Max-wrt* (*map-relation* (*inv f*) *le*)
⟨*proof*⟩

lemma *Max-wrt-among-mono*:
le x y $\Longrightarrow$ $x \in$ *Max-wrt-among le A* $\Longrightarrow$ $y \in A$ $\Longrightarrow$ $y \in$ *Max-wrt-among le A*
⟨*proof*⟩

lemma *Max-wrt-mono*:
le x y $\Longrightarrow$ $x \in$ *Max-wrt le* $\Longrightarrow$ $y \in$ *Max-wrt le*
⟨*proof*⟩

end

context *total-preorder-on*
begin

lemma *Max-wrt-among-total-preorder*:
Max-wrt-among le A = $\{x \in$ *carrier* $\cap A$. $\forall y \in$ *carrier* $\cap A$. *le y x*$\}$
⟨*proof*⟩

lemma *Max-wrt-total-preorder*:
Max-wrt le = $\{x \in$ *carrier*. $\forall y \in$ *carrier*. *le y x*$\}$
⟨*proof*⟩

lemma *decompose-Max*:
assumes *A*: $A \subseteq$ *carrier*
defines $M \equiv$ *Max-wrt-among le A*
shows *restrict-relation A le* = $(\lambda x\, y.\ x \in A \land y \in M \lor (y \notin M \land$ *restrict-relation* $(A - M)$ *le x y*$))$
⟨*proof*⟩

end

1.6 Weak rankings

inductive *of-weak-ranking* :: *'alt set list* $\Rightarrow$ *'alt relation* **where**

$i \leq j \Longrightarrow i < \mathit{length}\ xs \Longrightarrow j < \mathit{length}\ xs \Longrightarrow x \in xs\ !\ i \Longrightarrow y \in xs\ !\ j \Longrightarrow$
$x \succeq[\mathit{of\text{-}weak\text{-}ranking}\ xs]\ y$

lemma *of-weak-ranking-Nil* [*simp*]: $\mathit{of\text{-}weak\text{-}ranking}\ [] = (\lambda\text{- -}.\ \mathit{False})$
⟨*proof*⟩

lemma *of-weak-ranking-Nil′* [*code*]: $\mathit{of\text{-}weak\text{-}ranking}\ []\ x\ y = \mathit{False}$
⟨*proof*⟩

lemma *of-weak-ranking-Cons* [*code*]:
$x \succeq[\mathit{of\text{-}weak\text{-}ranking}\ (z\#zs)]\ y \longleftrightarrow x \in z \land y \in \bigcup(\mathit{set}\ (z\#zs)) \lor x \succeq[\mathit{of\text{-}weak\text{-}ranking}\ zs]\ y$
(**is** $?lhs \longleftrightarrow ?rhs$)
⟨*proof*⟩

lemma *of-weak-ranking-indifference*:
assumes $A \in \mathit{set}\ xs\ x \in A\ y \in A$
shows $x \preceq[\mathit{of\text{-}weak\text{-}ranking}\ xs]\ y$
⟨*proof*⟩

lemma *of-weak-ranking-map*:
$\mathit{map\text{-}relation}\ f\ (\mathit{of\text{-}weak\text{-}ranking}\ xs) = \mathit{of\text{-}weak\text{-}ranking}\ (\mathit{map}\ ((-\text{`})\ f)\ xs)$
⟨*proof*⟩

lemma *of-weak-ranking-permute′*:
assumes $f\ \mathit{permutes}\ (\bigcup(\mathit{set}\ xs))$
shows $\mathit{map\text{-}relation}\ f\ (\mathit{of\text{-}weak\text{-}ranking}\ xs) = \mathit{of\text{-}weak\text{-}ranking}\ (\mathit{map}\ ((\text{`})\ (\mathit{inv}\ f))\ xs)$
⟨*proof*⟩

lemma *of-weak-ranking-permute*:
assumes $f\ \mathit{permutes}\ (\bigcup(\mathit{set}\ xs))$
shows $\mathit{of\text{-}weak\text{-}ranking}\ (\mathit{map}\ ((\text{`})\ f)\ xs) = \mathit{map\text{-}relation}\ (\mathit{inv}\ f)\ (\mathit{of\text{-}weak\text{-}ranking}\ xs)$
⟨*proof*⟩

definition *is-weak-ranking* **where**
$\mathit{is\text{-}weak\text{-}ranking}\ xs \longleftrightarrow (\{\} \notin \mathit{set}\ xs) \land$
$(\forall i\ j.\ i < \mathit{length}\ xs \land j < \mathit{length}\ xs \land i \neq j \longrightarrow xs\ !\ i \cap xs\ !\ j = \{\})$

definition *is-finite-weak-ranking* **where**
$\mathit{is\text{-}finite\text{-}weak\text{-}ranking}\ xs \longleftrightarrow \mathit{is\text{-}weak\text{-}ranking}\ xs \land (\forall x{\in}\mathit{set}\ xs.\ \mathit{finite}\ x)$

definition *weak-ranking* :: $'\mathit{alt}\ \mathit{relation} \Rightarrow\ '\mathit{alt}\ \mathit{set}\ \mathit{list}$ **where**
$\mathit{weak\text{-}ranking}\ R = (\mathit{SOME}\ xs.\ \mathit{is\text{-}weak\text{-}ranking}\ xs \land R = \mathit{of\text{-}weak\text{-}ranking}\ xs)$

lemma *is-weak-rankingI* [*intro?*]:
assumes $\{\} \notin \mathit{set}\ xs \bigwedge i\ j.\ i < \mathit{length}\ xs \Longrightarrow j < \mathit{length}\ xs \Longrightarrow i \neq j \Longrightarrow xs\ !\ i$

$\cap$ *xs* ! *j* = {}
shows *is-weak-ranking xs*
⟨*proof*⟩

lemma *is-weak-ranking-nonempty*: *is-weak-ranking xs* $\Longrightarrow$ {} $\notin$ *set xs*
⟨*proof*⟩

lemma *is-weak-rankingD*:
assumes *is-weak-ranking xs* $i <$ *length xs* $j <$ *length xs* $i \neq j$
shows *xs* ! $i \cap$ *xs* ! j = {}
⟨*proof*⟩

lemma *is-weak-ranking-iff*:
is-weak-ranking xs $\longleftrightarrow$ *distinct xs* $\wedge$ *disjoint* (*set xs*) $\wedge$ {} $\notin$ *set xs*
⟨*proof*⟩

lemma *is-weak-ranking-rev* [*simp*]: *is-weak-ranking* (*rev xs*) $\longleftrightarrow$ *is-weak-ranking xs*
⟨*proof*⟩

lemma *is-weak-ranking-map-inj*:
assumes *is-weak-ranking xs inj-on f* ($\bigcup$(*set xs*))
shows *is-weak-ranking* (*map* ((') *f*) *xs*)
⟨*proof*⟩

lemma *of-weak-ranking-rev* [*simp*]:
of-weak-ranking (*rev xs*) (x::$'a$) y $\longleftrightarrow$ *of-weak-ranking xs* y x
⟨*proof*⟩

lemma *is-weak-ranking-Nil* [*simp*, *code*]: *is-weak-ranking* []
⟨*proof*⟩

lemma *is-finite-weak-ranking-Nil* [*simp*, *code*]: *is-finite-weak-ranking* []
⟨*proof*⟩

lemma *is-weak-ranking-Cons-empty* [*simp*]:
$\neg$*is-weak-ranking* ({} # *xs*) ⟨*proof*⟩

lemma *is-finite-weak-ranking-Cons-empty* [*simp*]:
$\neg$*is-finite-weak-ranking* ({} # *xs*) ⟨*proof*⟩

lemma *is-weak-ranking-singleton* [*simp*]:
is-weak-ranking [x] $\longleftrightarrow$ $x \neq$ {}
⟨*proof*⟩

lemma *is-finite-weak-ranking-singleton* [*simp*]:
is-finite-weak-ranking [x] $\longleftrightarrow$ $x \neq$ {} $\wedge$ *finite* x
⟨*proof*⟩

lemma *is-weak-ranking-append*:
  $\mathit{is\text{-}weak\text{-}ranking}\ (\mathit{xs}\ @\ \mathit{ys}) \longleftrightarrow$
    $\mathit{is\text{-}weak\text{-}ranking}\ \mathit{xs} \wedge \mathit{is\text{-}weak\text{-}ranking}\ \mathit{ys} \wedge$
    $(\mathit{set}\ \mathit{xs} \cap \mathit{set}\ \mathit{ys} = \{\} \wedge \bigcup(\mathit{set}\ \mathit{xs}) \cap \bigcup(\mathit{set}\ \mathit{ys}) = \{\})$
  ⟨*proof*⟩

lemma *is-weak-ranking-Cons* [*code*]:
  $\mathit{is\text{-}weak\text{-}ranking}\ (x\ \#\ \mathit{xs}) \longleftrightarrow$
    $x \neq \{\} \wedge \mathit{is\text{-}weak\text{-}ranking}\ \mathit{xs} \wedge x \cap \bigcup(\mathit{set}\ \mathit{xs}) = \{\}$
  ⟨*proof*⟩

lemma *is-finite-weak-ranking-Cons* [*code*]:
  $\mathit{is\text{-}finite\text{-}weak\text{-}ranking}\ (x\ \#\ \mathit{xs}) \longleftrightarrow$
    $x \neq \{\} \wedge \mathit{finite}\ x \wedge \mathit{is\text{-}finite\text{-}weak\text{-}ranking}\ \mathit{xs} \wedge x \cap \bigcup(\mathit{set}\ \mathit{xs}) = \{\}$
  ⟨*proof*⟩

primrec *is-weak-ranking-aux* **where**
  $\mathit{is\text{-}weak\text{-}ranking\text{-}aux}\ A\ [] \longleftrightarrow \mathit{True}$
| $\mathit{is\text{-}weak\text{-}ranking\text{-}aux}\ A\ (x\#\mathit{xs}) \longleftrightarrow x \neq \{\} \wedge$
    $A \cap x = \{\} \wedge \mathit{is\text{-}weak\text{-}ranking\text{-}aux}\ (A \cup x)\ \mathit{xs}$

lemma *is-weak-ranking-aux*:
  $\mathit{is\text{-}weak\text{-}ranking\text{-}aux}\ A\ \mathit{xs} \longleftrightarrow A \cap \bigcup(\mathit{set}\ \mathit{xs}) = \{\} \wedge \mathit{is\text{-}weak\text{-}ranking}\ \mathit{xs}$
  ⟨*proof*⟩

lemma *is-weak-ranking-code* [*code*]:
  $\mathit{is\text{-}weak\text{-}ranking}\ \mathit{xs} \longleftrightarrow \mathit{is\text{-}weak\text{-}ranking\text{-}aux}\ \{\}\ \mathit{xs}$
  ⟨*proof*⟩

lemma *of-weak-ranking-altdef*:
  assumes $\mathit{is\text{-}weak\text{-}ranking}\ \mathit{xs}\ x \in \bigcup(\mathit{set}\ \mathit{xs})\ y \in \bigcup(\mathit{set}\ \mathit{xs})$
  shows $\mathit{of\text{-}weak\text{-}ranking}\ \mathit{xs}\ x\ y \longleftrightarrow$
    $\mathit{find\text{-}index}\ ((\in)\ x)\ \mathit{xs} \geq \mathit{find\text{-}index}\ ((\in)\ y)\ \mathit{xs}$
⟨*proof*⟩

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

lemma *restrict-relation-of-weak-ranking-Cons*:
  assumes $\mathit{is\text{-}weak\text{-}ranking}\ (A\ \#\ \mathit{As})$
  shows $\mathit{restrict\text{-}relation}\ (\bigcup(\mathit{set}\ \mathit{As}))\ (\mathit{of\text{-}weak\text{-}ranking}\ (A\ \#\ \mathit{As})) = \mathit{of\text{-}weak\text{-}ranking}$
As
⟨*proof*⟩

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

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

context
fixes x :: *'alt set* **and** xs :: *'alt set list*
assumes *wf*: *is-weak-ranking* $(x\#xs)$
begin

interpretation *R*: *total-preorder-on* $\bigcup$(*set* $(x\#xs)$) *of-weak-ranking* $(x\#xs)$
⟨*proof*⟩

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

lemma *of-weak-ranking-Cons'*:
assumes $a \in \bigcup(set\ (x\#xs))$ $b \in \bigcup(set\ (x\#xs))$
shows *of-weak-ranking* $(x\#xs)$ a $b \longleftrightarrow b \in x \vee (a \notin x \wedge$ *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 {} = []
| $A \neq \{\} \Longrightarrow A \subseteq$ *carrier* $\Longrightarrow$ *weak-ranking-aux* A =
Max-wrt-among le A # *weak-ranking-aux* (A − *Max-wrt-among le* A)
| $\neg(A \subseteq$ *carrier*$)$ $\Longrightarrow$ *weak-ranking-aux* A = *undefined*
⟨*proof*⟩
termination ⟨*proof*⟩

lemma *weak-ranking-aux-Union*:
$A \subseteq$ *carrier* $\Longrightarrow \bigcup$(*set* (*weak-ranking-aux* A)) = A
⟨*proof*⟩

lemma *weak-ranking-aux-wf*:
$A \subseteq$ *carrier* $\Longrightarrow$ *is-weak-ranking* (*weak-ranking-aux* A)
⟨*proof*⟩

lemma *of-weak-ranking-weak-ranking-aux'*:
assumes $A \subseteq$ *carrier* $x \in A$ $y \in A$
shows *of-weak-ranking* (*weak-ranking-aux* A) x $y \longleftrightarrow$ *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 $\bigcup$(*set As*) $\subseteq$ *carrier is-weak-ranking As*
of-weak-ranking As = *restrict-relation* ($\bigcup$(*set As*)) *le*
shows *As* = *weak-ranking-aux* ($\bigcup$(*set As*))

⟨*proof*⟩

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

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

lemma *weak-ranking-altdef*:
weak-ranking le $=$ *weak-ranking-aux carrier*
⟨*proof*⟩

lemma *weak-ranking-Union*: $\bigcup$(*set* (*weak-ranking le*)) $=$ *carrier*
⟨*proof*⟩

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

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

lemma *weak-ranking-index-unique*:
assumes *is-weak-ranking xs* $i <$ *length xs* $j <$ *length xs* $x \in$ *xs* ! i $x \in$ *xs* ! j
shows $i = j$
⟨*proof*⟩

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

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

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

lemma *hd-weak-ranking*:

assumes $x \in hd$ (*weak-ranking le*) $y \in$ *carrier*
shows *le y x*
⟨*proof*⟩

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

lemma *nth-weak-ranking-index*:
assumes $x \in$ *carrier*
shows *weak-ranking-index x* < *length* (*weak-ranking le*)
$x \in$ *weak-ranking le* ! *weak-ranking-index x*
⟨*proof*⟩

lemma *ranking-index-eqI*:
$i <$ *length* (*weak-ranking le*) $\Longrightarrow x \in$ *weak-ranking le* ! $i \Longrightarrow$ *weak-ranking-index* $x = i$
⟨*proof*⟩

lemma *ranking-index-le-iff* [*simp*]:
assumes $x \in$ *carrier* $y \in$ *carrier*
shows *weak-ranking-index x* $\geq$ *weak-ranking-index y* $\longleftrightarrow$ *le x y*
⟨*proof*⟩

end

lemma *weak-ranking-False* [*simp*]: *weak-ranking* (λ- -. *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 $\longleftrightarrow$ *total-preorder-on A R* $\wedge$ *finite A*
⟨*proof*⟩

lemma *finite-total-preorder-of-weak-ranking*:
assumes $\bigcup$(*set xs*) = *A is-finite-weak-ranking xs*
shows *finite-total-preorder-on A* (*of-weak-ranking xs*)
⟨*proof*⟩

lemma *weak-ranking-of-weak-ranking*:
assumes *is-finite-weak-ranking 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* ⟷ *R1* = *R2*
⟨*proof*⟩

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

lemma (**in** *preorder-on*) *preferred-alts-refl* [*simp*]: *x* ∈ *carrier* ⟹ *x* ∈ *preferred-alts le x*
⟨*proof*⟩

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

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

1.7 Rankings

definition *ranking* :: *'a relation* ⇒ *'a list* **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* ∈ *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*]: $i \in dom \Longrightarrow$ *preorder-on carrier* (*R i*)
assumes *not-in-dom* [*simp*]: $i \notin dom \Longrightarrow \neg R\ i\ x\ y$
begin

lemma *not-in-dom'*: $i \notin dom \Longrightarrow R\ i = (\lambda$- -. *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*]: $i \in agents \Longrightarrow$ *finite-total-preorder-on alts* (*R i*)
assumes *prefs-undefined* [*simp*]: $i \notin agents \Longrightarrow \neg R\ i\ x\ y$
begin

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

lemma *prefs-wf'* [*simp*]:
$i \in agents \Longrightarrow$ *total-preorder-on alts* (*R i*) $i \in agents \Longrightarrow$ *preorder-on alts* (*R i*)
⟨*proof*⟩

lemma *not-outside*:

assumes $x \preceq[R\ i]\ y$
shows $i \in agents\ x \in alts\ y \in alts$
⟨*proof*⟩

sublocale *preorder-family agents alts R*
⟨*proof*⟩

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

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

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

lemma (**in** $-$) *pref-profile-eqI*:
assumes *pref-profile-wf agents alts R1 pref-profile-wf agents alts R2*
assumes $\bigwedge x.\ x \in agents \Longrightarrow R1\ x = R2\ x$
shows $R1 = R2$
⟨*proof*⟩

end

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

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

lemma *permute-profile-map-relation*:
permute-profile $\sigma\ R = (\lambda i.\ map\text{-}relation\ (inv\ \sigma)\ (R\ i))$
⟨*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 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 (**in** *pref-profile-wf*) *wf-permute-alts*:

assumes σ *permutes alts*
shows *pref-profile-wf agents alts* (*permute-profile* σ 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 σ *permutes alts* $x \in$ *alts* $y \in$ *alts*
defines $R' \equiv$ *permute-profile* σ R
shows $\sigma\ x \preceq[R'\ i]\ \sigma\ y \longleftrightarrow x \preceq[R\ i]\ y$
⟨*proof*⟩

2.1 Pareto dominance

definition *Pareto* :: (*'agent* $\Rightarrow$ *'alt relation*) $\Rightarrow$ *'alt relation* **where**
$x \preceq[Pareto(R)]\ y \longleftrightarrow (\exists j.\ x \preceq[R\ j]\ x) \wedge (\forall i.\ x \preceq[R\ i]\ x \longrightarrow x \preceq[R\ i]\ y)$

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 $R = \{x.\ \exists y.\ y \succ[Pareto(R)]\ x\}$

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

context *preorder-family*
begin

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

lemma *Pareto-strict-iff*:
$x \prec[Pareto(R)]\ y \longleftrightarrow (\forall i{\in}dom.\ x \preceq[R\ i]\ y) \wedge (\exists i{\in}dom.\ x \prec[R\ i]\ y)$
⟨*proof*⟩

lemma *Pareto-strictI*:
assumes $\bigwedge i.\ i \in dom \Longrightarrow x \preceq[R\ i]\ y$ $i \in dom$ $x \prec[R\ i]\ y$
shows $x \prec[Pareto(R)]\ y$
⟨*proof*⟩

lemma *Pareto-strictI'*:
assumes $\bigwedge i.\ i \in dom \Longrightarrow x \preceq[R\ i]\ y$ $i \in dom$ $\neg x \succeq[R\ i]\ y$
shows $x \prec[Pareto(R)]\ y$
⟨*proof*⟩

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

lemma *pareto-loser-in-alts*:
assumes $x \in$ *pareto-losers* R
shows $x \in$ *carrier*
⟨*proof*⟩

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

end

2.2 Preferred alternatives

context *pref-profile-wf*
begin

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

lemma *preferred-alts-altdef*:
$i \in$ *agents* $\Longrightarrow$ *preferred-alts* $(R\ i)\ x = \{y{\in}alts.\ y \succeq[R\ i]\ x\}$
⟨*proof*⟩

end

2.3 Favourite alternatives

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

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

definition *has-unique-favorites* :: (*'agent*, *'alt*) *pref-profile* $\Rightarrow$ *bool* **where**
has-unique-favorites $R \longleftrightarrow (\forall\, i.$ *favorites* $R\ i = \{\} \lor$ *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*]: $i \notin$ *agents* $\Longrightarrow$ *favorites* $R\ i = \{\}$
⟨*proof*⟩

lemma *favorites-altdef'*:
  favorites R i = {*x*∈*alts*. ∀ *y*∈*alts*. *x* ≽[*R i*] *y*}
⟨*proof*⟩

lemma *favorites-subset-alts*: *favorites R i* ⊆ *alts*
  ⟨*proof*⟩

lemma *finite-favorites* [*simp*, *intro*]: *finite* (*favorites R i*)
  ⟨*proof*⟩

lemma *favorites-nonempty*: *i* ∈ *agents* ⟹ *favorites R i* ≠ {}
⟨*proof*⟩

lemma *favorites-permute*:
  assumes *i*: *i* ∈ *agents* **and** *perm*: *σ permutes alts*
  shows *favorites* (*permute-profile σ R*) *i* = *σ* ‘ *favorites R i*
⟨*proof*⟩

lemma *has-unique-favorites-altdef*:
  has-unique-favorites R ⟷ (∀ *i*∈*agents*. *is-singleton* (*favorites R i*))
⟨*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

lemma *unique-favorites*: *i* ∈ *agents* ⟹ *favorites R i* = {*favorite R i*}
  ⟨*proof*⟩

lemma *favorite-in-alts*: *i* ∈ *agents* ⟹ *favorite R i* ∈ *alts*
  ⟨*proof*⟩

end

2.4 Anonymous profiles

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

definition *anonymous-profile* :: (*'agent*, *'alt*) *pref-profile* ⇒ (*'agent*, *'alt*) *apref-profile*

  where *anonymous-profile-auxdef*:
    anonymous-profile R = *image-mset* (*weak-ranking* ∘ *R*) (*mset-set* {*i*. *R i* ≠ (λ-
-. *False*)})

lemma (**in** *pref-profile-wf*) *agents-eq*:

$agents = \{i.\ R\ i \neq (\lambda\text{- -}.\ False)\}$
⟨*proof*⟩

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

lemma (**in** *pref-profile-wf*) *anonymous-profile-permute*:
assumes σ *permutes alts* *finite agents*
shows $anonymous\text{-}profile\ (permute\text{-}profile\ \sigma\ R) =$
$image\text{-}mset\ (map\ ((`)\ \sigma))\ (anonymous\text{-}profile\ R)$
⟨*proof*⟩

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

2.5 Preference profiles from lists

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

definition *prefs-from-table-wf* **where**
$prefs\text{-}from\text{-}table\text{-}wf\ agents\ alts\ xss \longleftrightarrow agents \neq \{\} \wedge alts \neq \{\} \wedge distinct\ (map\ fst\ xss) \wedge$
$set\ (map\ fst\ xss) = agents \wedge (\forall xs \in set\ (map\ snd\ xss).\ \bigcup(set\ xs) = alts \wedge$
$is\text{-}finite\text{-}weak\text{-}ranking\ xs)$

lemma *prefs-from-table-wfI*:
assumes $agents \neq \{\}$ $alts \neq \{\}$ $distinct\ (map\ fst\ xss)$
assumes $set\ (map\ fst\ xss) = agents$
assumes $\bigwedge xs.\ xs \in set\ (map\ snd\ xss) \Longrightarrow \bigcup(set\ xs) = alts$
assumes $\bigwedge xs.\ xs \in set\ (map\ snd\ xss) \Longrightarrow is\text{-}finite\text{-}weak\text{-}ranking\ xs$
shows *prefs-from-table-wf agents alts xss*
⟨*proof*⟩

lemma *prefs-from-table-wfD*:
assumes *prefs-from-table-wf agents alts xss*
shows $agents \neq \{\}$ $alts \neq \{\}$ $distinct\ (map\ fst\ xss)$
and $set\ (map\ fst\ xss) = agents$
and $\bigwedge xs.\ xs \in set\ (map\ snd\ xss) \Longrightarrow \bigcup(set\ xs) = alts$
and $\bigwedge xs.\ xs \in set\ (map\ snd\ xss) \Longrightarrow is\text{-}finite\text{-}weak\text{-}ranking\ xs$
⟨*proof*⟩

lemma *pref-profile-from-tableI*:
prefs-from-table-wf agents alts xss $\Longrightarrow$ *pref-profile-wf agents alts* (*prefs-from-table xss*)
⟨*proof*⟩

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

lemma *prefs-from-table-undef*:
assumes *prefs-from-table-wf agents alts xss* $i \notin$ *agents*
shows *prefs-from-table xss* i = (λ- -. *False*)
⟨*proof*⟩

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

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

lemma *prefs-from-table-swap*:
$x \neq y \Longrightarrow$ *prefs-from-table* ((x,x')#(y,y')#*xs*) = *prefs-from-table* ((y,y')#(x,x')#*xs*)
⟨*proof*⟩

lemma *permute-prefs-from-table*:
assumes σ *permutes fst* ‘ *set xs*
shows *prefs-from-table xs* $\circ\ \sigma$ = *prefs-from-table* (*map* (λ(x,y). (*inv* σ x, y)) *xs*)
⟨*proof*⟩

lemma *permute-profile-from-table*:
assumes *wf*: *prefs-from-table-wf agents alts xss*
assumes *perm*: σ *permutes alts*
shows *permute-profile* σ (*prefs-from-table xss*) =
prefs-from-table (*map* (λ(x,y). (x, *map* ((‘) σ) y)) *xss*) (**is** *?f* = *?g*)
⟨*proof*⟩

2.6 Automatic evaluation of preference profiles

lemma *eval-prefs-from-table* [*simp*]:

$prefs\text{-}from\text{-}table\ []\, i = (\lambda\text{- -}.\ False)$
$prefs\text{-}from\text{-}table\ ((i,\ y)\ \#\ xs)\ i = of\text{-}weak\text{-}ranking\ y$
$i \neq j \Longrightarrow prefs\text{-}from\text{-}table\ ((j,\ y)\ \#\ xs)\ i = prefs\text{-}from\text{-}table\ xs\ i$
⟨*proof*⟩

lemma *eval-of-weak-ranking* [*simp*]:
$a \notin \bigcup(set\ xs) \Longrightarrow \neg of\text{-}weak\text{-}ranking\ xs\ a\ b$
$b \in x \Longrightarrow a \in \bigcup(set\ (x\#xs)) \Longrightarrow of\text{-}weak\text{-}ranking\ (x\ \#\ xs)\ a\ b$
$b \notin x \Longrightarrow of\text{-}weak\text{-}ranking\ (x\ \#\ xs)\ a\ b \longleftrightarrow of\text{-}weak\text{-}ranking\ xs\ a\ b$
⟨*proof*⟩

lemma *prefs-from-table-cong* [*cong*]:
assumes $prefs\text{-}from\text{-}table\ xs = prefs\text{-}from\text{-}table\ ys$
shows $prefs\text{-}from\text{-}table\ (x\#xs) = prefs\text{-}from\text{-}table\ (x\#ys)$
⟨*proof*⟩

definition *of-weak-ranking-Collect-ge* **where**
$of\text{-}weak\text{-}ranking\text{-}Collect\text{-}ge\ xs\ x = \{y.\ of\text{-}weak\text{-}ranking\ xs\ y\ x\}$

lemma *eval-Collect-of-weak-ranking*:
$Collect\ (of\text{-}weak\text{-}ranking\ xs\ x) = of\text{-}weak\text{-}ranking\text{-}Collect\text{-}ge\ (rev\ xs)\ x$
⟨*proof*⟩

lemma *of-weak-ranking-Collect-ge-empty* [*simp*]:
$of\text{-}weak\text{-}ranking\text{-}Collect\text{-}ge\ []\ x = \{\}$
⟨*proof*⟩

lemma *of-weak-ranking-Collect-ge-Cons* [*simp*]:
$y \in x \Longrightarrow of\text{-}weak\text{-}ranking\text{-}Collect\text{-}ge\ (x\#xs)\ y = \bigcup(set\ (x\#xs))$
$y \notin x \Longrightarrow of\text{-}weak\text{-}ranking\text{-}Collect\text{-}ge\ (x\#xs)\ y = of\text{-}weak\text{-}ranking\text{-}Collect\text{-}ge\ xs\ y$
⟨*proof*⟩

lemma *of-weak-ranking-Collect-ge-Cons'*:
$of\text{-}weak\text{-}ranking\text{-}Collect\text{-}ge\ (x\#xs) = (\lambda y.$
$(if\ y \in x\ then\ \bigcup(set\ (x\#xs))\ else\ of\text{-}weak\text{-}ranking\text{-}Collect\text{-}ge\ xs\ y))$
⟨*proof*⟩

lemma *anonymise-prefs-from-table*:
assumes *prefs-from-table-wf agents alts xs*
shows $anonymous\text{-}profile\ (prefs\text{-}from\text{-}table\ xs) = mset\ (map\ snd\ xs)$
⟨*proof*⟩

lemma *prefs-from-table-agent-permutation*:
assumes *wf*: *prefs-from-table-wf agents alts xs prefs-from-table-wf agents alts ys*
assumes *mset-eq*: $mset\ (map\ snd\ xs) = mset\ (map\ snd\ ys)$
obtains π **where** π $permutes\ agents\ prefs\text{-}from\text{-}table\ xs \circ \pi = prefs\text{-}from\text{-}table\ ys$
⟨*proof*⟩

lemma *permute-list-distinct*:
  assumes f ‘ $\{..<length\ xs\} \subseteq \{..<length\ xs\}$ *distinct xs*
  shows *permute-list f xs* $=$ *map* $(\lambda x.\ xs\ !\ f\ (index\ xs\ x))\ xs$
  ⟨*proof*⟩

lemma *image-mset-eq-permutation*:
  assumes $\{\#f\ x.\ x \in\#\ mset\text{-}set\ A\#\} = \{\#g\ x.\ x \in\#\ mset\text{-}set\ A\#\}$ *finite A*
  obtains π **where** π *permutes A* $\bigwedge x.\ x \in A \Longrightarrow g\ (\pi\ x) = f\ x$
⟨*proof*⟩

lemma *anonymous-profile-agent-permutation*:
  assumes *eq*: *anonymous-profile R1* $=$ *anonymous-profile R2*
  assumes *wf*: *pref-profile-wf agents alts R1 pref-profile-wf agents alts R2*
  assumes *fin*: *finite agents*
  obtains π **where** π *permutes agents* $R2 \circ \pi = R1$
⟨*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* $\equiv$ *pref-profile-wf agents alts*

lemma *finite-total-preorder-on-iff'* [*simp*]:
  finite-total-preorder-on alts R $\longleftrightarrow$ *total-preorder-on alts R*
  ⟨*proof*⟩

lemma *pref-profile-wfI'* [*intro?*]:
  $(\bigwedge i.\ i \in agents \Longrightarrow total\text{-}preorder\text{-}on\ alts\ (R\ i)) \Longrightarrow$
   $(\bigwedge i.\ i \notin agents \Longrightarrow R\ i = (\lambda\text{-}\ \text{-}.\ False)) \Longrightarrow is\text{-}pref\text{-}profile\ R$
  ⟨*proof*⟩

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

lemma *election* [*simp*,*intro*]: *election agents alts*

⟨*proof*⟩

context
fixes *R* **assumes** *R*: *total-preorder-on alts R*
begin

interpretation *R*: *total-preorder-on alts R* ⟨*proof*⟩

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

lemma *Max-wrt-prefs-nonempty*: *Max-wrt R* $\neq$ {}
⟨*proof*⟩

lemma *maximal-imp-preferred*:
x $\in$ *alts* $\Longrightarrow$ *Max-wrt R* $\subseteq$ *preferred-alts R x*
⟨*proof*⟩

end

end

end

3 Auxiliary facts about PMFs

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

The type of lotteries (a probability mass function)

type-synonym *'alt lottery* = *'alt pmf*

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

lemma *pmf-of-set-lottery*:
A $\neq$ {} $\Longrightarrow$ *finite A* $\Longrightarrow$ *A* $\subseteq$ *B* $\Longrightarrow$ *pmf-of-set A* $\in$ *lotteries-on B*
⟨*proof*⟩

lemma *pmf-of-list-lottery*:
pmf-of-list-wf xs $\Longrightarrow$ *set* (*map fst xs*) $\subseteq$ *A* $\Longrightarrow$ *pmf-of-list xs* $\in$ *lotteries-on A*
⟨*proof*⟩

lemma *return-pmf-in-lotteries-on* [*simp*,*intro*]:
x $\in$ *A* $\Longrightarrow$ *return-pmf x* $\in$ *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 $u :: 'a \Rightarrow real$
assumes *utility-le-iff*: $x \in carrier \Longrightarrow y \in carrier \Longrightarrow u\ x \le u\ y \longleftrightarrow x \preceq[le]\ y$
begin

lemma *utility-le*: $x \preceq[le]\ y \Longrightarrow u\ x \le u\ y$
⟨*proof*⟩

lemma *utility-less-iff*:
$x \in carrier \Longrightarrow y \in carrier \Longrightarrow u\ x < u\ y \longleftrightarrow x \prec[le]\ y$
⟨*proof*⟩

lemma *utility-less*: $x \prec[le]\ y \Longrightarrow u\ x < u\ y$
⟨*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 $p \in lotteries\text{-}on\ carrier$
shows $measure\text{-}pmf.expectation\ p\ u =$
$(\sum A \leftarrow weak\text{-}ranking\ le.\ u\ (SOME\ x.\ x \in A) * measure\text{-}pmf.prob\ p\ A)$
⟨*proof*⟩

lemma *scaled*: $c > 0 \Longrightarrow vnm\text{-}utility\ carrier\ le\ (\lambda x.\ c * u\ x)$
⟨*proof*⟩

lemma *add-right*:
assumes $\bigwedge x\ y.\ le\ x\ y \Longrightarrow f\ x \le f\ y$
shows $vnm\text{-}utility\ carrier\ le\ (\lambda x.\ u\ x + f\ x)$
⟨*proof*⟩

lemma *add-left*:
$(\bigwedge x\ y.\ le\ x\ y \Longrightarrow f\ x \le f\ y) \Longrightarrow vnm\text{-}utility\ carrier\ le\ (\lambda x.\ f\ x + u\ x)$
⟨*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: $\bigwedge x\ y.\ le\ x\ y \Longrightarrow le\ y\ x \Longrightarrow f\ x = f\ y$
  shows $\exists\varepsilon>0.\ vnm\text{-}utility\ carrier\ le\ (\lambda x.\ u\ x + \varepsilon * f\ x)$
$\langle proof \rangle$

lemma *diff-epsilon*:
  assumes $\bigwedge x\ y.\ le\ x\ y \Longrightarrow le\ y\ x \Longrightarrow f\ x = f\ y$
  shows $\exists\varepsilon>0.\ vnm\text{-}utility\ carrier\ le\ (\lambda x.\ u\ x - \varepsilon * f\ x)$
$\langle proof \rangle$

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* $\Rightarrow$ *'alt lottery relation* **where**
  $p \succeq[SD(R)]\ q \longleftrightarrow p \in lotteries\text{-}on\ \{x.\ R\ x\ x\} \wedge q \in lotteries\text{-}on\ \{x.\ R\ x\ x\} \wedge$
    $(\forall x.\ R\ x\ x \longrightarrow measure\text{-}pmf.prob\ p\ \{y.\ y \succeq[R]\ x\} \geq$
        $measure\text{-}pmf.prob\ q\ \{y.\ y \succeq[R]\ x\})$

lemma *SD-empty* [*simp*]: $SD\ (\lambda\text{- -}.\ False) = (\lambda\text{- -}.\ False)$
  $\langle proof \rangle$

Stochastic dominance over any relation is a preorder.

lemma *SD-refl*: $p \preceq[SD(R)]\ p \longleftrightarrow p \in lotteries\text{-}on\ \{x.\ R\ x\ x\}$
  $\langle proof \rangle$

lemma *SD-trans* [*simp*, *trans*]: $p \preceq[SD(R)]\ q \Longrightarrow q \preceq[SD(R)]\ r \Longrightarrow p \preceq[SD(R)]\ r$
  $\langle proof \rangle$

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

context *preorder-on*
begin

lemma *SD-preorder*:
$p \succeq[SD(le)]\ q \longleftrightarrow p \in$ *lotteries-on carrier* $\wedge\ q \in$ *lotteries-on carrier* $\wedge$
$(\forall x{\in}carrier.$ *measure-pmf.prob p* (*preferred-alts le x*) $\geq$
measure-pmf.prob q (*preferred-alts le x*))
⟨*proof*⟩

lemma *SD-preorderI* [*intro?*]:
assumes $p \in$ *lotteries-on carrier* $q \in$ *lotteries-on carrier*
assumes $\bigwedge x.\ x \in carrier \Longrightarrow$
measure-pmf.prob p (*preferred-alts le x*) $\geq$ *measure-pmf.prob q* (*preferred-alts le x*)
shows $p \succeq[SD(le)]\ q$
⟨*proof*⟩

lemma *SD-preorderD*:
assumes $p \succeq[SD(le)]\ q$
shows $p \in$ *lotteries-on carrier* $q \in$ *lotteries-on carrier*
and $\bigwedge x.\ x \in carrier \Longrightarrow$
measure-pmf.prob p (*preferred-alts le x*) $\geq$ *measure-pmf.prob q* (*preferred-alts le x*)
⟨*proof*⟩

lemma *SD-refl'* [*simp*]: $p \preceq[SD(le)]\ p \longleftrightarrow p \in$ *lotteries-on carrier*
⟨*proof*⟩

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

lemma *SD-singleton-left*:
assumes $x \in carrier$ $q \in$ *lotteries-on carrier*
shows *return-pmf x* $\preceq[SD(le)]\ q \longleftrightarrow (\forall y{\in}$*set-pmf q*$.\ x \preceq[le]\ y)$
⟨*proof*⟩

lemma *SD-singleton-right*:
assumes *x*: $x \in carrier$ **and** *q*: $q \in$ *lotteries-on carrier*
shows $q \preceq[SD(le)]$ *return-pmf x* $\longleftrightarrow (\forall y{\in}$*set-pmf q*$.\ y \preceq[le]\ x)$
⟨*proof*⟩

lemma *SD-strict-singleton-left*:
assumes $x \in carrier$ $q \in$ *lotteries-on carrier*
shows *return-pmf x* $\prec[SD(le)]\ q \longleftrightarrow (\forall y{\in}$*set-pmf q*$.\ x \preceq[le]\ y) \wedge (\exists y{\in}$*set-pmf q*$.\ (x \prec[le]\ y))$
⟨*proof*⟩

lemma *SD-strict-singleton-right*:
assumes $x \in$ *carrier* $q \in$ *lotteries-on carrier*
shows $q \prec[SD(le)]$ *return-pmf* $x \longleftrightarrow (\forall\, y{\in}set\text{-}pmf\ q.\ y \preceq[le]\ x) \wedge (\exists\, y{\in}set\text{-}pmf\ q.\ (y \prec[le]\ x))$
⟨*proof*⟩

lemma *SD-singleton* [*simp*]:
$x \in carrier \Longrightarrow y \in carrier \Longrightarrow return\text{-}pmf\ x \preceq[SD(le)]\ return\text{-}pmf\ y \longleftrightarrow x \preceq[le]\ y$
⟨*proof*⟩

lemma *SD-strict-singleton* [*simp*]:
$x \in carrier \Longrightarrow y \in carrier \Longrightarrow return\text{-}pmf\ x \prec[SD(le)]\ return\text{-}pmf\ y \longleftrightarrow x \prec[le]\ y$
⟨*proof*⟩

end

context *pref-profile-wf*
begin

context
fixes *i* **assumes** *i*: $i \in agents$
begin

interpretation *Ri*: *preorder-on alts R i* ⟨*proof*⟩

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

end
end

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

4.2 Stochastic Dominance for preference profiles

context *pref-profile-wf*
begin

lemma *SD-pref-profile*:
assumes $i \in agents$
shows $p \succeq[SD(R\ i)]\ q \longleftrightarrow p \in lotteries\text{-}on\ alts \wedge q \in lotteries\text{-}on\ alts \wedge$
$(\forall\, x{\in}alts.\ measure\text{-}pmf.prob\ p\ (preferred\text{-}alts\ (R\ i)\ x) \geq$

$measure\text{-}pmf.prob\ q\ (preferred\text{-}alts\ (R\ i)\ x))$
⟨*proof*⟩

lemma *SD-pref-profileI* [*intro?*]:
  assumes $i \in agents\ p \in lotteries\text{-}on\ alts\ q \in lotteries\text{-}on\ alts$
  assumes $\bigwedge x.\ x \in alts \Longrightarrow$
    $measure\text{-}pmf.prob\ p\ (preferred\text{-}alts\ (R\ i)\ x) \geq$
    $measure\text{-}pmf.prob\ q\ (preferred\text{-}alts\ (R\ i)\ x)$
  shows $p \succeq[SD(R\ i)]\ q$
  ⟨*proof*⟩

lemma *SD-pref-profileD*:
  assumes $i \in agents\ p \succeq[SD(R\ i)]\ q$
  shows $p \in lotteries\text{-}on\ alts\ q \in lotteries\text{-}on\ alts$
  and $\bigwedge x.\ x \in alts \Longrightarrow$
    $measure\text{-}pmf.prob\ p\ (preferred\text{-}alts\ (R\ i)\ x) \geq$
    $measure\text{-}pmf.prob\ q\ (preferred\text{-}alts\ (R\ i)\ x)$
  ⟨*proof*⟩

end

4.3 SD efficient lotteries

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

context *pref-profile-wf*
begin

sublocale *SD*: $preorder\text{-}family\ agents\ lotteries\text{-}on\ alts\ SD \circ R$ ⟨*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*:
  $SD\text{-}efficient\ R\ p \longleftrightarrow \neg(\exists\, q{\in}lotteries\text{-}on\ alts.\ q \succ[Pareto\ (SD \circ R)]\ p)$
⟨*proof*⟩

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

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

$i \in agents$ $q \succ[SD(R\ i)]\ p$
shows $\neg SD\text{-}efficient\ R\ p$
⟨*proof*⟩

lemma *SD-inefficientI′*:
assumes $q \in lotteries\text{-}on\ alts$ $\bigwedge i.\ i \in agents \Longrightarrow q \succeq[SD(R\ i)]\ p$
$\exists\, i \in agents.\ q \succ[SD(R\ i)]\ p$
shows $\neg SD\text{-}efficient\ R\ p$
⟨*proof*⟩

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

lemma *SD-efficientD*:
assumes $SD\text{-}efficient\ R\ p$ $q \in lotteries\text{-}on\ alts$
and $\bigwedge i.\ i \in agents \Longrightarrow q \succeq[SD(R\ i)]\ p$ $\exists\, i{\in}agents.\ \neg(q \preceq[SD(R\ i)]\ p)$
shows *False*
⟨*proof*⟩

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

end

4.4 Equivalence proof

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

context *finite-total-preorder-on*
begin

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

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

lemma *SD-iff-expected-utilities-le*:
assumes $p \in lotteries\text{-}on\ carrier$ $q \in lotteries\text{-}on\ carrier$
shows $p \preceq[SD(le)]\ q \longleftrightarrow$
$(\forall\, u.\ is\text{-}vnm\text{-}utility\ u \longrightarrow measure\text{-}pmf.expectation\ p\ u \le measure\text{-}pmf.expectation\ q\ u)$

⟨*proof*⟩

lemma *not-strict-SD-iff*:
  assumes $p \in$ *lotteries-on carrier* $q \in$ *lotteries-on carrier*
  shows $\neg(p \prec[SD(le)]\ q) \longleftrightarrow$
  $(\exists\, u.$ *is-vnm-utility* $u \wedge$ *measure-pmf.expectation* $q\ u \leq$ *measure-pmf.expectation* $p\ u)$
⟨*proof*⟩

lemma *strict-SD-iff*:
  assumes $p \in$ *lotteries-on carrier* $q \in$ *lotteries-on carrier*
  shows $(p \prec[SD(le)]\ q) \longleftrightarrow$
  $(\forall\, u.$ *is-vnm-utility* $u \longrightarrow$ *measure-pmf.expectation* $p\ u <$ *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*: $\neg$*SD-efficient* $R\ p'$
  assumes *support*: *set-pmf* $p' \subseteq$ *set-pmf* p
  assumes *lotteries*: $p \in$ *lotteries-on alts*
  shows $\neg$*SD-efficient* $R\ p$
⟨*proof*⟩

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

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

lemma *SD-efficient-iff*:
  assumes $p \in$ *lotteries-on alts*

shows $\textit{SD-efficient } R\ p \longleftrightarrow \textit{SD-efficient } R\ (\textit{pmf-of-set } (\textit{set-pmf } p))$
⟨*proof*⟩

lemma *SD-efficient-no-pareto-loser*:
assumes *efficient*: $\textit{SD-efficient } R\ p$ **and** *p-wf*: $p \in \textit{lotteries-on alts}$
shows $\textit{set-pmf } p \cap \textit{pareto-losers } R = \{\}$
⟨*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 $p \in \textit{lotteries-on alts}\ q \in \textit{lotteries-on alts}\ q \succ[\textit{Pareto}(SD \circ R)]\ p$
$\textit{set-pmf } p = \textit{set-pmf } q$
obtains r **where** $r \in \textit{lotteries-on alts}\ r \succeq[\textit{Pareto}(SD \circ R)]\ q\ \textit{set-pmf } r \subset \textit{set-pmf}$
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 $p :: {}'\textit{alt lottery}$
assumes *lott*: $p \in \textit{lotteries-on alts}$
begin

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

private lemma *improve-lottery*:
assumes $\neg\textit{SD-efficient } R\ q$
defines $r \equiv \textit{improve-lottery } q$
shows $r \in \textit{lotteries-on alts}\ r \succ[\textit{Pareto}(SD{\circ}R)]\ q$
$\bigwedge r'.\ r' \in \textit{lotteries-on alts} \Longrightarrow r' \succ[\textit{Pareto}(SD{\circ}R)]\ q \Longrightarrow \neg(\textit{set-pmf } r' \subset$
$\textit{set-pmf } r)$
⟨*proof*⟩ **fun** $\textit{sd-chain} :: \textit{nat} \Rightarrow {}'\textit{alt lottery option}$ **where**
$\textit{sd-chain } 0 = \textit{Some } p$
$|\ \textit{sd-chain } (\textit{Suc } n) =$
$(\textit{case sd-chain } n \textit{ of}$
$\textit{None} \Rightarrow \textit{None}$
$|\ \textit{Some } p \Rightarrow \textit{if SD-efficient } R\ p \textit{ then None else Some } (\textit{improve-lottery } p))$

private lemma *sd-chain-None-propagate*:

$m \geq n \Longrightarrow$ *sd-chain* n = *None* $\Longrightarrow$ *sd-chain* m = *None*
⟨*proof*⟩ **lemma** *sd-chain-Some-propagate*:
$m \geq n \Longrightarrow$ *sd-chain* m = *Some* $q \Longrightarrow \exists\, q'$. *sd-chain* n = *Some* q'
⟨*proof*⟩ **lemma** *sd-chain-NoneD*:
sd-chain n = *None* $\Longrightarrow \exists\, n\ p$. *sd-chain* n = *Some* $p \wedge$ *SD-efficient* $R\ p$
⟨*proof*⟩ **lemma** *sd-chain-lottery*: *sd-chain* n = *Some* $q \Longrightarrow q \in$ *lotteries-on alts*
⟨*proof*⟩ **lemma** *sd-chain-Suc*:
assumes *sd-chain* m = *Some* q
assumes *sd-chain* (*Suc* m) = *Some* r
shows $q \prec$[*Pareto*(*SD*$\circ$*R*)] r
⟨*proof*⟩ **lemma** *sd-chain-strictly-preferred*:
assumes $m < n$
assumes *sd-chain* m = *Some* q
assumes *sd-chain* n = *Some* s
shows $q \prec$[*Pareto*(*SD*$\circ$*R*)] s
⟨*proof*⟩ **lemma** *sd-chain-preferred*:
assumes $m \leq n$
assumes *sd-chain* m = *Some* q
assumes *sd-chain* n = *Some* s
shows $q \preceq$[*Pareto*(*SD*$\circ$*R*)] s
⟨*proof*⟩

lemma *SD-efficient-lottery-exists*:
obtains q **where** $q \in$ *lotteries-on alts* $q \succeq$[*Pareto*(*SD*$\circ$*R*)] p *SD-efficient* $R\ q$
⟨*proof*⟩

end

lemma
assumes $p \in$ *lotteries-on alts*
shows $\exists\, q \in$*lotteries-on alts*. $q \succeq$[*Pareto*(*SD*$\circ$*R*)] $p \wedge$ *SD-efficient* $R\ q$
⟨*proof*⟩

end

end

5 Social Decision Schemes

theory *Social-Decision-Schemes*
imports
Complex-Main
HOL−Probability.Probability
Preference-Profiles
Elections
Order-Predicates
Stochastic-Dominance
SD-Efficiency
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-wf*: *is-pref-profile R* ⟹ *sds R* ∈ *lotteries*

5.3 Anonymity

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

locale *anonymous-sds* = *social-decision-scheme agents alts sds*
for *agents* :: *'agent set* **and** *alts* :: *'alt set* **and** *sds* +
assumes *anonymous*: *π permutes agents* ⟹ *is-pref-profile R* ⟹ *sds* (*R* ∘ *π*) = *sds R*
begin

lemma *anonymity-prefs-from-table*:

assumes *prefs-from-table-wf agents alts xs prefs-from-table-wf agents alts ys*
assumes *mset* (*map snd xs*) = *mset* (*map snd ys*)
shows *sds* (*prefs-from-table xs*) = *sds* (*prefs-from-table ys*)
⟨*proof*⟩

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

end

5.4 Neutrality

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

locale *neutral-sds* = *social-decision-scheme agents alts sds*
for *agents* :: *'agent set* **and** *alts* :: *'alt set* **and** *sds* +
assumes *neutral*: *σ permutes alts* ⟹ *is-pref-profile R* ⟹
sds (*permute-profile σ R*) = *map-pmf σ* (*sds R*)
begin

Alternative formulation of neutrality that shows that our definition is equivalent to that in the paper.

lemma *neutral'*:
assumes *σ permutes alts*
assumes *is-pref-profile R*
assumes *a* ∈ *alts*
shows *pmf* (*sds* (*permute-profile σ R*)) (*σ a*) = *pmf* (*sds R*) *a*
⟨*proof*⟩

end

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

lemma *sds-anonymous-neutral*:
assumes *perm*: *σ permutes alts* **and** *wf*: *is-pref-profile R1 is-pref-profile R2*
assumes *eq*: *anonymous-profile R1* =
image-mset (*map* ((\`) *σ*)) (*anonymous-profile R2*)
shows *sds R1* = *map-pmf σ* (*sds R2*)

⟨*proof*⟩

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* ((‘) *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* ((‘) *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* ((‘) (*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

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 an 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* $\Longrightarrow$ *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* $\in$ *lotteries*
  assumes $\bigwedge i.\ i \in agents \Longrightarrow q \succeq[SD(R\ i)]\ sds\ R\ i \in agents\ q \succ[SD(R\ i)]\ sds\ R$
  shows *P*
⟨*proof*⟩

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

sublocale *ex-post-efficient-sds*
⟨*proof*⟩

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

lemma *SD-inefficient-support*:
  assumes *A*: $A \neq \{\}\ A \subseteq alts$ **and** *inefficient*: $\neg$*SD-efficient R* (*pmf-of-set A*)
  assumes *wf*: *is-pref-profile R*
  shows $\exists x \in A.\ pmf\ (sds\ R)\ x = 0$
⟨*proof*⟩

lemma *SD-inefficient-support'*:
  assumes *wf*: *is-pref-profile R*
  assumes *A*: $A \neq \{\}\ A \subseteq alts$ **and**
    wit: $p \in lotteries\ \forall i \in agents.\ p \succeq[SD(R\ i)]\ pmf\text{-}of\text{-}set\ A\ i \in agents$
      $\neg p \preceq[SD(R\ i)]\ pmf\text{-}of\text{-}set\ A$
  shows $\exists x \in A.\ pmf\ (sds\ R)\ x = 0$
⟨*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 agent if the lottery obtained if the agent submits the alternative preferences strictly SD-dominates that obtained if the original preferences are submitted. (SD-dominated w.r.t. the original preferences)

definition *manipulable-profile*
  $:: ('agent, 'alt)\ pref\text{-}profile \Rightarrow 'agent \Rightarrow 'alt\ relation \Rightarrow bool$ **where**
 $manipulable\text{-}profile\ R\ i\ Ri' \longleftrightarrow sds\ (R(i := Ri')) \succ[SD\ (R\ i)]\ sds\ R$

end

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

locale *strategyproof-sds* = *social-decision-scheme agents alts sds*
 for $agents :: 'agent\ set$ **and** $alts :: 'alt\ set$ **and** *sds* +
 assumes *strategyproof*:
  $is\text{-}pref\text{-}profile\ R \Longrightarrow i \in agents \Longrightarrow total\text{-}preorder\text{-}on\ alts\ Ri' \Longrightarrow$
   $\neg manipulable\text{-}profile\ R\ i\ Ri'$

5.8 Strong strategyproofness

context *social-decision-scheme*
begin

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

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

definition *strongly-strategyproof-profile*
  $:: ('agent, 'alt)\ pref\text{-}profile \Rightarrow 'agent \Rightarrow 'alt\ relation \Rightarrow bool$ **where**
 $strongly\text{-}strategyproof\text{-}profile\ R\ i\ Ri' \longleftrightarrow sds\ R \succeq[SD\ (R\ i)]\ sds\ (R(i := Ri'))$

lemma *strongly-strategyproof-profileI* [*intro*]:
 assumes $is\text{-}pref\text{-}profile\ R\ total\text{-}preorder\text{-}on\ alts\ Ri'\ i \in agents$
 assumes $\bigwedge x.\ x \in alts \Longrightarrow lottery\text{-}prob\ (sds\ (R(i := Ri')))\ (preferred\text{-}alts\ (R\ i)$
$x)$
        $\leq lottery\text{-}prob\ (sds\ R)\ (preferred\text{-}alts\ (R\ i)\ x)$
 shows *strongly-strategyproof-profile R i Ri'*
⟨*proof*⟩

lemma *strongly-strategyproof-imp-not-manipulable*:
 assumes *strongly-strategyproof-profile R i Ri'*
 shows $\neg manipulable\text{-}profile\ R\ i\ Ri'$

⟨*proof*⟩

end

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

locale *strongly-strategyproof-sds* = *social-decision-scheme agents alts sds*
for *agents* :: *'agent set* **and** *alts* :: *'alt set* **and** *sds* +
assumes *strongly-strategyproof*:
is-pref-profile $R \Longrightarrow i \in$ *agents* $\Longrightarrow$ *total-preorder-on alts* $Ri' \Longrightarrow$
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' $R = (\lambda i\ x\ y.$
$x \in$ *alts'* $\wedge\ y \in$ *alts'* $\wedge\ i \in$ *agents'* $\wedge$
$(x = y \vee x \notin$ *alts* $\vee\ i \notin$ *agents* $\vee\ (y \in$ *alts* $\wedge\ R\ i\ x\ y)))$

lemma *lift-pref-profile-wf*:
assumes *pref-profile-wf agents alts R*
assumes *agents* ⊆ *agents' alts* ⊆ *alts' finite alts'*
defines $R' \equiv$ *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* ⊆ *agents'*

shows *lift-pref-profile agents alts agents′ alts′* $(R \circ \pi) =$
lift-pref-profile agents alts agents′ alts′ $R \circ \pi$
⟨*proof*⟩

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

lemma *lotteries-on-subset*: $A \subseteq B \Longrightarrow p \in$ *lotteries-on* $A \Longrightarrow p \in$ *lotteries-on* B
⟨*proof*⟩

lemma *lottery-prob-carrier*: $p \in$ *lotteries-on* $A \Longrightarrow$ *measure-pmf.prob* p A = *1*
⟨*proof*⟩

context
fixes *agents alts R agents′ alts′ R′*
assumes *R-wf*: *pref-profile-wf agents alts R*
assumes *election*: *agents* $\subseteq$ *agents′ alts* $\subseteq$ *alts′ alts* $\neq \{\}$ *agents* $\neq \{\}$ *finite alts′*
defines $R' \equiv$ *lift-pref-profile agents alts agents′ alts′ R*
begin

interpretation *R*: *pref-profile-wf agents alts R* ⟨*proof*⟩
interpretation *R′*: *pref-profile-wf agents′ alts′ R′*
⟨*proof*⟩

lemma *lift-pref-profile-strict-iff*:
$x \prec$[*lift-pref-profile agents alts agents′ alts′ R i*] $y \longleftrightarrow$
$i \in$ *agents* $\wedge$ (($y \in$ *alts* $\wedge$ $x \in$ *alts′* $-$ *alts*) $\vee$ $x \prec$[*R i*] y)
⟨*proof*⟩

lemma *preferred-alts-lift-pref-profile*:
assumes *i*: $i \in$ *agents′* **and** *x*: $x \in$ *alts′*
shows *preferred-alts* (R' i) x =
(*if* $i \in$ *agents* $\wedge$ $x \in$ *alts then preferred-alts* (R i) x *else alts′*)
⟨*proof*⟩

lemma *lift-pref-profile-Pareto-iff*:
$x \preceq$[*Pareto*(R')] $y \longleftrightarrow x \in$ *alts′* $\wedge$ $y \in$ *alts′* $\wedge$ ($x \notin$ *alts* $\vee$ $x \preceq$[*Pareto*(R)] y)
⟨*proof*⟩

lemma *lift-pref-profile-Pareto-strict-iff*:
$x \prec$[*Pareto*(R')] $y \longleftrightarrow x \in$ *alts′* $\wedge$ $y \in$ *alts′* $\wedge$ ($x \notin$ *alts* $\wedge$ $y \in$ *alts* $\vee$ $x \prec$[*Pareto*(R)] y)
⟨*proof*⟩

lemma *pareto-losers-lift-pref-profile*:

shows $\textit{pareto-losers } R' = \textit{pareto-losers } R \cup (\textit{alts}' - \textit{alts})$
$\langle \textit{proof} \rangle$

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

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

lemma *lift-SD-iff'*:
$p \in \textit{lotteries-on alts} \Longrightarrow q \in \textit{lotteries-on alts} \Longrightarrow i \in \textit{agents}' \Longrightarrow$
$p \preceq[SD(R'\ i)]\ q \longleftrightarrow i \notin \textit{agents} \vee p \preceq[SD(R\ i)]\ q$
$\langle \textit{proof} \rangle$

end

lemma *lift-SD-strict-iff*:
assumes $p \in \textit{lotteries-on alts } q \in \textit{lotteries-on alts}$ **and** i: $i \in \textit{agents}$
shows $p \prec[SD(R'\ i)]\ q \longleftrightarrow p \prec[SD(R\ i)]\ q$
$\langle \textit{proof} \rangle$

lemma *lift-Pareto-SD-iff*:
assumes $p \in \textit{lotteries-on alts } q \in \textit{lotteries-on alts}$
shows $p \preceq[Pareto(SD \circ R')]\ q \longleftrightarrow p \preceq[Pareto(SD \circ R)]\ q$
$\langle \textit{proof} \rangle$

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

lemma *lift-SD-efficient-iff*:
assumes p: $p \in \textit{lotteries-on alts}$
shows $\textit{SD-efficient } R'\ p \longleftrightarrow \textit{SD-efficient } R\ p$
$\langle \textit{proof} \rangle$

end

locale *sds-lowering* =
ex-post-efficient-sds agents alts sds
for *agents* :: *'agent set* **and** *alts* :: *'alt set* **and** *sds* +
fixes *agents' alts'*

assumes *agents′-subset*: $agents' \subseteq agents$ **and** *alts′-subset*: $alts' \subseteq alts$
and *agents′-nonempty* [*simp*]: $agents' \neq \{\}$ **and** *alts′-nonempty* [*simp*]: $alts' \neq \{\}$
begin

lemma *finite-agents′* [*simp*]: *finite agents′*
⟨*proof*⟩

lemma *finite-alts′* [*simp*]: *finite alts′*
⟨*proof*⟩

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

definition *lowered* :: (′*agent*, ′*alt*) *pref-profile* ⇒ ′*alt lottery* **where**
lowered = *sds* ∘ *lift*

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

sublocale *lowered*: *election agents′ alts′*
⟨*proof*⟩

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

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

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

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

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

lemma *lowered-in-lotteries* [*simp*]: *lowered.is-pref-profile R* ⟹ *lowered R* ∈ *lotteries*
⟨*proof*⟩

end

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

sublocale *lowered*: *anonymous-sds agents′ alts′ lowered*
⟨*proof*⟩

end

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

sublocale *lowered*: *neutral-sds agents′ alts′ lowered*
⟨*proof*⟩

end

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

sublocale *sd-efficient-sds agents′ alts′ lowered*
⟨*proof*⟩

end

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

sublocale *strategyproof-sds agents′ alts′ lowered*
⟨*proof*⟩

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 agents' most preferred alternatives uniformly at random. Note that this definition also works for weak preferences.

definition *random-dictatorship* :: *'agent set* ⇒ *'alt set* ⇒ (*'agent*, *'alt*) *pref-profile* ⇒ *'alt lottery* **where**
  random-dictatorship-auxdef:
  random-dictatorship agents alts R =
      do {
        i ← *pmf-of-set agents*;
        pmf-of-set (*Max-wrt-among* (*R i*) *alts*)
      }

context *election*
begin

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

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

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

lemma *pmf-random-dictatorship*:
  assumes *is-pref-profile R*
  shows *pmf* (*RD R*) *x* =
  ($\sum$ *i*∈*agents*. *indicator* (*favorites R i*) *x* /
  real (*card* (*favorites R i*))) / *real* (*card agents*)
⟨*proof*⟩

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

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

7.1 Anonymity

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

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

7.2 Neutrality

The proof of neutrality is similar to that of anonymity. We have proven elsewhere that the most preferred alternatives of an agent in a profile with permuted alternatives are simply the image of the originally preferred alternatives. Since we pick one alternative from the most preferred alternatives of the selected agent uniformly at random, this means that we effectively pick an agent, then pick 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*
⟨*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 strategyproof, 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* ::
$'agent\ set \Rightarrow 'alt\ set \Rightarrow ('agent, 'alt)\ pref\text{-}profile \Rightarrow 'alt\ lottery$ **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 $\Longrightarrow$ *agents* $\neq$ {} $\Longrightarrow$
random-serial-dictatorship agents alts R =
do {
$i \leftarrow$ *pmf-of-set agents*;
random-serial-dictatorship (*agents* − {*i*}) (*Max-wrt-among* (*R i*) *alts*) *R*
}
⟨*proof*⟩

We define the RSD winners w.r.t. a given set of alternatives and a fixed permutation (i.e. list) of agents. In contrast to the above definition, the RSD winners are determined by traversing the list of agents from right to left. This may seem strange, but it makes induction much easier, since induction over *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* (λ*i alts*. *Max-wrt-among* (*R i*) *alts*) *agents alts*

lemma *rsd-winners-empty* [*simp*]: *rsd-winners R alts* [] = *alts*
⟨*proof*⟩

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

lemma *rsd-winners-map* [*simp*]:
rsd-winners R alts (*map f agents*) = *rsd-winners* ($R \circ f$) *alts agents*
⟨*proof*⟩

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' $\leftarrow$ *pmf-of-set* (*permutations-of-set agents*);
pmf-of-set (*rsd-winners R alts agents'*)
}
⟨*proof*⟩

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

lemma *random-serial-dictatorship-foldl*:
assumes *finite agents*
shows *random-serial-dictatorship agents alts R* =
do {

$agents' \leftarrow$ *pmf-of-set* (*permutations-of-set agents*);
pmf-of-set (*foldl* (λ*alts i. Max-wrt-among* (*R i*) *alts*) *alts agents'*)
}
⟨*proof*⟩

8.1 Auxiliary facts about RSD

8.1.1 Pareto-equivalence classes

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

definition *RSD-pareto-eqclass* **where**
RSD-pareto-eqclass agents alts R A $\longleftrightarrow$
$A \neq \{\} \wedge A \subseteq alts \wedge (\forall x{\in}A.\ \forall y{\in}alts.\ y \in A \longleftrightarrow (\forall i{\in}agents.\ R\ i\ x\ y))$

lemma *RSD-pareto-eqclassI*:
assumes $A \neq \{\}$ $A \subseteq alts$ $\bigwedge x\ y.\ x \in A \Longrightarrow y \in alts \Longrightarrow y \in A \longleftrightarrow (\forall i{\in}agents.\ R\ i\ x\ y)$
shows *RSD-pareto-eqclass agents alts R A*
⟨*proof*⟩

lemma *RSD-pareto-eqclassD*:
assumes *RSD-pareto-eqclass agents alts R A*
shows $A \neq \{\}$ $A \subseteq alts$ $\bigwedge x\ y.\ x \in A \Longrightarrow y \in alts \Longrightarrow y \in A \longleftrightarrow (\forall i{\in}agents.\ R\ i\ x\ y)$
⟨*proof*⟩

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

lemma *RSD-pareto-eqclass-empty* [*simp*, *intro*!]:
$alts \neq \{\} \Longrightarrow$ *RSD-pareto-eqclass* {} *alts R alts*
⟨*proof*⟩

lemma (**in** *pref-profile-wf*) *RSD-pareto-eqclass-insert*:
assumes *RSD-pareto-eqclass agents' alts R A finite alts*
$i \in agents$ $agents' \subseteq agents$
shows *RSD-pareto-eqclass* (*insert i agents'*) *alts R* (*Max-wrt-among* (*R i*) *A*)
⟨*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*$'$ $\subseteq$ *agents*
shows *rsd-winners R alts*$'$ *agents*$'$ $\subseteq$ *alts*$'$
$\langle$*proof*$\rangle$

There is always at least one RSD winner.

lemma *rsd-winners-nonempty*:
assumes *finite*: *finite alts* **and** *alts*$' \neq \{\}$ *set agents*$'$ $\subseteq$ *agents alts*$'$ $\subseteq$ *alts*
shows *rsd-winners R alts*$'$ *agents*$' \neq \{\}$
$\langle$*proof*$\rangle$

Obviously, the set of RSD winners is always finite.

lemma *rsd-winners-finite*:
assumes *set agents*$'$ $\subseteq$ *agents finite alts alts*$'$ $\subseteq$ *alts*
shows *finite* (*rsd-winners R alts*$'$ *agents*$'$)
$\langle$*proof*$\rangle$

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

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

lemma *RSD-pareto-eqclass-rsd-winners-aux*:
assumes *finite*: *finite alts* **and** *alts* $\neq \{\}$ **and** *set agents*$'$ $\subseteq$ *agents*
shows *RSD-pareto-eqclass* (*set agents*$'$) *alts R* (*rsd-winners R alts agents*$'$)
$\langle$*proof*$\rangle$

lemma *RSD-pareto-eqclass-rsd-winners*:
assumes *finite*: *finite alts* **and** *alts* $\neq \{\}$ **and** *set agents*$'$ = *agents*
shows *RSD-pareto-eqclass agents alts R* (*rsd-winners R alts agents*$'$)
$\langle$*proof*$\rangle$

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 $\longleftrightarrow$ $A \neq \{\} \wedge (\forall x{\in}A.\ \forall y{\in}A.\ S\ x\ y)$

private lemma *indiff-set-mono*: *indiff-set S A* $\Longrightarrow$ $B \subseteq A$ $\Longrightarrow$ $B \neq \{\}$ $\Longrightarrow$ *indiff-set 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 $\longleftrightarrow$ *indiff-set S B* $\wedge$ $(\forall x{\in}A.\ \forall y{\in}B.\ S\ x\ y)$

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

private lemma *indiff-set-Max-wrt-among*:
assumes *finite carrier* $A \subseteq$ *carrier* $A \neq \{\}$ *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*: $i \in$ *agents* **and** *set agents'* $\subseteq$ *agents finite agents*
and *finite*: *finite alts* **and** *alts* $\neq \{\}$
defines [*simp*]: $w' \equiv$ *rsd-winners* ($R(i := Ri')$) *alts* **and** [*simp*]: $w \equiv$ *rsd-winners R alts*
shows w' *agents'* $=$ w *agents'* $\vee$ *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*: $i \in$ *agents* **and** *set agents'* $=$ *agents finite agents*
and *finite*: *finite alts* **and** *alts* $\neq \{\}$
defines [*simp*]: $w' \equiv$ *rsd-winners* ($R(i := Ri')$) *alts* **and** [*simp*]: $w \equiv$ *rsd-winners R alts*
shows $\forall x{\in}w'\ agents'.\ \forall y{\in}w\ agents'.\ x \preceq[R\ i]\ y$
⟨*proof*⟩

end

The lottery that RSD yields is well-defined.

lemma *random-serial-dictatorship-support*:
  assumes *finite agents finite alts agents'* $\subseteq$ *agents alts'* $\neq \{\}$ *alts'* $\subseteq$ *alts*
  shows *set-pmf* (*random-serial-dictatorship agents' alts' R*) $\subseteq$ *alts'*
⟨*proof*⟩

Permutation of alternatives commutes with RSD winners.

lemma *rsd-winners-permute-profile*:
  assumes *perm*: σ *permutes alts* **and** *set agents'* $\subseteq$ *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'* $\subseteq$ *agents* $x \in$ *alts*
  shows *random-serial-dictatorship agents'* $\{x\}$ *R* = *return-pmf x* (**is** *?d* = -)
⟨*proof*⟩

end

8.2 Proofs of properties

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

context *election*
begin

abbreviation *RSD* $\equiv$ *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 *wf*: *is-pref-profile R* **and** *unique*: *has-unique-favorites R*
  shows *RSD R* = *RD R*
⟨*proof*⟩

8.2.3 Anonymity

Anonymity is a direct consequence of the fact that we randomise over all permutations in a uniform way.

sublocale *RSD*: *anonymous-sds agents alts RSD*
⟨*proof*⟩

8.2.4 Neutrality

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

sublocale *RSD*: *neutral-sds agents alts RSD*
⟨*proof*⟩

8.2.5 Ex-post efficiency

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

sublocale *RSD*: *ex-post-efficient-sds agents alts RSD*
⟨*proof*⟩

8.2.6 Strong strategy-proofness

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

sublocale *RSD*: *strongly-strategyproof-sds agents alts RSD*
⟨*proof*⟩

end

end
theory *Randomised-Social-Choice*
imports
  Complex-Main
  SDS-Lowering
  Random-Dictatorship
  Random-Serial-Dictatorship
begin

end

9 Automatic definition of Preference Profiles

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

⟨*ML*⟩

context *election*
begin

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

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

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

end

9.1 Automatic definition of preference profiles from tables

function *favorites-prefs-from-table* **where**
$i = j \Longrightarrow$ *favorites-prefs-from-table* ((*j*,*x*)#*xs*) i = *hd x*
| $i \neq j \Longrightarrow$ *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**
$i \neq j \Longrightarrow$ *weak-ranking-prefs-from-table* ((*i*,*x*)#*xs*) *j* = *weak-ranking-prefs-from-table xs j*
| $i = j \Longrightarrow$ *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 $R \equiv$ *prefs-from-table xs prefs-from-table-wf agents alts xs*
shows *R i a b* $\longleftrightarrow$ *prefs-from-table xs i a b*
a ≺[*R i*] *b* $\longleftrightarrow$ *prefs-from-table xs i a b* $\wedge$ $\neg$*prefs-from-table xs i b a*
anonymous-profile R = *mset* (*map snd xs*)
election agents alts $\Longrightarrow$ $i \in$ *set* (*map fst xs*) $\Longrightarrow$
preferred-alts (*R i*) *x* =
of-weak-ranking-Collect-ge (*rev* (*the* (*map-of xs i*))) *x*
election agents alts $\Longrightarrow$ $i \in$ *set* (*map fst xs*) $\Longrightarrow$
favorites R i = *favorites-prefs-from-table xs i*
election agents alts $\Longrightarrow$ $i \in$ *set* (*map fst xs*) $\Longrightarrow$
weak-ranking (*R i*) = *weak-ranking-prefs-from-table xs i*
election agents alts $\Longrightarrow$ $i \in$ *set* (*map fst xs*) $\Longrightarrow$
favorite R i = *the-elem* (*favorites-prefs-from-table xs i*)
election agents alts $\Longrightarrow$
has-unique-favorites R $\longleftrightarrow$ *list-all* (λz. *is-singleton* (*hd* (*snd z*))) *xs*
⟨*proof*⟩

lemma *pref-profile-from-tableI′*:
assumes *R1* $\equiv$ *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

⟨*ML*⟩

end

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*: $P \vee False \longleftrightarrow P$ $\langle proof \rangle$

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

lemma *less-or-eq-real*:
$(x{::}real) < y \vee x = y \longleftrightarrow x \leq y$ $x < y \vee y = x \longleftrightarrow x \leq y$ $\langle proof \rangle$

lemma *multiset-Diff-single-normalize*:
fixes a c **assumes** $a \neq c$
shows $(\{\#a\#\} + B) - \{\#c\#\} = \{\#a\#\} + (B - \{\#c\#\})$
$\langle proof \rangle$

lemma *ex-post-efficient-aux*:
assumes *prefs-from-table-wf agents alts xss R* $\equiv$ *prefs-from-table xss*
assumes $i \in$ *agents* $\forall i \in$*agents*. $y \succeq$[*prefs-from-table xss i*] x $\neg y \preceq$[*prefs-from-table xss i*] x
shows *ex-post-efficient-sds agents alts sds* $\longrightarrow$ *pmf* (*sds R*) $x = 0$
$\langle proof \rangle$

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

definition *pref-classes* **where**
pref-classes alts le = *preferred-alts le* ‘ *alts* − {*alts*}

primrec *pref-classes-lists* **where**
pref-classes-lists [] = {}
| *pref-classes-lists* (*xs*#*xss*) = *insert* ($\bigcup$(*set* (*xs*#*xss*))) (*pref-classes-lists xss*)

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

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

lemma *pref-classes-lists-aux*:
assumes *is-weak-ranking xss acc* ∩ ($\bigcup$(*set xss*)) = {}
shows *pref-classes-lists-aux acc xss* =
(*insert acc* ((λ*A*. *A* ∪ *acc*) ' *pref-classes-lists* (*rev xss*)) − {*acc* ∪ $\bigcup$(*set xss*)})
⟨*proof*⟩

lemma *pref-classes-list-aux-hd-tl*:
assumes *is-weak-ranking xss xss* ≠ []
shows *pref-classes-lists-aux* (*hd xss*) (*tl xss*) = *pref-classes-lists* (*rev xss*) − {$\bigcup$(*set xss*)}
⟨*proof*⟩

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

lemma *eval-pref-classes-of-weak-ranking*:
assumes $\bigcup$(*set xss*) = *alts is-weak-ranking xss alts* ≠ {}
shows *pref-classes alts* (*of-weak-ranking xss*) = *pref-classes-lists-aux* (*hd xss*) (*tl xss*)
⟨*proof*⟩

context *preorder-on*
begin

lemma *SD-iff-pref-classes*:
assumes *p* ∈ *lotteries-on carrier q* ∈ *lotteries-on carrier*
shows *p* ⪯[*SD*(*le*)] *q* ⟷
(∀ *A*∈*pref-classes carrier le*. *measure-pmf.prob p A* ≤ *measure-pmf.prob q A*)
⟨*proof*⟩

end

lemma (**in** *strategyproof-an-sds*) *strategyproof′*:
  assumes *wf*: *is-pref-profile R total-preorder-on alts Ri′* **and** *i*: $i \in$ *agents*
  shows $(\exists A \in$ *pref-classes alts* $(R\ i)$. *lottery-prob* $($*sds* $(R(i := Ri′)))\ A <$
        lottery-prob $($*sds* $R)\ A) \vee$
      $(\forall A \in$ *pref-classes alts* $(R\ i)$. *lottery-prob* $($*sds* $(R(i := Ri′)))\ A =$
        lottery-prob $($*sds* $R)\ A)$
⟨*proof*⟩

lemma *pref-classes-lists-aux-finite*:
  $A \in$ *pref-classes-lists-aux acc xss* $\Longrightarrow$ *finite acc* $\Longrightarrow$ $(\bigwedge A.\ A \in$ *set xss* $\Longrightarrow$ *finite* $A)$
    $\Longrightarrow$ *finite* A
  ⟨*proof*⟩

lemma *strategyproof-aux*:
  assumes *wf*: *prefs-from-table-wf agents alts xss1 R1* $=$ *prefs-from-table xss1*
      prefs-from-table-wf agents alts xss2 R2 $=$ *prefs-from-table xss2*
  assumes *sds*: *strategyproof-an-sds agents alts sds* **and** *i*: $i \in$ *agents* **and** *j*: $j \in$ *agents*
  assumes *eq*: $R1(i := R2\ j) = R2$ *the* $($*map-of xss1* $i) = xs$
      pref-classes-lists-aux $($*hd xs*$)$ $($*tl xs*$) = ps$
  shows $(\exists A \in ps.\ (\sum x \in A.\ pmf\ (sds\ R2)\ x) < (\sum x \in A.\ pmf\ (sds\ R1)\ x)) \vee$
      $(\forall A \in ps.\ (\sum x \in A.\ pmf\ (sds\ R2)\ x) = (\sum x \in A.\ pmf\ (sds\ R1)\ x))$
⟨*proof*⟩

lemma *strategyproof-aux′*:
  assumes *wf*: *prefs-from-table-wf agents alts xss1 R1* $\equiv$ *prefs-from-table xss1*
      prefs-from-table-wf agents alts xss2 R2 $\equiv$ *prefs-from-table xss2*
  assumes *sds*: *strategyproof-an-sds agents alts sds* **and** *i*: $i \in$ *agents* **and** *j*: $j \in$ *agents*
  assumes *perm*: *list-permutes ys alts*
  defines $\sigma \equiv$ *permutation-of-list ys* **and** $\sigma' \equiv$ *inverse-permutation-of-list ys*
  defines $xs \equiv$ *the* $($*map-of xss1* $i)$
  defines *xs′*: $xs' \equiv$ *map* $((‘)\ \sigma)$ $($*the* $($*map-of xss2* $j))$
  defines $Ri' \equiv$ *of-weak-ranking* xs'
  assumes *distinct-ps*: $\forall A \in ps.$ *distinct* A
  assumes *eq*: *mset* $($*map snd xss1*$) - \{\#$*the* $($*map-of xss1* $i)\#\} + \{\#xs'\#\} =$
        mset $($*map* $($*map* $((‘)\ \sigma) \circ$ *snd*$)$ *xss2*$)$
      pref-classes-lists-aux $($*hd xs*$)$ $($*tl xs*$) =$ *set* ‘ *ps*
  shows *list-permutes ys alts* $\wedge$
      $((\exists A \in ps.\ (\sum x \leftarrow A.\ pmf\ (sds\ R2)\ (\sigma'\ x)) < (\sum x \leftarrow A.\ pmf\ (sds\ R1)\ x)) \vee$
        $(\forall A \in ps.\ (\sum x \leftarrow A.\ pmf\ (sds\ R2)\ (\sigma'\ x)) = (\sum x \leftarrow A.\ pmf\ (sds\ R1)\ x)))$
      (**is** - $\wedge$ *?th*)
⟨*proof*⟩

$\langle ML \rangle$

end

References

[1] F. Brandl, F. Brandt, and C. Geist. Proving the incompatibility of Efficiency and Strategyproofness via SMT solving. *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*, 2016. Forthcoming.